Constructions and applications of quasi-random point sets with negative dependence

by

Gracia Yunruo Dong

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Statistics

Waterloo, Ontario, Canada, 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Boxin Tang Professor, Dept. of Stats. and Act. Sci., Simon Fraser University
Supervisor:	Christiane Lemieux Professor, Dept. of Stats. and Act. Sci., University of Waterloo
Internal Members:	Fan Yang Assistant Professor, Dept. of Stats. and Act. Sci., University of Waterloo
	Ruodu Wang Professor, Dept. of Stats. and Act. Sci., University of Waterloo
Internal-External Member:	Xiaoheng Wang Associate Professor, Dept. of Pure Math., University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Randomized Quasi-Monte Carlo (RQMC) methods are used as an alternative to the Monte Carlo (MC) method when performing numeric integration by replacing the random point set of MC with a randomized low-discrepancy sequence (LDS). Although RQMC methods have been shown to have better convergence rates than MC, especially for smooth functions, it does not hold in general that the RQMC method has lower variance than the MC method. Using the framework of negative dependence, a quasi-monotone function integrated using an LDS with the property of negative dependence has been shown to have variance no larger than that of the MC estimator. We show by numerical examples how to use the framework of negative dependence to evaluate the quality of various point sets, including Sobol' and Faure sequences.

We show, in a similar vein, how scrambled Halton sequences also have a form of negative dependence that is desirable for the purpose of improving upon the MC method for multivariate integration. The scrambling methods with such properties are based on either the nested uniform permutations of Owen or the random linear scrambling of Matoušek. The framework of negative dependence is also used to develop new criteria for assessing the quality of generalized Halton sequences, in such a way that they can be analyzed for finite (potentially small) point set sizes and be compared to digital net constructions. Using this type of criteria, parameters for a new generalized Halton sequence are derived. Numerical results are presented to compare different generalized Halton sequences and their randomizations.

Applications of these point sets include mapping them onto surfaces that are not the unit hypercube. K. Basu and A. Owen have recently developed RQMC methods on the triangle based on the van der Corput sequence. We improve upon the poor one-dimensional projections of this deterministic triangular van der Corput sequence. Rather than using scrambling directly to address this issue, we show how to modify the triangular van der Corput sequence to construct a stratified sampling scheme. More precisely, we show that nested scrambling is a way to implement an extensible stratified estimator based on a stochastic but balanced allocation. We also perform a numerical study to compare the different constructions.

Acknowledgements

I would first like to thank my supervisor, Dr. Christiane Lemieux, who has provided me with endless support throughout both my Master's and PhD studies. Without her encouragement, patience, knowledge and insight, I would not have been able to make it through the many difficult times. I cannot imagine a better supervision throughout my graduate studies than the one I experienced with Christiane.

I would also like to thank Dr. Boxin Tang, Dr. Xiaoheng Wang, Dr. Fan Yang, and Dr. Ruodu Wang for agreeing to serve on my thesis committee.

I credit Mr. Timothy Watters, my grade 6 teacher, with whom I have kept in touch since then, for inspiring me to go into academia and pursue a career in teaching. Mr. Watters, thank you for always continuing to care about your former students.

To my friends, I extend my gratitude – to Hillary, for making sure I stayed off social media whenever deadlines were approaching, to Vanessa, for helping me stay organized, and to Cecilia, for being unwavering in your support for whatever decisions I made.

Lastly, I would like to thank my family – to my parents for their endless help and guidance, and helping me become the person I am now. To my partner Andrew, you are my rock; your love and support have helped keep me focused. And to my unborn son, thank you for your little kicks of encouragement which have helped me stay awake a little longer, especially through these last few nights.

Dedication

This thesis is dedicated to my family.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In many industries and sectors, problems of interest can often be formulated as integrals of high-dimensional functions. These integrals are typically solved numerically, as an analytical solution may not exist or is too difficult to find. For example, for financial applications, percentiles including the Value at Risk of investments need to be computed daily by banks to manage and mitigate risk [47, Chapter 7.6]. Computer graphics also use integration when performing the physics simulations used in path tracing and global illumination problems to render scenes [86]. Other applications that use simulated models in lieu of analytic formulas include those related to climate modelling [12, 28, 42, 83], where time series models are routinely constructed as extremely high-dimensional problems, or biochemistry, which models chemical reactions using complex stochastic models [32].

A problem arises when integrating high-dimensional functions numerically: many algorithms used to integrate low-dimensional problems do not scale well as the number of dimensions increases. For example, the trapezoid rule is often used for one-dimensional integration methods to estimate the area underneath a function, but the number of function evaluations increases exponentially with the number of dimensions to maintain the same level of accuracy, rendering this method infeasible for high-dimensional settings. This phenomenon is known as the "curse of dimensionality" [8]. Thus, Monte Carlo (MC) integration is the standard method for high-dimensional integration. Here, the function is evaluated at n random points, and the estimate for the integral is the average of these n function values. However, even in one dimension, the number of points needs to increase by a factor of k^2 to decrease the error by a factor of k.

(Randomized) Quasi-Monte Carlo ((R)QMC) methods, which replace the random point set used in the MC method with a deterministic low-discrepancy sequence (LDS) that

more evenly covers the area to integrate, generally yield estimates with lower error using the same number of points, as the LDS avoids gaps or clusters that occur with random sampling. This is often used in financial applications to improve the accuracy of estimates for calculating the Value at Risk, [41, 65] as well as for pricing financial products [50]. Even in high dimensions, (R)QMC has been shown to outperform MC, such as in [67], which uses a 360-dimension integration problem from finance. In computer graphics, LDS has also been used in lieu of MC for more efficient rendering and smoother images [14, 15, 43].

Although (R)QMC methods have been shown to have better convergence rates than MC [31, 62], especially for smooth functions [64], it does not hold in general that the (R)QMC method has lower variance than the MC variance. The framework of negative dependence as introduced in [48], which is a multivariate extension of negative quadrant dependence (NQD) [46], gives us a way to determine for what point sets and functions (R)QMC is guaranteed to outperform MC. It has been shown that point sets that have a certain form of multivariate negative dependence when integrating functions that are quasi-monotone (and for 2−dimensional functions, it is sufficient for the function to be monotone in each dimension) are guaranteed to outperform MC [48, 90]. Other works on studying quasi-Monte Carlo point sets through the lens of negative dependence include [92], who derived upper bounds on the discrepancy of the point sets and [91], who gave randomizations for lattice rules that produce point sets with negative dependence. In [90], the C_b criterion and the $C_b(\boldsymbol{k}, P_n)$ values are derived from analyzing the negative dependence properties of point sets. These values can be used to measure the goodness of a point set: the lower these values are, the more well-distributed the point set is. Specifically, if for a base b, C_b is less than 1, then it is possible to construct a sampling scheme with the property of negative dependence by scrambling in base b.

My research is on the constructions and applications of low-discrepancy point sets that have the property of negative dependence. In this thesis, we will study constructions of specific LDS with the property of negative dependence, how to find "good" constructions using negative dependence as a tool, and how these constructions perform on numerical integration problems. When comparing point sets, we expect a "good" quality point set to be uniformly distributed on the unit cube, with few, if any, gaps or clusters, whereas "bad" quality point sets might have clusters or gaps where areas of the unit cube do not have any points at all. We expect a higher quality point set to perform better on numerical integration problems, in the sense that the RQMC integration error is smaller than a lower quality one. We specifically study the negative dependence properties of Halton sequences, deriving theoretical results that support using a specific kind of scrambling as a randomization, and give a "good" set of parameters for generalizing the Halton sequence. As well, on the triangle, scrambling is used to improve the projection properties of an

existing construction. Theoretical results as well as numerical examples are given for our proposed constructions.

Chapter 2 introduces relevant background information that is needed for the core chapters of this thesis, including information about the specific low-discrepancy sequences and randomizations used within this thesis. The concept of negative dependence, including defining the C_b criterion and the $C_b(\boldsymbol{k}, P_n)$ values, is also explained in this chapter. The set of test functions used throughout this thesis is introduced in this chapter as well.

In Chapter 3, we provide an extensive numerical study to demonstrate the power of the $C_b(\boldsymbol{k}, P_n)$ values to measure the quality of point sets, using examples on the Sobol' [75] and Faure [24] sequences, which are two of the most well-known and used low-discrepancy sequences. Working with the Sobol' sequence in Section 3.1, the Faure sequence in Section 3.2, and comparing the Sobol' and Faure sequences with each other in Section 3.3, we show that the $C_b(\boldsymbol{k}, P_n)$ values are more powerful in practice than the t parameter of a $(t, m, s)-$net for differentiating between different point sets in terms of quality.

We also show that using the C_b criterion can help us determine how to best "repair" point sets that may not have good distribution properties to start off with in Section 3.1. Using two different two-dimensional projections of the Sobol' sequence that were purposely constructed to be of extremely poor quality, we use the C_b criterion to choose a base to randomize the point set such that the randomized point set is of good quality and evenly covers the unit square.

One of the limitations of the C_b value as proposed in [90] is that it requires a significant amount of computational resources to calculate, in terms of both runtime and memory. In Section 3.2.2 we propose an alternative criterion, also based on the $C_b(\boldsymbol{k}, P_n)$ values, that allows us to mitigate this issue. This alternative criterion restricts the values of $\boldsymbol{k}$ such that we only consider specific lower-dimensional projections of the point set, rather than all possible $\boldsymbol{k}$. Using this alternative criterion, we are able to compare the quality of point sets in higher dimensions without any concerns about computational power. This alternative criterion allows us to simultaneously consider all one- and two-dimensional projections of the point set, and we use it in this way to compare different forms of scrambling on the Faure sequence in Section 3.2.3 by looking at scramblings based on permutations originally proposed for the van der Corput sequence. Here, we use said permutations as multiplicative factors to generalize the Faure sequence. This work on assessing forms of scrambling on the Faure sequence is in collaboration with Christiane Lemieux and Henri Faure and is the topic of the working paper [20].

The experiments in this chapter not only show the power of the $C_b(\boldsymbol{k}, P_n)$ values to compare point sets and constructions, but also allow us to explore an important question

of whether it is better to address defects in a deterministic point set by searching for a "good" deterministic scrambling in the sense of dependence qualities, or by applying random scrambling. If a well-chosen deterministic scrambling is used, it can then be randomized with a faster randomization for error estimation, whereas random scrambling is a more resource intensive randomization. However, searching for a good deterministic scrambling can be quite computationally demanding.

Next, in Chapter 4, we study the dependence properties of the Halton sequence. The majority of the work in this chapter has been published in [22]. In this chapter, we have several contributions.

Firstly, [90] proves the property of negative dependence for $(t, m, s)-$nets when $t = 0$. We use a similar process here to prove the property of negative dependence for the Halton sequence by introducing a multi-base version of the C_b value. Based on these results, we propose using a multi-base version of the random linear scrambling of Matoušek [52] to randomize the Halton sequence, which has not yet been proposed as a randomization for the Halton sequence. We not only derive theoretical results regarding this randomization, including a formula for the joint pdf of two distinct points randomly chosen from the scrambled point set, but show its usefulness in practice with numerical integration examples.

Secondly, using the same alternative criterion based on the $C_b(\boldsymbol{k}, P_n)$ values as in the previous chapter, we search for a set of permutations to generalize the Halton sequence. These permutations were found by minimizing the criterion value and are given in Table A.1 in Appendix A. We compare these well-chosen permutations with other choices of permutations such as the ones from [26] and [25] as well as the random linear scrambling mentioned above in numerical integration problems, to again answer the question of if it is better to address defects in a deterministic point set by using a "good" deterministic scrambling, or by applying random scrambling.

Thirdly, Section 4.6 extends our work on assessing different forms of scrambling with a similar setup of the experiments as the ones from Section 3.2.3. The work in this section is not part of [22] – it is within the working paper [20], which is done in collaboration with Christiane Lemieux and Henri Faure. Here, we assess two different sets of permutations for the Halton sequence based on the ones introduced in [25].

Finally, in Chapter 5, we focus on the construction of low-discrepancy point sets on two-dimensional triangles, rather than the unit hypercube. Two of the most recent approaches for constructing low-discrepancy point sets on the triangle were proposed by Basu and Owen in [7]. They propose the triangular van der Corput sequence and the triangular Kronecker lattice, which are both used to sample deterministic points on the

two-dimensional triangle. We focus on their deterministic triangular van der Corput sequence. A limitation of this sequence is that while the points are indeed evenly spread out on the two-dimensional triangle, the one-dimensional projections are poor, since they have non-unique points.

This chapter has two main contributions. The first is that we improve upon the one-dimensional projection properties of the triangular van der Corput sequence of Basu and Owen. We do this by modifying the triangular van der Corput to map a scrambled van der Corput sequence to the two-dimensional triangle, rather than the deterministic van der Corput sequence.

The second is we show how to implement the scrambled van der Corput sequence using a stratified sampling scheme, which is much less computationally intensive than applying scrambling directly. This stratified sampling scheme has the same distribution as the nested scrambling of the van der Corput sequence, allowing us to keep the good properties of a scrambled estimator while being more efficient to implement. In fact, we show that the connection between stratified sampling and nested scrambling can be used both ways – not only to give a fast implementation of nested scrambling but also to give a way to create an extensible stratified estimator.

The work on constructing low-discrepancy point sets on the triangle is done in collaboration with Erik Hintz, Christiane Lemieux, and Marius Hofert. This work has not been published yet and is the topic of the working paper [21]. The results presented within Chapter 5 are the work of the author of this thesis. The working paper also includes the development of an extensible lattice construction on the triangle that coincides with the approach of Basu and Owen, but is omitted from this thesis, as the majority of the work on that topic was done by Erik Hintz.

Chapter 6 concludes this thesis.

Chapter 2

Background

This chapter introduces relevant background information that is needed for the core chapters of this thesis. In Section 2.1, we introduce the Monte Carlo method. In Section 2.2, we introduce the Halton, Sobol' and Faure sequences, which are the low-discrepancy sequences that we work with in this thesis. Here, the concept of equidistribution is also introduced. We then discuss some common randomization techniques used with these sequences in practice in Section 2.3. The concept of negative dependence is introduced in Section 2.4, including the C_b value, which is a criterion used to measure how evenly spread out the points within a unit cube are, and can also be used to determine if a specific point set has the property of negative dependence. Finally, in Section 2.5, we also introduce the test functions that will be used throughout this thesis to compare estimators constructed from the different point sets we work with.

2.1 Monte Carlo methods

The Monte Carlo (MC) method for numerical integration uses repeated random sampling to obtain numerical results for integrals that either do not have a closed form or are difficult to integrate theoretically. Quantities of interest in this thesis are integrals over the s-dimensional hypercube $[0,1)^s$, which can be written as

$$\mu = I(f) = \int_{[0,1)^s} f(\mathbf{u}) d(\mathbf{u}),$$

where $f(\mathbf{u}) \in \mathbb{R}$ on $[0,1)^s$. The MC estimator of $I(f)$ based on a sample size of n is

$$\hat{\mu}_{mc} = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{u}_i),$$

where $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ are independent and identically distributed (i.i.d.) samples from the uniform distribution over $[0,1)^s$, and n is the number of random draws.

From the Central Limit Theorem, the integration error follows a Normal distribution asymptotically:

$$\sqrt{n}(I(f) - \hat{\mu}_{mc}) \xrightarrow{D} N(0, \sigma^2),$$

where $\xrightarrow{D}$ denotes convergence in distribution and

$$\sigma = \sqrt{(I(f^2) - I(f)^2},$$

which is estimated by

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(f(\mathbf{u}_i) - \hat{\mu}_{mc})^2}. \tag{2.1}$$

Since the error of MC integration is $O_p(\frac{1}{\sqrt{n}})$, to decrease the error by a factor of k, k^2 times the number of simulations are needed. This is one of the drawbacks of the MC method - numerous simulations are required, and thus improving the accuracy of an estimate can be computationally expensive.

2.2 Quasi-Monte Carlo methods

A way to improve the error of the estimator of our quantity of interest is to use *quasi-Monte Carlo* (QMC) methods instead of the MC method. These methods replace the randomly sampled point set of an MC estimator with a deterministic point set that is chosen to approximate the uniform distribution as closely as possible. Such sets are known as *low-discrepancy sequences* (LDS) or low-discrepancy point sets, as the sequences cover the area over the unit hypercube more evenly than that of a pseudo-randomly generated sequence of Uniform$[0,1)^s$ random variables. Using QMC methods allows the error of the estimator to decrease at a faster rate than $O(\frac{1}{\sqrt{n}})$, which makes it superior for the purpose of estimation [64].

This can be seen in Figure 2.1. The left plot contains 128 points in two dimensions where

the x and y coordinates are independently generated from a pseudo-random Uniform$(0, 1)$ distribution, using the built-in `runif()` function in R [72], which uses the Mersenne Twister generator [53]. The right plot contains 128 points generated from a quasi-random sequence, the Sobol' sequence [75] in two dimensions, generated using `sobol()` from qrng [39]. We can see that the point set in the right plot in Figure 2.1 covers the unit square more evenly than the one on the left. This is a desirable property of quasi-random number sequences, and will be further touched upon in Section 2.2.2.

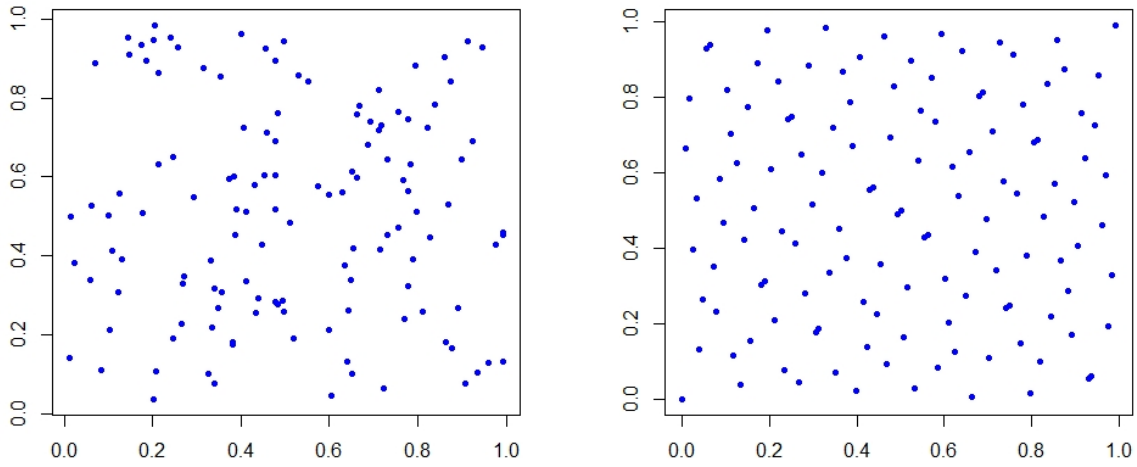


Figure 2.1: Uniformly generated (left) vs quasi-random (right) samples

Since these quasi-random sequences are meant to imitate a uniform distribution, random numbers from a specific distribution cannot be generated directly. Instead, uniform random variables must be transformed using the cumulative distribution functions into the desired distribution, i.e., the inverse transformation method must be used. If we generate $u \sim \text{Uniform}[0, 1)$ from the quasi-random sequence, the random number we need is $F^{-1}(u)$, where F is the cumulative distribution function of our random variable of interest.

We now describe a few different constructions for low-discrepancy sequences. These constructions will be used in the later chapters of this thesis.

2.2.1 The Halton sequence

The *Halton sequence* [35] is one of the oldest low-discrepancy sequences. It is still popular among practitioners because of its simple implementation [36]. It has received renewed attention in the last 10–15 years, both from theoretical and practical perspectives [4, 13, 18, 26, 58, 63, 85, 88]. It is widely accepted that in its original form, the Halton sequence has important defects in higher dimensions, which mainly appear as unwanted correlations over certain two-dimensional projections, for reasons we will explain shortly. For this reason, several proposals have been made to either randomize the Halton sequence or to generalize it via well-chosen permutations: the above list of references contains such proposals. Figure 2.2 shows on the left panel the two-dimensional projection of the first 100 points of the original Halton sequence over its 49th and 50th coordinates, thereby illustrating the unwanted correlations mentioned previously. The middle panel shows how this behaviour can be mitigated by using well-chosen deterministic permutations. However, permutations by themselves cannot break up the linear correlations completely.

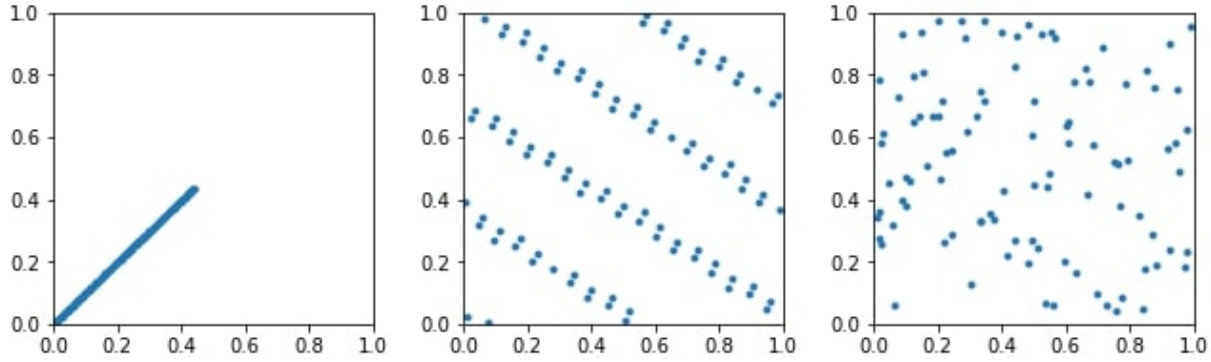


Figure 2.2: First 100 points of the two-dimensional projection over the 49^{th} and 50^{th} coordinates of the Halton sequence (left), generalized Halton sequence using permutations from [26]) (middle), and pseudo random points (right).

To define the Halton sequence, we follow the notation and framework of [26]. The building block for the Halton sequence is the *van der Corput sequence in base* b, denoted by S_b, which has its n^{th} term ($n \geq 1$) defined as

$$S_b(n) = \sum_{r=0}^{\infty} \frac{a_r(n)}{b^{r+1}}, \tag{2.2}$$

where $a_r(n)$ is the r^{th} digit of the b-adic expansion of $n-1 = \sum_{r=0}^{\infty} a_r(n)\ b^r$.

The Halton sequence is an s-dimensional sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots$ in $[0,1)^s$ defined as

$$\mathbf{u}_n = (S_{b_1}(n), \ldots, S_{b_s}(n)), \tag{2.3}$$

where the b_j's, for $j = 1, \ldots, s$, are pairwise coprime. That is, the j^{th} coordinate is defined using S_{b_j}, the van der Corput sequence in base b_j. These b_j's are typically chosen as the first s prime numbers.

A *generalized van der Corput sequence* [25] is obtained by choosing a sequence $\Sigma = (\sigma_r)_{r\geq 0}$ of permutations of $Z_b = \{0, 1, \ldots, b-1\}$. Then, the n^{th} term of the sequence is defined as

$$S_b^{\Sigma}(n) = \sum_{r=0}^{\infty} \frac{\sigma_r\big(a_r(n)\big)}{b^{r+1}}. \tag{2.4}$$

If the same permutation σ is used for all digits (i.e., if $\sigma_r = \sigma$ for all $r \geq 0$), then we use the notation S_b^{σ} to denote S_b^{Σ}. The van der Corput sequence in base b defined in (2.2) is obtained by taking $\sigma_r = I$ for all $r \geq 0$, where I stands for the identity permutation over Z_b.

A *generalized Halton sequence* is obtained by choosing s sequences of permutations $\Sigma_j = (\sigma_{j,r})_{r\geq 0}$, $j = 1, \ldots, s$, and defining the n^{th} point as

$$\mathbf{u}_n = (S_{b_1}^{\Sigma_1}(n), \ldots, S_{b_s}^{\Sigma_s}(n)),\ n \geq 1. \tag{2.5}$$

Another way to generalize the van der Corput sequence is to apply well-chosen linear transformations to the digits $a_r(n)$ before multiplying them by $b^{-(r+1)}$ in (2.2). More precisely, let C be an $\infty \times \infty$ matrix with elements in $\mathbb{Z}_b$, where we assume b is prime. Let $C_{r,\ell}$ be the element on the r^{th} row and ℓ^{th} column of C. The n^{th} term is then defined as

$$S_b^{C}(n) = \sum_{r=0}^{\infty} \sum_{\ell=0}^{\infty} C_{r,\ell} a_\ell(n) b^{-(r+1)}. \tag{2.6}$$

Using different transformations C_j on a given van der Corput sequence in a fixed base b to obtain the j^{th} coordinate of a point in $[0,1)^s$ breaks the clearly wrong pattern of an s-dimensional sequence that would otherwise have all its points along the main diagonal in $[0,1)^s$.

More generally speaking, a sequence in $[0,1)^s$ with n^{th} term given by

$$(S_b^{C_1}(n), \ldots, S_b^{C_s}(n)) \tag{2.7}$$

with b prime is an example of a *digital sequence* [55]. The C_j here are usually referred to as *generating matrices.* Specific choices of generating matrices to create point sets with good equidistribution are discussed in [19], and below, we will see examples of different generating matrices with the Sobol' and the Faure sequences.

More modifications that are made to the Halton sequence are explained in Chapter 4.

2.2.2 Discrepancy, equidistribution, and $(t, m, s)-$nets

Together, the constructions (2.5) and (2.7) cover most of the constructions used in practice that are known to be low-discrepancy sequences, which refers to the fact that the obtained sequences cover the unit hypercube $[0,1)^s$ more evenly than a sequence of independent, randomly chosen points would.

More precisely, the discrepancy of a point set $P_n = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\} \subseteq [0,1)^s$ measures by how much the empirical distribution induced by P_n deviates from the uniform distribution over $[0,1)^s$. To describe this measure, for a subinterval of $[0,1)^s$ of the form $J = \prod_{j=1}^s [0, z_j)$, where $0 < z_j \leq 1$, we consider the difference $E(J;n) = A(J;n) - nV(J)$, where $A(J;n) = \#\{i; 1 \leq i \leq n, \mathbf{u}_i \in J\}$ is the number of points in P_n that fall in the subinterval J, and $V(J) = \prod_{j=1}^s z_j$ is the volume of J. The *star discrepancy* D^* of P_n is then defined as $D^*(P_n) = \sup_{J^*} |E(J^*;n)|/n$. For an infinite sequence, if its first n points satisfy $D^*(P_n) = O((\log n)^s/n)$—which is conjectured to be the best possible convergence rate for the discrepancy—then it is considered to be of low-discrepancy. Other discrepancy measures can be defined by either changing the sets J over which the difference $E(J;n)$ is computed, or by using a norm other than the sup norm in the definition of $D^*(P_n)$ [47, Sec. 5.6.1].

Definition 2.2.1. P_n is $(k_1, \ldots, k_s)-$*equidistributed* in base b if for $n = b^m$ where m is a non-negative integer and $\sum_{i=1}^s k_i \leq m$, every elementary interval of the form

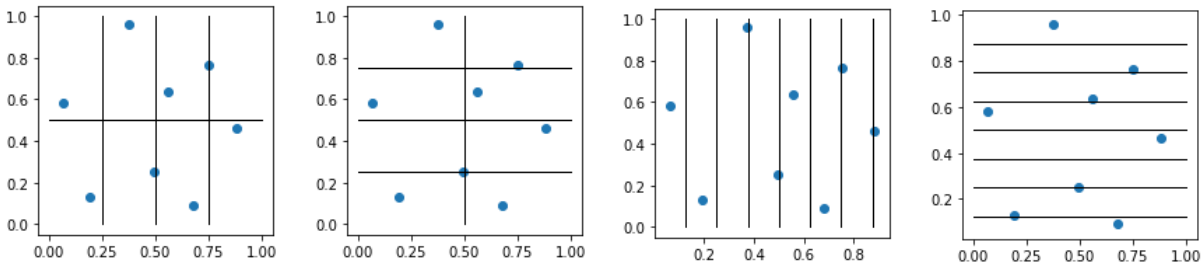
$$\prod_{l=1}^{s} \left[\frac{a_l}{b^{k_l}}, \frac{a_l + 1}{b^{k_l}} \right) \tag{2.8}$$

with $0 \leq a_l < b^{k_l}$ has exactly $b^{m-(k_1+\cdots+k_s)}$ points.

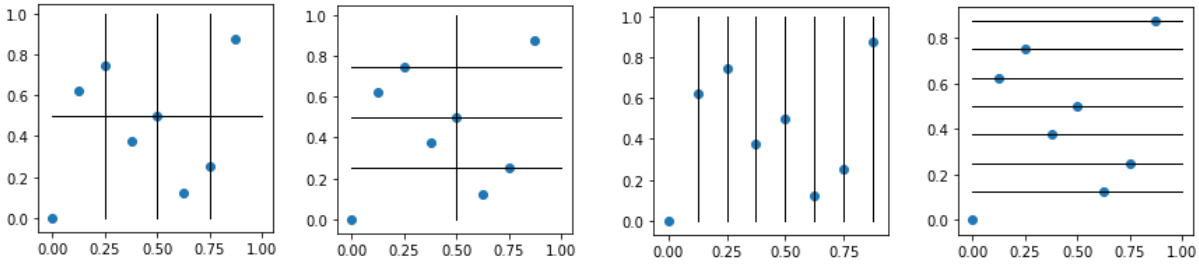
We can now define the concepts of a $(t, m, s)-$net and a $(t, s)-$sequence.

Definition 2.2.2. A (t, m, s)-net [59] in base b (b prime) is a point set $P_n = \{\mathbf{U}_1, \ldots, \mathbf{U}_n\} \subseteq [0, 1)^s$ with $n = b^m$ that is $(q_1, ..., q_s)$-equidistributed in base b for all s-dimensional vectors of non-negative integers $(q_1, ..., q_s)$ such that $q_1 + ... + q_s \leq m - t$. A (t, s)-sequence in base b is a sequence of points where every subsequence of the form $\mathbf{u}_{lb^k}, ..., \mathbf{u}_{(l+1)b^k-1}$ for integers $k \geq t$ and $l \geq 0$ is a $(t, k, s)-$net in base b.

The value t is a quality parameter: the lower t is, the more evenly spread out P_n is. Figure 2.3 shows both a deterministic and a randomized $(0, 3, 2)-$net, specifically, a Faure sequence, which are $(1, 2)-$,$(2, 1)-$,$(0, 3)-$ and $(3, 0)-$equidistributed. For this point set, $t = 0$ as it is $(k_1, k_2)-$equidistributed for all $k_1 + k_2 \leq 3$.



(a) A randomized (using Owen's scrambling) $(0, 3, 2)$-net in base 2



(b) A deterministic $(0, 3, 2)$-net in base 2

Figure 2.3: A $(0, 3, 2)-$net is $(1, 2)-$, $(2, 1)-$, $(0, 3)-$ and $(3, 0)-$equidistrbuted

2.2.3 The Sobol' sequence

The *Sobol' sequence*, a digital sequence, is another popular low-discrepancy sequence. Using Equation 2.6, the Sobol' sequence is obtained by taking $b = 2$ and defining the C_j's using recurrences based on primitive polynomials over $\mathbb{F}_2$ [75].

Specifically, the algorithm to generate the generating matrices C_j's is described in Algorithm 1.

Algorithm 1: Generating Matrices for the Sobol' Sequence

For each coordinate j, $j = 1, \ldots, s$:

1. Define a primitive polynomial over $\mathbb{F}_2$ in the form of $p_j(z) = z^{d_j} + a_{j,1}z^{d_j-1} + \cdots + a_{j,d_j}z^0$ where d_j is the degree and each $a_{j,l} \in \mathbb{F}_2$;
2. Choose d_j direction numbers of the form $v_{j,r} = m_{j,r}/2^r$, where $m_{j,r}$ is an odd integer between 1 and $2^r - 1$ for all $1 \leq r \leq d_j$;
3. For $r > d_j$, $v_{j,r} = a_{j,1}v_{j,r-1} \oplus \cdots \oplus a_{j,d_j-1}v_{j,r-d_j+1} \oplus v_{j,r-d_j} \oplus v_{j,r-d_j}/2^{d_j}$, where $\oplus$ is the binary XOR operation;
4. Write each $v_{j,r} = \sum_{l=1}^{d_j} v_{j,r,l}2^{-l}$;
5. The r^{th} column of C_j is formed by the base 2 expansion of $v_{j,r}$, that is, the r^{th} column of C_j is $(v_{j,r,1}, v_{j,r,2}, \ldots)$.

The $t-$values were first proposed by Sobol' to characterize his sequence. Every one-dimensional projection of the Sobol' sequence is a $(0, 1)-$sequence in base 2. However, in s dimensions, the corresponding $(t, m, s)-$nets created have $t = O(s \log s)$ [75].

2.2.4 The Faure sequence

Using 2.6, the *Faure sequence* [24] is obtained in any prime base $b \geq s$ and is a popular way to create $(0, s)-$sequences and $(0, m, s)-$nets. The generating matrices C_j here are successive powers of the transpose of the Pascal matrices, with operations done in $\mathbb{F}_b$: C_j is the transpose of the Pascal matrix raised to the power of $j - 1$ for $j = 1, \ldots, s$.

The Faure sequence can be generalized by multiplying the generating matrices (from the left) by nonsingular lower triangular matrices. More details about generalizing the Faure sequence is given in Section 3.2.3.

2.3 Randomized Quasi-Monte Carlo

Quasi-random point sets can be randomized in practice to be able to estimate the variance of the *Randomized Quasi-Monte Carlo* (RQMC) estimator. This randomization function

produces a new point set $\tilde{P}_n = \{\tilde{\mathbf{u}}_\mathbf{1}, \ldots, \tilde{\mathbf{u}}_\mathbf{n}\}$ which satisfies the following properties:

1. $\tilde{\mathbf{u}}_\mathbf{i} \sim$ Uniform$([0,1)^s)$ $\forall$ i, and
2. The low-discrepancy of P_n is preserved after the randomization.

Something to note is that even after applying the randomization function to the quasi-random numbers, the generated points $\tilde{\mathbf{u}}_\mathbf{i}$ are not independent across simulations. Thus, any parts of the Monte Carlo methodology that depends on the assumption of independence will need to be adjusted when using the RQMC method.

The MC estimate uses the fact that $\{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$ are i.i.d. samples to estimate the standard error of our estimator. Since this does not hold for quasi-random numbers, when using RQMC to estimate $I(f)$, we cannot use (2.1) to estimate the error of our estimate. Instead, as per [47, Chapter 6.2], we create a random sample of V quasi-random estimators, which are each based on a randomized point set of size n. Let $\tilde{P}_{n,l} = \{\tilde{\mathbf{u}}_{1,l}, \ldots, \tilde{\mathbf{u}}_{n,l}\}$, where $\tilde{P}_{n,1}, \ldots, \tilde{P}_{n,V}$ are V independent copies of $\tilde{P}_n$. Define the l^{th} RQMC estimator as:

$$\hat{\mu}_{rqmc,l} = \frac{1}{n}\sum_{i=1}^{n} f(\tilde{\mathbf{u}}_{i,l}) \text{ for } l = 1, \ldots, m,$$

which has expectation:

$$\begin{aligned}
\mathrm{E}(\hat{\mu}_{rqmc,l}) &= \mathrm{E}\left(\frac{1}{n}\sum_{i=1}^{n} f(\tilde{\mathbf{u}}_{i,l})\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}(f(\tilde{\mathbf{u}}_{i,l})) \\
&= \frac{1}{n}\sum_{i=1}^{n} \int_{[0,1)^s} f(\tilde{\mathbf{u}}_{i,l}) d\tilde{\mathbf{u}}_{i,l} \\
&= I(f), \text{ as each } \tilde{\mathbf{u}}_{i,l} \sim \text{Uniform}([0,1)^s), \text{ from Property 1 above.}
\end{aligned}$$

Thus, $\hat{\mu}_{rqmc,l}$ is an unbiased estimator of $I(f)$.

The overall RQMC estimator of $I(f)$, based on these V i.i.d. estimators, is:

$$\hat{\mu}_{rqmc} = \tfrac{1}{V}\textstyle\sum_{l=1}^{V} \hat{\mu}_{rqmc,l},$$

which has its variance estimated by:

$$\hat{\sigma}^2_{rqmc} = \frac{1}{V}\left(\frac{1}{V-1}\right)\sum_{l=1}^{V}(\hat{\mu}_{rqmc,l} - \hat{\mu}_{rqmc})^2. \tag{2.9}$$

The empirical variance in Equation (2.9) can then be compared with the one in Equation (2.1), the regular MC variance estimate, with a sample size of nV.

In addition to allowing for error estimation, randomization may also improve the quality of the point set. For example, in Chapter 3, we explore finding good randomizations for $(t, m, s)-$nets that improve how evenly the points are spaced out on the unit cube, and in Chapter 4, we explore looking for good randomizations of the Halton sequence from the viewpoint of negative dependence.

When using RQMC methods in practice, there are many packages that can implement various low-discrepancy sequences in different coding languages. For example, there is the qrng [39] package in R [72], which provides implementations for the Sobol' sequence and Halton sequence with various forms of randomizations. In Python [84], the SciPy [87] module has a QMC submodule that provides implementation of the Sobol' sequence, Halton sequence, and Latin Hypercube [54, 79], as well as methods to calculate the discrepancy for each of these sequences. In C [44] and C++ [77], the GNU Scientific Library [34] provides an implementation for the Sobol' sequence, Niederreiter sequence [10], Halton sequence, and reverse Halton sequence (the Halton sequence with permutations $[0, b-1, b-2, \ldots, 1]$, as detailed in [85]), with wrappers available for many other languages. The International Mathematics and Statistics Library (IMSL Numerical Libraries) [57] is implemented in C, Java [1], C#.NET [37], and Fortran [6], with a Python interface. Within the Random Number Generation module in the stat library, there is functionality for generating points from the Faure sequence [24]. Many other implementations of different sequences with a variety of randomization techniques exist.

There are many randomization techniques for low-discrepancy sequences, we describe some of them here, namely the *digital shift* and *digital scrambling* randomization techniques. These are the randomizations that will be used in this thesis.

2.3.1 Digital shift

The digital shift randomization method is a commonly used randomization technique for RQMC, inspired by the random shift method of Cranley and Patterson for lattices, which

are another family of low-discrepancy constructions [17]. It adds a random uniform shift using operations in $\mathbb{Z}_b$.

Specifically, given a point set P_n with elements $\mathbf{u}_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,s})$, and a base b, we generate a random vector $\mathbf{g} = (g_1, \ldots, g_s)$ uniformly in $[0,1)^s$. Then, we take the base b expansion of the $g_j's$: $g_j = \sum_{l=1}^{\infty} g_{j,l} b^{-l}$. The digitally shifted version of P_n, $\tilde{P}_n$, then has elements $\mathbf{U}_i = (U_{i,1}, \ldots, U_{i,s}), i = 1, \ldots, n$. Here, $U_{i,j} = \sum_{l=1}^{\infty} (u_{i,j,l} + g_{j,l}) b^{-l}$. The addition is done in $\mathbb{Z}_b$ and the digits $u_{i,j,l}$ are from the base b expansion of $u_{i,j}$.

The computational complexity of performing a digital shift on a point set is $O(ns)$, as each coordinate of each point needs to be shifted. The base b expansion does not need to be taken to infinity, rather it only needs to be taken to a level of precision equivalent to the precision used to store the points. Usually, we only need to take the first $\lfloor 31 \log_b(2) \rfloor$ digits of the base b expansion.

Digital shifts preserve equidistribution in base b as defined in Section 2.2.2, as applying the digital shift just relabels the b-ary boxes.

2.3.2 Scrambling

Another form of randomization for a point set is scrambling. Here, we explain the nested uniform scrambling method of Owen [60] and the Matoušek style random linear scrambling [52]. Many properties are shared by nested uniform and random linear scrambling, as explained in [52]. As shown in [90] for digital nets and in Section 4.3 for Halton sequences, these equivalences also hold when looking at scrambling from the point of view of negative dependence. A more general definition for a base b-digital scramble, which includes both nested uniform scrambling and random linear scrambling, is given in Definition 2.4.4.

Nested uniform scrambling

The nested uniform scrambling method of Owen [60], sometimes referred to as Owen's scrambling, scrambles P_n by applying random permutations to the digits $u_{i,j,l}$. Permutations π are uniformly randomly distributed over all $b!$ permutations of $[0, 1, \ldots, b-1]$. The permutation of each digit depends on all the digits that came before it, and a new set of permutations is used for each dimension. That is, the permutation used for $u_{i,j,1}$ is π_j, the permutation used for $u_{i,j,2}$ is $\pi_{j,u_{i,j,1}}$ (the permutation applied to the second digit $u_{i,j,2}$ depends on the first digit $u_{i,j,1}$), the permutation used for $u_{i,j,3}$ is $\pi_{j,u_{i,j,1}u_{i,j,2}}$, and so on. To permute the first k digits, there needs to be a total of b^{k-1} permutations.

Nested scrambling is very costly to implement, in both time and memory. It requires having some sort of dictionary or other lookup data structure to have all the permutations stored, and in addition to the $O(ns)$ complexity to go through all the coordinates of every point, the lookup time for each permutation needs to be considered. Our implementation of nested scrambling is similar to Tan and Boyle [78] to be able to save on computational cost. We apply nested scrambling to levels up to k', and take the original digits for $k \geq k'$. In all numerical examples in this thesis, we use $k' = \lfloor 31 \log_b(2) \rfloor$ to keep approximately 32 bits of precision. This differs from that of Tan and Boyle in that they take $\pi_{j,u_{i,j,1}u_{i,j,2}\ldots u_{i,j,k}} = \pi_{j,u_{i,j,1}u_{i,j,2}\ldots u_{i,j,k'}}$ for all $k \geq k'$, but we take $\pi_{j,u_{i,j,1}u_{i,j,2}\ldots u_{i,j,k}} = [0, 1, 2, 3, \ldots, b-1]$ for all $k \geq k'$.

However, despite being costly to implement, nested scrambling is often used because it has the potential to reduce the variance of the corresponding RQMC estimator to $O(n^{-3}\log(n)^{s-1}) \approx O(n^{-3+\epsilon})$ for sufficiently smooth functions [64].

Figure 2.3 illustrates a point set before and after applying Owen's scrambling. Owen's nested scrambling in base b also preserves equidistribution in base b, which can be seen in the aforementioned figure, and satisfies the requirement of being a base $b-$digital scramble as defined in [90].

In Chapter 5, we provide a way to implement the nested scrambled van der Corput sequence as stratified sampling, a much more computationally efficient implementation.

Random linear scrambling

The random linear scrambling [52] of Matoušek is another way to randomize digital nets.

The random linear scrambling of [52] for a van der Corput sequence can be explained as a randomized version of (2.7) where the matrix C_j is replaced by $\tilde{C}_j = R_j C_j$ where R_j is chosen randomly among all non-singular lower-triangular matrices with entries in $\mathbb{Z}_{b_j}$, and then a random digital shift based on random digits $g_{j,r}$, $r \geq 0$, in $\mathbb{Z}_{b_j}$ is added. That is, the n^{th} term of the randomly linearly scrambled van der Corput sequence in base b_j is given by

$$\sum_{r=0}^{\infty} \left(\sum_{\ell=0}^{\infty} \tilde{C}_{r,\ell} a_\ell(n) + g_r \right) b_j^{-(r+1)}. \tag{2.10}$$

As is the case with nested uniform scrambling, here the transformation applied to $a_r(n)$ as part of the randomization process depends on the previous digits $a_0(n), \ldots, a_{r-1}(n)$ via the multiplication of the vector of digits $a_\ell(n)$ with $\ell \geq 0$ by $\tilde{C}_j$. In fact, random linear scrambling is a special case of nested uniform scrambling in which all the permutations

used are linear. This form of scrambling typically results in faster implementations than nested uniform scrambling, both in terms of time and memory requirements [40, 52]. Thus, it is a good alternative to nested uniform scrambling, since it has similar properties in terms of variance analysis without the computational requirements [62, 90].

2.4 Negative dependence

Now that we have discussed the fundamentals of numerical integration as well as the specific low-discrepancy sequences to be used in the later chapters of this thesis, we can discuss the concept of point sets with negative dependence. Formally, negative dependence is defined as follows:

Definition 2.4.1. The random variables X and Y have *negative quadrant dependence (NQD)* if

$$P(X \leq x, Y \leq y) \leq P(X \leq x)P(Y \leq y)$$

for all $x, y \in \mathbb{R}$ [48].

Definition 2.4.2. A vector $\mathbf{X} = (X_1, \ldots, X_r)$ of random variables is *negative lower orthant dependent (NLOD)* if:

$$P(X_1 \leq x_1, \ldots, X_r \leq x_r) \leq \prod_{\ell=1}^{r} P(X_\ell \leq x_\ell),$$

and it is *negative upper orthant dependent (NUOD)* if

$$P(X_1 > x_1, \ldots, X_r > x_r) \leq \prod_{\ell=1}^{r} P(X_\ell > x_\ell).$$

for all $\mathbf{x} \in \mathbb{R}^r$ [90].

Now consider a randomized point set $\tilde{P}_n = \{\mathbf{U}_1, \ldots, \mathbf{U}_n\}$.Let

$$H(\mathbf{x}, \mathbf{y}; \tilde{P}_n) := \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j>i} P(\mathbf{U}_i \leq \mathbf{x}, \mathbf{U}_j \leq \mathbf{y}), \tag{2.11}$$

where $\mathbf{U} \leq \mathbf{x}$ means $U_j \leq x_j$ for all $j = 1, \ldots, s$, and $\mathbf{x}, \mathbf{y} \in [0,1]^s$. If $H(\mathbf{x}, \mathbf{y}; \tilde{P}_n) \leq \prod_{\ell=1}^{s} x_\ell y_\ell$ for all $0 \leq x_\ell, y_\ell \leq 1$, $\ell = 1, \ldots, s$, then we say $\tilde{P}_n$ is an *NLOD sampling scheme* [90].

We can think of $H(\mathbf{x}, \mathbf{y}; \tilde{P}_n)$ as the joint distribution function of a pair of (distinct) points $(\mathbf{U}_I, \mathbf{U}_J)$ randomly chosen in $\tilde{P}_n$. Thus, we can think of the NLOD property as requiring that the probability that the $\mathbf{U}_j$'s be all simultaneously small is no larger than if the $\mathbf{U}_j$'s were independent; the NUOD property similarly requires that the probability that they be all simultaneously large is no larger than if they were independent.

Intuitively speaking, the negative dependence that holds for an NLOD sampling scheme $\tilde{P}_n$ is desirable because it implies the points are less likely to be clustered together, as they instead tend to repel each other, thus ensuring the sampling space is well covered by the points in $\tilde{P}_n$.

We can also look at how this negative dependence would affect the quality of an estimator for

$$I(f) = \int_{[0,1)^s} f(\mathbf{u}) d(\mathbf{u}), \tag{2.12}$$

where $f : [0, 1)^s \to \mathbb{R}$ is assumed to be square-integrable.

We consider the kinds of functions that would have a lower variance when using RQMC compared to MC. Recall that we are interested in approximations of the form

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{u}_i) \tag{2.13}$$

where $P_n = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\} \subset [0, 1)^s$. For instance, one can use the first n points of the Halton sequence to form P_n, thereby obtaining a deterministic approximation for P_n. Alternatively, with the MC method, one takes P_n as a set of n i.i.d. points uniformly distributed over $[0, 1)^s$.

As discussed in [48, 90], we have that

$$\mathrm{Var}(\hat{\mu}_n) = \frac{\sigma^2}{n} + \frac{n-1}{n} \mathrm{Cov}(f(\mathbf{U}_I), f(\mathbf{U}_J)),$$

where $\sigma^2 = \mathrm{Var}(f(\mathbf{U}))$ and $\mathrm{Cov}(f(\mathbf{U}_I), f(\mathbf{U}_J))$ represents the covariance term between the value of the integrand f at two distinct, randomly chosen points $\mathbf{U}_I$ and $\mathbf{U}_J$ from $\tilde{P}_n$. The randomization applied to the point set leads to exchangeable $\mathbf{U}_1, \ldots, \mathbf{U}_n$. This covariance term differentiates the variance of $\hat{\mu}_n$—when $\tilde{P}_n$ is an NLOD sampling scheme—from that of a Monte Carlo estimator with the same number of points n and captures the effect of the randomization and how it causes the estimator $\hat{\mu}_n$ to depart from the behaviour of the independent random points used by the MC method.

This covariance can be written as

$$\sigma_{I,J} := \mathrm{Cov}(f(\mathbf{U}_I), f(\mathbf{U}_J)) = \int_{[0,1]^{2s}} f(\mathbf{x})f(\mathbf{y})\psi(\mathbf{x},\mathbf{y})d\mathbf{x}d\mathbf{y} - \int_{[0,1]^{2s}} f(\mathbf{x})f(\mathbf{y})d\mathbf{x}d\mathbf{y}. \quad (2.14)$$

where $\psi(\mathbf{x},\mathbf{y})$ is the joint pdf of $(\mathbf{U}_I, \mathbf{U}_J)$ evaluated at $(\mathbf{x},\mathbf{y})$. An expression for the joint pdf $\psi(\mathbf{x},\mathbf{y})$ is given in [90]. Having the joint pdf means we can write $H(\mathbf{x},\mathbf{y};\tilde{P}_n)$ as an integral:

$$H(\mathbf{x},\mathbf{y};\tilde{P}_n) = \int_{[\mathbf{0},\mathbf{x})\times[\mathbf{0},\mathbf{y})} \psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v}. \quad (2.15)$$

Note that formally speaking, the definition of $H(\mathbf{x},\mathbf{y};\tilde{P}_n)$ given in (2.11) should lead to a closed integration domain in (2.15). The reason why we instead integrate over a half-open interval is because it aligns better with the properties of $\psi(\mathbf{x},\mathbf{y})$, and is a convention we will follow throughout this paper. It is a valid approach because the boundary has measure 0, and thus the integral is unchanged whether we use a half-open interval or a closed one.

We already saw in Section 2.3 a few different ways to randomize a point set. This yields constructions that can be used to construct RQMC estimators, which can be used for integration and simulation of multivariate functions.

Since the points $\mathbf{U}_i$ from a randomized point set used for RQMC integration are not independent, the hope is that their dependence structure allows for an estimator with better quality than the MC method, as it avoids the gaps and clusters that are inherent to random sampling. When using RQMC to define $\hat{\mu}_n$, an important feature one would like to be able to guarantee is that the estimator will be *better*, in some sense, than the one obtained using the MC method, which we denote as $\hat{\mu}_{mc,n}$. By better, typically we mean that we want the variance of $\hat{\mu}_n$ to be no larger than the variance of the MC estimator. Since for regular MC, $\sigma_{I,J} = 0$, for the variance of the RQMC estimator to be lower, $\sigma_{I,J}$ must be negative.

Now, we introduce a family of functions where the $\sigma_{I,J}$ are guaranteed to be negative under specific sampling schemes.

Definition 2.4.3. Consider a function $f : [0,1]^s \to \mathbb{R}$, and an interval of the form $A = [\mathbf{a},\mathbf{b}] = \prod_{j=1}^s [a_j, b_j] \subseteq [0,1]^s$, with $0 \le a_j \le b_j \le 1, j = 1,\dots,s$. Let the dimension d of A be defined as $d = \sum_{j=1}^s \mathbf{1}_{a_j<b_j}$. Let

$$\Delta^{(s)}(f;A) = \sum_{\mathcal{I}\subseteq\{1,\dots,s\}} (-1)^{|\mathcal{I}|} f(\mathbf{a}^{\mathcal{I}};\mathbf{b}^{-\mathcal{I}}),$$

where $f(\mathbf{a}^{\mathcal{I}}; \mathbf{b}^{-\mathcal{I}})$ is the function f evaluated at $\mathbf{x}$ with $x_j = a_j$ if $j \in \mathcal{I}$ and $x_j = b_j$ if $j \notin \mathcal{I}$. If $\Delta^{(s)}(f; A) \geq 0$ for all A of dimension $1 \leq d \leq s$, then f is said to be *quasi-monotone* or *completely monotone*.

For two-dimensional functions, a sufficient condition to be quasi-monotone is to be monotone in each coordinate. It is shown that point sets that have the property of negative dependence (either NLOD or NUOD) when integrating functions that are quasi-monotone have $\sigma_{I,J} \leq 0$ [48, 90].

Other works on studying quasi-Monte Carlo point sets through the lens of negative dependence include [92], who derived upper bounds on the discrepancy of the point sets and [91], who gave randomizations for lattice rules that produce point sets with negative dependence.

The framework of negative dependence gives us a way to determine sufficient conditions for RQMC to outperform MC. Namely, in [48] it is shown that if f is a bounded quasi-monotone function, then an NUOD sampling scheme integrates it with variance no larger than MC. This is in contrast with results showing that some RQMC estimators have a variance that converges much more quickly than MC, as these results do not guarantee RQMC will have lower variance than MC for a fixed sample size n.

In [90], it was shown that certain types of scrambling applied to $(0, m, s)$-nets produces NLOD/NUOD sampling schemes. Both the nested uniform scrambling of Owen [60] and the random linear scrambling of Matoušek [52] are examples of this type of scrambling, which, following [90], we refer to as *base b-digital scrambling*. For point sets that have been randomized using a base $b-$digital scramble, the NLOD and NUOD properties have been shown to be equivalent [90, Thm. 4.14], which is why, in this thesis, the terms NLOD and NUOD are used interchangeably.

Definition 2.4.4 (Base b-digital scramble)**.** Let $P_n = \mathbf{U}_1, \dots \mathbf{U}_n$ and $\tilde{P}_n = \tilde{\mathbf{U}}_1, \dots \tilde{\mathbf{U}}_n$ be the scrambled version of P_n, that is, $\tilde{P}_n = \mathcal{S}(P_n)$. A randomization $\mathcal{S}$ is a base b-digital scrambling if the following two properties hold: Let $\tilde{U}_{i,\ell} = \sum_{r=1}^{\infty} \tilde{U}_{i,\ell,r} b^{-r}$, that is, $\tilde{U}_{i,\ell,r}$ represents the r^{th} digit in the base b expansion of the ℓ^{th} coordinate of the i^{th} point $\tilde{\mathbf{U}}_i$ in the scrambled point set $\tilde{P}_n$. Then we must have:

1. Each $\mathbf{U}_i \sim U([0,1)^s)$;
2. For two distinct points $\tilde{\mathbf{U}}_i = \mathcal{S}(\mathbf{U}_i), \tilde{\mathbf{U}}_j = \mathcal{S}(\mathbf{U}_j)$ and for each coordinate $\ell = 1, \dots, s$, if the two deterministic points $U_{i,\ell}, U_{j,\ell}$ have the same first r digits in base b and differ on the $(r+1)^{\text{th}}$ digit, then:

(a) the scrambled points $(\tilde{U}_{i,\ell}, \tilde{U}_{j,\ell})$ also have the same first r digits in base b;

(b) the pair $(\tilde{U}_{i,\ell,r+1}, \tilde{U}_{j,\ell,r+1})$ is uniformly distributed over $\{(k_1, k_2), 0 \le k_1 \ne k_2 < b\}$;

(c) the pairs $(\tilde{U}_{i,\ell,v}, \tilde{U}_{j,\ell,v})$ for $v > r+1$ are mutually independent and uniformly distributed over $\{(k_1, k_2), 0 \le k_1, k_2 < b\}$.

2.4.1 C_b values

In [90], a criterion C_b was introduced, which gives a way to determine if a point set is NLOD/NUOD. The following definitions are required to define C_b and present Theorem 2.4.8, which gives us a way to determine if a point set is an NLOD/NUOD sampling scheme, which turns out to be equivalent to $C_b \le 1$ for that point set.

The following quantities are essentially used to count pairs of points from a given point set P_n that share a certain number of initial digits in their base b expansion. Geometrically speaking, they count how many pairs of distinct points are in a given cell from a given partition of the unit hypercube $[0,1)^s$.

Definition 2.4.5. For $x, y \in [0,1)$, we define $\gamma_b(x,y) \ge 0$ as the exact number of initial digits shared by x and y in their base b expansion, i.e. the smallest number $i \ge 0$ such that $\lfloor b^i x \rfloor = \lfloor b^i y \rfloor$ but $\lfloor b^{i+1} x \rfloor \ne \lfloor b^{i+1} y \rfloor$.

If $x = y$ then we let $\gamma_b(x,y) = \infty$.

For $\mathbf{x}, \mathbf{y} \in [0,1)^s$, we define $\boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y}) = (\gamma_b(x_1, y_1), \dots, \gamma_b(x_s, y_s))$ and $\gamma_b(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^s \gamma_b(x_j, y_j)$.

Definition 2.4.6. Let $P_n = \{\mathbf{U}_1, \dots, \mathbf{U}_n\}$ be a point set in $[0,1)^s$, $b \ge 2$ be an integer, and $\boldsymbol{i}, \boldsymbol{k} \in \mathbb{N}^s$, where $\mathbb{N} = \{0, 1, 2, \dots\}$. Then,

1. $N_b(\boldsymbol{i}; P_n)$ is the number of ordered pairs of distinct points $(\mathbf{U}_l, \mathbf{U}_j)$ in P_n such that $\boldsymbol{\gamma}_b^s(\mathbf{U}_l, \mathbf{U}_j) = \boldsymbol{i}$, as

2. $M_b(\boldsymbol{k}; P_n)$ is the number of ordered pairs of distinct points $(\mathbf{U}_l, \mathbf{U}_j)$ in P_n such that $\boldsymbol{\gamma}_b^s(\mathbf{U}_l, \mathbf{U}_j) \ge \boldsymbol{k}$ componentwise.

Now, define $C_b(\boldsymbol{k}; P_n) = \frac{b^{|\boldsymbol{k}|} M_b(\boldsymbol{k}; P_n)}{n(n-1)}$, and $C_b = \sup_{\boldsymbol{k}} C_b(\boldsymbol{k}; P_n)$. Here, $|\boldsymbol{k}| = \sum_{i=1}^s k_i$.

As well, we define the concept of a point set being completely quasi-equidistributed (c.q.e.) in a base, from Definition 4.4 of [90]:

Definition 2.4.7. Let P_n be a point set of size n in $[0,1)^s$ and $b \geq 2$ be a base. Let $\boldsymbol{k} = (k_1, \ldots, k_s) \in \mathbb{N}^s$. Then we say P_n is $\boldsymbol{k}-$*quasi-equidistributed in base* b if $C_b(\boldsymbol{k}; P_n) \leq 1$. If $C_b(\boldsymbol{k}; P_n) \leq 1$ for all $\boldsymbol{k} \in \mathbb{N}^s$ then we say P_n is *completely quasi-equidistributed (c.q.e) in base* b.

Theorem 4.16 in [90] shows the relationship between $C_b(\boldsymbol{k}; P_n)$ and negative dependence:

Theorem 2.4.8. *Let P_n be a deterministic point set of size n in $[0,1)^s$ and $b \geq 2$ be an integer. Assume P_n is such that the j^{th} coordinate of the points are all distinct. Let ${}_b\tilde{P}_n$ be the sampling scheme obtained by applying a base $b-$digital scramble to P_n. Then ${}_b\tilde{P}_n$ is NLOD/NUOD if and only if $C_b(\boldsymbol{k}; P_n) \leq 1 \; \forall \; \boldsymbol{k}$, or equivalently, P_n is c.q.e. in base b.*

The proof of this theorem makes use of the following inequality [90, Eqn. 4.1]:

$$H(\mathbf{x}, \mathbf{y}; {}_b\tilde{P}_n) = \int_{R(\mathbf{x},\mathbf{y})} \psi(\mathbf{u}, \mathbf{v}) d\mathbf{u} d\mathbf{v} = \sum_{\boldsymbol{k} \in \mathbb{N}^s} t_{\boldsymbol{k}} C_b(\boldsymbol{k}; P_n) \leq C_b \operatorname{Vol}(R(\mathbf{x}, \mathbf{y})),$$

where $R(\mathbf{x}, \mathbf{y}) = \{(\mathbf{u}, \mathbf{v}) \in [0,1)^{2s} : u_j < x_j, v_j < y_j, j = 1, \ldots, s\} = [\mathbf{0}, \mathbf{x}) \times [\mathbf{0}, \mathbf{y})$ is the rectangle formed by the origin and $(\mathbf{x}, \mathbf{y})$. The intuition behind the inequality is that the $t_{\boldsymbol{k}}$ values decompose the integral of the pdf by decomposing the box $R(\mathbf{x}, \mathbf{y})$ into a sum over elementary boxes, such that $\sum_{\boldsymbol{k} \in \mathbb{N}^s} t_{\boldsymbol{k}} = \operatorname{Vol}(R(\mathbf{x}, \mathbf{y}))$. The $t_{\boldsymbol{k}}$ values can be thought of as being weights and the $C_b(\boldsymbol{k}; P_n)$ values as the propensity for negative dependence. For more details, see [90, Sec. 4].

The quantity $M_b(\boldsymbol{k}; P_n)$ is closely connected to the concept of equidistribution: if P_n with $n = b^m$ is $\boldsymbol{k}-$equidistributed, then $M_b(\boldsymbol{k}; P_n) = b^{|\boldsymbol{k}|}(b^{m-|\boldsymbol{k}|}(b^{m-|\boldsymbol{k}|} - 1)) = b^m(b^{m-|\boldsymbol{k}|} - 1)$. In [90] it is shown via an analysis of the C_b value that a scrambled $(t, m, s)-$net [60] has the NLOD/NUOD property if and only if $t = 0$. When $t = 0$, it means we have $M_b(\boldsymbol{k}; P_n) = b^m(b^{m-|\boldsymbol{k}|} - 1)$ for all $\boldsymbol{k}$ such that $|\boldsymbol{k}| \leq m$, and is 0 if $\boldsymbol{k} > m$. In other words, in this case we can partition $[0,1)^s$ into intervals of the form (2.8) until we have as many "boxes" as we have points, so there will be exactly one point per box, and from then on there are zero pairs of points in boxes of that size or smaller. This in turn implies $C_b(\boldsymbol{k}; P_n) \leq 1$, which is why scrambled digital nets with $t = 0$ are NLOD/NUOD.

However, we also want to see what other qualities of a point set $C_b(\boldsymbol{k}; P_n)$ can capture. As mentioned earlier, the t parameter of a (t, m, s)-net is a criterion that indicates how well-distributed the points of a LDS are. The lower t is, the more evenly spread out the points are. However, there are point sets with the same t parameter that perform very differently on integration problems. It is also possible for two nets with the same value of

t, m, and s to have different $C_b(\boldsymbol{k}; P_n)$ values. The converse of this statement is not true, as from Proposition 4.7 of [90], $t = m - \max\{\ell : \ell \leq m \wedge \forall \boldsymbol{k} \in \mathbb{N}^s : |\boldsymbol{k}| = \ell \Rightarrow C_b(\boldsymbol{k}; P_n) \leq 1\}$, meaning that if two point sets have the same $C_b(\boldsymbol{k}; P_n)$ values, they would also have the same t. This means that the $C_b(\boldsymbol{k}; P_n)$ values contain more information about the point set than the t parameter. Additionally, the t parameter is only meaningful when the point set has n points, where n is a power of an integer base b. The quality measure C_b circumvents these issues as it takes values over the rational numbers, whereas t only takes non-negative integer values. C_b also works for any n, rendering it a more precise criterion and allowing us to more easily differentiate between point sets. The usefulness of the C_b criterion will be explored in more detail in Chapter 3 with numerical examples.

Corollary 4.14 of [90] shows that the values $C_b(\boldsymbol{k}; P_n)$ can be used to differentiate two point sets with respect to their propensity for negative dependence. More precisely, it shows that the $C_b(\boldsymbol{k}; P_n)$ values are able to capture the difference between the two nets in their ability to keep the integral of the joint pdf small. The parameter t fails to capture this difference because it aggregates too much information regarding the equidistribution properties of P_n.

Corollary 2.4.9. *Let P_n and P'_n be deterministic point sets of size n in $[0,1)^s$ such that the j^{th} coordinate of the points are all distinct, for each $j = 1, \ldots, s$. Let ${}_b\tilde{P}_n$ and ${}_b\tilde{P}'_n$ be the sampling schemes obtained by applying a base $b-$digital scramble to P_n and P'_n, respectively. Let $\psi(\mathbf{u}, \mathbf{v})$ and $\psi'(\mathbf{u}, \mathbf{v})$ be the joint pdf of two distinct points randomly chosen from ${}_b\tilde{P}_n$ and ${}_b\tilde{P}'_n$, respectively. Then the following are equivalent:*

1. *For all $\mathbf{x}, \mathbf{y} \in [0,1]^s$,*

$$\int_{R(\mathbf{x},\mathbf{y})} \psi(\mathbf{u}, \mathbf{v}) d\mathbf{u} d\mathbf{v} \leq \int_{R(\mathbf{x},\mathbf{y})} \psi'(\mathbf{u}, \mathbf{v}) d\mathbf{u} d\mathbf{v}.$$

2. *$C_b(\boldsymbol{k}; P_n) \leq C_b(\boldsymbol{k}; P'_n)$ for all $\boldsymbol{k} \in \mathbb{N}^s$.*

2.5 Test functions

Since a large portion of the work in this thesis involves comparing different constructions of point sets using criteria based on the $C_b(\boldsymbol{k}, P_n)$ values, there needs to be a way to determine if these criteria are actually a good method of deciding on if a construction is "good" or not. One way to do this is to use a set of functions to compare performances of these

functions on, where we assume the goal is to integrate them, i.e., compute $I(f)$ as defined in (2.12).

Here are the test functions that we use:

1. $g(\mathbf{x}) = \Pi_{j=1}^{s} 1 + c(x_j - 0.5)$ where c is a constant [3]. This test function has fairly low effective dimension [3], meaning that the majority of the variance can be explained by low dimensional projections. This function integrates to 1 over the unit cube. We consider two combinations of s and c when working with higher-dimensional constructions. First, we use $s = 120, c = 0.1$, which has an effective dimension of four in the superposition sense with a threshold of 0.99; second, we use $s = 96, c = 0.25$, which has an effective dimension of six in the superposition sense with a threshold of 0.99. This means that for these settings, 99% of the variance can be explained by four- and six- dimensional projections, respectively, despite the dimension of the function being much higher. These settings for s and c are from [3] and also used in [26]. We also use this function when comparing lower-dimensional point sets, with the setting of $c = 0.25$. Since this function is monotone in each coordinate, in two dimensions it is also quasi-monotone.

2. $g_1(x) = \Pi_{j=1}^{s} \frac{|4x_j - 2| + \alpha_j}{1 + \alpha_j}$, where α is a constant [76]. This function integrates to 1 over the unit cube, and formulas for the variance of the ANOVA components are given in [61]. We use the following choices of α, which are the same as the choices from [26]: 1) $\alpha_j = 0.01$, 2) $\alpha_j = 1$, 3) $\alpha_j = j$, 4) $\alpha_j = j^2$. As we go from choices 1) to 4) for the choice of α_j, the effective dimension in the truncation sense for this function decreases [26]. A function is said to have low effective dimension in the truncation sense if it can be well approximated by a function that only depends on the first few variables.

3. $h_0(\mathbf{x}) = \sum_{j=1}^{s} (e^{x_j} - e + 1)$ from [66], which integrates to 0 over the unit cube.

4. $h_1(\mathbf{x}) = (\sum_{j=1}^{s} x_j)^2$ from [66], which integrates to $s/3 + s(s-1)/4$ over the unit cube.

5. The Genz integrand family of functions [29]. The specific functions within this family will be defined in Chapter 4 when they are used. Unlike the other functions above, parameters are typically not fixed and instead chosen randomly, and the reported error is the average error over all such randomizations.

We also consider some functions that represent real-life applications, including higher dimension finance problems.

1. Stochastic activity network (SAN). This is a 13-dimensional problem described in [47], page 99, originally from [5]; however, here we also use a 12-dimensional version, with activity 10 removed and all other parameters the same. A probability is estimated using naive Monte Carlo, which means the corresponding integrand is an indicator function.

2. The European Arithmetic Asian Call option pricing. We want to estimate the value at time 0 of an arithmetic Asian call option that can only be executed at expiry on an underlying asset that follows a lognormal distribution. Formally, the value of the option is $C_0 = \mathrm{E}\left[\max\left(0, e^{-rT}\left(\frac{1}{s}\sum_{j=1}^{s} S\left(t_j\right) - K\right)\right)\right]$, where $S(t_j)$ is the price of the asset at time t_j, $T = t_s$ is the time to expiry in years, r is the risk-free interest rate, and σ is the volatility. The lognormal model means we can write $S(t_j) = S(t_{j-1})e^{\left(r-\sigma^2/2\right)(t_j - t_{j-1}) + \sigma\sqrt{(t_j - t_{j-1})}Z}$, where $Z \sim N(0,1)$ is a standard Gaussian random variable. For our applications, $T = 1$, $r = 0.05$, $\sigma = 0.3$ and we use an initial stock price of $S_0 = 50$. We use three options of K, the strike price. We use $K \in \{45, 50, 55\}$ to account for out-of-the-money, at-the-money, and in-the-money options, respectively. For s, the number of dimensions or time steps, we have $s = 17$ for the experiments for the experiments in Chapter 3 and $s \in \{40, 70\}$ for the experiments in Chapter 4. These choices of parameters are taken from [26].

3. A mortgage-backed security problem. We estimate the value at time 0 of a mortgage-backed security, whose value is given by $E\left(\sum_{j=1}^{s} v_j c_j\right)$, where v_j is the discount rate for month j and c_j is the cash flow in month j. Both of these values are random quantities that are functions of the stochastic interest rate process $i_0, i_1, \ldots, i_s$. Specifically, we use the interest rate model from [11], as follows:

$$i_l = K_0 e^{\eta_l} i_{l-1},$$

where $\eta_l \sim N(0,1)$, then

$$v_l = \prod_{k=0}^{l-1} (1 + i_k)^{-1},$$

and

$$c_l = c r_l((1 - w_l) + w_l \alpha_l),$$

where:

- c: The monthly mortgage payment, here, we set it to 1.

- w_l: The fraction of remaining mortgages prepaying in month l. This is calculated as $K_1 + K_2 \arctan(K_3 i_l + K_4)$.
- r_l: The fraction of remaining mortgages at month l, calculated as $\prod_{k=1}^{l-1}(1 - w_l)$.
- α_l: The remaining annuity at month l divided by the monthly mortgage payment. This is calculated as $\sum_{k=0}^{s-l}(1 + i_0)^{-k}$.

Thus, we need to specify (i_0, K_0, σ^2) for the interest rate model and (K_1, K_2, K_3, K_4) for the prepayment model. We use $K_0 = e^{-\sigma^2/2}$ so that $E(i_k) = i_0$ for all k. For the remaining parameters, We use the following three sets of parameters from [56] and [11], also used in [26]. For each, we assume a 30-year contract, which means that $s = 360$ as payments are monthly.

(a) "Linear": $(K_1, K_2, K_3, K_4, \sigma, I_0) = (0.01, -0.005, 10, 0.5, 0.02, 0.007)$. From [11], this set of parameters leads to an almost linear function in its 360 inputs $x_1, \ldots, x_s$. Over 99.9% of the variance is explained by the one-dimensional structure, so the effective dimension is one in the superposition sense. The true value of this expectation is 131.78706.

(b) "Nonlinear": $(K_1, K_2, K_3, K_4, \sigma, I_0) = (0.04, 0.0222, -1500, 7, 0.02, 0.007)$. Also, from [11], this has less of a linear component and only about 94% of variance is explained by the one-dimensional structure the and the true value is 130.712365.

(c) "Tezuka": $(K_1, K_2, K_3, K_4, \sigma, I_0) = (0.24, 0.134, -261.17, 12.72, 0.2, 0.00625)$. From [56], the true value is 143.0182.

To test the performance of the different triangular constructions in Chapter 5, we consider the following test functions over the right-angle triangle with corners at (0, 0), (0, 1), (1, 0), which we denote as $\triangle_R$:

1. $f_1(x, y) = (|x - \beta| + y)^d$, from [69]. This function has a singularity, so we anticipate this function to be harder to integrate than the others. This integral evaluates to $\frac{1}{(d+1)(d+2)}\left((d + 1/2)(1 - \beta)^{d+2} + (\beta + 1)^{d+2}/2 - \beta^{d+2}\right)$ over $\triangle_R$.

2. $f_2(x, y) = \cos(2\pi\beta + \alpha_1 x + \alpha_2 y)$, from [69]. This is a smooth oscillatory function. This integral evaluates to $\frac{1}{\alpha_2}(\frac{1}{\alpha_1 - \alpha_2}(\cos(2\pi\beta + \alpha_2) - \cos(2\pi\beta + \alpha_1)) + \frac{1}{\alpha_1}(\cos(2\pi\beta + \alpha_1) - \cos(2\pi\beta)))$ over $\triangle_R$.

3. $f_3(x, y) = x^{\alpha_3} + y^{\alpha_3}$. Since this is the sum of univariate functions, it will help us determine if poor one-dimensional projections affect the integration power of the point set. This integral evaluates to $\frac{2}{(\alpha_3+1)(\alpha_3+2)}$ over $\triangle_R$.

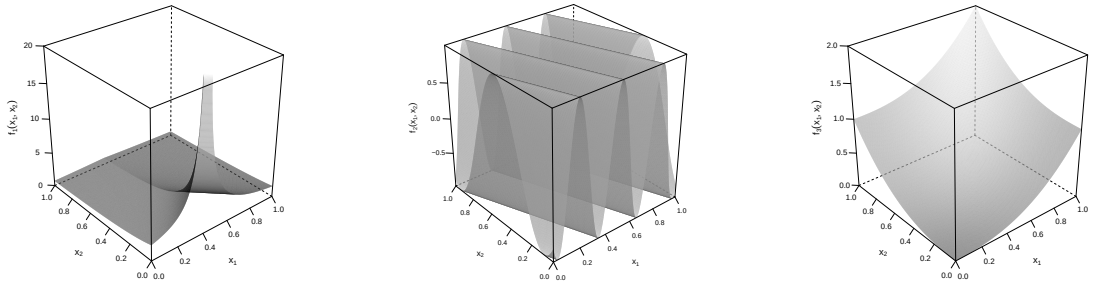


Figure 2.4: Test functions f_1 (left), f_2 (middle) and f_3 (right) used in the numerical study.

We use $\beta = 0.4$, $d = -0.9$, $\alpha_1 = e^3$, $\alpha_2 = e^2$, $\alpha_3 = 2.5$ and estimate $\mu_j = \int_{\triangle_R} f_j(\mathbf{x})\, d\mathbf{x}$, whose theoretical values are known for $j = 1, 2, 3$. Figure 2.4 displays f_k for $k = 1, 2, 3$ with these parameter settings. Although the figures show the functions over the unit square, we integrate over the right angle triangle with corners at (0, 0), (0, 1), (1, 0).

We use these functions to numerically compare different constructions, i.e., to demonstrate that our "good" and "bad" constructions, as determined by our selected metric, do indeed perform well and poorly respectively on integration problems.

We compare estimators using either their mean-squared error (MSE) or variance, which are obtained as follows: when integrating a function f, we assume that for $v = 1, \ldots, V$, $P_{n,v} = \{\mathbf{x}_{1,v}, \ldots, \mathbf{x}_{n,v}\}$ is an s-dimensional point set that has been randomized, and that the $P_{n,v}$ for $v = 1, \ldots, V$ are independent from each other (e.g., they have been randomized using independent shifts or independent scramblings). Then we either compute

$$\text{MSE} = \frac{1}{V} \sum_{v=1}^{V} \left(\hat{\mu}_v - \mu(f)\right)^2,$$

where $\mu(f) = \int_{[0,1)^s} f(\mathbf{x}) d\mathbf{x}$ if the true value of the integrand is known, or

$$\text{Var} = \frac{1}{V-1} \sum_{v=1}^{V} \left(\hat{\mu}_v - \hat{\mu}\right)^2,$$

where $\hat{\mu} = \frac{1}{V} \sum_{v=1}^{V} \hat{\mu}_v$ is the sample mean over all V independent randomizations if the

true value of the integrand is not known. In both cases,

$$\hat{\mu}_v = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{x}_{i,v}).$$

That is, $\hat{\mu}_v$ is the estimator for μ obtained from the v^{th} randomized point set $P_{n,v}$.

Chapter 3

Negative dependence as a tool to measure the quality of point sets

In [90], the $C_b(\boldsymbol{k}; P_n)$ values and the C_b criterion were introduced, as well as a theoretical framework explaining how they connect to the NLOD property for scrambled point sets. In this chapter, we explore how we can use these $C_b(\boldsymbol{k}; P_n)$ in practice to decide how and when to randomly scramble, and to differentiate point sets and sequences with an equal parameter t, but a significant difference in their quality. As discussed in Section 2.4.1, the $C_b(\boldsymbol{k}; P_n)$ values are more powerful than the t parameter in measuring the quality of point sets: it takes rational number values, whereas t only takes integer values. As well, the $C_b(\boldsymbol{k}; P_n)$ values do not require the number of points n to be a power of b, whereas the t is only defined when the number of points is an integer power of the base b. The $C_b(\boldsymbol{k}; P_n)$ values also provide more information about the point set, since multiple values of C_b can correspond to the same t value, while the converse is not true. This leads to a more precise criterion that can be used with a wider range of point sets.

The work in this chapter is on the theme of using the criterion $C_b(\boldsymbol{k}; P_n)$ to measure the quality of point sets. We expect that generally, a "better" point set will lead to lower RQMC integration errors when used on numerical problems. In this chapter, we give numerical examples on the Sobol' and Faure sequences, primarily using two-dimensional projections, to illustrate the ability of the $C_b(\boldsymbol{k}; P_n)$ values to differentiate between "good" and "bad" point sets in terms of their integration power when used as a scrambled RQMC estimator.

We first explore the Sobol' sequence in Section 3.1, where we show the criterion's ability to differentiate between point sets of different quality. We also show how to use the criterion

to find a base for randomization, where we can "repair" a poorly constructed point set by scrambling in a higher base than the base used in construction, and demonstrate this with numerical integration problems. We then explore similar experiments for the Faure sequence in Section 3.2, where we attempt to answer the question of "Is it worth trying to find a good fixed scrambling and then simply shift for the randomization, which is cheaper to run, or should we randomly choose a scramble?". Section 3.2.3 contains the joint work with Christiane Lemieux and Henri Faure that has not been published yet and is the topic of the working paper [20]. Here, we assess different forms of scrambling on the Faure sequence by looking at scramblings based on the ones originally proposed in [25]. These deterministic scramblings were originally proposed as permutations for the van der Corput sequence, but here we use these permutations as multiplicative factors and assess their performance on the Faure sequence. Lastly, in Section 3.3, we explore examples in higher dimensions by considering all two-dimensional projections of each point set simultaneously when comparing the Sobol' and the Faure sequence. We conclude our work and note some future direction in Section 3.4, including discussing the efficiency of calculating the $C_b(\boldsymbol{k}; P_n)$ values.

Part of the work in this chapter, particularly, the results and discussions about Figures 3.1, 3.3, 3.12 and Tables 3.1, 3.2, 3.7, 3.10 is done jointly with Jaspar Wiart and Christiane Lemieux and taken from Section 5 of [90]. From [90], my contributions include the aforementioned results as well as implementation of programs for calculating the $C_b(\boldsymbol{k}; P_n)$ values and for randomizing point sets via nested scrambling. The implementations for both calculating the $C_b(\boldsymbol{k}; P_n)$ values and for nested scrambling is generalized such that it only requires a point set P_n and an integer base b as inputs. That is, we do not need to be working with a digital net constructed in the same base as the base used for scrambling.

3.1 Scrambling and negative dependence concepts for point sets constructed in base 2

In this section, we work with the Sobol' sequence, a popular digital sequence constructed in base 2. The Sobol' sequence is a popular low-discrepancy sequence that is often used in practice with implementations available in many programming languages. The base 2 construction of the Sobol' sequence allows implementations to take advantage of binary logical operations for point sets to be created quickly, and the Sobol' sequence has been shown to perform well on a variety of numerical problems as long as the direction numbers chosen are reasonable.

Here, we use two two-dimensional projections of a "bad" Sobol' sequence with direction numbers all set to 1 with $n = 1024$ points each, as well as a two-dimensional projection of a "good" Sobol' sequence, based on direction numbers provided in [27] for the so-called irreducible Sobol'- Nieddereiter sequences. With these point sets, we illustrate the following:

1. Even if the values of the t parameter are the same, the $C_b(\boldsymbol{k}; P_n)$ values can be very different.

2. How to find a good base to scramble in, i.e., the smallest prime b for each point set such that $C_b \leq 1$.

3. Why it is a good idea to use two different bases to measure the quality of a point set.

The second point, in particular, is of interest as it gives us a way to "repair" point sets by scrambling them in a different base than the one it was originally constructed in. We will see, specifically, that scrambling in a larger base has more potential to repair point sets.

Our simulation experiments detailed in this section suggest that the C_b criterion can be used to understand how applying nested scrambling in different bases affects different deterministic point sets.

First, we compare the different two-dimensional projections of a Sobol' sequence. The two "bad" projections are of poor quality both in terms of their t parameter (for both these projections, $t = 9$), and visually, as they do not cover the unit square. The one on the top row of Figure 3.1 is obtained by taking the projection of that sequence over coordinates (27,28) and the one on the middle row is obtained by taking the projection over coordinates (22,23). The bottom row of this image shows the "good" Sobol' sequence, which has a t parameter of 0. We denote these three point sets by "Bad Sobol' 1", "Bad Sobol' 2", and "Good Sobol'," respectively.

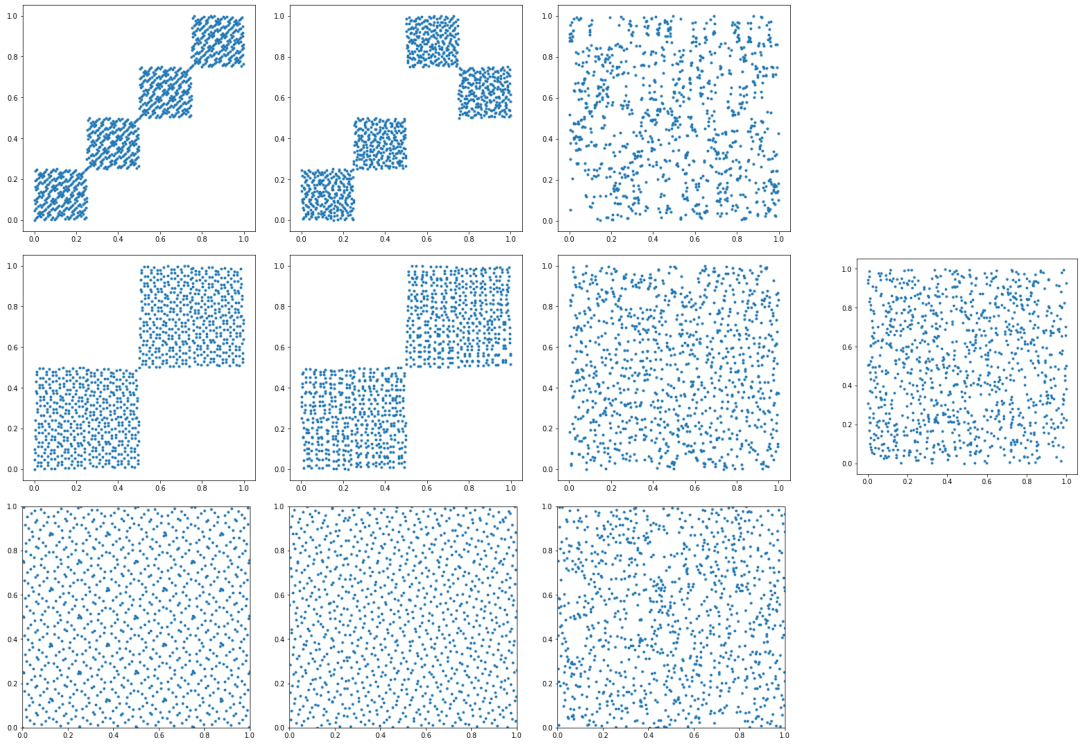


Figure 3.1: The left column shows three different $(t, m, 2)-$nets in base 2 with $m = 10$; the middle column shows the point sets after scrambling in base 2; the right column shows the point sets after scrambling in base 53. The separate image on the right is of 1024 points uniformly sampled over the unit square.

For the images in the left column of Figure 3.1, Table 3.1 gives the value of $\beta_{b,k} = \max_{\boldsymbol{k}:|\boldsymbol{k}|=k} C_b(\boldsymbol{k}; P_n)$ for $k \geq 1$, for $b = 2$. It also gives the maximum value $C_b = \max_{k\geq 1} \beta_{b,k}$, again for $b = 2$.

The plots in the middle column of Figure 3.1 show the point sets after being scrambled in base 2, using the nested uniform scrambling method of Owen [60]. Visually, we see that scrambling does not fix the issues of the deterministic point sets on the left – for the first two point sets, there are just as many areas without any points. This is consistent with the fact that scrambling does not change the $C_b(\boldsymbol{k}; P_n)$ values and thus the t values, so if they are large in a given base, scrambling in that base will not address or improve the lack of equidistribution.

However, when measuring $C_b(\boldsymbol{k}; P_n)$ in a base other than that used to construct P_n, if we find they are small (i.e., close to 1), it suggests that scrambling in that base could

improve the equidistribution and "repair" the point set. To illustrate this, we performed a base 53 scramble of the three point sets, with the resulting point sets shown on the right column of Figure 3.1. Visually, all point sets appear much better equidistributed after this base 53 scrambling, although the "Bad Sobol' 2" point set still appears to be better equidistributed than the "Bad Sobol' 1" point set. Note that in this case there is no parameter t that can be computed to assess the quality of ${}_{53}\tilde{P}_n$, as $n = 1024$ is not a power of b. However, there is no such restriction for the $C_{53}(\boldsymbol{k}; P_n)$ values. Table 3.2 gives the $\beta_{b,k}$ and C_b values for $b = 53$. The two point sets yield a maximum C_{53} of 2.9282 and 0.9498 for the two "bad" point sets, so the "Bad Sobol' 2" scrambled point set is c.q.e. in base 53. For the "Bad Sobol' 1" point set, C_{53} is much smaller than C_2 which explains why visually, the points are more equidistributed than they are after a base 2 scramble. This experiment shows that scrambling point sets constructed in base 2 in a larger base can be used to fix bad projections that are not "repaired" by the base 2 scrambling.

Despite having the same t value, the "Bad Sobol' 1" point set in Figure 3.1 visually appears to be worse than the "Bad Sobol' 2" point set, as there are larger regions with no points and the points are packed into a smaller region along the diagonal. However, we can see that the $C_b(\boldsymbol{k}; P_n)$ values are at least as large for the "Bad Sobol' 1" point set compared to the "Bad Sobol' 2" point set in both base $b = 2$ and $b = 53$. Thus, the $C_b(\boldsymbol{k}; P_n)$ values managed to differentiate the quality of the point sets when the t values could not.

k	1	2	3	4	5	6	7
Bad Sobol' 1	0.9990	1.9980	1.9941	3.9883	3.9726	3.9413	3.8788
Bad Sobol' 2	0.9990	1.9980	1.9941	1.9863	1.9707	1.9394	1.8768
Good Sobol'	0.9990	0.9971	0.9932	0.9853	0.9697	0.9384	0.8759
k	8	9	10	11	12	13	C_b
Bad Sobol' 1	3.7537	3.5034	3.0029	6.0059	4.0039	8.0078	**8.0078**
Bad Sobol' 2	1.7517	3.5034	3.0029	6.0059	4.0039	8.0078	**8.0078**
Good Sobol'	0.7507	0.5005	0	0	0	0	**0.9990**

Table 3.1: Values of $\beta_{b,k}$ and C_b for $b = 2$, for nets from left column of Figure 3.1.

k	1	2	C_b
Bad Sobol' 1	0.9498	2.9282	**2.9282**
Bad Sobol' 2	0.9498	0.7562	**0.9498**
Good Sobol'	0.9498	0.6221	**0.9498**

Table 3.2: Values of $\beta_{b,k}$ and C_b for $b = 53$, for nets from left column of Figure 3.1.

We now discuss how to use the C_b criterion to find good randomizations for the Sobol' sequence. This work has not yet been submitted for publication.

To randomize $(t, m, s)-$nets in base b, we use a base $\tilde{b}-$digital scramble, which preserves the number of shared initial digits between pairs of points when considering the base $\tilde{b}$ representations of the points. Note that the $\tilde{b}$ of the randomization can be a different base from the one the net the constructed in. That is, we do not need $\tilde{b} = b$. Since nested scrambling [60] satisfies the properties of a digital scramble as in Definition 2.4.4, it follows from the properties of the digital scramble that even if $\tilde{b} \neq b$, we get an unbiased estimator, as each point is uniformly distributed. The proof for the scrambled points being uniformly distributed given in [60] does not rely on how the original point is constructed, so the base b in which the point set was originally constructed does not affect whether the resulting scrambled estimator is unbiased or not.

Another way of measuring the quality of these point sets is to look for the smallest base $\tilde{b}$ that gives $C_{\tilde{b}} \leq 1$, that is, the smallest prime $\tilde{b}$ that we can use to scramble in to "repair" the point set. A "good" point set would require a small $\tilde{b}$ to repair, whereas a "bad" point set would require a larger $\tilde{b}$. This is because the larger $\tilde{b}$ is in relation to n, the fewer times we must "cut" the unit cube until there are more $\tilde{b}-$ary boxes than points, and thus it is easier for P_n to obtain small $C_{\tilde{b}}(\boldsymbol{k}, P_n)$ values. This is yet another way the $C_{\tilde{b}}$ values can be used to assess the quality of a point set and to compare point sets. Since we can use a different base other than the base the point set was constructed in to scramble and calculate the $C_{\tilde{b}}$ criterion, this again leads us to the idea of using multiple bases to evaluate the quality of a point set.

We illustrate this with the point sets in Figure 3.1. Table 3.3 shows the $C_{\tilde{b}}$ values for prime bases $\tilde{b}$ for the three point sets. The "Bad Sobol' 1" point set is visually the worst distributed over the unit square, and the "Good Sobol'" point set is the best distributed over the unit square, and the $C_{\tilde{b}}$ values show this as well. Firstly, the $C_{\tilde{b}}$ values for the "Bad Sobol' 1" point set are always the greatest, and the $C_{\tilde{b}}$ values for the "Good Sobol'" point set are always the smallest for every value of $\tilde{b}$. As well, the "Bad Sobol' 1" point set only achieves $C_{\tilde{b}} \leq 1$ at $\tilde{b} = 109$, while the "Bad Sobol' 2" point set achieves this at

$\tilde{b} = 53$ and the "Good Sobol'" point set achieves this at $\tilde{b} = 2$. This comparison between the three point sets suggests that "better" point sets can be repaired by being scrambled in a smaller base than a "worse" point set.

If P_n is such that the j^{th} coordinate of the points are all distinct, then $C_{\tilde{b}}$ will converge to 0, as eventually we will subdivide the unit cube such that each subdivision has at most one point. Thus, it is always possible to find a $\tilde{b}$ that satisfies $C_{\tilde{b}} \leq 1$. We stress that we want to scramble in the smallest base that satisfies $C_{\tilde{b}} \leq 1$, rather than picking any large $\tilde{b}$ that satisfies $C_{\tilde{b}} \leq 1$.

We have now shown multiple ways to be able to tell that the two "bad" Sobol' point sets are not well distributed over the unit square. Now, we explore how effective the "repairs" to these point sets are. We know that scrambling in a higher base can lead to a more uniform coverage over the unit square, so both point sets are scrambled 100 times in bases 53 and 109 to see the distribution of the C_2 values. If we scramble in base 2, the point sets would not be "repaired", as the C_2 values would not change after a base 2 scrambling – They would stay 8.0078 for both of these point sets.

Figure 3.2 shows the distribution of the C_2 values after scrambling in different bases. We can see that the "Bad Sobol' 1" point set (left side) yields higher C_2 higher values than the "Bad Sobol' 2" point set even after scrambling, which is consistent with all of our results above that suggest that the former is "worse" than the latter. Table 3.4 has the number of scrambles out of these 100 that have $C_2 < 1$, and again, our results are consistent with that the "Bad Sobol' 1" point set produces worse point sets even after scrambling. As well, we can see that since almost all the scrambles produced a C_2 value smaller than the original C_2 value of 8.0078, this suggests that scrambling in a larger base is very likely to improve the quality of the point set.

$\tilde{b}$	Bad Sobol' 1	Bad Sobol' 2
53	0	7
109	29	63

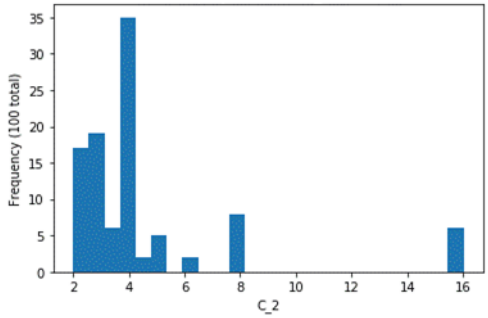
Table 3.4: Number of scrambles in base 53 and 109 that give $C_2 < 1$ for the two point sets from Figure 3.1, out of 100 scrambles.

Now, we want to test the performance of these Sobol' point sets on test functions to see if the criterion captures the performance of these point sets in integration problems.

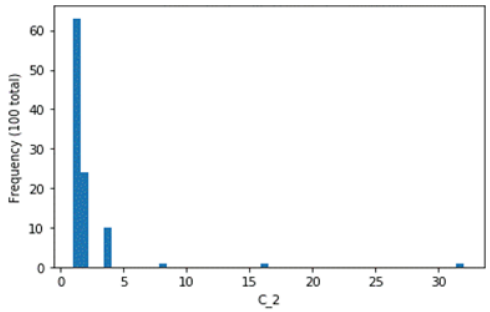
We use the test function $g_1(\mathbf{x}) = \Pi_{j=1}^{s} \frac{|4x_j - 2| + \alpha_j}{1 + \alpha_j}$ from Section 2.5 with $V = 25$ replications to estimate the RQMC error, with the following choices of α: 1) $\alpha_j = 0.01$, 2) $\alpha_j = 1$,

$\tilde{b}$	Bad Sobol' 1	Bad Sobol' 2	Good Sobol'
2	8.00782014	8.00782014	**0.99902248**
3	3.19839397	2.31319877	0.99804688
5	3.20389823	1.84955019	0.99609375
7	2.96586136	2.28284228	0.99414444
11	3.08081317	1.73006972	0.99023438
13	3.17720361	1.97143435	0.98829843
17	3.2267076	2.32623106	0.98440937
19	3.26624168	2.39643855	0.98243715
23	3.21070649	1.95126924	0.97862063
29	3.17757782	1.38567441	0.97280135
31	3.33191288	1.42376894	0.97070312
37	3.08418102	1.26503696	
41	3.02966153	1.23561408	
43	3.0747476	1.14729388	
47	2.97330347	1.04592803	
53	2.92818686	**0.94975333**	
59	2.56534473	0.94406578	
61	2.48646368	0.94194465	
67	2.25403035	0.93635447	
71	2.24247197	0.93274797	
73	2.04501352		
79	1.91837923		
83	2.06495334		
89	1.64839359		
97	1.34728395		
101	1.34383591		
103	1.1140163		
107	1.68311072		
109	**0.97538452**		
113	0.8912264		
127	0.87774163		
131	0.87537421		
137	0.87152523		

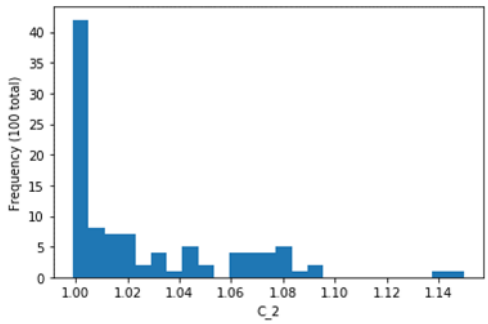
Table 3.3: $C_{\tilde{b}}$ values for the two point sets from Figure 3.1. Bolded values are the first $C_{\tilde{b}} \leq 1$ for each point set.



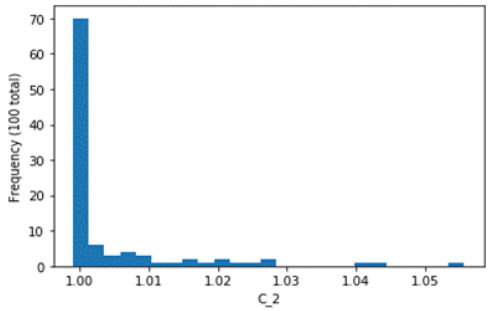
(a) Bad Sobol' 1, Scrambled in base 53



(b) Bad Sobol' 2, Scrambled in base 53



(c) Bad Sobol' 1, Scrambled in base 109



(d) Bad Sobol' 2, Scrambled in base 109

Figure 3.2: Distribution of C_2 values after scrambling 100 times in bases 53 and 109.

3) $\alpha_j = j$, 4) $\alpha_j = j^2$. The error of the MC estimator with $Vn = 25 \times 1024 = 25600$ points is also reported. As expected, the results in Table 3.5 show that the "Bad Sobol' 2" point set outperforms the "Bad Sobol' 1" point set in terms of estimated RQMC error for all choices of α when randomizing by scrambling in either base 53 or 109. The difference is more prominent in base 53, which is also to be expected because both point sets are c.q.e. in base 109 but only the "Bad Sobol' 2" point set is c.q.e. in base 53, implying that in base 53, the "Bad Sobol' 2" point set is a NLOD/NUOD sampling scheme, but not "Bad Sobol' 1". As well, the "Good Sobol'" point set performs the best when we randomize in base 2, which is expected as out of the three point sets, this one was the only one that was c.q.e. in base 2. Something that may seem unexpected is for the "Bad Sobol' 2" point set, the base 2 scrambling gives superior results than scrambling in base 53 or base 109, despite the point set being c.q.e. in bases 53 and 109. This is due to the fact that the function is symmetric, and the base 2 scrambling preserves the pattern of the point set having all the points being in the two quadrants along the main diagonal. The results here also illustrate the importance of choosing a good point set and randomization when using RQMC methods – the "Bad Sobol' 1" point set did not outperform MC when scrambled in base 2. Thus, we also explore the performance of these point sets on another function below.

Point Set	Base of Scramble	$\alpha = [0.01, 0.01]$	$\alpha = [1, 1]$	$\alpha = [1, 2]$	$\alpha = [1, 4]$
Bad Sobol' 1	2	3.13E-02	2.03E-03	1.04E-03	4.00E-04
Bad Sobol' 2	2	2.32E-06	6.61E-08	5.71E-08	2.53E-08
Good Sobol'	2	4.57E-08	8.06E-09	2.91E-09	1.07E-09
Bad Sobol' 1	53	2.01E-04	9.69E-06	8.88E-06	2.38E-06
Bad Sobol' 2	53	8.94E-05	5.38E-06	3.29E-06	2.28E-06
Good Sobol'	53	8.33E-05	4.94E-06	5.14E-06	2.74E-06
Bad Sobol' 1	109	1.13E-04	8.81E-06	7.46E-06	4.22E-06
Bad Sobol' 2	109	9.04E-05	7.80E-06	4.45E-06	3.39E-06
Good Sobol'	109	7.82E-05	9.45E-06	6.53E-06	5.32E-06
Monte Carlo	N/A	7.33E-04	1.68E-04	1.20E-04	9.59E-05

Table 3.5: Estimated RQMC errors of test function $g_1(\mathbf{x}) = \Pi_{j=1}^{s} \frac{|4x_j - 2| + \alpha_j}{1 + \alpha_j}$ for the point sets in Figure 3.1.

We now use the test function $g(\mathbf{x}) = \Pi_{j=1}^{s} 1 + c(x_j - 0.5)$ with $c = 0.25$ from Section 2.5 with $V = 25$ replications to estimate the RQMC error. This function is monotone in each coordinate, and being a two-dimensional function, this means that it is quasi-monotone.

Thus, the results from Section 2.4 stating that if we have a NLOD/NUOD sampling scheme, the variance is guaranteed to be lower than Monte Carlo apply. RQMC errors are given in Table 3.6. The error of the MC estimator with $Vn = 25 \times 1024 = 25600$ points is also reported. In base 2, the "Good Sobol'" point set performs the best, which follows from the fact that it is the only point set that is c.q.e. in base 2. In base 53, the "Bad Sobol' 1" point set performs the worst, which follows from the fact that it is the only one of the three that is not c.q.e. in base 53. In base 109, all the point sets are NLOD/NUOD sampling schemes, and thus it makes sense that the RQMC errors are similar. In addition, if we look at the two "bad Sobol'" point sets, scrambling in a larger base than what the point set is constructed in results in much better errors than scrambling in base 2, which still leaves large gaps within the unit square. In all cases, the error is lower when we use a scrambled Sobol' sequence than with Monte Carlo.

Point Set	Base of Scramble	RQMC Error
Bad Sobol' 1	2	1.64E-05
Bad Sobol' 2	2	1.53E-05
Good Sobol'	2	1.62E-11
Bad Sobol' 1	53	2.30E-07
Bad Sobol' 2	53	8.47E-08
Good Sobol'	53	1.51E-07
Bad Sobol' 1	109	2.13E-07
Bad Sobol' 2	109	2.66E-07
Good Sobol'	109	3.27E-07
Monte Carlo	N/A	9.52E-05

Table 3.6: Estimated RQMC errors of test function $g(\mathbf{u}) = (1+0.25(u_1-0.5))(1+0.25(u_2-0.5))$ for the point sets in Figure 3.1.

Although the point sets used in these examples are extremes in terms of quality, these experiments on two-dimensional point sets show us that the C_b value can be used to compare point sets as well as choose a base to use for randomization that can "repair" the point set.

3.2 Scrambling and negative dependence concepts for $(0, s)-$sequences

We now consider some examples of using the $C_b(\boldsymbol{k}; P_n)$ values on point sets that are two-dimensional projections of the Faure sequence.

The work in this section includes work that has not been published yet and is the topic of the working paper [20], which is done in collaboration with Christiane Lemieux and Henri Faure. Similar work, also from this working paper, on the Halton sequence will be discussed in the next chapter, in Section 4.6.

The Faure sequence, along with some of its generalizations that have been proposed over the years, [26, 80, 81] stand out among low-discrepancy sequences thanks to the fact that it achieves perfect equidistribution. The Faure sequence is a special case of a digital $(t, s)-$sequence in base b, and the perfect equidistribution is captured by the quality parameter, t, which as mentioned previously, can be shown to be 0 [24].

While the Faure sequence achieve an optimal equidistribution as described above, this optimal behaviour can require a very large number of points before being observed, depending on the bases used, which in turn depends on the dimension s of the sequence, as the larger s is, the larger the prime bases required are. For larger bases, if we use the first n points of the sequence and n is too small compared to the number of points where the optimal equidistribution properties are shown to hold, the corresponding point set may not be of very good quality, thus yielding quasi-Monte Carlo estimators that are potentially less accurate than those obtained using the Monte Carlo method. This phenomenon can be seen in the top-left image of Figure 3.3, where there are not enough points to fill the unit square when the sequence is constructed in base 53.

3.2.1 Assessing the quality of point sets derived from $(0, s)-$sequences

In this section, we consider Faure [24] sequences, which are (t, s)-sequences in a prime base b, and, as mentioned in Section 2.2.4, have $t = 0$ if $b \geq s$. If we want $t = 0$, then in large dimensions we must work with large bases. Hence, it is typical to use a number of points n that is not a power of b, as in large bases the powers grows very quickly. For example, in base 53, the first four integer powers are 53, 2809, 148877, and 7890481, which is very restrictive for choices of n. For this reason, we want to make sure the construction used is such that the first n points are uniformly distributed, for any value of n. As discussed in, e.g., [47, Chapter 5.4.4], when working with the original Faure sequences, there can be some

unwanted behaviour for smaller values of n, i.e., smaller than b^d where d is the dimension of the space (or projection) considered. It is possible to construct $(0, s)$-sequences with better properties (often referred to as *generalized Faure sequences* see e.g., [47, Chapter 5.4.4]), but it can be challenging to quantify what we mean by "better" since $t = 0$ by definition for all these sequences, and we also know from Proposition 4.5 in [90] that their first n points form point sets that are all c.q.e. in base b. This is where the $C_b(\boldsymbol{k}; P_n)$ values can help. Figure 3.3 shows different point sets obtained from $(0, 2)$-sequences in base 53.

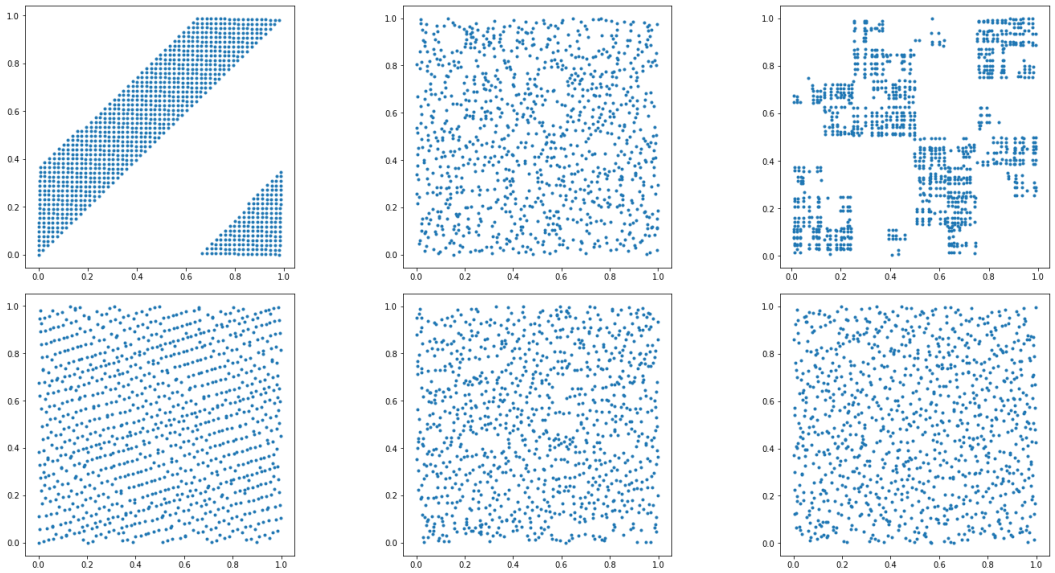


Figure 3.3: First 1024 points of $(0, 2)$-sequences taken from coordinates $(49, 50)$ of the following construction: original Faure sequence in base 53 (top left); generalized Faure sequence in base 53 randomized using random linear scrambling (bottom left); the middle and right columns shows the point sets after nested uniform scrambling in base 53 and 2, respectively.

Since the point sets in the left column both come from the first $n = 1024$ points of a $(0, 2)$-sequence in base 53 (as $1024 < 53^2$), they have the same values of $C_{53}(\boldsymbol{k}; P_n)$, namely $\beta_{53,k} = 0.94975$ for $k = 1$ and is 0 otherwise. We can interpret this as follows: both point sets should have similarly good uniformity properties after being scrambled in base 53, since $C_{53} < 1$ for both point sets. This is confirmed by the two figures in the middle column being very similar, although the base 53 digital scrambling was applied to point sets (left column) that appear very different: the top one having much less desirable uniformity than the bottom.

In order to detect the difference between the two point sets in the left column, we compute the $C_2(\boldsymbol{k}; P_n)$ values for both. The motivation for doing this as follows: as seen in the right column of Figure 3.3 and the middle column of Figure 3.1, a base 2 digital scrambling does not address issues in a badly designed point set. This means scrambling in base 2 can only produce a uniform point set if the point set being scrambled is already uniform with respect to that base, and not only with respect to base 53. This is precisely what the $C_2(\boldsymbol{k}; P_n)$ can detect.

Since the $C_2(\boldsymbol{k}; P_n)$ values capture the dependence structure of the base 2 scrambling of P_n and we see that the two point sets look very different from each other after scrambling in base 2 (right column), those values should detect the difference between the point sets on the left. In other words, since the upper right point set is not uniform even though a base 2 scrambling has been applied, the upper left point set is not uniformly distributed with respect to base 2. We expect this will be captured by larger $C_2(\boldsymbol{k}; P_n)$ values for this point set. Similarly, since the lower right point set looks uniform, the lower left point set is not only uniformly distributed with respect to base 53 but also with respect to base 2. Table 3 shows the $C_2(\boldsymbol{k}, P_n)$ values of both point sets on the left. We see that the $C_2(\boldsymbol{k}; P_n)$ values do indeed detect the visual difference we see in the point sets on the left, with the top one giving $C_2 = 16.8289$ and the bottom one giving $C_2 = 1.0753$.

k	1	2	3	4	5	6	7	8	9
Faure (top left)	1.0001	1.0975	1.4429	1.8507	2.0287	2.2466	2.3306	2.5191	3.8270
GFaure (bottom left)	0.9991	0.9976	0.9939	0.9887	0.9791	0.9554	0.9135	0.8436	0.7840
k	10	11	12	13	14	15	16	C_b	
Faure (top left)	5.8905	7.9062	11.1830	13.2786	16.8289	15.6403	0	**16.8289**	
GFaure (bottom left)	0.9717	1.0753	0.7820	0.4536	0.4692	0.6882	0.6256	**1.0753**	

Table 3.7: Values of $\beta_{b,k}$ and C_b for $b = 2$ for point sets in left column of Figure 3.3.

This experiment suggests that for future work, the $C_2(\boldsymbol{k}; P_n)$ values can be used to define a criterion that can be used to search for good generalized Faure sequences, along the lines of what will be done in Chapter 4 for the Halton sequence.

3.2.2 Modifications to C_b to capture only lower-dimensional projection properties

So far, we have compared different point sets using the quantity $C_b(\boldsymbol{k}; P_n)$ for different vectors $\boldsymbol{k}$ corresponding to partitions of the unit hypercube $[0, 1)^s$ that are of particular

interest. To decrease the runtime of calculating the C_b value as well as to only capture the information that is of the most interest to us, we introduce a new criterion based on the $C_b(\boldsymbol{k}; P_n)$ values where we restrict $\boldsymbol{k}$. This allows us to make comparisons between point sets using much less computational resources, in terms of both memory and time.

Define $c(b, \mathcal{K}; P_n) = \max_{\boldsymbol{k} \in \mathcal{K}} C_b(\boldsymbol{k}; P_n)$, where $\mathcal{K} \subset \mathbb{N}^s$. We are interested in a special class of subsets $\mathcal{K}$ defined by two parameters (d, w) where $2 \leq d \leq s$ and $w \geq 1$, in the following way:

$$\mathcal{K}_{d,w,s} = \{\boldsymbol{k} \in \mathbb{N}^s : 2 \leq \sum_{j=1}^{d} \mathbf{1}_{k_j > 0} \leq d, r(\mathbf{k}) \leq w\}$$

where $r(\boldsymbol{k}) = \max\{1 \leq j \leq s : k_j > 0\} - \min\{1 \leq j \leq s : k_j > 0\}$ can be thought as the *range* of indices where a non-zero component of $\boldsymbol{k}$ can be found. Hence, d refers to the largest number of non-zero components allowed for $\boldsymbol{k}$ to be included in $\mathcal{K}_{d,w,s}$ and w is in some sense a window size, which limits the range of the indices j where a non-zero k_j is found.

For instance, if $s = 4$, $d = 2$ and $w = 2$, then

$$\mathcal{K}_{d,w,s} = \{(k, \ell, 0, 0), (k, 0, \ell, 0), (0, k, \ell, 0), (0, k, 0, \ell), (0, 0, k, \ell) : k, \ell > 1\}.$$

That is, we have excluded all vectors $\boldsymbol{k}$ with only 1 non-zero component or with more than 2 non-zero components, and we also excluded vectors such as $(1, 0, 0, 3)$ because its range is 3 which is larger than the largest allowed range of $w = 2$.

As an alternative to the criterion $c(b, \mathcal{K}; P_n)$ which returns the worst (largest) $C_b(\boldsymbol{k}; P_n)$ value, we also consider one based on the mean, defined as $\bar{c}(b, \mathcal{K}; P_n) = \frac{1}{|\mathcal{K}|} \sum_{\boldsymbol{k} \in \mathcal{K}} C_b(\boldsymbol{k}; P_n)$.

This restriction still gives us a way to compare point sets, as many high-dimensional functions have low effective dimension. In addition, we may be most interested in breaking up the correlation between consecutive coordinates – for example, in Chapter 4, we will use this restricted criterion with a small window size to search for factors to generalize the Halton sequence for this reason.

3.2.3 Assessing different forms of scrambling for the Faure sequence

Now that we have shown that the framework of negative dependence can be used to assess the quality of point sets coming from the initial portion of a $(0, s)-$sequence, we attempt to answer the following questions: when a deterministic $(0, s)-$sequence has known defects

for small sample sizes, should we address these defects by finding a "good" deterministic scrambling, or by applying random scrambling? On one hand, one would think that a very well-chosen deterministic scrambling should do better than one randomly chosen, and would have a less computationally demanding implementation since the point set could then be used with a "cheap" randomization such as a digital shift for the purpose of error estimation. On the other hand, finding a good deterministic scrambling that performs well on a large variety of problems may not be possible, and from that point of view, perhaps random scramblings are a better option, especially since the optimal equidistribution of the Faure sequence is a sufficient and necessary condition for random scrambling to induce a certain form of negative dependence, meaning that random scrambling can "repair" the defects of these sequences. And if we choose to use a deterministic scrambling, how do we choose one, and how do we assess whether it is good or not? We extend our work to use the framework of dependence to assess the benefits of scrambling randomly versus deterministically using well-chosen factors, for the Faure sequence.

Previously, we compared the Faure sequence with a generalized Faure sequence in base 53 obtained by randomly choosing nonsingular lower triangular matrices and multiplying them with original Faure sequence matrices. As mentioned previously in Section 3.2, the original Faure sequence has poor projection properties if n is not large enough.

To alleviate this problem, several authors have proposed to apply certain types of *scrambling* to the Faure sequence. A few different generalizations of the original construction from [24] have been proposed, starting with [81] and then [49]. In addition, random scramblings have been proposed for digital (t, s)-sequences [52, 60]. Both deterministic and random scramblings are widely accepted as providing an improvement to the original constructions, especially for small number of points in medium to large dimensions.

We also examine how the permutations from [25] can be used not only for van der Corput sequences as originally intended but can also be used as factors for Faure sequences, thereby proposing a new form of deterministic scramblings for the latter. Numerical experiments comparing deterministic and random scramblings are performed on a variety of problems to provide empirical evidence toward answering our main question.

These proposed permutations from [25] are especially useful because they can be obtained via a recursive process (over the bases b) consisting of a descent method, which is illustrated using checkerboards in Section 3.1 of [70]. As can be seen there, even bases (and thus checkerboards with an even dimension) can be intricated together to create a larger point set. To get a permutation for an odd base, a point needs to be inserted in the middle of an even-sized checkerboard. The process begins with the two first points (i.e., base 2) and grows. This construction is useful because of its simplicity and step-by-step induction.

Algorithm 2 summarizes the steps for generating a permutation π_b for any integer b (in fact, the recursive algorithm generates all permutations π_j for $j \leq b$).

Algorithm 2: Algorithm to generate permutations from [25].

INPUT:b;
OUTPUT: list of permutations $(\pi_2, \ldots, \pi_b)$;
j = 2;
$\pi_j = (0,1)$;
while $j \leq b$*:* **do**
  j = j + 1;
  if j *is even:* **then**
    $\pi_j = (2\pi_{j/2}, 2\pi_{j/2} + 1)$;
  if j *is odd:* **then**
    temp = π_{j-1};
    k = $\frac{j-1}{2}$;
    for $\ell = 1$ *to* $j - 1$*:* **do**
      if $\pi_j[\ell] \geq k$*:* **then**
        $\pi_j[\ell] = \pi_j[\ell] + 1$
    for $\ell = k + 1$ *to* j*:* **do**
      $\pi_j[\ell + 1] = \pi_j[\ell]$
    $\pi_j[k + 1] = k$;
return $(\pi_2, \ldots, \pi_b)$

Here, we propose to use these permutations to create generalized Faure sequences using diagonal non-singular lower-triangular matrices based on a factor f_j, $j = 1, \ldots, s$, where we assume $b = s$. Clearly, the factors f_j should be non-zero, so we cannot simply use $f_j = \pi_b[j]$. We propose two ways to address this, where here $j = 1, \ldots, s - 1$. That is, in both cases, we only obtain $s - 1$ factors, and hence can only define a $(s - 1)-$dimensional sequence based on this method. Specifically, the non-singular lower-triangular matrices we multiply the generating matrices by are of the form $\text{diag}([\text{f}_\text{j}] \times (\text{s} - 1))$.

With the first method, we simply observe that all permutations from [25] are such that $\pi_b[0] = 0$, so we simply omit the first term of the permutation. With the second method, we add an "offset term" $m = \lfloor b/2 \rfloor$ (modulo b) to each term of the permutation, which based on the algorithm used to generate π_b, is such that we now have $\pi_b[m] = 0$. That is, the addition of the offset term has the effect of placing the 0 in the middle of the permutation vector. More precisely, the two different methods for using the permutations from [25] to

define factors for the Faure sequence are:

1. Let $f_{j,1} = \pi_b[j+1]$ for $0 \leq j \leq b-2$.
2. Let $f_{j,2} = (\pi_b[j] + m) \bmod b$ for $j = 0, \ldots, m-1$ and $f_j = (\pi_b[j+1] + m) \bmod b$ for $j = m, \ldots, b-2$

3.2.4 Comparisons based on negative dependence criterion

In Tables 3.8 and 3.9, we compute the two criteria as explained in Section 3.2.2 for point sets based on the (generalized) Faure sequence, where "Faure 1992" and "Offset" respectively refer to the two methods for fixing the factors f_j created using Algorithm 2, and "Regular" refers to the original construction from [24]. The tables differ in the samples size n used for the point sets. In all cases, we consider every two-dimensional projection over the point set when computing the criteria. In the tables, b refers to the base the point set was constructed in, and all the criteria values are computed in base 2.

Faure Type	b	s	d	n	$c(2, \mathcal{K}_{d,w=s,s}, P_n)$	$\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$
Regular	5	4	2	3125	0.99968	0.99968
Faure 1992	5	4	2	3125	0.99968	0.99968
Offset	5	4	2	3125	0.99968	0.99968
Regular	13	12	2	2197	1.755691	1.422891
Faure 1992	13	12	2	2197	1.782858	1.162811
Offset	13	12	2	2197	1.782858	1.136818
Regular	53	52	2	2809	6.5223	1.584402
Faure 1992	53	52	2	2809	14.88912	2.019255
Offset	53	52	2	2809	14.88912	1.892771

Table 3.8: Values of criteria based on $C_2(\boldsymbol{k}; P_n)$ for point sets obtained from (generalized) Faure Sequences.

Faure Type	b	s	d	n	$c(2, \mathcal{K}_{d,w=s,s}, P_n)$	$\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$
Regular	5	4	2	5000	1.19561576	1.0977079
Faure 1992	5	4	2	5000	3.35611442	1.4793424
Offset	5	4	2	5000	3.35611442	1.5871413
Regular	13	12	2	5000	7.71906317	3.4452165
Faure 1992	13	12	2	5000	9.54395039	1.9958566
Offset	13	12	2	5000	7.63516031	2.1053137
Regular	53	52	2	5000	111.422999	16.197021
Faure 1992	53	52	2	5000	170.658418	4.89776
Offset	53	52	2	5000	153.710041	4.696813

Table 3.9: C_2 values of point sets based on the Faure Sequence with $n = 5000$.

We can see that even though the maximum value taken by $C_2(\boldsymbol{k}; P_n)$ for $\boldsymbol{k} \in \mathcal{K}$ for is not smaller for the two generalized Faure sequences compared to the original Faure sequence, the average over all two-dimensional projections is smaller. This means that after multiplying the generating matrices by the chosen factors ("Faure 1992" or "Offset"), there are fewer "poor" two-dimensional projections, but the poorest projections are worse than prior to applying the factors.

3.2.5 Integration problems

In this section, we numerically investigate the fundamental question of whether it is best to randomize via random scrambling or via a well-chosen generalized construction that can then be randomized using a simple digital shift.

We use the following test functions from Section 2.5: for h_0, h_1, and g we respectively have $\mu(h_0) = 0$, $\mu(h_1) = s/3 + s(s-1)/4$ and $\mu(g) = 1$. We also consider experiments on the stochastic activity network problem.

The different constructions and scramblings are compared using two approaches in this section. First, we plot the MSE or variance as a function of n to see how quickly they converge to 0. Then, we fix n and plot a histogram of the MSE or Variance constructed from randomly scrambled point sets and look at where different deterministically scrambled point sets compare with this distribution, as approximated by the histogram.

Convergence results

In all the results of this section, the Faure sequence in constructed in base b, where b is the smallest prime number b such that $b \geq s$. Note that for this experiment, all values of n plotted in the graph are an integer multiple of an integer power of this base b.

What is plotted on Figures 3.4 to 3.7 are the MSE for functions h_0, h_1, and g, and the variance for the SAN. In all cases, the MSE or variance is estimated using $V = 25$ randomizations.

For the Faure sequence, we compare the following constructions:

1. Faure sequence, randomized with a digital shift ("Regular, Shifted");
2. Generalized Faure sequence, using factors from Algorithm 2, then randomized with a digital shift ("Faure 1992, Shifted");
3. Generalized Faure sequence, using factors from Algorithm 2 with an offset term added, then randomized with a digital shift ("Offset, Shifted");
4. Faure sequence, randomized with Owen's scrambling ("Regular, Scrambled").

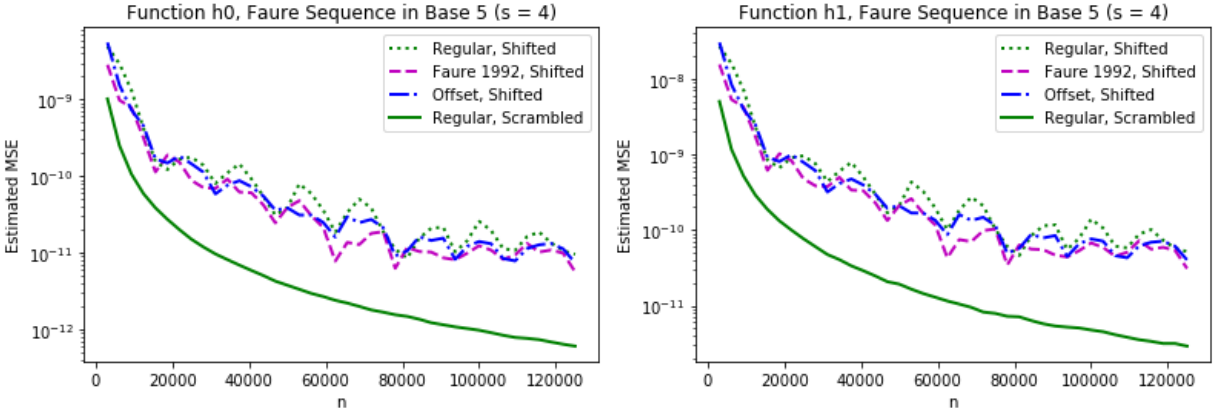


Figure 3.4: Estimated MSEs of test functions at $n \in \{3125m, 1 \leq m \leq 40\}$, using the 4-dimensional Faure Sequence constructed in base 5, using factors [3,2,1,4] (Faure 1992) and [3,1,4,2] (Offset).

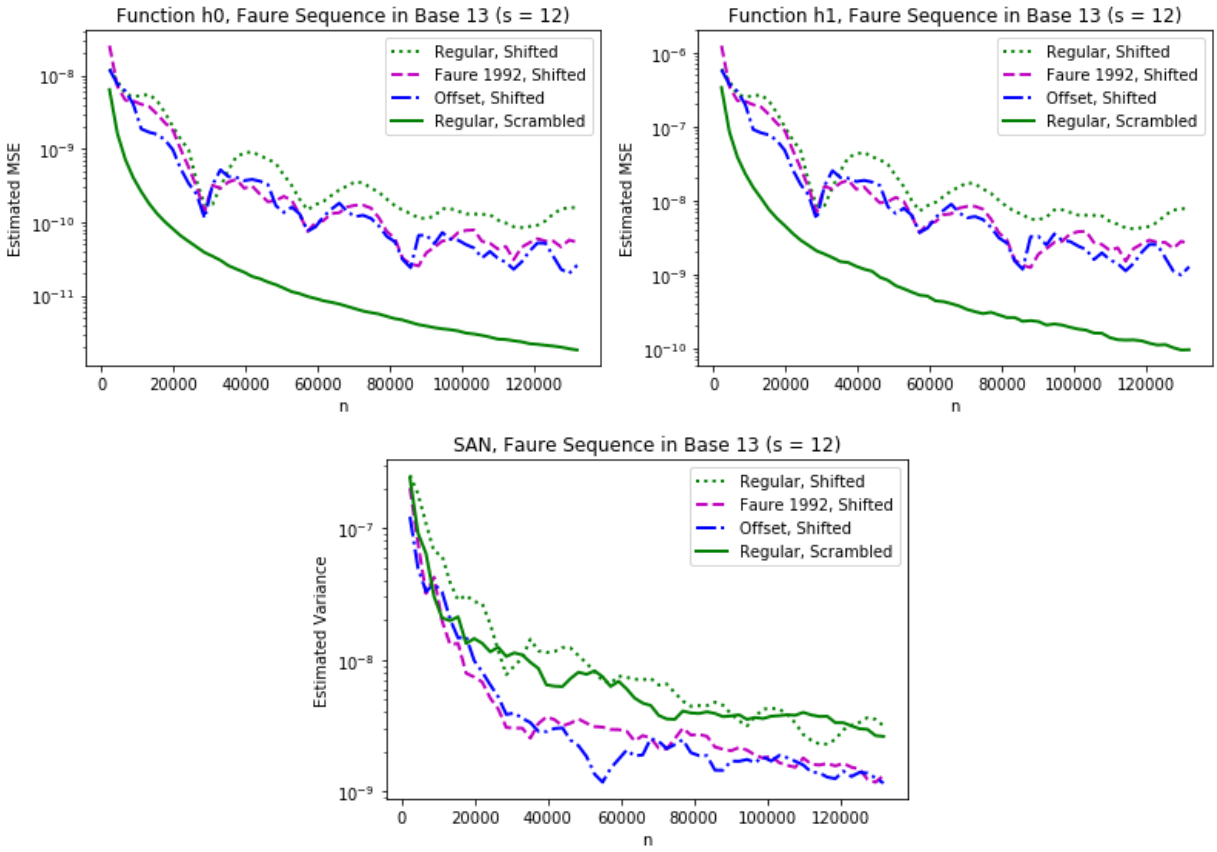


Figure 3.5: Estimated MSEs and Variances at $n \in \{2197m, 1 \leq m \leq 60\}$ of test functions using the 12-dimensional Faure Sequence constructed in base 13, using factors [4, 9, 2, 7, 11, 6, 1, 5, 10, 3, 8, 12] (Faure 1992) and [7, 11, 3, 9, 1, 5, 8, 12, 4, 10, 2, 6] (Offset).

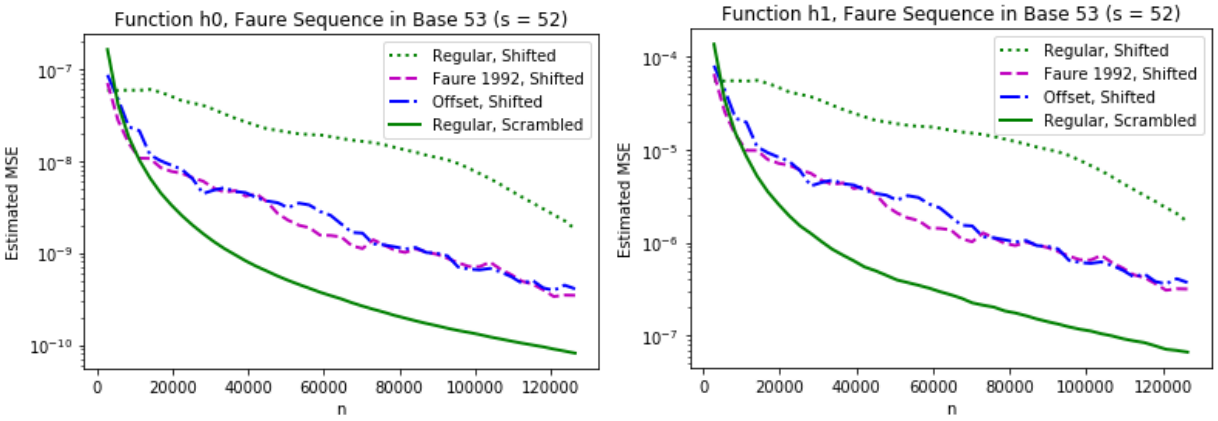


Figure 3.6: Estimated MSEs of test functions at $n \in \{2809m, 1 \leq m \leq 45\}$, using the 52-dimensional Faure Sequence constructed in base 53, using factors [16, 37, 8, 29, 45, 24, 4, 20, 41, 12, 33, 49, 2, 18, 39, 10, 31, 47, 27, 6, 22, 43, 14, 35, 51, 26, 1, 17, 38, 9, 30, 46, 25, 5, 21, 42, 13, 34, 50, 3, 19, 40, 11, 32, 48, 28, 7, 23, 44, 15, 36, 52] (Faure 1992) and [27, 43, 11, 35, 3, 19, 51, 31, 47, 15, 39, 7, 23, 29, 45, 13, 37, 5, 21, 1, 33, 49, 17, 41, 9, 25, 28, 44, 12, 36, 4, 20, 52, 32, 48, 16, 40, 8, 24, 30, 46, 14, 38, 6, 22, 2, 34, 50, 18, 42, 10, 26] (Offset).

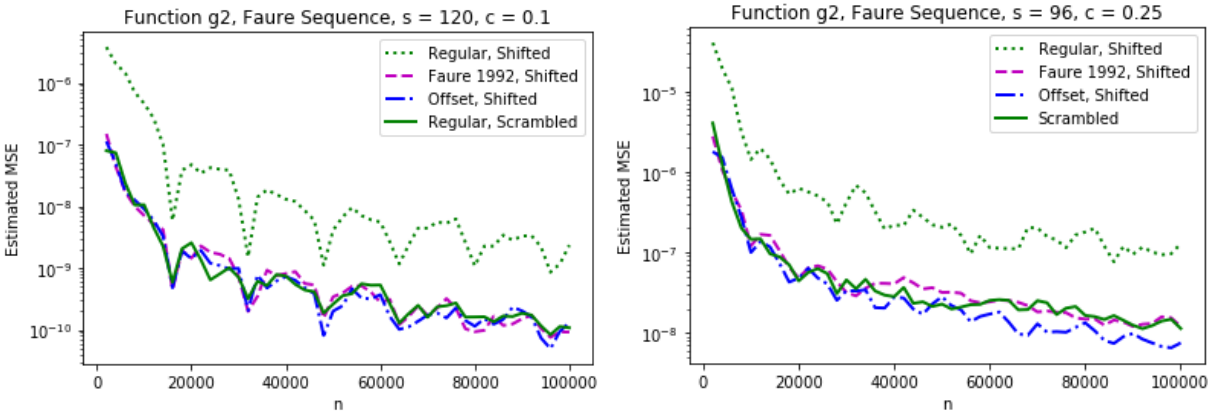


Figure 3.7: Estimated MSEs at $n \in \{2000m, 1 \leq m \leq 50\}$, when integrating test function g using the Faure Sequence.

The majority of the results show that scrambling is superior to generalizing then randomizing with a digital shift. For the Faure sequence, when n is an integer multiple of a

larger power of the constructing base b, we can see an improvement in performance for the shifted results.

The results from this numerical experiment can also be somewhat explained by the criterion values in Tables 3.8 and 3.9. If the c values are high, that suggests a "bad" point set that cannot be fixed via a shift, as a digital shift has little effect on the quality of a point set. However, even a "bad" point set can be fixed with scrambling in a base b such that $C_b \leq 1$. Since the values of n in Tables 3.8 and 3.9 are much smaller than the values of n used in the convergence plots, not all the behaviour can be explained by the criterion values.

In Figure 3.8, we do comparisons on the 13-dimensional SAN problem, as well as 13-dimensional versions of the test functions h_0 and h_1, with $n \in \{2197m, 1 \leq m \leq 60\}$. This problem set does not use the generalized Faure sequence using factors from Algorithm 2. Rather, it is the one of Tezuka and Tokuyama [81], where the non-singular lower-triangular matrix used to multiply the generating matrices by are the lower-triangular powers of the Pascal matrix: they are the the transpose of the original generating matrix given by powers of the Pascal matrix. Again, we can see that scrambling is superior to generalizing then randomizing with a digital shift.

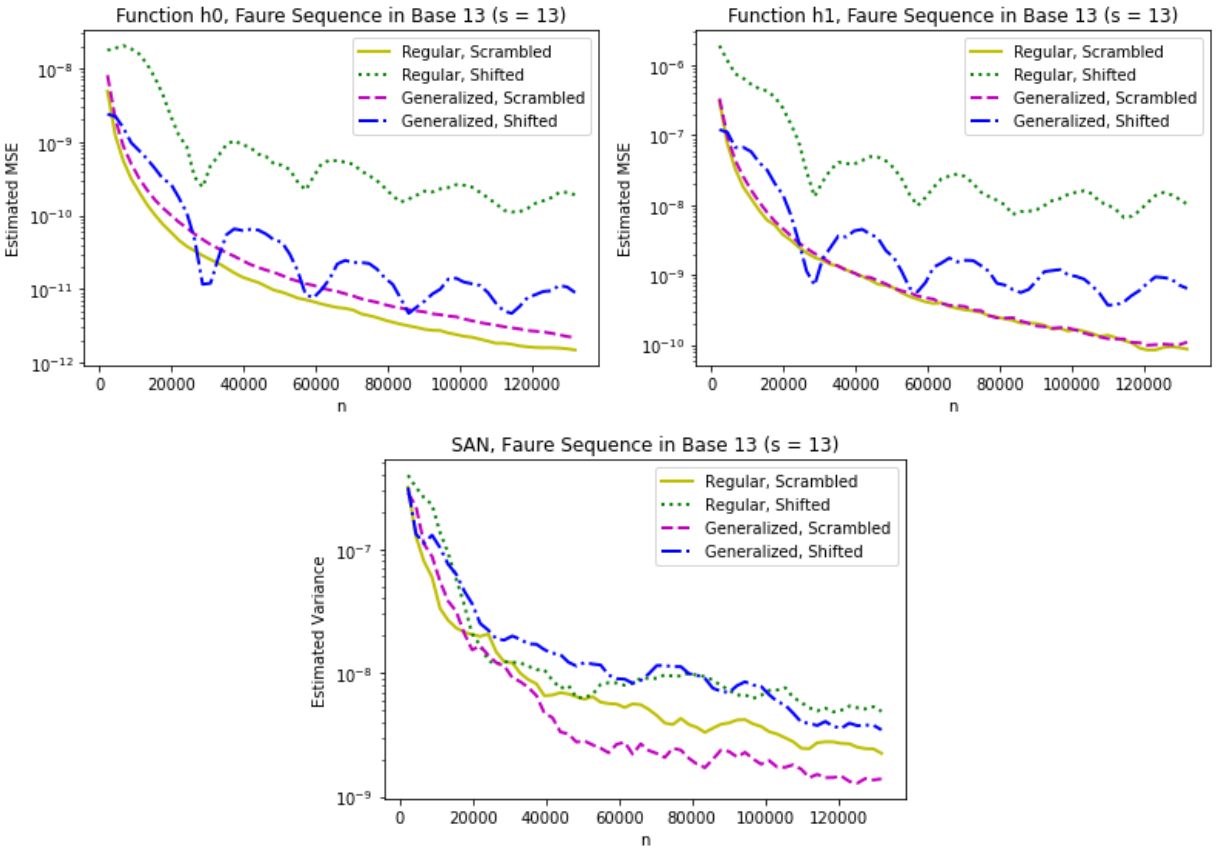


Figure 3.8: Estimated MSEs and Variances of test functions using the 13-dimensional Faure Sequence constructed in base 13. The generalized Faure sequence here is the one of Tezuka and Tokuyama. The $c(2, \mathcal{K}_{d=2,w=13,s=13}, P_{n=2197})$ value for this sequence is 2.39 and this value is for the projection over [1,4], and the $c(2, \mathcal{K}_{d=3,w=13,s=13}, P_{n=2197})$ value for this sequence is 27.82, and this value is for the projection over [2, 5, 10].

Comparing results using histograms

To compare the use of deterministic permutations versus scrambling, we use histograms to compare the integration error of the functions h_0 and h_1 obtained using specific generalized Faure sequences— randomized via a digital shift —with the error distribution obtained using a base b nested scrambling. If a specific construction can easily be "beaten" by (have a larger error than) a randomized sequence obtained through scrambling, then this would be an argument in favour of using scrambling to improve the sequence instead of relying

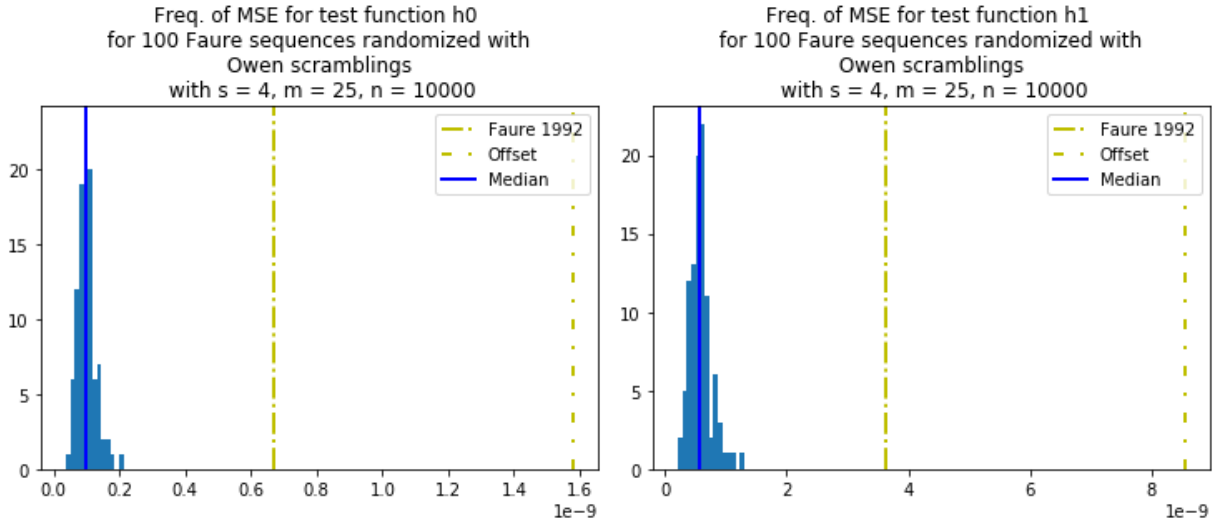
on a specific generalized sequence construction. If a construction consistently ranks better than most scrambled sequences, then this would be an argument to use those instead of scrambled sequences.

We consider 3 values of s: 4, 12, and 52. For the Faure sequence, the constructing base is the smallest prime larger or equal to s. Here, it is always taken equal to $s+1$. The reported error is the Mean Squared Error (MSE), and is estimated using $V = 25$ replications and a sample size of $n = 10000$, except in the $s = 52$ case where $n \in \{2809, 10000\}$. We chose to visualize the $n = 2809, s = 52$ case as in Section 3.2.5, the scrambled Faure sequence did not seem to outperform the generalized Faure sequence.

The experiments on the histograms can be thought of taking a "slice" of the results from Section 3.2.5 at specific values of n. Note that the value of $n = 10000$ is not explicitly used in the convergence plots.

We compare the distribution of randomly scrambled point sets (as shown by the histogram) with the same deterministically chosen factors as in Section 3.2.5, as well as the Monte Carlo methods.

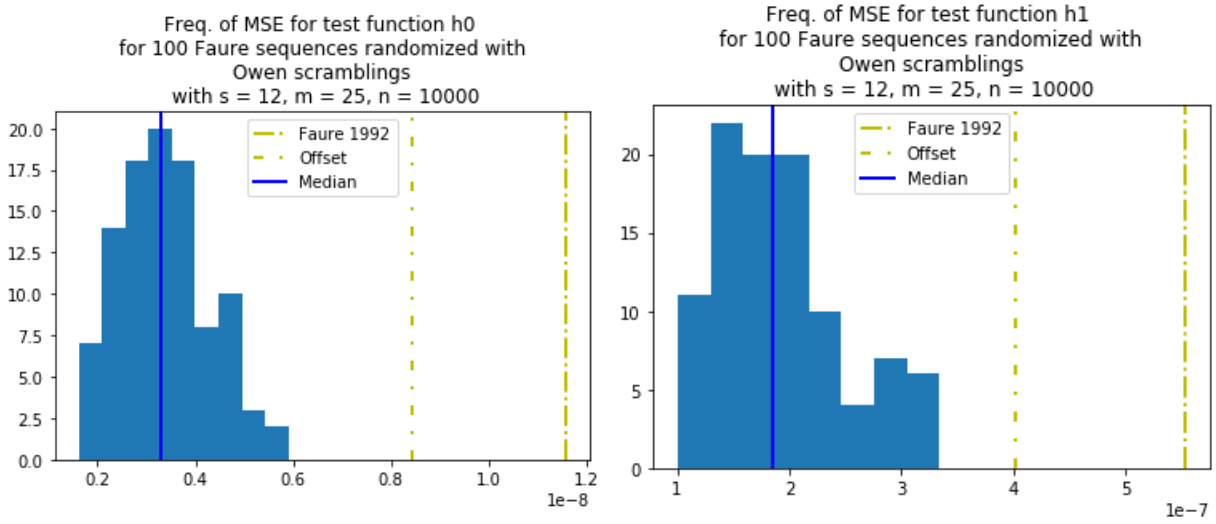
Some values (namely "Regular Shifted" and Monte Carlo) obtained are sometimes quite large compared to the rest of the results. Therefore, to better visualize the results, these large values have been excluded from the plots and their error values are reported in the caption instead.



(a) Faure: 9.516e-10, MC: 3.879e-06

(b) Faure: 5.144e-09, MC: 2.207e-05

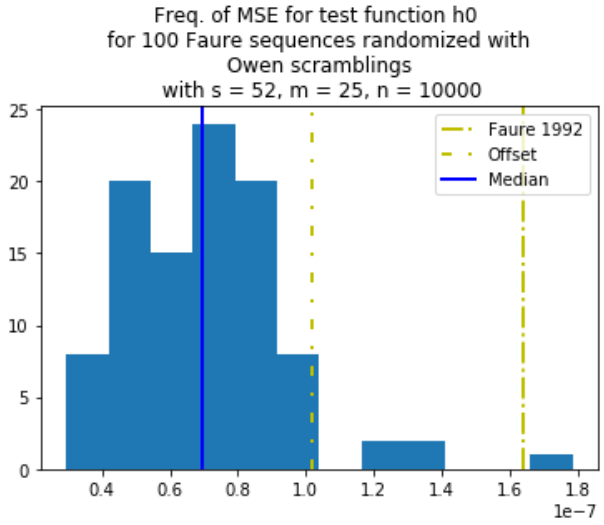
Figure 3.9: Comparing Nested Scrambling of the Faure sequence with other Faure sequences with $s = 4$.



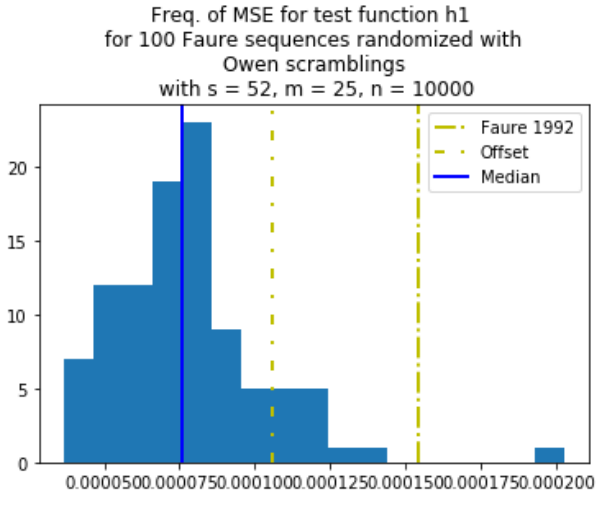
(a) Faure: 3.912e-08, MC: 1.163e-05

(b) Faure: 1.86e-06, MC: 0.000584

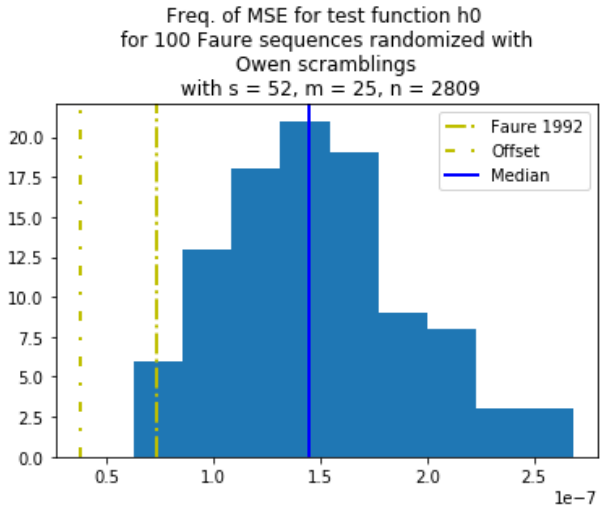
Figure 3.10: Comparing Nested Scrambling of the Faure sequence with other Faure sequences with $s = 12$.



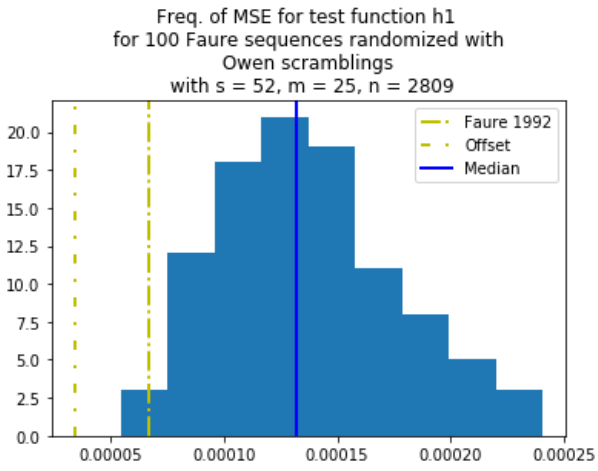
(a) Faure: 8.763e-07, MC: 5.0247e-05



(b) Faure: 0.0006621, MC: 0.04699



(c) Faure: 4.573e-08, MC: 0.0001795



(d) Faure: 4.230e-05, MC: 0.1677

Figure 3.11: Comparing nested Scrambling of the Faure sequence with other Faure sequences with $s = 52$.

The results suggest that in almost all cases, scrambling outperforms generalizing and then randomizing with a digital shift. The exception is the case where $s = 52, b = 53, n = 2809$, where generalizing and then applying a shift is more accurate (i.e., in the lower

quartile of the histogram) scrambling, but this result cannot be extrapolated into the general case. When n is a small power of the constructing base b, the digital shift is more likely to perform well compared to scrambling.

3.3 Comparing Sobol' and Faure sequences

We now compare the Sobol' sequence and the Faure sequence. The Faure sequence is used to create $(0, m, s)-$nets, that is, the t parameter is guaranteed to be 0, and the Sobol' sequence does not have any such property. Thus, one might think that the Faure sequence is superior to the Sobol' sequence. However, the t parameter does not allow us to make comparisons based on an arbitrary number of points, nor does it allow us to make comparisons in an arbitrary base. Criteria based on the $C_b(\boldsymbol{k}; P_n)$ values allows us to do so, and thus will give us more meaningful comparisons.

3.3.1 Two-dimensional examples

First, we consider the projection over coordinates (16,17) of the first 1024 points of the Sobol' and Faure sequences, the latter being constructed in base 17, and the former based on direction numbers provided in [27] for the so-called irreducible Sobol'- Nieddereiter sequences.

Table 3.10 shows the $C_b(\boldsymbol{k}, P_n)$ values for b equal to 2, 3, and 17.

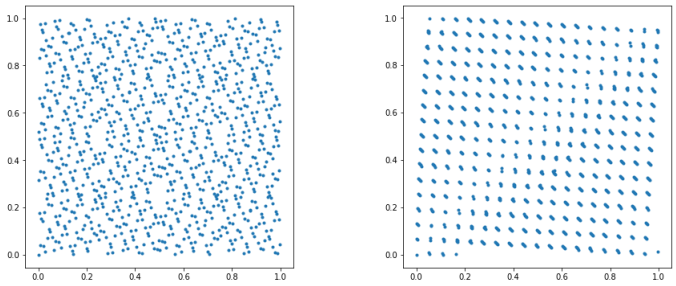


Figure 3.12: First 1024 points of Sobol' (left) and Faure (right) sequence over coordinates (16,17).

k	1	2	3	4	5	6	7	8	9
Sobol' ($b = 2$)	0.9990	0.9971	0.9932	0.9853	0.9697	0.9384	0.8759	0.7507	1.5015
Faure ($b = 2$)	0.9990	1.0032	1.0104	1.0134	1.0133	1.3788	1.4773	2.0406	2.4261
Sobol' ($b = 3$)	0.9980	0.9923	0.9768	0.9384	0.8662	0.7404	0.5553	0.1754	0.0752
Faure ($b = 3$)	0.9981	1.0069	1.0086	1.2735	1.9723	2.5401	3.2276	6.0628	7.3655
k	10	11	12	13	14	15	16	17	C_b
Sobol' ($b = 2$)	1.0010	2.0020	0	0	0	0	0	0	**2.0020**
Faure ($b = 2$)	2.7605	3.0928	4.9267	6.5376	7.2883	3.8162	4.0039	0.7507	**7.2883**
Sobol' ($b = 3$)	0.1127	0	0	0	0	0	0	0	**0.9980**
Faure ($b = 3$)	3.6076	7.1024	0	0	0	0	0	0	**7.3655**
k	1	2	3	4					C_{17}
Sobol' ($b = 17$)	0.9844	0.8436	0.0938	0.3189					**0.9844**
Faure ($b = 17$)	0.9844	0.7383	0	0					**0.9844**

Table 3.10: Values of $\beta_{b,k}$ and C_b for different b for point sets in Figure 3.12.

Visually, the Faure sequence looks worse than the Sobol' sequence in Figure 3.12. The Sobol' point set has $t = 2$ and is noticeably better than the point sets from Figure 3.1. We also see that it is c.q.e. in base 3 with $C_3 = 0.9980$, while for the Faure sequence $C_3 = 7.3655$. In base 2, $C_2 = 2.0020$ for Sobol' and $C_2 = 7.2883$ for Faure. In base 17, both constructions have $C_{17} = 0.9844$. In other words, the base 17 equidistribution properties of the two point sets are both good but for base 2 or 3, the Sobol' point set is clearly better. One could argue that the base 2 comparison is not fair, as the Sobol' point set has been constructed in this base while the Faure one has been constructed in base 17, i.e., it is expected that the Sobol' sequence will perform well for $b = 2$. This is why we also included results for $C_3(\boldsymbol{k}; P_n)$, which confirm the superiority of the Sobol' point set for this example.

Next, we use these two-dimensional point sets to integrate the test function g with $c = 0.25$ to indeed show the superiority of the Sobol' point set. Estimated RQMC errors using 25 independent randomizations are displayed in Table 3.11. As expected, the Sobol' sequence outperforms the Faure sequence, and this behaviour is captured by the $C_b(\boldsymbol{k}; P_n)$ values.

Point Set	Base of Scramble	RQMC Error
Sobol'	2	3.61E-11
Faure	2	3.35E-07
Sobol'	3	1.39E-08
Faure	3	4.40E-07
Sobol'	17	3.32E-08
Faure	17	1.42E-07

Table 3.11: Estimated RQMC errors of test function $g(\mathbf{u}) = (1+c(u_1-0.5))(1+c(u_2-0.5))$ with $c = 0.25$ for the point sets in Figure 3.12.

The purpose of this example was to show that $C_b(\boldsymbol{k}; P_n)$ values can be used for comparisons for point sets constructed in different bases for an arbitrary number of points (i.e., n not being an integer power of b), as well as how different bases can illustrate differences in how equidistributed the point set is. Point sets that are "better" should have smaller $C_b(\boldsymbol{k}; P_n)$ values for multiple bases, and not just one. If we only compared these two point sets in base 17, we would have believed that they were of equal quality, despite the Faure sequence being worse both visually and by using $C_b(\boldsymbol{k}; P_n)$ values with other choices of b.

We now continue this line of thought and provide an example where we compare the Sobol' sequence with a generalized Faure sequence for various values of b and n. The generalized Faure sequence in these experiments are constructed using the "Offset" factors, which have been discussed in Section 3.2.3. In all cases, we consider the two-dimensional projections over the last 2 dimensions of the Faure sequence constructed in base b', that is, we consider dimensions $(b'-2, b'-1)$ for this experiment. Figures 3.13, 3.14, and 3.15 show the point sets being used. Visually, these point sets all look well distributed over the unit square, so we cannot make a visual comparison between the point sets as we did in the previous examples in this section. We first calculate the C_b value over this two-dimensional projection in various bases as reported in Tables 3.12, 3.13 and 3.14. Finally, we estimate the integration variance of $g(\mathbf{u}) = (1 + c(u_1 - 0.5))(1 + c(u_2 - 0.5))$ with $c = 0.25$, with scrambling done in the same bases, to determine if the C_b criterion can predict good bases to scramble in, as reported in Table 3.15.

Unsurprisingly, when n is a power of the constructing base and when the scrambling base is the same as the constructing base, the integration variance is the smallest, as seen from the bolded values. However, in the other cases, the C_b criterion is able to differentiate between the point sets. For example, take when the scrambling base is 13 and $n = 53^2$. In these cases, it can be hard to intuitively determine which point set would perform the

best, since the sample size n is not an integer power of the scrambling base. Now, we refer to the C_{13} values. The Sobol' sequence has a C_{13} value greater than 1, which means that this sampling scheme does not have the property of negative dependence. Looking at the integration variances, the Sobol' sequence also performs the worst in this case.

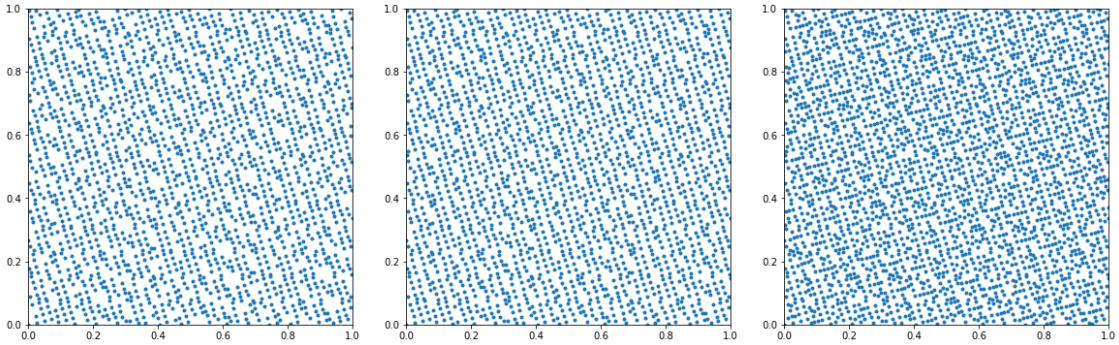


Figure 3.13: First $2^{11} = 2048$ (left), $13^3 = 2197$ (middle), and $53^2 = 2809$ (right) points of Faure sequence over coordinates (11,12) constructed in base 13. The multiplicative (mod 13) factors used for the generating matrices are $[7, 11, 3, 9, 1, 5, 8, 12, 4, 10, 2, 6]$.

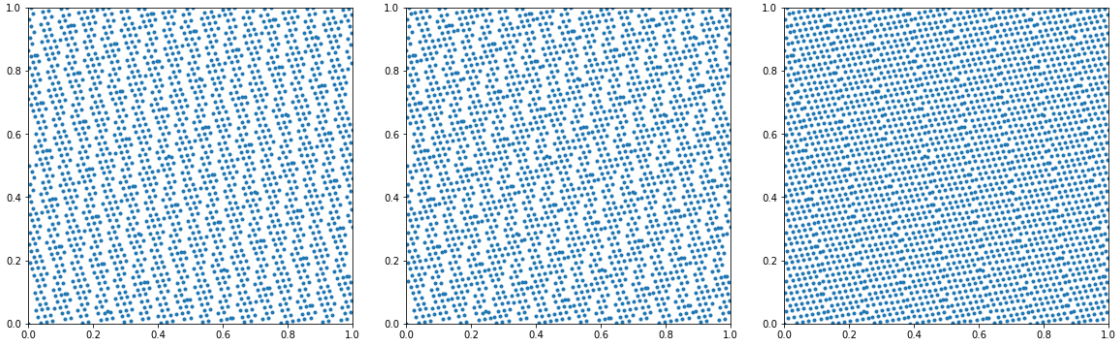


Figure 3.14: First $2^{11} = 2048$ (left), $13^3 = 2197$ (middle), and $53^2 = 2809$ (right) points of Faure sequence over coordinates (51,52) constructed in base 53. The multiplicative (mod 53) factors used for the generating matrices are [27, 43, 11, 35, 3, 19, 51, 31, 47, 15, 39, 7, 23, 29, 45, 13, 37, 5, 21, 1, 33, 49, 17, 41, 9, 25, 28, 44, 12, 36, 4, 20, 52, 32, 48, 16, 40, 8, 24, 30, 46, 14, 38, 6, 22, 2, 34, 50, 18, 42, 10, 26].

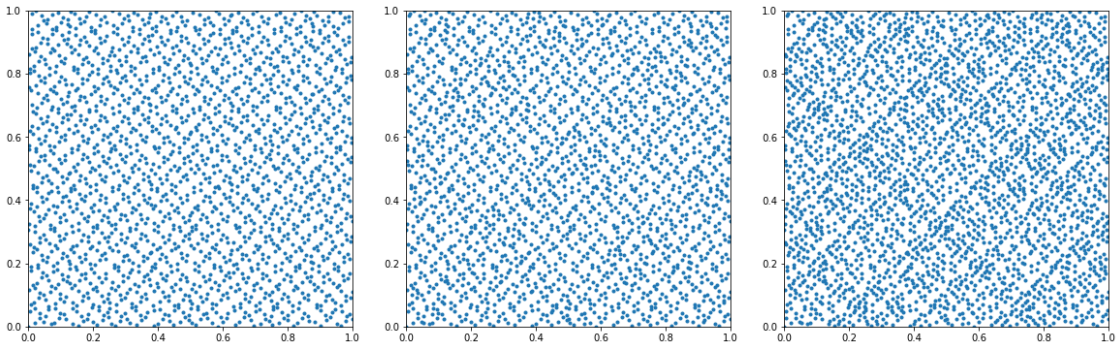


Figure 3.15: First $2^{11} = 2048$ (left), $13^3 = 2197$ (middle), and $53^2 = 2809$ (right) points of Sobol' sequence.

k		1	2	3	4	5	6	
Faure (base 13), Figure 3.13	$n = 2^{11} = 2048$	0.999515	0.998548	0.99663	0.992794	0.985344	0.972185	
Faure (base 53), Figure 3.14	$n = 2^{11} = 2048$	0.999512	0.998723	0.996775	0.993237	0.986581	0.976276	
Sobol', Figure 3.15	$n = 2^{11} = 2048$	0.999511	0.998534	0.99658	0.992672	0.984856	0.969223	
Faure (base 13), Figure 3.13	$n = 13^3 = 2197$	0.999545	0.998639	0.996829	0.993287	0.986316	0.973117	
Faure (base 53), Figure 3.14	$n = 13^3 = 2197$	0.99955	0.998639	0.996835	0.993678	0.987072	0.97577	
Sobol', Figure 3.15	$n = 13^3 = 2197$	0.999545	0.998636	0.996816	0.993267	0.986037	0.976221	
Faure (base 13), Figure 3.13	$n = 53^2 = 2809$	0.999644	0.998933	0.997538	0.995084	0.989802	0.979594	
Faure (base 53), Figure 3.14	$n = 53^2 = 2809$	0.999644	0.998932	0.997508	0.99467	0.989047	0.978377	
Sobol', Figure 3.15	$n = 53^2 = 2809$	0.999644	0.998932	0.997508	0.994666	0.988982	0.985469	
k		7	8	9	10	11	12	
Faure (base 13), Figure 3.13	$n = 2^{11} = 2048$	0.947118	0.908647	0.824133	0.705911	0.601856	0.578407	
Faure (base 53), Figure 3.14	$n = 2^{11} = 2048$	0.950354	1.000122	1.005618	0.846116	0.734734	0.69956	
Sobol', Figure 3.15	$n = 2^{11} = 2048$	0.937958	0.875427	0.750366	1.500733	1.000489	2.000977	
Faure (base 13), Figure 3.13	$n = 13^3 = 2197$	0.947197	0.901829	0.823299	0.706352	0.593437	0.577307	
Faure (base 53), Figure 3.14	$n = 13^3 = 2197$	0.950752	0.962107	0.944066	0.792523	0.77427	0.743707	
Sobol', Figure 3.15	$n = 13^3 = 2197$	0.947144	0.893976	0.792099	1.55703	1.122352	1.738711	
Faure (base 13), Figure 3.13	$n = 53^2 = 2809$	0.959521	0.925118	0.867996	0.775562	0.657423	0.523446	
Faure (base 53), Figure 3.14	$n = 53^2 = 2809$	0.95819	0.916939	0.855662	0.733499	0.821519	0.76959	
Sobol', Figure 3.15	$n = 53^2 = 2809$	0.962734	0.917263	0.907591	1.652645	1.451419	1.322116	
k		13	14	15	16	17	18	C_b
Faure (base 13), Figure 3.13	$n = 2^{11} = 2048$	0.281387	0.343918	0.328285	0.093796	0	0	0.999515
Faure (base 53), Figure 3.14	$n = 2^{11} = 2048$	0.652662	0.257938	0.390816	0.53151	0.375183	0.125061	1.005618
Sobol', Figure 3.15	$n = 2^{11} = 2048$	0	0	0	0	0	0	2.000977
Faure (base 13), Figure 3.13	$n = 13^3 = 2197$	0.264882	0.298841	0.285257	0.081502	0	0	0.999545
Faure (base 53), Figure 3.14	$n = 13^3 = 2197$	0.6656	0.224131	0.339592	0.461845	0.326008	0.108669	0.99955
Sobol', Figure 3.15	$n = 13^3 = 2197$	0.505992	1.011984	2.023969	0	0	0	2.023969
Faure (base 13), Figure 3.13	$n = 53^2 = 2809$	0.54214	0.216025	0.174482	0.066469	0	0	0.999644
Faure (base 53), Figure 3.14	$n = 53^2 = 2809$	0.664693	0.166173	0.207717	0.282495	0.199408	0.066469	0.999644
Sobol', Figure 3.15	$n = 53^2 = 2809$	1.580723	3.161446	6.322892	0	0	0	6.322892

Table 3.12: Values of $\beta_{b,k}$ and C_b for $b = 2$.

k		1	2	3	4	C_b
Faure (base 13), Figure 3.13	$n = 2^{11} = 2048$	0.994148	0.91864			0.994148
Faure (base 53), Figure 3.14	$n = 2^{11} = 2048$	0.994346	0.969594	0.579612	0.149882	0.994346
Sobol', Figure 3.15	$n = 2^{11} = 2048$	0.994148	0.929766	0.689665	0.013626	0.994148
Faure (base 13), Figure 3.13	$n = 13^3 = 2197$	0.994536	0.923497			0.994536
Faure (base 53), Figure 3.14	$n = 13^3 = 2197$	0.994589	0.956775	0.612022	0.130237	0.994589
Sobol', Figure 3.15	$n = 13^3 = 2197$	0.994562	0.93933	0.765938	0.485428	0.994562
Faure (base 13), Figure 3.13	$n = 53^2 = 2809$	0.995728	0.941023	0.340928		0.995728
Faure (base 53), Figure 3.14	$n = 53^2 = 2809$	0.995728	0.941966	0.63729	0.079661	0.995728
Sobol', Figure 3.15	$n = 53^2 = 2809$	0.995738	0.957392	0.867918	1.564258	1.564258

Table 3.13: Values of $\beta_{b,k}$ and C_b for $b = 13$.

k		1	2	C_b
Faure (base 13), Figure 3.13	$n = 2^{11} = 2048$	0.975004	0.336363	1.558525
Faure (base 53), Figure 3.14	$n = 2^{11} = 2048$	0.974751		0.974751
Sobol', Figure 3.15	$n = 2^{11} = 2048$	0.974751	0.506554	0.974751
Faure (base 13), Figure 3.13	$n = 13^3 = 2197$	0.976465	0.320223	1.548713
Faure (base 53), Figure 3.14	$n = 13^3 = 2197$	0.976465		0.976465
Sobol', Figure 3.15	$n = 13^3 = 2197$	0.976707	0.529823	0.976707
Faure (base 13), Figure 3.13	$n = 53^2 = 2809$	0.982073	0.461538	1.126068
Faure (base 53), Figure 3.14	$n = 53^2 = 2809$	0.981481		0.981481
Sobol', Figure 3.15	$n = 53^2 = 2809$	0.981683	0.569801	0.981683

Table 3.14: Values of $\beta_{b,k}$ and C_b for $b = 53$.

n	$n = 2^{11} = 2048$		
scrambling base	2	13	53
Faure (base 13), Figure 3.13	7.91E-10	2.88E-10	4.22E-09
Faure (base 53), Figure 3.14	4.65E-09	3.32E-09	1.43E-09
Sobol', Figure 3.15	**1.15E-13**	3.23E-10	1.84E-09
Monte Carlo	2.03E-07		
n	$n = 13^3 = 2197$		
scrambling base	2	13	53
Faure (base 13), Figure 3.13	1.53E-10	**1.32E-13**	2.00E-09
Faure (base 53), Figure 3.14	1.56E-09	1.76E-09	1.58E-09
Sobol', Figure 3.15	1.35E-10	8.26E-10	2.63E-09
Monte Carlo	1.90E-07		
n	$n = 53^2 = 2809$		
scrambling base	2	13	53
Faure (base 13), Figure 3.13	2.07E-10	7.47E-11	4.73E-09
Faure (base 53), Figure 3.14	4.69E-11	4.14E-11	**3.57E-13**
Sobol', Figure 3.15	7.57E-11	4.49E-10	1.69E-09
Monte Carlo	1.48E-07		

Table 3.15: Estimated variances of the integration function $g(\mathbf{u}) = (1 + c(u_1 - 0.5))(1 + c(u_2 - 0.5))$ with $c = 0.25$ with $m = 25$ randomizations using Owen's nested scramble.

3.3.2 Example in higher dimensions

We now consider the first 17 dimensions of the first 1024 points of the Sobol' and Faure sequences, the latter being constructed in base 17, and the former based on direction numbers provided in [27] for the so-called irreducible Sobol'- Nieddereiter sequences, that is, the same point sets as pictured in Figures 3.1 and 3.12, but over the first 17 dimensions rather than only over coordinates specific two-dimensional projections.

We then compare the two point sets using the test function $g(\mathbf{x}) = \Pi_{j=1}^{s} 1 + c(x_j - 0.5)$ with $c = 0.25$. We also compare the two point sets using a 17-dimensional Asian Call Option with parameters as described in Section 2.5.

Numerical results

First, we calculate the $c(b, \mathcal{K}_{d,w=s,s}, P_n)$ and $\bar{c}(b, \mathcal{K}_{d,w=s,s}, P_n)$ values as defined earlier in this chapter, based on the Sobol' and Faure sequences for $b = 2, 3, 17$. For all the values in Table 3.16, $s = 17$, $d = 2$, and $n = 1024$. For the function g, in 17 dimensions, approximately 99.89% of the variance is explained by the one- and two-dimensional projections. Thus, these criteria, which considers only the one- and two-dimensional projections as $d = 2$, should be a good indicator of how well the point sets can integrate this function.

Point Set	b	$c(b, \mathcal{K}_{d,w=s,s}, P_n)$	$\bar{c}(b, \mathcal{K}_{d,w=s,s}, P_n)$
Sobol'	2	3.50342131	1.70292105
Faure	2	2.86021505	1.89641194
Sobol'	3	2.27979518	1.15282475
Faure	3	3.2777275	2.28363291
Sobol'	17	2.82336915	1.13553456
Faure	17	0.98440937	0.98440937

Table 3.16: Values of criteria based on $C_b(\boldsymbol{k}; P_n)$ for point sets obtained from the Sobol' and Faure sequences.

Here, it is important to look at both the maximum and mean values of this criterion, as it could be the case that the majority of the projections are well-distributed, but there is one bad one. From these results, we can see that the Sobol' sequence seems better distributed than the Faure sequence. Again, one may argue that the comparisons in base 2 and base 17 are unfair, as the Sobol' sequence is constructed in base 2 and the Faure sequence is constructed in base 17, which is why the base 3 comparison is also included. On average, the Sobol' sequence has better distributed two-dimensional projections than the Faure sequence.

Point Set	Base of Scramble	RQMC Error
Sobol'	2	3.68E-08
Faure	2	4.61E-06
Sobol'	3	4.99E-07
Faure	3	5.72E-06
Sobol'	17	1.50E-06
Faure	17	1.22E-06

Table 3.17: Estimated RQMC errors of test function $g(\mathbf{u}) = (1+c(u_1-0.5))(1+c(u_2-0.5))$ with $c = 0.25$ and $n = 1024$ for the point sets in Figure 3.12.

Table 3.17 has the estimated RQMC error for integrating function g with $c = 0.25$. As mentioned, almost all the variance is explained by the one- and two-dimensional projections, so the values from Table 3.16 should indicate which point set performs better. As expected, the Sobol' sequence outperforms the Faure sequence when scrambling in bases 2 and 3, while the performance is similar in base 17.

Point Set	Base of Scramble	K = 45	K = 50	K = 55
Sobol'	2	1.38E-03	8.05E-04	3.17E-04
Faure	2	3.43E-03	2.57E-03	1.90E-03
Sobol'	3	1.08E-03	4.86E-04	2.15E-04
Faure	3	2.23E-03	1.44E-03	1.88E-03
Sobol'	17	7.89E-04	8.21E-04	2.35E-04
Faure	17	1.32E-03	7.73E-04	4.42E-04
Monte Carlo	N/A	2.32E-03	1.57E-03	9.00E-04

Table 3.18: Estimated RQMC errors of the Asian Call Option pricing problem for the point sets in Figure 3.12.

Table 3.18 shows the estimated RQMC variance for the Asian Call Option pricing problem with parameters as described in Section 2.5 on page 24, for out-of-the-money, at-the-money, and in-the-money options. Unlike the previous experiment, less of the variance is explained by one- and two-dimensional projections. As before, in base 2 and 3, the Sobol' sequence outperforms the Faure sequence. In base 17, the performance is more similar between the two sequences, but the Sobol' sequence still slightly outperforms the Faure sequence.

These two experiments suggest that the values of the criteria based on $C_2(\boldsymbol{k}; P_n)$ as shown in Table 3.16 are able to predict how well the point sets are able to perform on integration problems.

3.4 Conclusion and future work

In this chapter, working with the Sobol' and Faure sequences, we showed that the C_b criterion can be used as a way to measure the quality of a point set and tell us about its performance when used as a scrambled RQMC estimator.

We also evaluated two choices for multiplicative factors used to generalize the Faure sequence based on the ones from [25]. We evaluated their performance using a numerical study as well as studying the $C_b(\boldsymbol{k}, P_n)$ values. While these factors can break up some poor projection properties of these point sets, they do not generally outperform using scrambling to randomize the point set unless n is a power of b.

Although we have shown through the examples in this chapter that the C_b criterion has many uses, a major drawback of calculating the $C_b = \sup_{\boldsymbol{k}} C_b(\boldsymbol{k}; P_n) = \sup_{\boldsymbol{k}} \frac{b^{|\boldsymbol{k}|} M_b(\boldsymbol{k}; P_n)}{n(n-1)}$ values for a generic point set, is that the time and memory usage becomes extremely high as n and s increases.

For the calculation of $M_b(\boldsymbol{k}; P_n)$, which is the number of ordered pairs $(\mathbf{x}, \mathbf{y})$ in P_n such that $\boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y}) \geq \boldsymbol{k}$, we need to first calculate the $\boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y})$ values for every pair of $(\mathbf{x}, \mathbf{y})$ in P_n, and then. It takes $O(n^2 s)$ time to calculate $\boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y})$ for every pair of $(\mathbf{x}, \mathbf{y})$ in P_n as there are $\frac{n^2}{2}$ pairs and γ has to be calculated separately for each coordinate. The time complexity also increases as $\max_{(\mathbf{x},\mathbf{y})} \boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y})$ increases because that means there are more vectors $\boldsymbol{k}$ such that $k \leq \max \gamma$ to check to find the max. For every $k = |\boldsymbol{k}|$, the number of possible $\boldsymbol{k}$ is $\binom{k+s-1}{k}$, which obviously increases with k and s. Thus, we have that the time complexity of calculating C_b is $O(n^2 s + n^2 s \sum_{k=1}^{\max \gamma} \binom{k+s-1}{k})$.

Another issue is the potential memory requirement, especially as s increases. As the time complexity is quite high, to decrease the runtime, counting the number of pairs $(\mathbf{x}, \mathbf{y})$ in P_n where $\boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y}) \geq \boldsymbol{k}$ is done in parallel for all pairs and all $\boldsymbol{k}$ where $|\boldsymbol{k}| = k$. This means that the memory requirement is $O\left(n^2 s \binom{k+s-1}{k}\right)$ for each k. Since this increases dramatically with s, it becomes unfeasible for point sets in many coordinates.

For specific types of low-discrepancy sequences, such as the (generalized) Halton sequence, as we will see in Chapter 4, we can calculate the C_b values theoretically. However, since we are proposing to use these values in practice to measure the quality of different

point sets, sometimes we must use calculate C_b using the point set P_n without relying on how it was constructed.

These drawbacks regarding computational requirements are why in Section 3.2.2, we restricted $\boldsymbol{k}$ to only consider two-dimensional projections, as even for a relatively small 17-dimensional point set, calculating $C_b(\boldsymbol{k}; P_n)$ for unrestricted $\boldsymbol{k}$ would have been unfeasible.

To have the $C_b(\boldsymbol{k}; P_n)$ values be more usable in practice, the next step to improve this criterion would be to find a way to simplify the calculation. This is especially important if the $C_b(\boldsymbol{k}; P_n)$ values are used to search for good constructions of the point set, as in Chapter 4 when we will search for Halton sequence permutations, or to search for other good constructions, for example, direction numbers for the Sobol' sequence. Since we need to run the calculations many times (possibly hundreds of times) to find the best permutation, it is crucial to reduce the time and memory requirements.

Chapter 4

Dependence properties of scrambled Halton sequences

The dependence framework developed in [48, 90] provides tools to study $\sigma_{I,J}$ in the case of scrambled digital nets. Here, we extend these results to scrambled Halton sequences, at the same time introducing the idea of using the random linear scrambling of Matoušek [52] and adapting it to a multi-base version applicable to Halton sequences. Sections 2.3.2 and 4.3 present these results. We propose to use the negative dependence framework developed in [90] to contribute further to the improvement of Halton sequences in the following two ways: first, we show that scrambling Halton sequences using a multi-base version of either Owen's nested permutations or the random linear scrambling approach of Matoušek endows Halton sequences with the same properties as those derived in [90] for scrambled $(0, m, s)$-nets. Namely, such scrambled Halton sequences have the property of being negatively upper orthant dependent, and thus they can be shown to have a variance no larger than the Monte Carlo method for bounded quasi-monotone functions. In two dimensions, this advantage can be shown to hold for any function that is monotone in each coordinate. Second, the negative dependence framework allows one to define new quality criteria that can be described as a generalization of the quality parameter t often used to judge the quality of digital nets. Compared to other discrepancy-based measures, these new quality criteria have the advantage of being meaningful even for small point sets. In addition, they can be used to assess the quality of deterministically chosen permutations meant to improve the quality of Halton sequences, and to compare such sequences to digital net constructions.

As mentioned before, several randomizations have been proposed for Halton sequences, and while the idea of using Owen's nested permutations has been mentioned by other au-

thors (e.g., [88]), as far as we know it has not been used or studied much. Furthermore, the idea of using the random linear scrambling of Matoušek adapted to Halton sequences has not appeared elsewhere, to the best of our knowledge. Here we not only derive important properties for this randomization, but also demonstrate its usefulness in practice. This is our first main contribution. Our second one is to propose a quality measure for Halton sequences that overcomes the difficulty of having to deal with multiple bases in their construction and that allows for direct comparison with other constructions.

Except for Section 4.6, the majority of the contents of this chapter have been published in [22]. Results establishing the negative dependence properties of the digitally-scrambled Halton sequence are presented in Section 4.3. Criteria that can be used to evaluate both the quality of Halton sequences and digital net constructions and that are based on negative dependence properties are discussed in Section 4.4. Section 4.5 includes numerical results comparing a new generalized Halton sequence constructed to optimize a specific criterion from the family introduced in Section 4.4 against other generalized Halton sequences and their randomizations, including our proposal to use a multi-base random linear scrambling. This section also includes numerical results on financial applications that were not part of [22]. Section 4.6 extends our work on assessing different forms of scrambling and is not part of [22]. This work in this section has not been published yet and is the topic of the working paper [20], which is done in collaboration with Christiane Lemieux and Henri Faure. The setup of the experiments in this section is similar to the ones from Section 3.2.3, and here we assess permutations for the Halton sequence based on the ones introduced in [25]. A summary of our work is given in Section 4.7.

4.1 The Halton sequence

The construction of the Halton sequence and the different randomizations and generalizations proposed to improve it were discussed in Section 2.2.1. Digital scramblings for the Halton sequence were discussed in Section 2.3.2. We now discuss some modifications that are specific to the Halton sequence.

4.1.1 Randomizations for the Halton sequence

In addition to the use of generalized Halton sequences as a way to improve upon the original Halton sequence, one can also *randomize* it. By this we mean one can apply certain random transformations to these sequences and this is typically done for the following two reasons:

first, it allows for easier error estimation (by making use of the replications and the central limit theorem, as we will see in Section 4.5) than when using purely deterministic sequences, and second, it can improve the quality of these sequences.

Some of the randomizations that have been proposed for Halton sequences are adaptations of ideas that were originally proposed for digital nets, such as the ones mentioned in Section 2.3.

For example, a simple way to randomize a digital net is to use a digital shift, which extends to van der Corput based sequences the random shift method of Cranley and Patterson [17]. For the Halton sequence, rather than using the same base for all dimensions, the digital shift in dimension j is done in base b_j, for $j = 1, \ldots, s$. Sequences randomized this way are referred to as "randomly digitally shifted Halton sequences". Due to how the digital shift is defined, it is not sufficient, by itself, to break up the correlation between dimensions of the Halton sequence. Thus, it is generally used as a randomization method for generalized Halton sequences.

It is also quite natural for the Halton sequences to use randomizations based on randomly chosen permutations $\sigma_{j,r}$ to define the generalized Halton sequence from (2.5). As there are different ways to select these permutations, here we use the terminology developed in [52]. Although this terminology was defined for digital nets, the use of the van der Corput sequence as a building block both for digital nets and Halton sequences coupled with the fact that randomizations are typically applied independently across the coordinates 1 to s implies we can use this terminology to describe randomizations for the Halton sequence as well. So we refer to the case where the permutations $\sigma_{j,r}$ are chosen independently and with replacement among all permutations of $\{0, 1, \ldots, b_j - 1\}$ for all j, r as a *random digit-scrambling*. The case where the $\sigma_{j,r}$ are chosen among linear permutations of the form $\sigma_{j,r}(a) = h_{j,r}a + g_{j,r}$ with $h_{j,r} \in \{1, \ldots, b_j - 1\}$ and $g_{j,r} \in \{0, \ldots, b_j - 1\}$ is referred to as a *random linear digit-scrambling*.

These two forms of scrambling are described in [52] as alternatives to nested uniform scrambling [62], which cannot be described by a given choice of random permutations in (2.5). Instead, it makes use of permutations for the digits $a_r(n)$ that depend on the previous digits $a_0(n), \ldots, a_{r-1}(n)$, hence the "nested" term to define it.

We can now review randomizations that have been proposed for the Halton sequence. In [63], Owen proposes to randomize the Halton sequence using random digit scrambling, resulting in a "randomly digitally scrambled Halton sequence". This construction is compared to the randomly shifted generalized Halton sequence provided in the `qrng` package, which is based on the factors from [26]. Numerical results on different test functions suggest that the accuracy of integrating with Owen's implementation is slightly better than

the one from the `qrng` package. An advantage of Owen's implementation is that it uses the Neumann-Kakutani transformation to extend a point set to $n' > n$ points or $s' > s$ dimensions by only computing the necessary $n' - n$ points or $s' - s$ coordinates.

Wang and Hickernell proposed the "random start Halton sequence" in [88], which randomizes the starting point for each coordinate. This randomization is not computationally more expensive than the original Halton sequence, as it takes advantage of the Neumann-Kakutani transformation to "skip" terms. In [88], the performance of the random start Halton sequence is compared with the randomly digitally shifted Halton sequence and the randomly digitally scrambled Halton sequence. Their numerical results with a test function show that as either n or s increase, the estimated variance using random start was far smaller than when using digital shift, and was similar to digital scrambling while being much faster to generate.

This randomization can also be combined with other randomizations or modifications to the Halton sequence. Indeed, in [58], Ökten used the random start randomization combined with deterministic scrambling based on the permutations from [25] to obtain the "random start scrambled Halton sequence". Numerical results in [58] suggest the random-start scrambled Halton sequence can improve the variance by factors of up to 7000 over the random-start Halton sequence.

4.1.2 Deterministic improvements to the Halton sequence

Faure and Lemieux proposed deterministic permutations in [26] to generalize the Halton sequence, which are the permutations used as a baseline comparison in Section 4.5. Their proposed permutations are of the form $\sigma(i) = (f_j i) \mod b_j$ for $j = 1, \ldots, s$, and they provide the factors f_j for $1 \leq j \leq 360$. The factors were found based on a search where the L_2-discrepancy was minimized over one and two-dimensional projections.

Other proposed permutations include those from Chi et al. in [13] in the form of deterministic linear digit scramblings selected using a criterion based on the serial test for two-dimensional sequences.

Another set of permutations was defined in [18], which only have the restriction of having $\sigma(0) = 0$ and were found using an evolutionary criterion to optimize discrepancy. The `ghalton` package in Python provides an implementation of the generalized Halton sequence using these permutations.

There are other deterministic modifications that can be made to the Halton sequence that improve performance for numerical integration in high dimensions. Kocis and Whiten

introduced the "Halton sequence leaped" in [45], which uses every l^{th} point of the sequence where $l \in \mathbb{N}$ is co-prime with b_j for $j = 1, \ldots, s$. This leaping strategy was also proposed for other low-discrepancy sequences, such as the Faure [24] and Sobol' [75] sequences, but the results for the Halton sequence outperform those for other sequences. C. J. Price and C. P. Price propose a way in [71] to reuse the prime bases used to construct each coordinate, with modifications to the radix inverse function of (2.2) to ensure that elements that share a base will not have identical values. Each base b_j can be used k times if b_j^{k-1} is not larger than the second-largest prime used within the point set. This way, smaller bases are used to construct the point set and the linear relationship between coordinates is improved, and fewer points are required to get uniform coverage within the unit cube.

This review confirmed that the most common way to assess the quality of generalized or randomized Halton sequences is via the use of different test-functions. This is largely due to the fact that there is no natural quality measure for the Halton sequence when considering point sets of a finite size. In Section 4.4 we propose a remedy to this shortcoming.

4.2 Digital scrambling of Halton sequences

In this section, we generalize some definitions from Chapter 3, so they can be applied to the Halton sequence, where each coordinate is constructed in a different base b_j.

For the Halton sequence, the concept of equidistribution must be adjusted to intervals of the form

$$\prod_{j=1}^{s} \left[\frac{d_j}{b_j^{k_j}}, \frac{d_j + 1}{b_j^{k_j}} \right)$$

with $0 \leq d_j < b_j^{k_j}$. The first $b_1^{p_1} \ldots b_s^{p_s}$ points of the Halton sequence can be shown to be $\boldsymbol{k}-$equidistributed for any $\boldsymbol{k}$ with $k_j \leq p_j$ for $j = 1, \ldots, s$. Here we have a similar property as for the case $t = 0$ with digital nets in base b, in that when n is a product of powers of the bases $b_1, \ldots, b_s$, we can partition $[0, 1)^s$ into boxes up until we have as many boxes as points, and from there on we will have exactly one point per box.

This gives us the intuition that the multibase version of $M_b(\boldsymbol{k}; P_n)$ and $C_b(\boldsymbol{k}; P_n)$ adjusted to Halton sequences will have similar properties as nets with $t = 0$: this is what we set out to demonstrate in Section 4.3.

Let $\mathbf{b} = (b_1, \ldots, b_s)$ and recall that, typically, b_j is the j^{th} smallest prime number.

Definition 4.2.1. For $\mathbf{x}, \mathbf{y} \in [0, 1)^s$, $\boldsymbol{\gamma}_{\mathbf{b}}^s(\mathbf{x}, \mathbf{y}) = (\gamma_{b_1}(x_1, y_1), \ldots, \gamma_{b_s}(x_s, y_s))$.

Next, we prove that $\boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{x}, \mathbf{y})$ is invariant under the type of permutations used to define generalized Halton sequences. As seen in Theorem 2.4.8, the $C_b(\boldsymbol{k}; P_n)$ values introduced in [90] are crucial to determine if a point set to which a $b-$digital scramble is applied is NLOD. In the next section, we prove a version of this result for Halton sequences. In order for us to do so, we need to introduce a multi-base version of the counting numbers $N_b(\boldsymbol{i}; P_n)$ and $M_b(\boldsymbol{k}; P_n)$, as well as the $C_b(\boldsymbol{k}; P_n)$ values, to be used with the Halton sequence.

Definition 4.2.2. Let $P_n = \{\mathbf{U}_1, \ldots, \mathbf{U}_n\}$ be a point set in $[0,1)^s$ and $\mathbf{b}, \boldsymbol{i}, \boldsymbol{k} \in \mathbb{N}^s$, with $b_j \geq 2$ for all $1 \leq j \leq s$. Then

1. $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$ is the number of ordered pairs of distinct points $(\mathbf{U}_l, \mathbf{U}_j)$ in P_n such that $\boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{U}_l, \mathbf{U}_j) = \boldsymbol{i}$,
2. $M_{\mathbf{b}}(\boldsymbol{k}; P_n)$ is the number of ordered pairs of distinct points $(\mathbf{U}_l, \mathbf{U}_j)$ in P_n such that $\boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{U}_l, \mathbf{U}_j) \geq \boldsymbol{k}$, and
3. $C_{\mathbf{b}}(\boldsymbol{k}, P_n) = \frac{\left(\prod_{j=1}^s b_j^{k_j}\right) M_{\mathbf{b}}(\boldsymbol{k}; P_n)}{n(n-1)}$.

We also need to extend our definition of base $b-$digital scramble to a multi-base version, as follows:

Definition 4.2.3. A randomization $\mathcal{S}$ is called a *base* $\mathbf{b}$*-digital scrambling* if it can be written as $\mathcal{S}(\mathbf{u}_i) = (\mathcal{S}_1(u_{i,1}), \ldots, \mathcal{S}_s(u_{i,s}))$ where each $\mathcal{S}_j$ is a base b_j-digital scrambling as defined in Definition 2.4.4 and the $\mathcal{S}_j$ are independent randomizations.

Claim 1. Let $P_n = \{\mathbf{U}_1, \ldots, \mathbf{U}_n\}$ be a point set in $[0,1)^s$, ${}_{\mathbf{b}}\tilde{P}_n$ be obtained after applying a base $\mathbf{b}-$digital scramble to P_n, and $\mathbf{b}, \boldsymbol{i}, \boldsymbol{k} \in \mathbb{N}^s, b_j \geq 2$.

Then, $N_{\mathbf{b}}(\boldsymbol{i}; P_n) = N_{\mathbf{b}}(\boldsymbol{i}; {}_{\mathbf{b}}\tilde{P}_n)$, $M_{\mathbf{b}}(\boldsymbol{k}; P_n) = M_{\mathbf{b}}(\boldsymbol{k}; {}_{\mathbf{b}}\tilde{P}_n)$, and $C_{\mathbf{b}}(\boldsymbol{k}, P_n) = C_{\mathbf{b}}(\boldsymbol{k}, {}_{\mathbf{b}}\tilde{P}_n)$.

Proof. By definition, a base $\mathbf{b}-$digital scramble preserves $\boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{U}_l, \mathbf{U}_j)$ for all pairs of points $(\mathbf{U}_l, \mathbf{U}_j)$ in P_n. Thus, the counting numbers $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$ and $M_{\mathbf{b}}(\boldsymbol{k}; P_n)$, which only depend on $\boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{U}_l, \mathbf{U}_j)$, and thus $C_{\mathbf{b}}(\boldsymbol{k}, P_n)$, do not change after P_n is scrambled. □

The random linear scrambling method can be adapted to the multi-base setting of the Halton sequence by simply juxtaposing randomly linearly scrambled van der Corput sequences in base b_j, as given in (2.10), for $j = 1, \ldots, s$.

Claim 2. The random linear scrambling randomization of the Halton sequence is a base $\mathbf{b}-$digital scrambling.

Proof. We need to show that the conditions in Definition 4.2.3 are satisfied. Since the nonsingular lower triangular matrices R_j are independently generated, the randomizations for each coordinate are independent. Now we check that, for coordinate j, the properties in Definition 2.4.4 are satisfied. Conditions 1, 2(a), and 2(b) are satisfied as per [52] so we only need to show condition 2(c) is satisfied for each coordinate j.

Given a point $\mathbf{u}$ from the Halton sequence, the randomized version $\mathbf{U}$ has elements $U_j = R_j(u_{j,0}, u_{j,1}, \ldots)^T + (g_{j,0}, g_{j,1}, \ldots)^T$, where the $u_{j,\ell}$ for $\ell \geq 0$ are the digits from the base b_j expansion of u_j and operations are done in $\mathbb{Z}_{b_j}$. If we write $U_j = \sum_{k\geq 0} U_{j,k} b_j^{-k-1}$, then $U_{j,k} = \sum_{l=1}^{k} R_{j,l,k} u_{j,l} + g_{j,k} = x u_{j,k} + y$ for each $k \geq 0$, where the operations are done modulo b, $x, y \in Z_b$, and y depends on $u_{j,l}$ for $l < k$. If two points u_j and v_j have r digits in common in their base b expansion, then for $k > r+1$, $U_{j,k} = x_1 u_{j,k} + y_1$ and $V_{j,k} = x_2 v_{j,k} + y_2$, where x_1, x_2, y_1, y_2 are uniformly and independently distributed over Z_b, and y_1 depends on $u_{j,l}$ and y_2 depends on $v_{j,l}$ for $l < k$. Thus, $U_{j,k}$ and $V_{j,k}$ are mutually independent and uniformly distributed over $\{(k_1, k_2), 0 \leq k_1, k_2 < b\}$. □

As far as we know, random linear scrambling has not been used to randomize the Halton sequence, and one of our contributions is a theoretical justification for this randomization method.

4.3 Dependence properties of scrambled Halton point sets

In what follows, we assume P_n consists of any n consecutive points from a (generalized) Halton sequence and refer to it as a *Halton point set*. In addition, we refer to the set of points ${}_{\mathbf{b}}\tilde{P}_n$ obtained after applying a base $\mathbf{b}-$digital scramble to P_n as a *scrambled Halton point set*.

The goal of this section is to prove that a scrambled Halton point set is an NLOD sampling scheme. This in turn guarantees a variance reduction relative to MC when integrating bounded quasi-monotone functions.

A key quantity to analyze whether ${}_{\mathbf{b}}\tilde{P}_n$ is an NLOD sampling scheme is the joint pdf $\psi(\cdot,\cdot)$ of two distinct points randomly chosen from ${}_{\mathbf{b}}\tilde{P}_n$. From [90], for a digital net randomized with a base b-digital scramble, the corresponding joint pdf is

$$\psi(\mathbf{x}, \mathbf{y}) = \begin{cases} \frac{N_b(\boldsymbol{i}; P_n)}{n(n-1)} \frac{b^{s+|\boldsymbol{i}|}}{(b-1)^s}, & \text{if } |\boldsymbol{i}| < \infty, \\ 0, & \text{if } |\boldsymbol{i}| = \infty, \end{cases} \tag{4.1}$$

where $\boldsymbol{i} = \boldsymbol{\gamma}_b^s(\mathbf{x}, \mathbf{y})$ and $|\boldsymbol{i}| = \gamma_b(\mathbf{x}, \mathbf{y})$.

As shown in the following proposition, it is fairly straightforward to derive an analog of (4.1) for the scrambled Halton sequence, since (4.1) applies to a one-dimensional scrambled Halton sequence.

Proposition 4.3.1. *Let ${}_{\mathbf{b}}\tilde{P}_n$ be a scrambled Halton point set. Let $\mathbf{x}, \mathbf{y} \in [0,1]^s$ and $\boldsymbol{i} = \boldsymbol{\gamma}_{\mathbf{b}}^s(\mathbf{x}, \mathbf{y})$. The joint pdf of two randomly chosen distinct points from ${}_{\mathbf{b}}\tilde{P}_n$ is*

$$\psi(\mathbf{x}, \mathbf{y}) = \begin{cases} \frac{N_{\mathbf{b}}(\boldsymbol{i}; P_n)}{n(n-1)} \Big/ \prod_{j=1}^s \frac{b_j - 1}{b_j^{1+i_j}}, & \text{if } |\boldsymbol{i}| < \infty, \\ 0, & \text{if } |\boldsymbol{i}| = \infty. \end{cases} \tag{4.2}$$

Proof. To prove this result, we make use of the following definitions: for each $\boldsymbol{k}, \boldsymbol{i} \in (\mathbb{N} \cup \{\infty\})^s$, and bases $b > 2$ and $\mathbf{b} \in \mathbb{N}^s$, $b_j > 2$ for all j, define:

$${}_{\mathbf{b}}C_{\boldsymbol{k}}^s = \{(\mathbf{x}, \mathbf{y}) \in [0,1)^{2s} : \boldsymbol{k} \le \boldsymbol{\gamma}_{\mathbf{b}}^s(\mathbf{x}, \mathbf{y})\} \text{ and } {}_{\mathbf{b}}D_{\boldsymbol{i}}^s = \{(\mathbf{x}, \mathbf{y}) \in [0,1)^{2s} : \boldsymbol{\gamma}_{\mathbf{b}}^s(\mathbf{x}, \mathbf{y}) = \boldsymbol{i}\}.$$

When $s = 1$, we drop the superscript 1 and write ${}_bD_i = {}_bD_i^1$. Then, $\mathrm{Vol}({}_{\mathbf{b}}D_{\boldsymbol{i}}^s) = \prod_{j=1}^s \frac{b_j - 1}{b_j^{1+i_j}}$ since $\mathrm{Vol}({}_bD_i) = (b-1)/b^{i+1}$.

The notation $\boldsymbol{\gamma}_{\mathbf{b}}^s(\mathbf{x}, \mathbf{y})$ together with ${}_{\mathbf{b}}D_{\boldsymbol{i}}^s$ give us a different way of describing the properties of a base $\mathbf{b}-$digital scramble from Definition 4.2.3. Namely, for any two scrambled points $\mathbf{U}_j, \mathbf{U}_l$ obtained from a base $\mathbf{b}-$digital scrambling of $\mathbf{V}_j, \mathbf{V}_l$, it holds that $(\mathbf{U}_j, \mathbf{U}_l) \sim U({}_{\mathbf{b}}D_{\boldsymbol{i}}^s)$, where $\boldsymbol{i} = \boldsymbol{\gamma}_b^s(\mathbf{V}_j, \mathbf{V}_l)$. This then implies the joint pdf $\psi(\mathbf{x}, \mathbf{y})$ is constant over each ${}_{\mathbf{b}}D_{\boldsymbol{i}}^s$ region.

It also means the integral of $\psi(\mathbf{x}, \mathbf{y})$ over ${}_{\mathbf{b}}D_{\boldsymbol{i}}^s$ is equal to the probability that a randomly chosen pair of distinct points from ${}_{\mathbf{b}}\tilde{P}_n$ lies in ${}_{\mathbf{b}}D_{\boldsymbol{i}}^s$. Hence,

$$\frac{N_{\mathbf{b}}(\boldsymbol{i}; {}_{\mathbf{b}}\tilde{P}_n)}{n(n-1)} = \int_{{}_{\mathbf{b}}D_{\boldsymbol{i}}^s} \psi(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} = \psi_{\boldsymbol{i}} \mathrm{Vol}({}_{\mathbf{b}}D_{\boldsymbol{i}}^s) = \psi_{\boldsymbol{i}} \prod_{j=1}^s \frac{b_j - 1}{b_j^{1+i_j}}.$$

For the $|\boldsymbol{i}| = \infty$ case, our assumptions on P_n are that the one-dimensional projections have n distinct points, so there cannot be two distinct points in ${}_bD_\infty$, and thus the joint pdf must be 0. □

As it turns out, the quantity $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$ used in the definition (4.2) of the joint pdf can be computed exactly when P_n is a Halton point set. Similarly, we can derive a formula for $M_{\mathbf{b}}(\boldsymbol{k}; P_n)$, which will be useful later on to prove our main result.

Both formulas rely on the following lemma, which builds on the fact that the number of shared common digits between two points of a van der Corput sequence only depends on how many terms are in between them. That is, $\gamma_b(x_i, x_j) = \gamma_b(x_{i+k}, x_{j+k})$ for all $k \in \mathbb{Z}$ and x_l denotes the l^{th} term in the sequence.

Lemma 4.3.2. *Assume P_n consists of n consecutive points from a (generalized) van der Corput sequence S_b^Σ in a prime base b. Then*

$$\gamma_b(x, y) = \log_b(\gcd(d, b^m)), \quad x, y \in P_n, \tag{4.3}$$

where $m = \lfloor \log_b(n) \rfloor$, and d is the "distance" between x and y in the sequence: if $x = S_b^\Sigma(i)$ and $y = S_b^\Sigma(j)$, then $d = |i - j|$.

Proof. Each combination of the terminal (least significant) k digits of the base b expansion for integers repeats every b^k integers. Thus, by construction of the van der Corput sequence, each combination of k initial (most significant) digits repeats every b^k terms of P_n: if $b^k | d$, there are at least k common initial digits between x and y since there are at least k common terminal digits between i and j in their base b expansions. If we want exactly k digits to be common (and not $(k+1)$ digits), we would then need to exclude those d with $b^{k+1} | d$. Thus, the number of common digits between two points is the largest integer l such that $b^l | d$ (or, equivalently, $d \bmod b^l = 0$). Note that the largest power of b that could divide any d is b^m, since $n < b^{m+1}$. Then, as b is prime by assumption, the only possible divisors are integer powers of b, and $\gcd(d, b^m)$ gives us exactly the largest b^l that divides d. Taking the logarithm shows the result. □

Proposition 4.3.3. *If P_n is a Halton point set, then*

$$N_{\mathbf{b}}(\boldsymbol{i}; P_n) = 2 \sum_{d=1}^{n-1} (n - d) \times I((\sum_{j=1}^{s} d \bmod b_j^{i_j}) = 0) \times I((\prod_{j=1}^{s} d \bmod b_j^{i_j+1}) \neq 0), \tag{4.4}$$

where $I()$ is the indicator function.

Proof. As in the proof for Lemma 4.3.2, for each dimension j, the number of initial common digits between two points is the largest integer l such that $d \bmod b_j^l = 0$, where d is the distance between the two points as defined in Lemma 4.3.2. To find a formula for $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$, we need to find out how many pairs of points have i_j initial digits in common, for $j = 1, \ldots, s$ simultaneously. For each $d \in \{1, \ldots, n-1\}$, there are $2(n-d)$ pairs of points within P_n with distance d. Then, if i_j is the largest integer for each coordinate $j = 1, \ldots s$, such that $d \bmod b_j^{i_j} = 0$, we count it towards $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$. □

In the one-dimensional case, we can simplify $N_b(i; P_n)$ further to not need the indicator functions.

Theorem 4.3.4. *For a van der Corput sequence in base b with a base b−digital scramble, we have*

$$\begin{aligned} N_b(i; P_n) &= b^{i+1}(b-1)d_1\left(d_1 - 1\right) + 2d_1\left(n - d_1 b^{i+1}\right)(b-1) \\ &\quad + b^i(sb(b-1) + r(r+1)) + 2\left(n - d_2 b^i\right)(d_2 \bmod b) \text{ and} \end{aligned} \tag{4.5}$$

where $d_1 = \lfloor (n-1)/b^{i+1} \rfloor$ *and* $d_2 = \lfloor (n-1)/b^i \rfloor$.

Proof. $N_b(i; P_n)$ can be written as $N_b(i; P_n) = 2\sum_{d=1}^{n-1} N_{(d)}(i)$, where $N_{(d)}(i)$ is the number of k that satisfy $\gamma(x_{n-d}, x_{n-k}) = i$, for $k < d$. When considering the points that occur after each x_j in P_n, there are $b-1$ points that occur every b^{i+1} points that satisfy this, specifically at $x_{j+b^i}, x_{j+2b^i}, \ldots x_{j+(b-1)b^i}$ (and similarly for the next group of b^{i+1} points). This means that

$$N_{(d)}(i) = \lfloor d/b^{i+1} \rfloor (b-1) + \lfloor d \bmod b^{i+1}/b^i \rfloor,$$

and

$$N_b(i; P_n) = 2\sum_{d=1}^{n-1} \lfloor d/b^{i+1} \rfloor (b-1) + \lfloor d \bmod b^{i+1}/b^i \rfloor. \tag{4.6}$$

Now, the goal is to simplify this so that it can be computed in $O(1)$ time.

First, note that $\lfloor d \bmod b^{i+1}/b^i \rfloor = \lfloor d/b^i \rfloor \bmod b$.

Let

$$\begin{aligned} N_b(i; P_n) &= 2\sum_{d=1}^{n-1} \lfloor d/b^{i+1} \rfloor (b-1) + \lfloor d/b^i \rfloor \bmod b \\ &= 2(N_1 + N_2), \end{aligned}$$

where $N_1 = \sum_{d=1}^{n-1} \lfloor d/b^{i+1} \rfloor (b-1)$ and $N_2 = \sum_{d=1}^{n-1} \lfloor d/b^i \rfloor \bmod b$. For $k \in \mathbb{Z}$: if $d \in [kb^{i+1}, (k+1)b^{i+1} - 1]$, then $\lfloor d/b^{i+1} \rfloor = k$, and if $d \in [kb^i, (k+1)b^i - 1]$, then $\lfloor d/b^i \rfloor = k$.

Let $d_1 = \lfloor (n-1)/b^{i+1} \rfloor$. Then, we can write:

$$
\begin{aligned}
N_1 &= \left(\sum_{k=0}^{d_1-1}\left(\sum_{d=kb^{i+1}}^{(k+1)b^{i+1}-1}\left\lfloor\frac{d}{b^{i+1}}\right\rfloor(b-1)\right)\right)+\left(\sum_{d=d_1b^{i+1}}^{n-1}\left\lfloor\frac{d}{b^{i+1}}\right\rfloor(b-1)\right)\\
&= \left(\sum_{k=0}^{d_1-1}\left(\sum_{d=kb^{i+1}}^{(k+1)b^{i+1}-1}k(b-1)\right)+\left(\sum_{d=d_1b^{i+1}}^{n-1}d_1(b-1)\right)\right.\\
&= \left(\sum_{k=0}^{d_1-1}kb^{i+1}(b-1)\right)+d_1\left(n-d_1b^{i+1}\right)(b-1)\\
&= b^{i+1}(b-1)\left(\sum_{k=0}^{d_1-1}k\right)+d_1\left(n-d_1b^{i+1}\right)(b-1)\\
&= b^{i+1}(b-1)\frac{d_1\left(d_1-1\right)}{2}+d_1\left(n-d_1b^{i+1}\right)(b-1).
\end{aligned}
$$

Likewise for N_2, let $d_2 = \lfloor (n-1)/b^i \rfloor$. Then

$$
\begin{aligned}
N_2 &= \left(\sum_{k=0}^{d_2-1}\left(\sum_{d=kb^{i}}^{(k+1)b^{i}-1}\left\lfloor\frac{d}{b^{i}}\right\rfloor \bmod b\right)\right)+\left(\sum_{d=d_2b^{i}}^{n-1}\left\lfloor\frac{d}{b^{i}}\right\rfloor \bmod b\right)\\
&= \left(\sum_{k=0}^{d_2-1}\left(\sum_{d=kb^{i}}^{(k+1)b^{i}-1}k \bmod b\right)\right)+\left(\sum_{d=d_2b^{i}}^{n-1}d_2 \bmod b\right)\\
&= \left(\sum_{k=0}^{d_2-1}b^i(k \bmod b)\right)+\left(n-d_2b^i\right)(d_2 \bmod b)\,.
\end{aligned}
$$

For the remaining sum, let $d_2-1 = sb+r$ where $s = \lfloor (d_2-1)/b \rfloor$ and $r = (d_2-1) \bmod b$. Since between $k=0$ and $k = sb-1$ there are s occurrences of $k \bmod b = m$, for $m \in \{0,1,\ldots,b-1\}$, I can write:

$$\begin{aligned}
\sum_{k=0}^{d_2-1} b^i(k \bmod b) &= b^i \sum_{k=0}^{sb+r}(k \bmod b) \\
&= \left(b^i \sum_{k=0}^{sb-1}(k \bmod b)\right) + \left(b^i \sum_{k=sb}^{sb+r}(k \bmod b)\right) \\
&= \left(b^i \sum_{j=0}^{b-1} sj\right) + \left(b^i \sum_{k=0}^{r}(k \bmod b)\right) \\
&= sb^i \left(\sum_{j=0}^{b-1} j\right) + b^i \left(\sum_{k=0}^{r} k\right) \\
&= sb^i \frac{b(b-1)}{2} + b^i \frac{r(r+1)}{2} \\
&= \frac{b^i}{2}(sb(b-1) + r(r+1)).
\end{aligned}$$

Putting it together,

$$N_2 = \frac{b^i}{2}(sb(b-1) + r(r+1)) + \left(n - d_2 b^i\right)(d_2 \bmod b).$$

□

As announced earlier, we can also provide a formula for $M_{\mathbf{b}}(\boldsymbol{k}; P_n)$, which will be key to prove our main result.

Proposition 4.3.5. *If P_n is a Halton point set, then*

$$M_{\mathbf{b}}(\boldsymbol{k}; P_n) = \max\left(\left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor \prod_{j=1}^{s} b_j^{k_j} - \prod_{j=1}^{s} b_j^{k_j}\right), 0\right). \tag{4.7}$$

Proof. Similar to Formula (4.4) for $N_{\mathbf{b}}(\boldsymbol{i}; P_n)$, we can write

$$M_{\mathbf{b}}(\boldsymbol{k}; P_n) = 2\sum_{d=1}^{n-1}(n-d) \times I\left(\left(\sum_{j=1}^{s} d \bmod b_j^{k_j}\right) = 0\right).$$

Since the indicator function is only non-zero at every multiple of $\prod_{j=1}^{s} b_j^{k_j}$ (as the bases used in the Halton sequence are all coprime with each other), $M_{\mathbf{b}}(\boldsymbol{k}; P_n)$ can be simplified as

$$\begin{aligned} M_{\mathbf{b}}(\boldsymbol{k}; P_n) &= 2 \sum_{\ell=1}^{\left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor} \left(n - \ell \prod_{j=1}^{s} b_j^{k_j} \right) \\ &= \left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor \prod_{j=1}^{s} b_j^{k_j} - \prod_{j=1}^{s} b_j^{k_j} \right). \end{aligned}$$ □

Our next goal is to show scrambled Halton point sets are NLOD sampling schemes. We do so by establishing that this NLOD property holds as long as the $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values of the corresponding deterministic point set P_n are no larger than 1 for all $\boldsymbol{k} \in \mathbb{N}^s$, a fact we then prove. We note that Lemma 1.15 of [90] can easily be adapted to show that the NLOD and NUOD properties are equivalent for Halton point sets randomized by a base$-\mathbf{b}$ scrambling, which is why in this section we focus on the NLOD property.

To do this, we use the same method from Section 4.2 of [90], but explain it instead along the lines of Section 3 of [51]. That is, to show the NLOD property we need to show that $\int_{R(\mathbf{x},\mathbf{y})} \psi(\mathbf{u},\mathbf{v}) d\mathbf{u} d\mathbf{v} \leq \text{Vol}(R(\mathbf{x},\mathbf{y}))$ for all $\mathbf{x}, \mathbf{y} \in [0,1]^s$, where $R(\mathbf{x},\mathbf{y}) = \{(\mathbf{u},\mathbf{v}) \in [0,1)^{2s} : u_j < x_j, v_j < y_j, j = 1, \ldots, s\}$. That is, $R(\mathbf{x},\mathbf{y}) = [\mathbf{0},\mathbf{x}) \times [\mathbf{0},\mathbf{y})$. To compute the LHS of this inequality, we use the fact that $\psi(\mathbf{u},\mathbf{v})$ is constant over the regions ${}_{\mathbf{b}}D_{\boldsymbol{i}}^s$: denote this common value by $\psi_{\boldsymbol{i}}$. Hence, we can rewrite this integral as a sum of the form $\sum_{\boldsymbol{i} \in \mathbb{N}^s} \eta_{\boldsymbol{i}} \psi_{\boldsymbol{i}}$, where $\eta_{\boldsymbol{i}} = \text{Vol}({}_{\mathbf{b}}D_{\boldsymbol{i}}^s \cap R(\mathbf{x},\mathbf{y}))$. We then show in Lemma 4.3.7 that the vector in $\ell^1(\mathbb{N}^s)$ formed by the $\eta_{\boldsymbol{i}}$—which we refer to below as the *volume vector* for $R(\mathbf{x},\mathbf{y})$— can be decomposed into a conic combination of volume vectors for simpler regions corresponding to elementary intervals of the form $\mathbf{1}_{\boldsymbol{k}} = \prod_{j=1}^{s} [0, b_j^{-k_j})$. The $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values are then simply the normalized values of the integral of the joint pdf over the region $\mathbf{1}_{\boldsymbol{k}} \times \mathbf{1}_{\boldsymbol{k}}$, as shown in Lemma 4.3.8. Together, these two facts lead us to our main result. Having explained the map to this proof, we can now proceed to define some notation, modified from Section 4.2 of [90], to state the different intermediate results and then our main result.

Definition 4.3.6 (Volume Vector)**.** For $\mathbf{x}, \mathbf{y} \in [0,1)^s$, the volume vector of the region $R(\mathbf{x},\mathbf{y})$ is defined as

$$V^s(\mathbf{x},\mathbf{y}) := (V_{\boldsymbol{i}}^s(\mathbf{x},\mathbf{y}))_{\boldsymbol{i} \in \mathbb{N}^s} \in \ell^1(\mathbb{N}^s),$$

where

$$V_{\boldsymbol{i}}^s(\mathbf{x},\mathbf{y}) = \int_{R(\mathbf{x},\mathbf{y})} 1_{{}_{\mathbf{b}}D_{\boldsymbol{i}}^s}(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} = \mathrm{Vol}(R(\mathbf{x},\mathbf{y})) \cap {}_{\mathbf{b}}D_{\boldsymbol{i}}^s).$$

Observe that $\|V^s(\mathbf{x},\mathbf{y})\|_1 = \mathrm{Vol}(R(\mathbf{x},\mathbf{y}))$.

We can now state the result proving that the volume vector of $R(\mathbf{x},\mathbf{y})$ can be decomposed into a conic combination of simpler volume vectors.

Lemma 4.3.7. *Let $\mathbf{x},\mathbf{y} \in [0,1)^s$ and for $j = 1,\ldots,s$ define*

$$t_{j,0} = \frac{b_j V_0(x_j,y_j)}{b_j - 1} \text{ and } t_{j,k} = \frac{b_j V_k(x_j,y_j) - V_{k-1}(x_j,y_j)}{b_j - 1} \text{ for } k \geq 1.$$

Let $t_{\boldsymbol{k}} = \prod_{j=1}^s t_{j,k}$, $c(\mathbf{b},\boldsymbol{k}) = \prod_{j=1}^s b_j^{2k_j}$, and $\mathbf{b}^{-\boldsymbol{k}} = (b_1^{-k_1},\ldots,b_s^{-k_s})$. Then $V^s(\mathbf{x},\mathbf{y}) = \sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}}\xi^{\boldsymbol{k}}$, where $t_{\boldsymbol{k}} \geq 0$, $\xi^{\boldsymbol{k}} = c(\mathbf{b},\boldsymbol{k})V^s(\mathbf{b}^{-\boldsymbol{k}},\mathbf{b}^{-\boldsymbol{k}})$, and $\sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}} = \mathrm{Vol}(R(\mathbf{x},\mathbf{y}))$.

Proof. The proof of this result follows exactly the proof of Lemma 4.11 from [90], as using a different base for each dimension does not change the behaviour of the sums and products used in the underlying calculations. Note that $\xi^{\boldsymbol{k}}$ is just the normalized version of the volume vector corresponding to the region $\mathbf{1}_{\boldsymbol{k}} \times \mathbf{1}_{\boldsymbol{k}}$, with $c(\mathbf{b},\boldsymbol{k})$ the normalizing constant. □

Lemma 4.3.8. *Let ${}_{\mathbf{b}}\tilde{P}_n$ be a scrambled Halton point set. If $\psi(\mathbf{x},\mathbf{y})$ denotes the joint pdf of two distinct points randomly chosen from ${}_b\tilde{P}_n$ then*

$$c(\mathbf{b},\boldsymbol{k}) \int_{\mathbf{1}_{\boldsymbol{k}}\times\mathbf{1}_{\boldsymbol{k}}} \psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} = C_{\mathbf{b}}(\boldsymbol{k};P_n).$$

Proof. Firstly, as mentioned above, the vector $\xi^{\boldsymbol{k}} \in \ell^1(\mathbb{N}^s)$ is the normalized version of the volume vector corresponding to the region $\mathbf{1}_{\boldsymbol{k}} \times \mathbf{1}_{\boldsymbol{k}}$. Any point $(\mathbf{x},\mathbf{y})$ in that region must be such that $\mathbf{x}$ and $\mathbf{y}$ share at least $\boldsymbol{k}$ digits. Hence, $\xi^{\boldsymbol{k}}$ is non-zero only in positions of the form $\boldsymbol{k}+\boldsymbol{i}$, where it is given by $\xi^{\boldsymbol{k}}_{\boldsymbol{i}+\boldsymbol{k}} = \prod_{j=1}^s \frac{b_j-1}{b_j^{1+i_j}}$.

Combining this fact with the form of the joint pdf given in (4.2), for each $\boldsymbol{k} \in \mathbb{N}^s$ we

have

$$
\begin{aligned}
c(\mathbf{b},\boldsymbol{k})\int_{\mathbf{1}_{\boldsymbol{k}}\times\mathbf{1}_{\boldsymbol{k}}}\psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} &= \sum_{\boldsymbol{i}\in\mathbb{N}^s}\xi_{\boldsymbol{i}}^{\boldsymbol{k}}\psi_{\boldsymbol{i}} = \sum_{\boldsymbol{i}\in\mathbb{N}^s}\psi_{\boldsymbol{i}+\boldsymbol{k}}\prod_{j=1}^{s}\frac{b_j-1}{b_j^{1+i_j}}\\
&= \sum_{\boldsymbol{i}\in\mathbb{N}^s}\frac{N_{\mathbf{b}}(\boldsymbol{i}+\boldsymbol{k};{}_{\mathbf{b}}\tilde{P}_n)}{n(n-1)}\prod_{j=1}^{s}\frac{b_j-1}{b_j^{1+i_j}}\prod_{j=1}^{s}\frac{b_j^{1+i_j+k_j}}{b_j-1}\\
&= \sum_{\boldsymbol{i}\in\mathbb{N}^s}\frac{N_{\mathbf{b}}(\boldsymbol{i}+\boldsymbol{k};{}_{\mathbf{b}}\tilde{P}_n)}{n(n-1)}\prod_{j=1}^{s}b_j^{k_j} = \left(\prod_{j=1}^{s}b_j^{k_j}\right)\frac{1}{n(n-1)}\sum_{\boldsymbol{i}\in\mathbb{N}^s}N_{\mathbf{b}}(\boldsymbol{i}+\boldsymbol{k};{}_{\mathbf{b}}\tilde{P}_n)\\
&= \left(\prod_{j=1}^{s}b_j^{k_j}\right)\frac{M_{\mathbf{b}}(\boldsymbol{k};{}_{\mathbf{b}}\tilde{P}_n)}{n(n-1)} = \left(\prod_{j=1}^{s}b_j^{k_j}\right)\frac{M_{\mathbf{b}}(\boldsymbol{k};P_n)}{n(n-1)} = C_{\mathbf{b}}(\boldsymbol{k};P_n). \qquad \square
\end{aligned}
$$

Now we have all the pieces to be able to show that $C_{\mathbf{b}}(\boldsymbol{k};P_n)\le 1$ for all $\boldsymbol{k}\in\mathbb{N}^s$ implies $\int_{R(\mathbf{x},\mathbf{y})}\psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v}\le \mathrm{Vol}(R(\mathbf{x},\mathbf{y}))$ for all $\mathbf{x},\mathbf{y}\in[0,1)^s$.

Theorem 4.3.9. *Let ${}_{\mathbf{b}}\tilde{P}_n$ be a scrambled Halton point set and let $\psi(\mathbf{u},\mathbf{v})$ be the joint pdf of two distinct points randomly chosen from ${}_{b}\tilde{P}_n$. Let $t_{\boldsymbol{k}}$ be the coefficient defined in Lemma 4.3.7 for a given $\mathbf{x},\mathbf{y}\in[0,1)^s$. Then*

$$
\int_{R(\mathbf{x},\mathbf{y})}\psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} = \sum_{\boldsymbol{k}\in\mathbb{N}^s}t_{\boldsymbol{k}}\,C_{\mathbf{b}}(\boldsymbol{k};P_n)
$$

$$
\textit{and thus}\quad \int_{R(\mathbf{x},\mathbf{y})}\psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v}\le \mathrm{Vol}(R(\mathbf{x},\mathbf{y}))\max_{\boldsymbol{k}\in\mathbb{N}^s}C_{\mathbf{b}}(\boldsymbol{k};P_n). \tag{4.8}
$$

Hence if $C_{\mathbf{b}}=\max_{\boldsymbol{k}\in\mathbb{N}^s}C_{\mathbf{b}}(\boldsymbol{k};P_n)\le 1$, then $\int_{R(\mathbf{x},\mathbf{y})}\psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v}\le\mathrm{Vol}(R(\mathbf{x},\mathbf{y}))$ and ${}_{\mathbf{b}}\tilde{P}_n$ is NLOD.

Proof. From Lemmas 4.3.7 and 4.3.8,

$$\begin{aligned}\int_{R(\mathbf{x},\mathbf{y})} \psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} &= \sum_{\boldsymbol{i}\in\mathbb{N}^s} \psi_{\boldsymbol{i}} V_{\boldsymbol{i}}^s(\mathbf{x},\mathbf{y}) \\ &= \sum_{\boldsymbol{i}\in\mathbb{N}^s} \psi_{\boldsymbol{i}} \sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}} c(\mathbf{b},\boldsymbol{k}) V_{\boldsymbol{i}}^s(\mathbf{b}^{-\boldsymbol{k}},\mathbf{b}^{-\boldsymbol{k}}) \\ &= \sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}} c(\mathbf{b},\boldsymbol{k}) \int_{\mathbf{1}_{\boldsymbol{k}}\times\mathbf{1}_{\boldsymbol{k}}} \psi(\mathbf{u},\mathbf{v})d\mathbf{u}d\mathbf{v} \\ &= \sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}} C_{\mathbf{b}}(\boldsymbol{k};P_n).\end{aligned}$$

The inequality (4.8) is obtained by using the fact that $t_{\boldsymbol{k}} \geq 0$, $\sum_{\boldsymbol{k}\in\mathbb{N}^s} t_{\boldsymbol{k}} = \mathrm{Vol}(R(\mathbf{x},\mathbf{y}))$. and also recalling that only a finite number of vectors $\boldsymbol{k}$ are such that $C_b(\boldsymbol{k};P_n) > 0$ by property of the Halton sequence. □

Theorem 4.3.10. *Let P_n be a Halton point set. Then $C_{\mathbf{b}}(\boldsymbol{k},P_n) \leq 1$ for all $\boldsymbol{k}\in\mathbb{N}^s$ and thus $C_{\mathbf{b}} := \sup_{\boldsymbol{k}\in\mathbb{N}^s} C_{\mathbf{b}}(\boldsymbol{k},P_n) \leq 1$.*

The proof for Theorem 4.3.10 is, unlike the previous proofs presented in this section, specific to the Halton sequence.

Proof. If $\prod_{j=1}^s b_j^{k_j} \geq n$, then, $M_{\mathbf{b}}(\boldsymbol{k};P_n) = 0$ and thus $C_{\mathbf{b}}(\boldsymbol{k},P_n) = 0$. Thus, we only need to consider the case where $\prod_{j=1}^s b_j^{k_j} \leq n-1$ which implies $M_{\mathbf{b}}(\boldsymbol{k};P_n) > 0$. Using Definition 4.2.2 and Proposition 4.3.5, we have

$$\begin{aligned}C_{\mathbf{b}}(\boldsymbol{k},P_n) &= \frac{\prod_{j=1}^s b_j^{k_j} M_{\mathbf{b}}(\boldsymbol{k};P_n)}{n(n-1)} \\ &= \frac{\prod_{j=1}^s b_j^{k_j} \left\lfloor \frac{n-1}{\prod_{j=1}^s b_j^{k_j}} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{\prod_{j=1}^s b_j^{k_j}} \right\rfloor \prod_{j=1}^s b_j^{k_j} - \prod_{j=1}^s b_j^{k_j}\right)}{n(n-1)}.\end{aligned}$$

Now, write $\left\lfloor \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} \right\rfloor = \frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} - r$, where $0 \leq r < 1$. Then

$$\begin{aligned}
C_{\mathbf{b}}(\boldsymbol{k}, P_n) &= \frac{\prod_{j=1}^{s} b_j^{k_j} \left(\frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} - r \right) \left(2n - \left(\frac{n-1}{\prod_{j=1}^{s} b_j^{k_j}} - r \right) \prod_{j=1}^{s} b_j^{k_j} - \prod_{j=1}^{s} b_j^{k_j} \right)}{n(n-1)} \\
&= \frac{-r^2 \left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 + \left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r - \left(\prod_{j=1}^{s} b_j^{k_j} \right) n}{n(n-1)} \\
&\quad + \frac{\left(\prod_{j=1}^{s} b_j^{k_j} \right) - 2 \left(\prod_{j=1}^{s} b_j^{k_j} \right) r + n^2 - 1}{n(n-1)} \\
&= \frac{n - \left(\prod_{j=1}^{s} b_j^{k_j} \right)}{n-1} \\
&\quad + \frac{1}{(n-1)n} \left(\left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r - \left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r^2 - 2 \left(\prod_{j=1}^{s} b_j^{k_j} \right) r + \left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1 \right).
\end{aligned}$$

Now, $C_{\mathbf{b}}(\boldsymbol{k}, P_n) \leq 1$ if and only if

$$\frac{\left(\left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r - \left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r^2 - 2 \left(\prod_{j=1}^{s} b_j^{k_j} \right) r + \left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1 \right)}{n} \leq \left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1.$$

Rearranging yields:

$$\begin{aligned}
&\frac{\left(\left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r - \left(\prod_{j=1}^{s} b_j^{k_j} \right)^2 r^2 - 2 \left(\prod_{j=1}^{s} b_j^{k_j} \right) r + \left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1 \right)}{\left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1} \leq n \\
&\qquad \iff \left(\prod_{j=1}^{s} b_j^{k_j} \right) (r - r^2) + (1 - r - r^2) - \frac{r + r^2}{\left(\prod_{j=1}^{s} b_j^{k_j} \right) - 1} \leq n.
\end{aligned}$$

The last inequality is satisfied, as the maximum value that $\prod_{j=1}^{s} b_j^{k_j}$ can take is $n-1$, and thus the first term in the sum is at most $n-1$ as $0 \leq r \leq 1$, the second term is at most 1, and the third term is negative. Thus, $C_{\mathbf{b}}(\boldsymbol{k}, P_n) \leq 1$ for all $\boldsymbol{k} \in \mathbb{N}^s$, so

$C_{\mathbf{b}} = \sup_{\boldsymbol{k} \in \mathbb{N}^s} C_{\mathbf{b}}(\boldsymbol{k}, P_n) \leq 1.$ □

Combining Theorem 4.3.10 with Theorem 4.3.9, we have the main result of this chapter:

Theorem 4.3.11. *Let ${}_{\mathbf{b}}\tilde{P}_n$ be a scrambled Halton point set. Then, ${}_{\mathbf{b}}\tilde{P}_n$ is an NLOD sampling scheme.*

4.4 Assessing the quality of Halton sequences via dependence measures

The dependence framework introduced in the previous sections has allowed us to analyze scrambled Halton point sets. It can also be used to design quality criteria for measuring the uniformity of different low-discrepancy sequences, including Halton sequences.

As argued in [90], the $C_b(\boldsymbol{k}; P_n)$ values can be interpreted as a measure such that when it is smaller than 1, scrambling P_n will yield a well-distributed point set that can in turn produce estimators for multivariate integrals with lower variance than MC, under some conditions on the integrand. It is also argued in [90] that one can compute $C_b(\boldsymbol{k}; P_n)$ values in a base b other than the one used to construct the point set, because if it remains relatively small in other bases (e.g., in base 2), in some sense it suggests that scrambling does not need to be performed in a specific base in order for the randomized point set to have good uniformity properties, which suggests a form of "robustness" for P_n that should hold only if it is already well distributed, in which case a simple digital shift might be enough to randomize it. In particular, when b is large, evaluating $C_2(\boldsymbol{k}; P_n)$ is more likely to identify potential issues with the point set P_n than using the base in which the point set was constructed, as when n is small compared to b, it is easy for P_n to obtain small $C_b(\boldsymbol{k}; P_n)$ values, as discussed in [90].

As mentioned in the introduction, the approaches that have been used to measure the quality of generalized and randomized Halton sequences have been to either use them to integrate certain test-functions, or compute their L_2-discrepancy. The latter is more useful to compare the asymptotic behaviour of low-discrepancy sequences, and is not necessarily that useful to compare point sets P_n with relatively small sample sizes [52].

Here, we suggest measuring the quality of low-discrepancy point sets by using a criterion based on the $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values for different vectors $\boldsymbol{k}$ and for $\mathbf{b} = (2, \ldots, 2)$. In order to make sense of the proposed criterion, it is useful to notice that the set of non-zero components of $\boldsymbol{k}$ correspond to the coordinates over which the quality of the point set is measured.

We use the criterion $c(\mathbf{b}, \mathcal{K}; P_n)$ as defined in Section 3.2.2. Since $\mathbf{b}$ is of the form $\mathbf{b} = (b, \ldots, b)$, we simply write $c(b, \mathcal{K}; P_n)$. We use this criterion for two purposes: first, we use it within a search algorithm to find good factors to define a generalized Halton sequence that we expect to have better uniformity properties than the original Halton sequence. Second, we use it to compare low-discrepancy point sets based on different construction paradigms (i.e., nets in base 2, base $b \geq s$, or Halton) that would otherwise be hard to compare, at least for finite sample sizes.

4.4.1 Searching for good multipliers for the Halton sequence

In this subsection, for a s-dimensional point set, we use the $c(2, \mathcal{K}_{d,w,s}; P_n)$ criterion with $d = 3$ and $w \in \{5, 8\}$ to find factors f_j for a simple one-factor linear deterministic scrambling of the Halton sequence of the form $\sigma_j(i) = (f_j i) \mod b_j$ for $j = 1, \ldots, s$, as in [26].

These simple permutations are chosen over more sophisticated permutations for several reasons: i) we want permutations that satisfy $\sigma(0) = 0$ to ensure the sequence remains unbiased [13], as in practice a finite number of digits are generated for each point within the Halton sequence; ii) it takes much less space to store a list of factors than a list of permutations; iii) it allows for a more comprehensive search—we can test every $f_j \in \{1, \ldots, b_j - 1\}$ rather than taking a subset of unrestricted permutations.

Algorithm 1 below describes the component-by-component approach [74] to find the successive factors f_j. The window size w is chosen to be equal to 8 for $j \leq 120$ and 5 for $j > 120$. These relatively small window sizes were chosen for runtime reasons. As mentioned above, the projection size is $d = 3$. A small projection size is used as it keeps the runtime low, and for our test functions as well as many financial applications, even high dimensional integration problems have low effective dimension [89]. The effective dimension of an integration problem is the size of the projection that explains the majority of the variance of a function. For example, if a function has an effective dimension of 2 in the superposition sense with a threshold of 0.99, it means that 99% of the variance can be explained with 2-dimensional projections of these functions. Thus, if the lower dimensional projections of the point set are good, the performance of the point set on the integration problem should be good as well.

The reason for decreasing the window and projection size for large j is for runtime reasons: the calculation for the unrestricted $C_{\mathbf{b}} = c(\mathbf{b}, \mathcal{K}_{s,s,s}; P_n)$ is $O(n^2 s + n^2 s \sum_{k=1}^{\gamma_{\max}} \binom{k+s-1}{k}) = O(n^2 s + n^2 s \binom{\gamma_{\max}+s}{s})$, where the sum is simplified using the Chu-Vandermonde identity. Here, $\gamma_{\max} = \max_{\mathbf{x},\mathbf{y}} \boldsymbol{\gamma}^s_{\mathbf{b}}(\mathbf{x}, \mathbf{y})$, the maximum number of common digits shared between two

points. The complexity depends on the maximum value of γ between all pairs of points, and as discussed previously, the correlation between consecutive dimensions increases with the dimension, causing the value of $\gamma_{\max}$ and thus the runtime to increase. This is in addition to the fact that the search for the best f_j in higher dimension has more potential values to consider as b_j increases with j. Thus, by restricting the window and projection size, we can obtain results within a reasonable timeframe.

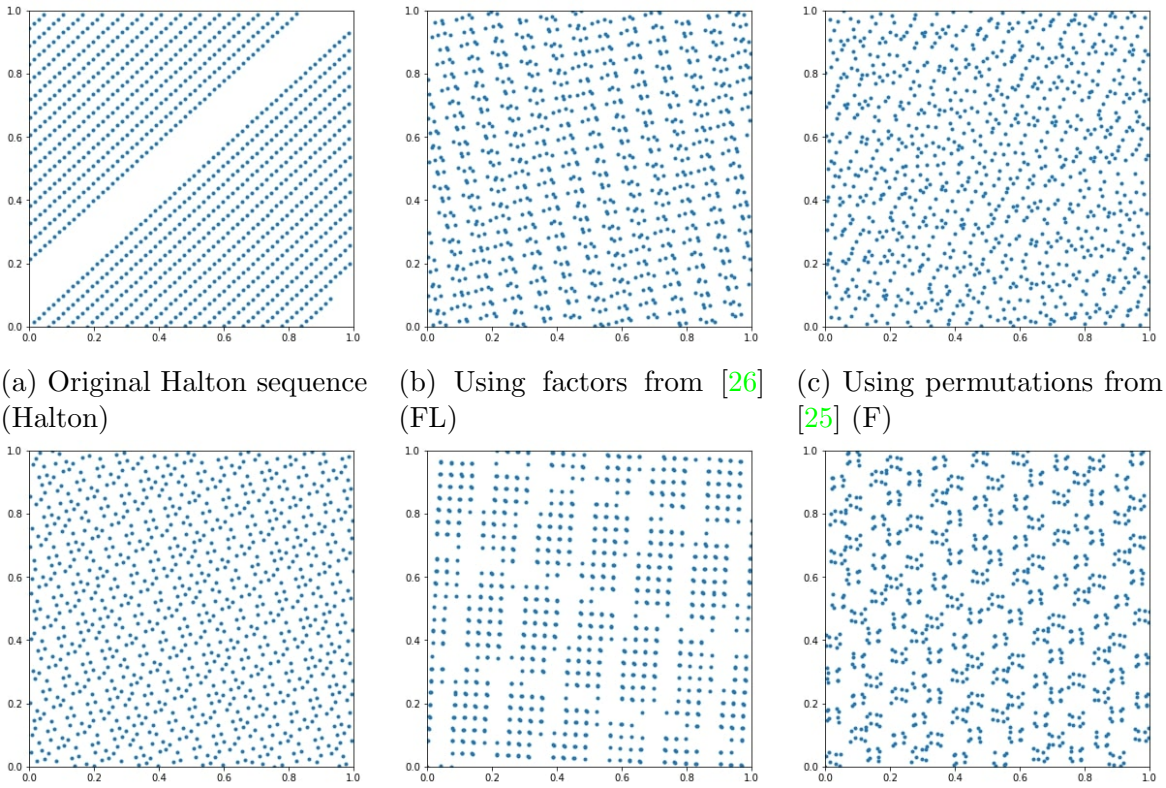
Algorithm 3: Algorithm to search for Halton factors using $c(2, \mathcal{K}_{d,w_j,s}; P_n)$ criterion to find j^{th} factor.

```
INPUT: s, n, d, w[1...s];
OUTPUT: factors[1...s];
factors[1] = 1;
for j in 2,...,s: do
    p = j^th smallest prime;
    wj = min(j,w[j]);
    for i in 1,...,p-1 do
        factors[j] = i;
        pointSet = genHalton(factors[1:j],n)[,(j-wj+1):j] # Generates an n by j
          array of points from the generalized Halton sequence using the first j
          factors and returns the last wj dimensions;
        c2Values[i] = c(2, K_{d,wj,j}; pointSet);
    factors[j] = index i where c2Values takes its minimum;
return factors
```

Table A.1 in the appendix gives the factors f_j found using this algorithm along with their values of $c(2, \mathcal{K}_{d,w_j,s}; P_n)$. Using the Compute Canada cluster Graham on a single NVIDIA Tesla P100 GPU with 12 GB RAM using a Python script, the results in Table A.1 were found after several days of searching.

4.4.2 Comparing different constructions

As mentioned earlier, the $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values can also be used to make comparisons between point sets based on different construction paradigms. To illustrate this point, in Table 4.1 we compute $c(2, \mathbb{N}^2, P_n)$ values for the point sets shown in Figure 4.1, which are two-dimensional point sets obtained by taking the projection of various constructions over the 19^{th} and 20^{th} coordinates. This demonstrates that point sets with fewer clusters and empty spaces results in smaller $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values.



(a) Original Halton sequence (Halton)

(b) Using factors from [26] (FL)

(c) Using permutations from [25] (F)

(d) Using factors found using $c(2, \mathcal{K}_{d,w,s}; P_n)$ values (DL)

(e) Generalized Faure Sequence (gFaure)

(f) Sobol' Sequence (Sobol)

Figure 4.1: First 1000 points of different point sets over coordinates 19 and 20.

k	1	2	3	4	5	6	7	8	9	C_b
Halton	0.999	1.01	1.01	1.03	1.04	1.05	1.51	2.50	3.37	3.37
FL	0.999	0.997	0.994	0.987	0.988	0.961	0.992	0.989	1.01	1.01
F	0.999	0.998	0.994	0.989	0.992	1.00	0.999	1.03	0.928	1.03
DL	0.999	0.997	0.993	0.986	0.972	0.947	0.909	0.841	0.853	0.999
Faure	0.999	1.01	1.01	1.02	1.03	1.09	1.34	1.77	2.43	2.43
Sobol	0.999	0.997	0.993	0.986	0.972	0.944	1.68	1.64	1.42	1.68

Table 4.1: $c(2, \mathbb{N}^2, P_n)$ values for the 2−dimensional point sets in Figures 4.1.

4.5 Numerical experiments on the Halton sequence

In this section, we conduct some experiments on numerical integration problems to compare different Halton sequences constructions. Our goals are two-fold: first, we want to assess the performance of the factors we found in the previous section compared with other constructions. Second, we want to numerically investigate the more fundamental question of whether it is best to address the shortcomings of the original Halton sequence via random scrambling or via a well-chosen generalized construction that can then be randomized using a simple (and cheaper to compute) digital shift.

To address this latter question, we designed an experiment involving the well-known family of Genz integrand functions. In this experiment, we use histograms to compare the integration error obtained using specific generalized Halton sequences— randomized via a digital shift—with the variance distribution obtained using a base $\mathbf{b}$ random linear scrambling. The point of this experiment is as follows: if a specific construction can easily be "beaten" by (have a larger error than) a randomized sequence obtained through scrambling, then this would be an argument in favour of using random scrambling to improve the Halton sequence instead of relying on a specific generalized Halton sequence construction. If a construction consistently ranks better than most scrambled sequences, then this would be an argument to use those instead of scrambled sequences.

The generalized Halton sequences considered in this experiment are: the one obtained using the factors from the previous section (referred to as "DL Factors"), the one based on the factors from [26]—referred to as FL factors—and we also assess the original Halton sequence. In addition, we provide results obtained using the MC method.

4.5.1 Experiments with Genz integrand functions

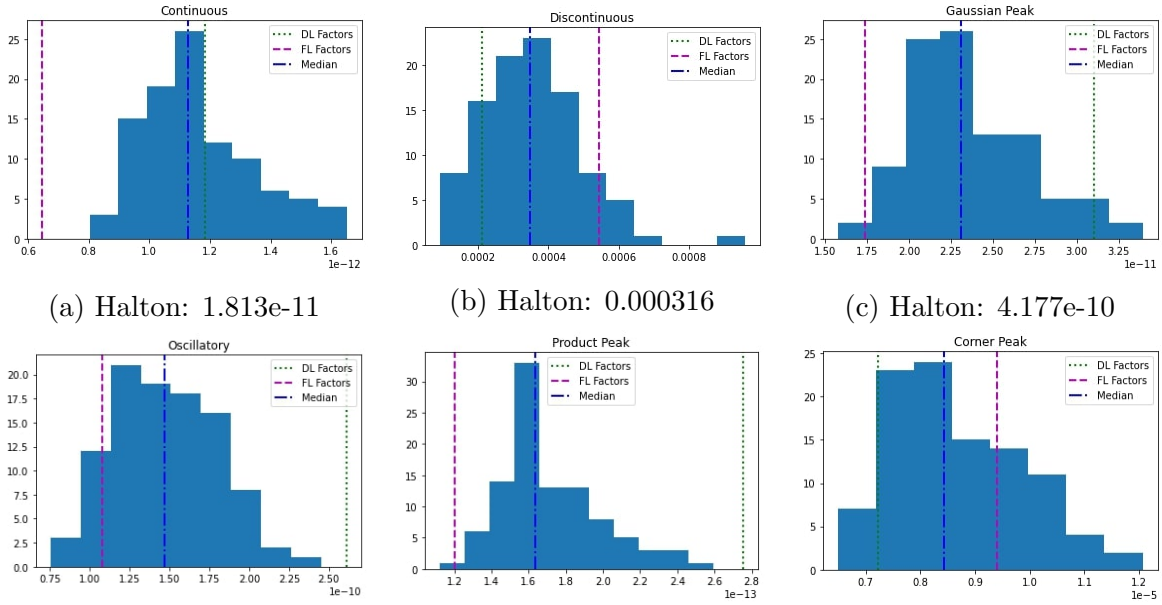
The Genz integrand functions [29] are summarized in Table 4.2. Each of the six integrand families has a different property that makes it "difficult" to integrate. Each function depends on two parameters. The vector $\mathbf{a}$ is the scaling parameter and is randomly generated uniformly over $[0,1)^s$, and then scaled such that $s^{e_j}|\mathbf{a}| = h_j$, where j indexes the integrand family. This is to fix the integration difficulty for a series of tests, and larger values of $||\mathbf{a}||$ result in a more difficult integration. We use $\mathbf{e} = (2,2,2,1,1.5,2)$ and $\mathbf{h} = (150,600,100,100,110,600)$, as suggested in [30] and used in [60]. The vector $\mathbf{b}$ is the location parameter, and is randomly generated uniformly over $[0,1)^s$ without any additional scaling. To obtain meaningful results, the parameters $\mathbf{a}$ and $\mathbf{b}$ are randomly generated, and the numerical experiment is repeated R = 50 times. The reported error

is the average of the 50 obtained estimated mean square relative errors (MSRE), and is estimated using $V = 25$ replications. The same set of parameters are used for all methods and randomizations for a fair comparison. The tests are performed in dimension $s = 20$ and with a sample size of $n = 10,000$.

Figure 4.2 shows histograms of 100 MSREs for each of the Genz functions based on a randomly linearly scrambled Halton point set, as explained above. For each of the six functions, either DL or FL appears to be more accurate than scrambled point sets (e.g., in the lower quartile of the histogram), but neither is consistently doing so. This suggests that well-chosen factors have the potential to do better than scrambling, but probably need to be chosen based on the type of function being integrated. Hence, if one is looking for a multipurpose construction, scrambling appears as a better choice.

Integrand Family	Attribute
$u_1(\mathbf{x}) = \cos\left(2\pi b_1 + \sum_{i=1}^{s} a_i x_i\right)$	Oscillatory
$u_2(\mathbf{x}) = \prod_{i=1}^{s} \left(a_i^{-2} + (x_i - b_i)^2\right)^{-1}$	Product Peak
$u_3(\mathbf{x}) = \left(1 + \sum_{i=1}^{s} a_i x_i\right)^{-(s+1)}$	Corner Peak
$u_4(\mathbf{x}) = \exp\left(-\sum_{i=1}^{s} a_i^2 (x_i - b_i)^2\right)$	Gaussian Peak
$u_5(\mathbf{x}) = \exp\left(-\sum_{i=1}^{s} a_i \lvert x_i - b_i\rvert\right)$	Continuous
$u_6(\mathbf{x}) = \exp\left(\sum_{i=1}^{s} a_i x_i\right) \mathbf{1}_{x_1 \le b_1, x_2 \le b_2}$	Discontinuous

Table 4.2: Six families of Genz integrand functions.



(a) Halton: 1.813e-11 (b) Halton: 0.000316 (c) Halton: 4.177e-10

(d) Halton: 3.213e-09 (e) Halton: 3.382e-12 (f) Halton: 1.558e-05

Figure 4.2: Genz integrand family functions: Comparing our factors, factors from `qrng` and the original Halton sequence with 100 Halton sequences randomized with Random Linear Scrambling.

4.5.2 Experiments with fixed-parameters integration problems

We now consider specific test-functions with fixed parameters and defined in higher dimensions than the previous experiment. Specifically, from Section 2.5, we consider functions g, the Asian Call Option pricing problem, as well as the Mortgage Backed Security problem. We study the behaviour of the estimated MSE or variance of the different estimators as n increases from 2000 to 100,000 rather than fixing the sample size to $n = 10,000$ as in the previous subsection. Table 4.3 describes the constructions being compared in the experiments in this section.

MC:	Monte Carlo
DL Factors:	Permutations found using $c(2, \mathcal{K}_{d,w,s}; P_n)$ values, then randomize with a digital shift in base b_j, $j = 1, \ldots, s$
FL Factors:	Permutations from [26] then randomize with a digital shift in base b_j, $j = 1, \ldots, s$
RLS:	Random linear scrambling in base b_j, $j = 1, \ldots, s$
Okten:	Ökten's "random start scrambled Halton sequence", with implementation provided in [85]
Owen:	Owen's scrambling from [63]
Shift:	Original Halton sequence with digital shift

Table 4.3: Methods compared in Figure 4.3.

Sobol' test function

The first test function we consider is given by $g(\mathbf{x}) = \prod_{j=1}^{s} (1 + c(x_j - 0.5))$, with $s = 120, c = 0.1$, and $s = 96, c = 0.25$. The true value of the integral of g is 1 for both cases. Figure 4.3 shows the estimated MSE for both cases. The MSEs are estimated using $V = 100$ independent randomizations. The larger number of independent randomizations are used for this experiment to create smoother plots for easier visual comparisons, especially for smaller values of n. We can see that the random linear scrambling outperforms using well-chosen factors, and as n increases, outperforms all other methods except for Owen's scrambling, which is much more computationally effective but gives comparable results. Comparing the two sets of permutations, the factors found using the $c(2, \mathcal{K}_{d,w,s}; P_n)$ values seems to outperform those from [26].

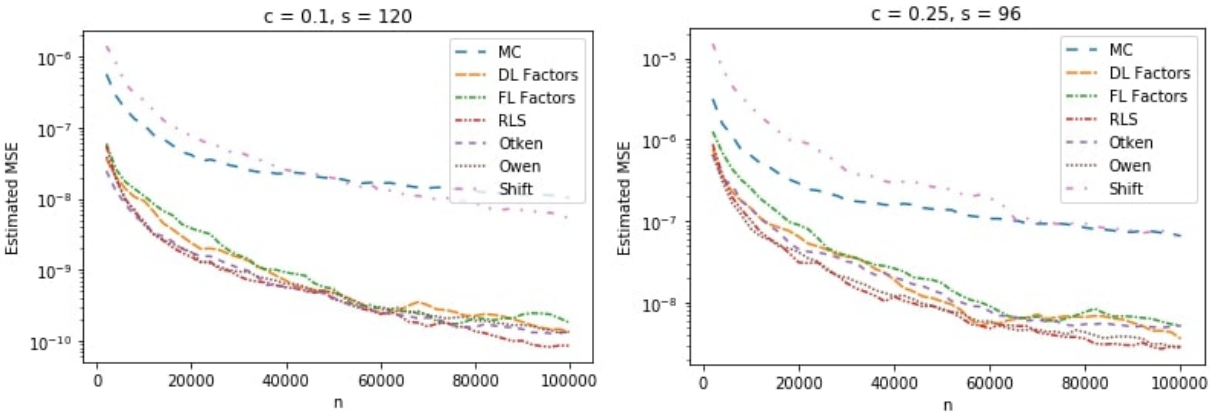


Figure 4.3: Estimated MSEs when Integrating Test Function $g(\mathbf{x})$.

We also repeat the experiment from Section 4.5.1 but with this test function instead of the Genz integrands. Hence, here the function is fixed. The MSE is again estimated using 25 RQMC randomizations. The sample size is $n = 10,000$. Figure 4.4 compares random linear scrambling with the DL and FL factors, as well as the Halton sequence (randomly shifted) and MC. As was seen in Figure 4.3, random linear scrambling outperforms using either set of well-chosen factors. The results on this test function further motivate using a base $\mathbf{b}$-digital scramble to randomize the Halton sequence.

Arithmetic Asian call option

Next, we consider the Asian Call option, with parameters specified in Section 2.5. These results were not part of [22].

Figure 4.5 shows the estimated variances for each of these cases. Variances are estimated using $V = 25$ independent randomizations. The variances seem similar between the different randomization methods, but as n increases, we can see some separation between the results, especially for the $s = 75, K = 55$ and $s = 40, K = 45$ case where the factors from [26] outperform the other methods, and in the $s = 75, K = 50$ case where using random linear scrambling gives the lowest variance.

Mortgage backed security

Finally, we consider the mortgage backed security problem. These results were not part of [22]. Figure 4.6 shows the estimated MSE using 25 replications.

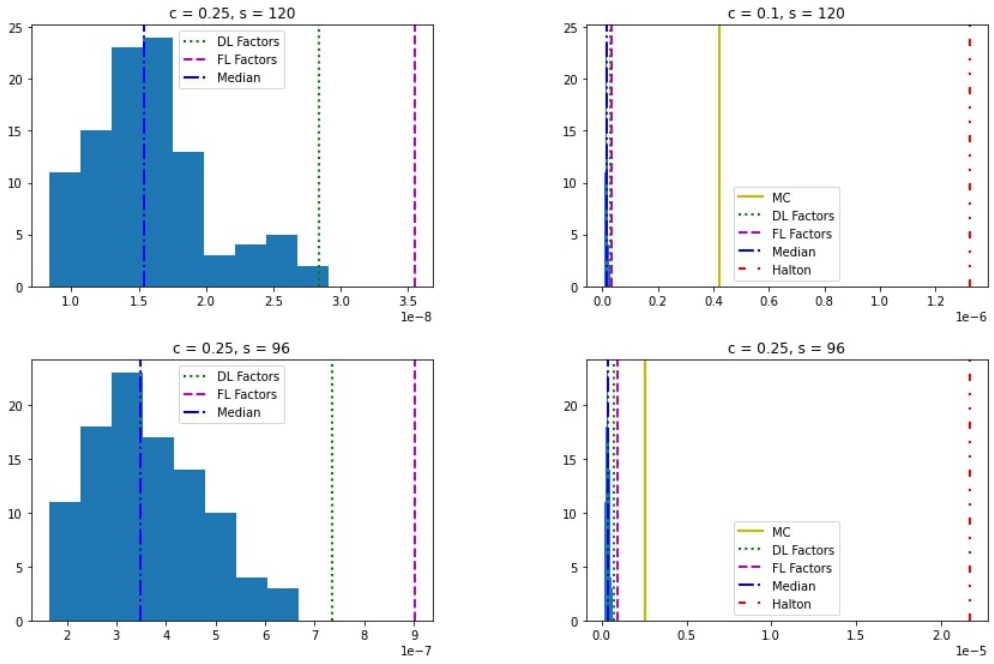


Figure 4.4: Test function $g(\mathbf{x})$: Comparing our factors, factors FL from `qrng`, MC, and the original Halton sequence with 100 Halton sequences randomized with Random Linear Scrambling; on the LHS we exclude Halton and MC to better see the histograms.

From these plots, we can see that random linear scrambling is consistently one of the best methods to use.

4.6 Assessing different forms of scrambling

This work in this section has not been published yet and is the topic of the working paper [20], which is done in collaboration with Christiane Lemieux and Henri Faure. This section uses the same setup and experiments as in Sections 3.2.3 - 3.2.5, and we include this section here rather than in the previous chapter due to the work being on the Halton sequence, which is the focus of this chapter, whereas the previous chapter focuses on experiments on the Sobol' and Faure sequences.

As shown earlier in this chapter, a form of perfect equidistribution holds for the Halton sequence. However, just like for the Faure sequence, this optimal behaviour can require a

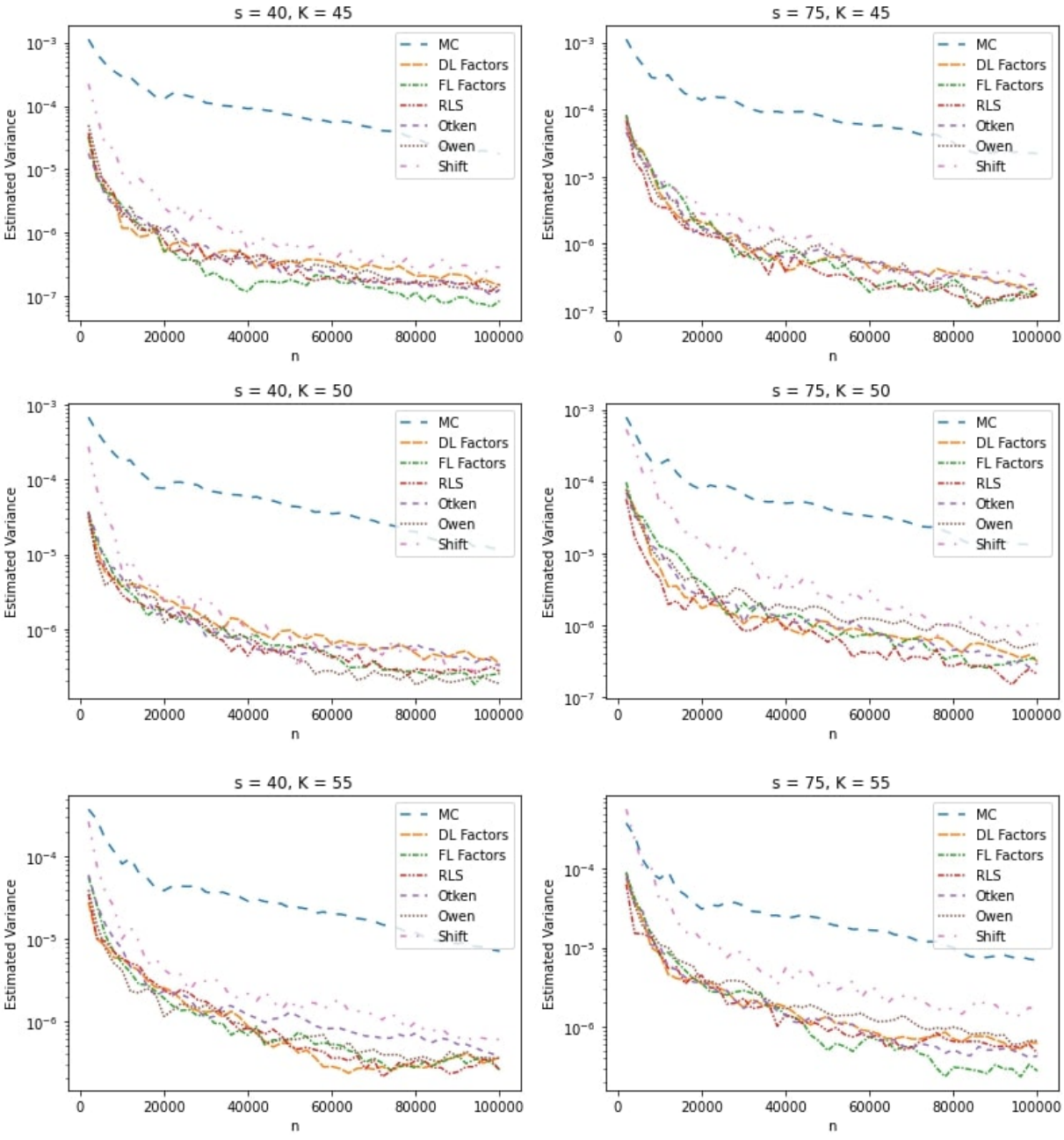


Figure 4.5: Estimated Variances of the Asian Option Pricing Problem.

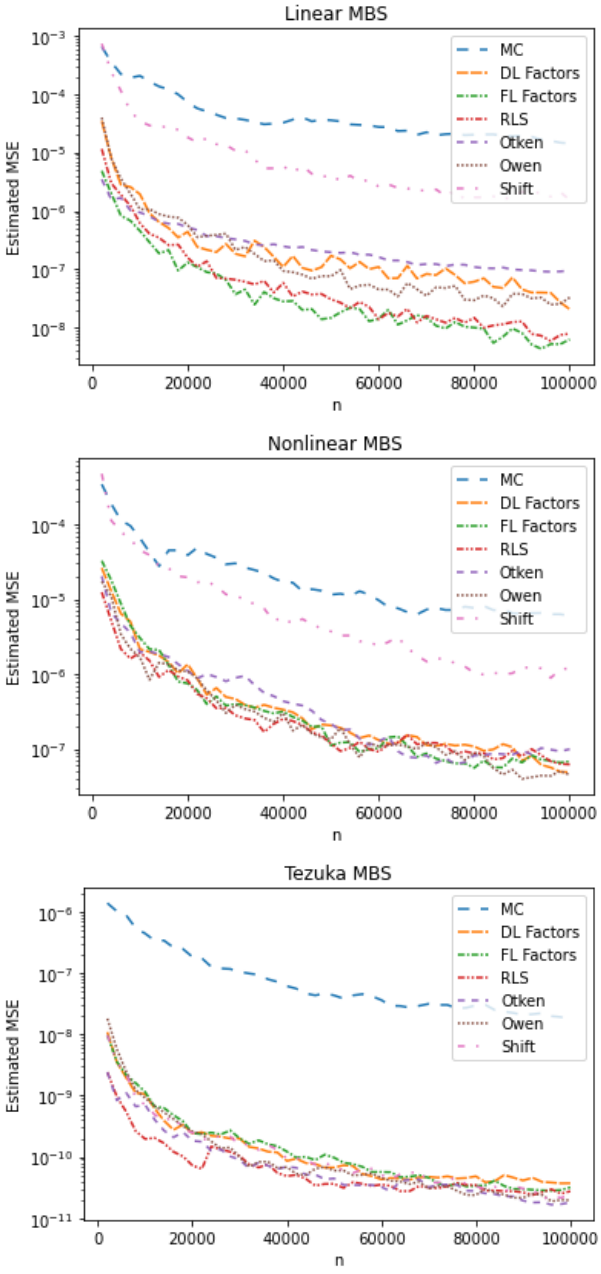


Figure 4.6: Estimated MSEs of the Mortgage Backed Security Pricing Problem.

very large number of points before being observed, which can be observed in the top-left image of Figure 4.1, which consists of points that are constructed in bases 67 and 71 in their two coordinates: there are not enough points to evenly fill out the unit square as the image is of the first 1000 points of the sequence and to see good properties in this two-dimensional projection, we would need to have $67 \times 71 = 4757$ points.

To alleviate this problem, as mentioned earlier in this chapter, several authors have proposed to apply both deterministic and random scrambling to the Halton sequence. Some early work in this area goes back to [25], who proposed to use permutations of the integers in $[0, \dots, b-1]$ for the van der Corput sequence in base b. Following this work, many other deterministic scramblings based on permutations have been proposed for the Halton sequences: see [22, 26, 85] and the references therein. The scramblings based on factors studied in the previous sections of this chapter can be thought of as a special case based on very simple permutations consisting of a multiplication by a factor.

We explore the use of permutations based on the ones originally proposed in [25] to construct generalized Halton sequences. Algorithm 2 on page 46 from Chapter 3 summarizes the steps for generating a permutation π_b for any integer b, and we consider the original permutations from [25], as well as ones reordered with an additional offset term as explained on page 47 in Section 3.2.3, which we refer to as the "Offset" permutation, e.g., in Table 4.5 and 4.4. While these permutations have been proposed to improve the discrepancy of one-dimensional van der Corput sequences, they can (and have been) used to create generalized Halton sequences [58], which is what we propose to analyze via dependence concepts.

4.6.1 Comparisons based on negative dependence criterion

In Tables 4.4 and 4.5, we compute the two criteria as explained in Section 3.2.2 for point sets based on the (generalized) Halton sequence. Here, "Regular" refers to the original construction from [35], "Faure 1992" refers to the permutations obtained using the method from [25], as detailed in Algorithm 2 on page 46, while "Offset" refers to the permutation obtained when we add $m = \lfloor b/2 \rfloor$ to each term of the "Faure 1992" permutation. In all cases, we consider every 2-dimensional projection over the point set when computing the criteria, and the values reported give us information about the two-dimensional projections of these point sets.

Halton Permutations	s	d	n	$c(2, \mathcal{K}_{d,w=s,s}, P_n)$	$\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$
Regular	4	2	3125	0.999685	0.999683
Faure 1992	4	2	3125	0.999682	0.999682
Offset	4	2	3125	0.999681	0.99968
Regular	12	2	2197	4.036052	1.892203
Faure 1992	12	2	2197	1.819365	1.071718
Offset	12	2	2197	1.635136	1.053001
Regular	52	2	2809	13.65113	6.534291
Faure 1992	52	2	2809	4.204183	1.21624
Offset	52	2	2809	4.179257	1.197399

Table 4.4: Values of criteria based on $C_2(\boldsymbol{k}; P_n)$ for point sets obtained from (generalized) Halton Sequences.

Halton Permutations	s	d	n	$c(2, \mathcal{K}_{d,w=s,s}, P_n)$	$\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$
Regular	4	2	5000	0.99980252	0.99980183
Faure 1992	4	2	5000	0.9998014	0.99980089
Offset	4	2	5000	0.99979996	0.99979996
Regular	12	2	5000	4.68872783	2.13327165
Faure 1992	12	2	5000	1.47878792	1.05099389
Offset	12	2	5000	1.40865037	1.03312189
Regular	52	2	5000	18.82439336	4.81624748
Faure 1992	52	2	5000	4.73002376	1.24112783
Offset	52	2	5000	4.62842264	1.21837901

Table 4.5: C_2 values of point sets based on the Halton Sequence with $n = 5000$.

Applying permutations clearly improves the quality of the original construction, especially for larger dimensions. This improvement is captured by the criteria – if we look at the Tables 4.4 and 4.5. In dimensions 12 and 52, both the $c(2, \mathcal{K}_{d,w=s,s}, P_n)$ and $\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$ values are significantly greater for the "regular" Halton sequence compared to using either set of permutations. This means that not only are the worst two-dimensional projections less well distributed, the average two-dimensional projection within the point set is not as well distributed. For the $s = 4$ case, all the two-dimensional projections give a C_b value of less than 1, meaning that they are all well distributed. This is not surprising, as the

points are constructed using smaller prime bases, and thus we need less points for the unit square to be evenly filled.

When comparing Tables 3.9 from the previous chapter and 4.5—which both use $n = 5000$—the Halton sequence and its generalizations appear to be of better quality compared to their respective counterpart for the Faure sequence, with the generalized Halton sequences achieving values for $\bar{c}(2, \mathcal{K}_{d,w=s,s}, P_n)$ that are not much higher than 1. We think this may be happening because the permutations introduce a more substantial "scrambling" than simply multiplying each generating matrix by a fixed factor.

4.6.2 Integration problems

In this section, we numerically investigate the fundamental question of whether it is best to randomize via random scrambling or via a well-chosen generalized construction that can then be randomized using a simple digital shift.

We again use h_0, h_1, g, and the stochastic activity network from Section 2.5. For these functions, we respectively have the true integrands of $\mu(h_0) = 0$, $\mu(h_1) = s/3 + s(s-1)/4$ and $\mu(g) = 1$.

The different constructions and scramblings are compared using two approaches. First in Section 4.6.2 we plot the MSE or variance as a function of n to see how quickly they converge to 0. Then in Section 4.6.2, we fix n and plot a histogram of the MSE or variance constructed from randomly scrambled point sets and look at where different deterministically scrambled point sets compare with this distribution, as approximated by the histogram. This approach is similar to what was done in Section 4.5 to assess deterministic scramblings based on factors.

Convergence results

What is plotted on Figures 4.7 to 4.11 are the MSE for functions h_0, h_1, and g, and the variance for the SAN. In all cases, the MSE or variance is estimated using $V = 25$ randomizations.

We compare the following constructions:

1. Halton sequence, randomized with a digital shift, ("Regular, Shifted");
2. Generalized Halton sequence, using permutations from Algorithm 2, then randomized with a digital shift ("Faure 1992, Shifted");

3. Generalized Halton sequence, using permutations from Algorithm 2 with an offset term added, then randomized with a digital shift ("Offset, Shifted");

4. Halton sequence, randomized with random linear scrambling ("Scrambled").

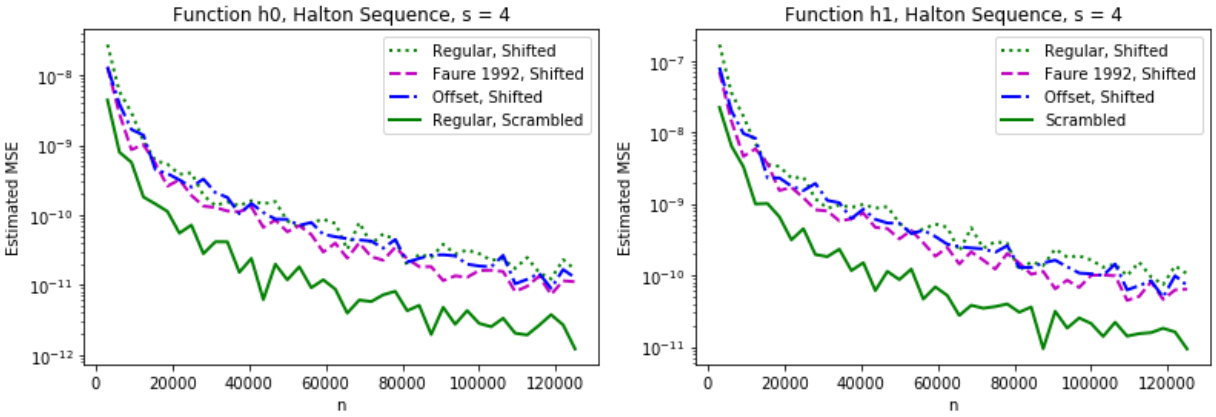


Figure 4.7: Estimated MSEs of test functions at $n \in \{3125m, 1 \leq m \leq 40\}$, using the 4-dimensional Halton Sequence.

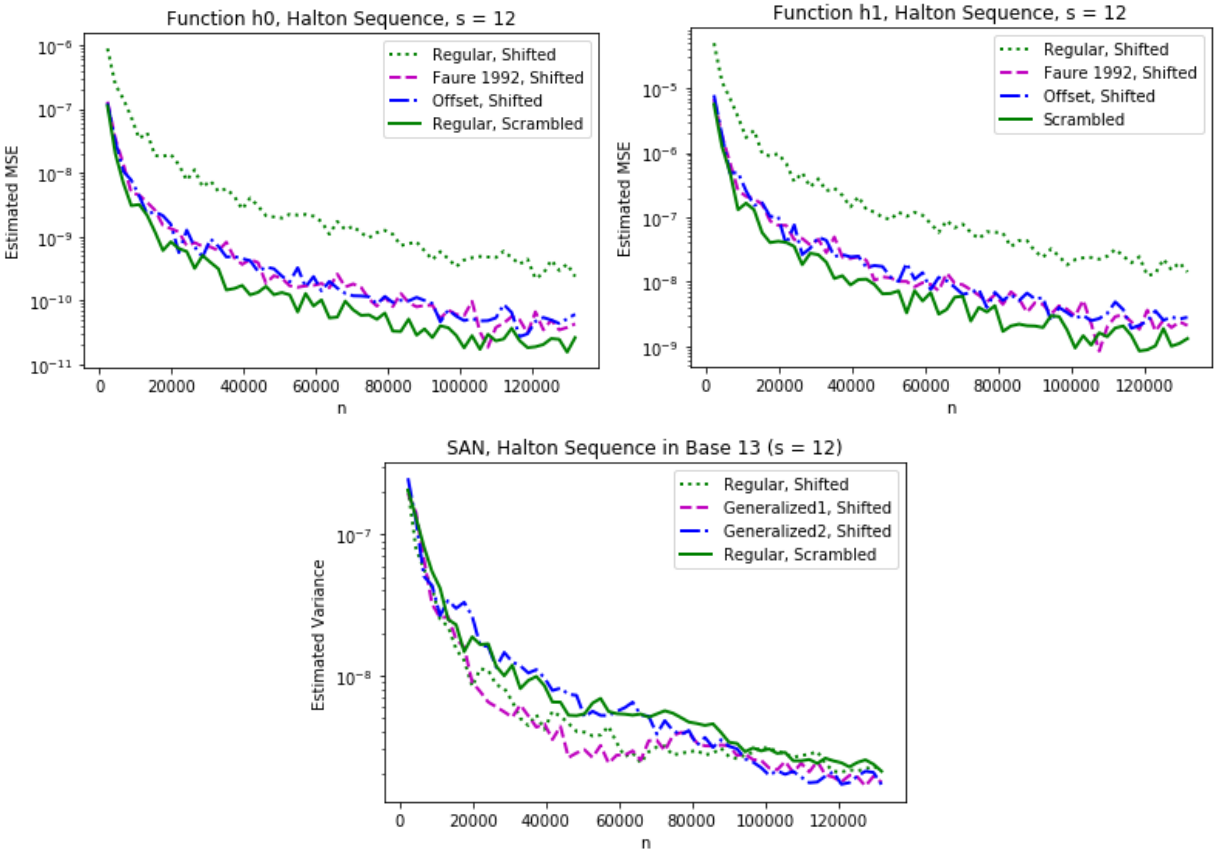


Figure 4.8: Estimated MSEs and Variances at $n \in \{2197m, 1 \leq m \leq 60\}$ of test functions using the 12-dimensional Halton Sequence.

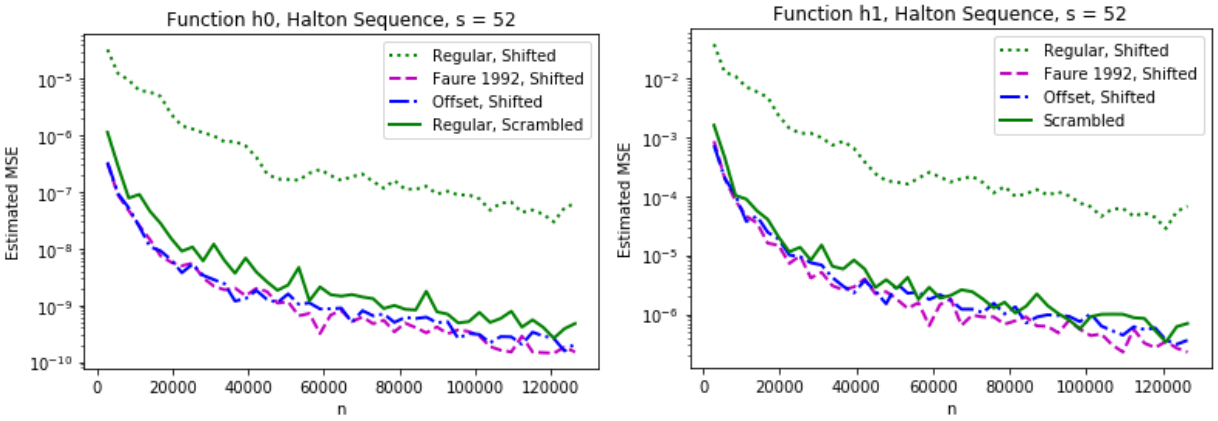


Figure 4.9: Estimated MSEs of test functions at $n \in \{2809m, 1 \leq m \leq 45\}$, using the 52-dimensional Halton Sequence.

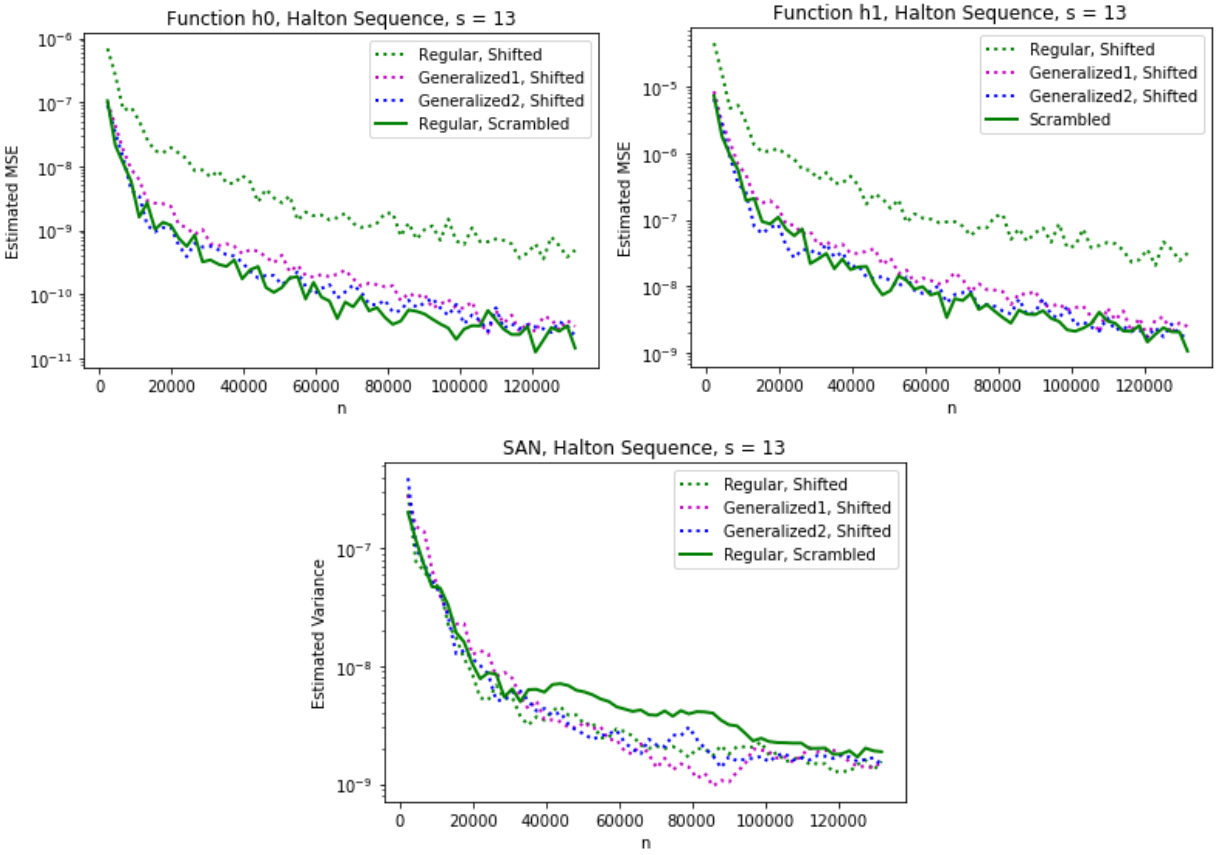


Figure 4.10: Estimated MSEs and Variances of test functions using the 13-dimensional Halton sequence.

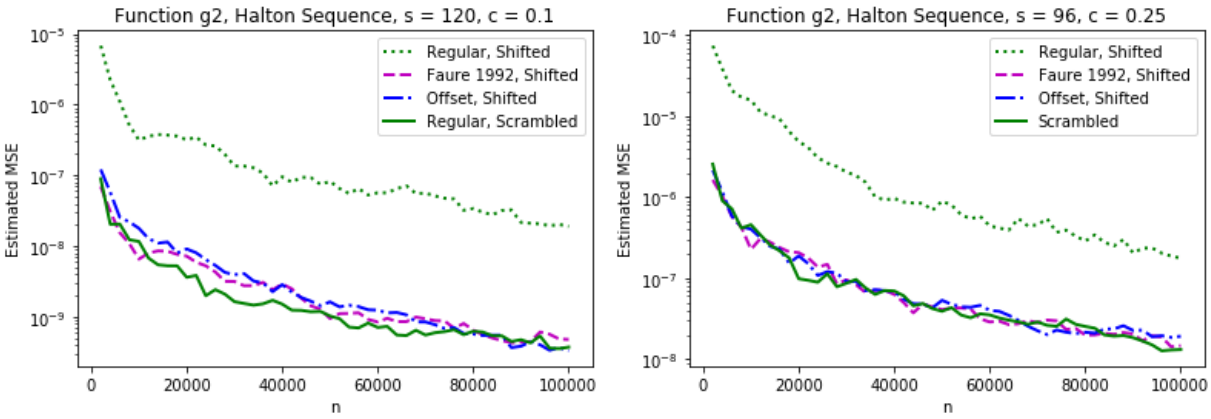


Figure 4.11: Estimated MSEs at $n \in \{2000m, 1 \leq m \leq 50\}$, when integrating Test Function g using Halton and Faure Sequences.

The majority of the results show that scrambling is superior to generalizing then randomizing with a digital shift. They also show that a point set that is well distributed over the unit cube performs much better when randomized with a shift compared to a point set with poor distribution properties.

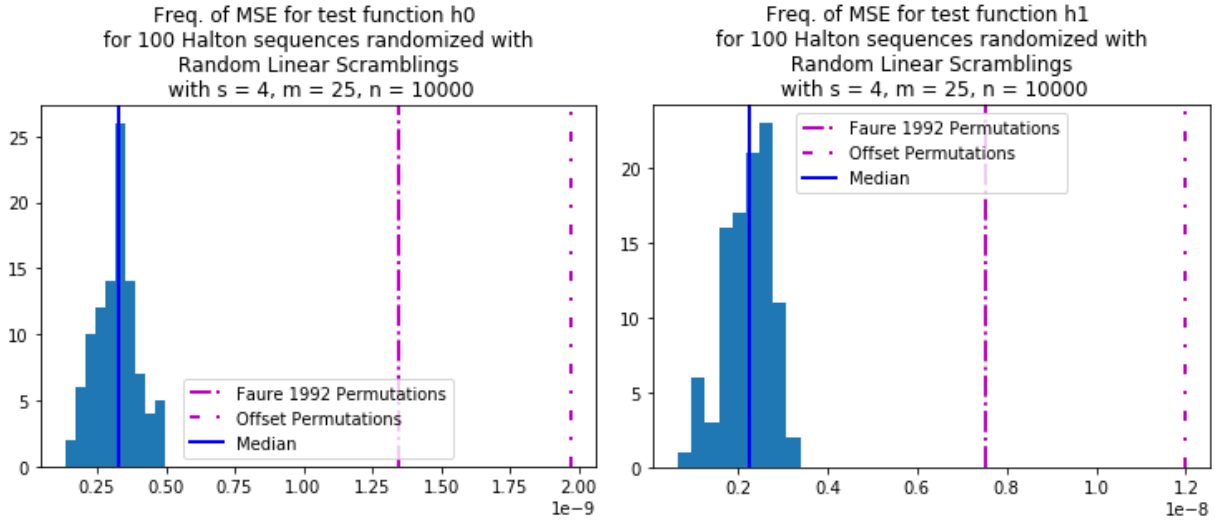
The results from this numerical experiment can also be somewhat explained by the criterion values in Tables 4.4 and 4.5. If the c values are high, that suggests a "bad" point set that cannot be fixed via a digital shift. However, even a "bad" point set can be fixed with scrambling. The "Regular" Halton sequence, which had the highest c values, had the worst performance by far when randomized with a digital shift. The "Faure 1992" and "Offset" sequences, which had lower c values, had RQMC errors almost as good as the scrambled Halton sequence, even when randomized with a digital shift.

Comparing results using histograms

As done in Sections 3.2.5 and 4.5.1, we use histograms of the integration error of the functions h_0 and h_1 to compare the use of deterministic permutations versus scrambling for the Halton sequence. We consider 3 values of s: 4, 12, and 52. The reported error is the Mean Squared Error (MSE), and is estimated using $V = 25$ replications and a sample size of $n = 10000$, except in the $s = 52$ case where $n \in \{2809, 10000\}$.

For the Halton sequence, we compare the distribution of randomly scrambled point sets (as shown by the histogram) with the same types of deterministic permutations as in Section 4.6.2.

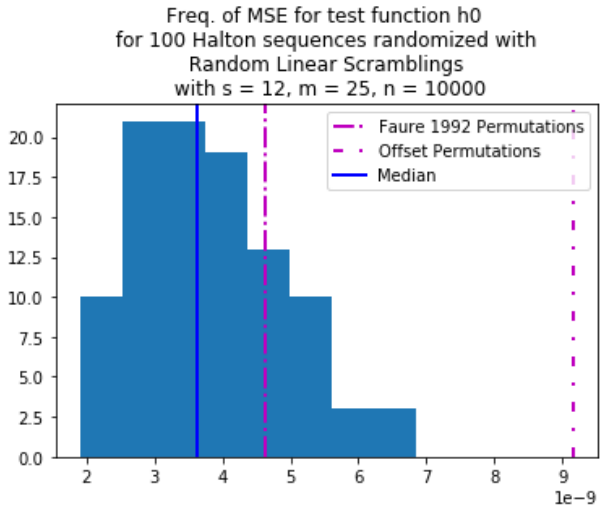
Some values (namely "Regular Shifted" and Monte Carlo) obtained are sometimes quite large compared to the rest of the results. Therefore, to better visualize the results, these large values have been excluded from the plots and their error values are reported in the caption instead.



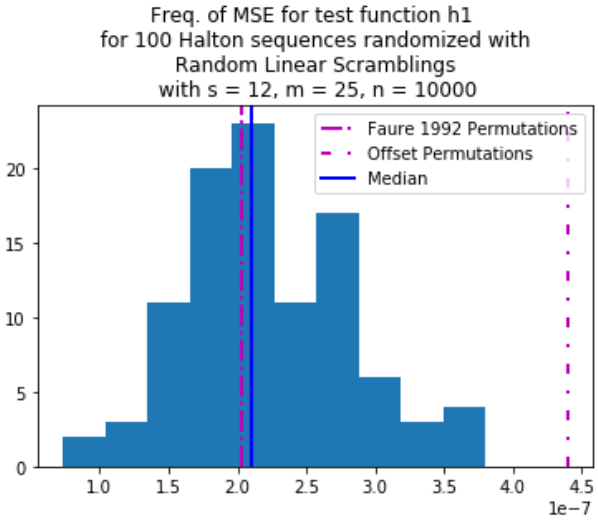
(a) Halton: 1.8421e-09, MC: 3.8787e-06

(b) Halton: 8.4177e-09, MC: 2.2075e-05

Figure 4.12: Comparing Random Linear Scrambling of the Halton sequence with other Halton sequences with $s = 4$

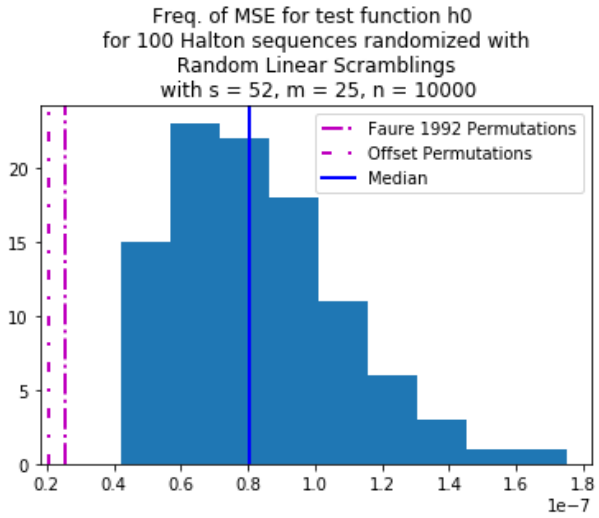


(a) Halton: 2.979e-08, MC: 1.1629e-05

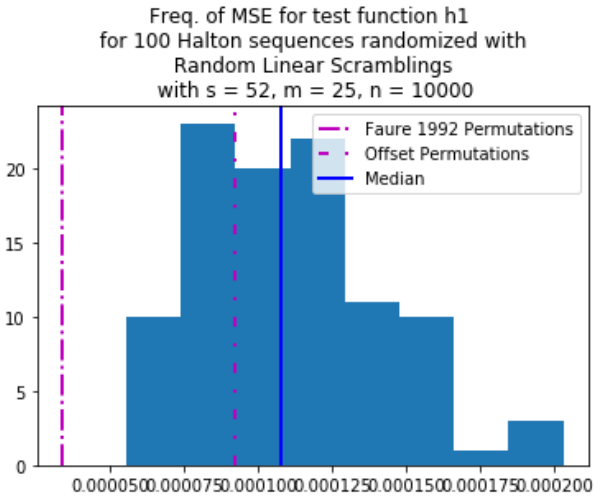


(b) Halton: 1.541e-06, MC: 0.000584

Figure 4.13: Comparing Random Linear Scrambling of the Halton sequence with other Halton sequences with $s = 12$



(a) Halton: 7.02e-06, MC: 5.024e-05



(b) Halton: 0.006982, MC: 0.04699

Figure 4.14: Comparing Random Linear Scrambling of the Halton sequence with other Halton sequences with $s = 52$

Just like the results in the previous sections of this chapter, in general, random linear scrambling is better than using well-chosen permutations. This suggests that well-chosen permutations have the potential to do better than scrambling, but probably need to be chosen based on the type of function being integrated. Hence, if one is looking for a multipurpose construction, scrambling appears as a better choice.

4.7 Conclusion

We showed that a Halton point set randomized with a base **b**-digital scramble is an NLOD sampling scheme. This result was established by defining and carefully studying the $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values, which are an extension of the $C_b(\boldsymbol{k}; P_n)$ values introduced in [90].

We established that random linear scrambling of Halton sequences satisfies the properties of being a base **b**-digital scramble and investigated its performance via numerical results on different integration problems. These results further motivate randomizing the Halton sequence with this type of scrambling, which has not been done in practice yet. In addition, unlike fixing a permutation, it does not require any data to be stored.

We also used a criterion based on the $C_{\mathbf{b}}(\boldsymbol{k}; P_n)$ values to search for permutations to generalize the Halton sequence. From our numerical results on different integration problems, our searched permutations give variance results comparable to those of other currently used generalized Halton sequence constructions. However, neither appears to consistently outperform random linear scrambling, which reinforces our assessment that this randomization offers promising outcomes in the quest to make the Halton sequence a robust choice for numerical integration.

Extending on our work on assessing different forms of scrambling, we evaluated two choices for permutations used to generalize the Halton sequence based on the ones from [25]. We evaluated their performance using a numerical study as well as studying the $C_b(\boldsymbol{k}, P_n)$ values. While these factors can break up some poor projection properties of these point sets, they do not generally outperform using scrambling to randomize the point.

An interesting extension of this work is that our main theoretical result could easily be proved for constructions based on juxtaposed van der Corput sequences from which consecutive points are extracted at different starting points, leading to additional randomization methods and modifications to the sequence. We plan to explore this idea in future work.

Chapter 5

A randomized implementation of the triangular van der Corput sequence

This chapter details joint work between myself, Erik Hintz, Christiane Lemieux, and Marius Hofert. This work has not been published yet and is the topic of the working paper [21]. My contribution to this topic includes the work on implementing the scrambled triangular van der Corput sequence as nested scrambling, i.e., Section 5.2 as well as the numerical experiments in Section 5.3. The working paper also includes the development of an extensible lattice construction on the triangle, but is omitted from this thesis, as the majority of the work on that topic was done by Erik Hintz.

In the previous chapters of this thesis, the constructions for low-discrepancy sequences discussed were designed for the unit hypercube $[0,1)^s$. Here, we consider constructions of point sets on the triangle in two dimensions, which has applications in, among other things, computer graphics [7, 68].

Basu and Owen [7] have proposed two low-discrepancy constructions with vanishing discrepancy for the triangle that construct points directly on the triangle, rather than mapping points from the unit cube to the triangle, such as in [2, 23, 69]. The first is based on a finite lattice. This construction attains a discrepancy of $O(\log(n)/n)$, which is the best possible rate [9] and work prior to this have only indicated that such a discrepancy was possible without providing a construction. The other is a triangular van der Corput sequence based on the one-dimensional van der Corput sequence in base 4 that places points in a two-dimensional triangle by recursively subdividing the triangle. The triangular van der Corput sequence can also be generalized by replacing each digit from a base 4 sequence with a sequence of two digits from a base 2 sequence, but using the same subdivisions [33].

Although various constructions and mappings for sampling on the triangle exist, there are very few, if any, work done comparing these different methods on numerical integration examples. We also perform numerical examples on the triangle that allow us to differentiate between the performance of these constructions on some integration problems.

A shortcoming of the deterministic triangular van der Corput sequence is that its one-dimensional projections over some of the sides of the triangle have the undesirable property of collapsing onto a few non-unique points. This is evident in Figure 5.1, which shows the first 1000 points of the triangular van der Corput sequence on the right-angle triangle. The points form vertical and horizontal lines that have the same x_1 or x_2 values.

One of the main contributions of this chapter is that we address the issue of having non-unique points in the one-dimensional projections. As mentioned in the previous chapters of this thesis, poor projection qualities of a point set can be fixed by applying a randomization to the point set. Since the triangular van der Corput sequence is based on the van der Corput sequence, a natural randomization to use is scrambling. In the triangular van der Corput sequence, the sampling scheme subdivides the triangle into a finer and finer partition of triangles until each sub-triangle gets at most one point, and the van der Corput sequence decides the order that the points are placed into the triangles, just as it does normally with the b-adic intervals over the unit interval. When we randomize the triangular van der Corput sequence, this is akin to randomizing the one-dimensional van der Corput sequence with randomizing before mapping it to the triangle. In one dimension, we will show how scrambling is very closely related to stratification by constructing an extensible stratified sampling scheme on the triangle that avoids the poor projection properties. This stratified sampling scheme is still based on the triangular van der Corput sequence – the strata we use are the sub-triangles used in each recursive step of constructing the triangular van der Corput sequence. Figure 5.2 on page 113 shows the subdivision of the equilateral and right-angle triangles into 4 such strata, and Figure 5.4 on page 131 shows the subdivision of the equilateral triangle into 16 strata.

We also show that, in general and not only on triangles, the stratified sampling scheme has the same distribution as the nested scrambling of the van der Corput sequence, while being more efficient to implement. This connection between stratified sampling and nested scrambling allows us to have the benefits of nested scrambling without the expensive computational costs. On the flip side, we also show that nested scrambling is a way to implement an extensible stratified estimator based on a stochastic but balanced allocation.

We begin this chapter in Section 5.1 by introducing notation and describing common methods used to sample on the triangle. We then explain how to construct points sets using the triangular van der Corput sequence of Basu and Owen, and then propose our

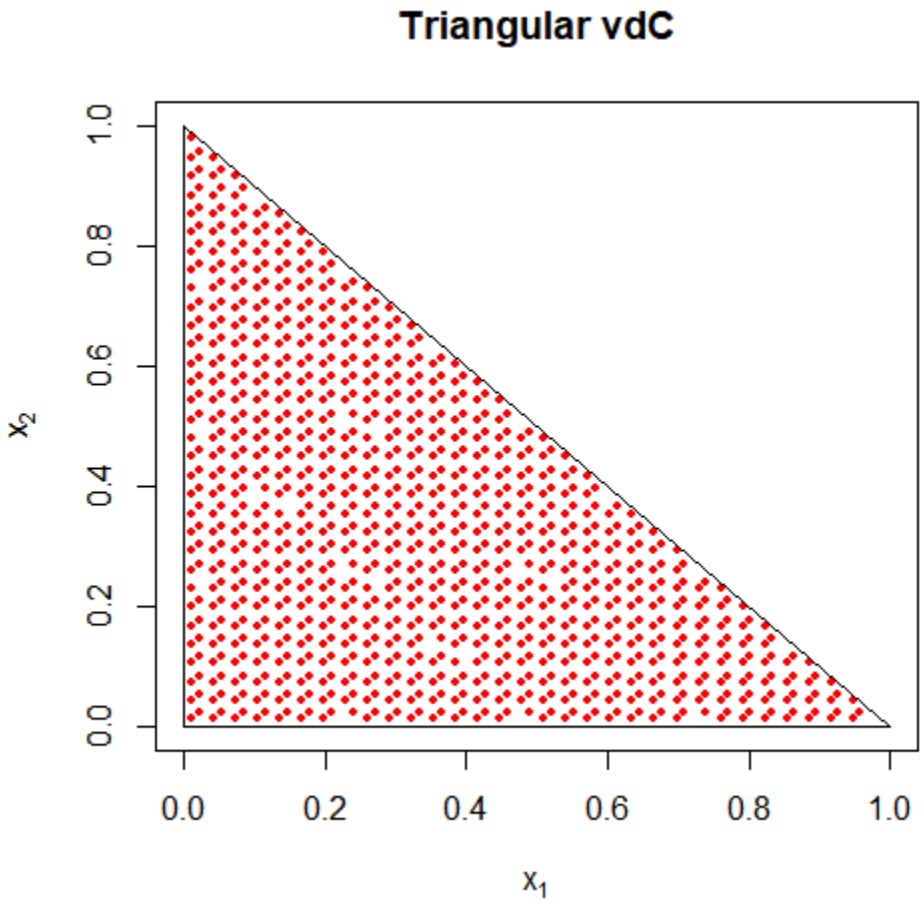


Figure 5.1: The first 1000 points of the triangular van der Corput (vdC) sequence of Basu and Owen

stratified sampling design in Section 5.2 and show that it is an efficient equivalent to nested scrambling using the tools from negative dependence concepts from the earlier chapters of this thesis, and show how to then map the resulting one-dimensional point set to the two-dimensional triangle. Finally, in Section 5.3, we conduct a numerical study to compare different constructions on the two-dimensional triangle.

5.1 Background

In this section, we review some definitions and properties of triangles and give some common methods to sample on the triangle, including methods that both sample directly on the triangle, as well as methods that transform point sets from the unit square to the triangle.

In the general case, let points A, B, C lie on a hyperplane in R^d, forming a non-degenerate triangle, i.e., not lying on the same line. Define the triangle with corners A, B and C as

$$\triangle(A, B, C) = \left\{ \lambda_1 A + \lambda_2 B + \lambda_3 C \mid \min\{\lambda_j\} \geq 0, \sum_{j=1}^{3} \lambda_j = 1 \right\}.$$

Without loss of generality, we can consider $A, B, C \in \mathbb{R}^2$ for this chapter. We often construct point-sets on special triangles, such as the equilateral triangle,

$$\triangle_E = \triangle\left((0,0), (1,0), (1/2, \sqrt{3}/2)\right),$$

or the right-angle triangle,

$$\triangle_R = \triangle\left((0,0), (0,1), (1,0)\right),$$

as it is simpler than constructing point sets on an arbitrary triangle. It is thus useful to be able to map a point set constructed on one triangle to any other arbitrary triangle.

Indeed, we can use an affine transformation which preserves the ratios of the lengths of parallel line segments and ratios of distances between points lying on a straight line. When mapping to a non-degenerate triangle $\triangle'$, this transformation is one-to-one.

Let $\triangle = \triangle(A, B, C)$ and $\triangle' = \triangle(A', B', C')$ be two arbitrary triangles. Algorithm 4 explains how to transform a point $\mathbf{x} = (x, y) \in \triangle$ to $\mathbf{x}' = (x', y') \in \triangle'$.

This transformation would be applied to every point within $\triangle$ to create a sampling scheme on $\triangle'$ [82].

Algorithm 4: Transforming a point from $\triangle$ to $\triangle'$.

Given $\triangle = \triangle(A, B, C)$, $\triangle' = \triangle(A', B', C')$, and a point $\mathbf{x} = (x, y) \in \triangle$:

1. Define the matrices

$$M(\triangle) = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ 1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad M(\triangle') = \begin{pmatrix} a'_1 & b'_1 & c'_1 \\ a'_2 & b'_2 & c'_2 \\ 1 & 1 & 1 \end{pmatrix},$$

   where $A = (a_1, a_2)$, $B = (b_1, b_2)$, $C = (c_1, c_2)$, $A' = (a'_1, a'_2)$, $B' = (b'_1, b'_2)$, and $C' = (c'_1, c'_2)$.

2. Let $M(\triangle, \triangle') = M(\triangle')M(\triangle)^{-1}$ be our affine transformation matrix.

3. The point $\mathbf{x}' = (x', y')$ are the first two components of the matrix-vector product

$$(x', y', z') = M(\triangle, \triangle') \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix}. \tag{5.1}$$

4. Return $\mathbf{x}'$.

We illustrate the affine transformation in Figure 5.2. The first four points of the scrambled triangular van der Corput sequence are generated on the equilateral triangle $\triangle_E$, and then mapped using Algorithm 4 to the right-angle triangle $\triangle_R$. Clearly, the ratios of distances between points are preserved, as are the low-discrepancy properties of the point set.

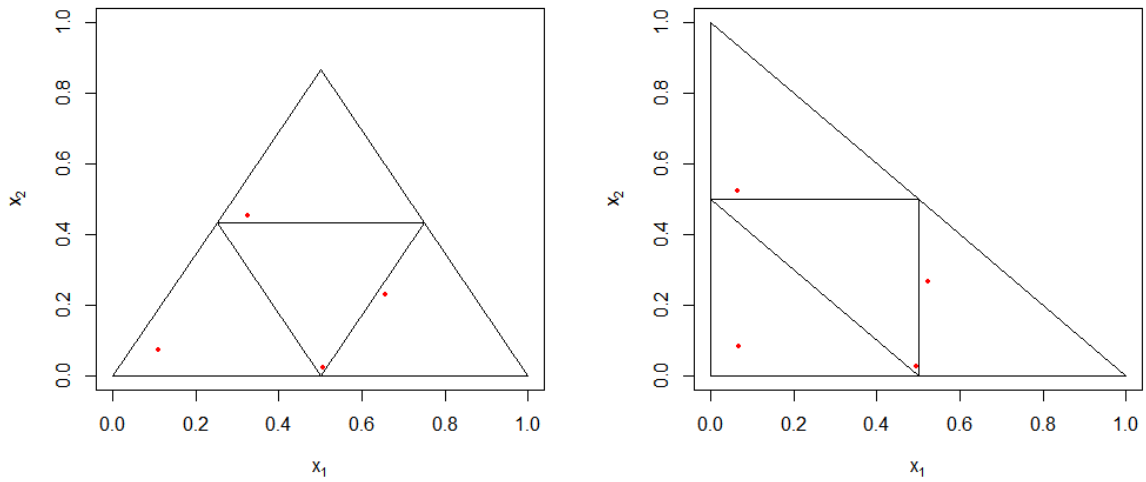


Figure 5.2: 4 points generated on the equilateral triangle and mapped to the right-angle triangle using Algorithm 4.

5.1.1 Creating point sets on the triangle

Transforming a low-discrepancy point-set from $[0,1)^2$ to $\triangle(A,B,C)$.

The end goal of creating point sets over some domain Ω is to estimate an integral over said domain, say, $\int_\Omega f(\mathbf{x})\, d\mathbf{x}$, where $f : \Omega \to \mathbb{R}$ is integrable. If constructing point sets on Ω directly is difficult or not possible, a natural and popular approach is to sample points from the unit hypercube $U(0,1)^d$ and formulate this integral as

$$\mu = \int_\Omega f(\mathbf{x})\, d\mathbf{x} = \lambda(\Omega) \int_{[0,1)^d} f(\phi(\mathbf{u}))\, d\mathbf{u},$$

where λ denotes the Lebesgue measure and $\phi : [0,1)^d \to \Omega$ is a mapping such that $\phi(U) \sim U(\Omega)$ for $U \sim U(0,1)^d$. That is, we can sample points from the domain Ω if we can find such a mapping ϕ.

For $\Omega = \triangle$ the right-angle triangle with $0 \le x_1 \le x_2 \le 1$, [69] give six possible transformations ϕ to map points from $U[0,1)^2$ to uniformly over $\triangle$. Note that this right

angle triangle is not the same as $\triangle_R$, but as mentioned, transforming points generated on one triangle to another triangle is very straightforward.

1. Transformation *Drop*. Points are kept if they are within $\triangle$ and dropped if not. This transformation is fast, but in higher dimensions a lot of points get lost - only 1 in every $s!$ points are kept when working in s dimensions.

2. Transformation *Sort*. Points lost by transformation drop are recovered by reordering the coordinates of a point in the unit square such that $x_1 \leq x_2$ to be within $\triangle$. This transformation is fast and continuous.

3. Transformation *Mirror*. We leave the points that are already in $\triangle$ and reflect the other ones at $(1/2, 1/2)$. The resulting transformation is fast, but discontinuous.

4. Transformation *Origami*. Here, Transformation sort is recursively used within the unit square. Starting by subdividing the unit square into b^{2m} subsquares, where b and m are user-chosen integer values, with each iteration increasing the side length of the subsquare by a factor of b, until it is used on the unit square. This transformation is discontinuous.

5. Transformation *Root*. This transformation is based on [23] and given by

$$\phi(u_1, u_2) = (u_1\sqrt{u_2}, \sqrt{u_2}).$$

   This transformation is continuous and smooth, but the two sharp corners of the triangle are treated in different ways.

6. Transformation *Shift*. For each point, draw a straight line with slope -1 through the point, and then the point is moved halfway on this line towards the closest axis. This results in

$$\phi(u_1, u_2) = \begin{cases} (u_1 - (1-u_1)/2, u_2 - (1-u_2)/2) \text{ if } u_1 \geq 1 - u_2, \\ (u_1 - u_1/2, u_2 + (u_2/2) \text{ otherwise.} \end{cases}$$

   This transformation is fast and continuous, but no generalization to higher dimensions exists.

In our numerical experiments later in this chapter, we use the transformation "root" on the bivariate Sobol' sequence as well as on pseudo-randomly generated numbers. We use this choice of transformation as it is smooth, continuous, fast and can be extended to go from higher-dimension cubes to simplexes.

Inverse Rosenblatt transformation

To sample from any multivariate distribution, we can use the inverse Rosenblatt transform [73]. In order to sample $(U_1, U_2) \sim U(\triangle_E)$, we first sample U_1 and then $U_2 \mid U_1$. The algorithm for sampling from $\triangle_E$ is given in Algorithm 5.

Algorithm 5: Inverse Rosenblatt transformation to sample from $U(\triangle_E)$

1. Sample $V_1, V_2 \sim U(0, 1)$.
2. Set
$$U_1 = \begin{cases} \sqrt{V_1/2}, & \text{if } V_1 \leq 1/2, \\ 1 - \sqrt{(1 - V_1)/2}, & \text{otherwise.} \end{cases}$$
3. Set
$$U_2 = \begin{cases} V_2 U_1 \tan \pi/3, & \text{if } U_1 \leq 1/2, \\ V_2 (1 - U_1) \tan \pi/3, & \text{otherwise.} \end{cases}$$
4. Return (U_1, U_2).

The algorithm described here is used in our implementation of the randomized triangular van der Corput sequence, which will be explained later in this chapter, in Algorithm 10.

Lattice constructions

Basu and Owen [7] give a rank-2 lattice construction for points on the triangle. The process to sample n points is described in Algorithm 6.

Algorithm 6: Rank-2 Triangular Lattice of Basu and Owen [7]

Assume a sample size n, an angle α such that $\tan(\alpha)$ is a quadratic irrational number, i.e., $\tan(\alpha) = (a + b\sqrt{c})/d$ for $b, d \neq 0$ and $c > 0$ not a perfect square, and thus badly approximable (e.g., $\alpha = 3\pi/8$), and an optional random vector $U \sim U(-0.5, 0.5)^2$ for a random shift, which transforms the otherwise deterministic points so that $\mathbf{x} \sim U(\triangle_R)$ for all $\mathbf{x} \in P$. The point-set P_n is then obtained by rotating the lattice counterclockwise by α and intersecting with $\triangle_R$, as follows:

1. $N \leftarrow \lceil \sqrt{2n} \rceil + 1$
2. Let $P = \{-N, \ldots, N\}^2$ and set $\mathbf{x} \leftarrow (\mathbf{x} + U)$ for $\mathbf{x} \in P$.
3. Set $\mathbf{x} \leftarrow \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \mathbf{x}/\sqrt{2n}$ for all $\mathbf{x} \in P$.
4. Set $P_n = P \cap \triangle_R$.
5. If $|P_n| \neq n$, add or remove $|P_n| - n$ points in $\triangle_R$ to P_n. In this step, up to $O(log(n))$ arbitrarily chosen points can be added or removed without worsening the discrepancy.
6. Return P_n.

In our numerical experiments, we also consider a rank-1 lattice with generating vector $\mathbf{z} = (1, 182667)$ [16]. The idea behind this rank-1 lattice is explained in more detail in the working paper [21].

An issue with this construction is that it is not extensible, that is, if the n points are generated and a larger sample size of $n_1 > n$ points is required, a new point set of size n_1 must be generated, rather than just the $n_1 - n$ additional points. Creating an extensible triangular lattice based on the van der Corput sequence to mitigate this is possible [38].

5.2 Stratified sampling based on the triangular van der Corput sequence

5.2.1 The triangular van der Corput sequence of Basu and Owen

Recall from Section 2.2.1 that the i^{th} point of the one-dimensional van der Corput sequence in base b is given by $u_i = \phi_b(i-1)$ where the radical inverse function ϕ_b is defined as

$$\phi_b(i) = \sum_{k\geq 0} d_k b^{-k-1}, \quad i = \sum_{k\geq 0} d_k b^k \in \{0, 1, \dots\}.$$

With this formula, the van der Corput sequence places points at the left-most boundaries of each of the intervals $[b^{-m}, b^{-m+1})$. Similarly, the triangular van der Corput sequence of Basu and Owen [7], which is based on the van der Corput sequence in base 4, replaces the intervals with $b^m = 4^m$ congruent sub-triangles and places the points in the centre of each terminal sub-triangle. Specifically, let $T = \triangle(A, B, C)$ be the triangle which we wish to generate points on. Define the sub-triangle of T with index d for $d \in \{0, 1, 2, 3\}$ as

$$T(d) = \begin{cases} \triangle\left(\frac{B+C}{2}, \frac{A+C}{2}, \frac{A+B}{2}\right), & d = 0, \\ \triangle\left(A, \frac{A+B}{2}, \frac{A+C}{2}\right), & d = 1, \\ \triangle\left(\frac{B+A}{2}, B, \frac{B+C}{2}\right), & d = 2, \\ \triangle\left(\frac{C+A}{2}, \frac{C+B}{2}, C\right), & d = 3. \end{cases} \tag{5.2}$$

For the i^{th} point in the sequence, write the base 4 representation of $i \geq 0$ as $i = \sum_{k\geq 0} d_k 4^k$. This representation has at most $K_i = \lceil \log_4(i) + 1 \rceil$ non-zero digits, which means that the expansion is finite, and we do not have to infinitely divide the triangle into sub-triangles. The original construction for the triangular van der Corput sequence in [7] obtains the i^{th} triangular point by mapping the integer i to the midpoint of the triangle $T(d_0, \dots, d_{K_i})$, which is recursively defined by $T(d_k, d_{k+1}) = T(d_k)(d_{k+1})$, where the midpoint function $\text{mdpt}(\triangle(\text{A}+\text{B}+\text{C})) = (\text{A}+\text{B}+\text{C})/3$ component-wise, as detailed in Algorithm 7. For example, if we have $T = \triangle_R = \triangle((0,0),(0,1),(1,0))$, then $T(2,3) = T(2)(3) = T'(3)$, where $T' = \triangle((0,0.5),(0,1),(0.5,0.5))$. Then, $T'(3) = \triangle((0.25,0.5),(0.25,0.75),(0.5,0.5))$.

Algorithm 7: Algorithm for generating points from the triangular van der Corput sequence

Given input $n \geq 1$ and target triangle $\triangle(A, B, C)$, generate the first n points, outputted in an $n \times 2$ array $\mathbf{x}$, as follows:

1. For $i = 1, \ldots, n$:
   (a) Compute $(d_0, \ldots, d_{K_i})$ such that $i - 1 = \sum_{k=0}^{K_i} d_k 4^k$;
   (b) Initialize $T = \triangle(A, B, C)$;
   (c) For $j = 0, \ldots, K_i$,
   Update $T = T(d_j)$ using (5.2);
   (d) Set $\mathbf{x}[i] = \text{mdpt(T)}$;
2. Return $\mathbf{x}$.

In other words, Algorithm 7 generates a van der Corput sequence in base 4, and each digit of the base 4 expansion denotes which sub-triangle the point lies in. Advantages of this method include that since it is based on the van der Corput sequence, it is extensible, balanced, can be modified to be randomized, and it is easily implemented.

However, when implementing the algorithm as originally described, once the terminal sub-triangles are identified, placing the sampling points at the centre of each results in points that suffer from poor projection properties. For example, consider the right-angle triangle. If $T = \triangle((0,0),(0,1),(1,0))$, such as in Figure 5.1, and P_n is the point-set consisting of the first n points produced by Algorithm 7 for some $n = 4^k$ and $k > 2$, then the projections of P_n onto either of the two axes contain $2\sqrt{n} = 2^{k+1} < n$ points [38]. That is, the one-dimensional projections have non-unique points, which will lead to poorer integration results, especially for functions with low effective dimension where the majority of the variance of the function is captured by one-dimensional projections. This behaviour where the one-dimensional projections contain non-unique points can also be observed in the equilateral triangle, as seen in Figure 5.4.

To improve the one-dimensional projections of the sequence, we propose a modification to the triangular van der Corput sequence, where we will use the terminal sub-triangles identified by the method to design a stratified sampling scheme.

We first explain a connection between stratified sampling and the nested scrambling of Owen [60] in the one-dimensional case. This will allow us to then propose an efficient implementation of nested scrambling that highlights this connection with stratified sam-

pling. More precisely, we show that nested scrambling is a way to implement an extensible stratified estimator based on a stochastic but balanced allocation. Since the triangular van der Corput sequence is a mapping from the one-dimensional van der Corput sequence in base 4 to the two-dimensional space, it is straightforward to transpose our proposed scrambling implementation in one dimension to a randomization method for the triangular van der Corput sequence.

We now describe how to implement the scrambled van der Corput sequence as stratified sampling, and then map this sequence to a triangle.

5.2.2 Stratified sampling

A simple way to reduce the variance of an estimator, instead of sampling randomly throughout the unit interval $[0, 1)$, is to stratify the unit interval into equally sized subintervals of length b^{-m} for some positive integers b and m. If we allocate an equal number of points uniformly and independently into each of these subcubes, then this is stratified sampling with proportional allocation by area, which is guaranteed to have a lower variance than regular Monte Carlo [47, p. 126]. That is, the RQMC estimator in this case would be equivalent to the stratified sampling estimator. If n is the total number of points, then we have b^m strata of size b^{-m} each with an equal number of points (nb^{-m} points) allocated in each stratum, then the stratified sampling estimator, $\hat{\mu}_{stratified}$, when estimating a function h, is

$$\begin{aligned}
\hat{\mu}_{stratified} &= \sum_{j=1}^{b^m} b^{-m} \sum_{i=1}^{nb^{-m}} \frac{1}{nb^{-m}} h(u_{i,j}) \\
&= \sum_{j=1}^{b^m} \sum_{i=1}^{nb^{-m}} \frac{1}{n} h(u_{i,j}) \\
&= \frac{1}{n} \sum_{j=1}^{b^m} \sum_{i=1}^{nb^{-m}} h(u_{i,j}) \\
&= \frac{1}{n} \sum_{l=1}^{n} h(u_l),
\end{aligned}$$

which is equivalent to the RQMC estimator with the aforementioned point set. The variance of this estimator is $\mathrm{Var}(\hat{\mu}_{stratified}) = \frac{1}{n} \sum_{j=1}^{b^m} b^{-m} \sigma_j^2 = \frac{1}{nb^m} \sum_{j=1}^{b^m} \sigma_j^2$, where σ_j^2 is the variance within the j^{th} subinterval.

In the context of this section, since the number of points is not necessarily an integer power of b, we refer to a *base b stratified sampling scheme* as a sampling scheme where for any number of points n, if we subdivide the unit interval into subintervals with length b^{-m} for any positive integer m, the number of points within each subcube is different by at most one from the number of points within any other subinterval. This gives us a connection with stratified sampling to give a more efficient implementation of nested scrambling, which we will explain in the next section.

5.2.3 Nested scrambling as stratified sampling

We now show that we can implement a nested scrambled van der Corput sequence as stratified sampling on the unit interval, as the two methods produce point sets with the same joint distribution.

First, we show that the scrambled van der Corput sequence in base b is an $(0,1)-$sequence in base b. Since scrambling does not change the equidistribution properties of a point set or sequence, it is sufficient to show that the deterministic van der Corput sequence is a $(0,1)-$sequence.

Lemma 5.2.1. *Any point set P_n that is a subsequence of $n = b^m$ consecutive points from a van der Corput sequence constructed in base b is an $(0, m, 1) - net$ in base b.*

Proof. If we consider the first m digits in the base b expansion of each point $u_i \in P_n$, there is exactly one point with each of the unique b^m combinations of digits. That is, there is exactly one point in each of the m-elementary intervals and thus, by definition, an $(0, m, 1) - net$ in base b and thus P_n is m-equidistributed in base b. □

Lemma 5.2.2. *Any point set P_n that is a subsequence of n consecutive points from a van der Corput sequence constructed in base b can be viewed as the first n points of a $(0,1)$-sequence.*

Proof. A $(0,1)$-sequence in base b is a sequence of points for which each b-ary segment of the form $u_{lb^k}, \ldots, u_{(l+1)b^k}$ with $k \geq 0$ and $l \geq 0$ is a $(0, m, 1)$-net in base b. Since every point set P_n that is a subsequence of $n = b^m$ consecutive points from a van der Corput sequence constructed in base b is an $(0, m, 1) - net$ in base b, the said b-ary segment is a $(0, m, 1)$-net and the result follows. □

Lemma 5.2.3. *Let $\hat{\mu}_{scr,n} = \frac{1}{n}\sum_{i=1}^{n} f(\tilde{u}_i)$ be the estimator where $\tilde{u}_i$, $i = 1 \ldots n$ be the first n points of a scrambled van der Corput sequence in base b. If $n = b^m$ for some positive integer*

m, then $\hat{\mu}_{scr,n}$ corresponds to the estimator that uses stratified sampling with proportional allocation.

Proof. Since the scrambled van der Corput sequence with exactly b^m points is a $(0, m, s)-$net in base b, its equidistribution properties mean that there is exactly one point in each of the b^m subintervals, and the scrambling as defined in 2.4.4 has the point placed independently and uniformly within each of the b^m subintervals. This is exactly stratified sampling with proportional allocation. □

However, in general, n is not a power of b, in which case the number of points in the intervals $[jb^{-l}, (j+1)b^{-l})$ are different by at most one from each other for $l \leq m$ for $m = \lfloor \log_b n \rfloor$.This is true for both our stratified sampling scheme by definition, as well as the van der Corput sequence, by property of the $(0, 1)-$sequence.

We also introduce the following notation to help define our stratified sampling algorithm:

$$n = \lambda b^m + r, \quad \lambda = \lfloor n/b^m \rfloor,\ 0 \leq r < b^m, \quad q = \lceil \log_b n \rceil, \quad M = b^q, \tag{5.3}$$

and $r = kb + j$, where $0 \leq j < b$.

We show formally in Proposition 5.2.4 that we can write $\hat{\mu}_{scr,n}$ for any sample size n as a stratified sampling estimator over strata of the form $[jb^{-l}, (j+1)b^{-l})$ for $l = 1, \ldots, m$ that is based on a stochastic allocation $N_1, \ldots, N_M$, where N_j is the number of points in $[jb^{-q}, (j+1)b^{-q})$. That is, N_j counts the number of points (either 0 or 1) in the smallest meaningful stratum – strata of size b^{-q} or smaller can have at most 1 point, thus there is no reason to subdivide further.

The scrambled van der Corput sequence satisfies the following two properties:

1. $N_j \in \{0, 1\}$, $j = 1, \ldots, M$, $\sum_{j=1}^{M} N_j = n$, and
2. The N_j's have the same marginal distribution.

To define the scrambled estimator, we simply need to generate the vector of N_j's with the properties listed above. Rather than obtaining these N_j via scrambling, we propose a more efficient algorithm to sample a vector of N_j's given a base b and a number of points n in Algorithm 8.

Algorithm 8: Sample $N_1, \ldots, N_M$ to compute $\hat{\mu}_{scr,n}$.

Given input $n \in \mathbb{N}$ and $b \geq 2$, sample $N_1, \ldots, N_M$ as follows:

1. Initialize n and b. Calculate the constants $q, m, M, \lambda, r, k, j$ as in (5.3).
2. If $M = n$, return $N_j = 1$ for $j = 1, \ldots, M$.
3. If $m = 0$, then $n = \lambda$, and $M = b$. Randomly choose λ of the b N_js to be 1 and $b - \lambda$ of the N_j's to be 0, and return N_j for $j = 1, \ldots, M$.
4. If $m > 0$, then we recursively generate the N_j's in each of the b sub-intervals of the form $[jb, (j+1)b)$, $j = 0, \ldots, b-1$ as follows:
   (a) Generate a vector of n_i's of length b that sums to n with $b - r$ entries of λb^{m-1} and r entries of $\lambda b^{m-1} + 1$:
      i. Randomly choose a subset $\mathcal{I}$ of j indices from $\{1, \ldots, b\}$.
      ii. If $i \in \mathcal{I}$, then $n_i = \lambda b^{m-1} + k + 1$.
      iii. Otherwise, $n_i = \lambda b^{m-1} + k$.
   (b) For each $i = 1, \ldots, b$, we generate $N_{(i-1)b^m+1}, \ldots, N_{ib^m}$ by restarting at Step 1 with the same base b but with $n = n_i$, and $q = m$. At this step, we have b recursive calls to the function.

   Note that in Step 4b, when we return to Step 1, if n_i is a power of b, q is not guaranteed to be equal to $\lceil \log_b n \rceil$, as it can also take the value $\lceil \log_b n \rceil + 1$. For example, this case may arise when sampling $n = 10$ points in base 3. We would need to allocate these 10 points into 27 strata, so we will need to, at this step, put 3 points into 9 strata. So even though $3 = 3^1$, we would want $q = 2$ as $9 = 3^2$: q is counting how many times we need to subdivide the unit interval.

After acquiring the N_j using Algorithm 8, we must generate the point set. This process is straightforward. To generate the point set based on the N_j, for every $j = 1 \ldots M$, if $N_j = 1$, generate a point uniformly in the interval $[jb^{-q}, (j+1)b^{-q})$. If $N_j = 0$, do not generate a point in the interval.

Proposition 5.2.4. *The point set created using the algorithm described in Algorithm 8 to define the N_j's is unbiased and has the same joint distribution as the scrambled van der Corput sequence.*

Proof. To prove Proposition 5.2.4, we only need to show that the point set created using

Algorithm 8 has the same joint distribution as the scrambled van der Corput sequence, as we know the scrambled van der Corput sequence is unbiased [22].

We show this property by deriving the joint pdf of our point set, and showing that it is the same as the joint pdf of a one-dimensional scrambled van der Corput sequence. Recall from earlier chapters and [22, 90], we have the following definitions which are used to define the joint pdf, which we will use again here:

Definition 5.2.5. Let $P_n = \{U_1, \ldots, U_n\}$ be a point set in $[0, 1)$ and $b, i, k \in \mathbb{N}, b \geq 2$. Then,

1. $N_b(i; P_n)$ is the number of ordered pairs of distinct points (U_l, U_j) in P_n such that $\boldsymbol{\gamma}_b(U_l, U_j) = i$,
2. $M_b(k; P_n)$ is the number of ordered pairs of distinct points (U_l, U_j) in P_n such that $\boldsymbol{\gamma}_b(U_l, U_j) \geq k$, and
3. $N_b(k; P_n, U_l) = \sum_{e \in \{0,1\}} (-1)^{|e|} M_b(k + e; P_n, U_l)$.

As well, the joint pdf of a one-dimensional scrambled van der Corput is as follows:

$$\psi(x, y) = \begin{cases} \frac{N_b(i;P_n)}{n(n-1)} \frac{b^{1+i}}{(b-1)}, & \text{if } i < \infty, \\ 0, & \text{if } i = \infty. \end{cases}$$

We show that the scrambled van der Corput point-set and the point set constructed using the proposed stratified sampling method based on Algorithm 8 yield the same $M_b(k)$ as defined in Definition 5.2.5. This means that $N_b(i)$ and thus the joint pdf $\psi(x, y)$ for all pairs of points (x, y) would then be the same for both methods.

From Chapter 4, since the van der Corput sequence in base b is equivalent to a one-dimensional Halton sequence, we know that

$$_{vdC}M_b(k; P_n) = \left\lfloor \frac{n-1}{b^k} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{b^k} \right\rfloor b^k - b^k \right). \tag{5.4}$$

For the stratified sampling estimator, for any k, we can write $n = qb^k + r$, where $q = \lfloor \frac{n}{b^k} \rfloor$ and $0 \leq r < b^k$. By the construction of the stratified sampling estimator, we divide the unit interval into b^k segments and any pairs of points in the same segment will share at least k initial common digits. There are $b^k - r$ of these segments with q points, and r segments with $q + 1$ points. Thus, the number of ordered pairs of points that are in the

same segment is $(b^k - r)q(q-1) + r(q+1)q$. Substituting in $r = n - qb^k$, we have $M_b(k; P_n)$ for the point set created via stratified sampling:

$$_{SS}M_b(k; P_n) = 2nq - q^2b^k - qb^k.$$

Now, we substitute in $q = \left\lfloor \frac{n}{b^k} \right\rfloor$ and simplify and get

$$_{SS}M_b(k; P_n) = 2n\left\lfloor \frac{n}{b^k} \right\rfloor - \left\lfloor \frac{n}{b^k} \right\rfloor^2 b^k - \left\lfloor \frac{n}{b^k} \right\rfloor b^k = \left\lfloor \frac{n}{b^k} \right\rfloor \left(2n - \left\lfloor \frac{n}{b^k} \right\rfloor b^k - b^k\right).$$

This is very similar to (5.4), except with $\left\lfloor \frac{n-1}{b^k} \right\rfloor$ rather than $\left\lfloor \frac{n}{b^k} \right\rfloor$. We show that $_{SS}M_b(k; P_n) = {}_{vdC}M_b(k; P_n)$ by considering the following two cases:

1. $\left\lfloor \frac{n}{b^k} \right\rfloor = \left\lfloor \frac{n-1}{b^k} \right\rfloor$. This case occurs when $r \neq 0$. If this is the case, then we can immediately conclude $_{SS}M_b(k; P_n) = \left\lfloor \frac{n-1}{b^k} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{b^k} \right\rfloor b^k - b^k\right) = {}_{vdC}M_b(k; P_n)$ and we are done.

2. $\left\lfloor \frac{n}{b^k} \right\rfloor \neq \left\lfloor \frac{n-1}{b^k} \right\rfloor$. In this case, $r = 0$ and $\left\lfloor \frac{n}{b^k} \right\rfloor = \frac{n}{b^k} = q$ and $\left\lfloor \frac{n-1}{b^k} \right\rfloor = \frac{n}{b^k} - 1 = q - 1$. Then, we can write $_{SS}M_b(k; P_n)$ as:

$$_{SS}M_b(k; P_n) = \frac{n}{b^k}\left(2n - \frac{n}{b^k}b^k - b^k\right) = \frac{n}{b^k}(2n - n - b^k) = \frac{n^2}{b^k} - n.$$

   Similarly, we can write $_{vdC}M_b(k; P_n)$ as:

$$\begin{aligned} {}_{vdC}M_b(k; P_n) &= \left(\frac{n}{b^k} - 1\right)\left(2n - \left(\frac{n}{b^k} - 1\right)b^k - b^k\right) = \left(\frac{n}{b^k} - 1\right)(2n - n + b^k - b^k) \\ &= \left(\frac{n}{b^k} - 1\right)n = \frac{n^2}{b^k} - n. \end{aligned}$$

   Thus, $_{SS}M_b(k; P_n) = {}_{vdC}M_b(k; P_n)$, as needed.

Hence, we find that $_{SS}M_b(k; P_n) = {}_{vdC}M_b(k; P_n)$ and thus $_{SS}N_b(k; P_n) = {}_{vdC}N_b(k; P_n)$ for all k. This implies $_{SS}\psi(x, y) = {}_{vdC}\psi(x, y)$ for all $(x, y) \in [0, 1)^2$. Thus, the stratified sampling estimator has the same distribution as the scrambled van der Corput estimator. □

Due to the connection between our stratified sampling estimator and nested scrambling, although n is not necessarily an integer power of b and thus it is not stratified sampling with proportional allocation, we can still apply results about the variance of scrambled

estimators for $(0,1)-$sequences [31]. That is, even though the variance is no longer guaranteed to be no higher than for Monte Carlo for any function, the superior asymptotic bounds for the variance of a scrambled estimator apply.

5.2.4 Comparative efficiency analysis for fixed n

Using the above connection between scrambling and stratified sampling, for fixed n we can implement a scrambling by sampling the N_j's as above and once the intervals where a point will be placed have been identified, we simply place a point uniformly in that interval. This approach is computationally more efficient than proceeding via recursive permutations, as is required when implementing nested scrambling.

The recursive Algorithm 8 for stratified sampling has $\lceil \log_b(n) \rceil$ layers, and each call to the function has b sub-calls. Thus, in total, there are $b + b^2 + \ldots + b^{\lceil \log_b(n) \rceil} = O(n)$ operations. For nested scrambling of the van der Corput sequence, we need to scramble the first $\lfloor \log_b(n-1) \rfloor + 1$ digits for each of the n points. This means that the number of operations needed is $O(n \log(n))$. In addition, permutations have to be stored in a lookup dictionary. There are b combinations of 1 digit, b^2 combinations of 2 digits, and so on. Thus, the amount of storage needed for nested scrambling is $b + 2b^2 + \ldots + (\lfloor \log_b(n-1) \rfloor + 1) b^{\lfloor \log_b(n-1) \rfloor + 1} = O(n \log(n))$. Another advantage for the stratified sampling algorithm is that it does not require any additional storage for a lookup dictionary, as the nested scrambling algorithm does.

Figure 5.3 shows the runtime needed to generate $n = 2000, 4000, \ldots 100,000$ points from the scrambled base 4 van der Corput sequence needed to construct point sets on the triangle. Our stratified sampling implementation is compared with Owen's nested scrambling. Our stratified sampling estimator is more computationally efficient, and increases in runtime occur only at integer powers of 4.

5.2.5 Extending a stratified estimator

The above approach for stratified sampling works well when n is fixed. There, we used the connection between stratified sampling and nested scrambling to give a faster implementation of nested scrambling. Now, we use the connection between stratified sampling and nested scrambling to be able to extend a stratified estimator. Typically, stratified sampling is applied for fixed n and if a larger point set a needed, a new one is generated from scratch instead of only generating the additional points. Here we argue that the

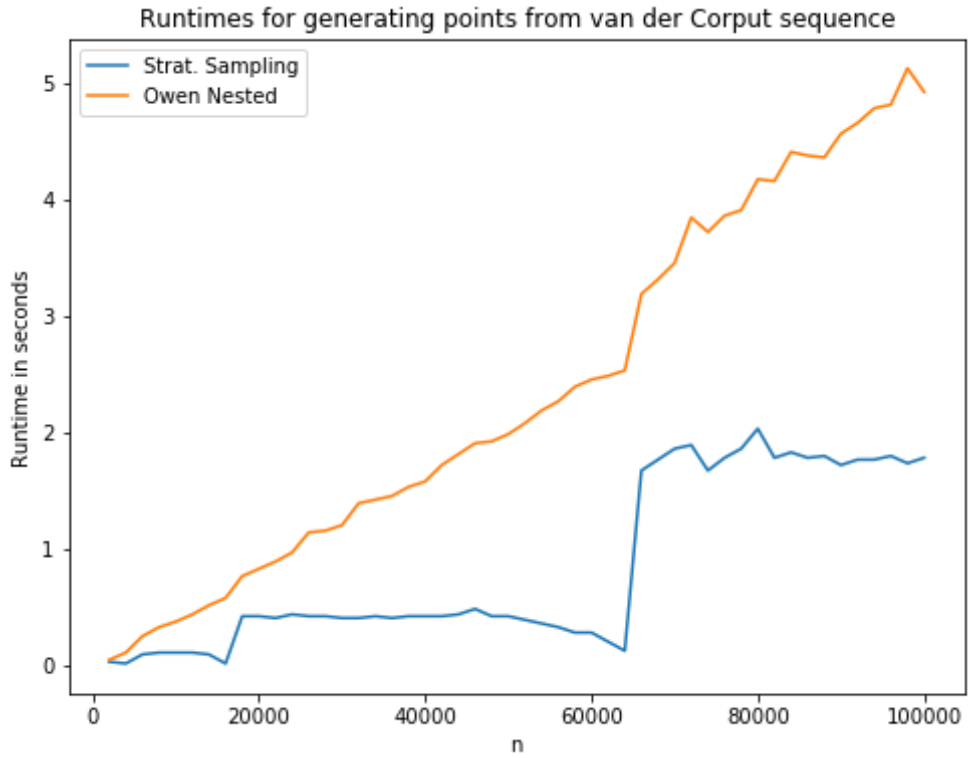


Figure 5.3: Runtime Comparison of the time it takes to generate a scrambled van der Corput sequence in base 4 using Owen's Scrambling vs Stratified Sampling.

nested permutations used for scrambling can be thought of as a way to allow for a stratified estimator to be extended easily, i.e., for points to be added without having to restart fresh with a completely new stratified estimator by recursively choosing a stratum at each level that has the least number of points, as explained in Algorithm 9. For simplicity, we deal directly with the strata $N_1, .., N_M$ as generated by Algorithm 8 instead of the point set P_n, since after generating the strata, it is simply a matter of placing a point uniformly within each stratum j with corresponding $N_j = 1$. This algorithm essentially works by recursively subdividing the interval into b subintervals, and randomly choosing one with the least points to add the next point into.

If we are working directly with the point set and not the strata, then we must modify Algorithm 9 by changing Step 1a so that we first determine which b^q strata are equivalent to 1, and set the rest to 0. Then in the following step, when putting 2 points within the same interval of size b^{-q}, since one of the N_j would already be set to 1 based on the point that is already there, we only select the second subinterval of size b^{-q+1} without replacement. Likewise, before Step 2a, we must first populate $N_1, .., N_M$ based on the existing point set.

Algorithm 9: Extending a stratified estimator

Given $N_1, .., N_M$ be indicators for M strata where M is defined as in (5.3) and $\sum_{j=1}^{M} N_j = n$, a base b, we sample one additional point as follows:

1. If $n = M$, then we have to increase the number of strata from $M = b^q$ to $M = b^{q+1}$
   (a) Initialize $N_1, \ldots, N_{b^{q+1}} = 0$
   (b) Randomly select an interval of size b^{-q} to have 2 points in. For this interval, subdivide into b intervals, and randomly select two of these subintervals to have a point, i.e., $N_j = 1$.
   (c) For the other intervals of size b^{-q}, subdivide into b intervals, and randomly select one of these intervals to have a point, i.e., $N_i = 1$.
   (d) return $N_1, \ldots, N_{b^{q+1}}$
2. If $n \neq M$, then we do not need to increase the number of strata. We work with $N_1, .., N_M$ as follows:
   (a) Divide the unit interval into b subintervals, such that each of these subintervals is represented by M/b strata.
   (b) Let $L_j = \sum_{i=1}^{M/b} N_{(M/b)(j-1)+i}$ for $j = 1, \ldots, b$
   (c) The L_j will differ by at most one. Randomly pick a j from the L_j that have the minimum number of points.
   (d) If $L_j = 0$, randomly choose one of the M/b strata to place a point in.
   (e) If $L_j > 0$, then repeat this algorithm from Step 1 on the j^{th} subinterval.

We now illustrate Algorithm 9 with an example showing how to extend a scrambled estimator from $n = 7$ to $n = 10$ when working in base 3. Let P_7 be the original point set with 7 points. Denote by π_l the number of points in $[(l-1)/3, l/3)$ for $l = 1, 2, 3$ within P_7. That is, $\pi_1 = N_1 + N_2 + N_3$, $\pi_2 = N_4 + N_5 + N_6$, and $\pi_3 = N_7 + N_8 + N_9$. That is, π_l and N_j both enumerate strata, just of different sizes. Given that $7 = 2 \times 3 + 1$, when constructing the estimator for $n = 7$ we would have had to sample $N_1, \ldots, N_9$ such that one of $\pi_1, \ldots, \pi_3$ is equal to 3 and the other two are equal to 2. Say we have $\pi_1 = \pi_2 = 2$ and $\pi_3 = 3$. Then $N_7 = N_8 = N_9 = 1$ and we also need to choose two indices in each of $\{1, 2, 3\}$ and $\{4, 5, 6\}$ whose corresponding N_j will be set to 1. Say we choose $1, 3, 4, 5$.

If we then want to add 3 points to go to $n = 10 = 1 \times 9 + 1$, it means we are now working with a stratified estimator over strata of size $1/27$ instead of $1/9$. In this case, we

have that π_ℓ now represents the total number of points in each interval of size 1/9, and only one of them will be equal to 2 with the other 8 being equal to 1.

Rather than jumping directly to $n = 10$, let us explain how each point is added.

1. ($n = 8$) Choose which of the two intervals of size 1/9 with no point will have a point uniformly sampled in it
2. ($n = 9$) Sample a point uniformly in the last interval of size 1/9 that has no point
3. ($n = 10$) Choose one of the 9 intervals of size 1/9 which will have a second point placed in it (i.e., which $\pi_1, \ldots, \pi_9$ will be equal to 2, as they are currently all equal to 1); determine in which of the intervals of size 1/27 is the point that is already placed in this interval of size 1/9; randomly choose one of the two empty intervals of size 1/27 to place the second point and then place a point uniformly in it.

Since intervals are always chosen without replacement within the group of b intervals of size b^{-q} are currently working with, it is clear that if we initially generate a random permutation of $[1, \ldots, b]$, we are simply deciding beforehand in which order points will be added within this group of sub-intervals.

5.2.6 From the one-dimensional van der Corput sequence to the triangular van der Corput sequence

It is now straightforward to use the ideas presented thus far for the one-dimensional van der Corput sequence to construct a scheme to sample n points on an arbitrary triangle. The "strata" are now the sub-triangles, and we can use Algorithm 8 to sample the number of points in each sub-triangle, say $N_1, \ldots, N_M$. Then, apply Algorithm 5 to sample N_j (either 0 or 1) points in each sub-triangle for $j = 1, \ldots, M$. This estimator can also be extended using the method described in Section 5.2.5.

An algorithm similar to Algorithm 7 can be used to sample in the triangle – the steps are described in Algorithm 10. Again, the sampling scheme subdivides the triangle into a finer and finer partition of triangles until each sub-triangle gets at most one point. The differences are that we now fill the sub-triangles in a non-deterministic order (that still satisfies the equidistribution properties), and as well, the points within each sub-triangle are uniformly placed rather than put in the centre.

Algorithm 10: Mapping the stratified sampling estimator to the triangle

Given input $n \geq 1$ and target triangle $\triangle(A, B, C)$, generate the first n points, outputted in an $n \times 2$ array $\mathbf{x}$, as follows:

1. Generate a vector of indexes $I_1, \ldots, I_n$ using Algorithm 8 with base $b = 4$. Each of the indexes $0 \leq I_i \leq 4^{\lceil log_4 n \rceil} - 1$, so we know that the base 4 representation is finite for each index. The representation has at most $K_i = \lceil \log_4(i) + 1 \rceil$ digits, which means that the expansion is finite, and we do not have to infinitely divide the triangle into sub-triangles.

2. For $i = 1, \ldots, n$:

   (a) Compute $(d_0, \ldots, d_{K_i})$ such that $I_i = \sum_{k=0}^{K_i} d_k 4^k$;

   (b) Initialize $T = \triangle(A, B, C)$;

   (c) For $j = 0, \ldots, K_i$,

   Update $T = T(d_j)$ using (5.2);

   (d) Set $\mathbf{x}[i]$ to be a random uniformly sampled point within T. In our implementation, we use the Inverse Rosenblatt transformation of Algorithm 5;

3. Return $\mathbf{x}$;

This avoids the poor projection properties of the original triangular van der Corput sequence as seen in Figure 5.4, as well as providing a way to randomize the point set which allows for error estimation.

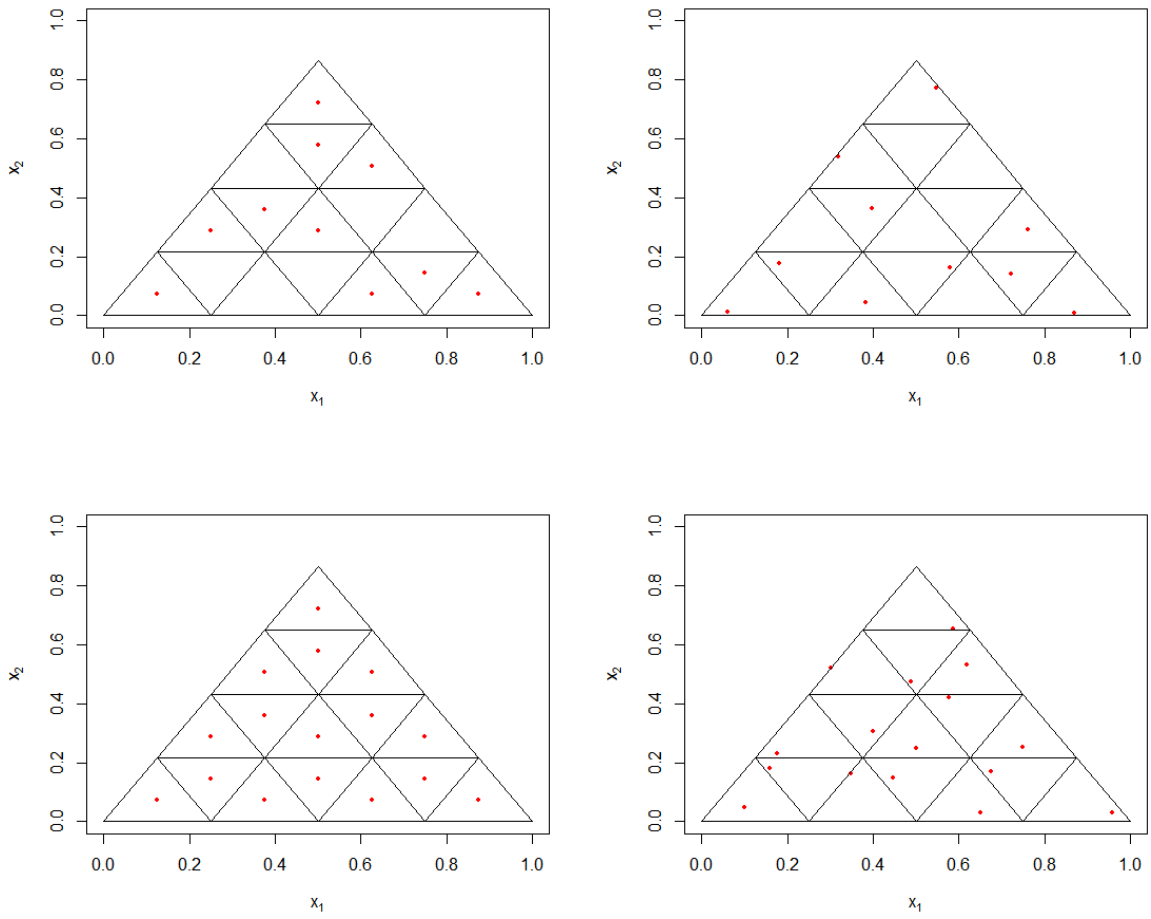


Figure 5.4: Examples of the triangular van der Corput points generated on $\triangle_E$, with $n = 10$ (top) and $n = 16$ (bottom). The images on the left have each point at the centre of the terminal sub-triangle, while the images on the right have the points scrambled. For the $n = 10$ case, the sub-triangles with points are selected via stratified sampling.

Proposition 5.2.6. *The estimator $\hat{\mu}_{scr,n}$ is unbiased.*

Proof. This is equivalent to showing that each point is uniformly distributed over the triangle. Without loss of generality, for the equilateral triangle bounded by $(0, 0)$, $(1, 0)$ and $(0.5, \sin \pi/3)$, the marginal pdf f for the point $\mathbf{u} = (u_1, u_2)$ uniformly distributed over

$\triangle_E$ is:

$$f(\mathbf{u}) = \begin{cases} \frac{2}{\sin \pi/3} & \text{if } 0 \le u_1 \le 1/2 \text{ and } u_2 \le u_1 \tan \pi/3 \\ & \text{or } 1/2 \le u_1 \le 1 \text{ and } (1-u_1) \tan \pi/3 \\ 0 & \text{otherwise.} \end{cases}$$

Given $n \ge 1$, for the stratified sampling estimator, we subdivide the triangle into 4^q sub-triangles each with area $\frac{2}{\sin(\pi/3)4^q}$. Since each sub-triangle is chosen with equal probability (as the N_j's have the same marginal distribution), and then we uniformly sample within each of the n sub-triangles, the value of the pdf $f(u)$ within the sub-triangle is equal to the reciprocal of its area, $\frac{\sin(\pi/3)4^q}{2}$. Then, a point sampled using the stratified sampling algorithm has marginal pdf $g(\mathbf{u})$.

$$g(\mathbf{u}) = \begin{cases} \frac{\sin(\pi/3)4^q}{2\times \#\text{ sub-triangles}} & \text{if } \mathbf{u} = (u_1, u_2) \text{ is inside the triangle bounded by} \\ & (0,0), (1,0) \text{ and } (0.5, \sin \pi/3), \\ 0 & \text{otherwise.} \end{cases}$$

$$= \begin{cases} \frac{2}{\sin \pi/3} & \text{if } (u_1, u_2) \text{ is inside the triangle bounded by } (0,0), (1,0) \text{ and } (0.5, \sin \pi/3), \\ 0 & \text{otherwise.} \end{cases}$$

Thus, since $g(\mathbf{u}) = f(\mathbf{u})$, the stratified sampling algorithm samples uniformly over the triangle and thus the resulting estimator is unbiased. □

5.3 Numerical experiments on the triangle

Now that we have described our proposed construction for a point set on the triangle, we compare its performance on numerical integration problems with existing constructions. As mentioned, there are very few, if any, numerical experiments on the triangle that compare RQMC integration variances, so we hope that these experiments give insight towards the performance of the various methods.

We use the following functions over $\triangle_R$ as described in Section 2.5 to compare different constructions on the triangle:

1. $f_1(x, y) = (|x - \beta| + y)^d$ from [69].

2. $f_2(x, y) = \cos(2\pi\beta + \alpha_1 x + \alpha_2 y)$, from [69].

3. $f_3(x, y) = x^{\alpha_3} + y^{\alpha_3}$.

We compare the following randomized methods to estimate μ_j:

1. PRNG + root: This is equivalent to the MC method: we generate pseudo-random points in the unit square and then apply the transformation "root" from [69],

2. rSobol' + root: the Sobol sequence, randomized with a digital shift with transformation root [69] applied,

3. rLattice1: the rank-1 lattice of [16, 21], randomized with a shift,

4. rLattice2: the rank-2 lattice of Basu and Owen [7], randomized with a shift. This method is not extensible, so a new point set must be generated every time n is increased.

5. rvdC: the randomized triangular van der Corput sequence based on stratified sampling.

The first $n = 1000$ points from these constructions are shown in Figure 5.5.

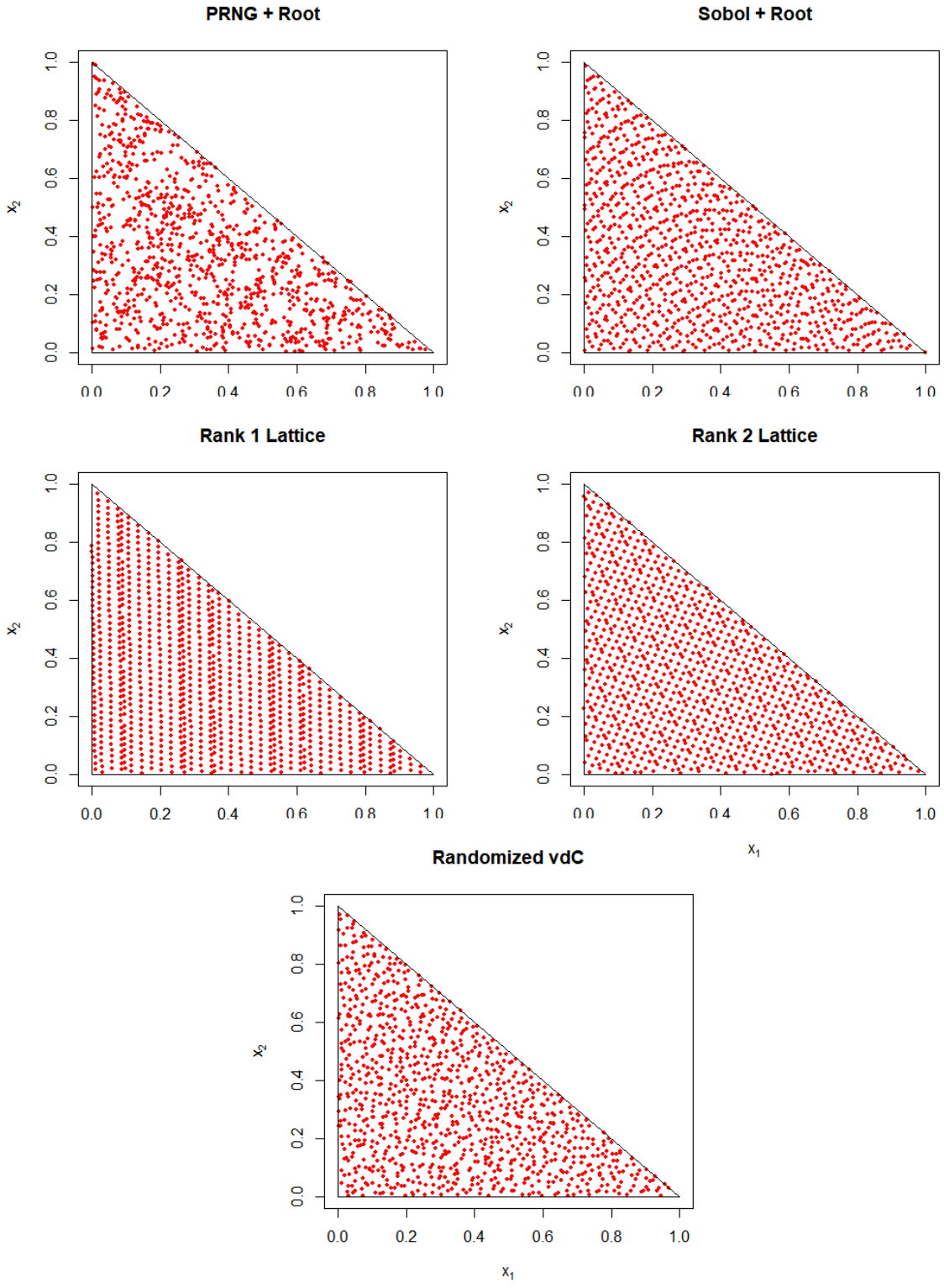


Figure 5.5: First $n = 1000$ points of the point sets used in our experiments.

For each method and each sample size $n \in 2^4, 2^5, \ldots 2^{17}$, $V = 25$ randomizations were used. The estimates are obtained as the sample average of the realizations, while the variance is estimated as the sample variance of the V independent draws. Figure 5.6 displays the results. We can see from the results that the bivariate Sobol' sequence mapped to the triangle using transformation root is typically the best performing method on these test functions. However, our stratified sampling method is approximately equal in performance to the other RQMC methods. The variance reduction of stratified sampling is more pronounced for functions where the within-strata variance is small and the between-strata variance is larger. As well, nested scrambling has been shown to have significant variance reductions for smooth functions. Thus, it is unsurprising that for the function f_3, the stratified sampling scheme had the best performance, as out of the three test functions used, it is the smoothest function with the most between-strata variance.

5.4 Conclusion

In this chapter, we improved upon the triangular van der Corput sequence of Basu and Owen by proposing a sampling scheme that uses their idea of recursively subdividing the triangle, but with superior one-dimensional projections. We also showed that the scrambled van der Corput sequence can be efficiently implemented using stratified sampling.

Future work in this area includes extending similar stratified sampling schemes onto other surfaces, such as on the surfaces of spheres and simplexes. As well, we wish to explore constructions and applications that require sampling on multiple triangles, such as objects constructed as meshes of triangles. The stratified sampling algorithm can be implemented in any base, and as such, it is also possible to create constructions on the triangle in higher bases such as base 9 or 16.

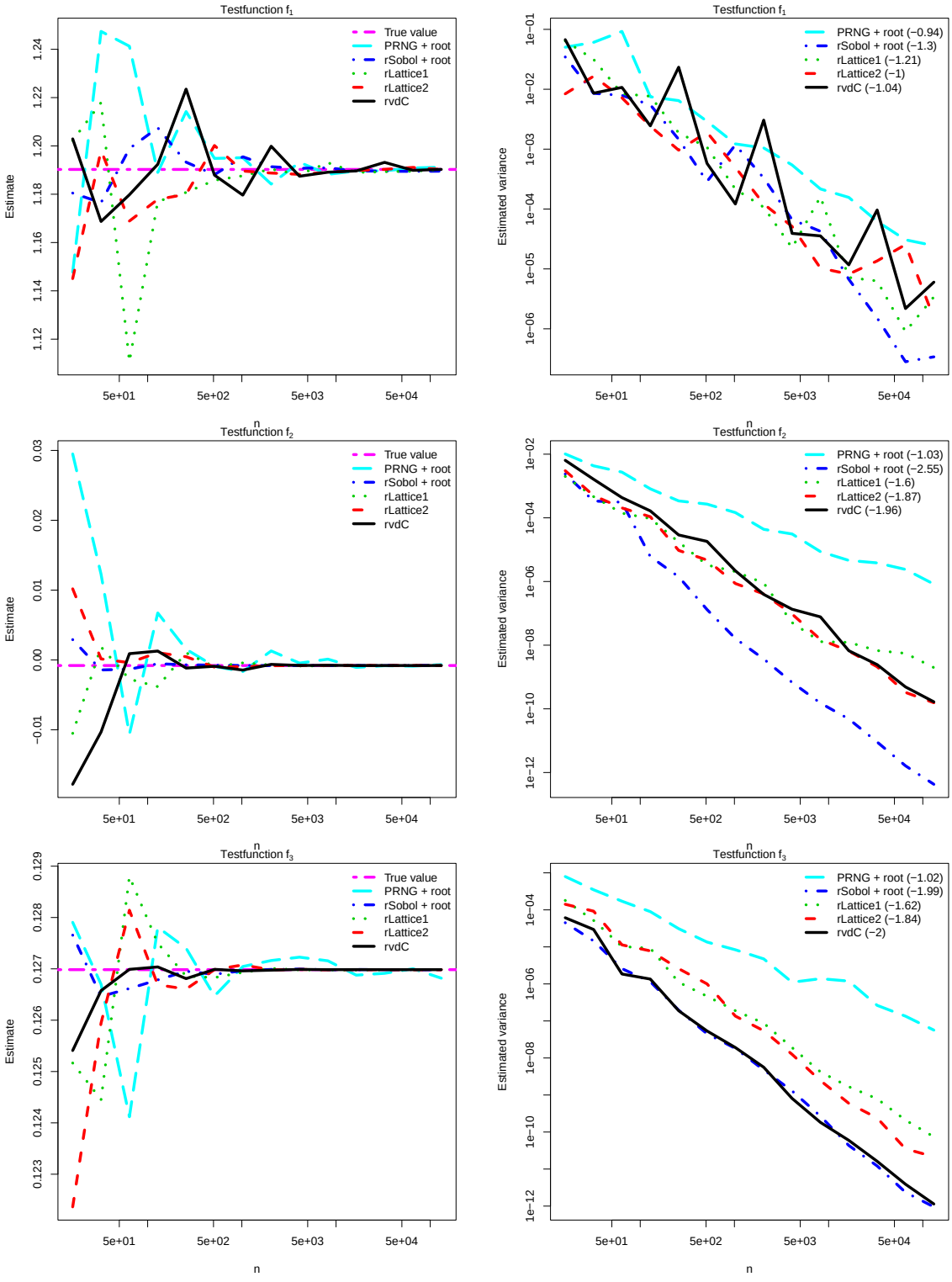


Figure 5.6: Estimates (left) and estimated variances (right) when integrating f_1 (top), f_2 (middle) or f_3 (right). For each n, $V = 25$ randomizations were used.

Chapter 6

Conclusion

This thesis is on the constructions and applications of (R)QMC point sets that have the property of negative dependence. We found constructions for point sets with good negative dependence properties and evaluated how these constructions perform on a variety of numerical integration problems, comparing the performance of these constructions with each other and with Monte Carlo.

In Chapter 3, we explored using the $C_b(\boldsymbol{k}; P_n)$ values to measure the quality of various point sets and compare their dependence properties. We gave numerical examples on the Sobol' and Faure sequences, primarily using two-dimensional projections, to illustrate the ability of the $C_b(\boldsymbol{k}; P_n)$ values to differentiate between "good" and "bad" point sets in terms of their integration power when used as a scrambled RQMC estimator. We also evaluated several generalized Faure sequences constructed using the permutations originally introduced in [25] for the van der Corput sequence as factors. Future research in this area includes improving the algorithm for calculating the $C_b(\boldsymbol{k}; P_n)$ values to be more efficient in terms of both runtime and memory.

In Chapter 4, we proved the property of negative dependence for the Halton sequence and introduced a multi-base analogous to C_b, and based on this result, propose using a multi-base scrambling for randomization that is not currently used in practice. We derived a formula for the joint pdf for the Halton sequence, as well as give a set of permutations to generalize the Halton sequence by optimizing the dependence properties of low-dimensional projections of the Halton sequence. We also evaluated permutations for the Halton sequence based on the ones introduced in [25].

In Chapter 5, we focused on the construction of low-discrepancy point sets on triangles. Specifically, we improve upon the triangular van der Corput sequence of Basu and

Owen by applying the scrambled van der Corput sequence rather than the deterministic van der Corput sequence to greatly improve the one-dimensional projection properties of the sequence. We provide an efficient implementation of the scrambled van der Corput sequence using stratified scrambling, and show that this connection between scrambling and stratification can also be used to extend a stratified estimator. We give numerical results comparing various constructions on the triangle. Future work in this area includes extending the stratified sampling estimator to higher-dimensional domains, such as on the surfaces of simplexes or spheres.

References

[1] K. Arnold, J. Gosling, and D. Holmes, *The java programming language*, Addison Wesley Professional, 2005.

[2] J. Arvo, *Stratified sampling of spherical triangles*, Proceedings of the 22nd annual conference on computer graphics and interactive techniques, 1995, pp. 437–438.

[3] D. I. Asotsky and I. M. Sobol', *One more experiment on estimating high-dimensional integrals by quasi-Monte Carlo methods*, Mathematics and Computers in Simulation **62** (2003), no. 3-6, 255–263.

[4] E. Atanassov, *On the discrepancy of the Halton sequences*, Mathematica Balkanica **18** (2004), 15–32.

[5] A. N. Avramidis and J. R. Wilson, *Integrated variance reduction strategies for simulation*, Operations Research **44** (1996), no. 2, 327–346.

[6] J. W. Backus and W. P. Heising, *Fortran*, IEEE Transactions on Electronic Computers **4** (1964), 382–385.

[7] K. Basu and A. Owen, *Low discrepancy constructions in the triangle*, SIAM Journal on Numerical Analysis **53** (2015), no. 2, 743–761.

[8] R. E. Bellman, *Adaptive control processes: a guided tour*, Vol. 2045, Princeton university press, 1961.

[9] L. Brandolini, L. Colzani, G. Gigante, and G. Travaglini, *A Koksma–Hlawka inequality for simplices*, Trends in harmonic analysis, 2013, pp. 33–46.

[10] P. Bratley, B. L. Fox, and H. Niederreiter, *Programs to generate Niederreiter's low-discrepancy sequences*, ACM Transactions on Mathematical Software (TOMS) **20** (1994), no. 4, 494–495.

[11] R. E. Caflisch, W. J. Morokoff, and A. B. Owen, *Valuation of mortgage backed securities using brownian bridges to reduce effective dimension*, Vol. 24, Department of Mathematics, University of California, Los Angeles, 1997.

[12] T. Chatzivasileiadis, *Quasi-Monte Carlo application in CGE systematic sensitivity analysis*, Applied Economics Letters **25** (2018), no. 21, 1521–1526.

[13] H. Chi, M. Mascagni, and T. Warnock, *On the optimal Halton sequence*, Mathematics and computers in simulation **70** (2005), no. 1, 9–21.

[14] P. Christensen, A. Kensler, and C. Kilpatrick, *Progressive multi-jittered sample sequences*, Computer graphics forum, 2018, pp. 21–33.

[15] M. Cieslak, C. Lemieux, J. Hanan, and P. Prusinkiewicz, *Quasi-Monte Carlo simulation of the light environment of plants*, Functional Plant Biology **35** (2008), no. 10, 837–849.

[16] R. Cools, F. Kuo, and D. Nuyens, *Constructing embedded lattice rules for multivariate integration*, SIAM Journal on Scientific Computing **28** (2006), no. 6, 2162–2188.

[17] R. Cranley and T. N. L. Patterson, *Randomization of number theoretic methods for multiple integration*, SIAM Journal on Numerical Analysis **13** (1976), no. 6, 904–914.

[18] F. De Rainville, C. Gagné, O. Teytaud, and D. Laurendeau, *Evolutionary optimization of low-discrepancy sequences*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **22** (2012), no. 2, 1–25.

[19] J. Dick and F. Pillichshammer, *Digital nets and sequences: discrepancy theory and quasi–Monte Carlo integration*, Cambridge University Press, 2010.

[20] G. Y. Dong, H. Faure, and C. Lemieux, *A negative dependence framework to assess different forms of scrambling*, Working Paper (2022).

[21] G. Y. Dong, E. Hintz, M. Hofert, and C. Lemieux, *Randomized quasi-Monte Carlo methods on triangles: extensible lattices and sequences*, Working Paper (2022).

[22] G. Y. Dong and C. Lemieux, *Dependence properties of scrambled Halton sequences*, Mathematics and Computers in Simulation (2022).

[23] K. Fang and Y. Wang, *Number-theoretic methods in statistics*, Vol. 51, CRC Press, 1993.

[24] H. Faure, *Discrépance de suites associées à un système de numération (en dimension s)*, Acta Arithmetica **41** (1982), no. 4, 337–351.

[25] ———, *Good permutations for extreme discrepancy*, Journal of Number Theory **42** (1992), no. 1, 47–56.

[26] H. Faure and C. Lemieux, *Generalized Halton sequences in 2008: A comparative study*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **19** (2009), no. 4, 1–31.

[27] ———, *Implementation of irreducible Sobol' sequences in prime power bases*, Mathematics and Computers in Simulation **161** (2019), 13–22.

[28] T. A. Fox, M. Gao, T. E. Barchyn, Y. L. Jamin, and C. H. Hugenholtz, *An agent-based model for estimating emissions reduction equivalence among leak detection and repair programs*, Journal of Cleaner Production **282** (2021), 125237.

[29] A. Genz, *Testing multidimensional integration routines*, Proc. of international conference on tools, methods and languages for scientific and engineering computation, 1984, pp. 81–94.

[30] ———, *A package for testing multiple integration subroutines*, Numerical integration, 1987, pp. 337–340.

[31] M. Gerber, *On integration methods based on scrambled nets of arbitrary size*, Journal of Complexity **31** (2015), no. 6, 798–816.

[32] D. T. Gillespie, *Exact stochastic simulation of coupled chemical reactions*, The journal of physical chemistry **81** (1977), no. 25, 2340–2361.

[33] T. Goda, K. Suzuki, and T. Yoshiki, *Quasi-Monte Carlo integration for twice differentiable functions over a triangle*, Journal of Mathematical Analysis and Applications **454** (2017), no. 1, 361–384.

[34] B. Gough, *Gnu scientific library reference manual*, Network Theory Ltd., 2009.

[35] J. H. Halton, *On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals*, Numerische Mathematik **2** (1960), no. 1, 84–90.

[36] ———, *Algorithm 247: Radical-inverse quasi-random point sequence*, Communications of the ACM **7** (1964), no. 12, 701–702.

[37] A. Hejlsberg, S. Wiltamuth, and P. Golde, *C# language specification*, Addison-Wesley Longman Publishing Co., Inc., 2003.

[38] E. Hintz, *Randomized quasi-Monte Carlo methods with applications to quantitative risk management* (2022).

[39] M. Hofert and C. Lemieux, *qrng: (Randomized) Quasi-Random Number Generators*, 2018. R package version 3.0.0, URL https://cran.r-project.org/web/packages/qrng/index.html.

[40] H. S. Hong and F. J. Hickernell, *Algorithm 823: Implementing scrambled digital sequences*, ACM Transactions on Mathematical Software **29** (2003), 95–109.

[41] X. Jin and A. X. Zhang, *Reclaiming quasi–Monte Carlo efficiency in portfolio value-at-risk simulation through fourier transform*, Management Science **52** (2006), no. 6, 925–938.

[42] M. R. Johnson, D. R. Tyner, and A. J. Szekeres, *Blinded evaluation of airborne methane source detection using bridger photonics LiDAR*, Remote Sensing of Environment **259** (2021), 112418.

[43] A. Keller, *Quasi-Monte Carlo methods for photorealistic image synthesis*, Shaker, 1997.

[44] B. W. Kernighan and D. M. Ritchie, *The C programming language*, 2006.

[45] L. Kocis and W. J. Whiten, *Computational investigations of low-discrepancy sequences*, ACM Transactions on Mathematical Software (TOMS) **23** (1997), no. 2, 266–294.

[46] E. L. Lehmann, *Some concepts of dependence*, The Annals of Mathematical Statistics (1966), 1137–1153.

[47] C. Lemieux, *Monte Carlo and quasi-Monte Carlo sampling*, Springer Series in Statistics, Springer New York, 2009.

[48] ———, *Negative dependence, scrambled nets, and variance bounds*, Mathematics of Operations Research **43** (2018), no. 1, 228–251.

[49] C. Lemieux and H. Faure, *New perspectives on* $(0, s)$*-sequences*, Monte Carlo and quasi-Monte Carlo methods 2008, 2009, pp. 113–130.

[50] C. Lemieux and P. L'Ecuyer, *On the use of quasi-Monte Carlo methods in computational finance*, International conference on computational science, 2001, pp. 607–616.

[51] C. Lemieux and J. Wiart, *On the distribution of scrambled* $(0, m, s)$*-nets over unanchored boxes*, 2021. To appear in Monte Carlo and Quasi Monte Carlo Methods 2020.

[52] J. Matousek, *On the L2-discrepancy for anchored boxes*, Journal of Complexity **14** (1998), no. 4, 527–556.

[53] M. Matsumoto and T. Nishimura, *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **8** (1998), no. 1, 3–30.

[54] M. D. McKay, R. J. Beckman, and W. J. Conover, *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*, Technometrics **42** (1979), no. 1, 55–61.

[55] H. Niederreiter, *Random number generation and quasi-Monte Carlo methods*, SIAM CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 63, SIAM, Philadelphia, 1992.

[56] S. Ninomiya and S. Tezuka, *Toward real-time pricing of complex financial derivatives*, Applied Mathematical Finance **3** (1996), no. 1, 1–20.

[57] Visual Numerics, *International mathematics and statistics library (IMSL) volumes 1 and 2*, Visual Numerics Inc., Houston, 1997.

[58] G. Ökten, *Generalized von Neumann–Kakutani transformation and random-start scrambled Halton sequences*, Journal of Complexity **25** (2009), no. 4, 318–331.

[59] A. B. Owen, *Orthogonal arrays for computer experiments, integration and visualization*, Statistica Sinica (1992), 439–452.

[60] ———, *Randomly permuted (t, m, s)-nets and (t, s)-sequences*, Monte Carlo and quasi-Monte Carlo methods in scientific computing, 1995, pp. 299–317.

[61] ———, *The dimension distribution and quadrature test functions*, Statistica Sinica (2003), 1–17.

[62] ———, *Variance with alternative scramblings of digital nets*, ACM Transactions on Modeling and Computer Simulation (TOMACS) **13** (2003), no. 4, 363–378.

[63] ———, *A randomized Halton algorithm in r*, arXiv preprint arXiv:1706.02808 (2017).

[64] A. B. Owen et al., *Scrambled net variance for integrals of smooth functions*, The Annals of Statistics **25** (1997), no. 4, 1541–1562.

[65] A. B. Owen and D. Tavella, *Scrambled nets for value at risk calculations*, VaR: Understanding and Applying Value-at-Risk, London: Risk (1997), 289–297.

[66] Art B Owen, *On dropping the first sobol'point*, International conference on Monte Carlo and quasi-Monte Carlo methods in scientific computing, 2022, pp. 71–86.

[67] S. Paskov and J. F. Traub, *Faster valuation of financial derivatives* (1996).

[68] M. Pharr, *Adventures in sampling points on triangles*, 2019. Accessed: 2022-06-14.

[69] T. Pillards and R. Cools, *Transforming low-discrepancy sequences from a cube to a simplex*, Journal of computational and applied mathematics **174** (2005), no. 1, 29–42.

[70] F. Pillichshammer, P. Kritzer, and H. Faure, *From van der Corput to modern constructions of sequences for quasi-Monte Carlo rules*, Indag. Math. **26** (2015), 760–822.

[71] C. J. Price and C. P. Price, *Recycling primes in Halton sequences: an optimization perspective*, Adv Model Optim **14** (2012), 17–29.

[72] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021.

[73] M. Rosenblatt, *Remarks on a multivariate transformation*, The Annals of Mathematical Statistics **23** (195209), no. 3, 470–472.

[74] I. H. Sloan and A. V. Rezstov, *Component-by-component construction of good lattice rules*, Math. Comput. **71** (2002), 263–273.

[75] I. M. Sobol', *On the distribution of points in a cube and the approximate evaluation of integrals*, Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki **7** (1967), no. 4, 784–802.

[76] ———, *Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates*, Mathematics and computers in simulation **55** (2001), no. 1-3, 271–280.

[77] B. Stroustrup, *The C++ programming language*, Pearson Education India, 2000.

[78] K. S. Tan and P. P. Boyle, *Applications of randomized low discrepancy sequences to the valuation of complex securities*, Journal of Economic Dynamics and Control **24** (2000), no. 11-12, 1747–1782.

[79] B. Tang, *Orthogonal array-based Latin hypercubes*, Journal of the American statistical association **88** (1993), no. 424, 1392–1397.

[80] S. Tezuka, *Uniform random numbers: Theory and practice*, Kluwer Academic Publishers, Norwell, MA, 1995.

[81] S. Tezuka and T. Tokuyama, *A note on polynomial arithmetic analogue of Halton sequences*, ACM Transactions on Modeling and Computer Simulation **4** (1994), 279–284.

[82] V. Tymchyshyn and A. Khlevniuk, *Beginner's guide to mapping simplexes affinely*, ResearchGate preprint (2019).

[83] D. R. Tyner and M. R. Johnson, *Where the methane is—insights from novel airborne LiDAR measurements combined with ground survey data*, Environmental Science & Technology **55** (2021), no. 14, 9773–9783.

[84] G. Van Rossum and F. L. Drake Jr, *Python tutorial*, Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[85] B. Vandewoestyne and R. Cools, *Good permutations for deterministic scrambled Halton sequences in terms of L_2-discrepancy*, Journal of computational and applied mathematics **189** (2006), no. 1-2, 341–361.

[86] E. Veach, *Robust Monte Carlo methods for light transport simulation*, Vol. 1610, Stanford University PhD thesis, 1997.

[87] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods **17** (2020), 261–272.

[88] X. Wang and F. J. Hickernell, *Randomized Halton sequences*, Mathematical and Computer Modelling **32** (2000), no. 7-8, 887–899.

[89] X. Wang and I. H. Sloan, *Why are high-dimensional finance problems often of low effective dimension?*, SIAM Journal on Scientific Computing **27** (2005), no. 1, 159–183.

[90] J. Wiart, C. Lemieux, and G. Y. Dong, *On the dependence structure and quality of scrambled* (t, m, s)*-nets*, Monte Carlo Methods and Applications (2021).

[91] M. Wnuk and M. Gnewuch, *Note on pairwise negative dependence of randomly shifted and jittered rank-1 lattices*, Operations Research Letters **48** (2020), no. 4, 410–414.

[92] M. Wnuk, M. Gnewuch, and N. Hebbinghaus, *On negatively dependent sampling schemes, variance reduction, and probabilistic upper discrepancy bounds*, Discrepancy Theory **26** (2020), 43–67.

APPENDICES

Appendix A

Factors for the Halton sequence

Dim	DL Factor	FL Factor	c (DL)	c (FL)	c (Halton)
1	1	1	0.998999	0.998999	0.998999
2	2	1	0.999003	0.99901502	0.99901502
3	2	3	0.999003	0.99901502	0.99901502
4	2	3	0.999003	0.99901502	0.99903504
5	9	4	1.01784985	1.1921041	1.08242643
6	11	9	1.16955355	1.77329329	1.22797998
7	12	7	1.24643043	1.65234034	1.69231632
8	1	5	1.37660861	1.61851451	1.69231632
9	21	9	1.30998198	1.43913514	1.69231632
10	25	18	1.63388989	1.58776376	2.35755756
11	5	18	1.63388989	1.94652252	2.66301502
12	22	8	1.63593994	1.94447247	1.9823984
13	32	13	1.63388989	2.33090691	3.55171171
14	28	31	1.77944344	2.04594995	2.089001
15	9	9	1.77944344	1.81736937	2.60253854
16	34	19	1.73946747	2.10437638	2.76551752
17	17	36	1.63388989	1.79174374	3.30160561
18	10	33	1.65029029	2.02544945	4.43015816
19	20	21	2.65481481	1.81121922	3.49226026
20	8	44	1.73639239	2.0849009	4.58186186
21	36	43	2.35960761	2.38010811	5.03082282
22	6	61	2.26120521	2.2407047	5.79651652

Dim	DL Factor	FL Factor	c (DL)	c (FL)	c (Halton)
23	3	60	2.26120521	1.9741982	3.87664464
24	7	56	2.26120521	2.67326527	5.68273874
25	6	26	2.26120521	2.27043043	5.23377778
26	7	71	2.26120521	2.74399199	5.57921121
27	3	32	2.26120521	2.2243043	9.69571171
28	8	77	2.26120521	2.2243043	10.1743984
29	7	26	2.26120521	2.2243043	10.64385986
30	95	95	2.33705706	2.79114314	11.1040961
31	32	92	2.33705706	2.2243043	5.36293093
32	21	47	2.33705706	1.9208969	5.42238238
33	128	29	2.42725926	2.02954955	9.44150551
34	12	61	2.30425626	4.61671271	12.71338539
35	4	57	2.30425626	2.25915516	9.95401802
36	26	69	2.25608008	2.55846246	10.84066466
37	43	115	2.25608008	3.77721722	15.21342142
38	8	63	2.25608008	3.52096096	10.67153554
39	5	92	2.25608008	3.08122523	13.36325125
40	4	31	2.23967968	3.81001802	13.90856456
41	14	104	2.23967968	2.00494895	12.09837037
42	16	126	2.23967968	2.68044044	17.38954955
43	12	50	2.23967968	2.86802002	14.60660661
44	10	80	2.12795195	2.27863063	14.82493694
45	14	55	2.12795195	1.883996	21.80023223
46	27	152	2.07977578	1.8921962	22.27174374
47	69	114	2.07977578	2.56256256	14.66810811
48	3	80	2.07977578	2.59331331	10.56083283
49	11	83	1.9823984	2.44263463	15.74950951
50	47	97	1.9823984	2.56461261	25.7681041
51	68	95	1.86964565	5.17022623	26.27651652
52	63	150	1.72716717	2.44878478	21.63827828
53	162	148	1.70769169	2.78294294	24.75947948
54	109	55	1.70769169	2.47236036	23.33776977
55	65	80	1.97727327	2.1382022	17.52587788
56	8	192	1.84094494	2.13615215	19.00293894
57	12	71	1.84094494	1.93012212	19.58515315
58	15	76	1.84094494	2.10642643	27.73102703
59	261	82	1.64619019	3.78644244	28.35731732

Dim	DL Factor	FL Factor	c (DL)	c (FL)	c (Halton)
60	134	109	1.76714314	2.0767007	26.38106907
61	159	105	1.760993	2.181253	34.49927
62	41	173	1.64824	2.480561	26.66705
63	246	58	1.83992	2.391383	18.49043
64	13	143	1.695391	4.08165	22.95339
65	173	56	1.767143	2.44981	38.73467
66	100	177	1.779443	2.43751	38.83
67	40	203	1.755868	2.076701	25.90033
68	79	239	1.722042	2.386258	21.82586
69	226	196	1.82967	1.824545	23.36442
70	1	143	1.626715	4.374807	30.65132
71	80	278	1.835307	2.704016	43.21198
72	140	227	1.62774	1.890146	33.66387
73	52	87	1.895271	2.0316	29.02666
74	23	274	1.760993	2.419059	27.95038
75	82	264	1.760993	2.030575	31.7686
76	119	84	1.760993	2.503111	36.52779
77	382	226	1.699491	2.327832	37.60509
78	359	163	1.699491	2.139227	30.76613
79	80	231	1.699491	2.25198	34.58127
80	344	177	1.785594	2.24378	35.9015
81	54	95	1.757918	2.489786	27.85608
82	44	116	1.687191	1.955748	37.80497
83	336	165	1.676941	2.684541	39.34456
84	155	131	1.741518	2.082851	39.45629
85	239	156	1.716917	2.099251	46.36291
86	361	105	1.752793	3.162202	42.36428
87	43	188	1.746643	2.432384	42.8973
88	315	142	1.746643	2.010074	37.33961
89	92	105	1.746643	1.997774	40.95077
90	305	125	1.746643	2.490811	51.70739
91	43	269	1.746643	1.997774	51.43063
92	113	292	1.746643	2.64764	36.33611
93	159	215	1.664641	1.997774	32.13658
94	87	182	1.664641	2.213029	42.19823
95	108	294	1.687191	2.149477	41.87842
96	370	152	1.794819	2.085926	42.37351

Dim	DL Factor	FL Factor	c (DL)	c (FL)	c (Halton)
97	128	148	1.756893	2.086951	46.43671
98	94	144	1.721017	1.908597	36.91628
99	290	382	1.721017	2.109502	42.40836
100	108	194	1.781493	1.859395	38.15861
101	116	346	1.721017	2.187403	34.56692
102	80	323	1.721017	2.415984	42.84605
103	245	220	1.721017	1.82352	43.15355
104	373	174	1.736392	2.138202	48.38323
105	420	133	1.736392	2.350382	54.46573
106	158	324	1.677966	2.44776	55.50306
107	211	215	1.677966	1.954723	46.32396
108	430	246	1.767143	1.977273	46.46233
109	465	159	1.743568	4.715115	51.36195
110	337	337	1.828645	3.07815	56.52295
111	458	254	1.735367	1.995724	56.89196
112	333	423	1.681041	1.849145	52.18812
113	194	484	1.639015	2.310406	54.78861
114	233	239	1.727167	2.107451	60.92031
115	129	440	1.727167	2.292981	52.88719
116	410	362	1.622615	2.332957	44.30261
117	511	464	1.680016	2.383183	54.50673
118	524	376	1.750743	4.005798	61.97609
119	487	398	1.702567	3.148877	57.15642
120	66	174	1.697441	1.84607	54.83576

Table A.1: Factors f_j found using $c(2, \mathcal{K}_{d,w,s}; P_n)$ values and the corresponding $c(2, \mathcal{K}_{d,w,s}; P_n)$ value over all projections within window for the deterministic Halton sequence and permuted using FL factors and our factors for the first 120 dimensions.