Efficient calculation of the worst-case error and (fast) component-by-component construction of higher order polynomial lattice rules

Jan Baldeaux[*] Josef Dick[†] Gunther Leobacher[‡] Dirk Nuyens[§]
Friedrich Pillichshammer[¶]

July 29, 2011

Abstract

We show how to obtain a fast component-by-component construction algorithm for higher order polynomial lattice rules. Such rules are useful for multivariate quadrature of high-dimensional smooth functions over the unit cube as they achieve the near optimal order of convergence. The main problem addressed in this paper is to find an efficient way of computing the worst-case error. A general algorithm is presented and explicit expressions for base 2 are given. To obtain an efficient component-by-component construction algorithm we exploit the structure of the underlying cyclic group.

We compare our new higher order multivariate quadrature rules to existing quadrature rules based on higher order digital nets by computing their worst-case error. These numerical results show that the higher order polynomial lattice rules improve upon the known constructions of quasi-Monte Carlo rules based on higher order digital nets.

Keywords: Numerical integration, quasi-Monte Carlo, polynomial lattice rules, digital nets.

2010 Mathematics Subject Classification: 65D30, 65C05.

[*] Jan Baldeaux, School of Finance and Economics, The University of Technology, Sydney, NSW 2007, Australia. jan.baldeaux@uts.edu.au

[†] Josef Dick, School of Mathematics and Statistics, The University of New South Wales, Sydney, NSW 2052, Australia. josef.dick@unsw.edu.au J.D. is supported by a Queen Elizabeth 2 Fellowship from the Australian Research Council.

[‡] Gunther Leobacher, Institut für Finanzmathematik, Universität Linz, Altenbergerstraße 69, A-4040 Linz, Austria. gunther.leobacher@jku.at G.L. is partially supported by the Austrian Science Foundation (FWF), Project P21196.

[§] Dirk Nuyens, Department of Computer Science, K.U.Leuven, Celestijnenlaan 200A – bus 2402, 3001 Heverlee, Belgium. dirk.nuyens@cs.kuleuven.be D.N. is a postdoctoral fellow of the Research Foundation Flanders (FWO).

[¶] Friedrich Pillichshammer, Institut für Finanzmathematik, Universität Linz, Altenbergerstraße 69, A-4040 Linz, Austria. friedrich.pillichshammer@jku.at F.P. is partially supported by the Austrian Science Foundation (FWF), Project S9609, that is part of the Austrian National Research Network "Analytic Combinatorics and Probabilistic Number Theory".

1 Introduction

In this paper we are concerned with quasi-Monte Carlo rules, which are equal weight multivariate quadrature rules (or cubature rules)

$$Q(f) := \frac{1}{N}\sum_{h=0}^{N-1} f(\boldsymbol{x}_h), \tag{1}$$

used to approximate multivariate integrals over the s-dimensional unit cube

$$I(f) := \int_{[0,1]^s} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}. \tag{2}$$

In contrast to the Monte Carlo method, which samples the function f randomly in its domain, the integration nodes $\{\boldsymbol{x}_h\}_{h=0}^{N-1}$ used in the quasi-Monte Carlo rule Q are chosen deterministically. The convergence of the error of the Monte Carlo method is $O(N^{-1/2})$ (in distribution), while the worst-case error for quasi-Monte Carlo is $O(N^{-\alpha}(\log N)^{s\alpha})$ [4, 5, 11, 15] and the random-case error for randomized quasi-Monte Carlo is $O(N^{-\alpha-1/2}(\log N)^{s(\alpha+1)})$ [3, 22]. The latter two methods require that the integrand has smoothness $\alpha \geq 1$ (which means for instance that the integrand has square integrable partial mixed derivatives up to order α in each variable), whereas the Monte Carlo method requires only that the integrands have finite variance.

Different types of point sets for quasi-Monte Carlo rules exist. Those of interest here are *digital nets* [11, 15]. These can be divided into (*classical*) *digital nets* [15], which achieve a convergence rate of $O(N^{-1}(\log N)^s)$ [15] for integrands of bounded variation, and their extension called *higher order digital nets* [4, 5], which achieve a convergence rate of $O(N^{-\alpha}(\log N)^{\alpha s})$ for integrands which have square integrable partial mixed derivatives of order $\alpha > 1$. In [5, Section 4.4] an explicit method for constructing such higher order digital nets, based on a classical digital net, can be found. The method in the current paper gives an alternative construction for a higher order version of a specific type of digital net, namely *polynomial lattice point sets*, see [15, Section 4.4] or [11, Chapter 10]. Constructions of classical polynomial lattice point sets based on a worst-case error criterion have previously been studied in [8]. Note that we call a quasi-Monte Carlo rule whose underlying quadrature points are polynomial lattice points a polynomial lattice rule.

Polynomial lattice point sets were generalized in [10] to obtain *higher order polynomial lattice point sets*. In [7] existence results on higher order polynomial lattice point sets were compared to the explicit construction of higher order digital nets [5] in terms of their t-value (a certain quality measure). For some values of dimension s and/or smoothness parameter α the higher order polynomial lattice point sets have a better existence bound than the best results which can currently be obtained using the explicit construction of higher order digital nets from [6]. The same is also true for classical digital nets, see [14, 24]. These findings motivated the quest for an explicit construction of higher order polynomial lattice rules in [2]. The construction employed there is an algorithm originally proposed for the construction of (integer) lattice rules, namely the *component-by-component construction* algorithm, see, e.g., [12, 13, 25]. The higher order polynomial lattice rules so constructed achieve nearly optimal rates of convergence. For analogous results on polynomial lattice point sets see [8] (and also [9] for more background).

Straightforward implementation of the component-by-component (CBC) algorithm is however very costly with respect to computational time, hence methods for reducing the computational cost are needed. The fast component-by-component algorithm, introduced in [20], uses fast Fourier transforms (FFTs) to speed up the calculations. Some notes concerning the application of the fast algorithm to the construction of polynomial lattice rules were already made in [21], with a more detailed analysis in [19]; see also [11, Section 10.3]. In this paper we will adapt the fast algorithm for higher order polynomial lattice rules. To do so, we find a closed form for the worst-case error of our function space (where we consider the worst-case error as a function of the quadrature points). We show that our algorithm has a computational cost of $O(sN^\alpha \alpha \log N)$ operations using $O(N^\alpha)$ memory, compared to $O(s^2 N^{\alpha+1})$ operations for the straightforward implementation of the algorithm in [2], using the same amount of memory. This speedup makes it possible to obtain higher order polynomial lattice rules for moderate dimensions and numbers of points. In the section on numerical results we provide constructions of higher order polynomial lattice rules in base $b = 2$ up to dimension 10 and up to 4096 points. These numbers could be increased with more computational effort, but we have to remark that the search space grows exponentially with respect to the smoothness parameter α.

The efficient calculation of the worst-case error of our function space is an essential ingredient in such an algorithm. We show that the kernel function associated with the worst-case error can be evaluated at a point x in time $O(\alpha n)$, where α is the smoothness of the space and x is a rational number v/b^n, $0 \le v < b^n$. Moreover, in the case of the greatest practical importance, i.e., where the base equals 2, we show explicit expressions for smoothness 2 and 3 which are exact for any real $x \in [0, 1)$ (see Corollary 1).

We compare the performance of higher order polynomial lattice rules constructed using our fast component-by-component algorithm to the explicit construction as outlined in [5] and find that the new algorithm performs better in the cases considered. Finally, for the benefit of the reader, we present some limited tables of higher order polynomial lattice rules constructed using the fast component-by-component algorithm, allowing the reader to apply the rules presented in this paper to problems of interest and to verify implementations of the algorithm.

In the next section we provide the reader with some background, and notation, on Walsh spaces, digital nets, and the worst-case error. More detailed information can be found in [11] and [2], where also bounds on the worst-case error for higher order polynomial lattice rules were proven. In Section 3 we show how to efficiently calculate the worst-case error and how the construction of higher order polynomial lattice rules can be done using the fast component-by-component approach of [19, 20].

2 Background

We first introduce some notation. Let $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ denote the set of non-negative integers and $\mathbb{N} = \{1, 2, 3, \ldots\}$ the set of positive integers. Further we need to be able to consider a non-negative integer $k \in \mathbb{N}_0$ in its unique base b representation:

$$k = (\kappa_a \ldots \kappa_0)_b = \sum_{i=0}^{a} \kappa_i \, b^i, \tag{3}$$

where $\kappa_i \in \{0, \ldots, b-1\}$ are the base b digits of k and $\kappa_a \neq 0$; $a = 0$ for $k = 0$. Note that the base, b, is considered a fixed integer throughout. Moreover, in the further development in this paper, b will be prime. We will be specifically interested in the non-zero base b digits of k. The number of non-zero base b digits of an integer k will be denoted by $\#k$; where $\#0 = 0$. We can then represent $k \in \mathbb{N}_0$ uniquely as

$$k = \sum_{i=1}^{\#k} \kappa_{a_i} \, b^{a_i}, \tag{4}$$

where now $\kappa_{a_i} \in \{1, \ldots, b-1\}$ and we demand $a_1 > \cdots > a_{\#k} \geq 0$. Thus κ_{a_1} is the most significant base b digit of k. For real $x \in [0,1)$ we write its base b representation

$$x = (0.\xi_1 \xi_2 \ldots)_b = \sum_{i=1}^{\infty} \xi_i \, b^{-i}, \tag{5}$$

where $\xi_i \in \{0, \ldots, b-1\}$. This representation is unique in the sense that we do not allow an infinite repetition of the digit $b-1$ to the right.

2.1 A function space based on Walsh series

For $k \in \mathbb{N}_0$ the one-dimensional kth Walsh function in base b, $\mathrm{wal}_k : [0,1) \to \mathbb{C}$, is defined by

$$\mathrm{wal}_k(x) := \exp(2\pi\mathrm{i}\; (\xi_1 \kappa_0 + \cdots + \xi_{a+1} \kappa_a)/b), \tag{6}$$

where we have used the base b digits of k and x as given in (3) and (5). Note that Walsh functions (in base b) are piecewise constant functions. For dimensions $s \geq 2$ and vectors $\boldsymbol{k} = (k_1, \ldots, k_s) \in \mathbb{N}_0^s$ and $\boldsymbol{x} = (x_1, \ldots, x_s) \in [0,1)^s$ we define $\mathrm{wal}_{\boldsymbol{k}} : [0,1)^s \to \mathbb{C}$ as

$$\mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x}) := \prod_{j=1}^{s} \mathrm{wal}_{k_j}(x_j).$$

The integrand functions in this paper are assumed to have an absolutely convergent Walsh series representation

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{N}_0^s} \widehat{f}_{\boldsymbol{k}} \; \mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x}),$$

where the Walsh coefficients $\widehat{f}_{\boldsymbol{k}}$ are given by

$$\widehat{f}_{\boldsymbol{k}} = \int_{[0,1]^s} f(\boldsymbol{x}) \, \overline{\mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x}.$$

Note that the Walsh functions form a complete orthonormal system of $L_2([0,1]^s)$. For more information on Walsh functions and their properties we refer to [11, Chapter 14 and Appendix A].

In the following we define a function space by demanding a certain decay rate of the Walsh coefficients. To do so, we introduce some further notation. We define, for a fixed integer $\alpha > 1$ and a fixed sequence of positive weights $\boldsymbol{\gamma} = \{\gamma_1, \gamma_2, \ldots\}$ (in the sense of [26]),

$$r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k}) := \prod_{j \in u} \gamma_j \, r_\alpha(k_j), \qquad\qquad r_\alpha(k) := b^{-\sum_{i=1}^{\min(\#k,\alpha)} (a_i+1)}, \tag{7}$$

where for $\boldsymbol{k} = (k_1, \ldots, k_s)$ we set $u = \{1 \leq j \leq s : k_j \neq 0\}$ and where we used the first α positions $a_1 + 1, \ldots, a_{\#k} + 1$ of the non-zero base b digits of k with the notation defined in (4) in the one-dimensional definition of r_α. For $\boldsymbol{k} = \boldsymbol{0} = (0, \ldots, 0)$ we set

$$r_\alpha(\boldsymbol{0}) = 1.$$

We are now ready to specify which functions are in our function space $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$, which was also used in [2, 5]. For functions $f \in \mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ we define the norm

$$\|f\|_{\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}} := \sup_{\boldsymbol{k} \in \mathbb{N}_0^s} \frac{|\widehat{f}_{\boldsymbol{k}}|}{r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k})}. \tag{8}$$

Then $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ consists of all functions $f \in L_2([0,1]^s)$ for which $\|f\|_{\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}} < \infty$. The Walsh coefficients of $f \in \mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ therefore satisfy a certain decay criterion, namely

$$|\widehat{f}_{\boldsymbol{k}}| \le \|f\|_{\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}}\, r_\alpha(\boldsymbol{\gamma},\boldsymbol{k}) \quad \forall \boldsymbol{k} \in \mathbb{N}_0^s. \tag{9}$$

It is clear that larger values of α might increase the norm of a function f, i.e., $\|f\|_{\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}} \le \|f\|_{\mathcal{W}_{\alpha',s,\boldsymbol{\gamma}}}$ for $\alpha \le \alpha'$. The weights $\gamma_1, \gamma_2, \ldots$ are used to describe how anisotropic the space is. Usually it is assumed that $\gamma_1 \ge \gamma_2 \ge \cdots \ge 0$, meaning that the first dimension is more important than the second one and so on. Under certain conditions on these weights, it can be shown that numerical integration is tractable in the number of dimensions, see, e.g., [11, 16, 17].

It is of course important to have an understanding of which functions exactly are in such a Walsh space $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ with smoothness parameter α. This analysis has been done in [4, 5, 6]. Classically, one is interested in (smooth) functions $f : [0,1]^s \to \mathbb{R}$ for which all mixed partial derivatives up to order α in each variable are square integrable. This is a Sobolev space of smoothness α which is often considered for this type of problems. In [5, 6] a continuous embedding of certain Sobolev spaces into $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ was shown. Consequently, the results we are going to establish in the following for functions in $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ also apply automatically to what we normally consider as "smooth" functions, for instance, functions which have square integrable partial mixed derivatives up to order α in each variable. One of the simplest type of functions in this space are multivariate polynomials which make up nice testing examples for computer implementations.

2.2 Higher order digital nets

Higher order digital nets were introduced in [5]. Higher order polynomial lattice point sets, which are the focal point of this paper, are a special class of higher order digital nets. For that reason and since we will compare the explicit construction for higher order digital nets from [5] with the construction given in this paper, we will review the necessary details here. For more information we refer to [11, Chapter 15].

For a prime number b we always identify $\mathbb{F}_b$, the finite field with b elements, with $\mathbb{Z}_b = \{0, \ldots, b-1\}$ endowed with the usual arithmetic operations modulo b.

First we define higher order digital nets using the digital construction scheme. As we will need to be able to identify integers with vectors over a finite field by using its base b representation, and then later have to be able to consider vectors of integers as well, we will denote a vector over a finite field $\mathbb{F}_b$ by $\vec{h}$, in contrast to vectors over $\mathbb{Z}$ or $\mathbb{R}$, which will be denoted by $\boldsymbol{h}$.

Definition 1 (Digital construction scheme of a digital net over $\mathbb{F}_b$)**.** Let b be a prime and let $n, m, s \ge 1$ be integers, where $n \ge m$. Let $C_1, \ldots, C_s$ be $n \times m$ matrices over the finite field $\mathbb{F}_b$ of order b. Now we construct b^m points in $[0,1)^s$: for $0 \le h < b^m$, identify each $h = \sum_{i=0}^{m-1} h_i\, b^i$ with a vector over the finite field

$$\vec{h} := (h_0, \ldots, h_{m-1})^\top \in \mathbb{F}_b^m.$$

For $1 \le j \le s$ multiply the matrix C_j by $\vec{h}$ using arithmetic over $\mathbb{F}_b$ to obtain a vector $\vec{y}_{h,j} \in \mathbb{F}_b^n$:

$$C_j\, \vec{h} =: \vec{y}_{h,j} = (y_{h,j,1}, \ldots, y_{h,j,n})^\top \in \mathbb{F}_b^n, \tag{10}$$

from which the hth point $\boldsymbol{x}_h$ of the digital net is found by interpreting the coordinates of $\vec{y}_{h,j}$ as the base b digits of $x_{h,j}$:

$$x_{h,j} := \sum_{i=1}^{n} y_{h,j,i}\, b^{-i} \in [0,1).$$

Now set $\boldsymbol{x}_h = (x_{h,1}, \ldots, x_{h,s})^\top \in [0,1)^s$ to be the hth point. The set $\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{b^m-1}\}$ is called a *digital net* over $\mathbb{F}_b$ with generating matrices $C_1, \ldots, C_s$.

This definition of a digital net generalizes the classical construction scheme, e.g., [15], on which classical digital (t,m,s)-nets are based upon, by allowing for generating matrices which are not necessarily square. The generating matrices C_j are of size $n \times m$, and so, the number of rows n determines the resolution at which the points of the net are placed in the unit cube, i.e., all base b digits after position n are zero. The integration error then behaves like $O(b^{-k}k^{\alpha s})$, see [5], where α is the smoothness of the integrand and k is the *strength* of the net (in accordance with the respective property of classical nets). The strength of the net is defined via linear independence properties of the rows of the generating matrices, see [4, 5]. For higher order nets one can achieve $k \approx \min(\alpha m, n)$ and hence, provided that $n \geq \alpha m$, one obtains a convergence order of $b^{-\alpha m}(\alpha m)^{\alpha s} \asymp N^{-\alpha}(\log N)^{\alpha s}$, where $N = b^m$ is the number of quadrature points.

We now explain the explicit construction of a higher order digital net in s dimensions for a maximum smoothness d as described in [5]. The explicit construction starts from a given (t', m, sd)-net in base b, that is, a classical digital net in sd dimensions for which the generating matrices $C_1, \ldots, C_{sd} \in \mathbb{F}_b^{m \times m}$ are known.

From these sd given matrices, s new generating matrices $C_j^{(d)}$ are constructed of size $dm \times m$ by vertically stacking the first rows from the group of d consecutive matrices $C_{(j-1)d+1}, \ldots, C_{jd}$, then the second rows of the same d matrices and so on, until all dm rows have been stacked. More precisely, let $C_j = (\boldsymbol{c}_{j,1}^\top \ldots \boldsymbol{c}_{j,m}^\top)^\top$, where $\boldsymbol{c}_{j,k}^\top$ denotes the kth row of the matrix C_j. Then $C_j^{(d)} = (\boldsymbol{c}_{(j-1)d+1,1}^\top \cdots \boldsymbol{c}_{jd,1}^\top \cdots \boldsymbol{c}_{(j-1)d+1,m}^\top \cdots \boldsymbol{c}_{jd,m}^\top)^\top$. For more information on these higher order digital nets we refer the reader to [5] and [11, Chapter 15].

An important concept for the error analysis in the next section is the *dual net*. It defines the set of Walsh coefficients which are not integrated exactly by the digital net and can therefore be used to write down the integration error.

Definition 2 (Dual net)**.** For a digital net over $\mathbb{F}_b$ with generating matrices $C_1, \ldots, C_s \in \mathbb{F}_b^{n \times m}$ we define its *dual net* by

$$\mathcal{D}(C_1, \ldots, C_s) := \left\{ \boldsymbol{k} \in \mathbb{N}_0^s : C_1^\top \vec{k}_1 + \cdots + C_s^\top \vec{k}_s = \vec{0} \right\},$$

where for a scalar component $k = \sum_{i=0}^{\infty} \kappa_i \, b^i$ in $\boldsymbol{k}$ we define an associated vector over the finite field $\vec{k} = (\kappa_0, \ldots, \kappa_{n-1})^\top \in \mathbb{F}_b^n$.

2.3 The worst-case error

We define the worst-case error of numerical integration using a cubature rule Q for functions in a Banach space $\mathcal{F}$ by

$$e(Q, \mathcal{F}) := \sup_{\substack{f \in \mathcal{F} \\ \|f\|_{\mathcal{F}} \leq 1}} |I(f) - Q(f)|.$$

We now assume the cubature rule Q to be a quasi-Monte Carlo rule (1) using a (higher order) digital net as its node set and denote it by Q^{net}. For any f having an absolutely convergent Walsh series representation we can write the integration error for Q^{net} as a sum over the dual net to obtain

$$|I(f) - Q^{\text{net}}(f)| = \left| \sum_{\boldsymbol{0} \neq \boldsymbol{k} \in \mathcal{D}} \widehat{f}_{\boldsymbol{k}} \right| \leq \sum_{\boldsymbol{0} \neq \boldsymbol{k} \in \mathcal{D}} |\widehat{f}_{\boldsymbol{k}}|. \tag{11}$$

For $f \in \mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ we can now use (9) to obtain

$$|I(f) - Q^{\mathrm{net}}(f)| \le \|f\|_{\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}} \sum_{\boldsymbol{0} \ne \boldsymbol{k} \in \mathcal{D}} r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k}). \tag{12}$$

Since we can obtain equality for a worst-case function $\zeta \in \mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ having Walsh series representation

$$\zeta(\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{N}_0^s} r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k}) \operatorname{wal}_{\boldsymbol{k}}(\boldsymbol{x}),$$

we find the following expression for the worst-case error in $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ for a quasi-Monte Carlo rule Q^{net} based on a higher order digital net in base b:

$$e(Q^{\mathrm{net}}, \mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}) = \sum_{\boldsymbol{0} \ne \boldsymbol{k} \in \mathcal{D}} r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k}). \tag{13}$$

The cubature rules in this paper will be constructed in such a way that they have a worst-case error which is near optimal for the given function space $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$. For a given value of $\alpha > 1$ the worst-case error behaves like $O(N^{-\alpha}(\log N)^{\alpha s})$ for N integration nodes (see [5]) which is essentially best possible according to a lower bound from Šarygin [29].

2.4 Higher order polynomial lattice rules

In [10] the classical polynomial lattice rules [15] were generalized to form higher order polynomial lattice rules. Just like classical polynomial lattice point sets are a special class of digital nets, higher order polynomial lattice point sets are a special class of higher order digital nets. For simplicity, we define the (higher order) polynomial lattice rules over a finite field $\mathbb{F}_b$ of prime order b only. The main object for the construction of polynomial lattice rules are formal Laurent series, i.e., expressions of the form $\sum_{i=\ell}^{\infty} w_i \, X^{-i}$, where $\ell \in \mathbb{Z}$ and $w_i \in \mathbb{F}_b$. We denote the set of formal Laurent series by $\mathbb{F}_b((X^{-1}))$. These Laurent series then need to be mapped to integration nodes over the unit interval $[0,1)$. Define the map $v_n : \mathbb{F}_b((X^{-1})) \to [0,1)$ by

$$v_n\left(\sum_{i=\ell}^{\infty} w_i \, X^{-i}\right) := \sum_{i=\max(\ell,1)}^{n} w_i \, b^{-i}. \tag{14}$$

Similar to the case for digital nets, we now need to be able to identify an integer h with a polynomial in $\mathbb{F}_b[X]$ by considering h in its base b representation, the associated polynomial will be denoted by $h(X)$. The details are given in the following definition.

Definition 3 (Polynomial lattice rule)**.** Let b be prime and $1 \le m \le n$. For a given dimension $s \ge 1$, choose $p(X) \in \mathbb{F}_b[X]$ with $\deg(p) = n \ge 1$ and let $q_1(X), \ldots, q_s(X) \in \mathbb{F}_b[X]$. Now we construct b^m points in $[0,1)^s$: for $0 \le h < b^m$, identify each $h = \sum_{i=0}^{m-1} h_i \, b^i$ with a polynomial over $\mathbb{F}_b$

$$h(X) := \sum_{i=0}^{m-1} h_i \, X^i \in \mathbb{F}_b[X].$$

Then the hth point is obtained by setting

$$\boldsymbol{x}_h := \left(v_n\left(\frac{h(X)\, q_1(X)}{p(X)}\right), \ldots, v_n\left(\frac{h(X)\, q_s(X)}{p(X)}\right) \right) \in [0,1)^s.$$

A quasi-Monte Carlo rule using this point set is called a *polynomial lattice rule.*

One obtains classical polynomial lattice rules from Definition 3 by taking $n = m$. For simplicity we will assume that $p(X)$ is irreducible over $\mathbb{F}_b$, though this assumption could be removed by a more intricate analysis. We define

$$G_{b,n} = \{v(X) \in \mathbb{F}_b[X] \setminus \{0\} : \deg(v) < n\},$$

which will be the set from which we will select the generating polynomials $q_j(X)$. Clearly, as $p(X)$ is irreducible and $\deg(p) = n$, this equals the multiplicative group

$$G_{b,n} = (\mathbb{F}_b[X]/p(X))^\times = \{g(X)^\beta : 0 \leq \beta < b^n - 1\},$$

where $g(X)$ is a generator for the multiplicative group $(\mathbb{F}_b[X]/p(X))^\times$, e.g., we can take $g(X) = X$ when $p(X)$ is primitive.

Since a polynomial lattice point set is a special case of a digital net, we can find the generating matrices $C_1, \ldots, C_s \in \mathbb{F}_b^{n\times m}$ from the generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X))$. For $1 \leq j \leq s$ consider the Laurent expansions

$$\frac{q_j(X)}{p(X)} = \sum_{i=\ell_j}^{\infty} u_i^{(j)} X^{-i} \in \mathbb{F}_b((X^{-1})).$$

Then the elements $c_{k,\ell}^{(j)}$ of the $n \times m$ generating matrix C_j over $\mathbb{F}_b$ are given by

$$c_{k,\ell}^{(j)} = u_{k+\ell}^{(j)}, \tag{15}$$

for $1 \leq k \leq n$, $0 \leq \ell \leq m-1$; see, e.g., [11, Section 10.1].

In (13) we used the dual net to obtain the worst-case error. In the case of a polynomial lattice rule the dual is given in the next definition. (We use the convention $\deg(0) = -\infty$.)

Definition 4 (Dual polynomial lattice)**.** A polynomial lattice with generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X)) \in (\mathbb{F}_b[X])^s$ modulo $p(X) \in \mathbb{F}_b[X]$ has a *dual polynomial lattice*

$$\mathcal{D}(\boldsymbol{q}(X), p(X)) := \left\{ \boldsymbol{k} \in \mathbb{N}_0^s : \sum_{j=1}^{s} k_j(X)\, q_j(X) \equiv a(X) \pmod{p(X)} \quad \text{with } \deg(a) < n - m \right\}.$$

A proof for the equivalence of Definition 2 and Definition 4 for polynomial lattices follows from [11, Lemma 15.25].

Specifically for a polynomial lattice rule with generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X))$ modulo $p(X)$ having b^m points, it follows from (13) that its worst-case error in $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$ satisfies

$$e_{b^m,\alpha}(\boldsymbol{q}(X), p(X)) = \sum_{\boldsymbol{0}\neq\boldsymbol{k}\in\mathcal{D}} r_\alpha(\boldsymbol{\gamma}, \boldsymbol{k}), \tag{16}$$

with $\mathcal{D}$ the dual polynomial lattice.

2.5 The component-by-component construction of higher order polynomial lattice rules

The component-by-component construction algorithm was introduced by Korobov [12], see also [13, Theorem 18, p. 120], and later re-invented in [25] to construct the generating vector of an integer lattice rule. This algorithm first finds the optimal one-dimensional generating vector, which is subsequently extended in an optimal way to a two-dimensional generating vector and so on. Algorithm 1 spells out the details for the construction in the case of higher order polynomial lattice rules.

Algorithm 1 General form of CBC construction of higher order polynomial lattice rules

```
Input: base $b$ a prime, number of dimensions $s$, number of points $b^m$, smoothness $\alpha > 1$, and weights $\boldsymbol{\gamma} = (\gamma_j)_{j \geq 1}$
Output: Generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X)) \in G_{b,n}^s$

Choose an irreducible polynomial $p(X) \in \mathbb{F}_b[X]$, with $\deg(p) = n$ and $n = \alpha m$
for $d = 1$to $s$ do
  Set $q_d(X) \in G_{b,n}$ by minimizing $e_{b^m,\alpha}((q_1(X), \ldots, q_d(X)), p(X))$ as a function of $q_d(X)$
end for
return $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X))$
```

The analysis of the component-by-component algorithm adjusted to the case of higher order polynomial lattice rule was done in [2]. The following theorem shows that Algorithm 1 achieves almost optimal rates of convergence. For a proof we refer to [2].

Theorem 1. *Let b be prime, $s, n \in \mathbb{N}$ and $p(X) \in \mathbb{F}_b[X]$ be irreducible with $\deg(p) = n = \alpha m$, $\alpha > 1$. Suppose $(q_1(X), \ldots, q_s(X)) \in G_{b,n}^s$ is constructed using Algorithm 1. Then for all $d = 1, \ldots, s$ we have a bound on the worst-case error as follows:*

$$e_{b^m,\alpha}((q_1(X), \ldots, q_d(X)), p(X)) \leq \frac{1}{b^{\tau m}} \prod_{j=1}^{d} \left(1 + 3\gamma_j^{1/\tau} C_{b,\alpha,\tau}\right)^{\tau}, \qquad \forall\, 1 \leq \tau < \alpha,$$

where

$$C_{b,\alpha,\tau} := \frac{(b-1)^{\alpha}}{b^{\alpha/\tau} - b} \prod_{i=1}^{\alpha-1} \frac{1}{b^{i/\tau} - 1} + \begin{cases} \alpha - 1 & \text{if } \tau = 1, \\ \dfrac{(b-1)((b-1)^{\alpha-1} - (b^{1/\tau} - 1)^{\alpha-1})}{(b - b^{1/\tau})(b^{1/\tau} - 1)^{\alpha-1}} & \text{if } \tau > 1. \end{cases}$$

Formula (16) for the worst-case error is not in a usable form for computation due to the infinite sum. The next lemma shows how to obtain a closed-form expression which resembles the formula for the worst-case error as it appears when the space of integrands is a reproducing kernel Hilbert space, see [1].

Lemma 1. *The worst-case integration error in $\mathcal{W}_{\alpha,s,\boldsymbol{\gamma}}$, $\alpha > 1$, associated with a polynomial lattice rule with generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X))$ modulo $p(X)$ having b^m points satisfies*

$$e_{b^m,\alpha}((q_1(X), \ldots, q_s(X)), p(X)) = -1 + \frac{1}{b^m} \sum_{h=0}^{b^m-1} \prod_{j=1}^{s} (1 + \gamma_j\, \omega_\alpha(x_{h,j})), \tag{17}$$

where, using (7),

$$\omega_\alpha(x) := \sum_{k=1}^{\infty} r_\alpha(k)\ \mathrm{wal}_k(x). \tag{18}$$

Proof. We make use of the character property of digital nets (see [11, Lemma 4.75]). When $\{\boldsymbol{x}_h\}_{h=0}^{b^m-1}$ are the points of a digital net (or a polynomial lattice rule) and $\mathcal{D}$ is its dual net, then

$$\frac{1}{b^m}\sum_{h=0}^{b^m-1} \mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x}_h) = \begin{cases} 1 & \text{if } \boldsymbol{k} \in \mathcal{D}, \\ 0 & \text{otherwise.} \end{cases}$$

Thus, starting from (16), we obtain

$$\begin{aligned} e_{b^m,\alpha}((q_1(X),\dots,q_s(X)),p(X)) &= \sum_{\boldsymbol{0}\neq\boldsymbol{k}\in\mathbb{N}_0^s} r_\alpha(\boldsymbol{\gamma},\boldsymbol{k}) \frac{1}{b^m}\sum_{h=0}^{b^m-1} \mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x}_h) \\ &= -1 + \frac{1}{b^m}\sum_{h=0}^{b^m-1}\sum_{\boldsymbol{k}\in\mathbb{N}_0^s} r_\alpha(\boldsymbol{\gamma},\boldsymbol{k})\ \mathrm{wal}_{\boldsymbol{k}}(\boldsymbol{x}_h) \\ &= -1 + \frac{1}{b^m}\sum_{h=0}^{b^m-1}\prod_{j=1}^{s}(1+\gamma_j\,\omega_\alpha(x_{h,j})). \end{aligned}$$

□

As in [20] we can now do a basic operation count for the computational cost of Algorithm 1. In comparison to [20], the analysis is a little bit more involved here as the evaluation of (17) involves calculating

$$\omega_\alpha\left(v_n\left(\frac{h(X)\,q(X)}{p(X)}\right)\right) \qquad \text{for } h=0,\dots,b^m-1 \text{ and all } q(X)\in G_{b,n}, \tag{19}$$

in each iteration. Assuming $c_\omega = c_\omega(\alpha,n)$ to be the cost of evaluating $\omega_\alpha(x)$ and $c_v = c_v(n)$ the cost of mapping and calculating the Laurent expansion as well as calculation the polynomial product, the cost of a straightforward implementation of Algorithm 1 is $O(s^2 b^n b^m (c_\omega + c_v))$ where $n=\alpha m$. However, calculating $\omega_\alpha(v_n(h(X)\,q(X)/p(X)))$ efficiently for given $h(X)$ and $q(X)$ is an important issue which is solved in the next section and in practice it would be inefficient to calculate these values on the fly whenever needed. To that end we model the algorithm into a more tangible form using the techniques from [20, 21] to obtain a fast component-by-component algorithm that makes use of a circular convolution which can be calculated by means of fast Fourier transforms (FFTs).

3 Fast construction of higher order polynomial lattice rules

The exposition here mainly follows the techniques from [20, 21], but, as mentioned in the previous section, the analysis is more complicated due to the need to calculate (19). The derivation of the fast algorithm is kept concise by relying as quickly as possible on the structure of the underlying multiplicative group, but we need to take into consideration the cost c_v of working with polynomials over finite fields. In Section 4 we will give efficient methods to calculate ω_α.

The product over j in Lemma 1 can be reused and extended from the previous iteration. We store this product in a vector $\boldsymbol{P}_d = (P_d(0),\dots,P_d(b^m-1))$ of length b^m, where

$$P_d(h) := \prod_{j=1}^{d}(1+\gamma_j\,\omega_\alpha(x_{h,j})) = P_{d-1}(h)\left(1+\gamma_j\,\omega_\alpha\left(v_n\left(\frac{h(X)\,q_d(X)}{p(X)}\right)\right)\right),$$

for all $0 \leq h < b^m$ and $P_0(h) = 1$. Thus $\boldsymbol{P}_d$ can be calculated using the stored value for $\boldsymbol{P}_{d-1}$. Hereby we reduce the construction cost by a factor of s at the cost of $O(b^m)$ memory.

The computations of $\omega_\alpha(v_n(h(X)\,q(X)/p(X)))$ could be done in the initialization of the algorithm. Since v_n only depends on the negative powers of X we in fact have

$$v_n\left(\frac{h(X)\,q(X)}{p(X)}\right) = v_n\left(\frac{h(X)\,q(X)}{p(X)} \bmod 1(X)\right) = v_n\left(\frac{h(X)\,q(X) \bmod p(X)}{p(X)}\right).$$

So, for fixed $p(X)$, we can think of v_n as being a function from $\mathbb{F}_b[X]/p(X) = \{w(X) \in \mathbb{F}_b[X] : \deg(w) < n\}$ to $[0,1)$. We can precompute these b^n values giving a construction cost of $O(sb^n b^m c_v + b^n(c_\omega + c_v))$ at a cost of $O(b^m + b^n)$ memory. However, the cost c_v is presumably dominating c_ω (most certainly so for the ω_α expressions we will derive in Corollary 1). It is standard practice to use a lookup table based on a generator when doing multiplications over a finite field. Making this change the construction cost becomes $O(sb^n b^m + b^n(c_\omega + \tilde{c}_v))$ at a cost of $O(b^m + 2b^n)$ memory (we have explicitly written the constant for clarity: there is a $O(b^n)$ cost for the values of ω_α and a $O(b^n)$ cost for the lookup table). Here, $\tilde{c}_v$ is a lot cheaper than c_v as one has to multiply only by the same generator to construct the table. We do note however that the $O(b^n)$ memory cost grows exponentially with α as $n = \alpha m$.

For the lookup table we made use of the fact that there exists a generator $g(X)$ for the multiplicative group for which

$$(\mathbb{F}_b[X]/p(X))^\times := \{g(X)^\beta \bmod p(X) : 0 \leq \beta < b^n - 1\} = \mathbb{F}_b[X]/(p(X)) \setminus \{0\},$$

since we assumed $p(X)$ to be irreducible over $\mathbb{F}_b[X]$. For brevity we define the auxiliary function ω to make use of the indices w.r.t. the generator $g(X)$:

$$\begin{aligned}
\omega : \mathbb{Z}_{b^n-1} \to [0,1) : \omega(\beta - \delta) &= \omega(\beta - \delta \bmod (b^n - 1)) \\
&:= \omega_\alpha\left(v_n\left(\frac{h(X)\,q(X) \bmod p(X)}{p(X)}\right)\right) \\
&= \omega_\alpha\left(v_n\left(\frac{g(X)^\beta\, g(X)^{-\delta} \bmod p(X)}{p(X)}\right)\right) \\
&= \omega_\alpha\left(v_n\left(\frac{g(X)^{\beta-\delta} \bmod p(X)}{p(X)}\right)\right),
\end{aligned}$$

where $h(X)$ and $q(X)$ are such that $h(X) = g(X)^\beta \bmod p(X)$ and $q(X) = g(X)^{-\delta} \bmod p(X)$.

Now consider the worst-case error explicitly in terms of $q_d(X)$ as $E_d(q_d(X))$. Then we can write the worst-case error iteratively in the form

$$\begin{aligned}
E_d(q_d(X)) &:= e_{b^m,\alpha}((q_1(X), \ldots, q_d(X)), p(X)) \\
&= -1 + \frac{1}{b^m} \sum_{h=0}^{b^m-1} P_d(h) \\
&= -1 + \frac{1}{b^m} \sum_{h=0}^{b^m-1} P_{d-1}(h) + \frac{\gamma_d}{b^m} \sum_{h=0}^{b^m-1} P_{d-1}(h)\, \omega_\alpha(v_n(h(X)\,q_d(X)/p(X))) \\
&= e_{b^m,\alpha}((q_1(X), \ldots, q_{d-1}(X)), p(X)) + \frac{\gamma_d}{b^m} \sum_{h=0}^{b^m-1} P_{d-1}(h)\, \omega_\alpha(v_n(h(X)\,q_d(X)/p(X))) \\
&= e_{b^m,\alpha}((q_1(X), \ldots, q_{d-1}(X)), p(X))
\end{aligned}$$

$$+ \frac{\gamma_d}{b^m} \omega_\alpha(0) + \frac{\gamma_d}{b^m} \sum_{h=1}^{b^m-1} P_{d-1}(h)\, \omega_\alpha(v_n(h(X)\, q_d(X)/p(X))),$$

where the worst-case error for the zero-dimensional rule is 0. The main computational burden is now hidden in calculating the last sum which we can write in terms of the auxiliary function ω as an *extended sum*:

$$\sum_{h=1}^{b^m-1} \omega_\alpha(v_n(h(X)\, q_d(X)/p(X)))\, P_{d-1}(h) = \sum_{\beta=0}^{b^n-2} \omega(\beta - \delta)\, Q_{d-1}(\beta) \tag{20}$$

where δ is such that $q_d(X) = g(X)^{-\delta} \bmod p(X)$ and

$$Q_{d-1}(\beta) := \begin{cases} P_{d-1}(g(X)^\beta \bmod p(X)) & \text{if } \deg(g(X)^\beta \bmod p(X)) < m, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{Q}_d = (Q_d(0), \ldots, Q_d(b^n-2))$, then we have $\boldsymbol{Q}_d = \Pi^\top_{g(X)^{-1}} \boldsymbol{P}_d$, where

$$\Pi^\top_{g(X)^{-1}} = (a_{u,v})_{0 \le u \le b^n-2, 1 \le v < b^m}$$

and

$$a_{u,v} = \begin{cases} 1 & v(X) \equiv g(X)^u \bmod p(X), \\ 0 & \text{otherwise.} \end{cases}$$

Thus $\boldsymbol{Q}_d$ is obtained by permuting the elements of the vector $(\boldsymbol{P}_d, \boldsymbol{0}) \in \mathbb{R}^{b^n-1}$.

This extended sum (20), calculated for all possible choices of $q_d(X) = g(X)^{-\delta} \bmod p(X) \in (\mathbb{F}_b[X]/p(X))^\times$, i.e., $0 \le \delta < b^n - 1$, is in fact a *circular convolution* of length $b^n - 1$

$$\begin{aligned} S_d(\delta) &:= \sum_{\beta=0}^{b^n-2} \omega(\beta - \delta \bmod (b^n-1))\, Q_{d-1}(\beta) \\ &= \sum_{\beta=0}^{b^n-2} \omega(\beta)\, Q_{d-1}(\beta + \delta \bmod (b^n-1)). \end{aligned} \tag{21}$$

Calculating this convolution in the Fourier domain by the use of fast Fourier transforms (FFTs) takes time $O(b^n \log b^n)$, see [18] for a general reference. We obtain a construction cost for the fast component-by-component algorithm using FFTs of $O(sb^n \log b^n + b^n(c_\omega + \tilde{c}_v))$ using $O(b^n)$ memory. In other words, as $n = \alpha m$, the factor b^m in the original complexity has been reduced to $\alpha \log b^m$. Asymptotically this is always faster for increasing m.

We end this section with an overview of the complexities and their memory trade of:

Algorithm	Construction cost $= s\{\text{iteration cost}\} + \{\text{initialization cost}\}$	Memory cost
Straightforward	$s^2 b^n b^m (c_\omega + c_v)$	
Cache $\boldsymbol{P}_d$ vector	$sb^n b^m (c_\omega + c_v)$	b^m
Precalculate ω	$sb^n b^m + b^n(c_\omega + \tilde{c}_v)$	b^n
Fast convolution	$sb^n \log b^n + b^n(c_\omega + \tilde{c}_v)$	b^n

All these algorithms, except the fast convolution algorithm, have iteration times of $O(b^n b^m)$ and so they will all be asymptotically slower than the fast convolution algorithm. Timings on a real machine for $b = 2$ show the break even point to be at $m = 5$ for $\alpha = 2$ and $m = 6$ for $\alpha = 3$. The fast component-by-component algorithm based on fast convolution is given in Algorithm 2.

Algorithm 2 Fast CBC construction of higher order polynomial lattice rules

```
**Input:** base $b$ a prime, number of dimensions $s$, number of points $b^m$, smoothness $\alpha > 1$, and
weights $\boldsymbol{\gamma} = (\gamma_j)_{j \geq 1}$
**Output:** Generating vector $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X)) \in G_{b,n}^s$
Choose an irreducible polynomial $p(X) \in \mathbb{F}_b[X]$, $\deg(p) = n$ and $n = \alpha m$ and generator $g(X)$
Set $e_0 = 0$, $\boldsymbol{Q}_0 = \Pi_{g(X)^{-1}}^{\top} \begin{pmatrix} \mathbf{1}_{b^m \times 1} \\ \mathbf{0}_{(b^n - b^m) \times 1} \end{pmatrix}$
and $\boldsymbol{\omega} = \Big( \omega_\alpha(v_n(g(X)^\delta \pmod{p(X)}/p(X))) \Big)_{\delta = 0, \ldots, b^n - 2}$
**for** $d = 1$ **to** $s$ **do**
    $\boldsymbol{S}_d = \boldsymbol{\omega} \circledast \boldsymbol{Q}_{d-1}$    (by (fast) circular convolution)
    $\delta = \underset{0 \leq \delta < b^n - 1}{\operatorname{argmin}} \; S_d(\delta)$
    Set $q_d(X) = g(X)^\delta \pmod{p(X)}$
    Update/set $\boldsymbol{Q}_d$ and $e_d = e_{d-1} + \frac{\gamma_d}{b^m} \, \omega_\alpha(0) + \frac{\gamma_d}{b^m} \, S_d(\delta)$
**end for**
**return** $\boldsymbol{q}(X) = (q_1(X), \ldots, q_s(X))$
```

4 Calculation of the worst-case error

In this section we show how to calculate the infinite sum (18) which appears in the worst-case error formula from Lemma 1. In Theorem 2 we show that if x can be represented exactly with n digit precision in base b, then $\omega_\alpha(x)$ can be computed in $O(\alpha n)$ operations. Following that, in Section 4.2, Theorem 3 will state explicit forms for general $x \in [0, 1)$. More importantly, for $b = 2$, Corollary 1 gives explicit forms to compute $\omega_\alpha(x)$ exactly for arbitrary x and $\alpha = 2$ and 3 using elementary computer operations.

4.1 Technical definitions

Before we can show how to compute $\omega_\alpha(x)$ we need to introduce some technical notation which will be used in the proofs in the next section. To motivate the notation we first look at ω_α after expanding the definitions of $\operatorname{wal}_k(x)$ and $r_\alpha(k)$, see (6) and (7), and using the non-zero digit expansion, (4), of $k = \sum_{i=1}^{\#k} \kappa_{a_i} \, b^{a_i}$, where all $\kappa_{a_i} \neq 0$:

$$\begin{aligned} \omega_\alpha(x) &= \sum_{k=1}^{\infty} r_\alpha(k) \; \operatorname{wal}_k(x) \\ &= \sum_{\substack{k=1 \\ k = \sum_{i=1}^{\#k} \kappa_{a_i} \, b^{a_i}}}^{\infty} \; \prod_{i=1}^{\min(\alpha, \#k)} b^{-(a_i+1)} \, \exp(2\pi \mathrm{i} \; \kappa_{a_i} \xi_{a_i+1}/b) \; \underbrace{\prod_{i=\alpha+1}^{\#k} \exp(2\pi \mathrm{i} \; \kappa_{a_i} \xi_{a_i+1}/b)}_{\operatorname{wal}_{k'}(x)} . \end{aligned} \tag{22}$$

Due to definition (4) we have $a_1 > \cdots > a_{\#k} \geq 0$, i.e., κ_{a_1} is the most significant base b digit of k, etc. The second product in (22) can be seen as $\operatorname{wal}_{k'}(x)$ where k' is defined by

$k' = k - \sum_{i=1}^{\min(\alpha,\#k)} \kappa_{a_i}\, b^{a_i}$ and thus $0 \le k' < b^{a_{\min(\alpha,\#k)}}$. Now observe that the sum over all $k \ge 1$ can be expanded into multiple sums over all possible digit expansions for all k's with r digits for $r \ge 1$. That is, $\#k$ sums for the a_i together with companioning sums for the κ_{a_i} from 1 to $b-1$, i.e.,

$$\sum_{\substack{k=1 \\ k=\sum_{i=1}^{\#k} \kappa_{a_i}\, b^{a_i}}}^{\infty} G(k,x) = \sum_{r=1}^{\infty} \underbrace{\sum_{a_1=r-1}^{\infty} \cdots \sum_{a_r=0}^{a_{r-1}-1}}_{\substack{r \text{ sums s.t.} \\ \infty > a_1 > \cdots > a_r \ge 0}} \underbrace{\sum_{\kappa_{a_1}=1}^{b-1} \cdots \sum_{\kappa_{a_r}=1}^{b-1}}_{r \text{ independent sums}} G\left(\sum_{i=1}^{r} \kappa_{a_i}\, b^{a_i}, x\right), \tag{23}$$

where $G(k,x) = r_\alpha(k)\, \mathrm{wal}_k(x)$.

To simplify notation and stress the structure in what follows, we define the following *triangular sum operator* which sums over all $M \ge a_1 > \cdots > a_r \ge m$:

$$T_m^M(r)(g) := \underbrace{\sum_{a_1=m+r-1}^{M} \sum_{a_2=m+r-2}^{a_1-1} \cdots \sum_{a_r=m}^{a_{r-1}-1}}_{r \text{ sums}} g(a_1,\ldots,a_r), \tag{24}$$

and formally set the zero index sum, i.e., no sums to be taken, to be the identity mapping,

$$T_m^M(0)(g) := g.$$

Define the *concatenation* of two such operators as putting the sums next to each other:

$$(T_m^M(t)\, T_{m'}^{M'}(r-t))(g) := \underbrace{\underbrace{\sum_{a_1=m+t-1}^{M} \cdots \sum_{a_t=m}^{a_{t-1}-1}}_{t \text{ sums}} \underbrace{\sum_{a_{t+1}=m'+(r-t)-1}^{M'} \cdots \sum_{a_r=m'}^{a_{r-1}-1}}_{(r-t) \text{ sums}}}_{r \text{ sums}} g(a_1,\ldots,a_r),$$

i.e., having two independent ranges $M \ge a_1 > \cdots > a_t \ge m$ and $M' \ge a_{t+1} > \cdots > a_r \ge m'$. We remark that although these sums might look haggardly, the interpretation of the sum operator by their summation range is a natural way to reason about it as the following lemma shows.

Lemma 2. *For any $M \ge n > m$ we can split $T_m^M(r)$ into $r+1$ sets of two independent ranges:*

$$T_m^M(r) = \sum_{t=0}^{r} T_n^M(t)\, T_m^{n-1}(r-t).$$

Proof. Applying $T_m^M(r)$ to a function $g(a_1,\ldots,a_r)$ can be interpreted combinatorially as having to distribute r objects in $M-m+1$ different positions, numbered from m to M, which can each hold at most one object and then accumulating the result of applying the function g to this ensemble. It is trivial to note that we can split the range in two non-overlapping ranges and consider all partitions of r to distribute the objects over the two ranges. □

Specifically we find the following expansions of this summation operator for $r = 1, 2, 3$:

$$T_0^\infty(1) = T_0^{n-1}(1) + T_n^\infty(1), \tag{25a}$$

$$T_0^\infty(2) = T_0^{n-1}(2) + T_n^\infty(1)\, T_0^{n-1}(1) + T_n^\infty(2), \tag{25b}$$

$$T_0^\infty(3) = T_0^{n-1}(3) + T_n^\infty(1)\, T_0^{n-1}(2) + T_n^\infty(2)\, T_0^{n-1}(1) + T_n^\infty(3). \tag{25c}$$

As we will apply Lemma 2 to a product function, $g(a_1,\ldots,a_r) = g(a_1)\cdots g(a_r)$ it is useful to obtain the following result.

Lemma 3. *If the function $g(a_1,\dots,a_r)$ is of product form $g_1(a_1)\cdots g_r(a_r)$, then $T_0^{n-1}(r)(g)$, with n finite, can be calculated in $O(nr)$ operations.*

The proof of the lemma follows from the number of operations needed in Algorithm 3.

Algorithm 3 Compute $S_1 = T_0^{n-1}(r)(g_1(a_1)\cdots g_r(a_r))$ in $O(nr)$ operations

```
Initialize $S_1 = 0, \dots, S_r = 0$
for $a_r = 0$ to $n - r$ do
  $S_r = S_r + g_r(a_r)$
  for $t = 1$ to $r - 1$ do
    $S_{r-t} = S_{r-t} + S_{r-t+1}\, g_{r-t}(a_r + t)$
  end for
end for
return $S_1$
```

At the end of this algorithm for $t = 1,\dots,r$ we have the post conditions:

$$\begin{aligned} S_t &= \sum_{a_r=0}^{n-r} g_r(a_r) \sum_{a_{r-1}=a_r+1}^{n-r+1} g_{r-1}(a_{r-1}) \cdots \sum_{a_{t+1}=a_{t+2}+1}^{n-t-1} g_{t+1}(a_{t+1}) \sum_{a_t=a_{t+1}+1}^{n-t} g_t(a_t) \\ &= \sum_{a_t=r-t}^{n-t} g_t(a_t) \sum_{a_{t+1}=r-t-1}^{a_t-1} g_{t+1}(a_{t+1}) \cdots \sum_{a_{r-1}=1}^{a_{r-2}-1} g_{r-1}(a_{r-1}) \sum_{a_r=0}^{a_{r-1}-1} g_r(a_r) \\ &= T_0^{n-t}(r-t+1)\left(\prod_{i=t}^{r} g_i(a_i)\right). \end{aligned}$$

With a slight modification we can calculate all values of

$$S_t = T_0^{n-1}(r-t+1)\left(\prod_{i=t}^{r} g_i(a_i)\right), \qquad \text{for } t = 1,\dots,r. \tag{26}$$

For this we just let the outer loop run up to $n-1$ and make a modification in the inner loop to only conditionally update the value of S_{r-t} as long as $a_r < n-t$. The modified algorithm can be found in Algorithm 4. This algorithm is still $O(nr)$.

Algorithm 4 Compute all $S_t = T_0^{n-1}(r-t+1)(g_t(a_t)\cdots g_r(a_r))$, for $t = 1,\dots,r$, in $O(nr)$ operations

```
Initialize $S_1 = 0, \dots, S_r = 0$
for $a_r = 0$ to $n - 1$ do
  $S_r = S_r + g_r(a_r)$
  for $t = 1$ to $\min(r, n - a_r) - 1$ do
    $S_{r-t} = S_{r-t} + S_{r-t+1}\, g_{r-t}(a_r + t)$
  end for
end for
return $(S_1, \dots, S_r)$
```

4.2 A general algorithm for x having a fixed base b precision of n

We now consider calculating $\omega_\alpha(x)$ in base b for $x \in [0,1)$ which can be represented exactly with n digit precision in base b: $x = (0.\xi_1\xi_2\ldots\xi_n)_b = \sum_{i=1}^{n}\xi_i\, b^{-i}$. That is, x is actually a rational number v/b^n, $0 \le v < b^n$. This is exactly the situation that occurs in the component-by-component construction of Section 2.5, as the v_n function (14) exactly maps the Laurent series over $\mathbb{F}_b((X^{-1}))$ to rationals v/b^n with $0 \le v < b^n$.

Theorem 2. *Let $\alpha, b \ge 2$ be integers. Then for any $x = vb^{-n}$ with $n \ge 1$ and $0 \le v < b^n$, the value of $\omega_\alpha(vb^{-n})$ can be computed in at most $O(\alpha n)$ operations as follows: for $x = (0.\xi_1\xi_2\ldots\xi_n)_b$ and*

$$z(x,a_i) := \begin{cases} b-1 & \text{if } \xi_{a_i+1} = 0, \\ -1 & \text{if } \xi_{a_i+1} \ne 0, \end{cases} \qquad \text{and} \qquad \beta(x) = -\lfloor \log_b(x) \rfloor,$$

calculate the vectors

$$\begin{aligned} \boldsymbol{T}(x) &= (T_{\alpha-1},\ldots,T_1) \\ &:= \left(T_0^{n-1}(\alpha-1)\left(\prod_{i=1}^{\alpha-1} b^{-(a_i+1)} z(x,a_i)\right), \ \ldots, \ T_0^{n-1}(1)(b^{-(a_{\alpha-1}+1)} z(x,a_{\alpha-1})) \right), \\ \tilde{\boldsymbol{T}}(x) &= (\tilde{T}_\alpha,\ldots,\tilde{T}_1) \\ &:= \left(T_0^{n-1}(\alpha)\left(b^{a_\alpha}[a_\alpha < \beta(x)-1] \prod_{i=1}^{\alpha} b^{-(a_i+1)} z(x,a_i)\right), \ \ldots, \right. \\ &\qquad\qquad \left. T_0^{n-1}(1)\left(b^{a_\alpha}[a_\alpha < \beta(x)-1] b^{-(a_\alpha+1)} z(x,a_\alpha)\right)\right), \end{aligned}$$

where

$$[a_\alpha < \beta(x)-1] := \begin{cases} 1 & \text{if } a_\alpha < \beta(x)-1, \\ 0 & \text{otherwise.} \end{cases}$$

The vectors $\boldsymbol{T}$ and $\tilde{\boldsymbol{T}}$ can both be computed by Algorithm 4. Now set

$$\begin{aligned} \boldsymbol{C} &= (C_0,\ldots,C_{\alpha-1}) := \left(b^{-nt} \prod_{i=1}^{t} \frac{b-1}{b^i-1} \right)_{t=0,\ldots,\alpha-1}, \\ \bar{\boldsymbol{C}} &= (\bar{C}_0, \bar{C}_1, \ldots, \bar{C}_{\alpha-1}) := (C_0, C_0+C_1, \ldots, C_0+\cdots+C_{\alpha-1}), \end{aligned}$$

where $C_0 = \bar{C}_0 = 1$, then for $0 \le v < b^n$

$$\omega_\alpha(vb^{-n}) = \begin{cases} \bar{\boldsymbol{C}}_{0:\alpha-2} \cdot \boldsymbol{T}(vb^{-n}) + (\bar{C}_{\alpha-1} - 1) + \boldsymbol{C} \cdot \tilde{\boldsymbol{T}}(vb^{-n}) & \text{if } 0 < v < b^n, \\ \displaystyle\sum_{r=1}^{\alpha-1} \prod_{i=1}^{r} \frac{b-1}{b^i-1} + \frac{b-1}{b^\alpha - b} \prod_{i=1}^{\alpha-1} \frac{b-1}{b^i-1} & \text{if } v = 0, \end{cases}$$

here $\boldsymbol{a} \cdot \boldsymbol{b}$ denotes the dot product and $\bar{\boldsymbol{C}}_{0:\alpha-2}$ is a vector of the first $\alpha - 1$ components of $\bar{\boldsymbol{C}}$.

Proof. We start from expression (22). For ease of manipulation we consider two different cases of the base b expansions for integer $k > 0$:

1. Integers k which have between 1 and $(\alpha-1)$ non-zero digits in base b:

$$k = \sum_{i=1}^{\#k} \kappa_{a_i}\, b^{a_i}, \qquad \text{where } 1 \leq \#k \leq \alpha - 1.$$

2. Integers k which have α or more non-zero digits in base b:

$$k = \sum_{i=1}^{\#k} \kappa_{a_i}\, b^{a_i} = \sum_{i=1}^{\alpha} \kappa_{a_i}\, b^{a_i} + k', \qquad \text{where } \#k \geq \alpha \text{ and } 0 \leq k' < b^{a_\alpha}.$$

As such we consider, for $0 < x < 1$,

$$\omega_\alpha(x) = \sum_{\substack{k=1 \\ \#k<\alpha \\ k=\sum_{i=1}^{\#k} \kappa_{a_i}\, b^{a_i}}}^{\infty} \prod_{i=1}^{\#k} b^{-(a_i+1)} \exp(2\pi\mathrm{i}\, \kappa_{a_i} \xi_{a_i+1}/b)$$
$$+ \sum_{\substack{k=1 \\ \#k\geq\alpha \\ k=\sum_{i=1}^{\alpha} \kappa_{a_i}\, b^{a_i}+k' \\ 0\leq k'<b^{a_\alpha}}}^{\infty} \mathrm{wal}_{k'}(x) \prod_{i=1}^{\alpha} b^{-(a_i+1)} \exp(2\pi\mathrm{i}\, \kappa_{a_i} \xi_{a_i+1}/b).$$

We will now expand these outer sums as in (23), but first note

$$z(x,a_i) = \sum_{\kappa_{a_i}=1}^{b-1} \exp(2\pi\mathrm{i}\, \kappa_{a_i} \xi_{a_i+1}/b) = \begin{cases} b-1 & \text{if } \xi_{a_i+1} = 0, \\ -1 & \text{if } \xi_{a_i+1} \neq 0, \end{cases} \tag{27}$$

to move all the independent κ_{a_i} sums, cf. (23), into the product function. Further, denote by $\beta(x)$ the power of b^{-1} of the first non-zero digit in the base b expansion of $x \in [0,1)$, then the sum over k' for case 2 becomes

$$\sum_{k'=0}^{b^{a_\alpha}-1} \mathrm{wal}_{k'}(x) = \begin{cases} b^{a_\alpha} & \text{if } a_\alpha < \beta(x) - 1, \text{ i.e., } x = (0.\underbrace{0\ldots\ldots0}_{\text{at least } a_\alpha} * * * \ldots)_b, \\ 0 & \text{otherwise} \end{cases}$$
$$=: b^{a_\alpha} [a_\alpha < \beta(x) - 1],$$

where the last line uses Iverson notation. Introducing the sum operator (24) we obtain

$$\omega_\alpha(x) = \sum_{r=1}^{\alpha-1} T_0^\infty(r) \Bigg(\prod_{i=1}^{r} b^{-(a_i+1)} z(x,a_i) \Bigg) + T_0^\infty(\alpha) \Bigg(b^{a_\alpha} [a_\alpha < \beta(x) - 1] \prod_{i=1}^{\alpha} b^{-(a_i+1)} z(x,a_i) \Bigg).$$

Since our function is a product function, it is convenient to only deal with the operators, which then shortens the notation.

We now deal with the two cases separately. For case 1, $1 \leq r \leq \alpha - 1$, we apply Lemma 2 and manipulate the following expression

$$\sum_{r=1}^{\alpha-1} T_0^\infty(r) = \sum_{r=1}^{\alpha-1} \sum_{t=0}^{r} T_n^\infty(r-t)\, T_0^{n-1}(t) = \sum_{t=1}^{\alpha-1} \Bigg(\sum_{r=t}^{\alpha-1} T_n^\infty(r-t) \Bigg) T_0^{n-1}(t) + \sum_{r=1}^{\alpha-1} T_n^\infty(r).$$

By assumption of the n digit base b precision of x the T_n^∞ sums do not depend on x. As we show next, they can be calculated off line in closed form. That means we are left to deal with the $T_0^{n-1}(t)(g_{r-t+1}(a_{r-t+1})\cdots g_r(a_r))$ for $t = 1, \ldots, \alpha-1$. We can use Algorithm 3 for each of these terms, but as they are nested, we can use Algorithm 4 to calculate them all at once in time $O(\alpha n)$ upon calculating $T_0^{n-1}(\alpha-1)$. The T_n^∞ sums are given by:

$$\begin{aligned} T_n^\infty(t)\left(\prod_{i=1}^{t} b^{-(a_i+1)}(b-1)\right) &= (b-1)^t b^{-t} \sum_{a_1=n+t-1}^{\infty} b^{-a_1} \sum_{a_2=n+t-2}^{a_1-1} b^{-a_2} \cdots \sum_{a_t=n}^{a_{t-1}-1} b^{-a_t} \\ &= (b-1)^t b^{-t} \sum_{a_t=n}^{\infty} b^{-a_t} \cdots \sum_{a_2=a_3+1}^{\infty} b^{-a_2} \sum_{a_1=a_2+1}^{\infty} b^{-a_1} \\ &= b^{-nt} \prod_{i=1}^{t} \frac{b-1}{b^i-1}. \end{aligned} \tag{28}$$

For case 2, $\#k \geq \alpha$, we can also apply Lemma 2 to obtain

$$\begin{aligned} T_0^\infty(\alpha) &= \sum_{t=0}^{\alpha} T_n^\infty(t) T_0^{n-1}(\alpha-t) \\ &= T_0^{n-1}(\alpha) + T_n^\infty(1)T_0^{n-1}(\alpha-1) + \cdots + T_n^\infty(\alpha-1)T_0^{n-1}(1) + T_n^\infty(\alpha), \end{aligned}$$

which is applied to the function

$$b^{a_\alpha}[a_\alpha < \beta(x)-1]\left(\prod_{i=1}^{\alpha} b^{-(a_i+1)} z(x,a_i)\right).$$

The T_n^∞ sums here become

$$\begin{aligned} T_n^\infty(t)\left(\prod_{i=1}^{t} b^{-(a_i+1)}(b-1)\right) &= b^{-nt}\prod_{i=1}^{t}\frac{b-1}{b^i-1}, \qquad \text{for } t<\alpha, \\ \text{and} \qquad T_n^\infty(\alpha)\left(b^{a_\alpha}[a_\alpha < \beta(x)-1]\prod_{i=1}^{\alpha} b^{-(a_i+1)}(b-1)\right) &= 0. \end{aligned}$$

For $x \neq 0$ the condition $[a_\alpha < \beta(x)-1]$ makes it such that $T_n^\infty(\alpha) = 0$ as $a_\alpha \geq n$ (and $\beta(x) \leq n$ by assumption). The other T_n^∞ values are the same as for case 1, and we can use the closed form (28). Also here we use Algorithm 4 to calculate all the sums $T_0^{n-1}(\alpha-t)$ in $O(\alpha n)$ upon calculating $T_0^{n-1}(\alpha)$.

When $x = 0$ there is no need to consider splitting at a given n. To obtain $T_0^\infty(\alpha)$ we can use a similar derivation as for (28) to obtain a closed form:

$$\begin{aligned} T_0^\infty(\alpha)\left(b^{a_\alpha}\prod_{i=1}^{\alpha} b^{-(a_i+1)}(b-1)\right) &= \sum_{a_\alpha=0}^{\infty} b^{-1}(b-1)\, T_{a_\alpha+1}^\infty(\alpha-1)\left(\prod_{i=1}^{\alpha-1} b^{-(a_i+1)}(b-1)\right) \\ &= \sum_{a_\alpha=0}^{\infty} b^{-1}(b-1)\, b^{-(a_\alpha+1)(\alpha-1)} \prod_{i=1}^{\alpha-1}\frac{b-1}{b^i-1} \\ &= \frac{b-1}{b^\alpha-b}\prod_{i=1}^{\alpha-1}\frac{b-1}{b^i-1}. \end{aligned}$$

Again (28) can be used to calculate the $T_0^\infty(r)$ for $r = 1, \ldots, \alpha-1$. This completes the proof. □

4.3 Explicit forms for arbitrary x and small α

Theorem 2 uses the fact that at most the first n digits of the coordinates of the polynomial lattice rule can be non-zero; it is hence not surprising that the resulting computational complexity depends on n. Here we take a similar approach, but explicitly look at the non-zero digits of x; this will turn out to be a favorable approach in case of $b = 2$, for which we find explicit expressions in Corollary 1. We will use the following similar notation as was set up in the beginning of Section 2: Let the non-zero digits base b expansion of $x = (0.\xi_1\xi_2\ldots)_b \in [0, 1)$ be given by

$$x = \sum_{i=1}^{\#x} \xi_{a_i} b^{-a_i},$$

where $1 \le a_1 < \cdots < a_{\#x}$, $\xi_{a_i} \in \{1, \ldots, b-1\}$. In particular, we will see that the power of b^{-1} for the most significant digit of x, i.e., a_1, plays a pivotal role. For $x = 0$ we set $a_1 = \infty$ and $\#x = 0$.

Theorem 3. *For $x \in [0, 1)$ with non-zero digit base b expansion*

$$x = \sum_{i=1}^{\#x} \xi_{a_i} b^{-a_i}, \qquad 1 \le a_1 < \cdots < a_{\#x}, \quad \xi_{a_i} \in \{1, \ldots, b-1\},$$

we have

$$\begin{aligned}\omega_2(x) &= s_1(x) + \tilde{s}_2(x),\\ \omega_3(x) &= s_1(x) + s_2(x) + \tilde{s}_3(x),\end{aligned}$$

where

$$\begin{aligned}s_1(x) &:= 1 - b\sum_{j=1}^{\#x} b^{-a_j},\\ s_2(x) &:= \frac{1}{b+1} - b(b-2)\frac{1}{2}\left(\left(\sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right) - \sum_{j=1}^{\#x} b^{-2a_j}\right)\\ &\quad - b(b-1)\left(\frac{1}{b-1} - \sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right),\end{aligned}$$

and for $x \neq 0$ we have

$$\begin{aligned}\tilde{s}_2(x) &:= b^{-1} - 2b^{-a_1} - b^{-(a_1+1)} - (a_1 b - a_1 - b)\sum_{j=1}^{\#x} b^{-a_j},\\ \tilde{s}_3(x) &:= (b+1)^{-2} b^{-1} b^{-2a_1}((a_1-2)b^2 - (a_1-1)b^4 + b^{2a_1})\\ &\quad - b^{-a_1}((a_1-2)b - (a_1-1)b^2 + b^{a_1})\sum_{j=1}^{\#x} b^{-a_j}\\ &\quad - b^{-1}(b-1)(a_1-1)b^{-2a_1} s_1(b^{a_1}x - 1)\\ &\quad + b^{-1}((b-1)(a_1-1)-1)b^{-2a_1} s_2(b^{a_1}x - 1).\end{aligned}$$

For $x = 0$ we set $\tilde{s}_2(0) = b^{-1}$ and $\tilde{s}_3(0) = b^{-1}(b+1)^{-2}$.

Proof. We start in exactly the same way as in Theorem 2, that is, we split $\omega_\alpha(x)$ into α parts (cf. the $\alpha-1$ parts in case 1 plus the case 2 case in the proof of Theorem 2):

$$\omega_\alpha(x) = \sum_{k=1}^{\infty} r_\alpha(k) \ \mathrm{wal}_k(x) = \sum_{r=1}^{\alpha-1} s_r(x) + \tilde{s}_\alpha(x),$$

where $s_r(x)$ contains all k with exactly r digits non-zero and $\tilde{s}_\alpha(x)$ contains all k with at least α digits non-zero. We only show the derivation of the formulae for s_1 and s_2 as examples. The ones for $\tilde{s}_2$ and $\tilde{s}_3$ can be obtained similarly. With z as in (27) we find

$$\begin{aligned} s_1(x) &= \sum_{\ell=0}^{\infty} b^{-(\ell+1)} z(x,\ell) \\ &= (b-1)\sum_{\ell=0}^{\infty} b^{-(\ell+1)} + (-(b-1)+(-1))\sum_{\ell=0}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} \neq 0] \\ &= 1 - b\sum_{j=1}^{\#x} b^{-a_j}. \end{aligned}$$

Likewise for s_2:

$$\begin{aligned} s_2(x) &= \sum_{\ell'=0}^{\infty} b^{-(\ell'+1)} z(x,\ell') \sum_{\ell=\ell'+1}^{\infty} b^{-(\ell+1)} z(x,\ell) \\ &= \frac{1}{b+1} \\ &\quad - b(b-2)\sum_{\ell'=0}^{\infty} b^{-(\ell'+1)}[\xi_{\ell'+1} \neq 0] \sum_{\ell=\ell'+1}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} \neq 0] \qquad (*) \\ &\quad - b(b-1)\sum_{\ell'=0}^{\infty} b^{-(\ell'+1)}[\xi_{\ell'+1} \neq 0] \sum_{\ell=\ell'+1}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} = 0] \qquad (**) \\ &\quad - b(b-1)\sum_{\ell'=0}^{\infty} b^{-(\ell'+1)}[\xi_{\ell'+1} = 0] \sum_{\ell=\ell'+1}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} \neq 0]. \qquad (**) \end{aligned}$$

This is a combinatorial formulation in terms of the possibilities for the digits of x. The two last lines, marked by (**), can be combined and interpreted as summing over all possible pairs of digits of x of which exactly one is non-zero. This then simplifies to two decoupled sums since a digit cannot be at the same time zero and non-zero:

$$\begin{aligned} \sum_{\ell'=0}^{\infty} b^{-(\ell'+1)}[\xi_{\ell'+1} = 0] \sum_{\ell=0}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} \neq 0] &= \left(\sum_{\ell'=0}^{\infty} b^{-(\ell'+1)} - \sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right) \\ &= \left(\frac{1}{b-1} - \sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right). \end{aligned}$$

The other double sum, marked by (*), can also be interpreted combinatorially: the sum is taken over all ordered pairs of non-zero digits of x. We can write:

$$\sum_{\ell'=0}^{\infty} b^{-(\ell'+1)}[\xi_{\ell'+1} \neq 0] \sum_{\ell=\ell'+1}^{\infty} b^{-(\ell+1)}[\xi_{\ell+1} \neq 0] = \sum_{j=1}^{\#x} b^{-a_j} \sum_{j'=j+1}^{\#x} b^{-a_{j'}}$$

$$= \frac{1}{2}\left(\left(\sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right) - \sum_{j=1}^{\#x} b^{-2a_j}\right).$$

Thus

$$\begin{aligned} s_2(x) = \frac{1}{b+1} & \\ & - b(b-2)\frac{1}{2}\left(\left(\sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right) - \sum_{j=1}^{\#x} b^{-2a_j}\right) \\ & - b(b-1)\left(\frac{1}{b-1} - \sum_{j=1}^{\#x} b^{-a_j}\right)\left(\sum_{j=1}^{\#x} b^{-a_j}\right). \end{aligned}$$

□

The case where b equals 2 is of greatest practical importance, since in that case the matrix-vector product (10) over $\mathbb{F}_b$ to generate the nodes of the QMC rule can be calculated most efficiently by using the bitwise operations of the computer. Additionally

$$\sum_{j=1}^{\#x} b^{-a_j} = x \qquad \text{when} \qquad b = 2.$$

By specializing the previous result to $b = 2$ we obtain the following explicit formulae.

Corollary 1. *For base $b = 2$ we obtain the following explicit results:*

$$\begin{aligned} \omega_2(x) &= s_1(x) + \tilde{s}_2(x), \\ \omega_3(x) &= s_1(x) + s_2(x) + \tilde{s}_3(x), \end{aligned}$$

where

$$\begin{aligned} s_1(x) &= 1 - 2x, & s_2(x) &= 1/3 - 2(1-x)x, \\ \tilde{s}_2(x) &= (1 - 5t_1)/2 - (a_1 - 2)x, & \tilde{s}_3(x) &= (1 - 43t_2)/18 + (5t_1 - 1)x + (a_1 - 2)x^2, \end{aligned}$$

with, for $0 < x < 1$,

$$a_1 = -\lfloor \log_2(x) \rfloor, \qquad t_1 := 2^{-a_1}, \qquad t_2 := 2^{-2a_1},$$

and $a_1 = 0$, $t_1 = 0$ and $t_2 = 0$ when $x = 0$.

5 Numerical tests

We compare the explicit construction from [5], with the CBC algorithm based on (fast) circular convolution presented in this paper, i.e., Algorithm 2. From [5] we note that, to obtain higher order digital nets of high quality, the underlying point sets in the construction should have small values of t. Consequently, we use Niederreiter-Xing points generated by Pirsic's implementation, see [23], to obtain the digital (t', m, sd)-nets. In Table 1 we present a typical result for $b = 2$, $\alpha = 2$ and $s = 5$ and two choices of weights $\gamma_j = 0.9^j$ and $\gamma_j = j^{-2}$. For the CBC construction

$\gamma_j = 0.9^j$	e_{CBC}	e_{explicit}
$m = 5$	0.9291	1.0930
$m = 6$	0.4085	0.4259
$m = 7$	0.1778	0.1984
$m = 8$	0.0747	0.0980
$m = 9$	0.0312	0.0403
$m = 10$	0.0128	0.0168
$m = 11$	0.0052	0.0071
$m = 12$	0.0020	0.0027

$\gamma_j = j^{-2}$	e_{CBC}	e_{explicit}
$m = 5$	0.028917	0.096254
$m = 6$	0.009912	0.014542
$m = 7$	0.003427	0.005895
$m = 8$	0.001175	0.002356
$m = 9$	0.000406	0.000827
$m = 10$	0.000139	0.000290
$m = 11$	0.000046	0.000091
$m = 12$	0.000014	0.000034

Table 1: Comparison of the worst-case errors of CBC construction and explicit construction for $b = 2$, $s = 5$, $\alpha = 2$

$b = 2$, $m = 10$, $\alpha = 2$: $n = 20$, $p = 1179649$					
j	1	2	3	4	5
q_j	453270	920860	324514	394664	106142
e	2.14e-6	4.55e-5	6.27e-4	3.75e-3	1.30e-2
j	6	7	8	9	10
q_j	587632	279628	676057	626366	856775
e	3.39e-2	7.45e-2	1.43e-1	2.51e-1	4.08e-1

$b = 2$, $m = 12$, $\alpha = 2$: $n = 24$, $p = 28311553$					
j	1	2	3	4	5
q_j	2028384	13051202	839202	14647583	6874738
e	1.34e-7	3.44e-6	6.58e-5	4.72e-4	2.02e-3
j	6	7	8	9	10
q_j	6522492	13569662	9821234	10570369	406897
e	6.09e-3	1.45e-2	2.97e-2	5.46e-2	9.19e-2

Table 2: Higher order rules up to 10 dimensions for $b = 2$, $\gamma_j = 0.9^j$ and $\alpha = 2$

we used the primitive polynomials from [27]. The numerical data in all our tests shows that the new construction produces better results.

For reference we conclude the paper with tables showing the generating vectors and worst case errors of higher order polynomial lattice rules in base 2 constructed using the new algorithm. All polynomials are given by their canonical integer representation which is the polynomial evaluated at $X = b = 2$ (note that these are different polynomials than those from [27]). The results can be found in Table 2 and Table 3 for $\alpha = 2$ and $\alpha = 3$ respectively.

References

[1] J. Baldeaux and J. Dick. QMC rules of arbitrary high order: Reproducing kernel Hilbert space approach. *Constr. Approx.*, 30(3):495–527, 2009.

[2] J. Baldeaux, J. Dick, J. Greslehner, and F. Pillichshammer. Construction algorithms for higher order polynomial lattice rules. *J. Complexity*, 27(3–4):281–299, 2011.

[3] J. Dick. Higher order scrambled digital nets achieve the optimal rate of the root mean square error for smooth integrands. *Ann. Statist.*, 39(3):1372–1398, 2011.

$b=2$, $m=7$, $\alpha=3$: $n=21$, $p=2621441$					
j	1	2	3	4	5
q_j	1492861	1022044	1785216	215936	1978368
e	2.02e-6	5.24e-4	8.20e-3	4.05e-2	1.22e-1
j	6	7	8	9	10
q_j	1197580	1837814	485609	1636853	48810
e	2.82e-1	5.54e-1	9.80e-1	1.60	2.48

$b=2$, $m=8$, $\alpha=3$: $n=24$, $p=28311553$					
j	1	2	3	4	5
q_j	10844342	2604270	5720893	8141702	3831799
e	2.51e-7	8.85e-5	2.43e-3	1.45e-2	4.95e-2
j	6	7	8	9	10
q_j	3616803	15701694	7750425	2240926	493873
e	1.21e-1	2.49e-1	4.54e-1	7.59e-1	1.19

Table 3: Higher order rules up to 10 dimensions for $b = 2$, $\gamma_j = 0.9^j$ and $\alpha = 3$

[4] J. Dick. Explicit constructions of quasi-Monte Carlo rules for the numerical integration of high dimensional periodic functions. *SIAM J. Numer. Anal.*, 45:2141–2176, 2007.

[5] J. Dick. Walsh spaces containing smooth functions and quasi-Monte Carlo rules of arbitrary high order. *SIAM J. Numer. Anal.*, 46(3):1519–1553, 2008.

[6] J. Dick. The decay of the Walsh coefficients of smooth functions. *Bull. Austral. Math. Soc.*, 80:430–453, 2009.

[7] J. Dick, P. Kritzer, F. Pillichshammer, and W. C. Schmid. On the existence of higher order polynomial lattices based on a generalized figure of merit. *J. Complexity*, 23(4–6):581–593, 2007.

[8] J. Dick, F. Y. Kuo, F. Pillichshammer, and I. H. Sloan. Construction algorithms for polynomial lattice rules for multivariate integration. *Math. Comp.*, 74(252):1895–1921, 2005.

[9] J. Dick and F. Pillichshammer. Multivariate integration in weighted Hilbert spaces based on Walsh functions and weighted Sobolev spaces. *J. Complexity*, 21(2):149–195, 2005.

[10] J. Dick and F. Pillichshammer. Strong tractability of multivariate integration of arbitrary high order using digitally shifted polynomial lattice rules. *J. Complexity*, 23(4–6):436–453, 2007.

[11] J. Dick and F. Pillichshammer. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration.* Cambridge University Press, 2010.

[12] N. M. Korobov. The approximate computation of multiple integrals / Approximate evaluation of repeated integrals. *Dokl. Akad. Nauk SSSR*, 124:1207–1210, 1959. In Russian. English translation of the theorems in Mathematical Reviews by Stroud.

[13] N. M. Korobov. *Number-Theoretic Methods in Approximate Analysis.* Goz. Izdat. Fiz.-Math., 1963. In Russian. English translation of results on optimal coefficients in [28].

[14] G. Larcher, A. Lauss, H. Niederreiter, and W. C. Schmid. Optimal polynomials for (t, m, s)-nets and numerical integration of multivariate Walsh series. *SIAM J. Numer. Anal.*, 33(6):2239–2253, 1996.

[15] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Number 63 in Regional Conference Series in Applied Mathematics. SIAM, 1992.

[16] E. Novak and H. Woźniakowski. *Tractability of Multivariate Problems — Volume I: Linear Information*, volume 6 of *EMS Tracts in Mathematics.* European Mathematical Society Publishing House, 2008.

[17] E. Novak and H. Woźniakowski. *Tractability of Multivariate Problems — Volume II: Standard Information for Functionals*, volume 12 of *EMS Tracts in Mathematics.* European Mathematical Society Publishing House, 2010.

[18] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, 2nd edition, 1982.

[19] D. Nuyens. *Fast Construction of Good Lattice Rules.* PhD thesis, Dept. of Computer Science, K.U.Leuven, 2007.

[20] D. Nuyens and R. Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. *Math. Comp.*, 75(254):903–920, 2006.

[21] D. Nuyens and R. Cools. Fast component-by-component construction, a reprise for different kernels. In H. Niederreiter and D. Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 371–385. Springer-Verlag, 2006.

[22] A. B. Owen. Monte Carlo variance of scrambled net quadrature. *SIAM J. Numer. Anal.*, 34(5):pp. 1884–1910, 1997.

[23] G. Pirsic. A software implementation of Niederreiter-Xing sequences. In K. T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 434–445. Springer-Verlag, 2002.

[24] W. C. Schmid. Improvements and extensions of the "Salzburg Tables" by using irreducible polynomials. In H. Niederreiter and J. Spanier, editors, *Monte Carlo and Quasi-Monte Carlo Methods 1998*, pages 436–447, Berlin, 2000. Springer-Verlag.

[25] I. H. Sloan and A. V. Reztsov. Component-by-component construction of good lattice rules. *Math. Comp.*, 71(237):263–273, 2002.

[26] I. H. Sloan and H. Woźniakowski. When are quasi-Monte Carlo algorithms efficient for high dimensional integrals? *J. Complexity*, 14(1):1–33, 1998.

[27] W. Stahnke. Primitive binary polynomials. *Math. Comp.*, 27(124):977–980, 1973.

[28] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Automatic Computation. Prentice-Hall, 1971.

[29] I. F. Šarygin. Lower bounds for the error of quadrature formulas on classes of functions. *U.S.S.R. Comput. Math. and Math. Phys.*, 3:489–497, 1965. Translation from Russian *Zh. Vychisl. Mat. Mat. Fiz.*, 3:370–376, 1963.