Randomized Quasi-Monte Carlo Methods on Triangles: Extensible Lattices and Sequences

Gracia Yunruo Dong[1,2*], Erik Hintz[3], Marius Hofert[4], Christiane Lemieux[3]

[1*]Dept. of Statistical Sciences, University of Toronto, 700 University Ave, Toronto, M5G1X6, Ontario, Canada.
[2]Dept. of Mathematics and Statistics, University of Victoria, 3800 Finnerty Road, Victoria, V8P5C2, British Columbia, Canada.
[3]Dept. of Statistics and Actuarial Science, University of Waterloo, 200 University Ave W, Waterloo, N2L3G1, Ontario, Canada.
[4]Dept. of Statistics and Actuarial Science, The University of Hong Kong, Pok Fu Lam, Hong Kong.

*Corresponding author(s). E-mail(s): gracia.dong@utoronto.ca; Contributing authors: erik.hintz@uwaterloo.ca; mhofert@hku.hk; clemieux@uwaterloo.ca;

Abstract

Two constructions were recently proposed for constructing low-discrepancy point sets on triangles. One is based on a finite lattice, the other is a triangular van der Corput sequence. We give a continuation and improvement of these methods. We first provide an extensible lattice construction for points in the triangle that can be randomized using a simple shift. We then examine the one-dimensional projections of the deterministic triangular van der Corput sequence and quantify their sub-optimality compared to the lattice construction. Rather than using scrambling to address this issue, we show how to use the triangular van der Corput sequence to construct a stratified sampling scheme. We show how stratified sampling can be used as a more efficient implementation of nested scrambling, and that nested scrambling is a way to implement an extensible stratified sampling estimator. We also provide a test suite of functions and a numerical study for comparing the different constructions.

Keywords: quasi-Monte Carlo, triangular van der Corput sequence, lattice methods, stratified sampling

MSC Classification: 11K31 , 11K36 , 11K45 , 65D30 , 65D32

1 Introduction

Numerical integration over triangular areas is a subject that has been of interest for many decades, for example when applying finite element methods; see Cowper (1973) where Gaussian quadrature formulas for triangles are derived. For sampling on a two-dimensional triangle with applications in computer graphics, see Pharr (2019); Basu and Owen (2015). Rather than mapping points from the unit cube to the triangle as done in, e.g., Heitz (2019); Pillards and Cools (2005); Fang and Wang (1993); Arvo (1995), instead Basu and Owen (2015) have proposed two low-discrepancy constructions that construct points directly on the triangle. The first is based on a finite lattice, which attains a parallelogram discrepancy of $O(\log(n)/n)$, the best possible rate. Brandolini et al (2013) and work prior to this have only indicated that such a discrepancy was possible, without providing a construction. The lattice construction in Basu and Owen (2015) is not extensible in the sample sizes n, i.e., if n_1 points are sampled, but $n_2 > n_1$ points are needed, the entire lattice with n_2 points needs to be constructed rather than adding $n_2 - n_1$ points to the existing lattice. The second is a triangular van der Corput (vdC) sequence based on the one-dimensional vdC sequence in base 4 that places points in a two-dimensional triangle by recursively subdividing the triangle.

In this paper, we extend the work of Basu and Owen (2015) and explore the use of randomized quasi-Monte Carlo methods on triangles and their success in practice. Required background is reviewed in Section 2. Here, we first introduce the triangular lattice and vdC sequence of Basu and Owen (2015). In Section 2.5, we then show that the vdC sequence has poor projection properties: The one-dimensional projections of the non-randomized triangular vdC sequence contain only $2\sqrt{n} < n$ points, which can lead to poorer integration results for functions with low effective dimension. This is different from rank-1 lattices, which can be made fully projection regular.

We then provide an extensible lattice construction for points in the triangle that can be randomized with a digital shift in Section 3, making this method more applicable in practice. In Section 4, we address the issue of the poor projection properties of the triangular vdC sequence by proposing a stratified sampling scheme using the sub-triangles constructed in the vdC sequence as strata. We also show that, in general over the unit interval and not only on triangles, the stratified sampling scheme has the same distribution as the nested scrambling of the vdC 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. This connection also allows us to show that nested scrambling provides a way to implement an extensible stratified estimator.

Finally, although various constructions and mappings for sampling on the triangle exist, there is little, if any, work done comparing these different methods on numerical integration examples. This motivates the inclusion of Section 5, a numerical study on a suite of test functions on the triangle that allow us to differentiate between the performance of these constructions. Section 6 concludes this paper.

2 Background

In this section, we review quasi-Monte Carlo methods, some definitions and properties of triangles, and show how one can uniformly sample within a triangle.

2.1 Monte Carlo and quasi-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 are integrals over the s-dimensional hypercube $[0,1)^s$, which can be written as

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

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

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

where $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ are independent and identically distributed (iid) samples from the uniform distribution over $[0,1)^s$. From the Central Limit Theorem, the integration error is asymptotically normal with

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

where $\xrightarrow{D}$ denotes convergence in distribution and $\sigma^2 = I(f^2) - I(f)^2$ is the variance of f, which can be estimated by the sample variance of $f(\boldsymbol{u}_1), \ldots, f(\boldsymbol{u}_n)$. The MC integration error is $O_p(\frac{1}{\sqrt{n}})$, meaning that k^2-many samples are needed to decrease the error by a factor of k. This is one of the drawbacks of the MC method, improving the accuracy of an estimate can be computationally expensive.

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 fill the unit hypercube as homogeneously as possible. Such sets are known as *low-discrepancy sequences* or low-discrepancy point sets, as the sequences cover the area over the unit hypercube more evenly than those of a pseudo-randomly generated sequence of Uniform$[0,1)^s$ random variables and thereby minimize the discrepancy of the point set. There are two main kinds of low-discrepancy point sets: integration lattices (Sloan and Joe, 1994) such as the Kronecker lattice of Korobov (1959), and digital nets and sequences (Dick and Pillichshammer, 2010) such as the vdC sequence of van der Corput (1935). 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; see Owen (1997).

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 produces a new point set $\tilde{P}_n = \{\tilde{\boldsymbol{u}}_{\boldsymbol{1}}, \ldots, \tilde{\boldsymbol{u}}_{\boldsymbol{n}}\}$ which satisfies that each point $\tilde{\boldsymbol{u}}_i \sim \text{Uniform}([0,1)^s)$ for all i while preserving the low-discrepancy of P_n. Note that after applying the randomization function to quasi-random numbers, the generated points $\tilde{\boldsymbol{u}}_i$ are not independent across different i. 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.

In particular, with the MC estimator, one uses the fact that $\{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n\}$ are iid samples to estimate the standard error of our estimator. This cannot be done in the same way when using RQMC to estimate $I(f)$ because in that case the sample points in $\tilde{P}_n$ are not independent. Instead, as per (Lemieux, 2009, 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 $\tilde{P}_{n,l} = \{\tilde{\boldsymbol{u}}_{1,l}, \ldots, \tilde{\boldsymbol{u}}_{n,l}\}$, where $\tilde{P}_{n,1}, \ldots, \tilde{P}_{n,v}$ are v independent copies of $\tilde{P}_n$. Define the l^{th} RQMC estimator

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

which has expectation

$$\mathrm{E}(\hat{\mu}_{rqmc,l}) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{E}(f(\tilde{\boldsymbol{u}}_{i,l})) = \frac{1}{n}\sum_{i=1}^{n}\int_{[0,1)^s} f(\tilde{\boldsymbol{u}}_{i,l})\,\mathrm{d}\tilde{\boldsymbol{u}}_{i,l} = I(f)$$

as each $\tilde{\boldsymbol{u}}_{i,l} \sim \mathrm{Uniform}([0,1)^s)$. Thus, $\hat{\mu}_{rqmc,l}$ is an unbiased estimator of $I(f)$. The overall RQMC estimator of $I(f)$, based on these v iid estimators, is

$$\hat{\mu}_{rqmc} = \frac{1}{v}\sum_{l=1}^{v}\hat{\mu}_{rqmc,l}, \tag{1}$$

with variance estimator

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

The empirical variance in Equation (2) can then be compared with that of the regular MC estimator with a sample size nv.

2.2 Point sets on the triangle

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

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

Without loss of generality, we can consider $A, B, C \in \mathbb{R}^2$. 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.

Let $\triangle = \triangle(A, B, C)$ and $\triangle' = \triangle(A', B', C')$ be two arbitrary triangles. Algorithm 1 from Tymchyshyn and Khlevniuk (2019) explains how to transform a point $\boldsymbol{x} = (x, y) \in \triangle$ to $\boldsymbol{x}' = (x', y') \in \triangle'$. This transformation can be applied to every point within $\triangle$ to create a sampling scheme on $\triangle'$. When $\triangle'$ is non-degenerate, this transformation is one-to-one. The algorithm can be extended to higher dimensions, for example, to map between simplexes.

Algorithm 1 (Transforming a point from $\triangle$ to $\triangle'$)**.** *Given $\triangle = \triangle(A, B, C)$, $\triangle' = \triangle(A', B', C')$, and a point $\boldsymbol{x} = (x, y) \in \triangle$:*

1. *Define the matrices*

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

   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. *Return the point $\boldsymbol{x}' = (x', y')$ consisting of the first two components of the matrix-vector product*

$$(x', y', z') = M(\triangle, \triangle') \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3}$$

We illustrate the affine transformation in Figure 1. The first four points of the scrambled triangular vdC sequence are generated on the equilateral triangle $\triangle_E$, and then mapped using Algorithm 1 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.

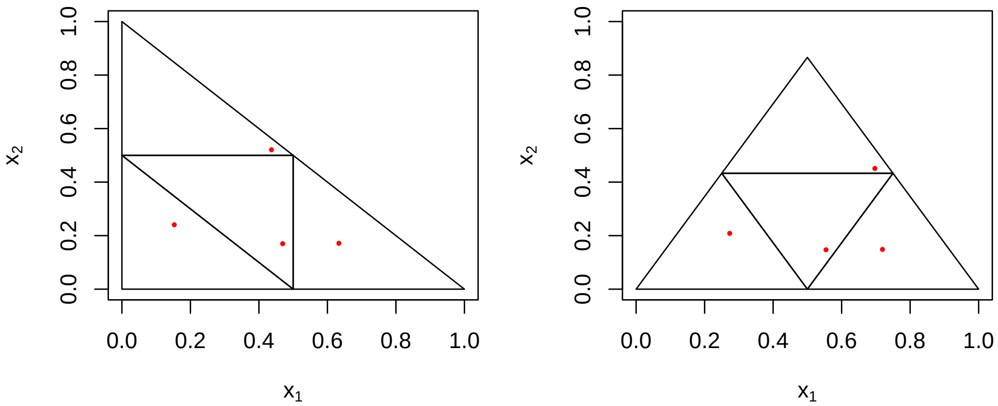

Fig. 1 Four points generated on the equilateral triangle and mapped to the right-angle triangle using Algorithm 1.

2.3 Transforming a low-discrepancy point set from the unit square to a triangle

The end goal of creating point sets over some domain Ω is often to estimate an integral over said domain, say, $\int_\Omega f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, 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 $[0,1)^d$ and to formulate this integral as

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

where $\phi : [0,1)^d \to \Omega$ for some $d \in \mathbb{N}$ is a mapping such that $\phi(\boldsymbol{U}) \sim \text{Uniform}(\Omega)$ for $\boldsymbol{U} \sim \text{Uniform}(0,1)^d$. That is, we can estimate expectations by sampling points uniformly from the domain Ω if we can find such a mapping ϕ.

For the right-angle triangle $\Omega = \triangle$ with $0 \leq x_1 \leq x_2 \leq 1$ (which is $\Omega = \triangle((0,0),(0,1),(1,1))$), Pillards and Cools (2005) give six possible transformations ϕ to map points from $\text{Uniform}[0,1)^2$ to a uniform distribution over $\triangle$. Note that this right-angle triangle $\triangle$ is not the same as $\triangle_R$, but as mentioned earlier, transforming points generated on one triangle to another triangle is straightforward.

1. Method *Drop*. Points are accepted if they are within $\triangle$ and rejected otherwise. In higher dimensions, many points get lost – only 1 in every $s!$ points are kept when working in s dimensions.
2. Method *Sort*. Points rejected by the drop method 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. Method *Mirror*. We accept 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. Method *Origami*. Here, the sort method 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 matches the unit square. This transformation is discontinuous.
5. Method *Root*. This method is based on Fang and Wang (1993) 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. Method *Shift*. This method treats the two sharp corners of the triangle the same way, in contrast with the root method. A line is drawn with slope -1 through each point, and each point is then moved halfway towards the nearest axis along this line. This transformation is fast, but is not generalisable to higher dimensions.

The "drop" method is of acceptance-rejection type in that it only keeps the points in the triangle and discards the other ones. The other methods, except "root" and "shift", can be thought of as improved variants of "drop". The method "root" is originally from Fang and Wang (1993) and is an application of the inverse Rosenblatt transform; see Rosenblatt (1952). Other methods, such as introduced in Heitz (2019), have been introduced for computer graphics applications, that transform points from the unit square to $\triangle_R$ with lower distortion than the "root" method, similarly to the "shift" method. Although the concept of "low-distortion" is not consistently defined (see Shirley and Chiu (1997)), such transformations aim to preserve correlation between points.

In our numerical experiments of Section 5, we use the method "root" on the bivariate Sobol' sequence as well as on pseudo-randomly generated numbers. We use this method as it is smooth, continuous, fast and can be extended to go from higher-dimensional cubes to simplexes.

2.4 The triangular rank-2 lattice

In this section, we describe lattice constructions in general, and then the triangular lattice of Basu and Owen (2015). First, we define a *lattice point set*.

$$P_n = \left\{ \left(\frac{i_1 \boldsymbol{z}_1}{n_1} + \frac{i_2 \boldsymbol{z}_2}{n_2} + \cdots + \frac{i_r \boldsymbol{z}_r}{n_r} \right) \bmod 1,\ 0 \le i_l < n_l, l = 1, \right\},$$

where the basis vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_r \in \mathbb{R}^s$. A lattice rule is of rank t if it can be expressed in this form with $r = t$, but not $r < t$. In particular, for rank-1 lattices, we can write

$$P_n = \left\{ \left(\frac{iz_1}{n}, \frac{iz_2}{n}, \ldots, \frac{iz_s}{n} \right) \bmod 1,\ 0 \le i_l < n_l, l = 1, \right\},$$

based on a single generating vector $\boldsymbol{z} = (z_1, \ldots, z_s)$. Rank-1 lattices have an advantage over higher-order lattices, as they can be made fully projection regular, i.e., each one-dimensional projection has n points. One way of constructing higher rank lattices is to employ the copy rules of Disney and Sloan (1992); that is, we can take a rank-1 lattice, scale it, and copy it into each of the 2^s subcubes obtained by partitioning the unit cube into two parts on each side, and obtain a rank-s lattice. Thus, rank-1 lattices are more often used. There is extensive work on choices of rank-1 lattices, both theoretical and computer searches for good lattices are available; see for example, Dick et al (2022); Goda and L'Ecuyer (2022).

Basu and Owen (2015) give a lattice construction for points on the triangle with optimal discrepancy: let $\alpha \in (0, 2\pi)$ be 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. The point set P_n obtained by rotating the lattice $(2n)^{-1/2}\mathbb{Z}^2$ counterclockwise by α and intersecting with $\triangle$ satisfies $D_n^P(P_n, \triangle_R) \le C \log n/n$. The following algorithm produces such point sets.

Algorithm 2 (Lattice construction of Basu and Owen (2015))**.** *Given the target sample size n, α such that $\tan(\alpha)$ is badly approximable in the sense of (Basu and Owen, 2015, Definition 4.1) (e.g., $\alpha = 3\pi/8$), a random vector $\boldsymbol{U} \sim \mathrm{Uniform}(0,1)^2$ (for an optional shift), an integer N, sample n points in $\triangle_R$ as follows:*

1. *Let $P = \{0, 1, \ldots, N-1\}^2$ and set $\boldsymbol{x} \leftarrow (\boldsymbol{x} + \boldsymbol{U})/N$ for $\boldsymbol{x} \in P$.*
2. *Map $\boldsymbol{x} \leftarrow 2\boldsymbol{x} - 1 \in [-1, 1]^2$ for all $\boldsymbol{x} \in P$.*

3. *Set* $\boldsymbol{x} \leftarrow \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \boldsymbol{x}$ *for all* $\boldsymbol{x} \in P$.

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.

6. *Return* P_n.

The randomization transforms the otherwise deterministic points so that $\boldsymbol{x} \sim \text{Uniform}(\triangle)$ for all $\boldsymbol{x} \in P$, which follows readily by observing that the density of each $\boldsymbol{x}$ is constant if a randomization is performed. In Step 5, we can choose arbitrarily which points to add or remove. We remark that the algorithm presented in (Basu and Owen, 2016, p. 757) differs slightly from Algorithm 2 in that their version uses $N = \lceil \sqrt{2n} \rceil + 1$ and the lattice $\{-N, \dots, N\}^2$.

2.5 The triangular vdC sequence

We now describe the triangular vdC sequence of Basu and Owen (2015), which is based on the one-dimensional vdC sequence in base 4. The i^{th} point of the one-dimensional vdC 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 vdC sequence places points at the left-most boundaries of each of the intervals $[b^{-m}, b^{-m+1})$. Similarly, the triangular vdC sequence, based on the vdC 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. Let $T = \triangle(A, B, C)$ denote the specific 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

$$T(d) = \begin{cases} \triangle\left(\frac{B+C}{2}, \frac{A+C}{2}, \frac{A+B}{2}\right), & d=0, \\ \triangle\left(A, \frac{A+B}{2}, \frac{A+C}{2}\right), & d=1, \\ \triangle\left(\frac{B+A}{2}, B, \frac{B+C}{2}\right), & d=2, \\ \triangle\left(\frac{C+A}{2}, \frac{C+B}{2}, C\right), & d=3. \end{cases} \tag{4}$$

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 vdC sequence obtains the i^{th} triangular point by mapping the integer i to the midpoint of the triangle $T(d_0, \ldots, d_{K_i})$, which is recursively defined by $T(d_k, d_{k+1}) = (T(d_k))(d_{k+1})$, as detailed in Algorithm 3. For example, if we have $T = \triangle_R = \triangle((0,0),(0,1),(1,0))$, then $T(2,3) = (T(2))(3)$, where $T(2) = \triangle((0,0.5),(0,1),(0.5,0.5))$. Then, $(T(2))(3) = \triangle((0.25,0.5),(0.25,0.75),(0.5,0.5))$.

Algorithm 3 (Triangular vdC sequence)**.** *Given the desired sample size n and a target triangle $\triangle(A,B,C)$, the first n points, outputted in an $n \times 2$ array $\boldsymbol{x}$, are generated 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 (4).
   (d) *Set the i^{th} coordinate $\boldsymbol{x}[i]$ of $\boldsymbol{x}$ as $\boldsymbol{x}[i] = \mathrm{mdpt}(T)$, where the midpoint function is $\mathrm{mdpt}(\triangle(A,B,C)) = (A+B+C)/3$ component-wise.*
2. *Return $\boldsymbol{x} = (x_1, \ldots, x_n)$.*

In other words, Algorithm 3 generates a vdC 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 vdC 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, as shown in Proposition 1. 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 onto these axes.

Proposition 1 (Non-unique one-dimensional projections for the triangular vdC sequence)**.** *Let $T = \triangle_R$, such as in Figure 2, and denote by P_n the point set consisting of the first n points produced by Algorithm 3 for $n = 4^k$ and $k > 2$. Then the projections of P_n onto the x- and y-axis contain $2\sqrt{n} = 2^{k+1} < n$ points.*

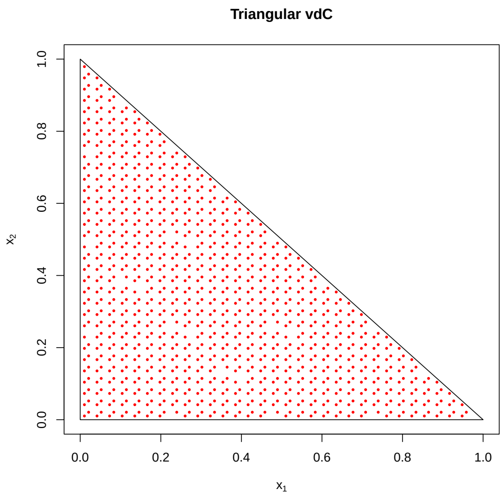

Fig. 2 The first 1000 points of the triangular vdC sequence of Basu and Owen (2015)

Proof. Since $n = 4^k$, each of the 4^k subtriangles contains one point, and there are 2^k rows of subtriangles. In each row j, the midpoints of all upright triangles have the same y coordinate, say y_{j1}. Similarly, the midpoints of all inverted triangles in the same row have the same y coordinate, say y_{j2}. Since there are only these two cases, the point set projects on $\{y_{ji} : j = 1, \ldots, 2^k; i = 1, 2\}$, which has $2 \cdot 2^k = 2^{k+1}$ elements. The x-axis case follows similarly. □

This behaviour where the one-dimensional projections contain non-unique points can also be observed in the equilateral triangle, as seen in Figure 6.

As a continuation of this work, Goda et al (2017) generalize the triangular vdC sequence by replacing the support $\{0, 1, 2, 3\}$ of the transformation in (4) by $\{(0, 0), (1, 0), (0, 1), (1, 1)\} = \mathbb{F}_2^2$, where the input strings (from $\mathbb{F}_2^m$ for some m or $\mathbb{F}_2^\infty$) come from a digital net. They prove that their construction gives worst case error in $\mathcal{O}((\log n)^3/n)$ for functions in $C^2(\triangle)$. Furthermore, they show that their construction includes the vdC sequence of Basu and Owen (2015); see (Goda et al, 2017, p. 369).

3 Extensible triangular lattice constructions

We now describe our extensible triangular rank-1 and rank-2 constructions, as an extension of the triangular lattice construction. The construction in Algorithm 2 is non-extensible. In this section, we propose an extensible scheme, for which we make use of the one-dimensional vdC sequence. Recall that the i^{th} point of an extensible rank-1 lattice sequence with generating vector $\boldsymbol{z} \in \mathbb{Z}^d$ is defined as

$$\boldsymbol{u}_i = \phi_b(i)\boldsymbol{z} \bmod 1 \in [0, 1]^d,$$

where $\phi_b(i)$ is the i^{th} term of the vdC sequence in base b, and the modulus 1 operation is applied component-wise; see Hickernell and Hong (1997); Hickernell et al (2001).

If we want to use this idea to define a triangular Kronecker lattice sequence, then we need to introduce a more general idea extending the rank-1 case. In particular, the grid that needs to be generated, given by

$$P = \left\{ -\frac{N}{N}, -\frac{N-1}{N}, \ldots, -\frac{1}{N}, 0, \frac{1}{N}, \ldots, \frac{N}{N} \right\}^2,$$

is a rank-2 lattice rather than a rank-1 lattice. We also note that the only reason why Basu and Owen (2015) use a grid over $[-1, 1]^2$ instead of $[0, 1)^2$ seems to be that when they rotate the points from the grid, they can just take the intersection with $\triangle_R$ without having to perform any modulo 1 operation. However, with our proposed approach to generate an extensible grid, we will focus on generating points in $[0, 1)^2$; we can then either make use of modulo 1 operations or extend the grid to $[-1, 1]^2$.

First, we must choose the base b in which the grid will be constructed. The choice of the base b is such that whenever n is of the form b^{2k} for some $k \geq 1$, the point set P_n obtained with this method will be exactly the same as if we had proceeded with the fixed-size approach based on $N = b^k$; see Proposition 2 below. We then make use of the base b decomposition of i as $i = \sum_{j \geq 0} d_j b^j$ to define the point

$$\boldsymbol{u}_i = (u_{i,1}, u_{i,2}) = \frac{d_0}{b}(1, 1) + \frac{d_1}{b}(0, 1) + \frac{d_2}{b^2}(1, 1) + \frac{d_3}{b^2}(0, 1) + \ldots \bmod 1. \qquad (5)$$

Note that the modulo 1 operation is only necessary for the second coordinate, as we have the bound

$$\frac{d_0}{b} + \frac{d_2}{b^2} + \frac{d_4}{b^3} + \ldots \leq \sum_{j \geq 1} \frac{d_j}{b^k} \leq \sum_{j \geq 1} \left(\frac{1}{b}\right)^k = \frac{1}{1 - 1/b} - 1 \leq 1.$$

Figure 3 shows the first 9, 50, and 81 points obtained with this method when $b = 3$.

The points $\boldsymbol{u}_i$ lie in $[0, 1)^2$. Next, we rotate the points by α and can then proceed as in Algorithm 2.

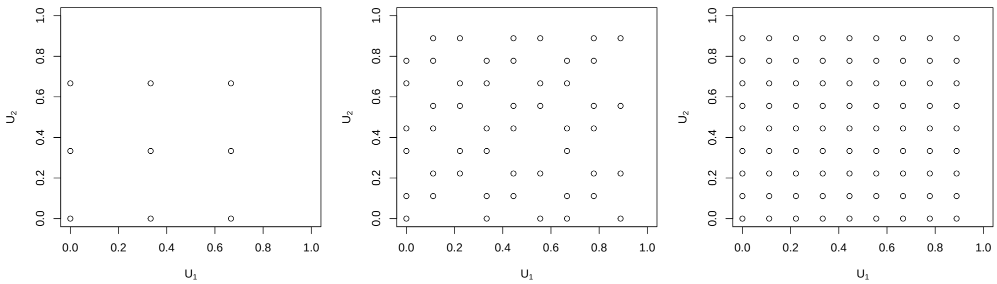

Fig. 3 First 9 (left), 50 (middle) and 81 points when using $b = 3$.

Algorithm 4 (Extensible Kronecker Lattice)**.** *Given* α, n, b*, a random vector* $\boldsymbol{U} \sim \mathrm{Uniform}(0,1)^2$ *(for the optional shift) and a skip* $s \geq 0$*, sample* n *points in* $\triangle_R$ *as follows:*

1. *Set* $P_n = \{\}$ *and* $k = 0$.
2. *While* $|P_n| < n$,
   (a) *Compute digits* d_j *such that* $k + s = \sum_{j \geq 0} d_j b^j$ *and compute* $\boldsymbol{u}$ *in* (5).
   (b) *Set* $k \leftarrow k + 1$, $\boldsymbol{u} \leftarrow 2((\boldsymbol{u} + \boldsymbol{U}) \mod 1) - 1 \in [-1, 1]^2$.
   (c) *Set* $\boldsymbol{u} \leftarrow \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \boldsymbol{u}$.
   (d) *If* $\boldsymbol{u} \in \triangle_R$*, set* $P_n = P_n \cup \{\boldsymbol{u}\}$.
3. *Return* P_n.

Proposition 2 (Equivalence of Algorithms 2 and Algorithm 4)**.** *Let* $n = b^{2k}$*,* $k > 0$ *and* $s = 0$*. The sets* P_n *of points obtained by Algorithm 2 with* $N = b^k$ *and Algorithm 4 coincide.*

Proof. Assume without loss of generality that $\boldsymbol{U} = \mathbf{0}$. Consider the set $Q = \{\boldsymbol{u}_i : i = 0, \ldots, n-1\}$ where the $\boldsymbol{u}_i$ are as in (5). Then Q is a rank-2 lattice and can be written as $Q = \{(i/b^k, j/b^k) : 0 \leq i, j < b^k\}$. This is exactly the rectangular grid in Step 1 of Algorithm 2. Since the remaining operations (intersection and rotation) are identical in both algorithms, the result follows. □

Algorithm 4 is based on a rank-2 lattice with generating vectors $\boldsymbol{z}_1 = (1, 1)$ and $\boldsymbol{z}_2 = (0, 1)$. In our numerical experiments, we also consider a rank-1 lattice with generating vector $\boldsymbol{z} = (1, 182667)$; see (Cools et al, 2006, p. 26). The lattice construction allows for an easy randomization by shifting the underlying grid by a uniform vector, as originally proposed by Cranley and Patterson (1976). Figure 4 displays five independently randomized copies of the lattice points, each having a different colour.

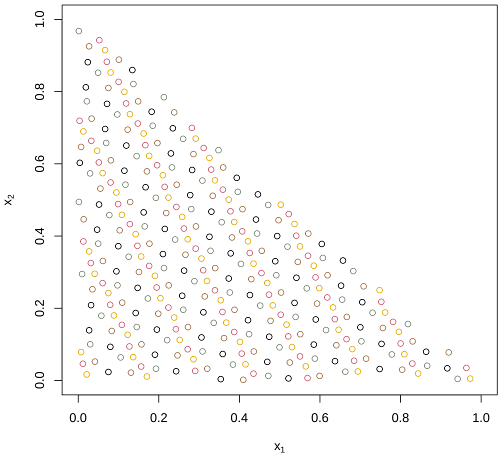

Fig. 4 Five independently randomized triangular Kronecker lattice point sets with 2^6 points each.

4 Stratified sampling based on the triangular vdC sequence

As previously mentioned, placing the sampling points at the centre of each terminal sub-triangle for the triangular vdC sequence results in points being mapped to the same coordinate on the x or y axis. Since the triangular vdC sequence is based on the one-dimensional vdC sequence over the unit interval, a natural randomization to "repair" the poor projection quality is scrambling.

In this section, we explain the equivalency between stratified sampling and the nested scrambling of Owen (1995) for any $(0, m, 1)$-net or $(0, 1)$-sequence. This equivalency is not limited to the vdC sequence, but will hold in general for all $(0, 1)$-sequences, including, for example, one-dimensional projections of the Sobol' sequence.. This will allow us to then propose an efficient implementation of nested scrambling via a stratified estimator. We also show that nested scrambling is a way to implement an extensible stratified estimator. Finally, since the triangular vdC sequence is essentially a mapping from the one-dimensional vdC sequence to the two-dimensional space, it is easy to transform our proposed scrambling implementation in one dimension into a randomization method for the triangular vdC sequence.

4.1 Stratified Sampling

A simple way to reduce the variance of an estimator for the integral of a function f over the unit interval $[0, 1)$ is to stratify $[0, 1)$ into M subintervals of length p_j and allocate n_j points to the jth subinterval (with $n_1 + \ldots + n_M = n$), thus resulting in the estimator

$$\hat{\mu}_{strat} = \sum_{j=1}^{M} p_j \frac{1}{n_j} \sum_{i=1}^{n_j} f(u_j^i),$$

where the u_j^i are iid uniforms in the jth subinterval, and independent from u_k^ℓ for $k \neq j$, $\ell = 1, \ldots, n_k$. If we allocate a number of points proportional to the length of each subinterval (i.e., $n_j = np_j$ which is assumed to be an integer), then this is stratified sampling with proportional allocation. This method is guaranteed to have a lower variance than regular Monte Carlo; see (Lemieux, 2009, p. 126). Furthermore, if $n = b^m$ and we set $M = b^m$ then one point $u_j^1 = u_j$ is allocated to the jth stratum, and we get $\hat{\mu}_{strat} = \sum_{j=1}^{n} f(u_j)/n$ where each u_j is uniformly distributed in the jth stratum. The variance of this estimator is $\mathrm{Var}(\hat{\mu}_{strat}) = \frac{1}{n^2} \sum_{j=1}^{n} \sigma_j^2$, where σ_j^2 is the variance of f 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 subinterval is different by at most one from the number of points within any other subinterval. That is, $|n_j - n_\ell| \leq 1$ for all $j, \ell = 1, \ldots, b^m$. Stratified sampling with this property can yield a more efficient implementation of nested scrambling, which we explain in the next section.

4.2 Nested Uniform Scrambling

The nested uniform scrambling method of Owen (1995), sometimes referred to as "Owen's scrambling" or "nested scrambling", scrambles a point set $P_n \subseteq [0,1)^s$ by applying random permutations to the digits $u_{i,j,l}$ that stem from the base b expansion of $u_{i,j}$ (we write $u_{i,j} = \sum_{l=1}^{\infty} u_{i,j,l} b^{-l}$). Permutations π are randomly uniformly 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 coordinate. 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. That is, to randomize the k^{th} digit of a given coordinate j, up to b^{k-1} different permutations may be needed.

Nested scrambling is costly to implement, both in terms of time and memory. It requires a dictionary or other lookup data structure to store all the permutations generated, and, in addition to the time to go through all the coordinates of every point, the lookup time for each permutation needs to be paid. However, despite being costly to implement, nested scrambling is often used because it has the potential to reduce the variance of the RQMC estimator in Equation (1) to $O(n^{-3} \log(n)^{s-1}) \approx$

$O(n^{-3+\epsilon})$ for sufficiently smooth functions as shown in Owen (1997). Use of nested scrambling is further justified by the fact that nested scrambling in base b also preserves equidistribution in base b as defined below, and satisfies the requirement of being a base b-digital scramble as defined in Hickernell (1996); Hong and Hickernell (2003); Owen (2003); Wiart et al (2021), which is also given below.

Definition 1 (Equidistribution). *We say that P_n with n of the form b^m with $m \geq 0$ for a single-base b construction, is $(k_1, \ldots, k_s)$-*equidistributed in base b *if every* elementary $(k_1, \ldots, k_s)$-interval *of the form*

$$I_{\boldsymbol{k}}(\boldsymbol{a}) = \prod_{\ell=1}^{s} \left[\frac{a_\ell}{b^{k_\ell}}, \frac{a_\ell + 1}{b^{k_\ell}} \right)$$

for $0 \leq a_\ell < b^{k_\ell}$ contains exactly $n b_1^{-k_1} \cdots b_s^{-k_s}$ points from P_n, assuming $\boldsymbol{k}$ is such that $n \geq b_1^{k_1} \ldots b_s^{k_s}$.

Definition 2 ((t, m, s)-nets). *We say that a point set P_n in base b has a* quality parameter t *if P_n is $(k_1, \ldots, k_s)$-equidistributed for all s-dimensional vectors of non-negative integers $\boldsymbol{k} = (k_1, \ldots, k_s)$ such that $k_1 + \ldots + k_s \leq m - t$. We then refer to P_n as a (t, m, s)-net in base b.*

Definition 3 ((t, s)-sequences). *We say a sequence is a (t, s)-sequence in base b if for every integer $m \geq 0$, every point set of the form $\boldsymbol{u}_j, \ldots, \boldsymbol{u}_{j+b^m-1}$, where j is of the form $j = vb^m + 1$ for some $v \geq 0$, is a (t, m, s)-net.*

Definition 4 (Base b-digital scramble). *A randomization $\mathcal{S}$ is a base b-digital scrambling if the following two properties hold: Let $U_{i,\ell} = \sum_{r=1}^{\infty} U_{i,\ell,r} b^{-r}$, that is, $U_{i,\ell,r}$ represents the r^{th} digit in the base b expansion of the ℓ^{th} coordinate of the i^{th} point $\boldsymbol{u}_i$ in the scrambled point set $\tilde{P}_n$. Then we must have:*

1. *Each $\boldsymbol{u}_i \sim$ Uniform$([0, 1)^s)$;*

2. *For two distinct points $\boldsymbol{u}_i = \mathcal{S}(\boldsymbol{v}_i), \boldsymbol{u}_j = \mathcal{S}(\boldsymbol{v}_j)$ and for each coordinate $\ell = 1, \ldots, s$, if the two deterministic points $V_{i,\ell}, V_{j,\ell}$ have the same first r digits in base b and differ on the $(r+1)^{th}$ digit, then:*
   (a) *the scrambled points $(U_{i,\ell}, U_{j,\ell})$ also have the same first r digits in base b;*
   (b) *the pair $(U_{i,\ell,r+1}, U_{j,\ell,r+1})$ is uniformly distributed over $\{(k_1, k_2), 0 \leq k_1 \neq k_2 < b\}$;*
   (c) *the pairs $(U_{i,\ell,v}, U_{j,\ell,v})$ for $v > r+1$ are mutually independent and uniformly distributed over $\{(k_1, k_2), 0 \leq k_1, k_2 < b\}$.*

4.3 Nested Scrambling as Stratified Sampling

We now show that we can implement a nested scrambled vdC sequence via a base b stratified sampling scheme on the unit interval, as the two methods produce point sets with the same joint distribution. First, we show that the scrambled vdC sequence in base b is a $(0, 1)$-sequence in base b. Since scrambling in the constructing base does not change the t parameter, it is sufficient to show that the deterministic vdC sequence is a $(0, 1)$-sequence.

Lemma 1 (A consecutive subset of the vdC sequence is a $(0, m, 1)$-net)**.** *Any point set P_n that is made up of $n = b^m$ consecutive points from the vdC sequence constructed in base b is a $(0, m, 1)$-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; see (Dong and Lemieux, 2022, Section 5). That is, there is exactly one point in each of the m-elementary intervals and thus, by definition, a $(0, m, 1)$-*net* in base b, and thus P_n, is m-equidistributed in base b. □

Lemma 2 (A consecutive subset of a vdC sequence has the same properties as the initial portion of a $(0, 1)$-sequence)**.** *Any point set P_n that is made up of n consecutive* *points from a vdC sequence in base b has the same equidistribution properties as the first n points of a $(0,1)$-sequence in base b.*

Proof. The result follows by observing that every point set P_k made up of $k = b^m$ consecutive points from the vdC sequence in base b is a $(0, m, 1)$-net in base b (by Lemma 1). □

Lemma 3 (Equivalence of the scrambled vdC sequence and stratified sampling for $n = b^m$)**.** *Let $\hat{\mu}_{scr,n} = \frac{1}{n}\sum_{i=1}^{n} f(\tilde{u}_i)$ where $\tilde{u}_i$, $i = 1 \ldots n$, are the first n points of a scrambled vdC 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 first b^m points of the scrambled vdC sequence in base b is a $(0, m, 1)$-net in base b (by Lemma 1), its equidistribution properties mean that there is exactly one point in each of the b^m subintervals, and the scrambling as defined in Definition 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 this case, we argue that the equidistribution properties of the $(0,1)$-sequence are such that the scrambled vdC estimator (by Lemma 2) places a number of points in the intervals $[jb^{-l}, (j+1)b^{-l})$ that are different by at most one from each other for $l \leq m$ for $m = \lfloor \log_b n \rfloor$.

To explain how to implement this estimator using our base b stratified sampling algorithm, we introduce the following notation. Write $n = \lambda b^m + r$, with the quotient $\lambda = \lfloor n/b^m \rfloor$ and remainder r satisfying $0 \leq r < b^m$. Further, break down r as $r = kb + j$, where $0 \leq j < b$. We also define M to be the smallest integer power of b greater than or equal to n. That is, $M = b^q$, where $q = \lceil \log_b n \rceil$.

In Proposition 3 we will prove that we can implement $\hat{\mu}_{scr,n}$ for any sample size n as a base b stratified sampling estimator over strata of the form $[jb^{-l}, (j+1)b^{-l})$ for $l = 1, \ldots, m$. This is done by using the vector $[N_1, \ldots, N_M]$, where N_j is the number of points in $[jb^{-q}, (j+1)b^{-q})$ (either 0 or 1). That is, N_j counts the number of points 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 vdC estimator satisfies the following 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 procedure for sampling a vector of N_j's given a base b and a number of points n in Algorithm 5 (the latter samples $N_1, \ldots, N_M$ for computing $\hat{\mu}_{scr,n}$).

Constant	Definition
n	Number of points to generate (input)
b	Base (input)
q	$q = \lceil \log_b n \rceil$
m	$m = \lfloor \log_b n \rfloor$
M	$M = b^q$.
λ	$\lfloor n/b^m \rfloor$
r	$r = n - \lambda b^m$
k	$k = \lfloor \log_b r \rfloor$
j	$j = r - kb$

Table 1 Constants used in Algorithm 5.

Algorithm 5 (Sampling $N_1, \ldots, N_M$ for computing $\hat{\mu}_{scr,n}$)**.** *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 summarized in Table 1.*
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 $m > 0$, 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 indices j 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.*
5. *Return $N_1, \ldots, N_M$.*

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 on 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 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's using Algorithm 5, 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 3 (Equivalence of stratified sampling and the one-dimensional vdC sequence)**.** *The point set created using Algorithm 5 to define the N_j's is such that pairs of distinct points have the same joint distribution as those coming from a scrambled vdC point set and thus the corresponding estimators have the same first two moments.*

Proof. We must derive the joint pdf of pairs of randomly chosen distinct points from the point set created using Algorithm 5, and show that it is the same as the joint pdf of pairs of randomly chosen distinct points from a one-dimensional scrambled vdC point set. To do so, we need the following definitions taken from Wiart et al (2021); Dong and Lemieux (2022):

Definition 5 ($\gamma_b(x,y)$)**.** *For $x, y \in [0,1)$, let $\gamma_b(x,y) \geq 0$ be the exact number of initial digits shared by x and y in their base b expansion, i.e. the smallest number $i \geq 0$ such that $\lfloor b^i x \rfloor = \lfloor b^i y \rfloor$ but $\lfloor b^{i+1} x \rfloor \neq \lfloor b^{i+1} y \rfloor$. If $x = y$ then we let $\gamma_b(x,y) = \infty$.*

Definition 6 (Counting numbers $N_b(i; P_n)$ and $M_b(k; P_n)$)**.** *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 $\gamma_b(U_l, U_j) = i$,*
2. *$M_b(k; P_n)$ is the number of ordered pairs of distinct points (U_l, U_j) in P_n such that $\gamma_b(U_l, U_j) \geq k$, and*
3. *$N_b(k; P_n, U_l) = \sum_{e \in \{0,1\}} (-1)^{|e|} M_b(k+e; P_n, U_l)$.*

(Proof, continued.) The joint pdf of a one-dimensional scrambled vdC is

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

where P_n refers to the first n points of the deterministic vdC sequence.

We now show that the scrambled vdC point set and the point set constructed using the proposed base b stratified sampling method from Algorithm 5 yield the same $M_b(k; P_n)$ as defined in Definition 6. This means that $N_b(i; P_n)$ and thus the joint pdf $\psi(x,y)$ for all pairs of points (x,y) are then the same for both methods.

Since the base b vdC sequence is equivalent to a one-dimensional Halton sequence, we know from Dong and Lemieux (2022) that in this case $M_b(k; P_n)$ is given by:

$$_{vdC}M_b(k; P_n) = \left\lfloor \frac{n-1}{b^k} \right\rfloor \left(2n - \left\lfloor \frac{n-1}{b^k} \right\rfloor b^k - b^k \right). \tag{6}$$

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 construction of the base b 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 $r = n - qb^k$, we have $_{SS}M_b(k; P_n) = 2nq - q^2 b^k - qb^k$ for the point set created via base b stratified sampling. Now, we substitute $q = \lfloor \frac{n}{b^k} \rfloor$ and simplify to 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 (6), except with $\lfloor \frac{n-1}{b^k} \rfloor$ rather than $\lfloor \frac{n}{b^k} \rfloor$. We show that $_{SS}M_b(k; P_n) = {}_{vdC}M_b(k; P_n)$ by considering the following two cases:

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

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

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 that ${}_{SS}N_b(k;P_n) = {}_{vdC}N_b(k;P_n)$ for all k. This implies that ${}_{SS}\psi(x,y) = {}_{vdC}\psi(x,y)$ for all $(x,y) \in [0,1)^2$. Thus, the estimator implemented using the base b stratified sampling has the same first two moments as the scrambled vdC estimator. □

Since n is not necessarily an integer power of b, our estimator does not inherit properties of stratified sampling with proportional allocation. However, its connection to nested scrambling implies we can apply results about the variance of scrambled estimators for $(0,1)$-sequences as found, e.g., in Gerber (2015). That is, even though the variance is not guaranteed to be no higher than for Monte Carlo for any function (as a purely stratified sampling estimator based on proportional allocation would), the superior asymptotic bounds for the variance of a scrambled estimator apply. Also, since we know the scrambled vdC estimator produces an unbiased estimator (see Dong and Lemieux (2022)) this means Algorithm 5 also produces an unbiased estimator.

4.4 Comparative efficiency analysis for fixed n

Using the above connection between scrambling and base b stratified sampling, for fixed n we can implement scrambling by sampling the N_j's as in Algorithm 5 and once those 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 5 for base b 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 vdC 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 of the base b stratified sampling algorithm is that it does not require any additional storage for a lookup dictionary, as the nested scrambling algorithm does.

Figure 5 shows the runtime needed to generate $n = 2000, 4000, \ldots, 200\,000$ points from the scrambled base 4 vdC sequence needed to construct point sets on the triangle. Our base b stratified sampling implementation is compared with nested scrambling and shown to be more computationally efficient. Furthermore, an increase in runtime occurs only at integer powers of 4. We also report, in the legend, a growth rate of the runtime α, such that the runtime is proportional to n^α. This is estimated by the regression coefficient α of $\log(\text{runtime}) = \alpha \log(n) + c$. α is estimated to be 1.07 for stratified sampling and 1.21 for nested scrambling, showing that, indeed, stratified sampling has close to linear time complexity while nested scrambling has a higher time complexity.

4.5 From the one-dimensional vdC sequence to the triangular vdC sequence

It is now straightforward to use the ideas presented thus far for the one-dimensional vdC sequence to construct a scheme to sample n points on an arbitrary triangle. The M "strata" are now the sub-triangles, and we can use Algorithm 5 to sample the number of points in each sub-triangle, say $N_1, \ldots, N_M$. Then, sample N_j (either 0 or

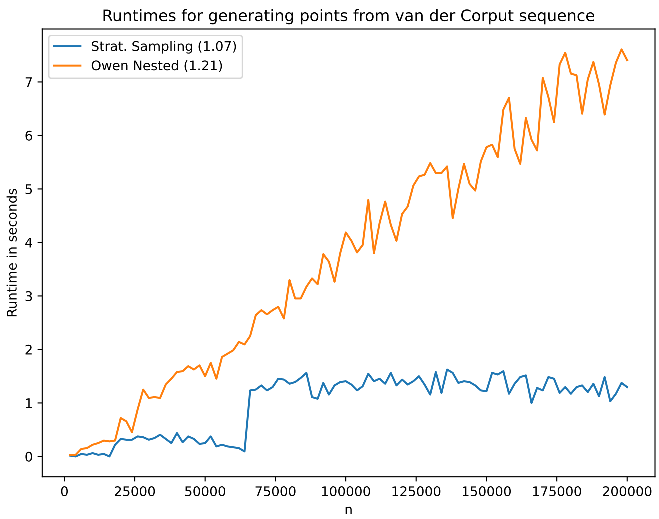

Fig. 5 Runtime comparison for generating a scrambled vdC sequence in base 4 using Owen's Scrambling vs Stratified Sampling. Estimated growth rate for the runtimes are in parentheses in the legends.

1) points uniformly in each sub-triangle for $j = 1, \dots, M$, for example, using one of the methods described in Section 2.3. This estimator can also be extended using the method described in Appendix A.

An algorithm similar to Algorithm 3 can be used to sample in the triangle – the steps are described in Algorithm 6. 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 6 (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 $\boldsymbol{x}$, as follows:*

1. *Generate a vector of indexes $I_1, \dots, I_n$ using Algorithm 5 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 (4);
   (d) *Set the i^{th} point $\boldsymbol{x}[i]$ to be a random uniformly sampled point within T. In our implementation, we use the "root" method as described in Section 2.3;*
3. *Return $\boldsymbol{x}$.*

This avoids the poor projection properties of the original triangular vdC sequence as seen in Figure 6, and it provides a way to randomize the point set to allow for error estimation.

We note that this method of mapping points from the one-dimensional vdC sequence to the two-dimensional triangle can be done in other bases – for any base b such that b is a perfect square, we can map a scrambled vdC sequence in base b to the triangle, which is simply divided into b sub-triangles of equal size. The enumeration of the sub-triangles does not matter – after scrambling, (or, equivalently, stratified sampling), the points are uniformly distributed between the sub-triangles.

Proposition 4 (Unbiasedness of $\hat{\mu}_{scr,n}$)**.** *The estimator $\hat{\mu}_{scr,n}$ is unbiased.*

Proof. The statement is equivalent to showing that each point is marginally uniformly distributed over the triangle. Without loss of generality, the marginal pdf f for the

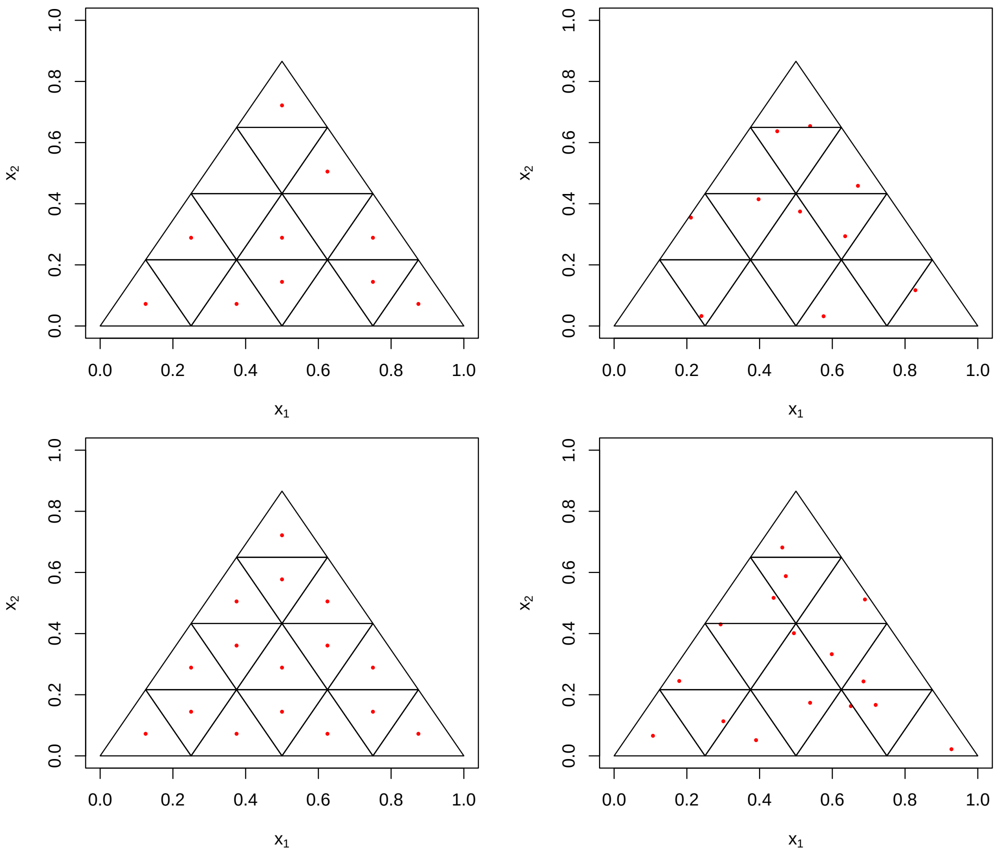

Fig. 6 Examples of the triangular vdC points generated on $\triangle_E$, with $n = 10$ (top) and $n = 16$ (bottom). The images on the left have each point at the centre of the terminal sub-triangle, while the images on the right have the points scrambled. For the $n = 10$ case, the sub-triangles with points are selected via stratified sampling.

point $\boldsymbol{u} = (u_1, u_2)$ uniformly distributed over $\triangle_E$ is:

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

For the stratified sampling estimator with given $n \geq 1$, 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 share the same marginal distribution), and we uniformly sample within each of the n sub-triangles, the value of the pdf f 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 $g(\boldsymbol{u})$ given by

$$g(\boldsymbol{u}) = \begin{cases} \frac{\sin(\pi/3)4^q}{2\times\#\text{ sub-triangles}}, & \text{if } \boldsymbol{u} = (u_1, u_2) \in \triangle_E, \\ 0, & \text{otherwise,} \end{cases}$$
$$= \begin{cases} \frac{2}{\sin(\pi/3)}, & \text{if } \boldsymbol{u} = (u_1, u_2) \in \triangle_E, \\ 0, & \text{otherwise.} \end{cases}$$

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

5 Numerical experiments

Now that we have described our proposed lattice and randomized triangular vdC constructions for a point set on the triangle, we compare their performance on numerical integration problems with existing constructions. 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.

To test the performance of the different triangular constructions, we consider the following 2-dimensional test functions over the right-angle triangle $\triangle_R$ with corners at (0, 0), (0, 1), (1, 0):

1. $f_1(x, y) = ((|x - \beta| + y)^d + (|y - \beta| + x)^d)/2$ This function has two singularities, so we anticipate this function to be harder to integrate than the others. This function is based on $f(x, y) = (|x - \beta| + y)^d$ from Pillards and Cools (2005). The 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 Pillards and Cools (2005). This is a smooth oscillatory function. The 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. The integral evaluates to $\frac{2}{(\alpha_3+1)(\alpha_3+2)}$ over $\triangle_R$.

We use $\beta = 0.4$, $d = -0.9$, $\alpha_1 = e^3$, $\alpha_2 = e^2$, $\alpha_3 = 2.5$ and estimate $\mu_j = \int_{\triangle_R} f_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, whose theoretical values are known for $j = 1, 2, 3$. Figure 7 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 $\triangle_R$ only.

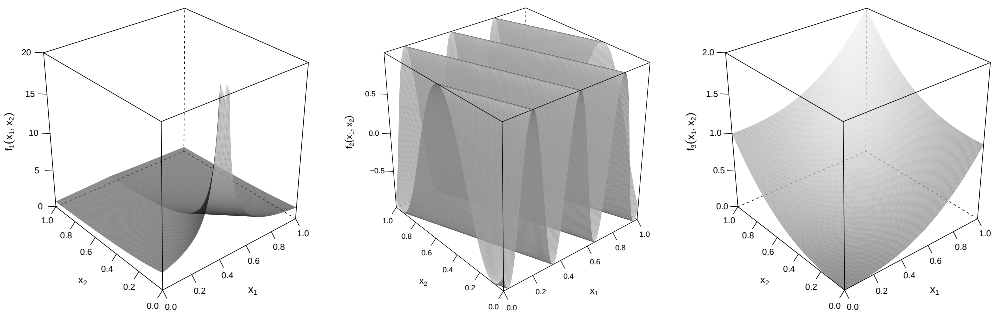

Fig. 7 Test functions f_1 (left), f_2 (middle) and f_3 (right) used in the numerical study.

We also estimate the value at time 0 of a European basket call option with maturity $T = 1$ based on two underlying assets that follow a lognormal distribution each. Formally, the value of the option is

$$C_0 = \mathrm{E}\left[\max\left(0, e^{-rT}\left(\frac{1}{2}\sum_{j=1}^{2} S_j(T) - K\right)\right)\right],$$

where $S_j(T)$ is the price of the asset j at maturity $T = 1$. The lognormal model means we can write $S_j(T) = S_j(0)e^{(r-\sigma^2/2)T+\sigma\sqrt{T}Z}$, where $Z \sim \mathrm{N}(0, 1)$ is a standard normal random variable.

For our applications, we use the following parameters: the strike price K is either 45, 50, or 55 (to account for out-of-the-money, at-the-money, and in-the-money); the asset price at time 0 is $S_0 = 50$; the risk-free rate $r = 0.05$; the volatility $\sigma = 0.3$, and the maturity $T = 1$ year.

We consider two types of dependence structures between the two lognormal assets in the basket. Each dependence structure is the copula implied by the uniform distribution on a triangle. We start with the following two triangles: $\triangle((0,0),(0,1)(1,0))$ and $\triangle((1,1),(0,1)(1,0))$. For each triangle, we generate point sets on that triangle and transform them to marginally standard uniform distributions with their marginal CDFs, $F(x_i) = 1-(1-x_i)^2$ for $i = 1,2$ for $\triangle((0,0),(0,1),(1,0))$ and $F(x_i) = 1-x_i^2$ for $i = 1,2$ for $\triangle((1,1),(0,1),(1,0))$ to obtain samples from the corresponding copula. Figure 8 shows such copula samples, where the underlying triangles were generated using the rSobol' + root method as detailed in the following section. The triangle-implied-copulas are used to model scenarios in which the asset prices are not simultaneously high (first triangle) or not simultaneously low (second triangle).

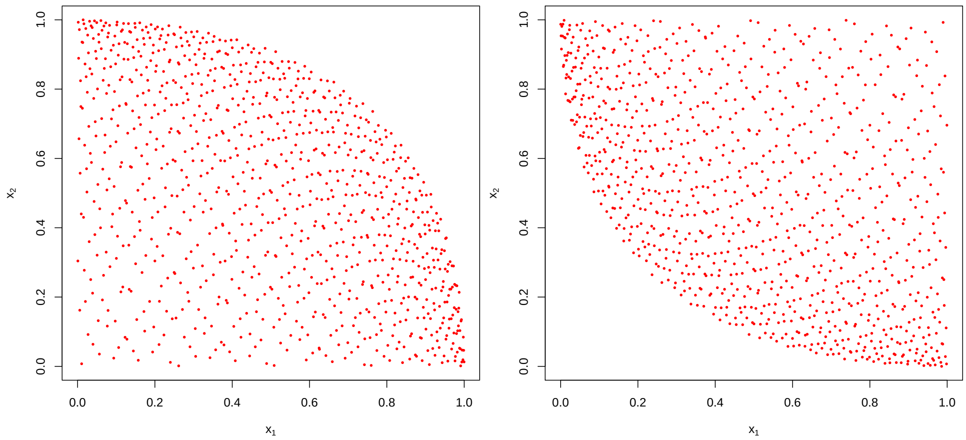

Fig. 8 Samples from the copulas based on $\triangle((0,0),(0,1)(1,0))$ (left) and $\triangle((1,1),(0,1)(1,0))$ (right) used in the basket option pricing example.

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 "root" method from Pillards and Cools (2005).
2. rSobol' + root: Here, the Sobol sequence randomized with a digital shift is generated, and then the "root" method from Pillards and Cools (2005) is applied.
3. rLattice1: This refers to using Algorithm 4 with the rank-1 lattice of Cools et al (2006) and $\alpha = 3\pi/8$, randomized with a shift.
4. rLattice2: This refers to the rank-2 lattice of Basu and Owen (2015), randomized with a shift. This method is not extensible, so a new point set must be generated every time n is changed.
5. rvdC: This refers to our randomized triangular vdC sequence based on stratified sampling.

If necessary, Algorithm 1 is then used to transform the points to the desired triangle. The first $n = 1000$ points from these constructions are shown in Figure 9.

For each method and each sample size $n \in 2^4, 2^5, \dots 2^{17}$, $v = 25$ 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 10 displays the results for the two-dimensional test functions. We also report the convergence speed (as measured by the regression coefficient α of $\log(\widehat{\text{Var}}) = \alpha\log(n) + c$ displayed in the legend). We can see from the results that the bivariate Sobol' sequence mapped to the triangle using the "root" method is typically the best performing method on these test functions. It does particularly well on function f_2, which can be explained by the parameters chosen for this function, in conjunction with how the root transformation is applied to map to the triangle $\triangle_R$. Indeed, since we use the mapping $(1-\sqrt{u_1}, \sqrt{u_1}u_2)$, it means the corresponding function on $[0,1)^2$ is smoother than f_2, i.e., it oscillates with a lower frequency. This is unlike what is happening for function f_3, where the methods sampling directly on the triangle deal with a sum

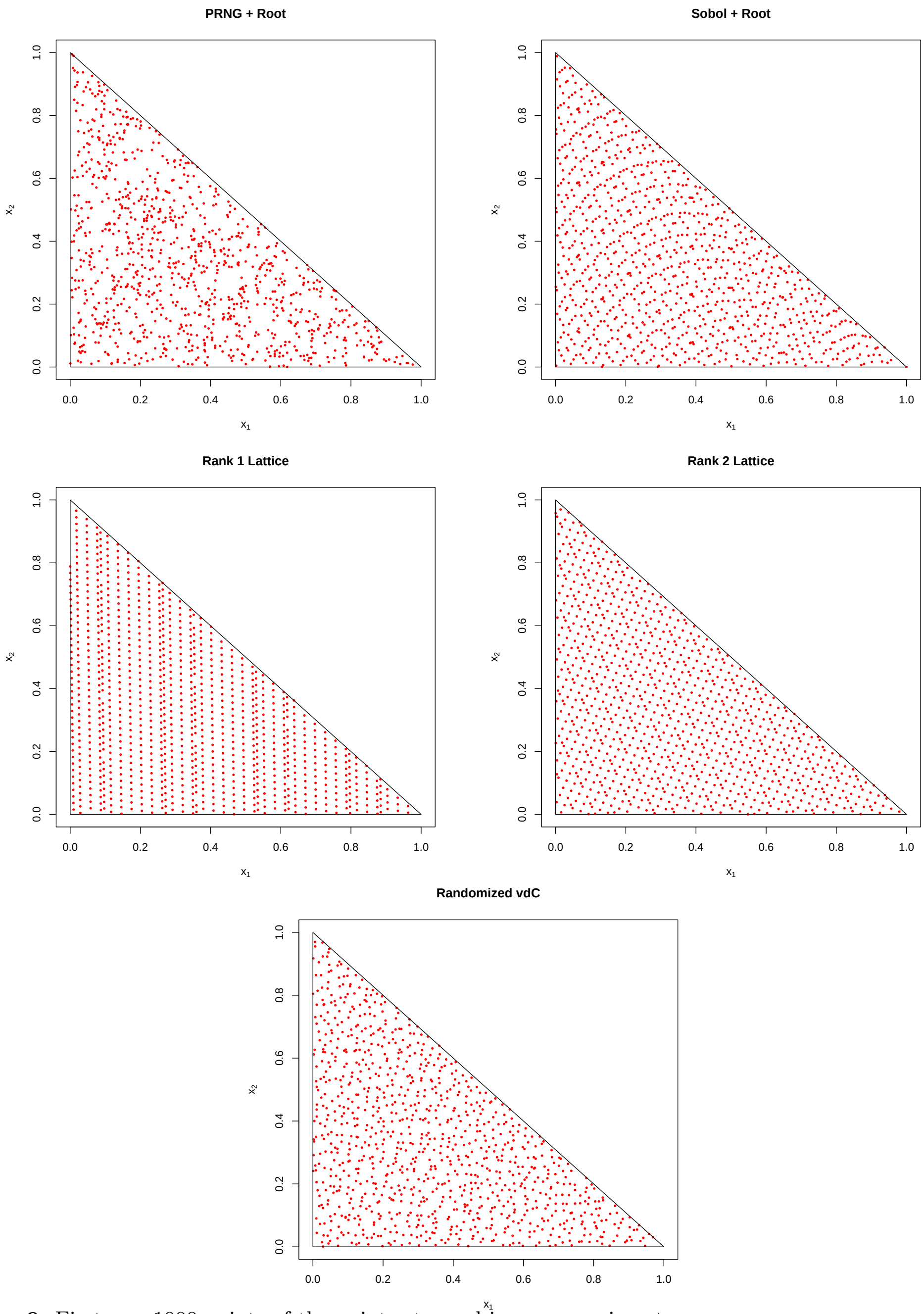

Fig. 9 First $n = 1000$ points of the point sets used in our experiments.

of univariate functions, while the corresponding function on $[0,1)^2$ based on the root transformation is truly two-dimensional.

Our stratified sampling method is approximately equal in performance to the lattice-based methods, except for f_3 where it does better and is in fact essentially as good as the Sobol' + root method. 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. Nested scrambling also 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 has the best performance out of the three test functions, as it is the smoothest function with the most between-strata variance.

Figures 11 and 12 display the results for the basket option pricing problem. Again, we also report the convergence speed in the legend. For all strike prices, the results are based on the same $v = 25$ randomized point sets. As was the case for the two-dimensional test functions, we can see from the results that the bivariate Sobol' sequence mapped to the triangle using the "root" method is typically the best performing method on these test functions. Our stratified sampling-based estimator tends to outperform the lattice-based methods. These results suggest that point sets directly constructed to have a low discrepancy in the integration space of interest may be outperformed by point sets obtained by applying a well-chosen transformation to a low-discrepancy point set constructed over the unit cube. Gaining a better understanding of when and why this happens is something that we plan to investigate in the near future.

6 Conclusion

In this paper, we provided an extensible rank-1 lattice construction for points in the triangle that can be randomized with a shift. We also examined the projection qualities of the triangular vdC sequence, and we improved upon the triangular vdC sequence

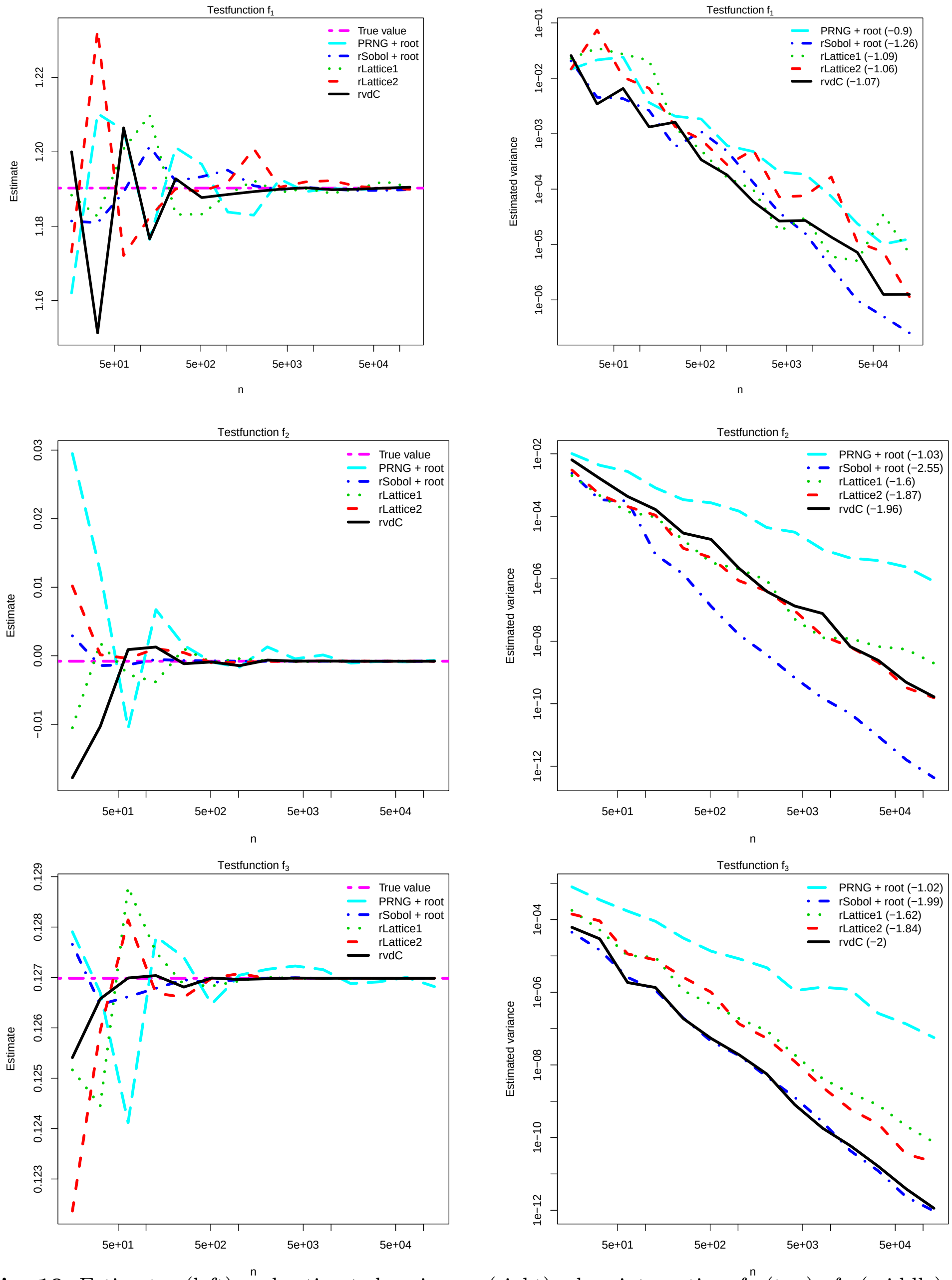

Fig. 10 Estimates (left) and estimated variances (right) when integrating f_1 (top), f_2 (middle) or f_3 (right). For each n, $v = 25$ randomizations were used. Regression coefficients for the estimated variance are in parentheses in the legends.

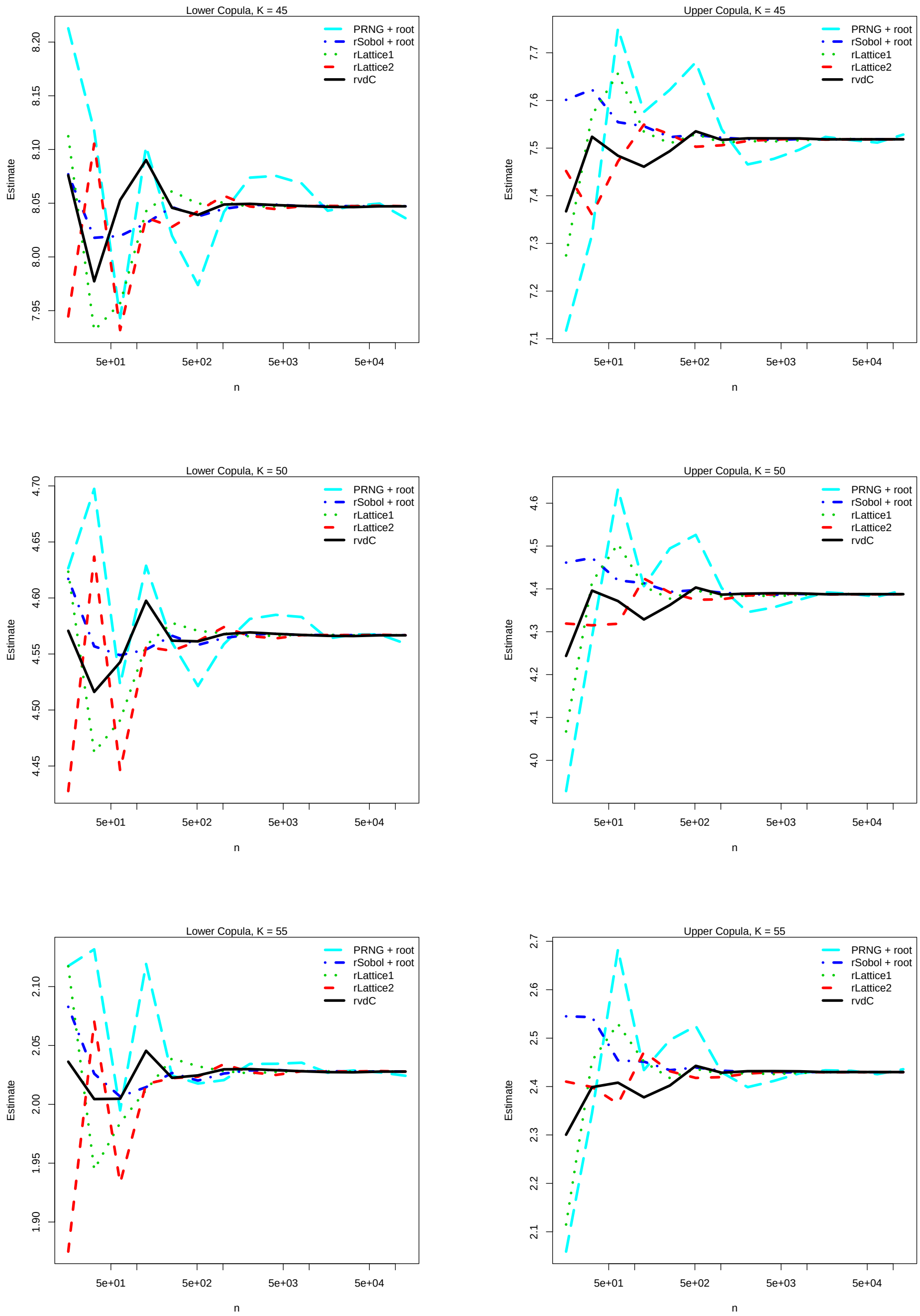

Fig. 11 Estimates when integrating the basket option pricing problem. For each n, $v = 25$ randomizations were used.

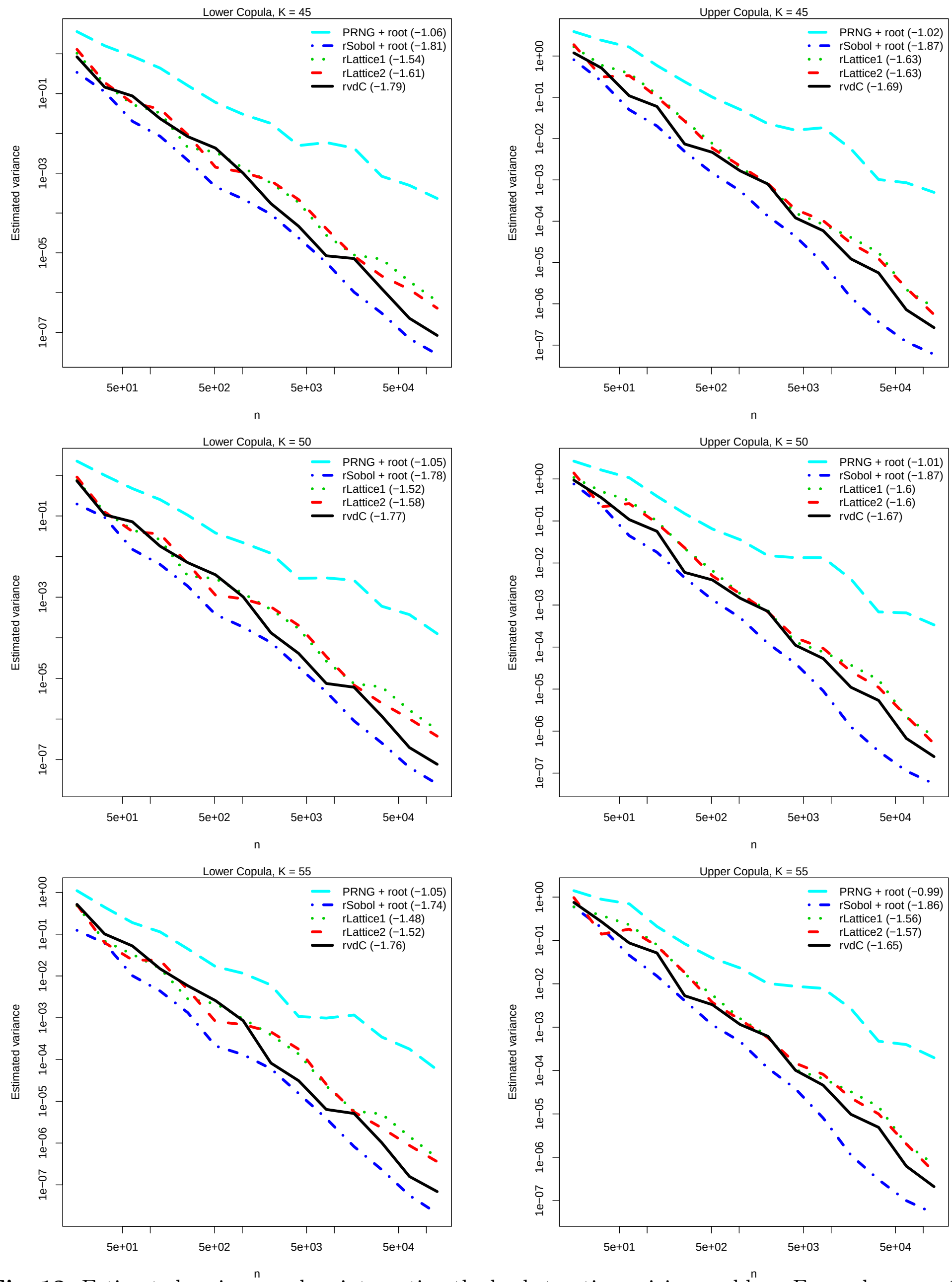

Fig. 12 Estimated variances when integrating the basket option pricing problem. For each n, $v = 25$ randomizations were used. Regression coefficients are in parentheses in the legends.

of Basu and Owen (2015) 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 sequence can be efficiently implemented using stratified sampling, giving the benefits of the reduced variance without the additional computational costs. This connection between stratified sampling and scrambling also gives an extensible stratified estimator. We give a test suite of functions and include a numerical study to compare the different sampling constructions over a triangular region.

Future work in this area includes extending similar stratified sampling schemes onto other surfaces, such as surfaces of spheres and simplexes. We would also like to explore constructions and applications that require sampling on multiple triangles, such as surfaces constructed with a mesh of triangles.

Declarations

Funding

This work was supported by the Natural Science and Engineering Research Council of Canada (NSERC) via grant 238959 to CL and the Canadian Statistical Sciences Institute (CANSSI) via the CANSSI Distinguished Postdoctoral Fellowship program to GD.

Conflicts of interest/Competing interests

The authors have no conflicts of interest to declare that are relevant to the content of this article.

Availability of data and material

No datasets were analysed during the current study. R implementation of the algorithms described within this article and R code for the numerical experiments are available upon request.

Authors' contributions

Gracia Dong and Erik Hintz wrote the main manuscript text and code required for the numerical experiments. Erik Hintz developed the methodology in Section 3. Gracia Dong developed the methodology in Section 4. Christiane Lemieux and Marius Hofert supervised the project. All authors reviewed the manuscript.

References

Arvo J (1995) Stratified sampling of spherical triangles. In: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pp 437–438, https://doi.org/10.1145/218380.218500

Basu K, Owen A (2015) Low discrepancy constructions in the triangle. SIAM Journal on Numerical Analysis 53(2):743–761. https://doi.org/10.1137/140960463

Basu K, Owen A (2016) Transformations and Hardy–Krause variation. SIAM Journal on Numerical Analysis 54(3):1946–1966. https://doi.org/10.1137/15M1052184

Brandolini L, Colzani L, Gigante G, et al (2013) A Koksma–Hlawka inequality for simplices. In: Trends in harmonic analysis. Springer, p 33–46, https://doi.org/10.1007/978-88-470-2853-1_3

Cools R, Kuo F, Nuyens D (2006) Constructing embedded lattice rules for multivariate integration. SIAM Journal on Scientific Computing 28(6):2162–2188. https://doi.org/10.1137/06065074X van der Corput J (1935) Verteilungsfunktionen i & ii. In: Nederl. Akad. Wetensch. Proc., pp 1058–1066

Cowper G (1973) Gaussian quadrature formulas for triangles. International Journal for Numerical Methods in Engineering 7(3):405–408. https://doi.org/10.1002/nme.1620070316

Cranley R, Patterson TNL (1976) Randomization of number theoretic methods for multiple integration. SIAM Journal on Numerical Analysis 13(6):904–914. https://doi.org/10.1137/0713071

Dick J, Pillichshammer F (2010) Digital nets and sequences: discrepancy theory and quasi–Monte Carlo integration. Cambridge University Press, https://doi.org/10.1017/CBO9780511761188

Dick J, Goda T, Suzuki K (2022) Component-by-component construction of randomized rank-1 lattice rules achieving almost the optimal randomized error rate. Mathematics of Computation 91(338):2771–2801. https://doi.org/10.1090/mcom/3769

Disney S, Sloan IH (1992) Lattice integration rules of maximal rank formed by copying rank 1 rules. SIAM journal on numerical analysis 29(2):566–577. https://doi.org/10.1137/0729036

Dong GY, Lemieux C (2022) Dependence properties of scrambled halton sequences. Mathematics and Computers in Simulation https://doi.org/10.1016/j.matcom.2022.04.016

Fang K, Wang Y (1993) Number-theoretic methods in statistics, vol 51. CRC Press

Gerber M (2015) On integration methods based on scrambled nets of arbitrary size. Journal of Complexity 31(6):798–816. https://doi.org/10.1016/j.jco.2015.06.001

Goda T, L'Ecuyer P (2022) Construction-free median quasi-monte carlo rules for function spaces with unspecified smoothness and general weights. SIAM Journal on Scientific Computing 44(4):A2765–A2788. https://doi.org/10.1137/22m1473625

Goda T, Suzuki K, Yoshiki T (2017) Quasi-Monte Carlo integration for twice differentiable functions over a triangle. Journal of Mathematical Analysis and Applications 454(1):361–384. https://doi.org/10.1016/j.jmaa.2017.04.051

Heitz E (2019) A low-distortion map between triangle and square. HAL preprint

Hickernell FJ (1996) The mean square discrepancy of randomized nets. ACM Transactions on Modeling and Computer Simulation (TOMACS) 6(4):274–296. https://doi.org/10.1145/240896.240909

Hickernell FJ, Hong HS (1997) Computing multivariate normal probabilities using rank-1 lattice sequences. In: Golub GH, Lui SH, Luk FT, et al (eds) Proceedings of the Workshop on Scientific Computing (Hong Kong). Springer-Verlag, Singapore, p 209–215

Hickernell FJ, Hong HS, L'Ecuyer P, et al (2001) Extensible lattice sequences for quasi-Monte Carlo quadrature. SIAM Journal on Scientific Computing 22:1117–1138. https://doi.org/10.1137/s1064827599356638

Hong HS, Hickernell FJ (2003) Algorithm 823: Implementing scrambled digital sequences. ACM Transactions on Mathematical Software (TOMS) 29(2):95–109. https://doi.org/10.1145/779359.779360

Korobov A (1959) The approximate computation of multiple integrals. In: Dokl. Akad. Nauk SSSR, pp 1207–1210

Lemieux C (2009) Monte Carlo and Quasi-Monte Carlo Sampling. Springer Series in Statistics, Springer New York, https://doi.org/10.1007/978-0-387-78165-5, URL https://books.google.ca/books?id=wj5OyydZ5bkC

Owen AB (1995) Randomly permuted (t, m, s)-nets and (t, s)-sequences. In: Monte Carlo and quasi-Monte Carlo methods in scientific computing. Springer, p 299–317, https://doi.org/10.1007/978-1-4612-2552-2_19

Owen AB (1997) Scrambled net variance for integrals of smooth functions. The Annals of Statistics 25(4):1541–1562. https://doi.org/10.1214/aos/1031594731

Owen AB (2003) Variance and discrepancy with alternative scramblings. ACM Transactions of Modeling and Computer Simulation 13(4). https://doi.org/10.1145/945511.945518

Pharr M (2019) Adventures in sampling points on triangles. URL https://pharr.org/matt/blog/2019/02/27/triangle-sampling-1, accessed: 2022-06-14

Pillards T, Cools R (2005) Transforming low-discrepancy sequences from a cube to a simplex. Journal of computational and applied mathematics 174(1):29–42. https://doi.org/10.1016/j.cam.2004.03.019

Rosenblatt M (1952) Remarks on a multivariate transformation. The Annals of Mathematical Statistics 23(3):470–472. https://doi.org/10.1214/aoms/1177729394

Shirley P, Chiu K (1997) A low distortion map between disk and square. Journal of graphics tools 2(3):45–52

Sloan IH, Joe S (1994) Lattice methods for multiple integration. Oxford University Press

Tymchyshyn V, Khlevniuk A (2019) Beginner's guide to mapping simplexes affinely. ResearchGate preprint https://doi.org/10.13140/RG.2.2.13787.41762

Wiart J, Lemieux C, Dong GY (2021) On the dependence structure and quality of scrambled (t, m, s)-nets. Monte Carlo Methods and Applications https://doi.org/10.1515/mcma-2020-2079

Appendix A Extending a stratified estimator

The approach described in Section 4, where we used the connection between stratified sampling and nested scrambling to give a faster implementation of nested scrambling, works well when n is fixed. Now, we use the connection between base b stratified sampling and nested scrambling to show how to extend a stratified estimator. Typically, stratified sampling is applied for fixed n and if a larger point set is needed, a new one is generated from scratch instead of only generating the additional points. Here, we argue that the 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 with a completely new stratified estimator. This works by recursively choosing a stratum at each level that has the least number of points, as explained in Algorithm 7. For simplicity, we deal directly with the stratum sample sizes $N_1, \ldots, N_M$ as generated by Algorithm 5 instead of the point set P_n, since after generating the strata sample sizes, 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 7 by changing Step 2a 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 3a, we must first populate $N_1, \ldots, N_M$ based on the existing point set.

Algorithm 7 (Extending a stratified estimator)**.** *Given $N_1, \ldots, N_M$ and a base b, we sample one additional point as follows.*

1. *Set $n = \sum_{j=1}^{M} N_j$.*
2. *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 contain two points. Subdivide this interval into b intervals, and randomly select two of these subintervals to have a point, i.e., $N_j = 1$.*
   (c) *Subdivide the other intervals of size b^{-q} into b intervals, and randomly select one of these intervals to have a point, i.e., $N_i = 1$.*
   (d) *Return $N_1, \ldots, N_{b^{q+1}}$.*
3. *If $n \neq M$, we do not need to increase the number of strata. We work with $N_1, .., N_M$.*
   (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 $L_j > 0$, repeat this algorithm from Step 2 on the j^{th} subinterval.*
   (f) *Return $N_1, \ldots, N_M$.*

We now illustrate Algorithm 7 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 Q_l the number of points in $[(l-1)/3, l/3)$ for $l = 1, 2, 3$ within P_7. That is, $Q_1 = N_1 + N_2 + N_3$, $Q_2 = N_4 + N_5 + N_6$, and $Q_3 = N_7 + N_8 + N_9$. That is, Q_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 Q_1, Q_2, Q_3 is equal to 3 and the other two are equal to 2. Say we have $Q_1 = Q_2 = 2, Q_3 = 3$. 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 Q_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., for which $Q_1, \ldots, Q_9$ will be equal to 2, as they are currently all equal to 1); determine in which of the intervals of size 1/27 the point lies 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} we 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.