KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Leuven

FAST CONSTRUCTION OF GOOD LATTICE RULES

Promotor:
Prof. Dr. ir. R. Cools

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Dirk NUYENS

April 2007

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Leuven

FAST CONSTRUCTION OF GOOD LATTICE RULES

Jury:
Prof. Dr. ir. P. Van Houtte, voorzitter
Prof. Dr. ir. R. Cools, promotor
Prof. Dr. ir. A. Haegemans
Prof. Dr. ir. S. Vandewalle
Prof. Dr. A. Bultheel
Prof. Dr. R. Boel
(Universiteit Gent, Belgium)
Prof. Dr. A. Genz
(Washington State University, USA)
Prof. Dr. I. H. Sloan
(University of New South Wales, Australia)

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Dirk NUYENS

U.D.C. 519.64

April 2007

D/2007/7515/38

ISBN 978-90-5682-804-2

Fast construction of good lattice rules

Dirk Nuyens
Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium

Abstract

We develop a fast algorithm for the construction of good rank-1 lattice rules which are a quasi-Monte Carlo method for the approximation of multivariate integrals. A popular method to construct such rules is the component-by-component algorithm which is able to construct good lattice rules that achieve the optimal theoretical rate of convergence. The construction time of this algorithm is $O(s^2n^2)$, or $O(sn^2)$ when using $O(n)$ memory, for an s-dimensional lattice rule with n points.

We show how to construct good lattice rules in time $O(sn\log(n))$, using $O(n)$ memory, by means of a new algorithm, called the fast component-by-component algorithm. First this is shown for the base case when n is a prime number and the underlying function space is a weighted, shift-invariant and tensor-product reproducing kernel Hilbert space. Then we show that, by a minor increase in construction cost, also more generally weighted function spaces can be handled by the fast algorithm. In particular we show this for order-dependent weights.

When n is not a prime number it turns out that fast construction is also possible, although the construction is more involved for numbers n which have a large number of unique prime factors. An additional advantage is obtained when choosing n to be a prime power, since then the rules are embedded for increasing powers of the prime. Using this embedding, we propose a new fast algorithm to construct lattice sequences which can be used point by point.

Two natural extensions of the algorithm are the construction of polynomial lattice rules and so called copy rules. We show that also here the fast component-by-component algorithm can be applied. The quality of the constructed point sets is finally demonstrated on some finance and statistics examples.

Preface

This thesis describes the development of a fast version of the component-by-component algorithm for the construction of lattice rules. It is the result of four and a half years of research, struggle and frustration, but also of four and a half years of excitement, fun and enlightenment.

Thanks...

At the start I never told Ronald that one of my biggest deficiencies is that I am easily distracted, but after four and a half years I am most certain that he now knows all about it. Therefore I want to thank Ronald, especially for his patience, but also for his profound knowledge, guidance and sound advice into this world of academic research. I would also like to heartily thank Frances Kuo for the lovely friendship, for her own very inspiring PhD thesis (which I got to read on the very first day I started) and for investing a lot of time in me, even at the other end of the globe—and to that end I would like to thank her husband Roger as well. I am also very grateful to Ian Sloan, especially for the advice he gave me that I need other advice and the memorable saying that, to build an academic career, you need to work like the dogs. During the past four years I found myself in the best research community one could think of; this thesis would not have been a success without the friendly discussions I had with lots of researchers. I would like to thank them for the inspiring force, especially Henryk Woźniakowski and Fred Hickernell. Thanks also to Stephen Joe, Ben Waterhouse and Peter Kritzer.

I would also like to thank all the members of the jury for many useful comments which certainly improved the text. I am very proud to have two excellent specialists, Alan Genz and Ian Sloan, in my jury.

A special thank you to all who made this work fun. Most certainly I thank Tim Volodine, Tim Pillards, Tine Verhanneman, Daan Huybrechs, Joris Van Deun (my local hero, who is not so local anymore), Hendrik Speleers and Bart Verleye. Thanks also to Bart Vandereycken, Bart Vandewoestyne, Yvette Vanberghen, Joris Vanbiervliet and Mike Maton for some fine help in the dark days of thesis writing.

Ik wil ook mijn ouders en zus bedanken voor ongeveer alles wat ik kan bedenken. Hun steun heeft me hier gebracht. Ook bedank ik Sarah's vader Francis voor een exacte vervanging van mijn veel te oude laptop, waarop een groot deel van deze tekst tot stand kwam, en nog veel meer.

Aan Sarah ben ik veel verschuldigd. Vrije tijd maken, is en was niet mijn sterkste kant, daarom wil ik je bedanken omdat jij in mij gelooft. Bedankt ook voor Liene, onze schattige dochter die in haar prille bestaan veelvuldig heeft bijgedragen tot deze tekst, o.a. door me te verplichten tot het nemen van de nodige pauzes, het verwijderen van grote stukken tekst (alsook de letter 'n' van mijn toetsenbord) en het meebrengen van de mooie zonnestralen uit Australië.

Dirk
Maart 2007

Acknowledgement

This research is part of a project financially supported by the Onderzoeksfonds K.U.Leuven / Research Fund K.U.Leuven. Additional support from the Fonds Wetenschappelijk Onderzoek – Vlaanderen / Research Foundation – Flanders and the Australian Research Council under its Centres of Excellence program is gratefully acknowledged.

Voor Liene.

Voor de muziek die was,
Mario, Danny en Bart.

Contents

List of Figures

List of Listings

Notation

s	number of dimensions
n	number of points
d	a divisor
p	a prime
j	running index for dimensions
i	imaginary unit
$\lvert A\rvert$	cardinality of the set
$\mathbb{R}$	set of real numbers
$\mathbb{Z}$	set of integers
$\mathbb{Z}_*$	set of non-negative integers
$\mathbb{Z}_n$	set of integers modulo n, i.e., $\mathbb{Z}/n\mathbb{Z}$
U_n	set of units modulo n, i.e., $\mathbb{Z}_n^\times$
$\mathbb{Q}$	set of rational numbers
$Q(f; P_n)$	mostly a quasi-Monte Carlo cubature rule Q applied to the function f with point set P_n, see page 2
P_n	a point set of size n over $[0,1]^s$
$x_j^{(k)}$	the jth component of $\boldsymbol{x}_k$, the kth element of a point set P_n
P_α	classical lattice criteria, worst-case error in E_α^s, see page 21
$\mathrm{vol}(J)$	volume of J
$\mathrm{discr}(J, P_n)$	local discrepancy of the point set P_n over J, see page 3
$V(f)$	variation of f, see page 6
$\sigma^2(f)$	variance of f
s^2	sample variance
$q \times P_n$	q randomizations of the point set P_n
$\mathcal{D}_s$	set of indices $\{1, \ldots, s\}$

$\mathfrak{u}$	subset of indices, $\mathfrak{u} \subseteq \mathcal{D}_s$
$f(\boldsymbol{x}_\mathfrak{u}, \mathbf{1})$	the function f restricted to the coordinates in $\mathfrak{u}$ and taking on the value 1 for coordinates not in $\mathfrak{u}$
$\{x\}$	fractional part of x, i.e., $\{x\} = x \bmod 1$
$\{\boldsymbol{x}\}$	componentwise fractional part of $\boldsymbol{x}$
$\boldsymbol{x}$	vector
$\boldsymbol{x}^\top$	transpose of $\boldsymbol{x}$
$\boldsymbol{A}$	matrix
L_p	space of pth power integrable functions (over $[0,1]^s$)
$\mathcal{H}$	Hilbert space of functions
$\mathcal{H}(K)$	reproducing kernel Hilbert space $\mathcal{H}$ with kernel K
$\langle f, g \rangle_\mathcal{H}$	inner product in space $\mathcal{H}$
$E^s_\alpha(c)$	Korobov class of functions, page 21
$\xi_{K,Q}$	error representer in space $\mathcal{H}(K)$ for cubature rule Q
$e(Q, K)$	worst-case error for cubature rule Q in reproducing kernel Hilbert space with kernel K
$r(h)$	defined as $\max(1, \lvert h \rvert)$, page 21
$r(\boldsymbol{h})$	defined as $\prod_{j=1}^s r(h_j)$, page 21
$\hat{f}(\boldsymbol{h})$	Fourier coefficient $\boldsymbol{h}$ of f
$r_b(h)$	polynomial equivalent of $r(h)$, page 145
$\hat{f}_b(\boldsymbol{h})$	Walsh coefficient $\boldsymbol{h}$ of f, page 145
$\varphi(n)$	Euler totient function (numbers in $\mathbb{Z}_n$ relatively prime to n)
$u \cdot v$	shorthand for multiplication modulo n when $u, v \in \mathbb{Z}_n$
$k \cdot \boldsymbol{z}$	vectorized version of $k \cdot z_j$, see just above
$\boldsymbol{h} \cdot \boldsymbol{z}$	ℓ_2 inner product, that is $\sum_j h_j z_j$
$(k_{m-1} \ldots k_0)_b$	a positive integer in radix b, see page 145
$(0.x_1 x_2 \ldots)_b$	a positive real $0 \le x < 1$ in radix b, see page 145
$a \equiv b \pmod{n}$	a and b are congruent modulo n
$a \simeq b$	a and b are equivalent (isomorph), see page 50
$\boldsymbol{A} \simeq \boldsymbol{B}$	the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are isomorph, see page 85
$G \simeq H$	the groups G and H are isomorph

Chapter 1

Introduction

For the approximation of high-dimensional integrals there are very few options to consider. One of them is the use of an equal-weight cubature rule based on a low-discrepancy point set. In this introductory chapter we show that such rules form a valuable alternative to the classical Monte Carlo method. It is therefore of interest to construct point sets with a discrepancy as low as possible.

1.1 Multivariate integration

Consider the problem of approximating an integral over the s-dimensional unit cube

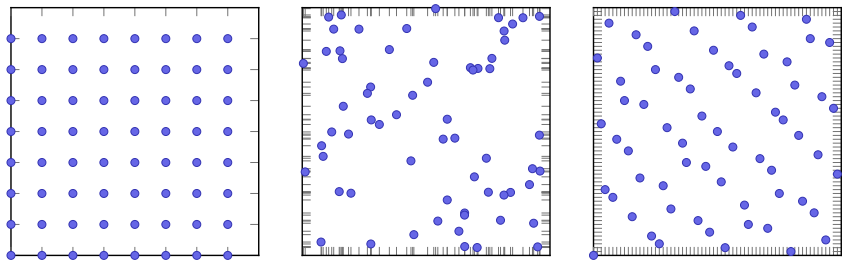
$$I(f) := \int_{[0,1)^s} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},$$

and assume that a suitable transformation from the original integration domain to the unit cube has already been performed. To approximate $I(f)$ we use a simple n-point cubature rule Q based on evaluating the function f at a set of sample points $\boldsymbol{x}_k$, with associated weights w_k. For $P_n := (\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}\}, \{w_0, \ldots, w_{n-1}\})$ set

$$Q(f; P_n) := \sum_{k=0}^{n-1} w_k\, f(\boldsymbol{x}_k). \tag{1.1}$$

When the number of dimensions is moderately high, it becomes infeasible to use iterated classical quadrature rules, so called *product rules*. Take for example an s-dimensional integral and a product of (possibly all different) one-dimensional quadrature rules, which, for simplicity, all use m points,

$$\int_0^1 \cdots \int_0^1 f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \sum_{k_1=0}^{m-1} w_{k_1}^{(1)} \cdots \sum_{k_s=0}^{m-1} w_{k_s}^{(s)}\, f(x_{k_1}^{(1)}, \ldots, x_{k_s}^{(s)}).$$



(a) product rule (b) Monte Carlo (c) quasi-Monte Carlo

Figure 1.1: Three point sets with each 64 samples in the unit square. The product left-rectangle rule has the disadvantage of having only 8 samples per dimension. The Monte Carlo rule (here using mt19937, the Mersenne Twister, with the default initial state [72]) has the disadvantage that random points tend to cluster and leave holes, but has 64 samples per dimension. Finally the quasi-Monte Carlo rule uses low-discrepancy points (here Sobol' points with direction numbers from [54]) to sample more uniformly and also has 64 samples per dimension.

Further assume that the function f has sufficient smoothness, e.g., absolutely continuous partial derivatives up to order r so that the one-dimensional quadrature rule obtains an error of $O(m^{-r})$ using m points. Then the error in terms of the total number of points $n = m^s$ is typically given by $O(n^{-r/s})$. This exponential dependence on the number of dimensions to obtain a specified level of accuracy is often called the *curse of dimensionality* or, in more recent literature, *intractability*. One apparent problem with product rules is that these points have a grid-like structure, i.e., in each one-dimensional projection m^s points get mapped onto only m points, see Figure 1.1(a), not maximizing the amount of one-dimensional information. As such the total number of points can only be specified as m^s for some integer m.

A better way to handle this problem might be to look at equal-weight cubature rules of the form

$$Q(f; P_n) := \frac{1}{n} \sum_{k=1}^{n} f(\boldsymbol{x}_k), \tag{1.2}$$

where from now on we interpret P_n as a point set, and the points $\boldsymbol{x}_k \in P_n$ are uniformly distributed over the unit cube. When these points are randomly and independently drawn, this type of integration is called *Monte Carlo integration*. On the other hand, when the points are deterministically chosen and distributed "better than random", in a way to be specified in the next section, this is called *quasi-Monte Carlo integration*. See Figure 1.1 for a comparison.

Both the Monte Carlo method and the quasi-Monte Carlo method obtain their

approximation for the integral by estimating an expected value by a sample mean. By the strong law of large numbers, the Monte Carlo method guarantees that this approximation converges in distribution. Furthermore, for $f \in L_2$, the mean-square error over all possible sets of random points is given by

$$\int_{[0,1)^{ns}} \left(\frac{1}{n} \sum_{k=0}^{n-1} f(\boldsymbol{x}_k) - I(f) \right)^2 \mathrm{d}\boldsymbol{x}_1 \cdots \mathrm{d}\boldsymbol{x}_n = \frac{\sigma^2(f)}{n}, \tag{1.3}$$

with $\sigma^2(f)$ the variance of the function. In other words, the error for integrating f is on the average $O(\sigma(f)\, n^{-1/2})$. From this it follows that the Monte Carlo method has a probabilistic error estimate of the form $O(n^{-1/2})$ (leaving out the part which depends only on the function). Comparing with the error bound for the classical product rule from above, it is surprising that this estimate does not depend on the number of dimensions. It seems to avoid the curse of dimensionality. However, here the setting is different: the error estimate is probabilistic and there are no hard guarantees that the expected accuracy is really achieved.

The quasi-Monte Carlo method looks exactly like the Monte Carlo method but uses low-discrepancy points instead of random points. Loosely speaking, since the error estimate for Monte Carlo is $O(n^{-1/2})$ *on the average*, there must exist point sets which have at most this error. (This averaging argument will return in the next chapter for existence results on good lattice rules but in a more theoretical sound setting, see §2.4.4.) For low-discrepancy points an error bound of $O(n^{-1} \log(n)^s)$ is obtained, which is asymptotically better than that of Monte Carlo. However, even for intermediate s this theoretical bound seems to indicate that using low-discrepancy point sets is intractable since the $\log(n)^s$ factor asks for impractical large n for intermediate s. In practice, quite often $O(n^{-1})$ can be observed, even for very high s.

An interesting area of ongoing research in information-based complexity is to specify for which functions quasi-Monte Carlo integration is of interest. In fact, it turns out that the problem sketched in the beginning of this discussion is in general intractable in the worst-case setting, see [5, 80]. Tractability in the worst-case setting will be discussed in Chapter 2, where it is shown for which spaces the problem of approximating multivariate integrals becomes tractable.

1.2 Low-discrepancy point sets

The discrepancy of a point set measures the deviation from the uniform distribution. Several discrepancy measures are possible. Suppose P_n is a point set with n points and J an arbitrary subset of the s-dimensional unit cube, then the *local discrepancy*

$$\mathrm{discr}(J, P_n) := \frac{|P_n \cap J|}{n} - \mathrm{vol}(J),$$

measures the mismatch between the effective number of points in J and the expected number of points in J, i.e., being proportional to the volume of J. Two important discrepancies are given by the next definitions.

Definition 1.1. The *extreme discrepancy* considers all rectangular intervals

$$D_\infty(P_n) := \sup_{\boldsymbol{x},\boldsymbol{y}\in[0,1)^s} \left|\mathrm{discr}\left([\boldsymbol{x},\boldsymbol{y}),P_n\right)\right|,$$

where $[\boldsymbol{x},\boldsymbol{y}) = [x_1,y_1)\times\cdots\times[x_s,y_s)$.

Definition 1.2. The *star discrepancy* considers all rectangular intervals anchored at the origin

$$D^\star_\infty(P_n) := \sup_{\boldsymbol{x}\in[0,1)^s} \left|\mathrm{discr}\left([\boldsymbol{0},\boldsymbol{x}),P_n\right)\right|.$$

The two definitions above both use the L_∞-norm, but variants with the L_2-norm or other L_p-norms are also common. Most of these forms are very hard to calculate. The L_2 variant of the star discrepancy is more manageable to calculate and reduces to the following double sum formula [110]

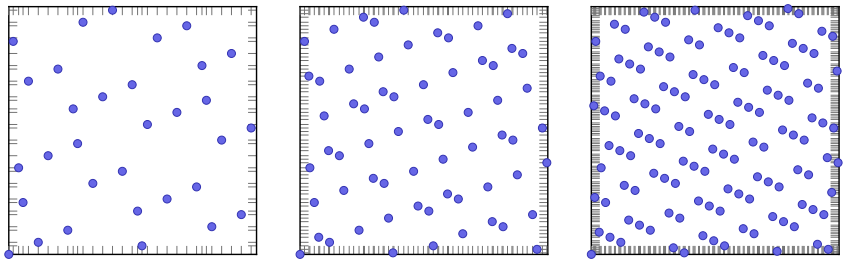
$$T_2^\star(P_n) := \left(\left(\frac{1}{3}\right)^s - \frac{2}{n}\sum_{\boldsymbol{x}\in P_n}\prod_{j=1}^{s}\frac{1-x_j^2}{2} + \frac{1}{n^2}\sum_{\boldsymbol{x},\boldsymbol{y}\in P_n}\prod_{j=1}^{s}(1-\max(x_j,y_j))\right)^{1/2}. \tag{1.4}$$

This formula is named $T_2^\star$ and not $D_2^\star$ for reasons that will become apparent in the next section where $D_2^\star$ is defined in (1.7). In the next chapter on lattice rules a discrepancy based on the L_2-norm will be used, being the worst-case error in a reproducing kernel Hilbert space which is embedded in L_2.

When the discrepancy of a point set is $O(n^{-1}(\log(n))^s)$ then it is commonly called a low-discrepancy point set. But even in Niederreiter's book [76] there is no formal definition for low-discrepancy point sets. Several point sets which achieve this discrepancy bound are known, they usually carry the name of the author who introduced them, e.g., Sobol' points [100], Faure points [31], Niederreiter points [76], Niederreiter-Xing points [111, 78] and Halton points [40].

As an example of such a low-discrepancy point set which is related to the main topic of this thesis, that of lattice rules, consider the Kronecker sequences (sometimes also called Weyl sequences). This type of sequences is obtained from taking multiples modulo 1 of a well chosen irrational vector $\boldsymbol{\alpha}\in\mathbb{R}^s$, where $\{1,\alpha_1,\ldots,\alpha_s\}$ are linearly independent over $\mathbb{Q}$, that is

$$\text{if } \sum_{j=1}^{s} w_j\alpha_j = 0 \text{ for } \boldsymbol{w}\in\mathbb{Q}^s \text{ then } \boldsymbol{w}=\boldsymbol{0},$$



(a) $n = 32$ (b) $n = 64$ (c) $n = 128$

Figure 1.2: A Kronecker sequence with the irrational vector from Niederreiter [74] in two dimensions for different number of points. This sequence could be considered to have lattice structure, which follows naturally from the similarity with how lattice points are generated (see §1.2.2), but it fails (1.11) since it is not discrete.

and the points are given by

$$\boldsymbol{x}_k = \{k\boldsymbol{\alpha}\} \qquad \text{for } k = 0, 1, \ldots . \tag{1.5}$$

The notation $\{\cdot\}$ denotes to take the fractional part componentwise. One such choice due to Niederreiter [74] is

$$\alpha_j = 2^{j/(s+1)} \qquad \text{for } j = 1, \ldots, s,$$

for a fixed maximum number of dimensions s. Using number theoretic properties and the continued fraction representations of the irrational numbers α_j it can be shown that this is a low-discrepancy sequence. For a 2-dimensional illustration see Figure 1.2.

1.2.1 The Koksma–Hlawka inequality

To obtain the classical error bound for numerical integration using quasi-Monte Carlo the notion of *bounded variation* must be introduced. The reader is referred to the book by Niederreiter [76] for further details.

Definition 1.3. The *variation* of a one-dimensional function f is defined as

$$V(f) := \sup_{\mathcal{P}} \sum_{x_i, x_{i+1} \in \mathcal{P}} |f(x_{i+1}) - f(x_i)| ,$$

where the supremum goes over all partitions of $[0, 1]$

$$\mathcal{P} := \{0 = x_0 \leq x_1 \leq \cdots \leq x_{n-1} \leq x_n = 1\} .$$

The direct generalization for multivariate functions is called the variation in the sense of Vitali. Set $\mathcal{D}_s := \{1, \ldots, s\}$ and let $V_{1,\mathcal{D}_s}(f)$ denote the variation in the sense of Vitali of the s-dimensional function f. The meaning of the 1 and the $\mathcal{D}_s$ in the subscript will become clear in the following.

Definition 1.4. The *variation in the sense of Vitali* of a function f is defined as

$$V_{1,\mathcal{D}_s}(f) := \sup_{\mathcal{P}} \sum_{J \in \mathcal{P}} |\Delta(f, J)|,$$

where $\mathcal{P}$ is the set of all partitions of $[0,1]^s$ and $\Delta(f, J)$ is the s-dimensional difference operator: an alternating sum of function values of f at the vertices of J (where adjacent vertices have opposite signs).

If the partial derivative used below exists and is continuous on $[0,1)^s$ then the variation in the sense of Vitali is also given by

$$V_{1,\mathcal{D}_s}(f) = \int_{[0,1)^s} \left| \frac{\partial^s f(\boldsymbol{x})}{\partial x_1 \cdots \partial x_s} \right| \mathrm{d}\boldsymbol{x}.$$

Since the variation in the sense of Vitali equals zero for functions which are constant in at least one variable, a more suitable variation needs to be defined.

Definition 1.5. The *variation in the sense of Hardy and Krause* of a function f is the sum over all lower dimensional projections up to s of the variation in the sense of Vitali

$$V_1(f) := \sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} V_{1,\mathfrak{u}}\left(f(\boldsymbol{x}_\mathfrak{u}, \mathbf{1})\right),$$

where $f(\boldsymbol{x}_\mathfrak{u}, \mathbf{1})$ is the function f restricted to the coordinates in $\mathfrak{u}$ and taking on the value 1 for all coordinates not in $\mathfrak{u}$.

Now the famous Koksma–Hlawka inequality can be introduced.

Theorem 1.6. *If the function f has bounded variation in the sense of Hardy and Krause then for any point set P_n*

$$|Q(f; P_n) - I(f)| \leq D^\star_\infty(P_n)\, V_1(f).$$

From the Koksma–Hlawka inequality it follows that the error for numerical integration can be written in terms of two independent factors. One involving only the integrand function, and one involving only the point set used. Most often the integrand function is unknown in advance. The point set however can be constructed ahead of time. Therefore it is crucial to be able to construct point sets with a discrepancy as low as possible. This is the main topic of this thesis.

It is important to recognize that the variation $V_1(f)$ measures the roughness of the function f with respect to quasi-Monte Carlo integration. The following chapter continues in this line of thought by assuming the function f to be in a certain function space with given smoothness. Also note that although the Koksma–Hlawka bound is normally a huge overestimate and has no use in practice, for given P_n there exist functions $f \in C^\infty([0,1)^s)$, in some sense having "infinite smoothness", with $V_1(f) = 1$, for which the bound is sharp, see [76, Theorem 2.12].

Similar to the definition of variation in the sense of Hardy and Krause, also the discrepancy can be defined as a sum over all lower dimensional projections. This leads to a stronger form of the Koksma–Hlawka error bound (see, e.g., [30]).

Theorem 1.7. *If the function f has bounded variation in the sense of Hardy and Krause then for any point set P_n*

$$|Q(f;P_n) - I(f)| \le \sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} D^\star_\infty(P_n(\mathfrak{u}))\, V_{1,\mathfrak{u}}\left(f(\boldsymbol{x}_\mathfrak{u}, \mathbf{1})\right),$$

with $P_n(\mathfrak{u})$ the point set restricted to the coordinates in $\mathfrak{u}$.

Hickernell [44] defines more general error bounds of the form

$$|Q(f;P_n) - I(f)| \le D_p(P_n)\, V_q(f), \qquad \frac{1}{p} + \frac{1}{q} = 1, \quad p \ge 1, \tag{1.6}$$

which could also include weights, i.e., a weighted discrepancy and variation, see further on in §2.4.3. This implies a more appropriate definition for L_p variants of the (star) discrepancy for values of $p \neq \infty$ by taking the norm over all projections of P_n as

$$D^\star_p(P_n) := \left(\sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \int_{[0,1)^{|\mathfrak{u}|}} \Big| \mathrm{discr}([\mathbf{0}, \boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u})) \Big|^p \, \mathrm{d}\boldsymbol{x}_\mathfrak{u} \right)^{1/p}, \qquad 1 \le p < \infty,$$

$$D^\star_\infty(P_n) := \max_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \sup_{\boldsymbol{x}_\mathfrak{u} \in [0,1)^{|\mathfrak{u}|}} |\mathrm{discr}\left([\mathbf{0}, \boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u})\right)| = \sup_{\boldsymbol{x} \in [0,1)^s} |\mathrm{discr}\left([\mathbf{0}, \boldsymbol{x}), P_n\right)|.$$

This definition then gives rise to the following L_2 variant of the star discrepancy which we will meet again in the next chapter

$$D^\star_2(P_n) := \left(\left(\frac{4}{3}\right)^s - \frac{2}{n} \sum_{\boldsymbol{x} \in P_n} \prod_{j=1}^s \frac{3 - x_j^2}{2} + \frac{1}{n^2} \sum_{\boldsymbol{x}, \boldsymbol{y} \in P_n} \prod_{j=1}^s (2 - \max(x_j, y_j)) \right)^{1/2}. \tag{1.7}$$

Intuitively this definition makes more sense than the straightforward definition of $T^\star_2$ given in (1.4), since the minimum of $T^\star_2$ decreases with increasing dimension

while that of $D_2^\star$ increases. One would indeed expect integration to become more difficult for increasing number of dimensions.

From this more general perspective the Hlawka–Zaremba identity [51, 112] for the error is a nice connection:

$$\frac{1}{n}\sum_{k=0}^{n-1} f(\boldsymbol{x}_k) - \int_{[0,1)^s} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \sum_{\emptyset\neq\mathfrak{u}\subseteq\mathcal{D}_s} (-1)^{|\mathfrak{u}|} \int_{[0,1)^{|\mathfrak{u}|}} \mathrm{discr}([\mathbf{0},\boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u}))\, \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x}_\mathfrak{u},\mathbf{1})}{\partial \boldsymbol{x}_\mathfrak{u}}\,\mathrm{d}\boldsymbol{x}_\mathfrak{u}. \quad (1.8)$$

This formula was already used by Zaremba [112] to derive (1.6) in the case $p = 2$. By Hölder's inequality one easily finds the expressions for the L_p star discrepancy and the L_q variation as

$$|Q(f;P_n) - I(f)| \le \left(\sum_{\emptyset\neq\mathfrak{u}\subseteq\mathcal{D}_s} \int_{[0,1)^{|\mathfrak{u}|}} \Big|\mathrm{discr}([\mathbf{0},\boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u}))\Big|^p \,\mathrm{d}\boldsymbol{x}_\mathfrak{u}\right)^{1/p} \times \left(\sum_{\emptyset\neq\mathfrak{u}\subseteq\mathcal{D}_s} \int_{[0,1)^{|\mathfrak{u}|}} \left|\frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x}_\mathfrak{u},\mathbf{1})}{\partial \boldsymbol{x}_\mathfrak{u}}\right|^q \mathrm{d}\boldsymbol{x}_\mathfrak{u}\right)^{1/q}, \quad \frac{1}{p}+\frac{1}{q}=1. \quad (1.9)$$

With the obvious change for the first $\sum_\mathfrak{u}\int \mathrm{d}\boldsymbol{x}_\mathfrak{u}$ to $\sup_\mathfrak{u}\sup_{\boldsymbol{x}_\mathfrak{u}}$ in the case of L_∞ discrepancy.

1.2.2 Lattice rules

A $\mathbb{Z}$-*module* is a set of quantities closed under addition and subtraction, such that for $\boldsymbol{m}\in\mathcal{M}$, $\boldsymbol{m}-\boldsymbol{m}=\mathbf{0}\in\mathcal{M}$ and integral multiples are denoted as, e.g., $3\boldsymbol{m}=\boldsymbol{m}+\boldsymbol{m}+\boldsymbol{m}$. (In fact a $\mathbb{Z}$-module is just an abelian group.) A *lattice* Λ is such a module with a finite basis $\{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_t\}$ such that any element $\boldsymbol{u}\in\Lambda$ can be written as

$$\boldsymbol{u} = \lambda_1\boldsymbol{w}_1 + \cdots + \lambda_t\boldsymbol{w}_t \qquad \text{for } \boldsymbol{\lambda}\in\mathbb{Z}^t. \quad (1.10)$$

Thus Λ is a vector space over $\mathbb{Z}$ spanned by $\boldsymbol{w}_1$, …, $\boldsymbol{w}_t$. Lattices are studied mainly in algebraic and geometric number theory, see, e.g., [12]. A module which is discrete and finite dimensional is thus a lattice, where discrete means that there exists a norm such that

$$\|\boldsymbol{u}\| \ge k \qquad \text{when } \boldsymbol{u}\neq\mathbf{0}, \quad (1.11)$$

for k a fixed, positive constant. In what follows $\mathcal{M}=\mathbb{R}^s$.

The number of elements in a lattice Λ is infinite. It is called an *integration lattice* if $\mathbb{Z}^s \subseteq \Lambda$, from which it follows that the lattice points are periodic with period 1. Now for Λ an integration lattice, the node set of a *lattice rule* is given by $\Lambda \cap [0,1)^s$.

Looking at (1.5), the Kronecker sequence example (shown in Figure 1.2) looks a lot like the node set of a lattice rule. However, it fails (1.11). Therefore, instead of taking an irrational vector $\boldsymbol{\alpha} \in \mathbb{R}^s$ for the Kronecker sequence, a rational vector $\boldsymbol{w} \in \mathbb{Q}^s$ could be chosen instead, and as such it will be discrete. Setting n as the common denominator of the components of $\boldsymbol{w}$ and defining $\boldsymbol{z} \in \mathbb{Z}^s$ by $\boldsymbol{w} = \boldsymbol{z}/n$, the point set

$$\boldsymbol{x}_k = \left\{ \frac{k\boldsymbol{z}}{n} \right\} \qquad \text{for } k = 0, 1, \ldots, n-1, \tag{1.12}$$

is obtained. The notation $\{\cdot\}$ means the fractional part is taken componentwise. The point set (1.12) is the node set of a rank-1 lattice rule. Rank-1 lattice rules are also called *number theoretic rules* (although all lattice rules are of number theoretic nature).

The point set of a rank-1 lattice rule can be interpreted as a finite cyclic additive subgroup of $\mathbb{R}^s/\mathbb{Z}^s$ generated by $\boldsymbol{z}/n + \mathbb{Z}^s$. A lattice rule can be generalized to being any equal-weight cubature formula where the point set P_n forms a finite additive abelian group and thus $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in P_n$

(i) $\boldsymbol{x} + \boldsymbol{y} \in P_n$ (closure);
(ii) $(\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{z} = \boldsymbol{x} + (\boldsymbol{y} + \boldsymbol{z})$ (associative);
(iii) $\exists \mathbf{0} \in P_n : \boldsymbol{x} + \mathbf{0} = \boldsymbol{x}$ (neutral element);
(iv) $\exists (-\boldsymbol{x}) \in P_n : \boldsymbol{x} + (-\boldsymbol{x}) = \mathbf{0}$ (inverse);
(v) $\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}$ (commutative).

Obviously since P_n is a subgroup of the factor group $\mathbb{R}^s/\mathbb{Z}^s$ the points lie in the half open unit cube $[0,1)^s$ and the origin $\mathbf{0}$ is always an element of the point set. This point of view is used in, e.g., [90], [70], [92] and also in the book by Sloan and Joe [89], and leads to the following definition.

Definition 1.8. A *lattice rule* is a rule Q of the form

$$Q(f) = \frac{1}{n} \sum_{\boldsymbol{x}_k \in P_n} f(\boldsymbol{x}_k),$$

where the n points from P_n form a finite additive abelian subgroup of $\mathbb{R}^s/\mathbb{Z}^s$ under the operation of componentwise addition modulo 1.

Continuing the small abuse of notation in using P_n for the group and using the points in $[0,1)^s$ as the elements, it would suffice to introduce fractional braces

around the operations in the list above to obtain the real operations on the points. In accordance with (1.10) and since P_n is a finite group of size n, it must be finitely generated by a set of generators G such that $P_n = \langle G \rangle$. It follows that every point $\boldsymbol{x} \in P_n$ can be written as a linear combination of the generators $G = \{\boldsymbol{z}_i/n_i\}_i$ as $\boldsymbol{x} = \sum_{i=1}^{t} \lambda_i \boldsymbol{z}_i/n_i \bmod 1$ with $t = |G|$ and $\boldsymbol{\lambda} \in \mathbb{Z}^s$.

These more general lattice rules defined in Definition 1.8 are thus of the form

$$Q(f) = \frac{1}{m} \sum_{k_1=0}^{n_1-1} \cdots \sum_{k_t=0}^{n_t-1} f\left(\left\{ \frac{k_1 \boldsymbol{z}_1}{n_1} + \cdots + \frac{k_t \boldsymbol{z}_t}{n_t} \right\} \right), \tag{1.13}$$

with a set of generators $\boldsymbol{z}_i/n_i \in \mathbb{Q}^s$ and $m = n_1 \cdots n_t$, $t \leq s$, but not necessarily having m distinct points (m actually being any multiple of $|P_n| = n$). In fact, the product left-rectangle rule, Figure 1.1(a), is in this way also a lattice rule (albeit not a very good one since it suffers the curse of dimensionality by construction) and it can be written as

$$Q(f) = \frac{1}{m^s} \sum_{k_1=0}^{m-1} \cdots \sum_{k_s=0}^{m-1} f\left(\left\{ \frac{k_1 \boldsymbol{u}_1}{m} + \cdots + \frac{k_t \boldsymbol{u}_s}{m} \right\} \right) \tag{1.14}$$

$$= \frac{1}{m^s} \sum_{k_1=0}^{m-1} \cdots \sum_{k_s=0}^{m-1} f\left(\frac{(k_1, \ldots, k_s)^\top}{m} \right), \tag{1.15}$$

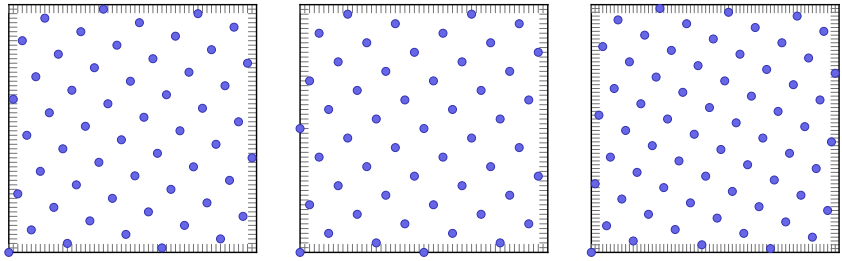
where $\boldsymbol{u}_j$ is the unit vector with a one at place j.

The multiple sum notation in (1.13) is not uniquely defined. The same lattice rule may be expressed in many different ways since different sets of generators G could span P_n, differing both in the generators and the number of generators. This was studied extensively in [92]. The minimum size of such a generating set is defined as the rank of the lattice rule.

Definition 1.9. The *rank* r, $1 \leq r \leq s$, of an s-dimensional lattice P_n is the smallest number of cyclic subgroups into which P_n may be decomposed.

The rank is thus the minimum number of sums in (1.13). The product left-rectangle rule (1.14) has maximal rank s, and m^s different points, while the classical lattice rule with point set (1.12) has rank 1, and n different points if $\gcd(z_1, \ldots, z_s, n) = 1$. A lattice rule can be written in its *canonical form* [92] which has then exactly the minimum of r sums. This will be discussed further in Chapter 7.

Pictures of the node sets of some lattice rules are given in Figure 1.3. In this text we are mainly concerned with the easiest breed of rank-1 rules. The generator $\boldsymbol{w} \in \mathbb{Q}^s$ of a rank-1 lattice uniquely defines a $\boldsymbol{z} \in \mathbb{Z}^s$ by $\boldsymbol{w} = \boldsymbol{z}/n$, n the common denominator, and it is common to speak of the *generating vector* $\boldsymbol{z}$ of the lattice rule. Since the following chapters mainly deal with rank-1 lattice rules, a specialized version of Definition 1.8 is given.



(a) Fibonacci lattice (b) rank-2 copy rule (c) rank-1 rule

Figure 1.3: Node sets of different lattice rules in two dimensions. The first one is a Fibonacci lattice rule with generating vector $\boldsymbol{z} = (1, 34)$ and $n = 55$ points ($F_9 = 34$ and $F_{10} = 55$ are consecutive Fibonacci numbers); the rank-2 lattice in the middle is a 2^s-copy rule of a Fibonacci rule with $F_7 = 13$ points for a total of $n = 13 \times 2^2 = 52$; and the next rank-1 rule has $\boldsymbol{z}_1 = (1, 8)$, $n_1 = 13$ and $\boldsymbol{z}_2 = (3, 1)$, $n_2 = 5$, a direct sum of two Fibonacci rules with $n = 13 \times 5 = 65$ points which could be written with one generating vector $\boldsymbol{z} = (1, 47)$. Fibonacci rules have very good properties in two dimensions and will be discussed in Chapter 2, §2.5.1; copy rules will be discussed in Chapter 7.

Definition 1.10. A *rank-1 lattice rule* is a rule Q of the form

$$Q(f) = \frac{1}{n} \sum_{k=0}^{n-1} f\left(\left\{\frac{k\boldsymbol{z}}{n}\right\}\right),$$

with $\boldsymbol{z} \in \mathbb{Z}^s$ the *generating vector*. The lattice rule has n different points if $\gcd(z_1, \ldots, z_s, n) = 1$ and each one-dimensional projection also has n different points if $\gcd(z_j, n) = 1$ for all $j = 1, \ldots, s$.

The dual of a lattice is used in the classical theory to obtain the error of a lattice rule Q. The definition is given for completeness but will only be used at a minor occasion in Chapter 2.

Definition 1.11. The *dual lattice* $\Lambda^\perp$ of a lattice Λ is given by

$$\Lambda^\perp := \{\boldsymbol{h} \in \mathbb{Z}^s : \boldsymbol{h} \cdot \boldsymbol{x} \in \mathbb{Z} \text{ for all } \boldsymbol{x} \in \Lambda\}.$$

The next chapter will deal entirely with the theoretical background of "good" lattice rules in certain function spaces. It will turn out that only in two dimensions an explicit construction for "good" lattice rules exists and so to obtain "good" lattice rules in higher dimensions a computer search is necessary. The discussion on one such search algorithm, the component-by-component construction algorithm by Sloan and his collaborators (e.g., see [93]), will take up the rest of the chapters.

1.2.3 Digital nets and sequences

Digital nets and sequences are another type of low-discrepancy points; the book by Niederreiter [76] contains a nice overview. These point sets are constructed to distribute their points optimally among boxes of volume $b^{-\kappa}$ which are anchored at, and span, multiples of negative powers of the integer base $b \geq 2$ in each dimension.

Definition 1.12. For a given integer basis $b \geq 2$ define the *elementary b-ary interval*

$$J(\boldsymbol{k}, \boldsymbol{\ell}) := \prod_{j=1}^{s} \left[\ell_j \, b^{-k_j}, (\ell_j + 1) \, b^{-k_j}\right),$$

for all $k_j \geq 0$ and $0 \leq \ell_j < b^{k_j}$. Such an elementary interval $J(\boldsymbol{k}, \boldsymbol{\ell})$ has volume

$$\mathrm{vol}(J(\boldsymbol{k}, \boldsymbol{\ell})) = b^{-\kappa} \qquad \text{with } \kappa = \sum_{j=1}^{s} k_j.$$

A digital net in base b can now be defined as having its points distributed in an optimal way with respect to such intervals.

Definition 1.13. A (t, m, s)*-net in base b* is an s-dimensional point set having b^m points and which has in each elementary interval of volume b^{t-m} exactly the expected number of points $b^{t-m} b^m = b^t$. The parameter $0 \leq t \leq m$ is sometimes called (counterintuitively) the *quality parameter* of the net, where a smaller value of t is better.

A (t, m, s)-net has a fixed number of points, however one can also define (t, s)-sequences in which the total number of points is undetermined. A similar effect will be obtained for the lattice sequences in Chapter 5.

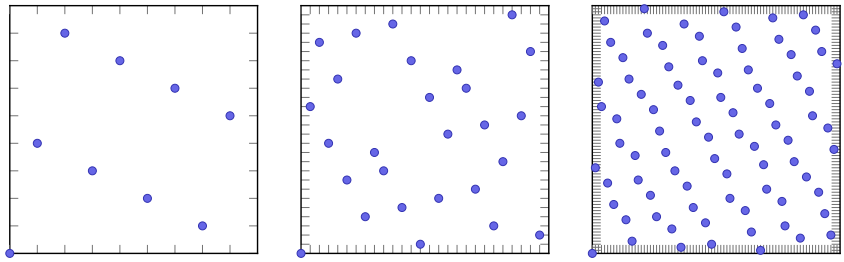
Definition 1.14. A (t, s)*-sequence in base b* is an infinite point set for which all subsets of size b^m, $m > t$, consisting of the points $\boldsymbol{x}_{k\,b^m}, \ldots, \boldsymbol{x}_{(k+1)\,b^m - 1}$ for $k \geq 0$, form a (t, m, s)-net in base b.

The following two theorems from Niederreiter [76, Theorem 4.10] and [76, Theorem 4.17] confirm that the t value must be small; obviously for $t = m$ any point set with b^m points has $b^t = b^m$ points in the elementary interval of size $b^{t-m} = 1$.

Theorem 1.15. *The star discrepancy of a (t, m, s)-net P_n in base b, with $n = b^m$, satisfies*

$$D^\star_\infty(P_n) \lesssim B(s, b) \; b^{t-m} (\log(n))^{s-1},$$

with $B(s, b)$ a constant only depending on b and s.



(a) $n = 9$ (b) $n = 27$ (c) $n = 81$

Figure 1.4: The Faure sequence is always a $(0, s)$-sequence where the base b is such that $b \geq s$ and prime. The figure shows the two-dimensional Faure sequence in base 3. Since these are the initial parts of the sequence with 9, 27 and 81 points (all powers of 3), they are respectively $(0, 2, 2)$-, $(0, 3, 2)$- and $(0, 4, 2)$-nets

Theorem 1.16. *The star discrepancy of the first $n \geq 2$ points of a (t, s)-sequence P_n in base b satisfies*

$$D^\star_\infty(P_n) \lesssim C(s, b)\, b^t n^{-1} (\log(n))^s.$$

A one-dimensional (t, s)-sequence which will be of use in later chapters is the radical inverse function. This is a $(0, 1)$-sequence.

Definition 1.17. The *radical inverse function in base b* maps integers into b-ary fractions by mirroring around the b-ary radix point:

$$\phi_b(k) = \phi_b((k_{m-1} \dots k_0)_b) := (0.k_0 \dots k_{m-1})_b = \sum_{\ell=0}^{m-1} k_\ell\, b^{-(\ell+1)}.$$

The Sobol' sequence in Figure 1.1(c) on page 2 is a (t, s)-sequence in base 2 where the t-parameter grows modestly with increasing number of dimensions (see [100, Theorem 3.4]). For two dimensions its t-value is the best possible, i.e., $t = 0$, and so this is a $(0, 2)$-sequence in base 2. Since these are the first 64 points, this is also a $(0, 6, 2)$-net in base 2. In Figure 1.4 some examples of the two-dimensional Faure sequence in base 3 are shown.

1.2.4 Error estimation by randomization

Although the Koksma–Hlawka bound, Theorem 1.6, is a strict and deterministic error bound, in practice it is mostly useless. Even in Chapter 2 were a Koksma–Hlawka bound is derived for a specific smoothness space the bound is useless in practice since it is rarely known in which space a given function is.

A more practical approach to error estimation involves randomly shifting the point set by independent random shifts as introduced by Cranley and Patterson [19]. A recent overview of randomized quasi-Monte Carlo can be found in [67]. For lattice rules the shift is just added to the points modulo 1. For digital nets and sequences the shift is added "digitally" to keep the good structure of the low-discrepancy point sets.

Definition 1.18. The point set $P_n = \{\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{n-1}\}$ of a lattice rule can be shifted by $\boldsymbol{\Delta} \in [0,1)^s$ to obtain a *shifted lattice rule* with points $P_n + \boldsymbol{\Delta} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}\}$ which are defined as

$$\boldsymbol{x}_k := \boldsymbol{y}_k + \boldsymbol{\Delta} \mod 1 \qquad \text{for } k = 0, 1, \ldots, n-1.$$

Definition 1.19. The point set P_n from a digital net (or sequence) in base b can be digitally shifted by $\boldsymbol{\Delta} \in [0,1)^s$ to obtain a *digitally shifted net (or sequence)* $P_n + \boldsymbol{\Delta} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}\}$ where the points are defined as

$$\boldsymbol{x}_k := \boldsymbol{y}_k \oplus_b \boldsymbol{\Delta} \qquad \text{for } k = 0, 1, \ldots, n-1,$$

and $\oplus_b$ is b-ary addition modulo b.

In [67] several other advanced randomization techniques for digital nets and sequences are discussed. However, the digital shift is the easiest one and is the most relevant to the present work. It will be useful later on in Chapter 6.

Now taking multiple shifts $\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_q$ distributed independently and uniformly in $[0,1)^s$ we obtain q different estimates of the integral. The following is the standard way to obtain a stochastic error estimate, see, e.g., [89].

Theorem 1.20. *For a number of random independent uniform shifts $\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_q$, define the n-point approximations*

$$Q_n^{(\ell)}(f) := Q_n(f; P_n + \boldsymbol{\Delta}_\ell) = \frac{1}{n} \sum_{k=0}^{n-1} f(\boldsymbol{x}_k) \qquad \text{with } \boldsymbol{x}_k \in P_n + \boldsymbol{\Delta}_\ell,$$

for $\ell = 1, \ldots, q$. Then the qn-point approximation

$$\overline{Q}_{q \times n}(f; P_n) := \frac{1}{q} \sum_{\ell=1}^{q} Q_n^{(\ell)}(f)$$

is an unbiased estimate of $I(f)$. Using the standard error $\sqrt{\sigma^2/q}$ of this approximation (with σ^2 the variance of the approximations $Q_n^{(\ell)}$), a $(1 - 1/\nu^2)$-confidence interval, $\nu > 1$, can be obtained as

$$\overline{Q}_{q \times n}(f; P_n) \pm \nu \sqrt{\sigma^2/q}.$$

Net structure:	$\leftrightarrow$	**Lattice structure:**
Points can be divided into boxes.		Having a lattice structure.
Sequence usage:	$\leftrightarrow$	**Non-sequence usage:**
Used point by point or in blocks, i.e., an 'open' method.		Must be used all at once, i.e., a 'closed' method.
Infinite size:	$\leftrightarrow$	**Finite size:**
The point set has infinite size.		The point set has finite size.
Dimension extensible:	$\leftrightarrow$	**Fixed dimension:**
Dimension is not fixed in advance.		Dimension is fixed in advance.

Table 1.1: Some properties of low-discrepancy point sets.

An unbiased estimate for the standard error is given by

$$\hat{s} = \left(\frac{1}{q(q-1)} \sum_{\ell=1}^{q} (Q_n^{(\ell)}(f) - Q(f))^2 \right)^{1/2}.$$

It is not really necessary to obtain a good estimate of the standard error. For the error bound one is only interested in the magnitude of the error.

Applying q random shifts to a point set with n points of course results in a cubature rule with a total of qn points. The confidence interval in the previous theorem follows from the Chebyshev inequality and makes absolutely no assumption on the distribution of the $Q_n^{(\ell)}$. One could optimistically assume a normal distribution and independence and then use Student's t-statistics to obtain a tighter confidence interval. E.g., for a 95%-confidence interval $\nu \approx 4.472$ while using t-statistics with $q = 10$ randomizations (i.e., 9 degrees of freedom) a multiplication factor of only approximately 2.262 has to be used.

In Mathematical Reviews Niederreiter remarks on [19] (MR0494820, 1979) that:

> *"The whole idea of randomizing the method of good lattice points is self-defeating since the method was introduced with a view to eliminating the need for random sampling in the Monte Carlo method by replacing it by a deterministic scheme."*

This is indeed a crucial point and so it only makes sense if q is taken as a small number; often q is set to 10.

1.2.5 Classification of point sets

In Table 1.1 a basic set of properties useful to classify low-discrepancy point sets is presented (but not all low-discrepancy point sets fall into these categories). From this table the classical view on a lattice rule could be identified as having

lattice structure, non-sequence usage, finite size and fixed dimension. However, the results in this thesis will show that any lattice rule can be used as a sequence and the technology presented is capable of constructing lattice rules with such large numbers of points that for most practical purposes the rule can be considered to be of infinite size (since infinity mostly coincides with 2^{31} for most calculations). Moreover, it will be shown that an embedded lattice sequence can be constructed where each intermediate rule is a good lattice rule (comparable with the structure of digital sequences in Definition 1.14). Also due to the nature of the component-by-component construction algorithm, these lattice rules are dimension extensible.

Obviously (t, m, s)-nets and (t, s)-sequences are classified as having net structure and respectively finite and infinite size. In the same way the Kronecker sequence has more or less a lattice structure, sequence usage and infinite size. Some of them are dimension extensible, e.g., the Sobol' sequence (Figure 1.1(c)), some of them are not, e.g., the Faure sequence (Figure 1.4). In Chapter 6 the fast construction of polynomial lattice rules will be discussed. These have net structure and finite size.

In general the discrepancy for a finite size point set is a little better than that of an infinite size point set, being $O(n(\log(n))^{s-1})$ versus $O(n^{-1}(\log(n))^s)$. The same effect can be noticed for fixed dimension versus dimension extensible.

1.3 Overview

In Chapter 2 the basic theory of lattice rules in reproducing kernel Hilbert spaces is presented. The worst-case error is introduced to assess the quality of a lattice rule as well as methods to search for good lattice rules. The component-by-component algorithm is introduced and a preview on the fast algorithm is given.

Chapter 3 and Chapter 4 then present the results for fast component-by-component construction for respectively a prime number of points and for a composite number of points. In Chapter 5 it is explained that any lattice rule could be used as a sequence, for which an algorithm is given, but also how to construct lattice rules which are particularly well suited to be used as a sequence. These rules are actually sequences of embedded lattice rules, where each lattice rule in the chain is a good lattice rule.

Chapter 6 generalizes the theory from Chapter 2 and Chapter 3 to polynomial lattice rules, obtaining fast construction for polynomial lattice rules, and Chapter 7 gives an extension to construct copy rules with the fast algorithm.

Chapter 8 presents some real world uses of these lattice rules. This includes randomized replication to obtain an estimate of the approximation error. Two applications from the financial mathematics world are given as well as a statistics example.

Some paths for future work and curious open problems are identified in Chapter 9. Also an overview of the main contributions in this thesis is presented there.

Chapter notes

This chapter only touches upon the large field of quasi-Monte Carlo methods and only the very basics were introduced. Historical overviews on quasi-Monte Carlo methods and lattice rules can be found in Niederreiter's book [76] and in the book by Sloan and Joe [89].

One problem is that the definitions of variation and discrepancy are in terms of boxes parallel to the coordinate axes and so the orientation more or less influences the view on what works and what does not. In [76] an isotropic discrepancy is defined to circumvent this problem. Furthermore, the anchor $\mathbf{0}$ in Definition 1.2 of the star discrepancy is an arbitrary choice; in [43, 44] a wrap-around discrepancy and an unanchored discrepancy are defined to remove the need to choose an anchor (similar to the extreme discrepancy, but defined in a reproducing kernel Hilbert space, see also the unanchored Sobolev space in Chapter 2).

Whether one uses digital shifts for nets and sequences or just ordinary Cranley–Patterson shifts as for lattice rules is a matter of taste and style. The argument that the shifts will destroy the net properties (i.e., the t-value will increase) can be countered by pointing at the rather arbitrary choice of anchors for the elementary intervals. In fact, the t-value would remain the same if we would allow the elementary intervals to wrap around the unit cube and since we are averaging over all (digital) shifts it is implausible that this would have a negative effect on the convergence. However, since digital nets and sequences in base b normally operate on b-ary representations it is often convenient to add a b-ary digital shift.

Chapter 2

Lattice rules and function spaces

Our aim is to search for good lattice rules which minimize the worst-case error in a reproducing kernel Hilbert space. Three function spaces will be studied: the Korobov space, the anchored Sobolev space and the unanchored Sobolev space. It will be explained how weights will make quasi-Monte Carlo integration tractable and how the weights can be used to accommodate the function space to the actual functions of interest.

The discussion on the actual construction of lattice rules will only consider n-point rank-1 lattice rules where all components of the generating vector are relatively prime to n. Such rank-1 lattice rules, or *number theoretic rules*, were introduced by number theorists like Hlawka and Korobov around 1960. Under certain conditions they are called *the method of good lattice points* by Hlawka (whereby $\boldsymbol{z}/n$ is a good lattice point modulo n) or *optimal coefficients* by Korobov.

2.1 Good lattice rules

Which lattice rules should be called *good*? Clearly when the goal is to use the point set for numerical integration, then the integration error is a sound choice to discriminate good from bad lattice rules. In two papers by Korobov in 1959 and 1960, [58] and [59], the components of the generating vector are called *optimal coefficients* if the error is of order $O(n^{-\alpha+\varepsilon})$, $\varepsilon > 0$, or equivalently $O(n^{-\alpha}(\log(n))^{\beta})$ for a specific class of functions with $\alpha > 1$, and β and the implied constant independent of n. In [58] existence of optimal coefficients is proven with $\beta = \alpha s$.

Assume that the integrand function f has an absolutely convergent Fourier

series,

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\mathbb{Z}^s} \hat{f}(\boldsymbol{h})\, \exp(2\pi \mathrm{i}\, \boldsymbol{h}\cdot\boldsymbol{x}),$$

and thus

$$\sum_{\boldsymbol{h}\in\mathbb{Z}^s} |\hat{f}(\boldsymbol{h})| < \infty, \tag{2.1}$$

with $\boldsymbol{h}\cdot\boldsymbol{x}$ the ordinary ℓ_2-inner product and the Fourier coefficients given by

$$\hat{f}(\boldsymbol{h}) := \int_{[0,1)^s} f(\boldsymbol{x})\, \exp(-2\pi \mathrm{i}\, \boldsymbol{h}\cdot\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}. \tag{2.2}$$

Note that this assumption limits the functions to be periodic over $[0,1)^s$. For functions which have such an absolutely convergent Fourier series there is a classical result from Sloan and Kachoyan [90] that gives an expression for the error of a lattice rule in terms of the Fourier coefficients of f. The theorem here is given limited to the rank-1 case.

Theorem 2.1. *For an s-dimensional n-point rank-1 lattice rule Q with generating vector $\boldsymbol{z}$ and a function f with absolutely convergent Fourier series the integration error can be written as*

$$Q(f) - I(f) = \sum_{\substack{\boldsymbol{h}\in\mathbb{Z}^s\setminus\{\boldsymbol{0}\} \\ \boldsymbol{h}\cdot\boldsymbol{z}\equiv 0 \pmod n}} \hat{f}(\boldsymbol{h}).$$

Proof. Apply the lattice rule to f:

$$Q(f) = \sum_{\boldsymbol{h}\in\mathbb{Z}^s} \hat{f}(\boldsymbol{h})\, \frac{1}{n} \sum_{k=0}^{n-1} \exp(2\pi \mathrm{i}\, (\boldsymbol{h}\cdot\boldsymbol{z})\, k/n).$$

Here the last sum equals 0 if $\boldsymbol{h}\cdot\boldsymbol{z} \not\equiv 0 \pmod n$ and n otherwise, thus

$$Q(f) = \hat{f}(\boldsymbol{0}) + \sum_{\substack{\boldsymbol{h}\in\mathbb{Z}^s\setminus\{\boldsymbol{0}\} \\ \boldsymbol{h}\cdot\boldsymbol{z}\equiv 0 \pmod n}} \hat{f}(\boldsymbol{h}).$$

The result now follows since $\hat{f}(\boldsymbol{0}) = I(f)$. □

The theorem in [90] gives the result for general rank rules as defined in Definition 1.8. There the exponential sum over the lattice points can be interpreted as

a character sum over the finite abelian group from the lattice rule and a similar expression is obtained:

$$Q(f) - I(f) = \sum_{\substack{\boldsymbol{h}\in\mathbb{Z}^s\setminus\{\mathbf{0}\} \\ \boldsymbol{x}\in P_n+\mathbb{Z}^s \\ \boldsymbol{h}\cdot\boldsymbol{x}\in\mathbb{Z}}} \hat{f}(\boldsymbol{h}).$$

The set of vectors $\boldsymbol{h}$ which fulfill these sum conditions is the dual lattice (see Definition 1.11) and plays an important role in the classical theory of lattice rules. Here the *worst-case error* will soon be introduced and the dual lattice will not be needed in subsequent chapters.

Following from the Riemann-Lebesgue lemma, for a $[0,1)^s$-periodic function f to be L_1-integrable over $[0,1)^s$ its Fourier coefficients $\hat{f}(\boldsymbol{h})$ should go to zero for $\boldsymbol{h}$ moving away from the origin. Naturally such a decay then defines a smoothness class of functions. The following class was introduced by Korobov [58].

Definition 2.2. Let $E^s_\alpha(c)$, $\alpha > 1$ and $c > 0$, be the class of $[0,1)^s$-periodic functions f for which

$$|\hat{f}(\boldsymbol{h})| \le c\, r(\boldsymbol{h})^{-\alpha},$$

with

$$r(\boldsymbol{h}) := \prod_{j=1}^{s} r(h_j), \qquad\qquad r(h) := \max(1, |h|). \tag{2.3}$$

From (2.1) it is easy to see that α must be larger than 1 since the Riemann zeta function for integer α is defined as

$$\zeta(\alpha) := \sum_{h=1}^{\infty} \frac{1}{h^\alpha},$$

and $\zeta(\alpha) < \infty$ for $\alpha > 1$.

Definition 2.3. For an n-point rank-1 lattice rule with generating vector $\boldsymbol{z} \in \mathbb{Z}^s$, $\alpha > 1$, define

$$P_\alpha(\boldsymbol{z}, n) := \sum_{\substack{\boldsymbol{h}\in\mathbb{Z}^s\setminus\{\mathbf{0}\} \\ \boldsymbol{h}\cdot\boldsymbol{z}\equiv 0 \pmod n}} r(\boldsymbol{h})^{-\alpha}.$$

Again, for higher rank lattice rules, the sum should be interpreted as going over the dual lattice. Now the following theorem follows easily.

Theorem 2.4. *If Q is an n-point lattice rule with generating vector $\boldsymbol{z} \in \mathbb{Z}^s$, $\alpha > 1$ and $c > 0$, then*

$$\max_{f \in E_\alpha^s(c)} |Q(f) - I(f)| = c\, P_\alpha(\boldsymbol{z}, n).$$

Proof. From Definitions 2.2 and 2.3 follows that for all $f \in E_\alpha^s(c)$

$$|Q(f) - I(f)| \le c\, P_\alpha(\boldsymbol{z}, n),$$

and equality is obtained for the function

$$\xi_\alpha(\boldsymbol{x}) = c \sum_{\boldsymbol{h} \in \mathbb{Z}^s} r(\boldsymbol{h})^{-\alpha} \exp(2\pi \mathrm{i}\, \boldsymbol{h} \cdot \boldsymbol{x}).$$

□

The importance of this theorem is illustrated by the next corollary which gives an easy method to calculate P_α (at least for even α).

Corollary 2.5. *The quantity $P_\alpha(\boldsymbol{z}, n)$ can be calculated as the integration error of the function*

$$f_\alpha(\boldsymbol{x}) := \sum_{\boldsymbol{h} \in \mathbb{Z}^s} r(\boldsymbol{h})^{-\alpha} \exp(2\pi \mathrm{i}\, \boldsymbol{h} \cdot \boldsymbol{x}).$$

Now since

$$f_\alpha(\boldsymbol{x}) = \prod_{j=1}^{s} \left(1 + \sum_{h \in \mathbb{Z}^s \setminus \{0\}} \frac{\exp(2\pi \mathrm{i}\, h x_j)}{|h|^\alpha} \right),$$

see (2.14) on page 34, for $\alpha > 1$ an even integer the infinite sum can be expressed in terms of the Bernoulli polynomial of degree α [1, p. 805]:

$$\sum_{h \in \mathbb{Z}^s \setminus \{0\}} \frac{\exp(2\pi \mathrm{i}\, h x)}{|h|^\alpha} = \frac{(2\pi)^\alpha}{(-1)^{\alpha/2-1} \alpha!} B_\alpha(x) \qquad \text{for } 0 \ge x \ge 1. \tag{2.4}$$

The first few even degree Bernoulli polynomials are given by

$$\begin{aligned} B_2(x) &= x^2 - x + \tfrac{1}{6}, \\ B_4(x) &= x^4 - 2x^3 + x^2 - \tfrac{1}{30}, \\ B_6(x) &= x^6 - 3x^5 + \tfrac{5}{2}x^4 - \tfrac{1}{2}x^2 + \tfrac{1}{42}. \end{aligned}$$

The Bernoulli polynomials have the nice properties that

$$B_\alpha(x) = (-1)^\alpha\, B_\alpha(1 - x), \tag{2.5}$$

$$\int_0^1 B_\alpha(x)\, \mathrm{d}x = 0 \qquad \text{for } \alpha > 0.$$

2.2 Reproducing kernel Hilbert spaces

Theorem 2.4 gives rise to more general error bounds in Banach spaces and reproducing kernel Hilbert spaces. This is studied thoroughly in [43, 44]. In this section reproducing kernel Hilbert spaces are defined following [73] and then the Riesz representation theorem will be used to define a representer for function evaluation. In the following the domain D could be substituted for $[0,1)^s$ in the context of integration over $[0,1)^s$ and the elements $x, y \in D$ should then be interpreted as vectors.

The Riesz theorem is stated as follows:

Theorem 2.6. *If L is a bounded linear functional on a Hilbert space $\mathcal{H}$, then there is a unique element $h_L \in \mathcal{H}$, called the* (Riesz) representer *of L, such that for all $f \in \mathcal{H}$*

$$L(f) = \langle f, h_L \rangle_{\mathcal{H}}.$$

A functional L is called *bounded* if there exists a real number $M < \infty$ such that $|L(f)| \leq M \|f\|_{\mathcal{H}}$ for all $f \in \mathcal{H}$. In a Hilbert space the notion of a bounded functional is equivalent to a continuous functional. Evaluating a function at a given point y can also be written as the application of a *point evaluation functional*, denoted by T_y for evaluation at y. In terms of these point evaluation functionals we can define reproducing kernel Hilbert spaces.

Definition 2.7. A Hilbert space $\mathcal{H}$ is called a *reproducing kernel Hilbert space* if and only if all the point evaluation functionals on $\mathcal{H}$, denoted by T_y, for all $y \in D$, are continuous (or equivalently: if they are all bounded).

It now follows by the Riesz representation theorem that there exists a unique representer for function evaluation in the reproducing kernel Hilbert space $\mathcal{H}$. This representer is called the reproducing kernel.

Definition 2.8. A function $K : D \times D \to \mathbb{R}$ is a *reproducing kernel* of the Hilbert space $\mathcal{H}$ if and only if for all $y \in D$ and all $f \in \mathcal{H}$

$$K(\cdot, y) = K_y \in \mathcal{H} \qquad \text{and} \qquad T_y(f) = f(y) = \langle f, K(\cdot, y) \rangle_{\mathcal{H}} = \langle f, K_y \rangle_{\mathcal{H}}.$$

A shorthand notation for a space $\mathcal{H}$ with reproducing kernel K is $\mathcal{H}(K)$. Such a reproducing kernel has the following properties

(i) $\forall x, y \in D : K(x, y) = \overline{K(y, x)}$ (conjugate symmetric);
(ii) $\forall n \geq 1, \forall a_i \in \mathbb{C}, \forall x_i \in D : \sum_{i,j=1}^{n} a_i \overline{a_j} K(x_i, x_j) \geq 0$ (positive definite).

We consider spaces of real-valued functions and thus here $K(x, y) = K(y, x)$.

From Theorem 2.6 follows that the reproducing kernel is unique if it exists, but the theorem does not give a method to find it. The following theorem does just that for a separable space. (A metric space is *separable* if it contains a countable dense subset $\{\varphi_k\}_k$.)

Theorem 2.9. *If $\mathcal{H}(K)$ is a separable reproducing kernel Hilbert space then the reproducing kernel is given by*

$$K(x,y) = \sum_k \varphi_k(x)\,\overline{\varphi_k(y)},$$

where $\{\varphi_k\}_k$ is any orthonormal basis for $\mathcal{H}$ (w.r.t. the inner product in $\mathcal{H}$).

In the development of the fast component-by-component algorithm a specific property of the kernel is important. From here on set again $D = [0,1)^s$.

Definition 2.10. A reproducing kernel is *shift-invariant* if for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Delta} \in [0,1)^s$

$$K(\{\boldsymbol{x}+\boldsymbol{\Delta}\}, \{\boldsymbol{y}+\boldsymbol{\Delta}\}) = K(\boldsymbol{x},\boldsymbol{y}).$$

Lemma 2.11. *If a reproducing kernel is shift-invariant then it can be written in terms of one variable*

$$K(\boldsymbol{x},\boldsymbol{y}) = K(\{\boldsymbol{x}-\boldsymbol{y}\}, \boldsymbol{0}) =: K(\{\boldsymbol{x}-\boldsymbol{y}\}).$$

Proof. Take the shift $\boldsymbol{\Delta} = \{-\boldsymbol{y}\}$ such that $\{\boldsymbol{y}+\boldsymbol{\Delta}\} = \boldsymbol{0}$. □

Lemma 2.12. *A shift-invariant kernel has the following symmetry property:*

$$K(\boldsymbol{x}) = K(\{1-\boldsymbol{x}\}).$$

Proof. It suffices to consider only the one-dimensional case. Since a real-valued reproducing kernel is symmetric, $K(x,y) = K(y,x)$, it follows that $K(\{x-y\}) = K(\{y-x\})$. Now set $t = \{x-y\}$, then $\{1-t\}$ equals

$$1 - t \equiv 1 - (x-y) \equiv y - x \pmod 1,$$

which is equivalent to $\{y-x\}$ and thus $K(\{1-t\}) = K(t)$. □

As an example of a reproducing kernel Hilbert space consider the space of s-dimensional trigonometric polynomials $\mathcal{T}_m$, where for $f \in \mathcal{T}_m$,

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{h}\in\Lambda_m} \hat{f}(\boldsymbol{h})\,\exp(2\pi \mathrm{i}\,\boldsymbol{x}\cdot\boldsymbol{h}), \tag{2.6}$$

and the product trigonometric degree is bounded by m,

$$\deg_{\mathcal{T}}(f) := \max_j |h_j| \le m.$$

(Note that since this space has a finite and bounded basis it is clear that it is a reproducing kernel Hilbert space. We will introduce spaces with infinite bases shortly.) It is clear that the index set Λ_m is given by

$$\Lambda_m := \left\{ \boldsymbol{h} \in \mathbb{Z}^s : \max_j |h_j| \le m \right\},$$

and the obvious orthonormal system of basis functions is

$$\{\varphi_k(\boldsymbol{x})\}_k = \{\exp(2\pi\mathrm{i}\ \boldsymbol{h}\cdot\boldsymbol{x})\}_{\boldsymbol{h}\in\Lambda_m},$$

such that

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{y}) &= \sum_{\boldsymbol{h}\in\Lambda_m} \varphi_{\boldsymbol{h}}(\boldsymbol{x})\,\overline{\varphi_{\boldsymbol{h}}(\boldsymbol{y})} \\ &= \sum_{h_1=-m}^{m} \cdots \sum_{h_s=-m}^{m} \exp(2\pi\mathrm{i}\ \boldsymbol{h}\cdot(\boldsymbol{x}-\boldsymbol{y})) \\ &= \prod_{j=1}^{s} \sum_{h=-m}^{m} \exp(2\pi\mathrm{i}\ h(x_j-y_j)) \\ &= \prod_{j=1}^{s} \eta(x_j-y_j), \qquad \text{with } \eta(t) = \sum_{h=-m}^{m} \exp(2\pi\mathrm{i}\ ht), \end{aligned} \tag{2.7}$$

is the reproducing kernel for $\mathcal{T}_m$. The function $\eta(t)$ equals $2m+1$ for $t \equiv 0 \pmod 1$. For $t \not\equiv 0 \pmod 1$ a nice form can be obtained by using the geometric series formula

$$\sum_{h=0}^{m} r^h = \frac{1-r^{m+1}}{1-r}$$

for the exponential

$$\exp(\mathrm{i}\ x) = \cos(x) + \mathrm{i}\ \sin(x)$$

and then taking the real part over the sum to be summing cosines. So

$$\begin{aligned} \sum_{h=-m}^{m} \exp(2\pi\mathrm{i}\ ht) &= -1 + 2\sum_{h=0}^{m} \cos(2\pi ht) \\ &= -1 + 2\,\mathrm{Re}\left\{\sum_{h=0}^{m} \exp(2\pi\mathrm{i}\ ht)\right\} \\ &= -1 + 2\,\mathrm{Re}\left\{\frac{1-\exp(2\pi\mathrm{i}\ t(m+1))}{1-\exp(2\pi\mathrm{i}\ t)}\right\}, \\ &= -1 + 2\,\mathrm{Re}\left\{\frac{\exp(2\pi\mathrm{i}\ t(m+1)/2)}{\exp(2\pi\mathrm{i}\ t/2)}\right. \\ &\qquad \left. \times\ \frac{\exp(-2\pi\mathrm{i}\ t(m+1)/2) - \exp(2\pi\mathrm{i}\ t(m+1)/2)}{\exp(-2\pi\mathrm{i}\ t/2) - \exp(2\pi\mathrm{i}\ t/2)}\right\}. \end{aligned}$$

The second fraction now simply equals $\sin(\pi t(m+1))/\sin(\pi t)$ and the first can be reduced to a single exponential, thus

$$\begin{aligned}\sum_{h=-m}^{m} \exp(2\pi\mathrm{i}\, ht) &= -1 + \frac{2\sin(\pi t(m+1))}{\sin(\pi t)} \operatorname{Re}\{\exp(\pi\mathrm{i}\, tm)\} \\ &= -1 + \frac{2\sin(\pi t(m+1))\cos(\pi tm)}{\sin(\pi t)} \\ &= \frac{\sin(\pi t(2m+1))}{\sin(\pi t)}.\end{aligned}$$

Substituting this back into (2.7) we obtain a nice form for the reproducing kernel of $\mathcal{T}_m$

$$K(\{\boldsymbol{x}-\boldsymbol{y}\}) = K(t) = \begin{cases} 2m+1 & \text{for } t \equiv 0 \pmod 1, \\ \displaystyle\prod_{j=1}^{s} \frac{\sin(\pi t(2m+1))}{\sin(\pi t)} & \text{otherwise.} \end{cases}$$

In (2.6) the index set (or spectrum) of the trigonometric functions could be taken arbitrary, see, e.g., [70, 13, 18]. However, not all choices lead to practical kernels to work with, and this is of course a general truth for all reproducing kernels. Clearly, the example here leads to a shift-invariant kernel for which the s-dimensional kernel is a simple product of the one-dimensional kernels $\eta(t)$. This always arises if the function space is a tensor-product space, see [4]. It suffices to keep the kernel from the previous example in mind as a prototype kernel in the development of the fast component-by-component algorithm since it is the most simple form suitable for the algorithm; being shift-invariant, tensor-product and having a nice calculable form. However, the product trigonometric degree, having nice properties for the algorithm that will be developed in the next chapter, has "little to recommend it" [70] from the practical point of view of constructing lattice rules for it. Luckily there are other function spaces which have the same nice properties for the fast construction algorithm. E.g., the product weighted Korobov space, to be introduced in §2.4, and the standard Korobov class E_α^s from Definition 2.2 as well as a whole collection of other spaces fall into this category.

Although the necessity of the tensor-product form of the kernel can be loosened, the shift-invariant property of the kernel is an absolute demand for the current fast component-by-component algorithm. Therefore it is interesting to be able to take any reproducing kernel Hilbert space $\mathcal{H}(K)$ and transform it into a related shift-invariant space $\mathcal{H}^{\text{shinv}}$. This is always possible by averaging the kernel over all possible shifts.

Lemma 2.13. *Given a reproducing kernel Hilbert space $\mathcal{H}(K)$ there is an associ-*

ated reproducing kernel K^{shinv}*, defined by*

$$K^{\text{shinv}}(\boldsymbol{x},\boldsymbol{y}) := \int_{[0,1)^s} K(\{\boldsymbol{x}+\boldsymbol{\Delta}\},\{\boldsymbol{y}+\boldsymbol{\Delta}\})\,\mathrm{d}\boldsymbol{\Delta} =: K^{\text{shinv}}(\{\boldsymbol{x}-\boldsymbol{y}\}),$$

that is shift-invariant and is the reproducing kernel of a shift-invariant reproducing kernel Hilbert space $\mathcal{H}^{\text{shinv}}$*.*

The relationship between the space $\mathcal{H}$ and $\mathcal{H}^{\text{shinv}}$ will be explained in the next section; see Theorem 2.22 which outlines the relation between the worst-case error in $\mathcal{H}^{\text{shinv}}$ and the worst-case error of a randomly shifted point set in the space $\mathcal{H}$. Note that the associated kernel has the symmetry property from Lemma 2.12.

2.3 Integration in the worst-case setting

We now use some concepts from the information-based complexity framework [105], where the idea is to prove complexity lower bounds independently of any algorithm. In particular we need to define the worst-case error.

Definition 2.14. The *worst-case error*, e, of an algorithm U, using discrete and noisy information about f, to approximate the solution S for problems $f \in \mathcal{F}$ is defined as

$$e(U,\mathcal{F}) := \sup_{f\in\mathcal{F}} \|S(f)-U(f)\|.$$

Specifically for numerical integration as discussed in this text, the following definition for the worst-case error is obtained.

Definition 2.15. The *worst-case error* for a cubature rule Q to approximate integration I of functions in the unit ball of a Banach space $\mathcal{F}$ is defined as

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

To define *tractability* of multivariate integration we need to look at the asymptotical behavior of the worst-case error, both in the number of dimensions and the number of information functionals used [97, 98]. For quasi-Monte Carlo integration we use only function values (and this is called *standard information*).

Denote by $e_{s,n}$ the worst-case error for an s-dimensional cubature rule using n function evaluations. Let $n_{\min}$ be the minimal number of points needed to reduce the initial error by a factor ε, $0<\varepsilon<1$, i.e.,

$$e_{s,n} \le \varepsilon\, e_{s,0},$$

where $e_{s,0}$ is the initial error (for $n = 0$, $Q(f)$ is an empty sum and evaluates to 0, see also (2.13) which states the initial error as $\|I\|$). Now if $n_{\min}$ can be bounded by a polynomial in ε and s then it makes sense to try and approximate integrals of functions in the space under consideration.

Definition 2.16. Integration in the space $\mathcal{F}$ is called *tractable* if there exist positive constants C, p and q such that

$$n_{\min}(\varepsilon, s) \leq C\, \varepsilon^{-p} s^q,$$

and *strongly tractable* if $q = 0$.

In §2.4 we will see that, by considering weighted function spaces, it is possible to achieve strong tractability. For the classical unweighted function spaces it turns out that there is an exponential dependence on the number of dimensions, and thus these spaces are intractable.

The remaining part of this section borrows heavily from Hickernell [44]. In [43, 44], Hickernell makes the link between the classical discrepancies as defined in Chapter 1 and the classical lattice criterion P_α defined in Definition 2.3. This is done in terms of worst-case error analysis (as well as average-case error analysis) for reproducing kernel Hilbert spaces and more general Banach spaces. Usually, obtaining explicit expressions for the worst-case error is nearly impossible, but in the case of reproducing kernel Hilbert spaces a very nice form exists.

Since in a reproducing kernel Hilbert space function evaluation is bounded then also integration and cubature (over a finite domain) are bounded, that is

$$\|I\|_{\mathcal{H}} = \sup_{\substack{f \in \mathcal{H} \\ \|f\|_{\mathcal{H}}=1}} I(f) \leq M_1 \qquad \text{and} \qquad \|Q\|_{\mathcal{H}} = \sup_{\substack{f \in \mathcal{H} \\ \|f\|_{\mathcal{H}}=1}} Q(f) \leq M_2.$$

Thus according to Theorem 2.6 there must be Riesz representers for both of them. If they are denoted by h_I and h_Q then the error for numerical integration can be expressed as

$$I(f) - Q(f) = (I - Q)(f) = \langle f, h_I - h_Q \rangle_{\mathcal{H}} = \langle f, \xi_{K,Q} \rangle_{\mathcal{H}} \tag{2.8}$$

for all $f \in \mathcal{H}$ where $\xi_{K,Q} \in \mathcal{H}$ is the Riesz representer for the error or the *error representer*. The dependence on the kernel K (or the space $\mathcal{H}$) and on the cubature formula Q is written out explicitly to denote its importance. If one of them changes, then also the error representer changes. This is important to note, since it turns out that the norm of the error representer can be interpreted as the discrepancy of the point set *in relation to the chosen function space.*

Theorem 2.17. *If $\mathcal{H}(K)$ is a reproducing kernel Hilbert space and $\xi_{K,Q}$ is the error representer for a cubature rule Q, then the error for functions $f \in \mathcal{H}(K)$ can be bounded by*

$$|I(f) - Q(f)| \leq \|f\|_{\mathcal{H}}\, \|\xi_{K,Q}\|_{\mathcal{H}}.$$

Proof. Since $I(f) - Q(f) = \langle f, \xi_{K,Q} \rangle_{\mathcal{H}}$ the result follows by applying the Cauchy–Schwarz inequality to $|\langle f, \xi_{K,Q} \rangle_{\mathcal{H}}|$. □

This is equivalent to Theorem 1.6 and can be interpreted as a Koksma–Hlawka type inequality for the space $\mathcal{H}(K)$ and, *specifically*, the cubature rule Q. In accordance with Theorem 1.6 the variation of f is given by $\|f\|_{\mathcal{H}}$ and the discrepancy is given by $\|\xi_{K,Q}\|_{\mathcal{H}}$. Again, this bound is sharp and is attained for functions that are scalar multiples of $\xi_{K,Q} \in \mathcal{H}$. Actually, since quasi-Monte Carlo rules are equal-weight cubature rules where the weights sum to one, constant functions are integrated exactly and the variation of the function could be more naturally defined as the norm of the nonconstant part, such that

$$\begin{aligned} V(f;\mathcal{H}) &:= \|f_\perp\|_{\mathcal{H}}, \qquad \text{with } f_\perp := f - \frac{1\langle f, 1\rangle_{\mathcal{H}}}{\langle 1, 1\rangle_{\mathcal{H}}}, \\ D(P_n;\mathcal{H}) &:= \|\xi_{K,Q}\|_{\mathcal{H}}. \end{aligned}$$

Comparing Definition 2.15 and Theorem 2.17 we notice that for equal-weight cubature rules the worst-case error is just another kind of discrepancy, since we have

$$e(Q, K) = \|\xi_{K,Q}\|_{\mathcal{H}}.$$

Theorem 2.18. *The squared worst-case error for integration in $\mathcal{H}(K)$ by using an equal-weight cubature rule Q of the form* (1.2) *is given by*

$$e^2(Q, K) = \int_{[0,1)^{2s}} K(\boldsymbol{x}, \boldsymbol{y}) \,\mathrm{d}\boldsymbol{x} \,\mathrm{d}\boldsymbol{y} - \frac{2}{n} \sum_{k=0}^{n-1} \int_{[0,1)^s} K(\boldsymbol{x}_k, \boldsymbol{y}) \,\mathrm{d}\boldsymbol{y} + \frac{1}{n^2} \sum_{k,\ell=0}^{n-1} K(\boldsymbol{x}_k, \boldsymbol{x}_\ell),$$

where $\boldsymbol{x}_k, \boldsymbol{x}_\ell \in P_n$ for $k, \ell = 0, \ldots, n-1$ are the points of the point set P_n of Q.

Proof. First use the reproducing property of the kernel to obtain the representers of integration h_I and cubature h_Q in terms of the reproducing kernel K

$$h_I(\boldsymbol{y}) = \langle h_I, K(\cdot, \boldsymbol{y}) \rangle_{\mathcal{H}} = I(K_{\boldsymbol{y}}) \quad \text{and} \quad h_Q(\boldsymbol{y}) = \langle h_Q, K(\cdot, \boldsymbol{y}) \rangle_{\mathcal{H}} = Q(K_{\boldsymbol{y}}),$$

and thus the error representer (introduced in (2.8)) can be expressed as

$$\xi_{K,Q} = h_I - h_Q = I(K) - Q(K).$$

Now since the worst-case error is equal to $\|\xi_{K,Q}\|_{\mathcal{H}} = (\langle \xi_{K,Q}, \xi_{K,Q} \rangle_{\mathcal{H}})^{1/2}$ it follows

that

$$\begin{aligned} e^2(Q,K) &= \langle \xi_{K,Q}, \xi_{K,Q} \rangle_{\mathcal{H}} \\ &= \langle I(K) - Q(K), I(K) - Q(K) \rangle_{\mathcal{H}} \\ &= \langle I(K), I(K) - Q(K) \rangle_{\mathcal{H}} - \langle Q(K), I(K) - Q(K) \rangle_{\mathcal{H}} \\ &= \langle I(K), I(K) \rangle_{\mathcal{H}} - 2\langle I(K), Q(K) \rangle_{\mathcal{H}} + \langle Q(K), Q(K) \rangle_{\mathcal{H}} \\ &= \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y}) \,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} - \frac{2}{n}\sum_{k=0}^{n-1} \int_{[0,1)^s} K(\boldsymbol{x}_k,\boldsymbol{y}) \,\mathrm{d}\boldsymbol{y} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K(\boldsymbol{x}_k,\boldsymbol{x}_\ell). \end{aligned}$$

□

Corollary 2.19. *If the kernel K is shift-invariant then the squared worst-case error is given by*

$$e^2(Q,K) = -\int_{[0,1)^s} K(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K(\{\boldsymbol{x}_k - \boldsymbol{x}_\ell\}). \tag{2.9}$$

Proof. Using that

$$\int_{[0,1)^s} f(\{\boldsymbol{x}+\boldsymbol{\Delta}\}) \,\mathrm{d}\boldsymbol{x} = \int_{[0,1)^s} f(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}, \qquad \forall \boldsymbol{\Delta} \in \mathbb{R}^s, \tag{2.10}$$

and from Theorem 2.18 respectively taking shifts $\boldsymbol{\Delta} = \{-\boldsymbol{y}\}$, $\boldsymbol{\Delta}_k = \{-\boldsymbol{x}_k\}$ and $\boldsymbol{\Delta}_\ell = \{-\boldsymbol{x}_\ell\}$ it follows that

$$e^2(Q,K) = \int_{[0,1)^s} K(\boldsymbol{x},\boldsymbol{0}) \,\mathrm{d}\boldsymbol{x} - 2\int_{[0,1)^s} K(\boldsymbol{x},\boldsymbol{0}) \,\mathrm{d}\boldsymbol{x} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K(\{\boldsymbol{x}_k - \boldsymbol{x}_\ell\},\boldsymbol{0})$$

from which the result follows. □

Corollary 2.20. *If the kernel K is shift-invariant and Q is a lattice rule then the squared worst-case error is given by*

$$e^2(Q,K) = -\int_{[0,1)^s} K(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} + \frac{1}{n}\sum_{k=0}^{n-1} K(\boldsymbol{x}_k).$$

Proof. The points of a lattice rule form a finite additive abelian group (Definition 1.8), i.e., $\{\boldsymbol{x}_k - \boldsymbol{x}_\ell\} \in P_n$ for all $\boldsymbol{x}_k, \boldsymbol{x}_\ell \in P_n$, and the double sum in (2.9) can be written as a single sum. □

Finally we arrive at the most interesting form of the worst-case error in light of the fast component-by-component algorithm.

Corollary 2.21. *If the kernel K is shift-invariant and also of tensor-product form and Q is a lattice rule then*

$$e^2(Q,K) = -\prod_{j=1}^{s}\int_0^1 \eta_j(x)\,\mathrm{d}x + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\eta_j(x_j^{(k)}), \tag{2.11}$$

with η_j the kernels of the one-dimensional reproducing kernel Hilbert spaces, such that $K(\boldsymbol{x}) = \prod_{j=1}^{s}\eta_j(x_j)$.

Now we specify the connection for the worst-case error for a generic kernel and its associated shift-invariant kernel from Lemma 2.13. Since we are discussing quasi-Monte Carlo cubature rules, i.e., rules of the form (1.2), it suffices to refer to the point set P_n instead of the cubature rule Q.

Theorem 2.22. *The mean squared worst-case error in $\mathcal{H}(K)$ using a randomly shifted point set $P_n + \boldsymbol{\Delta}$, with shift $\boldsymbol{\Delta}$, is the squared worst-case error in the associated space $\mathcal{H}^{\mathrm{shinv}}$ using the point set P_n, that is*

$$E_{\boldsymbol{\Delta}}[e^2(P_n+\boldsymbol{\Delta},K)] = e^2(P_n,K^{\mathrm{shinv}}).$$

(With $E_{\boldsymbol{\Delta}}[\cdot]$ the expected value over $\boldsymbol{\Delta}$.) We call $e^2(P_n,K^{\mathrm{shinv}})$ the randomly shifted worst-case error *for the reproducing kernel Hilbert space $\mathcal{H}(K)$.*

Proof. From Theorem 2.18 we immediately obtain

$$\begin{aligned}\int_{[0,1)^s} e^2(P_n+\boldsymbol{\Delta},K)\,\mathrm{d}\boldsymbol{\Delta} &= \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}\\ &\quad - \frac{2}{n}\sum_{k=0}^{n-1}\int_{[0,1)^{2s}} K(\{\boldsymbol{x}_k+\boldsymbol{\Delta}\},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}\,\mathrm{d}\boldsymbol{\Delta}\\ &\quad + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1}\int_{[0,1)^s} K(\{\boldsymbol{x}_k+\boldsymbol{\Delta}\},\{\boldsymbol{x}_\ell+\boldsymbol{\Delta}\})\,\mathrm{d}\boldsymbol{\Delta}.\end{aligned}$$

Now using (2.10) and Lemma 2.13 this becomes

$$\begin{aligned}E_{\boldsymbol{\Delta}}[e^2(P_n+\boldsymbol{\Delta},K)] &= -\int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K^{\mathrm{shinv}}(\boldsymbol{x}_k,\boldsymbol{x}_\ell)\\ &= -\int_{[0,1)^s} K^{\mathrm{shinv}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K^{\mathrm{shinv}}(\{\boldsymbol{x}_k-\boldsymbol{x}_\ell\}),\end{aligned}$$

and the result follows (see Corollary 2.19). □

All these formulas contain the term

$$\kappa_s := \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} = \int_{[0,1)^s} K^{\mathrm{shinv}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}. \tag{2.12}$$

Since the initial approximation is zero (evaluating an empty sum), $\kappa_s^{1/2}$ can be seen as the *initial error* for cubature in the space $\mathcal{H}$, which can be written as the operator norm of integration

$$\|I\| = \|h_I\|_{\mathcal{H}} = \left(\int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}\right)^{1/2} = \kappa_s^{1/2}. \tag{2.13}$$

(Equivalently, in the average-case setting this is the expected value of integrating a random function from the space $\mathcal{H}$ with covariance kernel K.)

For function spaces which are not shift-invariant it makes sense to study the randomly shifted worst-case error, especially when one wants to use the error estimation based on randomization from §1.2.4. To obtain such a stochastic error estimator one shifts the low-discrepancy point set q times by independent uniform random shifts. Of course, the number of random shifts needs to be taken small to still benefit from the low-discrepancy points. This is summarized in the next theorem [91, Theorem 3.2].

Theorem 2.23. *For a randomly shifted rule $\overline{Q}_{q\times n}$,*

$$\overline{Q}_{q\times n}(f;P_n) := \frac{1}{q}\sum_{k'=1}^{q} Q(f;P_n+\boldsymbol{\Delta}_{k'}) = \frac{1}{q}\sum_{k'=1}^{q}\frac{1}{n}\sum_{k=0}^{n-1} f\left(\{\boldsymbol{x}_k+\boldsymbol{\Delta}_{k'}\}\right),$$

using n points $\boldsymbol{x}_k \in P_n$ and q independent uniform random shifts $\boldsymbol{\Delta}_{k'}$, one has

$$E[e^2(q\times P_n,K)] = \frac{1}{q}\,e^2(P_n,K^{\mathrm{shinv}}),$$

where $q\times P_n$ is used to denote the q uniform random shifts of P_n.

Proof. From Theorem 2.18 we immediately obtain

$$\begin{aligned}
&\int_{[0,1)^{qs}} e^2(q \times P_n, K)\,\mathrm{d}\boldsymbol{\Delta}_1 \cdots \mathrm{d}\boldsymbol{\Delta}_q \\
&\quad = \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} \\
&\qquad - \frac{2}{n}\frac{1}{q}\sum_{k'=1}^{q}\sum_{k=0}^{n-1}\int_{[0,1)^{2s}} K(\{\boldsymbol{x}_k+\boldsymbol{\Delta}_{k'}\},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}\,\mathrm{d}\boldsymbol{\Delta}_{k'} \\
&\qquad + \frac{1}{n^2}\frac{1}{q^2}\sum_{\substack{k',\ell'=1\\ k'\neq\ell'}}^{q}\sum_{k,\ell=0}^{n-1}\int_{[0,1)^{2s}} K(\{\boldsymbol{x}_k+\boldsymbol{\Delta}_{k'}\},\{\boldsymbol{x}_\ell+\boldsymbol{\Delta}_{\ell'}\})\,\mathrm{d}\boldsymbol{\Delta}_{k'}\,\mathrm{d}\boldsymbol{\Delta}_{\ell'} \\
&\qquad + \frac{1}{n^2}\frac{1}{q^2}\sum_{k'=1}^{q}\sum_{k,\ell=0}^{n-1}\int_{[0,1)^{s}} K(\{\boldsymbol{x}_k+\boldsymbol{\Delta}_{k'}\},\{\boldsymbol{x}_\ell+\boldsymbol{\Delta}_{k'}\})\,\mathrm{d}\boldsymbol{\Delta}_{k'} \\
&\quad = \kappa_s - 2\kappa_s + \frac{q-1}{q}\kappa_s + \frac{1}{q}\frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K^{\mathrm{shinv}}(\boldsymbol{x}_k,\boldsymbol{x}_\ell) \\
&\quad = \frac{1}{q}\left(-\kappa_s + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K^{\mathrm{shinv}}(\boldsymbol{x}_k,\boldsymbol{x}_\ell)\right),
\end{aligned}$$

from which the result follows (see Corollary 2.19) and where we used (2.10) and Lemma 2.13, and κ_s as defined in (2.12). □

Let us assume for a moment that typically Monte Carlo gives a convergence of $O(N^{-1/2})$ for N sample points and quasi-Monte Carlo gives $O(N^{-1})$. Then, for a total number of $N = qn$ points, the limiting cases would be to use $q = 1$ and $n = N$, having no error estimator at all with convergence $O(N^{-1})$; or to use $q = N$ and $n = 1$ and being back to the Monte Carlo case which achieves only $O(N^{-1/2})$. Loosely speaking, the case in between then is expected to show $O(q^{-1/2}n^{-1})$.

2.4 Function spaces, tractability and existence

Having reviewed the theory of reproducing kernel Hilbert spaces and the expressions for the worst-case error, we are now able to define some useful function spaces for multivariate integration. We will introduce weighted function spaces for which multivariate integration becomes (strongly) tractable in the sense of Definition 2.16. For these spaces it is known that cubature rules exist which achieve the optimal rate of convergence.

First we mention the following (elementary but) very useful facts which will come into play in the next sections. For arbitrary sets $\mathcal{A}$ and $\mathcal{D}$, and k_j, c_j,

$f_j(h_j) \in \mathbb{R}$ we have

$$\prod_{j\in\mathcal{D}}(1+k_j) = \sum_{\mathfrak{u}\subseteq\mathcal{D}}\prod_{j\in\mathfrak{u}} k_j, \qquad \prod_{j\in\mathcal{D}}(c_j+k_j) = \sum_{\mathfrak{u}\subseteq\mathcal{D}}\prod_{\substack{j\notin\mathfrak{u}\\ j\in\mathcal{D}}} c_j \prod_{j\in\mathfrak{u}} k_j,$$

$$\sum_{\boldsymbol{h}\in\mathcal{A}^s}\prod_{j=1}^{s} f_j(h_j) = \prod_{j=1}^{s}\sum_{h_j\in\mathcal{A}} f_j(h_j). \tag{2.14}$$

2.4.1 Unweighted function spaces

The two classical function spaces for multivariate integration are the Korobov space and the Sobolev space. The classical Korobov space parallels the space E^s_α introduced in Definition 2.2.

Definition 2.24. The classical *Korobov space* is a reproducing kernel Hilbert space which consists of one-periodic functions with absolutely convergent Fourier series and has reproducing kernel

$$K(\boldsymbol{x},\boldsymbol{y}) = \prod_{j=1}^{s}\left(1+\sum_{\substack{h\in\mathbb{Z}\\ h\neq 0}} r(h)^{-\alpha}\,\exp(2\pi\mathrm{i}\,h(x_j-y_j))\right),$$

inner product, in terms of the Fourier coefficients (2.2),

$$\langle f,g\rangle = \sum_{\boldsymbol{h}\in\mathbb{Z}^s} r(\boldsymbol{h})^{\alpha}\,\hat{f}(\boldsymbol{h})\,\overline{\hat{g}(\boldsymbol{h})},$$

and norm

$$\|f\| = \left(\sum_{\boldsymbol{h}\in\mathbb{Z}^s} r(\boldsymbol{h})^{\alpha}\,|\hat{f}(\boldsymbol{h})|^2\right)^{1/2},$$

where $r(\boldsymbol{h})$ is as defined in (2.3) and the smoothness parameter $\alpha > 1$.

This function space contains all functions f for which $\|f\| < \infty$ and is obviously shift-invariant and of tensor-product form. Trivial manipulations deliver the following equivalent forms for the reproducing kernel:

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{y}) &= \prod_{j=1}^{s}\left(1+\sum_{\substack{h\in\mathbb{Z}\\ h\neq 0}} r(h)^{-\alpha}\,\exp(2\pi\mathrm{i}\,h(x_j-y_j))\right)\\ &= \prod_{j=1}^{s}\sum_{h\in\mathbb{Z}} r(h)^{-\alpha}\,\exp(2\pi\mathrm{i}\,h(x_j-y_j)) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\boldsymbol{h}\in\mathbb{Z}^s} \prod_{j=1}^{s} r(h_j)^{-\alpha} \exp(2\pi \mathrm{i}\, h_j(x_j - y_j)) \\
&= \sum_{\boldsymbol{h}\in\mathbb{Z}^s} r(\boldsymbol{h})^{-\alpha} \exp(2\pi \mathrm{i}\, \boldsymbol{h}\cdot(\boldsymbol{x}-\boldsymbol{y})) \\
&= 1 + \prod_{j=1}^{s} \sum_{\substack{h\in\mathbb{Z}\\ h\neq 0}} \frac{\exp(2\pi \mathrm{i}\, h(x_j - y_j))}{|h|^{\alpha}}. \qquad (2.15)
\end{aligned}
$$

It is obvious that a set of orthonormal basis functions is given by the scaled trigonometric monomials

$$\left\{ r(\boldsymbol{h})^{-\alpha/2} \exp(2\pi \mathrm{i}\, \boldsymbol{h}\cdot\boldsymbol{x}) \right\}_{\boldsymbol{h}\in\mathbb{Z}^s}.$$

As was already mentioned, when α is an even integer the infinite sum (2.15) can be written in terms of the Bernoulli polynomial of degree α, see (2.4). For P_n the node set of a lattice rule, the worst-case error for $\alpha = 2$ is then given by using Corollary 2.21 where

$$
\begin{aligned}
K(\boldsymbol{x}) &= \prod_{j=1}^{s} \left(1 + 2\pi^2 B_2(x_j)\right), \\
\kappa_s &= \left(1 + 2\pi^2 \int_0^1 B_2(x)\,\mathrm{d}x\right)^s = 1,
\end{aligned}
$$

and thus the worst-case error for $\alpha = 2$ is given by

$$e(P_n, K) = \left(-1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(1 + 2\pi^2 B_2(x_j^{(k)})\right) \right)^{1/2}.$$

The following theorem from Sloan and Woźniakowski [96] shows that integration in such an (unweighted) Korobov space is intractable since the number of points needed to reduce the error can be as high as 2^s which is exponential in the dimension.

Theorem 2.25. *If the number of points in a rule of the form* (1.1) *is* $n < 2^s$ *then*

$$e_{s,n} = 1 = e_{s,0}$$

for multivariate integration in an s*-dimensional Korobov space with smoothness parameter* $\alpha > 1$.

The proof consists of constructing a "fooling function" $g \in E_\alpha^s$ for which $I(g) = 1$ and $Q(g) = 0$. To find the function g a non-trivial solution of a system with

n linear equations and 2^s unknowns has to be found; this solution always exists when $n < 2^s$.

More frequently one finds Sobolev spaces in the numerical mathematics literature. The term Sobolev space refers to the use of derivatives in the inner product (and norm) of the space to characterize a certain smoothness. Therefore, depending on the application area, one Sobolev space could differ completely from another Sobolev space. Some classical examples can be found in [7]. Here we consider tensor-product spaces in arbitrary dimensions where the smoothness parameter of the one-dimensional spaces is limited to one. In [99] three different kinds of (weighted) Sobolev spaces are studied for multivariate integration differing in how the function itself enters the inner product. Here we just give one. In §2.4.3 another one will be given.

Definition 2.26. The classical *anchored Sobolev space* is a reproducing kernel Hilbert space which consists of functions with square-integrable mixed first order derivatives. The reproducing kernel is given by

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{j=1}^{s} \eta_j(x_j, y_j),$$

where

$$\begin{aligned} \eta_j(x,y) &:= \begin{cases} 1 + \min(|x-a_j|, |y-a_j|) & \text{if } (x-a_j)(y-a_j) > 0, \\ 1 & \text{otherwise,} \end{cases} \\ &= 1 + \tfrac{1}{2}(|x-a_j| + |y-a_j| - |x-y|). \end{aligned}$$

The inner product and norm are given by

$$\begin{aligned} \langle f, g \rangle &= \sum_{\mathrm{u} \subseteq \mathcal{D}_s} \int_{[0,1)^{|\mathrm{u}|}} \frac{\partial^{|\mathrm{u}|} f(\boldsymbol{x}_\mathrm{u}, \boldsymbol{a})}{\partial \boldsymbol{x}_\mathrm{u}} \frac{\partial^{|\mathrm{u}|} g(\boldsymbol{x}_\mathrm{u}, \boldsymbol{a})}{\partial \boldsymbol{x}_\mathrm{u}} \, \mathrm{d}\boldsymbol{x}_\mathrm{u}, \\ \|f\| &= \left(\sum_{\mathrm{u} \subseteq \mathcal{D}_s} \int_{[0,1)^{|\mathrm{u}|}} \left| \frac{\partial^{|\mathrm{u}|} f(\boldsymbol{x}_\mathrm{u}, \boldsymbol{a})}{\partial \boldsymbol{x}_\mathrm{u}} \right|^2 \mathrm{d}\boldsymbol{x}_\mathrm{u} \right)^{1/2}, \end{aligned}$$

where $\boldsymbol{a} \in [0,1]^s$ is called the *anchor* (usually all a_j are 0, 1/2 or 1) and the integral for the empty set is just $\int_{[0,1)^{|\emptyset|}} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}_\emptyset = f(\boldsymbol{x})$.

The norm here can be recognized as the L_2 version of the Hardy and Krause variation (with arbitrary anchor) given in Definition 1.5 for functions from this space. For cubature rules which integrate constant functions exactly, like quasi-Monte Carlo rules, the empty set could be dropped from the sum (to obtain a seminorm which can be interpreted as the variation). Likewise, the worst-case

error for this function space is the L_2 version of the star discrepancy $D_2^\star$. Setting $\boldsymbol{a} = \mathbf{1}$ and working out Theorem 2.18 one obtains formula (1.7), since then

$$K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{j=1}^{s} \left(2 - \max(x_j, y_j)\right), \tag{2.16}$$

$$\int_{[0,1)^s} K(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{d}\boldsymbol{y} = \prod_{j=1}^{s} \left(2 - \int_0^1 \max(x_j, y) \, \mathrm{d}y\right) = \prod_{j=1}^{s} \frac{3 - x_j^2}{2},$$

$$\int_{[0,1)^{2s}} K(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{y} = \prod_{j=1}^{s} \int_0^1 \frac{3 - x^2}{2} \, \mathrm{d}x = \left(\frac{4}{3}\right)^s,$$

and thus for this space the worst-case error is given by

$$e(P_n, K) = \left(\left(\frac{4}{3}\right)^s - \frac{2}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \frac{3 - (x_j^{(k)})^2}{2} + \frac{1}{n^2} \sum_{k,\ell=0}^{n-1} \prod_{j=1}^{s} \left(2 - \max(x_j^{(k)}, x_j^{(\ell)})\right) \right)^{1/2}.$$

Also for such (unweighted) Sobolev spaces multivariate integration turns out to be intractable (see, e.g., [99, 79, 80]). An equivalent theorem like Theorem 2.25 can be formulated as well. Similarly, when one sets $K(\boldsymbol{x}, \boldsymbol{y}) = \prod_{j=1}^{s}(1 - \max(x_j, y_j))$ the formula for $T_2^\star$ is obtained, see (1.4).

As was already mentioned, for the fast component-by-component algorithm, a shift-invariant kernel is needed. An expression for the randomly shifted worst-case error can be derived by first calculating the associated shift-invariant kernel. For the L_2 star discrepancy kernel (2.16) that is

$$K^{\mathrm{shinv}}(\{\boldsymbol{x} - \boldsymbol{y}\}) = \int_{[0,1)^s} \prod_{j=1}^{s} (2 - \max(\{x_j + \Delta_j\}, \{y_j + \Delta_j\})) \, \mathrm{d}\boldsymbol{\Delta}.$$

First assume that $x > y$, then

$$\max(\{x + \Delta\}, \{y + \Delta\}) = \begin{cases} x + \Delta & \text{for } 0 \le \Delta < 1 - x, \\ y + \Delta & \text{for } 1 - x \le \Delta < 1 - y, \\ x + \Delta - 1 & \text{for } 1 - y \le \Delta < 1, \end{cases}$$

and thus

$$\begin{aligned} &\int_0^1 \max(\{x + \Delta\}, \{y + \Delta\}) \, \mathrm{d}\Delta \\ &\qquad = \int_0^{1-x} (x + \Delta) \, \mathrm{d}\Delta + \int_{1-x}^{1-y} (y + \Delta) \, \mathrm{d}\Delta + \int_{1-y}^{1} (x + \Delta - 1) \, \mathrm{d}\Delta \\ &\qquad = -(x - y)^2 + (x - y) + \frac{1}{2}. \end{aligned}$$

Obviously when $y > x$ then the roles of x and y are interchanged. Now since for both cases $(x - y)$ and $(y - x)$ are positive they could be written as $|x - y|$ and we notice that this is part of the Bernoulli polynomial of degree 2 which is $B_2(x) = x^2 - x + 1/6$. Furthermore $B_2(|x - y|) = B_2(\{x - y\})$ for $0 \le x, y \le 1$ because of (2.5). It follows that

$$K^{\text{shinv}}(\boldsymbol{x}) = \prod_{j=1}^{s} \left(B_2(x_j) + \frac{4}{3} \right),$$

$$\int_{[0,1)^s} K^{\text{shinv}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \prod_{j=1}^{s} \left(\int_0^1 B_2(x) \, \mathrm{d}x + \frac{4}{3} \right) = \left(\frac{4}{3} \right)^s,$$

and thus for this space and P_n the node set of a lattice rule, the randomly shifted worst-case error is given by

$$e(P_n, K^{\text{shinv}}) = \left(-\left(\frac{4}{3} \right)^s + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(B_2(x_j^{(k)}) + \frac{4}{3} \right) \right)^{1/2}.$$

For a general anchor $\boldsymbol{a}$, similar derivations as above give the following form for the associated shift-invariant kernel:

$$K^{\text{shinv}}(\boldsymbol{x}) = \prod_{j=1}^{s} \left(B_2(x_j) + a_j^2 - a_j + \frac{4}{3} \right).$$

2.4.2 The functional ANOVA decomposition

A useful way of looking at an s-dimensional function is to write it as a superposition of functions over all 2^s subsets of $\mathcal{D}_s = \{1, \ldots, s\}$, see, e.g., [102]. Let $\boldsymbol{x}_{\mathfrak{u}}$ be the subset of the vector $\boldsymbol{x}$ with components in $\mathfrak{u} \subseteq \mathcal{D}_s$. The ANOVA term corresponding to a set $\mathfrak{u}$ is denoted by $f_{\mathfrak{u}}$ and does only depend on the coordinates in $\mathfrak{u}$.

Definition 2.27. A function $f \in L_2([0,1)^s)$ has *analysis of variance (ANOVA) decomposition*

$$f(\boldsymbol{x}) = \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} f_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}),$$

where the order-$|\mathfrak{u}|$ terms are given by

$$f_{\mathfrak{u}}(\boldsymbol{x}) := \int_{[0,1)^{s-|\mathfrak{u}|}} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}_{\mathcal{D}_s \setminus \mathfrak{u}} - \sum_{\mathfrak{v} \subset \mathfrak{u}} f_{\mathfrak{v}}(\boldsymbol{x}).$$

Following from the definition, the constant ANOVA term, that is for $\mathfrak{u} = \emptyset$, is just the integral of the function,

$$f_\emptyset = \int_{[0,1)^s} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = I(f),$$

and the definition enforces that

$$\int_0^1 f_\mathfrak{u}(\boldsymbol{x})\,\mathrm{d}x_j = 0 \qquad \forall j \in \mathfrak{u},$$

from which it follows that $I(f_\mathfrak{u}) = 0$ for all $\mathfrak{u} \neq \emptyset$. Furthermore the different terms obey the following orthogonality condition

$$\int_{[0,1)^s} f_\mathfrak{u}(\boldsymbol{x})\, f_\mathfrak{v}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = 0 \qquad \text{for } \mathfrak{u} \neq \mathfrak{v}.$$

The ANOVA decomposition is a useful tool in studying the importance of the different sets of formal variables of a function. Since $f_\mathfrak{u}$ only depends on the coordinates in $\mathfrak{u}$, it fully describes the interaction between these variables. Given an s-dimensional function, in practice it is very unlikely that all 2^s subsets are equally important. It might well be that the variables are ordered in such a way that only the first few variables are important and that the effect of the remaining variables is negligible. Or maybe only terms up to order ℓ are important, i.e., where $|\mathfrak{u}| \leq \ell$. In such a case it might seem unfair to speak of an s-dimensional function.

In [10] the concept of *effective dimension* was formalized. Using the decomposition from Definition 2.27 the variance of the function f can be expressed as

$$\begin{aligned}\sigma^2(f) &:= \int_{[0,1)^s} (f(\boldsymbol{x}) - I(f))^2\,\mathrm{d}\boldsymbol{x} \\ &= \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} \sigma_\mathfrak{u}^2(f),\end{aligned}$$

where

$$\sigma_\mathfrak{u}^2(f) = \int_{[0,1)^s} (f_\mathfrak{u}(\boldsymbol{x}))^2\,\mathrm{d}\boldsymbol{x} \qquad \text{for } \mathfrak{u} \neq \emptyset \qquad \text{and} \qquad \sigma_\emptyset^2(f) = 0.$$

Now by comparing the variance over a specifically chosen subset of all possible $\mathfrak{u} \subseteq \mathcal{D}_s$ two kinds of effective dimension can be defined.

Definition 2.28. The function f has *truncation dimension* d_t if d_t is the smallest integer for which

$$\sum_{\mathfrak{u} \subseteq \{1,\ldots,d_t\}} \sigma_\mathfrak{u}^2(f) \geq p\,\sigma^2(f),$$

and the critical level $p \leq 1$ is an arbitrary value close to 1, e.g., $p = 0.99$.

Definition 2.29. The function f has *superposition dimension* d_s if d_s is the smallest integer for which

$$\sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_s\\ |\mathfrak{u}|\le d_s}} \sigma_{\mathfrak{u}}^2(f) \ge p\,\sigma^2(f),$$

and the critical level $p \le 1$ is an arbitrary value close to 1, e.g., $p = 0.99$.

As a simple example consider the function

$$f(\boldsymbol{x}) = 3x_1^2 + \cos(x_2)\sin(2\pi x_3). \tag{2.17}$$

This function has the following ANOVA terms

$$\begin{aligned} f_\emptyset &= 1,\\ f_{\{1\}} = 3x_1^2 - 1, \quad f_{\{3\}} &= \sin(1)\sin(2\pi x_3),\\ f_{\{2,3\}} &= \cos(x_2)\sin(2\pi x_3) - \sin(1)\sin(2\pi x_3), \end{aligned}$$

and all other ANOVA terms for this example are zero. Calculating the variance for $\{2,3\}$ shows that this term captures only 0.8% of the total variance. This means that this function has superposition dimension 1 when taking $p = 0.99$, i.e., 99%, in Definition 2.29. On the other hand, the function has truncation dimension 3 since the variance for $\{3\}$ does capture 30.4%. Simply reordering the variables by interchanging dimension 2 and 3 would make the function have truncation dimension 2 (again with $p = 0.99$).

The truncation dimension of a function might seem a not so useful concept. However, for many constructions of low-discrepancy point sets, it is the case that the first few dimensions have a better discrepancy. So it makes sense to be able to reorder the variables in non-increasing order of importance. For some problems this ordering follows naturally, e.g., for mortgage-backed securities [86], where each month money may or may not flow in, and where the integral calculates the expected value of the mortgage-backed security, the truncation dimension is much smaller than the nominal dimension [52, 109]. This is because (i) the value of the money decreases and (ii) the statistical distribution of the expectancy favors cash flows happening in the beginning of the time interval (modelling reality).

For the example function (2.17) we could use (1.9) to deduce that the point set used for integrating the function must have excellent discrepancies over $\{1\}$ and $\{3\}$, and from the order-2 discrepancy terms only $\{2,3\}$ is of importance. In fact using a lattice rule we could even take $z_1 \equiv z_3$ (such that the projection onto $\{1,3\}$ is the worst possible) since there is no interaction at all between these coordinates. For z_2 and z_3 we could take an optimal Fibonacci lattice rule.

Doing these things in practice is of course not possible (although there exist schemes to estimate the effective dimension of a function, see, e.g., [102]), but it

makes sense to define function spaces in which certain sets of coordinates dominate the problem and other sets are of lesser importance. This is done by weighting parts of the function space.

2.4.3 Weighted function spaces

The unweighted function spaces of the previous section are intractable for numerical integration, that is, for increasing number of dimensions the number of points needed is not bounded by a polynomial. By introducing weights Sloan and Woźniakowski [97] were able to show that, under certain conditions on these weights, numerical integration could be (strongly) tractable.

Sloan and Woźniakowski [97], as well as Hickernell [42, 43, 44] introduced weights $\gamma_{\mathfrak{u}} \geq 0$ for each set $\mathfrak{u} \subseteq \mathcal{D}_s$. As in [97], one possible method would start by multiplying and dividing with $\gamma_{\mathfrak{u}}^{1/2}$ in the Hlawka–Zaremba identity (1.8),

$$\begin{aligned} &|Q(f;P_n) - I(f)| \\ &\quad = \sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} (-1)^{|\mathfrak{u}|} \int_{[0,1)^{|\mathfrak{u}|}} \gamma_{\mathfrak{u}}^{1/2} \operatorname{discr}([\mathbf{0}, \boldsymbol{x}_{\mathfrak{u}}), P_n(\mathfrak{u}))\, \gamma_{\mathfrak{u}}^{-1/2} \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x}_{\mathfrak{u}}, \mathbf{1})}{\partial \boldsymbol{x}_{\mathfrak{u}}}\, \mathrm{d}\boldsymbol{x}_{\mathfrak{u}}, \end{aligned} \tag{2.18}$$

and then by using the Cauchy–Schwarz inequality (or Hölder for a more general form) obtain a bound in terms of the weighted L_2 discrepancy and variation

$$\begin{aligned} |Q(f;P_n) - I(f)| \leq &\left(\sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} \int_{[0,1)^{|\mathfrak{u}|}} \left| \operatorname{discr}([\mathbf{0}, \boldsymbol{x}_{\mathfrak{u}}), P_n(\mathfrak{u})) \right|^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}} \right)^{1/2} \\ &\times \left(\sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1)^{|\mathfrak{u}|}} \left| \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x}_{\mathfrak{u}}, \mathbf{1})}{\partial \boldsymbol{x}_{\mathfrak{u}}} \right|^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}} \right)^{1/2}. \end{aligned} \tag{2.19}$$

Another natural way is to start from a reproducing kernel Hilbert space based on the ANOVA decomposition and then directly add weights to control the importance of lower and higher order ANOVA terms as was done in [42].

If for a given set of components $\mathfrak{u}$ the weight is set to 0 then this models that there is no interaction amongst the variables in $\mathfrak{u}$ (then leaving out $\mathfrak{u}$ in (2.18) as well as in (2.19) by formally setting $0 \times \infty = 0$).

Such a weighting scheme is the most general possible, i.e., there are 2^s weights. Limiting the weights to a reasonable number is necessary if one wants to be able to specify these weights to describe a specific function space. This is even more so if one wants to calculate the worst-case error in such a weighted function space.

Reproducing kernel Hilbert spaces with general weights

Consider a weighted reproducing kernel Hilbert space $\mathcal{H}(K)$ with kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} K_{\mathfrak{u}}(\boldsymbol{x}, \boldsymbol{y}), \tag{2.20}$$

which is a direct sum of 2^s reproducing kernel Hilbert spaces $\mathcal{H}_{\mathfrak{u}}(K_{\mathfrak{u}})$,

$$\mathcal{H}(K) = \bigoplus_{\mathfrak{u} \subseteq \mathcal{D}_s} \mathcal{H}_{\mathfrak{u}}(K_{\mathfrak{u}}),$$

where

$$K_{\mathfrak{u}}(\boldsymbol{x}, \boldsymbol{y}) = K_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}, \boldsymbol{y}_{\mathfrak{u}}) = \prod_{j \in \mathfrak{u}} \eta(x_j, y_j). \tag{2.21}$$

To fix the scaling, set $K_\emptyset = 1$ and $\gamma_\emptyset = 1$. Tractability of multivariate integration in the worst-case setting in such spaces was studied in, e.g., [49], [48], [28] and [95].

Since the kernels (2.21) are of product form, it follows that the spaces $\mathcal{H}_{\mathfrak{u}}(K_{\mathfrak{u}})$ are tensor-product spaces. However, the space $\mathcal{H}(K)$ with kernel (2.20) is not necessarily a tensor-product space.

A function $f \in \mathcal{H}(K)$ has a unique decomposition

$$f(\boldsymbol{x}) = \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} f_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}), \qquad \text{with} \quad f_{\mathfrak{u}} \in \mathcal{H}_{\mathfrak{u}}(K_{\mathfrak{u}}),$$

and the inner product with the kernel $K_{\mathfrak{u}}$ is a projection operator for $f \in \mathcal{H}(K)$ to the space $\mathcal{H}_{\mathfrak{u}}(K_{\mathfrak{u}})$,

$$f_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}) = \langle K_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}, \cdot), f \rangle_{\mathcal{H}}.$$

In (2.20) a weight $\gamma_{\mathfrak{u}}$ is associated with each space $\mathcal{H}_{\mathfrak{u}}$ to model the level of interaction amongst variables in $\mathfrak{u}$ for the complete space. The 2^s weights model the relative importance between all the possible sets of variables. Naturally a small $\gamma_{\mathfrak{u}}$ means that the contribution of $f_{\mathfrak{u}}$ is small compared to the other terms.

For such a general weighted function space the worst-case error can be calculated by Theorem 2.18. Here, we are the most interested in the randomly shifted worst-case error based on the associated shift-invariant kernel, see Theorem 2.22. The associated shift-invariant kernel of (2.20) is given by

$$K^{\mathrm{shinv}}(\boldsymbol{x}) = \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} K_{\mathfrak{u}}^{\mathrm{shinv}}(\boldsymbol{x}),$$

where

$$K_{\mathfrak{u}}^{\text{shinv}}(\boldsymbol{x}) = \prod_{j\in\mathfrak{u}} \omega(x_j) \qquad \text{and} \qquad \omega(x) = \int_0^1 \eta\left(\{x+\Delta\},\Delta\right) \mathrm{d}\Delta.$$

Assuming P_n to be the point set of a lattice rule, it follows that the randomly shifted worst-case error has the form

$$e(P_n, K^{\text{shinv}}) = \left(-\kappa_s + \frac{1}{n}\sum_{k=0}^{n-1}\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \omega(x_j^{(k)})\right)^{1/2},$$

with

$$\kappa_s = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \int_0^1 \omega(x)\,\mathrm{d}x.$$

It is convenient to have

$$\int_0^1 \omega(x)\,\mathrm{d}x = 0,$$

such that $\kappa_s = \gamma_\emptyset = 1$. Furthermore it follows from (2.13) that the initial error in the space $\mathcal{H}$ is then 1. In fact, in Hickernell [44], it is recommended to scale the kernel such that $\kappa_s = 1$ to be able to compare worst-case errors. This is the case when (2.20) is an ANOVA decomposition.

The Korobov space which was introduced in §2.4.1 can of course be extended with general weights [29].

Definition 2.30. The *weighted Korobov space* has reproducing kernel

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \sum_{0\neq h\in\mathbb{Z}} r(h)^{-\alpha} \exp(2\pi\mathrm{i}\, h(x_j - y_j)),$$

inner product

$$\langle f, g\rangle = \sum_{\boldsymbol{h}\in\mathbb{Z}^s} \gamma_{\mathfrak{u}_{\boldsymbol{h}}}^{-1}\, r(\boldsymbol{h})^{\alpha}\, \hat{f}(\boldsymbol{h})\, \overline{\hat{g}(\boldsymbol{h})},$$

and norm

$$\|f\| = \left(\sum_{\boldsymbol{h}\in\mathbb{Z}^s} \gamma_{\mathfrak{u}_{\boldsymbol{h}}}^{-1}\, r(\boldsymbol{h})^{\alpha}\, |\hat{f}(\boldsymbol{h})|^2\right)^{1/2},$$

where $\mathfrak{u}_{\boldsymbol{h}} := \{j \in \mathfrak{u} : h_j \neq 0\}$, $r(\boldsymbol{h})$ is as defined in (2.3) and the smoothness parameter $\alpha > 1$.

This space is naturally shift-invariant and thus

$$\omega(x) = \sum_{0\neq h\in\mathbb{Z}} r(h)^{-\alpha}\, \exp(2\pi \mathrm{i}\, hx),$$

which for even $\alpha > 1$ can be expressed in terms of the Bernoulli polynomial of degree α, see (2.4). The worst-case error for a lattice rule with point set P_n is given by

$$\begin{aligned} e(P_n, K) &= \left(-1 + \frac{1}{n}\sum_{k=0}^{n-1}\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \omega(x_j^{(k)})\right)^{1/2} \\ &= \left(\frac{1}{n}\sum_{k=0}^{n-1}\sum_{\emptyset\neq\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \omega(x_j^{(k)})\right)^{1/2}. \end{aligned} \tag{2.22}$$

In §2.4.1 an anchored Sobolev space was defined. For such a space the choice of the anchor is an arbitrary but important choice. It is also possible to define an unanchored space, e.g., by integrating away the part of the function where the derivative is not taken. One such choice which is recommended in [99] to be used as the preferred Sobolev space is the following.

Definition 2.31. The *weighted unanchored Sobolev space* has reproducing kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \tfrac{1}{2} B_2(\{x_j - y_j\}) + (x_j - \tfrac{1}{2})(y_j - \tfrac{1}{2}),$$

inner product $\langle f, g\rangle =$

$$\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1)^{|\mathfrak{u}|}} \left(\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}}\, \mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right)\left(\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} g(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}}\, \mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right) \mathrm{d}\boldsymbol{x}_{\mathfrak{u}},$$

and norm

$$\|f\| = \left(\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1)^{|\mathfrak{u}|}} \left|\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}}\, \mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right|^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}}\right)^{1/2}.$$

The kernel of this unanchored Sobolev space is an ANOVA decomposition while this is not so for the anchored Sobolev space. Note that

$$\int_0^1 (\{x + \Delta\} - \tfrac{1}{2})(\{y + \Delta\} - \tfrac{1}{2})\, \mathrm{d}\Delta = \tfrac{1}{2} B_2(|x - y|) = \tfrac{1}{2} B_2(\{x - y\}),$$

and thus the associated shift-invariant kernel has

$$\omega(x) = B_2(x).$$

For the associated space the initial error equals 1 and the randomly shifted worst-case error has the form (2.22). Also the anchored Sobolev space can be weighted with trivial changes to Definition 2.26 but then the initial error for randomly shifted rules is given by

$$1 + \sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j \in \mathfrak{u}} \left(a_j^2 - a_j + \frac{1}{3} \right).$$

Reproducing kernel Hilbert spaces with product weights

The first limiting choice for the weights is to consider only s weights, one per dimension, and to set

$$\gamma_{\mathfrak{u}} = \prod_{j \in \mathfrak{u}} \gamma_{\{j\}}.$$

When there is no confusion the initially chosen weights $\gamma_{\{j\}}$, $j = 1, \ldots, s$, are denoted by γ_j. It is assumed that the dimensions are ordered in such a way that the weights are non-increasing,

$$\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_s \geq 0,$$

and thus the first dimension is the most important and the following become less and less important. If this is not naturally the case for a given integrand function then, e.g., principal component analysis (PCA) can be used to obtain a new set of variables for which this assumption holds (see also §8.1). This weight setting is naturally connected to the truncation dimension which was introduced in Definition 2.28.

The product weights setting leads to a tensor-product space, since the kernel can now be written as a product:

$$\begin{aligned} K(\boldsymbol{x}, \boldsymbol{y}) &= \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} \prod_{j \in \mathfrak{u}} \gamma_j \, \eta(x_j, y_j) \\ &= \prod_{j=1}^{s} \left(1 + \gamma_j \, \eta(x_j, y_j)\right). \end{aligned}$$

For P_n the node set of a lattice rule, the randomly shifted worst-case error takes the form

$$e(P_n, K^{\mathrm{shinv}}) = \left(-\kappa_s + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(1 + \gamma_j \, \omega(x_j)\right) \right)^{1/2}. \tag{2.23}$$

For the weighted Korobov space and the weighted unanchored Sobolev space $\kappa_s = 1$. For the anchored Sobolev space with anchor $\boldsymbol{a}$, set $A_j = a_j^2 - a_j + \frac{1}{3}$, then the randomly shifted worst-case error can be written as

$$e(P_n, K^{\text{shinv}}) = \left(-\prod_{j=1}^{s}(1+\gamma_j A_j) + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}(1+\gamma_j (B_2(x_j)+A_j)) \right)^{1/2}$$
$$= \left(-\prod_{j=1}^{s}\beta_j + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}(\beta_j+\gamma_j B_2(x_j)) \right)^{1/2}, \tag{2.24}$$

with

$$\beta_j = 1 + \gamma_j A_j.$$

The slightly more general form (2.24) will be used in the next chapter in the development of the fast algorithm for product weights.

Reproducing kernel Hilbert spaces with order dependent weights

Another sensible choice is to fix weights for each order-ℓ interaction, that is

$$\gamma_{\mathfrak{u}} = \Gamma_{|\mathfrak{u}|},$$

with $\Gamma_\emptyset = \gamma_\emptyset = 1$. Again these are only s weights instead of 2^s, and the relative scaling amongst the Γ_ℓ denote the relative importance of groups of ℓ dimensions.

A kernel with order dependent weights has the form

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\ell=0}^{s}\Gamma_\ell \sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_s\\|\mathfrak{u}|=\ell}}\prod_{j\in\mathfrak{u}}\eta(x_j,y_j)$$
$$= 1 + \sum_{\ell=1}^{s}\Gamma_\ell \sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_s\\|\mathfrak{u}|=\ell}}\prod_{j\in\mathfrak{u}}\eta(x_j,y_j).$$

For P_n the node set of a lattice rule, the randomly shifted worst-case error is given by

$$e(P_n, K^{\text{shinv}}) = \left(-\kappa_s + 1 + \frac{1}{n}\sum_{k=0}^{n-1}\sum_{\ell=1}^{s}\Gamma_\ell \sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_s\\|\mathfrak{u}|=\ell}}\prod_{j\in\mathfrak{u}}\omega(x_j^{(k)}) \right)^{1/2}. \tag{2.25}$$

If $\int_0^1 \omega(x)\,\mathrm{d}x = 0$ and thus $\kappa_s = 1$, this becomes

$$e(P_n, K^{\mathrm{shinv}}) = \left(\frac{1}{n} \sum_{k=0}^{n-1} \sum_{\ell=1}^{s} \Gamma_\ell \sum_{\substack{\mathfrak{u} \subseteq \mathcal{D}_s \\ |\mathfrak{u}|=\ell}} \prod_{j \in \mathfrak{u}} \omega(x_j^{(k)}) \right)^{1/2}. \tag{2.26}$$

Often higher order interactions are of no importance. In that case the weights for $\ell > q^\star$, for some $q^\star \leq s$, are set to zero. This weight setting is called *finite order weights* of order $q^\star$. Trivially, the sums in (2.25) and (2.26) then only have to go up to $q^\star$ instead of s.

Note that equal product weights,

$$\gamma_{\{j\}} = r, \qquad \text{such that } \gamma_{\mathfrak{u}} = \prod_{j \in \mathfrak{u}} r = r^{|\mathfrak{u}|},$$

is the same as taking order dependent weights of the form

$$\Gamma_\ell = r^\ell.$$

As such the scaling of product weights is important. This is not the case for order dependent weights, where only the relative ratios matter; multiplying all Γ_ℓ by r just multiplies the squared worst-case error by r.

2.4.4 Existence and tractability

Although the main topic of this text is fast construction of lattice rules (for a given weighted reproducing kernel Hilbert space), something must be said about the purpose of the weights. In this section we glance over conditions on the weights for certain function spaces and the existence of good lattice rules for these spaces. A nice overview of this theory is given in [64] as well as in the survey [80].

First define the quasi-Monte Carlo mean for the worst-case error.

Definition 2.32. The *quasi-Monte Carlo mean worst-case error* is the root mean square average of the worst-case error over all possible choices for each point:

$$e_n^{\mathrm{avg}}(K) := \left(\int_{[0,1)^{ns}} e_n^2(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}, K)\,\mathrm{d}\boldsymbol{x}_1 \cdots \mathrm{d}\boldsymbol{x}_n \right)^2.$$

The wording "quasi-Monte Carlo mean" is used to denote that this is the mean over all equal weight cubature rules of the from (1.2). Compare this definition with (1.3) for the calculation of the average Monte Carlo error.

The quasi-Monte Carlo mean can be written in terms of the kernel, see, e.g., [80], as given in the following theorem.

Theorem 2.33. *The quasi-Monte Carlo mean worst-case error can be written as*

$$\left(e_n^{\mathrm{avg}}(K)\right)^2 = \frac{1}{n}\left(\int_{[0,1)^s} K(\boldsymbol{x},\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} - \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}\right).$$

Proof. From Theorem 2.18 we can write

$$e^2(Q,K) = \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} - \frac{2}{n}\sum_{k=0}^{n-1}\int_{[0,1)^s} K(\boldsymbol{x}_k,\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} + \frac{1}{n^2}\sum_{\substack{k,\ell=0\\ k\neq\ell}}^{n-1} K(\boldsymbol{x}_k,\boldsymbol{x}_\ell) + \frac{1}{n^2}\sum_{k=0}^{n-1} K(\boldsymbol{x}_k,\boldsymbol{x}_k).$$

Now apply Definition 2.32 to find

$$\left(e_n^{\mathrm{avg}}(K)\right)^2 = \kappa_s - 2\kappa_s + \tfrac{n-1}{n}\kappa_s + \tfrac{1}{n}\tau_s,$$

with

$$\kappa_s = \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} \qquad \text{and} \qquad \tau_s = \int_{[0,1)^s} K(\boldsymbol{x},\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}.$$

□

Having defined the mean worst-case error we can pick up the thought from §1.1 again: *there must exist point sets which have at most this error*. Some expressions for the squared mean worst-case error for the function spaces from the previous section with product weights are given in Table 2.1. For fixed s it can be seen that for each n there exist point sets which achieve a rate of convergence at least as good as $O(n^{-1/2})$, which is the expected Monte Carlo rate of convergence (taking the points randomly).

For the question of tractability we are interested in the behavior for increasing s. Here the weights come into play. It can be shown from the results in Table 2.1 that if

$$\sum_{j=1}^{\infty} \gamma_j < \infty \tag{2.27}$$

holds, then integration in these spaces is strongly tractable and if

$$\limsup_{s\to\infty} \frac{\sum_{j=1}^{\infty}\gamma_j}{\log(s)} < \infty$$

holds, then integration in these spaces is tractable (see, e.g., [97], [50] and [98]).

space	$(e_n^{\mathrm{avg}}(K))^2$
Korobov space	$n^{-1}\left(\prod_{j=1}^{s}(1+2\gamma_j\zeta(\alpha))-1\right)$
unanchored Sobolev space	$n^{-1}\left(\prod_{j=1}^{s}(1+\frac{\gamma_j}{6})-1\right)$
anchored Sobolev space	$n^{-1}\left(\prod_{j=1}^{s}(1+\gamma_j(a_j^2-a_j+\frac{1}{2}))-\prod_{j=1}^{s}(1+\gamma_j(a_j^2-a_j+\frac{1}{3}))\right)$

Table 2.1: Squared mean worst-case error for different tensor-product spaces.

So if the sum of the weights is summable, i.e., condition (2.27) holds, then it is known that there exist quasi-Monte Carlo point sets that achieve a worst-case error bound at least as good as the average, independent of the dimension (and s can be set to ∞ in Table 2.1). This proves existence, but for which point sets?

By averaging over all possible choices of the generating vector for a rank-1 lattice rule it is possible to show that good rank-1 lattice rules do exist by bounding this average with those in Table 2.1. Moreover, the weights make it possible to achieve a better convergence order than $O(n^{-1/2})$. Obviously one can never do better than the optimal one-dimensional rate of convergence.

The optimal rate of convergence for the Korobov space with smoothness α, $\alpha > 1$, in the one-dimensional case is $O(n^{-\alpha/2})$. Given the stronger condition

$$\sum_{j=1}^{\infty} \gamma_j^{\frac{1}{\alpha-2\delta}} < \infty, \tag{2.28}$$

the order of convergence for multivariate integration in the Korobov space with smoothness α is given by $O(n^{-\alpha/2+\delta})$, for any $\delta > 0$, see [98, 62].

For the Sobolev space it is known that it is impossible to do better than $O(n^{-1})$ for the one-dimensional case. Taking the stronger condition

$$\sum_{j=1}^{\infty} \gamma_j^{\frac{1}{2(1-\delta)}} < \infty, \tag{2.29}$$

the order of convergence for multivariate integration in the Sobolev space is given by $O(n^{-1+\delta})$, for any $\delta > 0$, see [98, 62].

(For observations on general weights and order dependent weights see [95].)

2.5 Constructions

Knowing that there exist lattice rules which attain the optimal order of convergence is of course a first step. Finding such rules is a completely different story. It turns

out that except for the two-dimensional case there are no explicit constructions known that achieve the optimal order. Therefore, one has to resort to an exhaustive search; calculating the worst-case error for each possible choice.

Only search strategies for rank-1 lattice rules will be discussed here. These have point set

$$\boldsymbol{x}_k = \left\{ \frac{k\boldsymbol{z}}{n} \right\} \qquad \text{for } k = 0, 1, \ldots, n-1, \tag{2.30}$$

with $\boldsymbol{z} \in \mathbb{Z}^s$ the generating vector and $\{\cdot\}$ means to take the fractional part, i.e., modulo 1, componentwise. Furthermore, the components of the generating vector are taken relatively prime to n, $\gcd(z_j, n) = 1$. Since then $\gcd(z_1, \ldots, z_s, n) = 1$ the rule uses n distinct points, see Definition 1.10.

The fractional notation in (2.30) can also be written with a modular notation like

$$\left\{ \frac{k\boldsymbol{z}}{n} \right\} = \frac{k\boldsymbol{z} \bmod n}{n},$$

and the components of $\boldsymbol{z}$ can thus be restricted to $\mathbb{Z}_n = \{0, \ldots, n-1\}$. Together with $\gcd(z_j, n) = 1$ the components of $\boldsymbol{z}$ are essentially limited to the set of *units modulo* n, i.e., the multiplicative group modulo n,

$$U_n := \{v \in \mathbb{Z}_n : \gcd(v, n) = 1\} = \mathbb{Z}_n^\times, \qquad |U_n| = \varphi(n).$$

The number of possible generating vectors for an s-dimensional lattice rule is therefore given by the Euler totient function as $\varphi(n)^s = O(n^s)$, which is exponential in the number of dimensions. Obviously, when the number of dimensions is higher, also the number of points gets much higher—integration becomes harder—so the situation is even worse than it seems at first sight.

When the function space under consideration is shift-invariant, or the randomly shifted worst-case error is considered, we can use the symmetry property of the kernel from Lemma 2.12. This means that the ω function has the symmetry

$$\omega(t) = \omega(1 - t).$$

It follows that this symmetry is also reflected in the worst-case error and we could change any component of $\boldsymbol{z}$ with its additive inverse modulo n, i.e.,

$$z \simeq n - z \equiv -z \pmod{n}. \tag{2.31}$$

Therefore it suffices to only look at $(\varphi(n)/2)^s$ choices but asymptotically this is still $O(n^s)$.

For non-weighted function spaces also a permutation of the components of $\boldsymbol{z}$ does not change the worst-case error. This leads to the concept of *geometrically equivalent* lattice rules, see [89, Definition 2.12]. Since this definition implies that all weights are 1 it is not useful in the context of this text.

2.5.1 Fibonacci lattice rules

For the two-dimensional case it was noticed that lattice rules based on Fibonacci numbers give an extremely good point set [5, 61]. The Fibonacci lattice rules have the minimal value for P_α in two dimensions. Recall that the classical P_α criterion is the worst-case error in an unweighted Korobov space. Fibonacci lattice rules also obtain optimal values for related error measures like the Zaremba index, the trigonometric degree and Niederreiter's R criterion which are not discussed here (see [89, 76] for more information).

The Fibonacci numbers are defined by the following recurrence

$$F_0 = 0; \qquad F_1 = 1; \qquad F_m = F_{m-1} + F_{m-2} \quad \text{for } m > 1. \tag{2.32}$$

Now take the total number of points of the lattice rule to be a Fibonacci number F_m, a Fibonacci lattice rule is then defined as having generating vector

$$\boldsymbol{z} = (1, F_{m-1}).$$

(Which due to the symmetry is equivalent to $\boldsymbol{z} = (1, F_{m-2})$, since $F_m - F_{m-1} = F_{m-2}$ according to (2.32).)

The Fibonacci lattice rules are related to the Kronecker sequence. Likewise their optimality is proven by using the continued fraction representation of $\frac{F_{m-1}}{F_m}$ for which all partial quotients are equal to 1. Remarkably

$$\lim_{m\to\infty} \frac{F_{m-1}}{F_m} = \tau = \frac{1+\sqrt{5}}{2}$$

is the golden ratio which has all partial quotients equal to 1 (an infinite number); an optimal one-dimensional Kronecker sequence is actually given by taking multiples of the golden ratio modulo 1, i.e., $\{k\tau\}$, $k = 0, 1, \ldots$. Some pictures with Fibonacci lattice rules were given in Chapter 1, see Figure 1.3.

2.5.2 Korobov-type rules

In [59] Korobov limited the search space for $\boldsymbol{z}$ by searching for generating vectors of a specific form while still being able to prove existence of optimal rules (for the unweighted Korobov space). Instead of searching for s optimal coefficients, Korobov defines the generating vector to be determined by only one parameter $a \in U_n$.

Definition 2.34. A *Korobov-type lattice rule* is a rank-1 lattice rule of the form

$$Q(f) = \frac{1}{n} \sum_{k=0}^{n-1} f\left(\left(\left\{\frac{k}{n}\right\}, \left\{\frac{ka}{n}\right\}, \left\{\frac{ka^2}{n}\right\}, \ldots, \left\{\frac{ka^{s-1}}{n}\right\}\right)\right),$$

which has generating vector

$$\boldsymbol{z} \equiv (a^0, a^1, a^2, \ldots, a^{s-1}) \pmod{n}. \tag{2.33}$$

Calculating the worst-case error for lattice rules of the Korobov-type has complexity $O(ns)$ in a shift-invariant tensor-product space, see Corollary 2.21. There are $\varphi(n)/2$ possible choices for a such that the total construction complexity is of the form $O(sn^2)$. Compared to $O(n^s)$ this is a huge improvement. Korobov also devised a way to speed up the calculations when n is a product of two primes. However, when s and n are large, say, 100 and 1 million, these calculations are infeasible.

2.5.3 Component-by-component construction

Another way of limiting the search space is by choosing the components of $\boldsymbol{z}$ component by component as introduced by Sloan and Reztsov [93]. Such a method was first deemed not likely to be a successful strategy by Niederreiter [75, p. 987] but shown to work well by Sloan and Reztsov.

An inductive argument proves existence by assuming a good lattice rule in s dimensions and then showing, by an averaging argument, that there is at least one choice for z_{s+1} which gives a good lattice rule in $s+1$ dimensions. This construction method makes the generated lattice rule naturally extensible in the number of dimensions. The algorithm is given in Algorithm 1, where $s_{\max}$ can be considered as an open parameter.

Algorithm 1 Component-by-component construction

for $s = 1$ **to** $s_{\max}$ **do**
  for all $z_s \in U_n$ **do**
    calculate $e_s^2(z_1, \ldots, z_{s-1}, z_s)$
  end for
  $z_s = \operatorname*{argmin}_{z \in U_n} e_s^2(z_1, \ldots, z_{s-1}, z)$
end for

The typical construction cost of this algorithm is $O(s^2n^2)$ for a shift-invariant tensor-product space, see Corollary 2.21. If the kernel has a more complicated form then the construction cost is $O(sn\theta)$ with θ the cost of evaluating the worst-case error. The same remarks as for the Korobov-type rules apply: for large s and n the cost is still too high. Dick and Kuo [23, 22] used similar methods as Korobov to reduce the cost for n a product of several primes.

More importantly, Frances Y. Kuo showed that this algorithm constructs lattice rules which achieve the optimal rate of convergence in Korobov and Sobolev space [62] (see also Dick [21]). For suitable weights, see (2.28) and (2.29), that is respectively $O(n^{-\alpha/2+\delta})$ and $O(n^{-1+\delta})$ for $\delta > 0$.

2.5.4 Preview of fast component-by-component

The construction cost of the component-by-component construction can be significantly reduced to $O(sn\log(n))$. The following chapters will all deal with the fast component-by-component construction. This fast algorithm makes it possible to construct lattice rules with a large number of points (say $n = 10^8$), which was practically intractable with the original algorithm.

Using number theoretic mappings it is possible to rewrite the calculations in the component-by-component algorithm in such a way that a fast Fourier transform (FFT) can be used to speed up the calculations. Basically the factor $O(n^2)$ in the construction complexity for tensor-product spaces can be seen as coming from a matrix-vector product which, with the use of FFTs, can be done in $O(n\log(n))$. The matrices that we obtain are (nested) circulant matrices.

The development of the fast algorithm will require the following properties:

(i) The error criterion must look like the worst-case error in a reproducing kernel Hilbert space.

(ii) This function space must be a tensor product of one-dimensional spaces (not necessarily all the same) or a (small) direct sum of such tensor-product spaces.

(iii) Its reproducing kernel must be shift-invariant.

Chapter notes

The classical theory about lattice rules can be found in the books from Sloan and Joe [89] and Niederreiter [76]. For the more recent results on the complexity side the overview article from Kuo and Sloan [64] is highly recommended. The part about the function spaces was derived from the work by Hickernell [44]. The product trigonometric space in §2.2 was the initial space under consideration for the research which resulted in this thesis and evolved into the search for a fast component-by-component algorithm.

This chapter only discusses a very limited view on the construction of lattice rules, being optimizing for the worst-case error. It is also possible to construct lattice rules which are exact for a given class of functions, e.g., the trigonometric polynomials up to a certain degree. The search for trigonometric lattice rules was mainly done by Russian researchers, and limited to low dimensions, see [6] for an overview. Also for this space the search space was limited to escape the exponential construction cost [16] and, in a totally different context, skew-circulant matrices came into play, see [14, 71] and also [17].

Chapter 3

Construction for a prime number of points

This first chapter on the construction of lattice rules will deal with rank-1 lattice rules which have a prime number of points n. Throughout the whole literature on lattice rules it turns out that the prime case is always the easy case. Here it is no different and in a way the fast construction algorithm for a prime number of points becomes the base case of the more general algorithm from Chapter 4.

For product weighted spaces the original component-by-component algorithm has a construction cost of $O(s^2n^2)$ in s dimensions for an n-point rule, or, more appropriately, $O(sn^2)$ when using $O(n)$ memory. In contrast, we will show that our fast algorithm has a construction cost of $O(sn\log(n))$ using $O(n)$ memory. The speedup is achieved by making use of an n-point fast Fourier transform (FFT).

3.1 Modular notation

We consider s-dimensional multivariate integration with a rank-1 lattice with point set

$$P_n := \left\{ \frac{k \cdot \boldsymbol{z}}{n} : 0 \leq k < n \right\}.$$

By $k \cdot \boldsymbol{z}$ we mean (componentwise) multiplication modulo n. This notation is preferred over the more usual notation where curly braces mean to take the (componentwise) fractional part. It is the modulo multiplication structure which will lead us to the fast construction algorithm. For clarity, the following are thus

all equivalent

$$\frac{k \cdot \boldsymbol{z}}{n} = \frac{k\boldsymbol{z} \bmod n}{n} = \frac{k\boldsymbol{z}}{n} \bmod 1 = \left\{ \frac{k\boldsymbol{z}}{n} \right\}. \tag{3.1}$$

The integer vector $\boldsymbol{z}$ is called the generating vector of the lattice and its components are taken from the set

$$U_n := \{z \in \mathbb{Z}_n : \gcd(z, n) = 1\} = \mathbb{Z}_n^\times,$$

and thus $\boldsymbol{z} \in U_n^s$. In this chapter n is prime, as such this set simplifies to

$$U_n = \mathbb{Z}_n \setminus \{0\} = \{1, \ldots, n-1\} \qquad (n \text{ prime}).$$

The derivation of the fast algorithm will first be done for product weights. The modifications for order dependent weights will be derived in §3.3.

3.2 Construction for product weights

3.2.1 Component-by-component construction

The setting for the algorithm will be the worst-case error in a shift-invariant tensor-product reproducing kernel Hilbert space. From §2.4.3, the expression for the squared worst-case error for product weights in this space is given by

$$e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_1, \ldots, z_s) = -\prod_{j=1}^{s} \beta_j + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(\beta_j + \gamma_j \, \omega\left(\frac{k \cdot z_j}{n} \right) \right). \tag{3.2}$$

The kernel is a product of one-dimensional weighted kernels,

$$K_{s,\gamma,\beta}(\boldsymbol{x}) = \prod_{j=1}^{s} \left(\beta_j + \gamma_j \, \omega(x_j) \right), \tag{3.3}$$

where the γ_j are positive weights which denote the importance of the different dimensions. The product weights γ_j are taken as a decaying sequence of positive weights,

$$\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_s \geq 0,$$

which means that the successive coordinates are less and less important. The function ω was called the *variable* part of the kernel by us in [83], since it can be seen to correspond to the non-constant frequency part of the Fourier series of the one-dimensional kernel (up to scaling):

$$K_{1,\gamma,\beta}(x) = \sum_{h \in \mathbb{Z}} \tilde{k}_h \exp(2\pi \mathrm{i} \, hx) = \tilde{k}_0 + \sum_{h \in \mathbb{Z} \setminus \{0\}} \tilde{k}_h \exp(2\pi \mathrm{i} \, hx).$$

Note that this makes perfect sense, since a shift-invariant space contains periodic functions and it would be natural to scale $\tilde{k}_0$ to 1 so that $I(f) = 1$. The weights β_j in (3.3) are there to accommodate the change from another reproducing kernel Hilbert space to the associated shift-invariant space, as was explained in Lemma 2.13 and the notation was introduced in (2.24).

A simple calculation shows that the worst-case error (3.2) for a function space with weights γ_j and β_j can be written in terms of the worst-case error in a function space with weights $\widehat{\gamma}_j = \gamma_j/\beta_j$ and $\widehat{\beta}_j = 1$. The worst-case error is then obtained as a scaled version of the worst-case error in this normalized function space as

$$e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}} = \prod_{j=1}^{s} \beta_j \; e^2_{s,\widehat{\boldsymbol{\gamma}},\widehat{\boldsymbol{\beta}}} = \prod_{j=1}^{s} \beta_j \; \left(-1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(1 + \frac{\gamma_j}{\beta_j} \, \omega\left(\frac{k \cdot z_j}{n} \right) \right) \right).$$

From this it can be seen that (up to scaling) only the ratios γ_j/β_j matter. We want to emphasize however, that the function ω can be *any* function in our derivations, and so the choice of the reproducing kernel is completely free.

Algorithm 2 (Original) Component-by-component construction for shift-invariant tensor-product reproducing kernel Hilbert space

for $s = 1$ **to** $s_{\max}$ **do**
  for all $z_s \in U_n$ **do**
    $$e^2_s(z_s) = -\prod_{j=1}^{s} \beta_j + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(\beta_j + \gamma_j \, \omega\left(\frac{k \cdot z_j}{n} \right) \right)$$
  end for
  $z_s = \operatorname*{argmin}_{z \in U_n} e^2_s(z)$
end for

Given the expression for the worst-case error (3.2) and the description of the component-by-component construction, see Algorithm 1, we can present the original algorithm as Algorithm 2. The algorithm is specified with a maximum dimension $s_{\max}$, but it can be stopped anywhere. Therefore the complexity of the algorithm will be presented in s, as an open parameter. In the algorithm the worst-case error in dimension s is considered as a function of all the possible choices of $z_s \in U_n$. We then pick the minimum and keep this choice fixed for the following iterations. As was briefly explained in §2.4.3, under certain conditions on the weights γ_j, it can be shown that the constructed lattice rule achieves the optimal rate of convergence, see [62, 53, 21], and thus this construction method is justified.

A quick inspection of Algorithm 2 shows that a direct implementation would have a construction cost of $O(s^2 n^2 c_\omega)$, with c_ω the cost of evaluating the ω function.

Since this is an iterative process in s, the product over s terms can be split in two parts: a product over the first $s-1$ terms and the last term, i.e.,

$$\prod_{j=1}^{s}\left(\beta_j+\gamma_j\,\omega\left(\frac{k\cdot z_j}{n}\right)\right)=\prod_{j=1}^{s-1}\left(\beta_j+\gamma_j\,\omega\left(\frac{k\cdot z_j}{n}\right)\right)\left(\beta_s+\gamma_s\,\omega\left(\frac{k\cdot z_s}{n}\right)\right). \tag{3.4}$$

In iteration s the product up to dimension $s-1$ is fixed, since z_j, for $j<s$, is not changed anymore. Consequently the product can be stored in an n-vector denoted $\boldsymbol{p}_{s-1}$. Equation (3.4) then becomes

$$\prod_{j=1}^{s}\left(\beta_j+\gamma_j\,\omega\left(\frac{k\cdot z_j}{n}\right)\right)=p_{s-1}(k)\;\left(\beta_s+\gamma_s\,\omega\left(\frac{k\cdot z_s}{n}\right)\right),$$

which needs to be calculated for each k and each possible choice for z_s. This immediately allows to rewrite the expression for the worst-case error as

$$e_s^2(z)=-\prod_{j=1}^{s}\beta_j+\frac{1}{n}\sum_{k=0}^{n-1}p_{s-1}(k)\;\left(\beta_s+\gamma_s\,\omega\left(\frac{k\cdot z}{n}\right)\right),\quad\forall z\in U_n. \tag{3.5}$$

Of course only one n-vector is needed in an implementation since it can be overwritten in each iteration step. Also the function ω needs to be evaluated in exactly n points, and, depending on that function, it might pay of to precalculate this function. These simple changes result in a construction cost for Algorithm 2 of $O(sn^2)$ by using $O(n)$ memory. Moreover, as actual timing results in §3.4.1 will show, there is (almost) no reason not to store the previous n products, as the time-penalty for not doing so is significant. This is especially so for large n.

3.2.2 A matrix-vector formulation

It is not hard to see that (3.5) is in fact a matrix-vector multiplication. Define the matrix

$$\boldsymbol{\Omega}_n:=\left[\omega\left(\frac{k\cdot z}{n}\right)\right]_{\substack{z=1,\ldots,n-1\\k=0,\ldots,n-1}}\qquad(n\text{ prime}), \tag{3.6}$$

which has at most n different elements (depending on the function ω). A literal translation from (3.5) now defines the worst-case error vector

$$\boldsymbol{e}_s^2=-\bar{\beta}_s\mathbf{1}_{n-1\times 1}+\frac{1}{n}(\beta_s\mathbf{1}_{n-1\times n}+\gamma_s\,\boldsymbol{\Omega}_n)\,\boldsymbol{p}_{s-1}, \tag{3.7}$$

were $\mathbf{1}_{m\times n}$ is an $m\times n$ matrix with all entries equal to one and we use the shorthand $\bar{\beta}$ as the cumulative product of the β_j's up to dimension s, i.e.,

$$\bar{\beta}_s = \prod_{j=1}^{s} \beta_j.$$

Algorithm 3 Component-by-component construction in matrix-vector form

$\boldsymbol{p}_0 = \mathbf{1}_{n\times 1}$
for $s = 1$ **to** $s_{\max}$ **do**
  $\boldsymbol{e}_s^2 = -\bar{\beta}_s \mathbf{1}_{\varphi(n)\times 1} + n^{-1}\big(\beta_s \mathbf{1}_{\varphi(n)\times n} + \gamma_s\, \boldsymbol{\Omega}_n\big)\; \boldsymbol{p}_{s-1}$
  $z_s = \operatorname*{argmin}_{z\in U_n} e_s^2(z)$
  $\boldsymbol{p}_s = \operatorname{diag}(\beta_s \mathbf{1}_{1\times n} + \gamma_s\, \boldsymbol{\Omega}_n(z_s,:))\; \boldsymbol{p}_{s-1}$
end for

The component-by-component algorithm can now be restated in matrix-vector form as Algorithm 3. (To accommodate non-prime n in later chapters, the $n-1$, the size of U_n for n prime, is replaced by the Euler totient function $\varphi(n)$, which is the number of positive integers relatively prime to n which are less than n.)

In the matrix-vector formulation we have to update the vector $\boldsymbol{p}$ after each iteration. Looking at (3.4) it is clear that every element $p_{s-1}(k)$ must be multiplied with $\beta_s + \gamma_s\, \omega((z_s \cdot k)/n)$ to obtain the new element $p_s(k)$:

$$p_s(k) = \left(\beta_s + \gamma_s\, \omega\left(\frac{z_s \cdot k}{n}\right)\right)\; p_{s-1}(k), \qquad \text{for all } k.$$

If we set

$$\boldsymbol{K}_{n,\gamma,\beta} := \beta \mathbf{1}_{n-1\times n} + \gamma\, \boldsymbol{\Omega}_n,$$

then this is the same as elementwise multiplication with row z_s of $\boldsymbol{K}_{n,\gamma_s,\beta_s}$. Let $\operatorname{diag}(\boldsymbol{x})$ denote the diagonal matrix with the elements of $\boldsymbol{x}$ on its diagonal and zero elsewhere; and let $\boldsymbol{v}_j$ denote a selection vector with 1 in position j and zero elsewhere. Then

$$\begin{aligned} \boldsymbol{p}_s &= \operatorname{diag}\left(\boldsymbol{v}_{z_s}^\top\, \boldsymbol{K}_{n,\gamma_s,\beta_s}\right)\; \boldsymbol{p}_{s-1} \\ &= \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s\, \boldsymbol{v}_{z_s}^\top\, \boldsymbol{\Omega}_n\right)\; \boldsymbol{p}_{s-1}. \end{aligned} \tag{3.8}$$

When convenient a Matlab-like notation will be used to select the j-th row of a matrix as

$$\boldsymbol{\Omega}_n(j,:) = \boldsymbol{v}_j^\top\, \boldsymbol{\Omega}_n.$$

Algorithm 3 is only a trivial rewrite of Algorithm 2. From a computational point of view this rewrite makes clear that the total construction cost of $O(sn^2)$ could be reduced if we could find a matrix-vector multiplication algorithm for the given matrix $\boldsymbol{\Omega}_n$ with cost less than $O(n^2)$.

3.2.3 The structure of $\boldsymbol{\Omega}_n$

It is clear that $\omega(x)$ is only evaluated in n different points, and so these can be calculated beforehand. Define an n-vector $\boldsymbol{\omega}$ and set

$$\omega_\ell := \omega(\ell/n) \qquad \text{for} \quad \ell = 0, \ldots, n-1.$$

When the value of $\omega((k \cdot z)/n)$ is needed, it is sufficient to take element $k \cdot z \pmod n$ of the $\boldsymbol{\omega}$ vector, see (3.1), which we will write as

$$\omega_{z \cdot k} := \omega\left(\frac{k \cdot \boldsymbol{z}}{n}\right). \tag{3.9}$$

By using this notation we can clearly demonstrate the structure of the matrix $\boldsymbol{\Omega}_n$ defined in (3.6):

$$\begin{aligned} \boldsymbol{\Omega}_n &= \begin{bmatrix} \omega_{1\cdot 0} & \omega_{1\cdot 1} & \omega_{1\cdot 2} & \cdots & \omega_{1\cdot(n-1)} \\ \omega_{2\cdot 0} & \omega_{2\cdot 1} & \omega_{2\cdot 2} & \cdots & \omega_{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_{(n-1)\cdot 0} & \omega_{(n-1)\cdot 1} & \omega_{(n-1)\cdot 2} & \cdots & \omega_{(n-1)\cdot(n-1)} \end{bmatrix} \\ &= \begin{bmatrix} \omega_0 & \omega_1 & \omega_2 & \cdots & \omega_{n-1} \\ \omega_0 & \omega_2 & \omega_4 & \cdots & \omega_{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \omega_0 & \omega_{n-1} & \omega_{n-2} & \cdots & \omega_1 \end{bmatrix}. \end{aligned} \tag{3.10}$$

Obviously the matrix $\boldsymbol{K}_{n,\gamma,\beta}$ has the same structure, but it may change in every iteration due to different γ and β, whereas the matrix $\boldsymbol{\Omega}_n$ is completely defined by the variable kernel part ω and the number of points n. We will therefore focus on $\boldsymbol{\Omega}_n$, which has the appealing visual fractal-like representation of Figure 3.1.

First we define the class of circulant matrices [20]. Given a column vector $\boldsymbol{c}$ of length m, define

$$\boldsymbol{C}_m = \operatorname{circ}(\boldsymbol{c}) := \begin{bmatrix} c_0 & c_{m-1} & c_{m-2} & \cdots & c_1 \\ c_1 & c_0 & c_{m-1} & \cdots & c_2 \\ c_2 & c_1 & c_0 & \cdots & c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{m-1} & c_{m-2} & c_{m-3} & \cdots & c_0 \end{bmatrix}.$$

The following definition introduces the concept more formally.

Definition 3.1. A *circulant matrix* $\boldsymbol{C}_m = \operatorname{circ}(\boldsymbol{c})$ of order m is a matrix defined by the m elements in the vector $\boldsymbol{c}$, indexed from 0, as

$$\big[\boldsymbol{C}_m\big]_{\substack{k=0,\ldots,m-1 \\ \ell=0,\ldots,m-1}} = c_{k-\ell \bmod m}.$$

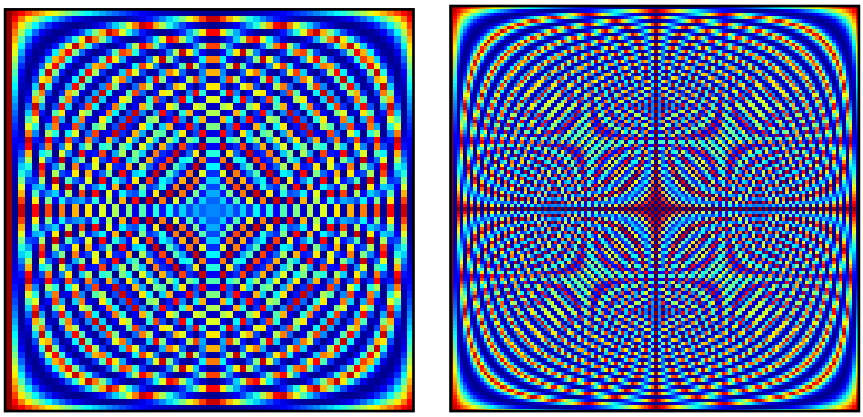


Figure 3.1: The structure of $\boldsymbol{\Omega}_n$ for $n = 61$ and the same structure for a matrix with $n = 122$ (using a Korobov kernel with $\alpha = 2$).

A matrix-vector multiplication of a vector $\boldsymbol{x}$ with a circulant matrix $\boldsymbol{C}_m = \mathrm{circ}(\boldsymbol{c})$, is equivalent to the circular convolution of $\boldsymbol{x}$ and $\boldsymbol{c}$,

$$\boldsymbol{C}_m \, \boldsymbol{x} = \boldsymbol{c} * \boldsymbol{x}.$$

It is well known that a circulant matrix has a similarity transform which has the Fourier matrix as its eigenvectors, see, e.g., [36].

Theorem 3.2. *A circulant matrix $\boldsymbol{C}_m$ of order m has a similarity transform*

$$\boldsymbol{C}_m = \boldsymbol{F}_m^{-1} \, \boldsymbol{D} \, \boldsymbol{F}_m,$$

where $\boldsymbol{F}_m$ is the Fourier matrix of order m and the eigenvalues are defined by the discrete Fourier transform of the first column $\boldsymbol{c}$ of $\boldsymbol{C}_m$, so that

$$\boldsymbol{D} = \mathrm{diag}(\boldsymbol{F}_m \, \boldsymbol{c}).$$

(Note that for this theorem to hold the following often used convention for the normalization of the discrete Fourier transform and its inverse must be used

$$\tilde{x}_h = \sum_{k=0}^{n-1} x_k \, \exp(-2\pi \mathrm{i} \, hk/n), \qquad x_k = \frac{1}{n} \sum_{h=0}^{n-1} \tilde{x}_h \, \exp(2\pi \mathrm{i} \, hk/n),$$

where the sequence $\{\tilde{x}_h\}_h$ is the discrete Fourier transform of $\{x_k\}_k$.)

Using fast Fourier transforms (FFT) a matrix-vector product with a circulant matrix can then be done in $O(n \log(n))$ (see, e.g., [36, p. 201]).

Corollary 3.3. *A matrix-vector multiplication of a vector $\boldsymbol{x}$ with a circulant matrix $\boldsymbol{C}_m$ can be done in $O(m\log(m))$ by using the similarity transform of $\boldsymbol{C}_m$,*

$$\boldsymbol{C}_m\,\boldsymbol{x} = \boldsymbol{F}_m^{-1}\,\operatorname{diag}(\boldsymbol{F}_m\,\boldsymbol{c})\,\boldsymbol{F}_m\,\boldsymbol{x},$$

where each discrete Fourier transform $\boldsymbol{F}_m$ (as well as its inverse $\boldsymbol{F}_m^{-1}$) is done using an FFT (or IFFT).

Note that for each m an m-point FFT (as well as an IFFT) is possible in time $O(m\log(m))$ (even for prime m). A possible implementation is [32], which is also used by Matlab.

In what follows we will write $\boldsymbol{x}'$ when we need the vector $\boldsymbol{x}$ without its first entry (the zeroth entry). Similarly, given a matrix $\boldsymbol{A}_n$, with

$$\boldsymbol{A}_n = \Big[a_{\ell,k}\Big]_{\substack{\ell=0,\ldots,n-1\\ k=0,\ldots,n-1}},$$

we denote with $\boldsymbol{A}'_n$ the submatrix

$$\boldsymbol{A}'_n = \Big[a_{\ell,k}\Big]_{\substack{\ell=1,\ldots,n-1\\ k=1,\ldots,n-1}},$$

i.e., $\boldsymbol{A}_n$ without its first row and column.

We first look closer at a matrix $\boldsymbol{M}_n$, which looks a lot like the matrix $\boldsymbol{\Omega}_n$, but is a square $n\times n$-matrix with the following structure:

$$\boldsymbol{M}_n := \begin{bmatrix} m_0 & m_0 & m_0 & \cdots & m_0\\ m_0 & m_1 & m_2 & \cdots & m_{n-1}\\ m_0 & m_2 & m_4 & \cdots & m_{n-2}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ m_0 & m_{n-1} & m_{n-2} & \cdots & m_1 \end{bmatrix}.$$

This matrix could be concisely defined, with the notation introduced in (3.9), as

$$\boldsymbol{M}_n = \Big[m_{\ell,k}\Big] = \Big[m_{\ell\cdot k}\Big]_{\substack{\ell=0,\ldots,n-1\\ k=0,\ldots,n-1}}. \tag{3.11}$$

The matrix $\boldsymbol{M}_n$ is fully specified by its second column $\boldsymbol{m}$ and in fact has the structure of a Fourier matrix but here the entries $m_{\ell,k}$ are arbitrary. For this kind of structured matrices a factorization for discrete Fourier matrices by the name of Rader factorization [87, 107] can be generalized. This factorization is based on the following number-theoretic permutation (for n prime):

$$\boldsymbol{z} = \boldsymbol{\Pi}_g^{\top}\,\boldsymbol{x} \quad\Leftrightarrow\quad z_\ell = x_k,\quad \begin{cases} k=\ell & \text{if}\quad \ell=0,\\ k=g^{\ell-1} \bmod n & \text{if}\quad \ell=1,\ldots,n-1, \end{cases} \tag{3.12}$$

where g is a primitive root of n, and so g is a generator of the cyclic group of order $n-1$ with elements $g^\alpha \bmod n$, $\alpha = 0, 1, \ldots, n-2$.

The Rader factorization [87, 107] of such a matrix is given in the next theorem.

Theorem 3.4. *Given a matrix $\boldsymbol{M}_n$, of order n, with structure as in* (3.11) *and given a primitive root g of n, the matrix $\boldsymbol{M}_n$ can be factored as*

$$\boldsymbol{M}_n = \boldsymbol{\Pi}_g \begin{bmatrix} m_0 & m_0 \mathbf{1}_{1\times n-1} \\ m_0 \mathbf{1}_{n-1\times 1} & \boldsymbol{C}_{n-1} \end{bmatrix} \boldsymbol{\Pi}_{g^{-1}}^\top.$$

Here the permutations $\boldsymbol{\Pi}_g$ and $\boldsymbol{\Pi}_{g^{-1}}$ are defined as in (3.12)*, where g^{-1} is the multiplicative inverse of g modulo n, the matrix $\boldsymbol{C}_{n-1}$ is a circulant matrix of order $n-1$ defined by*

$$\boldsymbol{C}_{n-1} = \operatorname{circ}(\boldsymbol{c}'), \boldsymbol{c} \qquad := \boldsymbol{\Pi}_g^\top \, \boldsymbol{m},$$

and $\mathbf{1}_{n-1\times 1}$ is a one-vector of size $n-1$.

Proof. Since row $\ell = 0$ and column $k = 0$ get mapped onto itself, we only have to prove that

$$\begin{aligned} \Big[\boldsymbol{C}_{n-1}\Big]_{\substack{\ell=1,\ldots,n-1 \\ k=1,\ldots,n-1}} &= \boldsymbol{\Pi}_g^{'\top} \, \boldsymbol{M}'_n \, \boldsymbol{\Pi}'_{g^{-1}} \\ &= \Big[\boldsymbol{M}'_n\Big]_{g^{\ell-1},(g^{-1})^{k-1}}, \end{aligned}$$

and since the elements of $\boldsymbol{M}_n$ are indexed as $i \cdot j \pmod n$ we have

$$g^{\ell-1} \, (g^{-1})^{k-1} \equiv g^{\ell-k} \pmod n.$$

This is a matrix of which the first column, $\ell = 1, \ldots, n-1$ and $k = 1$ is given by

$$\boldsymbol{\Pi}_g^{'\top} \, \boldsymbol{m}' = \boldsymbol{c}',$$

as stated in the theorem. To prove this matrix is circulant we note that all diagonal elements are the same, and we are only left to prove that the elements above the main diagonal, i.e., $k > \ell$, match their corresponding element from the first column,

$$\begin{aligned} g^{n-k+\ell-1} &\equiv g^{n-1} g^{\ell-k} \pmod n \\ &\equiv g^{\ell-k} \pmod n, \end{aligned}$$

where we used the fact that $g^{n-1} \equiv 1$, since g is a primitive root of n. □

Note that since n is prime, a primitive root of n which generates $\{1, \ldots, n-1\}$ always exists, and thus the factorization from Theorem 3.4 is always possible.

Since the matrix $\boldsymbol{\Omega}_n$ is just like the matrix $\boldsymbol{M}_n$ without its first row, see (3.6) and (3.11), we are able to obtain a similar factorization.

Corollary 3.5. *Given an* $n-1\times n$ *matrix* $\boldsymbol{\Omega}_n$ *with structure as in* (3.6) *and given a primitive root* g *of* n*, the matrix* $\boldsymbol{\Omega}_n$ *can be factored as*

$$\boldsymbol{\Omega}_n = \boldsymbol{\Pi}'_g \begin{bmatrix} \omega_0 \mathbf{1}_{n-1\times 1} & \boldsymbol{C}_{n-1} \end{bmatrix} \boldsymbol{\Pi}^{\top}_{g^{-1}}.$$

Here the permutations $\boldsymbol{\Pi}_g$ *and* $\boldsymbol{\Pi}_{g^{-1}}$ *are defined as in* (3.12)*, where* g^{-1} *is the multiplicative inverse of* g *modulo* n*, and the matrix* $\boldsymbol{C}_{n-1}$ *is a circulant matrix of order* $n-1$ *defined by*

$$\begin{aligned} \boldsymbol{C}_{n-1} &= \operatorname{circ}(\boldsymbol{\psi}'), \\ \boldsymbol{\psi} &:= \boldsymbol{\Pi}_g^{\top}\, \boldsymbol{\omega}, \end{aligned}$$

and $\mathbf{1}_{n-1\times 1}$ *is a one-vector of size* $n-1$*.*

Proof. This is a trivial modification of Theorem 3.4. □

In accordance with subsequent chapters it is of interest to define a special notation for the circulant form of the matrix $\boldsymbol{\Omega}_n$. For a given primitive root g of n, and n prime, define

$$\boldsymbol{\Omega}_n^{\langle g\rangle} := \begin{bmatrix} \omega_0 \mathbf{1}_{n-1\times 1} & \boldsymbol{C}_{n-1} \end{bmatrix}, \tag{3.13}$$

where the matrix $\boldsymbol{C}_{n-1}$ is the circulant matrix from the previous corollary. (Note that in later chapters the ordering of the horizontal blocks will be reversed.)

Since the kernel is symmetric, that is $\omega(x) = \omega(1-x)$, there is another computationally important structural property of the matrix $\boldsymbol{\Omega}_n$: its horizontal and vertical symmetry, and this in addition to $\boldsymbol{\Omega}_n$ already being centrosymmetric. These structural properties are clearly visible in Figure 3.1 where only the first column of $\boldsymbol{\Omega}_n$ is non-regular.

E.g., for a Korobov space with α taken as an even integer, the kernel is symmetric, since from (2.5)

$$B_\alpha(1-x) = (-1)^\alpha B_\alpha(x),$$

but also for odd α the kernel is symmetric since the sum in (2.4) can be written with cosines instead of exponentials since for each summand h we also have $-h$,

$$\sum_{h\in\mathbb{Z}\setminus\{0\}} \frac{\exp(2\pi \mathrm{i}\, hx)}{|h|^\alpha} = 2\sum_{h=1}^{\infty} \frac{\cos(2\pi hx)}{h^\alpha}.$$

(For the scalar lattice rules a shift-invariant kernel is in fact always symmetric, see Lemma 2.12; but for polynomial lattice rules, see Chapter 6, this is not the case.)

Due to this symmetry we only have to search half of the possible z values, and we have $e_s^2(n-z) = e_s^2(z)$ and $p_s(n-k) = p_s(k)$. As a consequence we can halve

the number of calculations per z value in (3.7), winning a constant factor of 4 in total.

If we would transform the matrix $\boldsymbol{\Omega}_n$ into the more computationally suitable form $\boldsymbol{\Omega}_n^{\langle g \rangle}$, then it would be nice to keep all of the advantages of its symmetries. We now show that such horizontal and vertical symmetry results in a periodic form when using permutations of the form (3.12) as used in Corollary 3.5.

Theorem 3.6. *Given a vector $\boldsymbol{w}$ of length $n-1$, which has symmetry $w_k = w_{n-k}$, and g a primitive root of n, we obtain a periodic vector $\boldsymbol{s}$ after permutation* (3.12)*,*

$$\boldsymbol{s} = \boldsymbol{\Pi}_g^{'\top}\, \boldsymbol{w} = \begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{t} \end{bmatrix},$$

with two copies of a vector $\boldsymbol{t}$ with length $m = (n-1)/2$.

Likewise, given a matrix $\boldsymbol{M}'_n$ of order $n-1$, with horizontal and vertical symmetry, and g a primitive root of n, after permutation (3.12)*,*

$$\boldsymbol{C}_{n-1} = \boldsymbol{\Pi}_g^{'\top}\, \boldsymbol{G}'_n\, \boldsymbol{\Pi}'_{g^{-1}} = \begin{bmatrix} \boldsymbol{C}_m & \boldsymbol{C}_m \\ \boldsymbol{C}_m & \boldsymbol{C}_m \end{bmatrix}, \tag{3.14}$$

we obtain a horizontally and vertically periodic matrix $\boldsymbol{C}_{n-1}$ with four copies of a circulant matrix $\boldsymbol{C}_m$ of order $m = (n-1)/2$.

Proof. Without loss of generality we only prove the periodicity for the vector case. It is also obvious that $\boldsymbol{C}_m$ is circulant if $\boldsymbol{C}_{n-1}$ is circulant, which is proven in Theorem 3.4.

The periodicity in the vector $\boldsymbol{s}$ can be expressed as

$$s_\ell = s_{\ell+(n-1)/2},$$

where these indices are generated by permutation (3.12) from a source index k, and we should prove that:

$$\begin{cases} g^{\ell-1} \equiv k, \\ g^{\ell+(n-1)/2-1} \equiv n-k. \end{cases}$$

The first part is true by definition (3.12). Using the fact that $g^{(n-1)/2} \equiv n-1$ if g is a primitive root of n, we find

$$\begin{aligned} g^{\ell+(n-1)/2-1} &\equiv g^{\ell-1}\; g^{(n-1)/2} \\ &\equiv k\;(n-1) \\ &\equiv n-k \end{aligned}$$

modulo n, completing the proof. □

3.2.4 A fast algorithm

The results in the previous section now lead to the main result.

Theorem 3.7. *Component-by-component construction of an n-point rank-1 lattice rule in s dimensions, with n prime, for a shift-invariant tensor-product reproducing kernel Hilbert space with product weights, can be done in time $O(sn\log(n))$ and memory $O(n)$.*

Proof. The proof just puts all the pieces from the previous section together.

With the result of Corollary 3.5 we can work out the matrix-vector multiplication for the calculation of the worst-case error vector. From (3.7) we find

$$\begin{aligned}\boldsymbol{e}_s^2 &= -\bar{\beta}_s \mathbf{1}_{n-1\times 1} + \frac{1}{n}(\beta_s \mathbf{1}_{n-1\times n} + \gamma_s\, \boldsymbol{\Omega}_n)\, \boldsymbol{p}_{s-1} \\ &= \left(-\bar{\beta}_s + \frac{\beta_s}{n}\sum_{k=0}^{n-1} p_{s-1}(k)\right)\mathbf{1}_{n-1\times 1} + \frac{\gamma_s}{n}\,\boldsymbol{\Omega}_n\,\boldsymbol{p}_{s-1}.\end{aligned}$$

Applying Corollary 3.5 with $\boldsymbol{\psi} = \boldsymbol{\Pi}_g^\top\,\boldsymbol{\omega}$ we immediately obtain

$$\begin{aligned}\boldsymbol{\Omega}_n\,\boldsymbol{p}_{s-1} &= \boldsymbol{\Pi}'_g\,\boldsymbol{\Omega}_n^{\langle g\rangle}\,\boldsymbol{\Pi}_{g^{-1}}^\top\,\boldsymbol{p}_{s-1} \\ &= \boldsymbol{\Pi}'_g \begin{bmatrix} \psi_0\mathbf{1}_{n-1\times 1} & \operatorname{circ}(\boldsymbol{\psi}') \end{bmatrix} \boldsymbol{\Pi}_{g^{-1}}^\top\,\boldsymbol{p}_{s-1}.\end{aligned}$$

Now set $\boldsymbol{q}_{s-1} := \boldsymbol{\Pi}_{g^{-1}}^\top\,\boldsymbol{p}_{s-1}$ and expand the matrix-vector product to get

$$\boldsymbol{\Omega}_n\,\boldsymbol{p}_{s-1} = \boldsymbol{\Pi}'_g\,\left(\psi_0\, q_{s-1}(0)\,\mathbf{1}_{n-1\times 1} + \operatorname{circ}(\boldsymbol{\psi}')\,\boldsymbol{q}'_{s-1}\right).$$

Note that $\boldsymbol{\psi}'$ is just given by

$$\boldsymbol{\psi}' = \left[\omega((g^{\ell-1} \bmod n)/n)\right]_{\ell=1,\ldots,n-1}. \tag{3.15}$$

The squared worst-case error vector can now be calculated in the permuted space as

$$\begin{aligned}\boldsymbol{E}_s^2 &:= \boldsymbol{\Pi}_g^{'\top}\,\boldsymbol{e}_s^2 \\ &= \left(-\bar{\beta}_s + \frac{\beta_s}{n}\sum_{k=0}^{n-1} q_{s-1}(k) + \frac{\gamma_s}{n}\,\psi_0\, q_{s-1}(0)\right)\mathbf{1}_{n-1\times 1} + \frac{\gamma_s}{n}\,\boldsymbol{\psi}' * \boldsymbol{q}'_{s-1} \qquad (3.16)\\ &= -\bar{\beta}_s\mathbf{1}_{n-1\times 1} + \frac{1}{n}(\beta_s\mathbf{1}_{n-1\times n} + \gamma_s\,\boldsymbol{\Omega}_n^{\langle g\rangle})\,\boldsymbol{q}_{s-1}.\end{aligned}$$

Formula (3.16) can be calculated in $O(n\log(n))$ by using FFTs of size $n-1$ for the convolution as explained in Corollary 3.3.

Searching the minimum can best be done in the permuted space. Call the permuted minimum

$$w_s = \operatorname*{argmin}_{w \in U_n} E_s^2(w).$$

The unpermuted component z_s can be found by mapping w_s back using the inverse mapping of $\boldsymbol{\Pi}_g^{'\top}$ which is just $\boldsymbol{\Pi}_g'$. Thus

$$z_s \equiv g^{w_s - 1}.$$

Keeping this mapping in memory is $O(n)$, and this is the same ordering that is needed to construct the vector $\boldsymbol{\psi}$.

An update rule similar to (3.8) is needed for $\boldsymbol{q}_s$. We have

$$\begin{aligned}
\boldsymbol{q}_s &= \boldsymbol{\Pi}_{g^{-1}}^\top \, \boldsymbol{p}_s \\
&= \boldsymbol{\Pi}_{g^{-1}}^\top \, \operatorname{diag}\left(\boldsymbol{v}_{z_s}^\top \, \boldsymbol{K}_{n,\gamma_s,\beta_s}\right) \, \boldsymbol{p}_{s-1} \\
&= \boldsymbol{\Pi}_{g^{-1}}^\top \, \operatorname{diag}\left(\boldsymbol{v}_{z_s}^\top \, \boldsymbol{K}_{n,\gamma_s,\beta_s}\right) \, \boldsymbol{\Pi}_{g^{-1}} \, \boldsymbol{q}_{s-1}.
\end{aligned}$$

Observe that we have the same permutation on the rows and columns of a diagonal matrix. This can be done by applying this same permutation to the vector that defines the diagonal and thus

$$\begin{aligned}
\boldsymbol{q}_s &= \operatorname{diag}\left(\boldsymbol{v}_{z_s}^\top \, \boldsymbol{K}_{n,\gamma_s,\beta_s} \, \boldsymbol{\Pi}_{g^{-1}}\right) \, \boldsymbol{q}_{s-1} \\
&= \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s \, \boldsymbol{v}_{z_s}^\top \, \boldsymbol{\Omega}_n \, \boldsymbol{\Pi}_{g^{-1}}\right) \, \boldsymbol{q}_{s-1}.
\end{aligned}$$

Using Corollary 3.5 we finally obtain

$$\begin{aligned}
\boldsymbol{q}_s &= \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s \, (\boldsymbol{v}_{z_s}^\top \, \boldsymbol{\Pi}_g') \begin{bmatrix} \psi_0 \mathbf{1}_{n-1} & \operatorname{circ}(\boldsymbol{\psi}') \end{bmatrix}\right) \, \boldsymbol{q}_{s-1} \\
&= \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s \, \boldsymbol{v}_{w_s}^\top \begin{bmatrix} \psi_0 \mathbf{1}_{n-1} & \operatorname{circ}(\boldsymbol{\psi}') \end{bmatrix}\right) \, \boldsymbol{q}_{s-1} \\
&= \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s \, \boldsymbol{v}_{w_s}^\top \, \boldsymbol{\Omega}_n^{\langle g \rangle}\right) \, \boldsymbol{q}_{s-1}. \qquad (3.17)
\end{aligned}$$

From this it is clear that we do not have to permute hence and forth and the vector $\boldsymbol{q}$ can be updated in time $O(n)$ (multiplication with a diagonal matrix). □

The update rule (3.17) for $\boldsymbol{q}$ is very similar to the update rule for $\boldsymbol{p}$ in (3.8). We recombine the old $\boldsymbol{q}'$ vector, elementwise with the row of the circulant matrix $\operatorname{circ}(\boldsymbol{\psi}')$ which creates the minimal value in the permuted worst-case error $\boldsymbol{E}_s^2$. It can be seen that the value for $q_s(0)$, which is more or less treated separately, is independent of the choice for z_s, for all s.

The fast variant is given in Algorithm 4. An example implementation of this algorithm is given in the next section.

Algorithm 4 Fast component-by-component construction for n prime

$\boldsymbol{q}_0 = \mathbf{1}_{n\times 1}$
for $s = 1$ **to** $s_{\max}$ **do**
  $\boldsymbol{E}_s^2 = -\bar{\beta}_s \mathbf{1}_{\varphi(n)\times 1} + n^{-1}(\beta_s \mathbf{1}_{\varphi(n)\times n} + \gamma_s\, \boldsymbol{\Omega}_n^{\langle g\rangle})\, \boldsymbol{q}_{s-1}$
  $w_s = \operatorname*{argmin}_{w\in U_n} E_s^2(w)$
  $\boldsymbol{q}_s = \operatorname{diag}\left(\beta_s \mathbf{1}_{1\times n} + \gamma_s\, \boldsymbol{\Omega}_n^{\langle g\rangle}(w_s,:)\right)\, \boldsymbol{q}_{s-1}$
end for

3.2.5 A Matlab implementation

In this section a possible implementation in Matlab of the fast algorithm for scalar rank-1 lattice rules with product weights is given, see Listing 3.1. (Instead of Matlab also Octave can be used for the code presented.) The use of Matlab or any other scripting language is justified if it gives the user access to an optimized FFT (e.g., also Numerical Python, or NumPy in short, is a valuable option). Since the dominating calculation is hidden away in the FFTs, the slower execution speed of the dynamic scripting language is not really a disadvantage.

In the example implementation we use that the variable kernel function ω is symmetric around 1/2, so that $\omega(x) = \omega(1-x)$. Using this assumption only half of the $\boldsymbol{q}'$ and $\boldsymbol{\psi}'$ vectors are needed, and so the FFTs are also only half the size (see Theorem 3.6). Since ω is a real-valued function it is advisable to use real data transforms, i.e., real to complex transforms for the FFT and complex to real transforms for the IFFT (possibly with a "halfcomplex" format). This results in an extra halving of the memory consumption and faster execution.

Matlab's FFT is however always complex to complex. In addition to this extra memory consumption, Matlab passes variables by value due to which the memory need for this script is again at least doubled. One could of course pull this code to a higher level, e.g., by abstracting out the $\boldsymbol{\Omega}_n$ matrix in a class, but the purpose of the example implementation is to show how it is done and trying not to incur an extra memory footprint by argument passing. (In Numerical Python both of these problems, complex FFTs and pass by value, could be avoided.) In total one must assume a fourfold memory consumption than what would be possible by doing, e.g., a C implementation.

It is not necessary to calculate the exact squared worst-case error vector in the permuted space to find the minimum. Just the result of the circulant matrix-vector product is enough, call this $\tilde{\boldsymbol{E}}^2$. In theory the vector $\tilde{\boldsymbol{E}}^2$ should be constant in the first iteration. This means that we can always pick $z_1 = w_1 = 1$. In practice there will be numerical noise and so this decision is forced. Calculating this first iteration gives some information about the noise level for the FFT, but this information is not used in the example implementation.

Finding a primitive root of n is handled in the `generatorp` function, see

Listing 3.1: Fast construction with product weights (`fastrank1pt.m`)

```
function [z, e2] = fastrank1pt(n, s_max, omega, gamma, beta)

if ~isprime(n), error('n must be prime'); end;
z = zeros(s_max, 1);
e2 = zeros(s_max, 1);

m = (n-1)/2;          % assume the ω function symmetric around 1/2
E2 = zeros(m, 1);     % the vector Ẽ^2 in the text
cumbeta = cumprod(beta);

g = generatorp(n);    % generator g for {1,2,...,n-1}
perm = zeros(m, 1);   % permutation formed by positive powers of g
perm(1) = 1; for j=1:m-1, perm(j+1) = mod(perm(j)*g, n); end;
perm = min(n - perm, perm); % map everything back to [1,n/2)
psi = omega(perm/n); % the vector ψ'
psi0 = omega(0);      % zero index: ψ(0)
fft_psi = fft(psi);

q = ones(m, 1);       % permuted product vector q' (without zero index)
q0 = 1;               % zero index of permuted product vector: q(0)

for s = 1:s_max
    % step 2a: circulant matrix−vector multiplication
    E2 = ifft(fft_psi .* fft(q));
    E2 = real(E2); % remove imaginary rounding errors
    % step 2b: choose w_s and z_s which give minimal value
    [min_E2, w] = min(E2); % pick index of minimal value
    if s == 1, w = 1; noise = abs(E2(1) - min_E2), end;
    z(s) = perm(w);
    % extra: we want to know the exact value of the worst−case error
    e2(s) = -cumbeta(s) + ( beta(s) * (q0 + 2*sum(q)) + ...
              gamma(s) * (psi0*q0 + 2*min_E2) ) / n;
    % step 2c: update q
    q = (beta(s) + gamma(s) * psi([w:-1:1 m:-1:w+1])) .* q;
    q0 = (beta(s) + gamma(s) * psi0) * q0;
    fprintf('s=%4d, z=%6d, w=%6d, e2=%.4e, e=%.4e\n', ...
            s, z(s), w, e2(s), sqrt(e2(s)));
end;
```

Listing 3.2: Finding a primitive root of prime n (`generatorp.m`)

```
function g = generatorp(p)

if ~isprime(p), error('p is not a prime'); end;

primef = unique(factor(p-1));
g = 2; i = 1;
while i <= length(primef)
    if powmod(g, (p-1)/primef(i), p) == 1
        g = g + 1; i = 0;
    end;
    i = i + 1;
end;
```

Listing 3.3: Modular exponentiation with the Russian peasant method (`powmod.m`)

```
function y = powmod(x, a, n)

if ~usejava('jvm')
    y = 1; u = x;
    while a > 0
        if mod(a, 2) == 1
            y = mod(y * u, n); % this could overflow
        end;
        u = mod(u * u, n); % this could overflow
        a = floor(a / 2);
    end;
else
    x_ = java.math.BigInteger(num2str(x));
    a_ = java.math.BigInteger(num2str(a));
    n_ = java.math.BigInteger(num2str(n));
    y = x_.modPow(a_, n_).doubleValue();
end;
```

Listing 3.2. This algorithm is described in [11, p. 25]. The implementation shown only finds a generator for n prime. The auxiliary routine `powmod` calculates $x^a \pmod n$ using the well known "Russian peasant method", see, e.g., [57, 11], and can be found in Listing 3.3. This method will make use of the Java interface in Matlab if available to allow for a larger range of input values.

We give an example usage of the routine for constructing a lattice rule for the randomly shifted anchored Sobolev space setting:

```
omega = @(x) x.^2 - x + 1/6;
n = 4001; s = 100; gamma = 0.9.^[1:s];
[z, e2] = fastrank1pt(n, s, omega, gamma, 1+gamma/3);
```

Which generates the following output on the screen in a fraction of a second:

```
noise =
     0
s=   1, z=     1, w=     1, e2=9.3703e-09, e=9.6800e-05
s=   2, z=  1478, w=   438, e2=4.9156e-08, e=2.2171e-04
s=   3, z=   823, w=   838, e2=2.0098e-07, e=4.4831e-04
s=   4, z=  1769, w=  1061, e2=6.3177e-07, e=7.9484e-04
s=   5, z=   555, w=   175, e2=1.7420e-06, e=1.3198e-03
s=   6, z=   527, w=   214, e2=3.9608e-06, e=1.9902e-03
s=   7, z=   901, w=  1327, e2=7.6585e-06, e=2.7674e-03
s=   8, z=  1128, w=  1760, e2=1.3661e-05, e=3.6961e-03
s=   9, z=  1065, w=  1038, e2=2.2958e-05, e=4.7914e-03
s=  10, z=  1559, w=   512, e2=3.5490e-05, e=5.9573e-03
```

... 90 more lines ...

Similarly, for an unweighted unanchored Sobolev space and $n = 514229$ (a Fibonacci prime):

```
omega = @(x) x.^2 - x + 1/6;
n = 514229; s = 10;
[z, e2] = fastrank1pt(n, s, omega, ones(s, 1), ones(s, 1));
```

This generates the following result:

```
noise =
   2.1205e-14
s=   1, z=     1, w=     1, e2=6.3016e-13, e=7.9383e-07
s=   2, z=196418, w=128558, e2=4.4236e-12, e=2.1032e-06
s=   3, z= 56428, w=235152, e2=3.2419e-11, e=5.6938e-06
s=   4, z= 94966, w= 64386, e2=1.8017e-10, e=1.3423e-05
s=   5, z= 53423, w=105093, e2=7.2092e-10, e=2.6850e-05
s=   6, z=236245, w=123041, e2=2.6179e-09, e=5.1165e-05
s=   7, z=200441, w=237318, e2=7.0158e-09, e=8.3760e-05
s=   8, z=246494, w=249445, e2=1.7048e-08, e=1.3057e-04
s=   9, z= 59817, w= 53508, e2=3.6647e-08, e=1.9143e-04
s=  10, z= 23043, w=128823, e2=7.1632e-08, e=2.6764e-04
```

Note that the two previous Fibonacci primes are 317811 and 196418, and the rule is thus a Fibonacci lattice rule in the first two dimensions.

3.3 Construction for order dependent weights

From §2.4.3 the expression for the squared worst-case error for order dependent weights, which may be of finite order $q^\star \leq s$, is given by

$$e_s^2(\boldsymbol{z}) = \frac{1}{n} \sum_{k=0}^{n-1} \sum_{\ell=1}^{q^\star} \Gamma_\ell \sum_{\substack{\mathfrak{u} \subseteq \mathcal{D}_s \\ |\mathfrak{u}|=\ell}} \prod_{j \in \mathfrak{u}} \omega\left(\frac{k \cdot z_j}{n}\right),$$

when $\int_0^1 \omega(x)\,\mathrm{d}(x) = 0$. In this setting the weights Γ_ℓ denote the importance of groups of ℓ variables. Also this expression can be rewritten in a form that is more suitable for recursion with respect to s. By simple formula manipulation we find the iterative formula

$$\begin{aligned} e_s^2(z_s) &= \frac{1}{n} \sum_{k=0}^{n-1} \sum_{\ell=1}^{\min(q^\star,s)} \Gamma_\ell \Bigg(\underbrace{\sum_{\substack{\mathfrak{u} \subseteq \mathcal{D}_{s-1} \\ |\mathfrak{u}|=\ell}} \prod_{j \in \mathfrak{u}} \omega\left(\frac{k \cdot z_j}{n}\right)}_{p_{s-1,\ell}(k)} \\ &\qquad + \omega\left(\frac{k \cdot z_s}{n}\right) \underbrace{\sum_{\substack{\mathfrak{u} \subseteq \mathcal{D}_{s-1} \\ |\mathfrak{u}|=\ell-1}} \prod_{j \in \mathfrak{u}} \omega\left(\frac{k \cdot z_j}{n}\right)}_{p_{s-1,\ell-1}(k)} \Bigg) \\ &= e_{s-1}^2 + \frac{1}{n} \sum_{k=0}^{n-1} \omega\left(\frac{k \cdot z_s}{n}\right) \left(\sum_{\ell=1}^{\min(q^\star,s)} \Gamma_\ell \, p_{s-1,\ell-1}(k) \right). \end{aligned}$$

Here e_{s-1}^2 is the squared worst-case error for the rule in $s-1$ dimensions, $e_0^2 := 0$, and the products $p_{s,\ell}(k)$ are defined recursively by

$$p_{s,\ell}(k) := \begin{cases} 1 & \text{if } \ell = 0, \\ p_{s-1,\ell}(k) \; + \; \omega\left(\frac{k \cdot z_s}{n}\right) \, p_{s-1,\ell-1}(k) & \text{if } \ell \leq s, \\ 0 & \text{otherwise.} \end{cases}$$

(Explicitly setting the values to zero when the condition $\ell \leq s$ is not fulfilled gets rid of the minimum function in the sum.)

The equivalent matrix-vector formulation is given by

$$\begin{aligned} \boldsymbol{e}_s^2 &:= e_{s-1}^2 \mathbf{1}_{\varphi(n)\times 1} + \frac{1}{n}\,\boldsymbol{\Omega}_n \left(\sum_{\ell=1}^{q^\star} \Gamma_\ell\, \boldsymbol{p}_{s-1,\ell-1} \right), \\ \boldsymbol{p}_{s,\ell} &:= \boldsymbol{p}_{s-1,\ell} + \operatorname{diag}(\boldsymbol{\Omega}_n(z_s,:))\, \boldsymbol{p}_{s-1,\ell-1}. \end{aligned}$$

Again, the core of the calculation is a matrix-vector product with $\boldsymbol{\Omega}_n$. In the permuted space this expands to

$$\begin{aligned} &\boldsymbol{\Omega}_n^{\langle g\rangle} \left(\sum_{\ell=1}^{q^\star} \Gamma_\ell\, \boldsymbol{q}_{s-1,\ell-1} \right) \\ &\qquad = \begin{bmatrix} \psi_0 \mathbf{1}_{n-1\times 1} & \operatorname{circ}(\boldsymbol{\psi}') \end{bmatrix} \left(\Gamma_1 \boldsymbol{q}_{s,0} + \Gamma_2 \boldsymbol{q}_{s,1} + \cdots + \Gamma_{q^\star} \boldsymbol{q}_{s,q^\star-1} \right). \end{aligned}$$

Two constant parts can be extracted: the zeroth elements of the vectors $\boldsymbol{q}$ are just multiplied by ψ_0; and, since the first vector $\boldsymbol{q}_{s,0} = \mathbf{1}_{n\times 1}$, multiplication with $\boldsymbol{q}_{s,0}$ would just sum the elements of $\boldsymbol{\psi}$ times Γ_1. Thus

$$\boldsymbol{\Omega}_n^{\langle g\rangle} \left(\sum_{\ell=1}^{q^\star} \Gamma_\ell\, \boldsymbol{q}_{s-1,\ell-1} \right) = \left(\Gamma_1 \sum_{k=0}^{n-1} \psi_k + \psi_0 t_0 \right) \mathbf{1}_{n-1\times 1} + \boldsymbol{\psi}' * \boldsymbol{t}', \tag{3.18}$$

where

$$\boldsymbol{t} := \Gamma_2 \boldsymbol{q}_{s,1} + \cdots + \Gamma_{q^\star} \boldsymbol{q}_{s,q^\star-1}.$$

For the calculation of (3.18) a cost of $O(nq^\star)$ is needed for the vector $\boldsymbol{t}$ plus $O(n\log(n))$ for the convolution. The obvious theorem follows.

Theorem 3.8. *Component-by-component construction of an n-point rank-1 lattice rule in s dimensions, with n prime, for a shift-invariant reproducing kernel Hilbert space with order dependent weights of order $q^\star$, can be done in time $O(s(n\log(n) + nq^\star))$ and memory $O(nq^\star)$.*

For $q^\star \ll s$ this complexity is comparable to the $O(sn\log(n))$ complexity of the product weights case. Also here we provide an example Matlab implementation in Listing 3.4 where again Theorem 3.6 was used to half the memory requirements.

3.4 Numerical experiments (for product weights)

3.4.1 Timings

In this section numerical results obtained by implementations of Algorithms 2, 3 and 4 are presented. The tests were run on a Linux workstation equipped with

Listing 3.4: Fast construction with order dependent weights (`fastrank1od.m`)

```
function [z, e2] = fastrank1od(n, s_max, omega, Gamma)

if ~isprime(n), error('n must be prime'); end;
z = zeros(s_max, 1);
e2 = zeros(s_max, 1);

qstar = length(Gamma);
m = (n-1)/2;
E2 = zeros(m, 1);

g = generator(n);
perm = zeros(m, 1);  % permutation formed by positive powers of g
perm(1) = 1; for j=1:m-1, perm(j+1) = mod(perm(j)*g, n); end;
perm = min(n - perm, perm);  % map everything back to [1, n/2)
psi = omega(perm/n);  % the vector psi'
psi0 = omega(0);      % zero index: psi(0)
fft_psi = fft(psi);

q = zeros(m, qstar); q(:, 1) = 1;  % q_{s,l} is given by q(:,l-1)
q0 = zeros(1, qstar); q0(1) = 1;
Gamma = reshape(Gamma, qstar, 1);  % i.e., q0*Gamma is inner product
diagGamma = spdiags(Gamma, 0, qstar, qstar);

prev_e2 = 0;
for s = 1:s_max
    E2 = real( ifft(fft_psi .* fft(sum(q*diagGamma, 2))) );
    [min_E2, w] = min(E2);  % pick index of minimal value
    if s == 1, w = 1; noise = abs(E2(1) - min_E2), end;
    z(s) = perm(w);
    e2(s) = prev_e2 + (psi0*(q0*Gamma) + 2*min_E2) / n;
    % update q
    for l = qstar:-1:2  % off by 1
        q(:,l) = q(:,l) + psi([w:-1:1 m:-1:w+1]) .* q(:,l-1);
    end;
    q0(2:qstar) = q0(2:qstar) + psi0*q0(1:qstar-1);
    prev_e2 = e2(s);
    fprintf('s=%4d, z=%6d, w=%6d, e2=%.4e, e=%.4e\n', ...
            s, z(s), w, e2(s), sqrt(e2(s)));
end;
```

algorithm	total time	memory
fastrank1	$O(sn\log(n))$	$O(2n)$
spmvrank1	$O(sn^2)$	$O(n^2/8)$
rank1	$O(sn^2)$	$O(n)$
slowrank1	$O(s^2n^2)$	–

Table 3.1: Time and memory complexity for component-by-component construction with product weights. For the memory complexity the constant has been retained.

a 2.4 GHz Pentium 4 processor with hyper-threading and 2 GB of RAM. All implementations used for timings use IEEE double precision floating point and were compiled with the same level of optimizations with the Intel® C Compiler version 7. Four versions were implemented in C:

- *fastrank1*: Algorithm 4 with construction cost $O(sn\log(n))$. The implementation uses the FFTW3 library [32] and the C99 double complex type.
- *spmvrank1*: Algorithm 3 using the optimized level 2 BLAS routine *dspmv* and having cost $O(sn^2)$. The BLAS used was the Intel® Math Kernel Library for Linux version 6.
- *rank1*: Algorithm 3 with caching of the product vector and precalculation of the $\boldsymbol{\omega}$-vector, having cost $O(sn^2)$.
- *slowrank1*: The original algorithm without caching and no precalculations, using the least possible memory and having cost $O(s^2n^2)$.

All implementations use the symmetry-trick explained in §3.2.3 to search only half the space of z candidates and to do half the number of calculations per z value. For the *fastrank1* algorithm this trick was justified in Theorem 3.6.

The purpose of *spmvrank1* was only to get an idea on how a heavily optimized version of *rank1* would do (neglecting the heavy memory bandwidth). Consequently, this routine is not of practical interest because the relevant part of the $\boldsymbol{\Omega}_n$ matrix must be kept in memory, which is $m(m+1)/2$ doubles, with $m = (n-1)/2$. Note that the extra memory needed for *spmvrank1* is just a side-effect of using the BLAS routine, and would not be needed for a fully optimized *rank1* routine (which would also need much less memory bandwidth).

An overview of the construction cost and memory consumption of all the algorithms is given in Table 3.1. The constants for the memory complexity are retained to give an approximate idea about the memory limitations of a certain algorithm. Since just counting floating point operations does not always give accurate running time predictions, we also timed the four routines for different n in 20 dimensions. The total running time is illustrated in Figure 3.2, which also gives a view on the constants for the time complexity. From the figure it can be seen that *fastrank1* is clearly the winner.

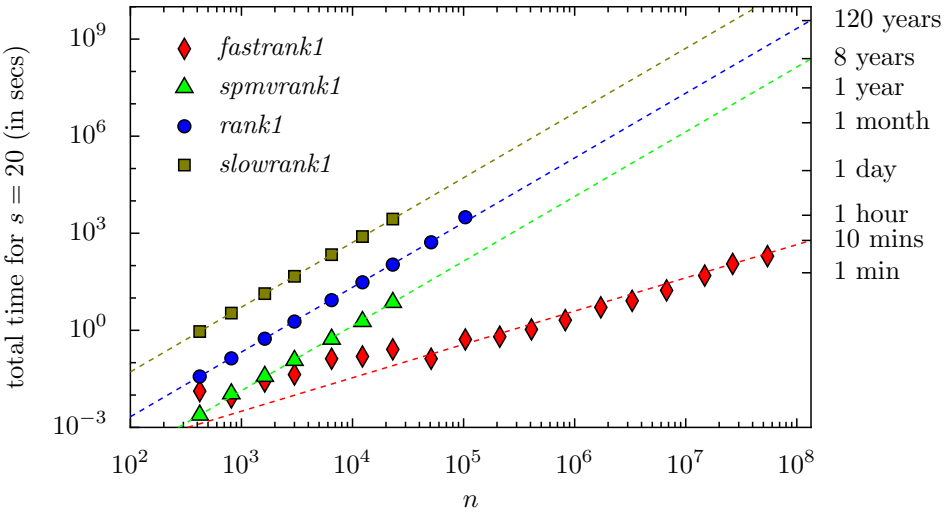


Figure 3.2: Construction times of the four routines for various point sizes in 20 dimensions. The extra axis on the right side gives a more readable view on the logarithmic time scale.

In Figure 3.3 the iteration times (i.e., the time needed for each component) for these three routines are shown. From this plot it is clear that the irregularities in the beginning of the data for *fastrank1* in Figure 3.2 can be accredited to irregularities in the setup phase for these n.

The documentation of FFTW states that the time complexity of its FFT implementation for an m point FFT is always $O(m \log(m))$ (even for prime m, for which the original Rader factorization is used and on which Theorem 3.4 is based) [32]. However the performance of the library degrades for large prime factors. Following the documentation, it is best to choose m of the form

$$m = 2^a\, 3^b\, 5^c\, 7^d\, 11^e\, 13^f,$$

with $e + f \leq 1$. The choices for n were made with the previous rule in mind. Starting from 421 and each time picking prime numbers which were approximately twice as large and having factors for $(n-1)$ of at most 7, the largest n used was 54 454 681. The importance of picking good n values is illustrated in Figure 3.4, where the iteration time for the previous five primes and the next five primes for each n were timed.

We now return to our remark about the two original algorithms: *slowrank1* (which is a straightforward implementation of Algorithm 2) and *rank1* (which is an implementation of Algorithm 3). As can be seen in Table 3.1 the memory cost for *rank1* is n doubles, being m cached products and an m-vector to represent half

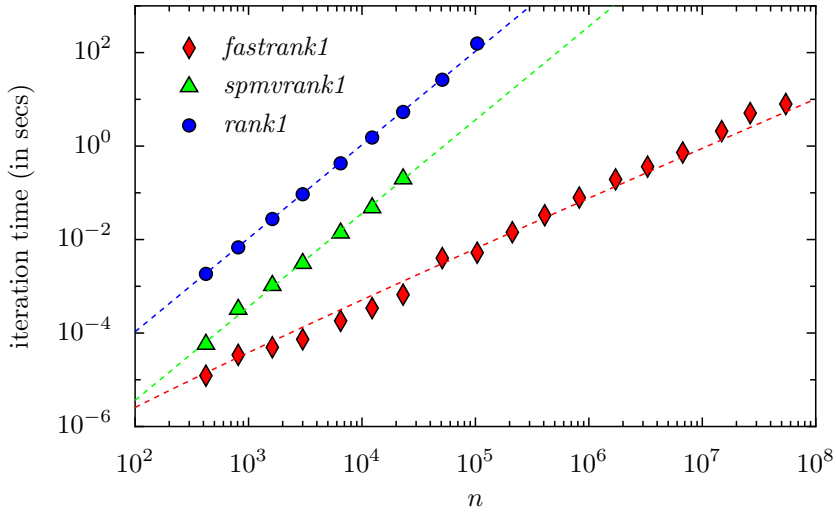


Figure 3.3: Iteration times for three of the routines.

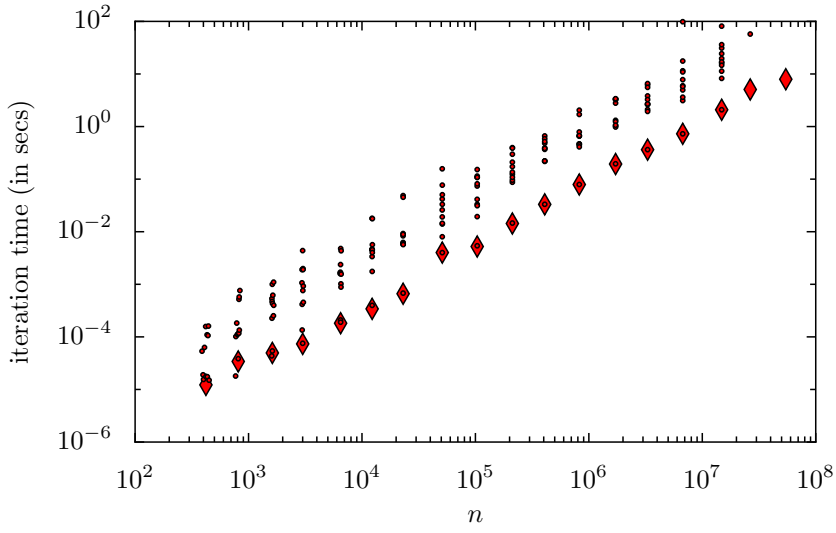


Figure 3.4: Importance of picking a good prime for the fast algorithm: the diamonds are the primes which where chosen for the experiments because they have a suitable good factorization of $n-1$ for FFTW.

n	0.9^j	0.5^j	0.1^j	j^{-1}	j^{-2}	j^{-6}
4001	2.0242e+02	9.8282e−03	1.9988e−04	1.0759e+01	3.1264e−02	6.8995e−04
8009	1.4256e+02	5.9293e−03	1.0241e−04	7.6069e+00	1.9793e−02	3.5772e−04
16001	1.0151e+02	3.5558e−03	5.1961e−05	5.3817e+00	1.2435e−02	1.8223e−04
32003	7.1876e+01	2.0631e−03	2.6526e−05	3.7939e+00	7.9071e−03	9.3695e−05
64007	5.0634e+01	1.1980e−03	1.3387e−05	2.6762e+00	4.9801e−03	4.7580e−05

Table 3.2: Worst-case errors e for a 100-dimensional weighted Korobov space with $\alpha = 2$, γ_j as specified above each column, $\beta_j = 1$ and n as specified in the first column.

of the $\boldsymbol{\omega}$-vector. Suppose we have available a modest memory size of 1 GB for storing these values; then $n = 2^{30}/2^3 = 2^{27}$. Using a least-squares fit on the data of Figure 3.2 we can estimate the time needed for $n = 2^{27} \approx 1.34 \times 10^8$ points in 20 dimensions. For our implementation of the *rank1* routine this estimate is more than 120 years. Since this time estimate makes *rank1* infeasible, even for a modest memory size of 1 GB, we find no use for *slowrank1*. Note that the time needed for *fastrank1* is only 10 minutes.

3.4.2 Comparison with published results and "Partial search"

In the appendix of [82] we listed some tables which verify the correct workings of the fast algorithm by comparing its output with some published results. Many of these results match up exactly, while some give a small difference. Two factors are involved: numerical stability and rounding errors, and choice between different z which obtain (almost) the same minima. Depending on the number of points n, but also depending on the function space, the algorithm can exhibit numerical problems. It must be noted however that these numerical problems will manifest themselves a lot sooner when using the classical $O(sn^2)$ algorithm due to the significantly larger number of operations. However, no investigations into the numerical qualities of these algorithms have been done.

In [91] an algorithm for constructing randomly shifted rank-1 lattices in a Sobolev space is presented. As explained in Chapter 2, a shift-invariant kernel can be associated with the kernel of an anchored Sobolev space, satisfying the form and structure for our 1-dimensional kernel. This shift-invariant kernel is the kernel for a weighted Korobov space with parameters

$$\alpha = 2, \qquad \widehat{\beta}_j = 1 + \gamma_j \left(a_j^2 - a_j + \frac{1}{3}\right), \qquad \widehat{\gamma}_j = \frac{\gamma_j}{2\pi^2}.$$

The squared worst-case error calculated for the shift-invariant kernel corresponds to the expected value for the squared worst-case error in the anchored Sobolev space.

Table 3.2 and Table 3.3 are similar tables as in [62] for different weighted Korobov spaces and different weighted Sobolev spaces. In these tables the differences

n	0.9^j	0.5^j	0.1^j	j^{-1}	j^{-2}	j^{-6}
4001	3.2060e−02	1.9776e−04	3.4727e−05	9.2597e−03	3.7846e−04	1.0653e−04
8009	2.0162e−02	1.0388e−04	1.7383e−05	5.6899e−03	2.0379e−04	5.3402e−05
16001	1.2824e−02	5.4924e−05	8.7074e−06	3.5744e−03	1.1128e−04	2.6767e−05
32003	8.0782e−03	2.8685e−05	4.3617e−06	2.2159e−03	6.0764e−05	1.3423e−05
64007	5.0783e−03	1.4800e−05	2.1803e−06	1.3817e−03	3.2951e−05	6.7183e−06

Table 3.3: Randomly shifted worst-case errors e for a 100-dimensional weighted anchored Sobolev space with $a_j = 1$, γ_j as specified above each column, $\beta_j = 1$ and n as specified in the first column.

with the tables in [62] are marked with underlines. Note that our results were calculated in double precision (i.e., 53 bit mantissa), while those by Frances Kuo were calculated in long double precision (i.e., 64 bit mantissa).

In [23] and [22], Dick and Kuo present a variation on the original algorithm which constructs rank-1 lattices for composite n which constructs the generator per factor of n. They call this algorithm "Partial search" in contrast to the original algorithm which is then dubbed "Full search". The "Partial search" algorithm has construction cost $O(sn(p_1 + p_2 + \cdots + p_r))$, where n is a product of the distinct prime numbers $p_1, p_2, \ldots, p_r$. In this way they can construct rank-1 lattice rules with "millions of points". However in the same papers it is also shown that the theoretical convergence of these lattice rules degrades as the number of factors of n increases. In Table 3.4 we compare the worst-case error of a lattice rule with n prime with those published in [22], calculated in long double precision (i.e., 64 bit mantissa). These numbers were missing in [22] because they could not be calculated. Figures 3.5 and 3.6 show the same information. As can be seen in Figure 3.6, the worst-case error for $n = 7\,989\,067$ and $\gamma_j = j^{-2}$ suffers from numerical noise. The other results behave as expected.

From these experiments we conclude that it is possible to construct rank-1 lattice rules with "millions of points" without using a composite number of points. In Table 3.5 we list the components for an equally weighted Korobov space with $\alpha = 2$, $\gamma_j = 1/s_{\max}$ and $n = 54\,454\,681$ as a reference point. According to [23] and [22] the theoretical rate of convergence for composite n consisting of r factors decreases as r increases, so it seems profitable to be able to construct rules with the number of points being prime as is done here. Figures 3.5 and 3.6 confirm this. While the construction cost depends on the factors of n in [23] and [22], the construction cost here depends on the factors of $n-1$ and does not have a negative influence on the theoretical convergence.

n	r	0.5^j	j^{-2}	n	r	0.5^j	j^{-2}
2005001		6.1091e−07	1.6863e−06	2022157		6.0392e−07	1.6746e−06
2005007	2	7.1750e−07	1.9173e−06	2022161	4	8.6847e−07	2.4180e−06
2005019		6.1467e−07	1.6871e−06	2022187		5.9751e−07	1.6699e−06
2825567		4.4693e−07	1.2554e−06	2857159		4.3669e−07	1.2442e−06
2825617	2	5.1953e−07	1.4570e−06	2857177	4	6.4611e−07	1.8787e−06
2825639		4.4360e−07	1.2412e−06	2857181		4.3771e−07	1.2406e−06
4003981		3.2042e−07	9.3030e−07	3963161		3.2586e−07	9.3449e−07
4003997	2	3.7002e−07	1.0686e−06	3963181	4	4.9601e−07	1.3965e−06
4003999		3.2266e−07	9.3199e−07	3963209		3.2574e−07	9.3602e−07
5659627		2.3416e−07	6.9021e−07	5699773		2.3282e−07	6.8156e−07
5659637	2	2.7406e−07	8.0221e−07	5699779	4	3.3709e−07	1.0358e−06
5659651		2.3364e−07	6.9164e−07	5699789		2.3068e−07	6.8772e−07
8037191		1.6884e−07	5.1508e−07	7989011		1.7033e−07	5.1630e−07
8037211	2	1.9148e−07	5.9812e−07	7989013	4	2.5473e−07	7.4932e−07
8037229		1.6788e−07	5.1271e−07	7989067		1.6952e−07	5.8935e−07
1966079		6.1410e−07	1.7121e−06	1937207		6.3266e−07	1.7341e−06
1966087	3	7.8342e−07	2.2806e−06	1937221	5	1.0260e−06	2.8180e−06
1966123		6.1996e−07	1.7204e−06	1937227		6.2941e−07	1.7234e−06
2837381		4.4037e−07	1.2499e−06	2956783		4.2416e−07	1.2011e−06
2837407	3	5.6658e−07	1.6320e−06	2956811	5	7.3529e−07	1.9358e−06
2837431		4.4132e−07	1.2503e−06	2956813		4.2714e−07	1.2029e−06
4055927		3.1438e−07	9.1864e−07	4075289		3.1548e−07	9.1508e−07
4055929	3	4.1256e−07	1.2326e−06	4075291	5	4.8902e−07	1.5027e−06
4055957		3.1800e−07	9.1966e−07	4075297		3.1291e−07	9.1487e−07
5604997		2.3561e−07	7.0158e−07	5513623		2.3793e−07	7.0828e−07
5605027	3	3.1262e−07	9.3287e−07	5513623	5	4.1240e−07	1.1734e−06
5605037		2.3412e−07	6.9903e−07	5513663		2.3706e−07	7.0871e−07
8022409		1.6866e−07	5.1711e−07	7971311		1.6928e−07	5.1866e−07
8022431	3	2.3335e−07	6.7881e−07	7971317	5	2.8110e−07	8.1762e−07
8022437		1.6784e−07	5.1246e−07	7971323		1.7056e−07	5.1687e−07

Table 3.4: Comparison of lattice rules with a prime number of points with lattice rules with a composite number of points. These are randomly shifted worst-case errors e for a 100-dimensional weighted anchored Sobolev space with $a_j = 1$, γ_j as specified above each column, $\beta_j = 1$ and n as specified in the first column. Values for composite n with r factors are from [22], values for prime n were calculated with *fastrank1*.

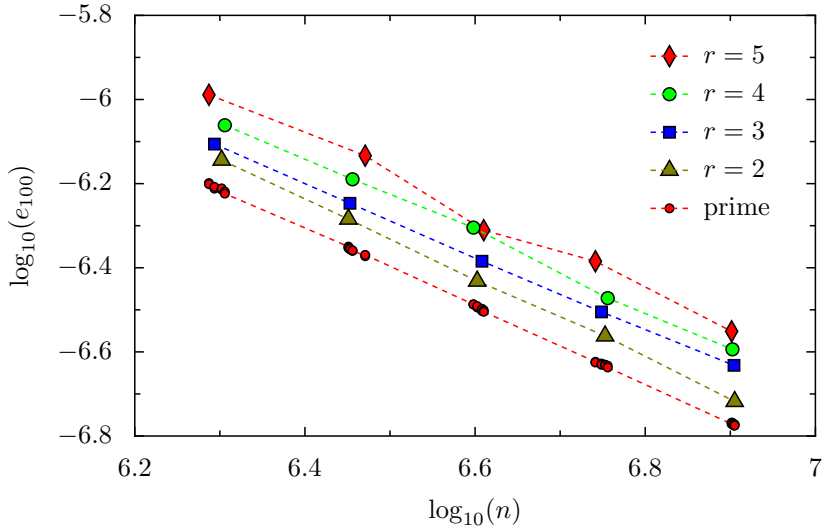


Figure 3.5: Comparison of composite versus prime n for randomly shifted lattice rules in a Sobolev space with $\gamma_j = 0.5^j$.

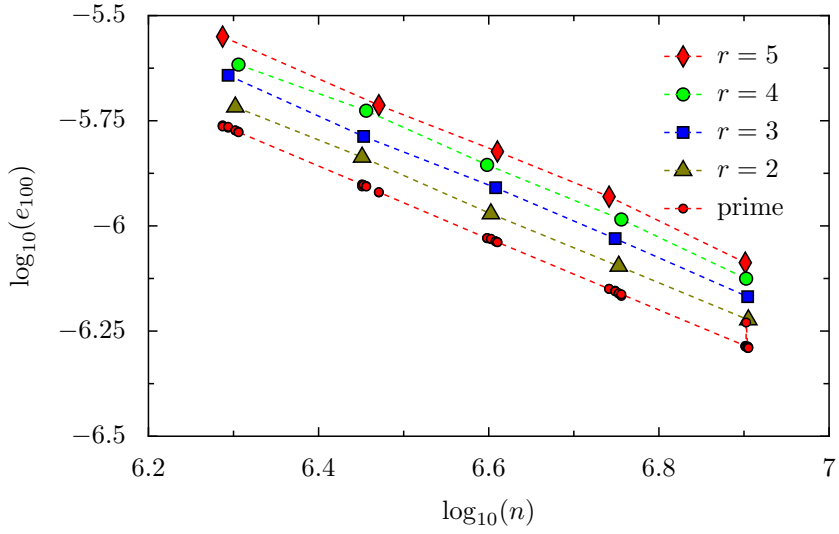


Figure 3.6: Comparison of composite versus prime n for randomly shifted lattice rules in a Sobolev space with $\gamma_j = j^{-2}$.

s	z	e	s	z	e
1	1	6.809e−09	11	3173994	1.253e−05
2	14625862	2.079e−08	12	5256076	1.797e−05
3	5824452	6.929e−08	13	8847489	2.471e−05
4	24617548	1.933e−07	14	5863079	3.341e−05
5	26921017	4.840e−07	15	26205728	4.432e−05
6	14116570	1.035e−06	16	21052882	5.764e−05
7	22111704	1.957e−06	17	10919917	7.345e−05
8	19715756	3.451e−06	18	20790066	9.159e−05
9	20234102	5.616e−06	19	24235075	1.135e−04
10	7806583	8.614e−06	20	21285727	1.383e−04

Table 3.5: Generating vector and worst-case errors for a 20-dimensional lattice rule in an equally weighted Korobov space, $\alpha = 2$, $\gamma_j = 1/s_{\max}$, $\beta_j = 1$, $n = 54\,454\,681$.

Chapter notes

The idea of fast component-by-component construction for product weights was published in [83] and the Matlab code appeared in [84]. In fact, a more engineered Matlab implementation, as hinted at in the text, was also created to test the algorithms. Fast component-by-component construction for order dependent weights appeared in [15]. In [84] some elaborations on different function spaces appearing in the literature were made, including the use of the fast construction algorithm for polynomial lattice rules which will be discussed in Chapter 6.

It might be surprising that the prime case is the easiest one for the fast construction since the FFT itself is the most simple to understand for powers of a (small) prime. In the derivation of the fast construction algorithm we meet sums over the function ω which are very similar looking as those that appear for the FFT. However, for FFT algorithms the summand is the exp-function and the fact that $\exp(a+b) = \exp(a)\exp(b)$ can be used. In general this is not true for our function ω and therefore we have to resort to purely number theoretic methods.

Chapter 4

Construction for a composite number of points

For both product weights and order dependent weights the construction cost of the component-by-component construction is dictated by the matrix-vector product. For prime number of points these matrices can be permuted such that the calculation is mainly replaced by one circulant matrix-vector product. In this chapter we investigate what happens when n is not a prime.

4.1 Component-by-component construction for n not a prime

Let us go through the essentials once again. We are considering rank-1 lattices with n points,

$$P_n = \left\{ \frac{k \cdot \boldsymbol{z}}{n} : 0 \le k < n \right\}, \tag{4.1}$$

where the integer vector $\boldsymbol{z}$ is called the generating vector of the lattice. The notation $k \cdot \boldsymbol{z}$ means (componentwise) multiplication modulo n. Both k and the components z_j are taken from $\mathbb{Z}_n$, the integers modulo n. Whenever we write $a \cdot b$ with $a, b \in \mathbb{Z}_n$ we mean multiplication modulo n; if ambiguity arises about the modulus then it will be written explicitly. The components of $\boldsymbol{z}$ are further restricted to the set

$$U_n = \{z \in \mathbb{Z}_n : \gcd(z, n) = 1\}, \qquad |U_n| = \varphi(n),$$

where the size of this set is given by the Euler totient function (the number of positive integers relatively prime to n which are less than n). This set contains

the *units* of $\mathbb{Z}_n$ and this assures that $(k \cdot z_j)/n$ is a permutation of the equispaced distribution k/n in each dimension j, where $k = 0, \ldots, n-1$. This also guarantees that the lattice has n distinct points in $[0,1)^s$.

The component-by-component construction algorithm finds the components z_j by minimizing the worst-case error in a reproducing kernel Hilbert space. In each iteration step s the formula for the worst-case error in fact contains a part similar to

$$v_s(z) = \sum_{k=0}^{n-1} p_{s-1}(k)\, \omega\left(\frac{k \cdot z}{n}\right), \qquad \forall z \in U_n,$$

which can easily be identified with a matrix-vector product (for all z at once) as

$$\boldsymbol{v}_s = \boldsymbol{\Omega}_n\, \boldsymbol{p}_{s-1},$$

where

$$\boldsymbol{\Omega}_n := \left[\omega\left(\frac{k \cdot z}{n}\right)\right]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}. \tag{4.2}$$

The matrix $\boldsymbol{\Omega}_n$ here is now defined for general n and has size $\varphi(n) \times n$. We have taken the liberty to define this matrix $\boldsymbol{\Omega}_n$ without specifying the iteration order of the elements $z \in U_n$ and $k \in \mathbb{Z}_n$ which define it. For now this order is arbitrary, although for correctness the ordering of $\boldsymbol{v}_s$ and $\boldsymbol{p}_{s-1}$ must match those of U_n and $\mathbb{Z}_n$ respectively, and this is silently assumed from now on.

By simple inspection of the component-by-component algorithm (see, e.g., Algorithm 2 on page 57 or Algorithm 3 on page 59), we find that the major cost in constructing such a rank-1 lattice rule is concentrated in the matrix-vector multiplication. A general matrix-vector product has time complexity $O(n^2)$ for a matrix of order n, and so the obvious component-by-component construction of a rank-1 lattice with n points in s dimensions is $O(sn^2)$. A more precise construction cost can be derived by using the actual size of the matrix $\boldsymbol{\Omega}_n$, which is $|U_n| \times |\mathbb{Z}_n| = \varphi(n) \times n$, and thus the construction cost is $O(s\varphi(n)n)$, where φ is the Euler totient function. Using a trivial bound this means that the construction cost is $O(sn^{2-\delta})$, with $0 < \delta \leq 1/2$, using a general matrix-vector product.

Since our matrix $\boldsymbol{\Omega}_n$ has at most n different elements, and since in such a case it is often possible to do a matrix-vector product in $O(n \log(n))$ instead of $O(n^2)$, we could of course hope that component-by-component construction could be done in $O(sn \log(n))$, similar to the technique for n prime from the previous chapter.

4.2 Cyclic groups and circulant matrices

Define the index-matrix $\boldsymbol{\Xi}_n$ to represent the structure of $\boldsymbol{\Omega}_n$. This index-matrix has the same size as $\boldsymbol{\Omega}_n$ where at position (z,k) there is not the value of $\omega((k \cdot z)/n)$,

but just the index $i = k \cdot z \bmod n$,

$$\boldsymbol{\Xi}_n = \big[k \cdot z \bmod n\big]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}.$$

The matrix $\boldsymbol{\Omega}_n$ can be formed in a simple way from $\boldsymbol{\Xi}_n$ by the application of the kernel function ω operating elementwise, assuming the same ordering of the index sets U_n and $\mathbb{Z}_n$,

$$\boldsymbol{\Omega}_n = \omega(\boldsymbol{\Xi}_n / n).$$

There is a matrix homomorphism from the matrix $\boldsymbol{\Xi}_n$ to the matrix $\boldsymbol{\Omega}_n$, i.e., the modulo multiplication structure present in $\boldsymbol{\Xi}_n$ is preserved in $\boldsymbol{\Omega}_n$. This matrix homomorphism will be formalized in the following definition where the term *codomain* is used to denote the set of entries in the matrix and the term *domain* to denote its index set.

Definition 4.1. A mapping ψ from the codomain of $\boldsymbol{A}$ onto the codomain of $\boldsymbol{B}$ defines a *matrix homomorphism* if there exist mappings σ_r and σ_c from the domain of $\boldsymbol{A}$ onto the domain of $\boldsymbol{B}$ such that

$$\forall (i,j) \in \text{domain}(\boldsymbol{A}) : \boldsymbol{A}(i,j) = t \Leftrightarrow \boldsymbol{B}(\sigma_r(i), \sigma_c(j)) = \psi(t).$$

Similarly a *matrix isomorphism* would mean that two matrices are isomorphic when the mappings σ_r, σ_c and ψ are one-to-one and onto. Note that these definitions are chosen in such a way that the Cayley tables (or multiplication tables) of two isomorphic groups A and B are isomorphic matrices and vice versa. This will play an important role in what follows.

For completeness some basic abstract algebra results are stated which will be useful further on, and which can all be found in a standard textbook, e.g., [33].

Definition 4.2. A group G is a *cyclic group* whenever all its elements can be generated by the powers of an element $g \in G$, called a *generator*, and thus

$$G = \langle g \rangle = \{g^k : k \in \mathbb{Z}\}.$$

Corollary 4.3. *The multiplicative group* $U_n = \{z \in \mathbb{Z}_n : \gcd(z,n) = 1\}$*, with order given by the Euler totient function as* $|U_n| = \varphi(n)$*, is cyclic whenever*

$$n = 2,\ 4,\ p^k \ \text{ or } \ 2p^k,$$

with p *an odd prime. A generator for the cyclic group* U_n *is called a* primitive root *modulo* n*.*

An algorithm to find a primitive root modulo n can be found in [11] and will be presented in Listing 5.1. If g is a generator for the cyclic group U_n (with n

given as in Corollary 4.3) then the elements of U_n can be listed in natural order of this generator as

$$U_n = \langle g \rangle = \{g^0, g^1, g^2, \ldots, g^{\varphi(n)-1}\}. \tag{4.3}$$

The connection between the Cayley table of a cyclic group and a circulant matrix is now apparent. Recall Definition 3.1.

Definition 4.4. A *circulant matrix* $\boldsymbol{C}_m = \operatorname{circ}(\boldsymbol{c})$ of order m is a matrix defined by the m elements in the vector $\boldsymbol{c}$, indexed from 0, as

$$\left[\boldsymbol{C}_m\right]_{k,\ell} = c_{k-\ell \bmod m}.$$

The Cayley table of a cyclic group can be made to look like a circulant matrix of order $\varphi(n)$. Since the powers of a generator g have period $\varphi(n)$, see (4.3), and using that

$$g^{\alpha} g^{-\beta} \equiv g^{\alpha-\beta \bmod \varphi(n)} \pmod{n},$$

for $0 \leq \alpha, \beta < \varphi(n)$ it is easy to see that the Cayley table of a cyclic group G can be identified with a circulant matrix.

Consider a cyclic group G with a generator g. The Cayley table can then be pictured as the left part in (4.4).

$$\begin{array}{c|ccccc} \cdot & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ \hline g^0 & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ g^1 & g^1 & g^2 & g^3 & \iddots & g^0 \\ g^2 & g^2 & g^3 & \iddots & \iddots & g^1 \\ \vdots & \vdots & \iddots & \iddots & \iddots & \vdots \\ g^{-1} & g^{-1} & g^0 & g^1 & \cdots & g^{-2} \end{array} \simeq \begin{array}{c|ccccc} \cdot & g^0 & g^{-1} & g^{-2} & \cdots & g^1 \\ \hline g^0 & g^0 & g^{-1} & g^{-2} & \ldots & g^1 \\ g^1 & g^1 & g^0 & g^{-1} & \ddots & \vdots \\ g^2 & g^2 & g^1 & \ddots & \ddots & g^{-2} \\ \vdots & \vdots & \ddots & \ddots & \ddots & g^{-1} \\ g^{-1} & g^{-1} & \ldots & g^2 & g^1 & g^0 \end{array} \tag{4.4}$$

Using the natural order of a generator results in constant anti-diagonals. Using the negative powers of the generator for the columns of the Cayley table, and keeping the positive powers of the generator for the rows, the table depicted at the right of (4.4) is obtained. This table has constant diagonals and can be interpreted as a circulant matrix.

Using the eigenvalue decomposition of a circulant matrix, a matrix-vector product only costs $O(m \log(m))$ for a matrix of order m (prime or composite), see Corollary 3.3, which was already used in Chapter 3.

4.3 Partitioning the index-matrix into circulant blocks

The technique that will be used for the fast construction is based on block-partitioning the matrix $\boldsymbol{\Xi}_n$ into smaller matrices which are isomorphic to circulant matrices (and thus isomorphic to cyclic groups). The derived techniques will work for any matrix $\boldsymbol{\Omega}_n$ as long as this matrix is homomorphic to $\boldsymbol{\Xi}_n$ (i.e., only the multiplicative structure is used).

In the following the numbers $v \in \mathbb{Z}_n$ will be represented in a "residue number system" (e.g., [38, Section 4.7]), where the remainders of v are taken with respect to moduli n_i that are prime to each other and $n = n_1 n_2 \cdots n_r$. The most natural way is to use the prime factorization of n, with the $n_i = p_i^{k_i}$, so

$$\begin{aligned} v &\simeq (v \bmod n_1, v \bmod n_2, \ldots, v \bmod n_r) \\ &\simeq (v_1, v_2, \ldots, v_r). \end{aligned}$$

The Chinese remainder theorem states that this representation is unique (whenever the n_i are prime to each other) and we can thus map from v to $(v_1, v_2, \ldots, v_r)$ and back (an algorithm can be found in [11]). Note that in this representation the units take the special form:

$$u \simeq (u_1, u_2, \ldots, u_r) \in U_n \Leftrightarrow u_i \in U_{n_i}.$$

4.3.1 Partitioning of U_n

First we partition U_n in terms of smaller cyclic groups. The structure of U_n is given by the next well known theorem and its corollary.

Theorem 4.5. *The multiplicative group U_n is isomorphic to the external direct product of groups U_{n_i} where $n = n_1 n_2 \cdots n_r$ is a factorization of n and the n_i are prime to each other*

$$U_n \simeq U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r}.$$

Corollary 4.6. *Every group U_n can be written as*

$$U_n \simeq \begin{cases} U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{if } 8 \nmid n, \\ \left(\langle 2^{k-1}-1\rangle_{2^k} \oplus \langle 5\rangle_{2^k}\right) \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{if } 8 \mid n,\ n_1 = 2^k,\ k \geq 3, \end{cases}$$

where every subgroup is a cyclic multiplicative group.

Proof. A proof can be found in most abstract algebra books (e.g., [33, p. 155]) mostly in the proximity of the fundamental theorem of abelian groups, but since some details for prime factors of the form 2^k are needed later on, we sketch the outline.

Consider the group U_n and a prime factorization of $n = n_1 n_2 \cdots n_r$, $n_i = p_i^{k_i}$. When $8 \nmid n_i$ the group U_{n_i} is already cyclic (see Corollary 4.3). Only the case where one of the prime factors, say n_1, is 2^k with $k \geq 3$ needs to be considered. For such a group the "pseudogenerator" 5 generates half of U_{2^k}, from which the other half can easily be derived. This gives

$$U_{2^k} = \{1 \cdot \langle 5 \rangle \bmod 2^k, (2^{k-1} - 1) \cdot \langle 5 \rangle \bmod 2^k\} \simeq \langle 2^{k-1} - 1 \rangle_{2^k} \oplus \langle 5 \rangle_{2^k}.$$

This comes from the well known isomorphism $U_{2^k} \simeq \mathbb{Z}_2 \oplus \mathbb{Z}_{2^{k-2}}$. We note that such a power of 2 thus makes an isomorphic copy of half of U_n. This proves the corollary. □

So, given a group U_n we can find r smaller cyclic groups U_{n_i} of order $\varphi(n_i)$ and generators g_i (we make abstraction of the special case for $8 \mid n_i$ since it would clutter the explanations following, however the reasoning still holds and an example follows in §4.5.2). We can then order the elements of $\oplus_i U_{n_i}$ in lexicographical order of the powers of these generators

$$\oplus_i U_{n_i} = \left\{ \begin{array}{ccc} (g_1^0, \ldots, g_{r-1}^0, g_r^0), & \ldots, & (g_1^0, \ldots, g_{r-1}^0, g_r^{-1}), \\ (g_1^0, \ldots, g_{r-1}^1, g_r^0), & \ldots, & (g_1^0, \ldots, g_{r-1}^1, g_r^{-1}), \\ & \vdots & \\ (g_1^{-1}, \ldots, g_{r-1}^{-1}, g_r^0), & \ldots, & (g_1^{-1}, \ldots, g_{r-1}^{-1}, g_r^{-1}) \end{array} \right\},$$

as well as in order of the negative powers of these generators to build a Cayley table.

As an example consider U_n, where $n = p_1 p_2$, a generator g_1 for U_{p_1} and g_2 for U_{p_2}, then the Cayley table can be made to look like:

$\cdot$	$(g_1^0, \boldsymbol{g_2^{-1}})$	$(g_1^{-1}, \boldsymbol{g_2^{-1}})$	$\cdots$	$(g_1^1, \boldsymbol{g_2^{-1}})$
$(g_1^0, \boldsymbol{g_2})$	$(g_1^0, \boldsymbol{C}_2)$	$(g_1^{-1}, \boldsymbol{C}_2)$	$\cdots$	$(g_1^1, \boldsymbol{C}_2)$
$(g_1^1, \boldsymbol{g_2})$	$(g_1^1, \boldsymbol{C}_2)$	$(g_1^0, \boldsymbol{C}_2)$	$\cdots$	$(g_1^2, \boldsymbol{C}_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(g_1^{-1}, \boldsymbol{g_2})$	$(g_1^{-1}, \boldsymbol{C}_2)$	$(g_1^{-2}, \boldsymbol{C}_2)$	$\cdots$	$(g_1^0, \boldsymbol{C}_2)$

where $\boldsymbol{C}_2 = \begin{bmatrix} g_2^0 & g_2^{-1} & \cdots & g_2^1 \\ g_2^1 & g_2^0 & \cdots & g_2^2 \\ \vdots & \vdots & \ddots & \vdots \\ g_2^{-1} & g_2^{-2} & \cdots & g_2^0 \end{bmatrix}$.

Here $\boldsymbol{C}_2$ is the circulant form of the Cayley table for U_{p_2} and the notation $(g_1^k, \boldsymbol{C}_2)$ means that g_1^k has to be combined with every entry from $\boldsymbol{C}_2$, likewise the notation $(g_1^k, \boldsymbol{g_2})$ means to combine every element from $\boldsymbol{g_2} = [g_2^0, g_2^1, \ldots, g_2^{-1}]$ with g_1^k and similarly $\boldsymbol{g_2^{-1}} = [g_2^0, g_2^{-1}, \ldots, g_2^1]$. The complete Cayley table can now be seen as a block circulant matrix with circulant blocks.

This can obviously be extended to more than 2 factors and then the Cayley table for r factors could be seen as an r times nested block circulant matrix. A matrix-vector product with such a (nested block) circulant matrix with r levels of nesting can be done in time $O(rn\log(n))$, see the forthcoming Lemma 4.14 in §4.4.

4.3.2 Partitioning of $\mathbb{Z}_n$

The set $\mathbb{Z}_n$ will now be split by considering a group action ψ by the transformation group U_n on $\mathbb{Z}_n$. For completeness we will introduce the necessary definitions.

Definition 4.7. A group G acts on a set X by a *group action* $\psi : G \times X \to X$ if for every $x \in X$:

(i) $\psi(e, x) = x$, where e is the identity element of G,
(ii) $\psi(g, \psi(h, x)) = \psi(gh, x)$ for all $g, h \in G$.

The *orbit* of $x \in X$ is defined as $\mathrm{orb}(x) = \{\psi(g, x) : g \in G\}$ and is the subset of X which can be reached by x under the transformations of G.

For our purpose the group $G = U_n$ acts on the set $X = \mathbb{Z}_n$, and the group action ψ is multiplication modulo n. This is in fact a very natural view on the generation of the point set P_n as given in (4.1), where the components of the generating vector are selected from the units of $\mathbb{Z}_n$, i.e., U_n, to assure that $(k \cdot z_j)/n$ is a permutation of the equispaced distribution k/n in each dimension j, where $k = 0, \ldots, n-1$. Note that since the group action is simply the operation of the group U_n, the action of an element $v \in U_n$ on just U_n (instead of $\mathbb{Z}_n$) can be written as the coset $v\,U_n := \{v \cdot u : u \in U_n\}$.

Every possible choice of z_s in the optimization process for dimension s can be seen as a possible permutation of the n 1-dimensional points. From algebra it is known that a given action G on X defines an equivalence relation where the different orbits are the partitions. We will now show that the divisors of n are valid representers for these orbits on $\mathbb{Z}_n$. This gives a way to partition $\mathbb{Z}_n$.

Theorem 4.8. *The union of all orbits generated by the divisors of n under U_n form a partition of $\mathbb{Z}_n$*

$$\bigcup_{d|n} \mathrm{orb}(d) = \bigcup_{d|n} d\,U_n = \mathbb{Z}_n,$$

where the partition represented by d has size $|\mathrm{orb}(d)| = \varphi(n/d)$. *Thus*

$$n = \sum_{d|n} |\mathrm{orb}(d)| = \sum_{d|n} |d\,U_n| = \sum_{d|n} \varphi(n/d).$$

Proof. Note that since $n \equiv 0 \pmod{n}$ we conveniently use n for 0, and vice versa, whichever is appropriate.

Observe that the orbit of d can be specified in different ways:

$$\begin{aligned} \mathrm{orb}(d) &= \{d \cdot u : u \in U_n\}, && \text{the definition,} \\ &= d\,U_n, && \text{as a coset of } U_n, \\ &= \{v \in \mathbb{Z}_n : \gcd(v, n) = d\}, && \text{having a common gcd.} \end{aligned}$$

The last form is the most interesting for us (and follows directly from observing the prime powers). Since for all $v \in \mathbb{Z}_n$

$$\gcd(v, n) = d \in \mathrm{divisors}(n),$$

this shows that the divisors are representers for the partitions.

Left to prove is $|\mathrm{orb}(d)| = \varphi(n/d)$. We will consider $v \in \mathbb{Z}_n$ in the residue number system with moduli given by the prime factorization of $n = \prod_i p_i^{k_i}$, so that

$$\mathrm{orb}(d) = d\,U_n \simeq \oplus_i\, d_i\, U_{p_i^{k_i}}, \qquad \text{with } d_i = d \bmod p_i^{k_i}.$$

As such the total number of elements in $d\,U_n$ for a divisor $d = \prod_i p_i^{\ell_i}$, with $0 \le \ell_i \le k_i$, is the product of the number of elements in the cosets $d_i\, U_{p_i^{k_i}}$ (modulo $p_i^{k_i}$) and these are given by

$$|d_i\, U_{p_i^{k_i}}| = \varphi(p_i^{k_i}/p_i^{\ell_i}), \qquad \text{since } \gcd(d, p_i^{k_i}) = \gcd(d_i, p_i^{k_i}) = p_i^{\ell_i}.$$

It follows that

$$|\mathrm{orb}(d)| = |d\,U_n| = \prod_i \varphi(p_i^{k_i}/p_i^{\ell_i}) = \varphi(n/d).$$

□

4.3.3 Block-partitioning of $\boldsymbol{\Xi}_n$

If we combine Theorem 4.5 and Theorem 4.8 then $\boldsymbol{\Xi}_n$ can be partitioned in blocks which are isomorphic to the Cayley table of groups $U_{n/d}$. Let us first introduce the following lemma.

Lemma 4.9. *The following are equivalent for $n = \prod_i n_i$ and all n_i prime to each other and d a divisor of n:*

$$\begin{aligned} d\,U_n \bmod n &= d\,U_{n/d} \bmod n \\ &\simeq \oplus_i \left(d_i\,U_{n_i/g_i} \bmod n_i\right) \\ &\simeq \left(d \oplus_i U_{n_i/g_i}\right) \bmod n, \end{aligned}$$

with $d_i = d \bmod n_i$ and $g_i = \gcd(d_i, n_i)$, all with operation modulo n (or modulo n_i for the parts in the residue number system). Moreover

$$\begin{aligned} &w\,U_n \bmod n = w\,U_{n/g} \bmod n, \\ &w\,\mathbb{Z}_n \bmod n = w\,\mathbb{Z}_{n/g} \bmod n, \qquad \text{with } g = \gcd(w,n) \text{ and } w \in \mathbb{Z}. \end{aligned}$$

Proof. (An elaborated version of this proof can be found in §4.8.)

We prove that $w\,\mathbb{Z}_n \bmod n = g\,\mathbb{Z}_{n/g} \bmod n$ (where w does not necessarily divide n, since that is trivial). The other cases then follow easily. First set $g = \gcd(w,n)$ such that $\gcd(w/g, n/g) = 1$ and $g \mid n$. Consequently $w/g \in U_{n/g}$ and multiplication of the complete set $\mathbb{Z}_{n/g}$ with an element from $U_{n/g}$ returns the complete set $\mathbb{Z}_{n/g}$. We then find

$$\begin{aligned} w\,\mathbb{Z}_n \bmod n &= \{w \cdot v \bmod n : v \in \mathbb{Z}_n\} \\ &= \{g \cdot (w/g \cdot v) \bmod n : v \in \mathbb{Z}_n\} \\ &= \{g \cdot (w/g \cdot v \bmod n/g) \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= \{g \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= g\,\mathbb{Z}_{n/g} \bmod n. \end{aligned}$$

Note that the g up front can be changed into a w by keeping w/g outside of the modulo n/g. □

In the next theorem the symbol $\mathbf{1}_{t\times 1}$ denotes a vector of length t with all components equal to one and the symbol $\otimes$ is used to denote the Kronecker tensor product. So $\mathbf{1}_{t\times 1} \otimes \boldsymbol{B}$ can be read as t replications of the matrix $\boldsymbol{B}$ on top of each other. Wherever we write $d\,U_{n/d}$ in the next theorem, we mean modulo n as in the spirit of Lemma 4.9. The following theorem describes how to block-partition the matrix $\boldsymbol{\Xi}_n$.

Theorem 4.10. *We can block-partition the matrix*

$$\boldsymbol{\Xi}_n = \Big[k \cdot z\Big]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}$$

by considering the divisors of n (denoted as $d^{(1)}$, $d^{(2)}$, …, $d^{(\nu)}$, with $\nu = d(n)$ the number of divisors of n), into vertical partitions $\boldsymbol{A}_d$ per divisor d,

$$\boldsymbol{\Xi}_n \simeq [\boldsymbol{A}_{d^{(1)}} | \boldsymbol{A}_{d^{(2)}} | \cdots | \boldsymbol{A}_{d^{(\nu)}}], \qquad \boldsymbol{A}_d = \Big[k \cdot z\Big]_{\substack{z \in U_n \\ k \in d\,U_{n/d}}},$$

of sizes $\varphi(n) \times \varphi(n/d)$. *These partitions* $\boldsymbol{A}_d$ *can* each separately *be horizontally partitioned into* t_d *identical square blocks* $\boldsymbol{B}_d$ *for which*

$$\boldsymbol{A}_d \simeq \begin{bmatrix} \boldsymbol{B}_d \\ \hline \vdots \\ \hline \boldsymbol{B}_d \end{bmatrix} = \mathbf{1}_{t_d \times 1} \otimes \boldsymbol{B}_d, \qquad \boldsymbol{B}_d = \Big[k \cdot z\Big]_{\substack{z \in U_{n/d} \\ k \in d\,U_{n/d}}} = \Big[d \cdot k \cdot z\Big]_{\substack{z \in U_{n/d} \\ k \in U_{n/d}}},$$

of size $\varphi(n/d) \times \varphi(n/d)$, *and so* $t_d = \varphi(n)/\varphi(n/d)$. *The blocks* $\boldsymbol{B}_d$ *are isomorphic with the Cayley tables of* $U_{n/d}$.

Proof. The vertical partitioning follows directly from Theorem 4.8 and the equality $d\,U_n \bmod n = d\,U_{n/d} \bmod n$ from Lemma 4.9.

We will now prove that these vertical partitions can be partitioned horizontally in $t_d = \varphi(n)/\varphi(n/d)$ identical square blocks which are isomorphic to the Cayley table of $U_{n/d}$. We will shift our problem to the residue number system, with moduli given by the prime factorization of $n = n_1 n_2 \cdots n_r$, $n_i = p_i^{k_i}$. From Lemma 4.9 we have that

$$d\,U_{n/d} \simeq \oplus_i\, d_i\, U_{n_i/g_i} \simeq d\ \oplus_i U_{n_i/g_i}, \quad \text{with } d_i = d \bmod n_i \text{ and } g_i = \gcd(d_i, n_i),$$

and it follows that

$$|d_i\, U_{n_i/g_i}| = |U_{n_i/g_i}| = \varphi(n_i/g_i).$$

Now consider the matrix $\boldsymbol{B}_{d_i}$ as a set B_{d_i} (i.e., the codomain of $\boldsymbol{B}_{d_i}$) and follow the same reasoning as in the proof of Lemma 4.9

$$\begin{aligned} B_{d_i} &= \{d_i \cdot k_i \cdot z_i \bmod n_i : z_i \in U_{n_i}, k_i \in U_{n_i/g_i}\} \\ &= d_i\, U_{n_i/g_i} \bmod n_i, \end{aligned}$$

which has size $|B_{d_i}| = \varphi(n_i/g_i)$. We observe that the $\varphi(n_i)$ elements of U_{n_i} can be partitioned in $t_{d,i} = \varphi(n_i)/\varphi(n_i/g_i)$ equivalence classes which have the elements of U_{n_i/g_i} as representers, i.e., B_{d_i} is isomorphic to U_{n_i/g_i}.

We thus have that the number of elements horizontally and vertically is both $\varphi(n_i/g_i)$ and that there are only $\varphi(n_i/g_i)$ distinct elements in $\boldsymbol{B}_{d_i}$, having a modulo multiplication structure. Using the Chinese remainder theorem, $B_d \simeq \oplus_i B_{d_i}$, it follows that we have $t_d = \varphi(n)/\varphi(n/d)$ copies and that $\boldsymbol{B}_d$ is isomorphic to the Cayley table of $U_{n/d}$. □

The previous theorem has not filled in the details of how exactly the rows and the columns of the matrix $\boldsymbol{\Xi}_n$ should be permuted. It only states that this is possible for each vertical partition $\boldsymbol{A}_d$ (cf. the wording *each separately*). The following corollary states that these two permutations can be fixed for the complete matrix at once.

Corollary 4.11. *If we fix the ordering of the rows for all partitions $\boldsymbol{A}_d$ and arrange the ordering of the columns so that $\boldsymbol{B}_1 \simeq \boldsymbol{C}_n$, where $\boldsymbol{C}_n$ is the (nested block) circulant form of the Cayley table of the group U_n, then the interleaving of the (nested block) circulant matrices $\boldsymbol{B}_d \simeq d\,\boldsymbol{C}_{n/d}$ for a certain divisor d of n in $\boldsymbol{A}_d$ is given by*

$$\boldsymbol{A}_d = (\otimes_i \boldsymbol{R}_{d,i})\; d\,\boldsymbol{C}_{n/d},$$

where the i indices go over the prime factors of n (with the exceptional case for powers of 2 as given in Corollary 4.6) and the $\boldsymbol{R}_{d,i}$ matrices are defined as

$$\boldsymbol{R}_{d,i} = \mathbf{1}_{t_{d,i}\times 1} \otimes \boldsymbol{I}_{\varphi(n_i/g_i)},$$

where $d_i = d \bmod n_i$, $g_i = \gcd(d_i, n_i)$, $t_{d,i} = \phi(n_i)/\phi(n_i/g_i)$ and $\boldsymbol{I}_{\varphi(n_i/g_i)}$ is the identity matrix of order $\varphi(n_i/g_i)$.

Proof. By simple calculation. □

This corollary gives a straightforward method to know where every element of a matrix $\boldsymbol{B}_d$ arrives in its corresponding matrix $\boldsymbol{A}_d$. The interpretation of multiplication with a matrix $\boldsymbol{R}_d = \otimes_i \boldsymbol{R}_{d,i}$ is quite natural since $d\,U_{n/d} = \oplus_i d_i\,U_{n_i/g_i}$. If we look at the basic case of multiplication with one matrix $\boldsymbol{R}_{d,i}$ we observe that the identity matrix has the effect of distributing every element of the group $d_i\,U_{n_i/g_i}$, and the replicating by $\mathbf{1}_{t_{d,i}\times 1}$ fills up the empty space left in $\boldsymbol{C}_{n_i}$ when $g_i \neq 1$.

By using the formulation of Corollary 4.11 we will be able to distribute the results of matrix-vector multiplications with the smaller (nested block) circulant matrices $\boldsymbol{B}_d$ to the final result vector of multiplication with the complete matrix.

4.4 Fast matrix-vector for general n

With the result from Theorem 4.10 we can now show that a fast matrix-vector product with matrices homomorphic to $\boldsymbol{\Xi}_n$ is possible in time $O(n\log(n))$. The proof of the next theorem uses two lemmas which are deferred to the end of this section and contain some technical details.

Theorem 4.12. *A matrix-vector product with a matrix $\boldsymbol{\Omega}_n$ which is homomorphic to $\boldsymbol{\Xi}_n$ can be done in time $O(n\log(n))$ and requires memory of order $O(n)$.*

Proof. Using Theorem 4.10 we can partition the matrix $\boldsymbol{\Omega}_n$ in vertical partitions $\boldsymbol{A}_d$ which have (interleaved) copies of (nested block) circulant matrices $\boldsymbol{B}_d$. Since the matrix $\boldsymbol{\Omega}_n$ is homomorphic to $\boldsymbol{\Xi}_n$ the same permutations can be done and we can assume an arbitrary elementwise mapping function ψ on $\boldsymbol{\Xi}_n$ to obtain $\boldsymbol{\Omega}_n$.

For our specific purpose this mapping function is $\psi(t) = \omega(t/n)$. We thus consider the matrix-vector product

$$\begin{aligned}\boldsymbol{v} &= \psi\left([\boldsymbol{A}_{d^{(1)}}|\boldsymbol{A}_{d^{(2)}}|\cdots|\boldsymbol{A}_{d^{(\nu)}}]\right)\left[\begin{array}{c}\boldsymbol{p}_{d^{(1)}}\\\hline\boldsymbol{p}_{d^{(2)}}\\\hline\vdots\\\hline\boldsymbol{p}_{d^{(\nu)}}\end{array}\right]\\ &= \psi(\boldsymbol{A}_{d^{(1)}})\,\boldsymbol{p}_{d^{(1)}} + \psi(\boldsymbol{A}_{d^{(2)}})\,\boldsymbol{p}_{d^{(2)}} + \cdots + \psi(\boldsymbol{A}_{p^{(\nu)}})\,\boldsymbol{p}_{d^{(\nu)}},\end{aligned} \tag{4.5}$$

which can be considered as the sum of ν smaller matrix-vector products, where $\nu = d(n)$, the number of divisors of n. Using Theorem 4.10 it then suffices to calculate $\psi(\boldsymbol{B}_d)\,\boldsymbol{p}_d$ instead of $\psi(\boldsymbol{A}_d)\,\boldsymbol{p}_d$, since $\boldsymbol{A}_d$ contains only stacked copies (in some order) of the same (nested block) circulant matrix $\boldsymbol{B}_d$.

The size of $\boldsymbol{B}_d$ is $\varphi(n/d)\times\varphi(n/d)$ and it follows from Lemma 4.14 (forthcoming) that its matrix-vector product can be done in $O(\kappa(n/d)\,\varphi(n/d)\log(\varphi(n/d)))$, where $\kappa(n/d)$ is the number of unique factors of n/d. Summing of the $d(n)$ result vectors can be done in $O(\kappa(n)\,n)$ (see Lemma 4.13 following). The total cost of the complete matrix-vector product is then

$$O\left(\sum_{d|n}\kappa(n/d)\,\varphi(n/d)\log(\varphi(n/d)) + \kappa(n)\,n\right) = O(n\log(n)), \tag{4.6}$$

where we used $\sum_{d|n}\varphi(d) = n$ and $\kappa(n)$ bounded by a constant.

The memory needed for this operation is $O(n)$ which will be explained in Lemma 4.14. □

In (4.6) it is actually assumed that $\kappa(n)$ is bounded by a constant. For practical implementations this will always be the case and will be reasonable, e.g., $\kappa(n) \le 9$ for all $n \le 2^{32}$. However, a crude bound like $\kappa(n) < \log_2(n)$ (the logarithm in base 2) could also be used, from which it follows that the total complexity is always less than $O(n\log^2(n))$ (the logarithm to the power 2). This is however a serious overestimate. Also it is known that on the average $\kappa(n) \sim \log(\log(n))$, and for the worst choice, i.e., n a product of distinct primes, it is known that on the average $\kappa(n) \sim \frac{\log(n)}{\log(\log(n))}$.

The second part in the total complexity for multiplication with $\boldsymbol{\Omega}_n$ is the summing of the $d(n)$ result vectors. This could be bound naively by $d(n)\,\varphi(n)$ and for prime n we have $d(p) = 2$ and $\varphi(p) = p - 1$, and as such this cost is negligible compared with the $O(n\log(n))$ for the matrix-vector products. Also for prime powers this works out fine, since then $d(p^k) = k + 1 = O(\log(n))$ and $\varphi(p^k) = p^{k-1}(p-1) = O(n)$. However, for general n this does not look so good. For composite n we could bound $d(n)\,\varphi(n)$ very crudely as

$$d(n)\,\varphi(n) \le 2\sqrt{n}\,(n - \sqrt{n}) = 2(n^{3/2} - n),$$

which will then dominate the cost over the $O(n\log(n))$ from the matrix-vector products. Summing like this will give a total complexity of $O(n^{3/2}-n)$ but is still asymptotically better than $O(n^{2-\delta})$ for doing the full matrix-vector product (without transforms) as shown in §4.1. Luckily it is possible to do a better summing job by carefully choosing the order of the divisors.

Lemma 4.13. *The summing of the $d(n)$ result vectors in (4.5) of the matrix-vector products $\boldsymbol{v}_d = \psi(\boldsymbol{B}_d)\,\boldsymbol{p}_d$ with the (nested block) circulant matrices $\boldsymbol{B}_d$ can be done in time $O(\kappa(n)\,n)$ by stepping through the powers of the divisors in lexicographical order.*

Proof. To achieve a good summing order consider the divisors of n in lexicographical ordering of the powers of their prime components (so the divisors itself appear out of order). For $n = p_1^{k_1}p_2^{k_2}\cdots p_r^{k_r}$, with $r=\kappa(n)$, we have divisors

$$d = p_1^{\ell_1}p_2^{\ell_2}\cdots p_r^{\ell_r}, \qquad\qquad \text{where } 0\le \ell_i \le k_i.$$

Observe that two adjoining vertical partitions, the first for a divisor $d = p_r^{\ell_r}\cdots p_2^{\ell_2}p_1^{\ell_1}$ and the other for a divisor $d' = p_r^{\ell_r}\cdots p_2^{\ell_2}p_1^{\ell_1+1}$, need

$$\varphi\left(\frac{n}{\gcd(d,d')}\right) = \varphi(n/d) = \varphi(p_r^{k_r-\ell_r}\cdots p_2^{k_2-\ell_2}p_1^{k_1-\ell_1})$$

operations to sum their results. This summing of the adjoining partitions for the first prime factor p_1 must be done for $\ell_1 = 0,\ldots,k_1-1$ and this for all powers of the other prime factors. If we do this for increasing ℓ_1 then the formula above holds for all intermediate results and gives a partial cost for the first prime factor of

$$\begin{aligned}\sum_{\ell_r=0}^{k_r}\cdots\sum_{\ell_2=0}^{k_2}\sum_{\ell_1=0}^{k_1-1}&\varphi(p_r^{k_r-\ell_r}\cdots p_2^{k_2-\ell_2}p_1^{k_1-\ell_1})\\ &= \sum_{\ell_r=0}^{k_r}\varphi(p_r^{k_r-\ell_r})\cdots\sum_{\ell_2=0}^{k_2}\varphi(p_2^{k_2-\ell_2})\sum_{\ell_1=0}^{k_1-1}\varphi(p_1^{k_1-\ell_1})\\ &= p_r^{k_r}\cdots p_2^{k_2}(p_1^{k_1}-1) = O(n).\end{aligned}$$

We now have result vectors of sizes $\varphi(p_r^{k_r-\ell_r}\cdots p_2^{k_2-\ell_2}p_1^{k_1})$, and thus, from now on, the part from the first prime will always count for its full size $\varphi(p_1^{k_1})$. Similarly, for the next prime factor

$$\begin{aligned}\sum_{\ell_r=0}^{k_r}\cdots\sum_{\ell_2=0}^{k_2-1}\varphi(p_r^{k_r-\ell_r}\cdots p_2^{k_2-\ell_2}p_1^{k_1}) &= \sum_{\ell_r=0}^{k_r}\varphi(p_r^{k_r-\ell_r})\cdots\sum_{\ell_2=0}^{k_2-1}\varphi(p_2^{k_2-\ell_2})\,\varphi(p_1^{k_1})\\ &= p_r^{k_r}\cdots(p_2^{k_2}-1)\,\varphi(p_1^{k_1}) = O(n).\end{aligned}$$

This continues up to the final prime factor, which has a cost of

$$\sum_{\ell_r=0}^{k_r-1} \varphi(p_r^{k_r-\ell_r} \cdots p_2^{k_2} p_1^{k_1}) = \sum_{\ell_r=0}^{k_r-1} \varphi(p_r^{k_r-\ell_r}) \cdots \varphi(p_2^{k_2})\, \varphi(p_1^{k_1})$$
$$= (p_r^{k_r} - 1) \cdots \varphi(p_2^{k_2})\, \varphi(p_1^{k_1}) = O(n).$$

Since there are $r = \kappa(n)$ levels (the number of unique prime factors) and the cost on each level is $O(n)$, the total complexity is $O(\kappa(n)\, n)$. □

The last piece of the puzzle is fast matrix-vector multiplication with nested block circulant matrices of any nesting. We provide a method for such a fast matrix-vector multiplication in the following lemma.

Lemma 4.14. *A matrix-vector product with a nested block circulant matrix $\boldsymbol{C}$ of order n (where the blocks are again block circulant, or circulant at the lowest level) can be done in time $O(kn \log(n))$ requiring memory $O(n)$, where k is the number of nested circulant levels.*

Proof. In Corollary 3.3 it was already shown that a matrix-vector product with a circulant matrix can be done in $O(n \log(n))$ using its eigenvalue decomposition which was given in Theorem 3.2.

Here we will show that any additional block circulant level can be made block diagonal, and furthermore completely diagonal by the use of a permutation and an additional sequence of FFTs (one per diagonal block).

Assume $\boldsymbol{C} = \operatorname{circ}(\boldsymbol{C}^{(0:m-1)}) \in \mathbb{R}^{nm \times nm}$ is a block circulant matrix with m circulant blocks of order n (i.e., there are 2 levels):

$$\boldsymbol{C} = \operatorname{circ}(\boldsymbol{C}^{(0:m-1)}) = \left[\begin{array}{c|c|c|c|c} \boldsymbol{C}^{(0)} & \boldsymbol{C}^{(m-1)} & \boldsymbol{C}^{(m-2)} & \cdots & \boldsymbol{C}^{(1)} \\ \hline \boldsymbol{C}^{(1)} & \boldsymbol{C}^{(0)} & \boldsymbol{C}^{(m-1)} & \ddots & \vdots \\ \hline \boldsymbol{C}^{(2)} & \boldsymbol{C}^{(1)} & \ddots & \ddots & \boldsymbol{C}^{(m-2)} \\ \hline \vdots & \ddots & \ddots & \ddots & \boldsymbol{C}^{(m-1)} \\ \hline \boldsymbol{C}^{(m-1)} & \cdots & \boldsymbol{C}^{(2)} & \boldsymbol{C}^{(1)} & \boldsymbol{C}^{(0)} \end{array}\right],$$

and $\boldsymbol{C}^{(j)} \in \mathbb{R}^{n \times n}$. Then we can diagonalize the smaller circulant matrices by m n-point Fourier transforms on the first column of $\boldsymbol{C}$ (i.e., on the first columns of the blocks $\boldsymbol{C}^{(j)} = \operatorname{circ}(\boldsymbol{c}^{(j)})$). This gives the start of an analog to the diagonalization in Theorem 3.2 (by applying this same theorem m times):

$$\boldsymbol{C} = (\boldsymbol{I}_m \otimes \boldsymbol{F}_n^{-1})\ \operatorname{circ}(\tilde{\boldsymbol{C}}^{(0:m-1)})\ (\boldsymbol{I}_m \otimes \boldsymbol{F}_n),$$

where $\tilde{\boldsymbol{C}}^{(j)} = \mathrm{diag}(\boldsymbol{F}_n\, \boldsymbol{c}^{(j)}) = \mathrm{diag}(\tilde{\boldsymbol{c}}^{(j)})$.

We observe that the elements in $\mathrm{circ}(\tilde{\boldsymbol{C}}^{(0:m-1)})$ are laid out in such a way that we actually have n interleaved circulant matrices of order m, e.g., the first circulant matrix is defined by the first elements of the diagonals $\tilde{\boldsymbol{c}}^{(j)}$, $j = 0, \ldots, m-1$, the second circulant matrix is defined by the second elements, etc... The next step is to exchange the single circ-operation and the m diag-operations with one diag-operation and n circ-operations. This can be done by the following permutation (as can easily be verified by a small example)

$$\mathrm{diag}(\tilde{\boldsymbol{C}}'^{(0:n-1)}) = \boldsymbol{P}_\sigma^\top\ \mathrm{circ}(\tilde{\boldsymbol{C}}^{(0:m-1)})\ \boldsymbol{P}_\sigma,$$

where $\sigma(i) = (i \bmod m)\, n + \lfloor i/m \rfloor$ and $\tilde{\boldsymbol{C}}'^{(k)} = \mathrm{circ}(\tilde{\boldsymbol{c}}_k^{(0:m-1)}) = \mathrm{circ}(\tilde{\boldsymbol{c}}'^{(k)})$.

The diagonalization of the complete matrix $\boldsymbol{C}$ can now be completed by applying n m-point Fourier transforms to these n circulant blocks. As such we find that

$$\boldsymbol{C} = (\boldsymbol{I}_m \otimes \boldsymbol{F}_n^{-1})\ \boldsymbol{P}_\sigma\ (\boldsymbol{I}_n \otimes \boldsymbol{F}_m^{-1})\ \tilde{\tilde{\boldsymbol{C}}}'\ (\boldsymbol{I}_n \otimes \boldsymbol{F}_m)\ \boldsymbol{P}_\sigma^\top\ (\boldsymbol{I}_m \otimes \boldsymbol{F}_n),$$

with $\tilde{\tilde{\boldsymbol{C}}}' = \mathrm{diag}(\boldsymbol{F}_m\ \tilde{\boldsymbol{c}}'^{(0:n-1)})$.

This can be reformulated in a computationally more efficient way when we consider a matrix $\boldsymbol{\mathcal{C}} \in \mathbb{R}^{n \times m}$ defined as

$$\boldsymbol{\mathcal{C}} = \left[\ \boldsymbol{c}^{(0)}\ \middle|\ \boldsymbol{c}^{(1)}\ \middle|\ \cdots\ \middle|\ \boldsymbol{c}^{(m-1)}\ \right] \qquad \sim \qquad \boldsymbol{C} = \mathrm{circ}(\boldsymbol{C}^{(0:m-1)}),$$

so that

$$\begin{aligned} \tilde{\boldsymbol{\mathcal{C}}} &= \boldsymbol{F}_n\, \boldsymbol{\mathcal{C}} && \sim && \mathrm{circ}(\tilde{\boldsymbol{C}}^{(0:m-1)}), \\ \tilde{\boldsymbol{\mathcal{C}}}' &= \tilde{\boldsymbol{\mathcal{C}}}^\top && \sim && \mathrm{diag}(\tilde{\boldsymbol{C}}'^{(0:n-1)}), \end{aligned}$$

and

$$\tilde{\tilde{\boldsymbol{\mathcal{C}}}}' = \boldsymbol{F}_m\, \tilde{\boldsymbol{\mathcal{C}}}' \qquad \sim \qquad \tilde{\tilde{\boldsymbol{C}}}' = \mathrm{diag}(\boldsymbol{F}_m\ \tilde{\boldsymbol{c}}'^{(0:n-1)}).$$

A matrix-vector product with such a 2-level block circulant matrix has an analog to Corollary 3.3 as

$$\boldsymbol{C}\, \boldsymbol{x} = (\boldsymbol{I}_m \otimes \boldsymbol{F}_n^{-1})\ \boldsymbol{P}_\sigma\ (\boldsymbol{I}_n \otimes \boldsymbol{F}_m^{-1})\ \tilde{\tilde{\boldsymbol{C}}}'\ (\boldsymbol{I}_n \otimes \boldsymbol{F}_m)\ \boldsymbol{P}_\sigma^\top\ (\boldsymbol{I}_m \otimes \boldsymbol{F}_n)\ \boldsymbol{x}.$$

From this follows that a matrix-vector product can be done in a fast way by considering $\boldsymbol{x}$ as an $n \times m$ matrix, denoted as $\boldsymbol{x}_{n \times m}$, in column order, so that the product can be calculated efficiently as

$$(\boldsymbol{C}\, \boldsymbol{x})_{n \times m} = \boldsymbol{F}_n^{-1}\ (\boldsymbol{F}_m^{-1}\ (\tilde{\tilde{\boldsymbol{C}}}' .* (\boldsymbol{F}_m\ (\boldsymbol{F}_n\ \boldsymbol{x}_{n \times m})^\top)))^\top,$$

where $.*$ means elementwise multiplication, e.g., $\mathrm{diag}(\boldsymbol{d})\ \boldsymbol{x} = \boldsymbol{d} .* \boldsymbol{x}$. This calculation has a preprocessing cost of $O(nm\log(nm))$ and a cost of $O(2nm\log(nm))$ per matrix-vector product (ignoring the constants of the FFTs).

If we now consider such matrices $\boldsymbol{C}$ to be embedded in another block circulant with ℓ blocks, and perform all the previous steps for each such matrix $\boldsymbol{C}$ then we arrive at a block circulant with diagonal blocks (at cost $O(\ell nm\log(nm))$). Again we can permute this form to a block diagonal matrix with circulant blocks, perform nm Fourier transforms of size ℓ (at cost $O(nm\ell\log(\ell))$) and we again arrive at a diagonal matrix (at cost $O(nm\ell\log(nm\ell))$).

The matrix-vector product can be handled in exactly the same way as for the 2-level case, with one extra level of FFTs and permutations. The diagonal multiplication is $O(nm\ell)$ and thus the total cost here is as expected $O(nm\ell\log(nm\ell))$.

The total cost for a matrix of size n which has k levels of circulant embeddings thus is $O(kn\log(n))$. The memory needed is the memory needed to store $\boldsymbol{\mathcal{C}}$ and $\boldsymbol{x}$ as well as their k-fold Fourier transforms, this is $O(n)$. □

It must be noted that the number of embedded circulant matrices in a $\boldsymbol{B}_d$ block is the number of distinct prime factors in n/d and thus $k = \kappa(n/d)$ and the size of such a $\boldsymbol{B}_d$ block is $\varphi(n/d)$ (again with the exceptional case when $8 \mid n$, where there is one extra level due to the isomorphic copy effect).

4.5 Illustrative examples

This section contains some examples which show the complete track of the previous sections in action. We will start with the trivial case for prime n. Since powers of 2 are an exceptional application we present such an example which also illustrates the concepts for other prime powers. Finally an example of more general nature will be considered.

Images of the matrix $\boldsymbol{\Xi}_n$ in nested block circulant configuration appear in Figures 4.1–4.3. Since such a matrix is filled with values from 0 up to $n-1$, there are n different colors per figure. On each row each color occurs only once, i.e., each row is a permutation of the first row of n colors. The matrices are organized as in Theorem 4.10 and so there is a vertical partition $\boldsymbol{A}_d$ for each divisor d of n, grouped as given by Lemma 4.13. The start of these partitions is marked with an arrow which has a text label to denote which d generates this partition. In each vertical partition we have nested repetitive blocks $\boldsymbol{B}_d$, the repetitions of this block are drawn with faded colors to make clear how this block is distributed (cf. Corollary 4.11).

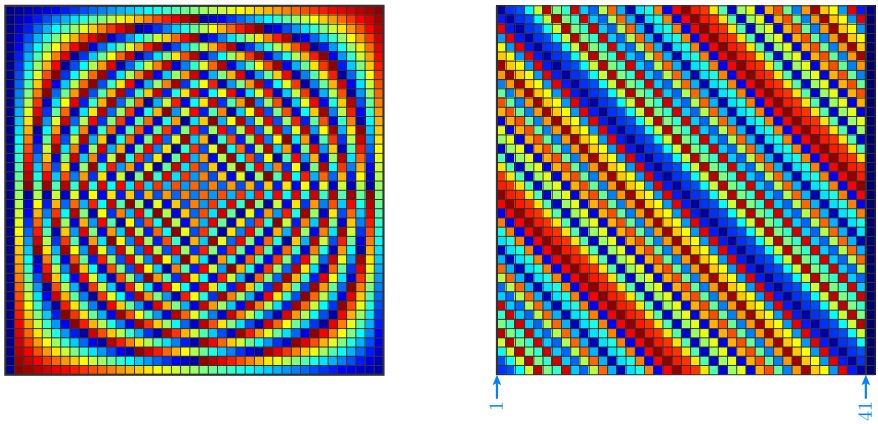


Figure 4.1: Example matrix for $n = 41$, left: natural order, right: generator order. For $n = 41$ there are only two trivial divisors, and thus there are two circulant blocks, one of size $\varphi(41) \times \varphi(41) = 40 \times 40$ and one of 1×1 which gets replicated to fill up the 40 rows of the matrix.

4.5.1 Prime n

The result for prime n from Chapter 3 can now compactly be restated as follows. If n equals a prime p, the divisors are simply

$$\text{divisors}(p) = \{1, p\}.$$

We obtain the trivial partition $\mathbb{Z}_p = 1\,U_p \cup p\,U_1 = U_p \cup \{0\}$. The index-matrix has thus two very simple vertical partitions $\boldsymbol{A}_1$ and $\boldsymbol{A}_p$:

$$\begin{aligned} \boldsymbol{A}_1 &= \mathbf{1}_{1\times 1} \otimes \boldsymbol{B}_1, & \boldsymbol{B}_1 &= [k \cdot z]_{z,k\in U_p}, \\ \boldsymbol{A}_p &= \mathbf{1}_{p-1\times 1} \otimes \boldsymbol{B}_p, & \boldsymbol{B}_p &= [0], \end{aligned}$$

of which $\boldsymbol{B}_1$ is isomorphic to a circulant matrix of size $\varphi(n)$.

An example of the structure for $p = 41$ is given in Figure 4.1. On the left the matrix is drawn in its natural ordering, while on the right the matrix is drawn in the ordering which allows a fast matrix-vector product. The last partition is the partition for $d = 41$ where the complete column is constant; only the first element in this column is drawn at full color, the rest of the column is faded to denote its redundancy.

4.5.2 Powers of 2 (and other prime powers)

For powers of a prime the divisors are

$$\text{divisors}(p^k) = \{p^\ell : 0 \le \ell \le k\},$$

resulting in circulant matrices $\boldsymbol{B}_d$ (being block circulant with 2 levels for a power of 2), which have regularly diminishing sizes

$$\begin{aligned} |B_{p^\ell}| &= \varphi(p^{k-\ell}) \\ &= \begin{cases} p^{k-\ell-1}(p-1) & \text{if } \ell < k, \\ 1 & \text{otherwise.} \end{cases} \end{aligned}$$

Figure 4.2 shows $\boldsymbol{\Xi}_n$ for $n = 2^6 = 64$. In this figure it is clearly visible that increasing powers of a prime have the effect of overlapping in a nice way. This effect is used in Lemma 4.13 to sum the result vectors (for general n). Assume we have calculated the 7 result vectors $\boldsymbol{v}_d$ for $n = 64$, then the summing order from Lemma 4.13 gives

$$\boldsymbol{v} = \boldsymbol{v}_1 + (\boldsymbol{v}_2 + (\boldsymbol{v}_4 + (\boldsymbol{v}_8 + (\boldsymbol{v}_{16} + \underbrace{(\boldsymbol{v}_{32} + \boldsymbol{v}_{64})}_{\text{1 addition}}}_{\text{+2 additions}}}_{\text{+4 additions}}}_{\text{+8 additions}}}_{\text{+16 additions}}}_{\text{+32 additions}}))))$$

resulting in a total of $1 + 2 + 4 + 8 + 16 + 32 = 64 - 1 = 63$ which can be verified on the figure. Naive summing would give $6 \times 32 = 192$ operations.

Note however that for powers of a prime it is not really necessary to use this summing order. Since as was worked out just before Lemma 4.13, $d(p^k) = k+1 = O(\log(n))$ and $\varphi(p^k) = p^{k-1}(p-1) = O(n)$, and thus naive summing also has the correct asymptotic cost of $O(n \log(n))$.

Also visible in the figure is the isomorphic copy effect when a power of 2 is involved (mentioned in Corollary 4.6). This can be used nicely for component-by-component construction since the kernel function $\omega(x)$ is symmetric around $1/2$, i.e., $\omega(x) = \omega(1-x)$. As was shown in Theorem 3.6, for prime n this symmetry has the effect that only half the space of z-candidates has to be searched and only one quarter of the $\boldsymbol{\Omega}_n$ matrix has to be considered. Similar effects occur for n which are not prime, for a power of 2 this means that the isomorphic copy can be left out, in other words, we get circulant matrices instead of block circulant matrices for free.

4.5.3 For general n

To get a better view of the interleaving of the matrices $\boldsymbol{B}_d$ as given in Corollary 4.11 we must of course consider more general n. Figure 4.3 shows $\boldsymbol{\Xi}_n$ for $n = 3 \times 5 \times 7 =$

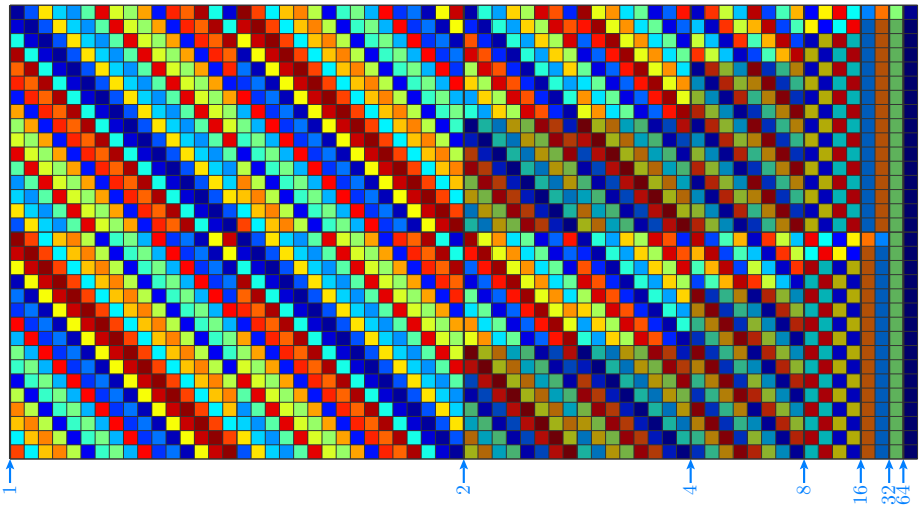


Figure 4.2: Example matrix for $n = 2^6 = 64$. Since this is a power of 2, every divisor block contains a block-circulant which is clearly visible as four quadrants. Since $\omega(x) = \omega(1-x)$ only the top-left quadrant has to be used for matrix-vector multiplication in the component-by-component algorithm.

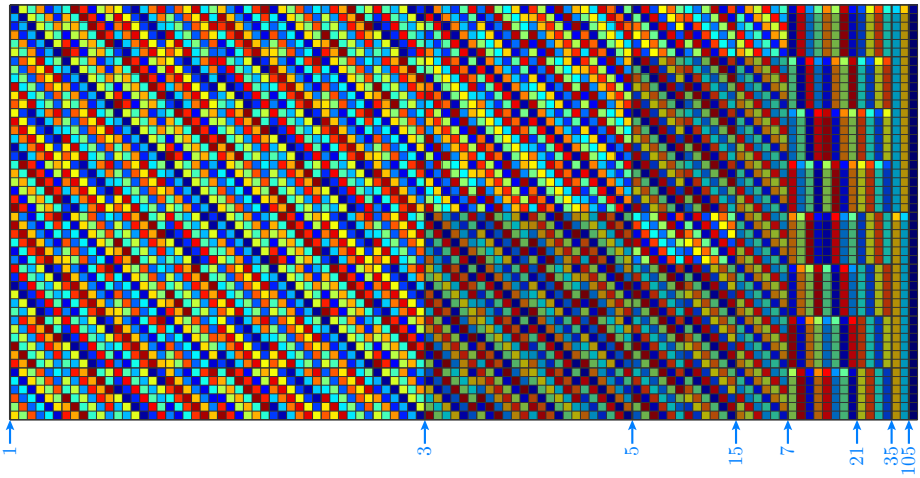


Figure 4.3: Example matrix for $n = 3 \times 5 \times 7 = 105$. On this figure the nesting of the smaller (block-)circulant matrices becomes visible, e.g., for the divisor 7.

105. Clearly visible in the part for $d = 1$, is the nested block circulant structure with 3 levels. At the highest level we see a circulant structure with $\varphi(3) = 2$ blocks of size $\varphi(5)\varphi(7) = 24$, in each such block we see again a circulant structure with $\varphi(5) = 4$ blocks of size $\varphi(7) = 6$, and these blocks in their turn are circulant matrices of order 6.

The summing as given by Lemma 4.13 here gives:

$$\boldsymbol{v} = \underbrace{\underbrace{\underbrace{(\boldsymbol{v}_1 + \boldsymbol{v}_3)}_{48\text{ additions}} + \underbrace{(\boldsymbol{v}_5 + \boldsymbol{v}_{15})}_{+12\text{ additions}}}_{+48\text{ additions}} + \underbrace{\underbrace{(\boldsymbol{v}_7 + \boldsymbol{v}_{21})}_{+8\text{ additions}} + \underbrace{(\boldsymbol{v}_{35} + \boldsymbol{v}_{105})}_{+2\text{ additions}}}_{+8\text{ additions}}}_{+48\text{ additions}}$$

which makes a total of $(48 + 12 + 8 + 2) + (48 + 8) + (48) = (7 \times 5 \times (3 - 1)) + (7(5 - 1)\varphi(3)) + ((7 - 1)\varphi(5)\varphi(3)) = 174$ additions and can again be verified on the figure. Naive summing would involve $7 \times 48 = 336$ operations.

Finally, we give a smaller textual example for $n = 2 \times 3 \times 5 = 30$ which has

$$\text{divisors}(30) = \{1, 2, 3, 5, 6, 10, 15, 30\}.$$

Or already put into the order of Lemma 4.13 we get

$$[\,1,\ 2,\ 3,\ 6,\ 5,\ 10,\ 15,\ 30\,] = \\ [\,5^0 3^0 2^0,\ 5^0 3^0 2^1,\ 5^0 3^1 2^0,\ 5^0 3^1 2^1,\ 5^1 3^0 2^0,\ 5^1 3^0 2^1,\ 5^1 3^1 2^0,\ 5^1 3^1 2^1\,].$$

The biggest block is as always for $d = 1$ and contains here the elements of $U_{30} \simeq U_2 \oplus U_3 \oplus U_5 = \langle 1 \rangle_2 \oplus \langle 2 \rangle_3 \oplus \langle 2 \rangle_5$. We have fixed the generators for the cyclic groups to 1, 2 and 2. With the help of the Chinese remainder theorem we find the correct ordering of the indices for the rows as

1. $(1, 1, 1) \simeq 1$,
2. $(1, 1, 2) \simeq 7$,
3. $(1, 1, 4) \simeq 19$,
4. $(1, 1, 3) \simeq 13$,
5. $(1, 2, 1) \simeq 11$,
6. $(1, 2, 2) \simeq 17$,
7. $(1, 2, 4) \simeq 29$,
8. $(1, 2, 3) \simeq 23$.

The ordering of the columns can easily be found by reversing the sequences per prime component except for the first element of each sequence. Using this

generator ordering we can write $\boldsymbol{A}_1 = \boldsymbol{B}_1$ as

$$\begin{bmatrix} 1 & 13 & 19 & 7 & 11 & 23 & 29 & 17 \\ 7 & 1 & 13 & 19 & 17 & 11 & 23 & 29 \\ 19 & 7 & 1 & 13 & 29 & 17 & 11 & 23 \\ 13 & 19 & 7 & 1 & 23 & 29 & 17 & 11 \\ 11 & 23 & 29 & 17 & 1 & 13 & 19 & 7 \\ 17 & 11 & 23 & 29 & 7 & 1 & 13 & 19 \\ 29 & 17 & 11 & 23 & 19 & 7 & 1 & 13 \\ 23 & 29 & 17 & 11 & 13 & 19 & 7 & 1 \end{bmatrix}$$

Note that this is a block circulant with 2 levels. We skip the second divisor $d = 2$, which would give a block as big as the first one since $\varphi(2) = 1$. For $d = 3$ we obtain $\boldsymbol{B}_3$ as

$$\boldsymbol{B}_3 = \left[3 \cdot k \cdot z\right]_{\substack{z \in U_{10} \\ k \in U_{10}}} \simeq \begin{bmatrix} 3 & 9 & 27 & 21 \\ 21 & 3 & 9 & 27 \\ 27 & 21 & 3 & 9 \\ 9 & 27 & 21 & 3 \end{bmatrix}.$$

We now work out the permutation and replication matrix $\boldsymbol{R}_3$ by looking at its tensor product components for each prime factor (for clarity we index with the actual prime instead of with the index of the prime):

$$\boldsymbol{R}_3 = \otimes_{p=2,3,5} \boldsymbol{R}_{3,p},$$

where

$$\boldsymbol{R}_{3,2} = \left[1\right] \otimes \left[1\right] = 1, \qquad \boldsymbol{R}_{3,3} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \left[1\right] = \mathbf{1}_{2\times 1}, \qquad \boldsymbol{R}_{3,5} = \left[1\right] \otimes \boldsymbol{I}_4 = \boldsymbol{I}_4,$$

and thus $\boldsymbol{R}_3 = \mathbf{1}_{2\times 1} \otimes \boldsymbol{I}_4$ and we thus get two stacked copies of $\boldsymbol{B}_3$ on top of each other. As a final example let's look at $d = 10$. There we have

$$\boldsymbol{B}_{10} \simeq \begin{bmatrix} 10 & 20 \\ 20 & 10 \end{bmatrix},$$

and we find

$$\boldsymbol{R}_{10,2} = 1, \qquad \boldsymbol{R}_{10,3} = \boldsymbol{I}_2, \qquad \boldsymbol{R}_{10,5} = \mathbf{1}_{4\times 1},$$

so that $\boldsymbol{R}_{10} = \boldsymbol{I}_2 \otimes \mathbf{1}_{4\times 1}$. In other words replicate the first row of $\boldsymbol{B}_{10}$ 4 times and then replicate the second row 4 times. A full view on the matrix is presented in Figure 4.4.

4.6 Comparison with "Partial search"

In [22] and [23], Dick & Kuo presented an adaptation of the component-by-component algorithm, called "Partial search", by using composite n having 2, 3, 4 and 5 factors. In their adaptation the worst-case error for each of the factors is calculated separately (by averaging over the components to be optimized) and then the optimal z_i are combined using the Chinese remainder theorem. From the analysis here we expect that their calculated errors differ from the true worst-case error probably only in a small amount, since the divisors they left out are large and thus would only give small blocks $\boldsymbol{B}_d$.

Also from those papers and [21] it is known that prime n have lower worst-case errors and thus are preferable over composite n. This is confirmed by our numerical experiments [83] (see §3.4.2). This clearly lowers the interest in implementing such a general n routine as presented in this chapter. However, the prime power case will be revisited in the next chapter since they provide the possibility of using the lattice rule point by point as a sequence.

Only preliminary testing has been done for more general n, but from the results in this chapter it should be clear that an implementation for prime n will probably be the most efficient (contrasting the results from [21, 22, 23] due to the fast construction). Especially n which have a large number of unique prime factors will slow the algorithm down because of the more complicated (nested block) circulant matrix-vector calculations involved.

4.7 A graphical view on the permutations

During the preparation of [85] (on which this chapter builds up to this section) a collection of number theory routines were developed in Python to investigate the permutations for non-prime n. The choice for Python can be motivated by the fact that it has arbitrary precision integers and a built-in modular exponentiation routine. These prove useful in implementing number theory algorithms. For numerical and matrix calculations, the availability of the Numerical and Scientific Python packages, which provide a Matlab like environment, is also a very big help.

The excellent PyX package (for the creation of Postscript and PDF publication quality graphics) then produced colorful images of the matrix $\boldsymbol{\Xi}_n$ for any n. This section shows some examples of these images. Figure 4.8 was used for the cover of the December 2005 AMS Notices and included a short explanation [81].

Figures 4.6–4.8 show the permutations on the matrix in three steps. The bottom matrices show the natural ordering of U_n and $\mathbb{Z}_n$. For the middle matrices, only the columns are permuted by grouping $\mathbb{Z}_n$ on the divisors of n. Finally, the top matrices show the number theoretic permutations for each divisor block, giving nested block circulant structures. Again, grayed out blocks denote the redundant parts for the matrix-vector product in the component-by-component algorithm.

1	13	19	7	11	23	29	17	2	26	8	14	22	16	28	4	3	9	27	21	6	18	24	12	5	25	10	20	15	0
7	1	13	19	17	11	23	29	14	2	26	8	4	22	16	28	21	3	9	27	12	6	18	24	*5*	*25*	*10*	*20*	*15*	*0*
19	7	1	13	29	17	11	23	8	14	2	26	28	4	22	16	27	21	3	9	24	12	6	18	*5*	*25*	*10*	*20*	*15*	*0*
13	19	7	1	23	29	17	11	26	8	14	2	16	28	4	22	9	27	21	3	18	24	12	6	*5*	*25*	*10*	*20*	*15*	*0*
11	23	29	17	1	13	19	7	22	16	28	4	2	26	8	14	*3*	*9*	*27*	*21*	*6*	*18*	*24*	*12*	25	5	20	10	*15*	*0*
17	11	23	29	7	1	13	19	4	22	16	28	14	2	26	8	*21*	*3*	*9*	*27*	*12*	*6*	*18*	*24*	*25*	*5*	*20*	*10*	*15*	*0*
29	17	11	23	19	7	1	13	28	4	22	16	8	14	2	26	*27*	*21*	*3*	*9*	*24*	*12*	*6*	*18*	*25*	*5*	*20*	*10*	*15*	*0*
23	29	17	11	13	19	7	1	16	28	4	22	26	8	14	2	*9*	*27*	*21*	*3*	*18*	*24*	*12*	*6*	*25*	*5*	*20*	*10*	*15*	*0*

Figure 4.4: Textual view on $\boldsymbol{\Xi}_{30}$ (italic font denotes redundancy). See also Figure 4.5.

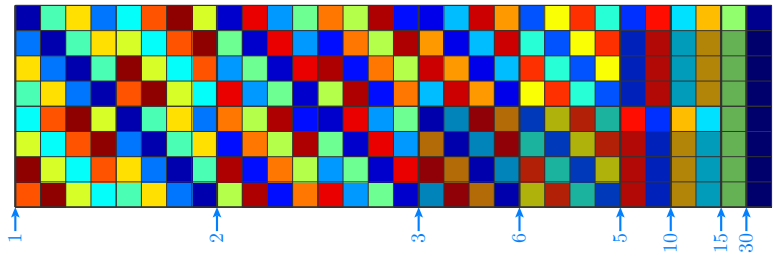


Figure 4.5: Full view on the structure of $\boldsymbol{\Xi}_{30}$ as an image.

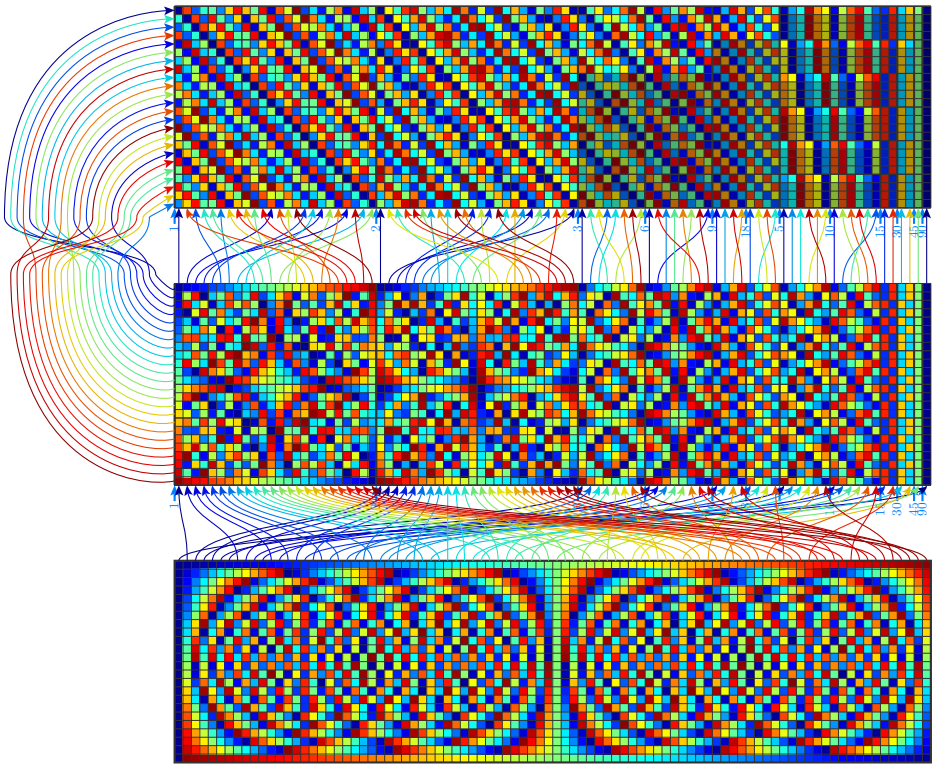


Figure 4.6: Permutations for $n = 2 \times 3^3 \times 5 = 90$.

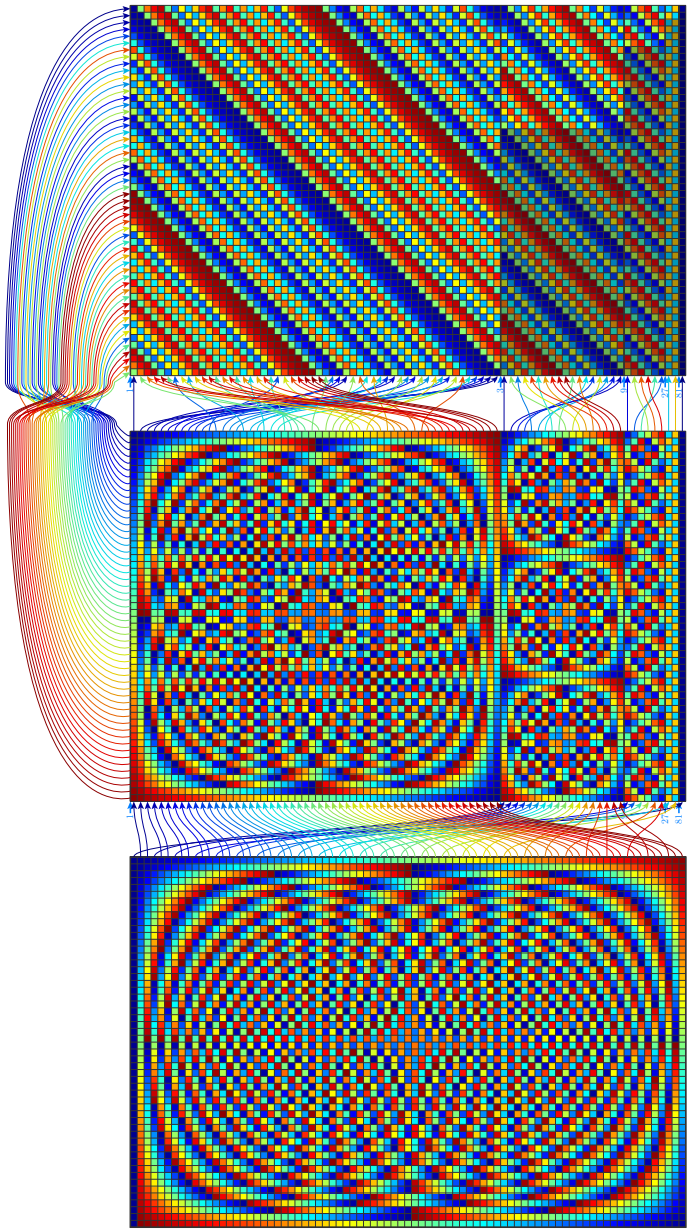


Figure 4.7: Permutations for $n = 3^4 = 81$. Odd prime powers give a nice regular repeating pattern with blocks of diminishing sizes.

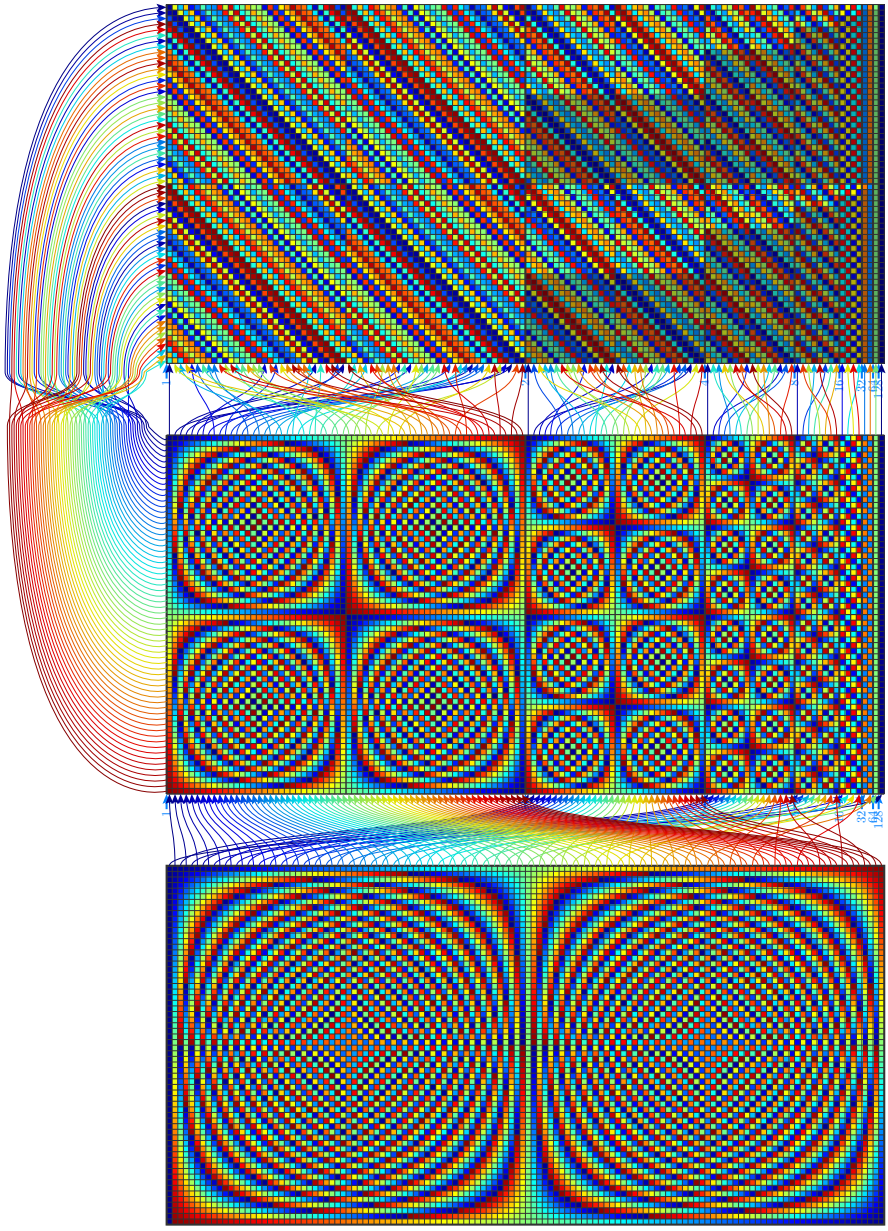


Figure 4.8: Permutations for $n = 2^7 = 128$. As is usual in number theory, powers of 2 (the only even prime) behave a little bit different than powers of odd primes, except for $n = 2$ and $n = 4$ which are the usual exceptions. In this case we get an extra circulant embedding in comparison with an odd prime power as in Figure 4.7.

4.8 Full proof of Lemma 4.9

In this section a more elaborate proof of Lemma 4.9 is given. Here the bits and pieces for the most general result are proven in turn.

(i) First we prove the following trivial result

$$d\,\mathbb{Z}_n \bmod n = d\,\mathbb{Z}_{n/d} \bmod n, \qquad \text{where } d \mid n. \tag{4.7}$$

A simple proof to start with:

Proof. If $d \mid n$ then $n = d\,d'$, with d and $d' \in \mathbb{Z}_n \setminus \{0\}$ (and $n \neq 0$). Thus:

$$\begin{aligned} d\,\mathbb{Z}_n \bmod n &= \{d \cdot v \bmod n : v \in \mathbb{Z}_n\} \\ &= \{0d, 1d, 2d, \ldots, (d'-1)d, \\ &\quad\; d'd, (d'+1)d, (d'+2)d, \ldots, \ldots, (dd'-1)d\} \bmod n \\ &= \{0d, 1d, 2d, \ldots, n-d, n, n+1d, n+2d, \ldots, \ldots, dn-d\} \bmod n \\ &= \{0d, 1d, 2d, \ldots, -d, 0, 1d, 2d, \ldots, \ldots, -d\} \bmod n \\ &= \{0d, 1d, 2d, \ldots, -d\} \bmod n \\ &= d\,\mathbb{Z}_{n/d} \bmod n = d\,\mathbb{Z}_{d'} \bmod n. \end{aligned}$$

□

(ii) Now we prove the easiest case:

$$w\,\mathbb{Z}_n \bmod n = w\,\mathbb{Z}_{n/g} \bmod n, \qquad \text{with } g = \gcd(w, n) \text{ and } w \in \mathbb{Z}.$$

Proof. With no loss of generality take $w \in \mathbb{Z}_n$ since the multiplication is modulo n. We get:

$$w\,\mathbb{Z}_n \bmod n = \{w \cdot v \bmod n : v \in \mathbb{Z}_n\}.$$

Set $g = \gcd(w, n)$, such that $g \mid w$ and $g \mid n$, and thus $w = w/g \cdot g$. Then we have

$$w\,\mathbb{Z}_n \bmod n = \{w/g \cdot (g \cdot v) \bmod n : v \in \mathbb{Z}_n\}.$$

Now use the trivial result (4.7): since $g \mid n$ we can replace the $\mathbb{Z}_n$ with $\mathbb{Z}_{n/g}$. Thus

$$\begin{aligned} w\,\mathbb{Z}_n \bmod n &= \{w/g \cdot g \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= \{w \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= w\,\mathbb{Z}_{n/g} \bmod n, \end{aligned} \tag{4.8}$$

where w/g and g were joined to get w. □

(iii) A small modification to get g in front instead of w:

$$w\,\mathbb{Z}_n \bmod n = g\,\mathbb{Z}_{n/g} \bmod n, \qquad \text{with } g = \gcd(w,n) \text{ and } w \in \mathbb{Z}. \tag{4.9}$$

Proof. We start from the previous proven result (4.8):

$$\begin{aligned} w\,\mathbb{Z}_n \bmod n &= \{w \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= \{g \cdot (w/g \cdot v) \bmod n : v \in \mathbb{Z}_{n/g}\}. \end{aligned}$$

It suffices to consider w modulo n, thus $w \in \mathbb{Z}_n$. It then follows that $w/g \in \mathbb{Z}_{n/g}$ since $g \mid n$ and $g \mid w$. Also since $w/g \cdot v \bmod n$ gets multiplied by g we could take the product between parentheses as $w/g \cdot v \bmod n/g$:

$$w\,\mathbb{Z}_n \bmod n = \{g \cdot (w/g \cdot v \bmod n/g) \bmod n : v \in \mathbb{Z}_{n/g}\}.$$

Now $\gcd(w/g, n/g) = 1$ (since $\gcd(w,n) = g$) and thus $w/g \in U_{n/g}$. Multiplication by a unit leaves the set as is, i.e., we can leave the term w/g out. Thus

$$\begin{aligned} w\,\mathbb{Z}_n \bmod n &= \{g \cdot v \bmod n : v \in \mathbb{Z}_{n/g}\} \\ &= g\,\mathbb{Z}_{n/g} \bmod n. \end{aligned}$$

□

(iv) The $\mathbb{Z}_n$ could be replaced by U_n:

$$w\,U_n \bmod n = w\,U_{n/g}, \qquad \text{with } g = \gcd(w,n) \text{ and } w \in \mathbb{Z}.$$

Proof. The proof is similar to that for (4.9). We only have to add an extra constraint.

$$\begin{aligned} w\,U_n \bmod n &= \{w \cdot v \bmod n : v \in U_n\} \\ &= \{w/g \cdot (g \cdot v) \bmod n : v \in \mathbb{Z}_n \text{ and } \gcd(v,n) = 1\} \\ &= \{w/g \cdot (g \cdot v) \bmod n : v \in \mathbb{Z}_{n/g} \text{ and } \gcd(v,n) = 1\}. \end{aligned}$$

From $\gcd(v,n) = 1$ follows also that $\gcd(v, n/g) = 1$ since $g \mid n$, and thus

$$\begin{aligned} w\,U_n \bmod n &= \{w/g \cdot g \cdot v \bmod n : v \in \mathbb{Z}_{n/g} \text{ and } \gcd(v,n/g) = 1\} \\ &= \{w \cdot v \bmod n : v \in \mathbb{Z}_{n/g} \text{ and } \gcd(v,n/g) = 1\} \\ &= \{w \cdot v \bmod n : v \in U_{n/g}\} \\ &= w\,U_{n/g}. \end{aligned}$$

□

(v) Again the g can be put upfront like

$$w\,U_n \bmod n = g\,U_{n/g}, \qquad \text{with } g = \gcd(w,n) \text{ and } w \in \mathbb{Z}.$$

Proof. We start from the result of the previous proof (with the w upfront).

$$\begin{aligned} w\,U_n \bmod n &= \{w \cdot v \bmod n : v \in U_{n/g}\} \\ &= \{g \cdot (w/g \cdot v) \bmod n : v \in U_{n/g}\}. \end{aligned}$$

Now consider w modulo n, thus $w \in \mathbb{Z}_n$. It then follows that $w/g \in \mathbb{Z}_{n/g}$ since $g \mid n$ and $g \mid w$. Also since $w/g \cdot v \bmod n$ gets multiplied by g we could take the product between parentheses as $w/g \cdot v \bmod n/g$:

$$w\,U_n \bmod n = \{g \cdot (w/g \cdot v \bmod n/g) \bmod n : v \in U_{n/g}\}.$$

Now $\gcd(w/g, n/g) = 1$ (since $\gcd(w,n) = g$) and thus $w/g \in U_{n/g}$ and multiplication by a unit leaves the set as is, i.e., we can leave the term w/g out. Thus

$$\begin{aligned} w\,U_n \bmod n &= \{g \cdot v \bmod n : v \in U_{n/g}\} \\ &= g\,U_{n/g} \bmod n. \end{aligned}$$

□

(vi) The last thing to prove is

$$w\,U_n \bmod n \simeq \oplus_i \left(\tilde{g}_i\,U_{n_i/\tilde{g}_i} \bmod n_i\right), \quad \text{with } \tilde{g}_i = \gcd(w_i,n_i) = \gcd(w,n_i) \text{ and } w \in \mathbb{Z}. \tag{4.10}$$

Proof. Consider the prime factorization of $n = \prod_i n_i = \prod_i p_i^{k_i}$ and take w in the residue number system as $w \simeq (w_1, w_2, \ldots, w_r) = (w \bmod n_1, w \bmod n_2, \ldots, w \bmod n_r)$. Then we have

$$w\,U_n \bmod n \simeq \oplus_i (w_i\,U_{n_i} \bmod n_i).$$

Now apply the previous result $w\,U_n \bmod n = g\,U_{n/g}$ with $g = \gcd(w,n)$ for each component:

$$w\,U_n \bmod n \simeq \oplus_i (\tilde{g}_i\,U_{n_i/\tilde{g}_i} \bmod n_i),$$

with $\tilde{g}_i = \gcd(w_i,n_i) = \gcd(w \bmod n_i, n_i) = \gcd(w,n_i)$. At this point the result (4.10) is already proven. But we now show the relation with $g = \gcd(w,n)$. Pull the $\tilde{g}_i$ out by using the Chinese remainder theorem and construct $\tilde{g} \simeq (\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_r)$:

$$\begin{aligned} w\,U_n \bmod n &= \tilde{g}\ \text{chinese}(\oplus_i U_{n_i/\tilde{g}_i}, \oplus_i n_i/\tilde{g}_i) \bmod n \\ &= \tilde{g}\,U_{n/\bar{g}} \bmod n. \end{aligned} \tag{4.11}$$

Here $\bar{g} = \prod_i \tilde{g}_i$ and chinese($\{\cdots\}, \{\cdots\}$) denotes to apply the Chinese remainder theorem to solve the system of congruences.
At (4.11) we should mention that $\tilde{g} \simeq (\gcd(w, n_1), \ldots, \gcd(w, n_r))$ is mostly *not equal* to $g = \gcd(w, n)$. We do of course have that $\gcd(\tilde{g}, n) = g = \gcd(w, n)$ (this follows from the previous proofs). In addition we need that

$$\bar{g} = \prod_i \tilde{g}_i = g,$$

which follows immediately since gcd is a multiplicative function. Note, that this is the property which tells us that if $u \in U_n$ then

$$\begin{aligned} u &\simeq (u \mod n_1, u \mod n_2, \ldots, u \mod n_r) \\ &\simeq (u_1, u_2, \ldots, u_r), \qquad \text{with all } u_i \in U_{n_i}. \end{aligned}$$

Since if $u \in U_n$ then $\gcd(u, n) = 1 = \prod_i \tilde{g}_i$, with all $\tilde{g}_i = \gcd(u_i, n_i)$. This can only be one if all individual $\tilde{g}_i$ are one, and thus we have that all $\gcd(u_i, n_i) = \tilde{g}_i = 1$. □

Chapter notes

Fast component-by-component construction for rank-1 lattice rules with a non-prime number of points was first presented at the Dagstuhl workshop "Algorithms and Complexity for Continuous Problems" 2004. The results were later published in [85]. The methods used in Lemma 4.14 are strongly connected to the group structure used in the rest of this chapter. The 2-level case directly relates to the prime factor framework for the FFT, see [107, §4.1], here applied for a convolution-like multiplication. The graphical view on the permutations [81] accompanied the nice article on the complexity side of the story by Kuo and Sloan [64].

Chapter 5

Lattice sequences

The fast constructions in the previous chapters deliver lattice rules which achieve the optimal rate of convergence for suitably weighted function spaces. Since the construction is component-by-component, it is possible to extend these lattice rules in the number of dimensions. However, the generating vector and the number of points come in pairs: a given generating vector is not guaranteed to be good for any number of points different from the one it was constructed for.

Existence of rules that are both extensible in the number of dimensions and the number of points was proven by Hickernell and Niederreiter [47]. Their proof is nonconstructive and leaves no clue how to find such a generating vector. Moreover, since the number of points is unbounded, the components of the generating vector can be arbitrary large.

This chapter deals with lattice rules which can be used for any number of dimensions and any number of points up to a given maximum. Fast construction of such rules is made possible by the structure which was revealed in the previous chapter.

5.1 Extensible lattice sequences

In 2000 Hickernell et al. [46] experimented with the construction of extensible lattice sequences in an empirical setting. By assuming that

$$n = b^m, \quad b \geq 2, \quad m_1 \leq m \leq m_2, \quad \text{and} \quad 1 \leq s \leq s_{\max},$$

they designed an algorithm which finds a generating vector of the Korobov form, see (2.33), with the parameter a chosen by minimizing a loss function

$$G_{m_1,m_2,s_{\max}}(a) := \max_{\substack{m_1 \leq m \leq m_2 \\ 1 \leq s \leq s_{\max}}} \frac{D_{b^m,s}(a)}{D^{(\mathrm{asy})}_{b^m,s}}.$$

Here $D_{b^m,s}(a)$ is some discrepancy measure for an s-dimensional b^m-point lattice rule with Korobov parameter a, and $D^{(\mathrm{asy})}_{b^m,s}$ is some asymptotically derived normalization chosen to ensure that for various values of m and s the discrepancy $D_{b^m,s}(a)$ is scaled appropriately for comparison.

The rules from [46] are by construction not really extensible rules, but they are embedded rules. With the powers of b, a sequence of point sets is associated:

$$\{L_m\}_{m_1 \le m \le m_2}.$$

Each L_m is the point set of a lattice rule with b^m points, and all the points in L_m are also points of L_{m+1}. This embedding property is very attractive in practice because the number of points can easily be increased without wasting any function evaluations and the error can be estimated based on the results from successive rules in the sequence.

5.2 Component-by-component construction of embedded lattice sequences

The algorithm presented here is similar to that of [46], but rather than using the Korobov form, it is based on component-by-component constructions and so there is no need to assume that s is bounded. For each m from m_1 to m_2, the standard component-by-component algorithm is used to find a generating vector $\boldsymbol{z}^{(m)}$ for a b^m-point rule and the resulting worst-case error is stored in $e_{b^m,s}(\boldsymbol{z}^{(m)})$ for each s. Now define the error measure

$$X_{m_1,m_2,s}(\boldsymbol{z}) := \max_{m_1 \le m \le m_2} \frac{e_{b^m,s}(\boldsymbol{z})}{e_{b^m,s}(\boldsymbol{z}^{(m)})}, \tag{5.1}$$

where $e_{b^m,s}(\boldsymbol{z}^{(m)})$ is used as the reference error to provide a good normalization. The component-by-component algorithm for extensible lattice sequences is then given in Algorithm 5.

Algorithm 5 Component-by-component for extensible lattice sequences

for $s = 1$ **to** $s_{\max}$ **do**
  for all $z_s \in U_{b^{m_2}}$ **do**
    $X_{m_1,m_2,s}(z_s) = \max_{m_1 \le m \le m_2} \dfrac{e_{b^m,s}(\boldsymbol{z})}{e_{b^m,s}(\boldsymbol{z}^{(m)})}$
  end for
  $z_s = \operatorname*{argmin}_{z \in U_{b^{m_2}}} X_{m_1,m_2,s}(z)$
end for

Just as the algorithm from [46], this algorithm does not have an a priori error bound, nor does it give fully extensible lattice sequences in the number of points. But we improve upon [46] in the following ways:

(i) The component-by-component construction leaves s unbounded, i.e., the rules are fully extensible in the dimension.

(ii) For suitably weighted function spaces it is known that the component-by-component algorithm constructs rules which have nearly the optimal rate of convergence, $O(n^{-1+\delta})$, $\delta > 0$, see [62, 21]. Therefore the normalization in (5.1) is reliable (and not asymptotic as in [46]). Moreover, the quantity $X_{m_1,m_2,s}(\boldsymbol{z})$ represents how much worse the chosen vector $\boldsymbol{z}$ is compared with the vectors $\boldsymbol{z}^{(m)}$ which are the near-optimal choices for different $n = b^m$. So if $X_{m_1,m_2,s}(\boldsymbol{z})$ is bounded by a small constant we have an a posteriori worst-case error bound. In the experiments given in §5.6 the value of $X_{10,20,s}(\boldsymbol{z}^\star)$ is never higher than 1.60 for different function spaces, see Table 5.1.

(iii) The total computational cost of Algorithm 5, including the precomputations is $O(sn(\log(n))^2)$ operations using the fast construction algorithm. When taking b prime it follows from Chapter 4 that the matrix $\boldsymbol{\Omega}_{b^m}$ has a very nice structure which can also be exploited.

5.3 Properties of embedded lattice rules

Before we get into the ideas behind the fast implementation, we need to review some fundamental properties of embedded lattice rules.

Given an integer base $b \geq 2$, a fixed dimension s, and any integer vector $\boldsymbol{z} \in \mathbb{Z}^s$, with $\gcd(z_j, b) = 1$ for all $j = 1, \ldots, s$, define

$$L_m = L_m(\boldsymbol{z}) := \left\{ \left\{ \frac{k\boldsymbol{z}}{b^m} \right\} : 0 \leq k \leq b^m - 1 \right\}, \quad m = 0, 1, \ldots . \tag{5.2}$$

The following properties hold for all $m \geq 0$.

Property 1. L_m is a point set for a rank-1 lattice rule with b^m distinct points.

Property 2. The generating vector for the lattice point set L_m can be reduced modulo b^m to give the same point set:

$$L_m(\boldsymbol{z}) = L_m(\boldsymbol{z} \bmod b^m).$$

Property 3. The lattice point set L_m is embedded in L_{m+1}:

$$L_0 \subset L_1 \subset \cdots \subset L_m \subset L_{m+1} \subset \cdots .$$

Property 4. The points in L_m are exactly those points in L_{m+1} whose indices k in (5.2) are multiples of b:

$$L_{m+1} \cap L_m = \left\{ \left\{ \frac{k\boldsymbol{z}}{b^{m+1}} \right\} : k \in b\,\mathbb{Z}_{b^{m+1}} \right\} = L_m.$$

For a given m, a generating vector $\boldsymbol{z} \in U_{b^m}^s$ can be constructed using the fast component-by-component construction with $n = b^m$ so that L_m is a "good" lattice point set with b^m points. Note that the goodness of the point set L_m in no way implies that all smaller lattice point sets embedded in L_m are good. On the contrary, it is quite possible that the same generating vector $\boldsymbol{z}$ leads to a very poor point set L_ℓ for $\ell < m$ (as will be demonstrated in §5.6).

This brings us to Algorithm 5, where a generating vector $\boldsymbol{z}$ is constructed with the hope that the lattice point sets L_m, for all values of m in a finite range $m_1 \le m \le m_2$, would be good simultaneously. The assumption that m is bounded from above by m_2 means that the maximum number of points is b^{m_2}; hence we can restrict the components of $\boldsymbol{z}$ to be in $U_{b^{m_2}}$, ensuring a finite search space. The quality of $\boldsymbol{z}$ as a "one for all" vector is ensured as long as $X_{m_1,m_2,s}(\boldsymbol{z})$ is uniformly bounded by a small constant, say, 2.

5.4 Fast construction of embedded lattice rules

In this section it will be shown that the structure of the matrix $\boldsymbol{\Omega}_n$, which was studied in the previous chapter, reveals that also the calculations of the worst-case errors are embedded when n is a prime power. So in calculating $e_{p^{m_2},s}(\boldsymbol{z})$ also all worst-case errors $e_{p^m,s}(\boldsymbol{z})$ for $m \le m_2$ can be calculated at the same time. This means that, with a minor modification to the fast algorithm, the actual cost of the bare Algorithm 5 is only $O(sn\log(n))$, with $n = p^{m_2}$, while the cost of the precomputations using fast construction is $O(sn(\log(n))^2)$. In other words, the embedded rule comes almost for free.

5.4.1 The structure of $\boldsymbol{\Omega}_{p^m}$

The notation introduced in (3.13) will be used here again (with the order of the horizontal blocks reversed). In addition a shorthand notation for the symmetry $\omega(x) = \omega(1-x)$ will be introduced. We give an example to clarify the notation.

Consider the matrix $\boldsymbol{\Omega}_n$ for n prime. As explained in Chapter 3, using the Rader factorization [87, 107], this matrix can be reordered such that it has a circulant submatrix of size $(n-1)\times(n-1)$. This is achieved by taking the indices as $z = g^i$ and $k = g^{-i'}$ for $0 \le i, i' \le n-2$, where g is a primitive root of n (i.e., g is a generator for the cyclic group U_n). This particular ordering of $\boldsymbol{\Omega}_n$ is denoted by $\boldsymbol{\Omega}_n^{\langle g\rangle}$.

Take for example $n = 11$; then the possible components for the generating vector which we want to consider are

$$U_{11} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}.$$

Since n is prime, a generator for this group can always be found. For our example we choose $g = 2$. The ordering for the z indices is then

$$[g^i : 0 \le i \le 9] = [1, 2, 4, 8, 5, 10, 9, 7, 3, 6].$$

For the indices $k \neq 0$ the negative powers of g give

$$[g^{-i'} : 0 \le i' \le 9] = [1, 6, 3, 7, 9, 10, 5, 8, 4, 2],$$

which can easily be seen as reversing the order of the z indices, but keeping the first index fixed (the zeroth power of g). This reordering, together with a column of zeros for $k = 0$, gives a matrix with the following structure:

$$\boldsymbol{\Omega}_{11}^{\langle 2\rangle} = \omega(\frac{1}{11} \left[\begin{array}{\underbrace{cccccccccc}_{\boldsymbol{C}_{11}} | c} 1 & 6 & 3 & 7 & 9 & 10 & 5 & 8 & 4 & 2 & 0 \\ 2 & 1 & 6 & 3 & 7 & 9 & 10 & 5 & 8 & 4 & 0 \\ 4 & 2 & 1 & 6 & 3 & 7 & 9 & 10 & 5 & 8 & 0 \\ 8 & 4 & 2 & 1 & 6 & 3 & 7 & 9 & 10 & 5 & 0 \\ 5 & 8 & 4 & 2 & 1 & 6 & 3 & 7 & 9 & 10 & 0 \\ 10 & 5 & 8 & 4 & 2 & 1 & 6 & 3 & 7 & 9 & 0 \\ 9 & 10 & 5 & 8 & 4 & 2 & 1 & 6 & 3 & 7 & 0 \\ 7 & 9 & 10 & 5 & 8 & 4 & 2 & 1 & 6 & 3 & 0 \\ 3 & 7 & 9 & 10 & 5 & 8 & 4 & 2 & 1 & 6 & 0 \\ 6 & 3 & 7 & 9 & 10 & 5 & 8 & 4 & 2 & 1 & 0 \end{array} \right]).$$

As was proved in Theorem 3.6, the number of rows and the number of columns of the submatrix $\boldsymbol{C}_{11}$ can be halved due to the symmetry $\omega(x) = \omega(1-x)$, see (3.14). We introduce a new notation to write $\boldsymbol{C}_{11}$ in a reduced form (denoted by a $\sim$ instead of $=$)

$$\boldsymbol{\Omega}_{11}^{\langle 2\rangle} \sim \omega(\frac{1}{11} \left[\begin{array}{\underbrace{ccccc}_{\tilde{\boldsymbol{C}}_{11}} \| c} 1 & 6 & 3 & 7 & 9 & 0 \\ 2 & 1 & 6 & 3 & 7 & 0 \\ 4 & 2 & 1 & 6 & 3 & 0 \\ 8 & 4 & 2 & 1 & 6 & 0 \\ 5 & 8 & 4 & 2 & 1 & 0 \end{array} \right]),$$

where the full matrix $\boldsymbol{\Omega}_{11}^{\langle 2\rangle}$ can be obtained by doubling each partition vertically and also horizontally up to the double bar.

The general idea for the fast algorithm for composite n is that the complete matrix $\boldsymbol{\Omega}_n$ can be partitioned in blocks which have a circulant or block-circulant structure. Basically, there is a vertical partition $\boldsymbol{A}_{d_i}^{\langle g \rangle}$ for each divisor d_i of n

$$\boldsymbol{\Omega}_n^{\langle g \rangle} := \left[\begin{array}{c|c|c|c} \boldsymbol{A}_{d_1}^{\langle g \rangle} & \boldsymbol{A}_{d_2}^{\langle g \rangle} & \cdots & \boldsymbol{A}_{d_\nu}^{\langle g \rangle} \end{array} \right],$$

where g is a generator (or technically, a set of generators) for the group U_n, and each $\boldsymbol{A}_{d_i}^{\langle g \rangle}$ is constructed out of circulant (or technically, nested block circulant) matrices $\boldsymbol{B}_{n/d_i}^{\langle g \rangle}$.

For the embedded lattice rules here, n is restricted to be a power of a prime p, because in this case the matrix $\boldsymbol{\Omega}_{p^m}^{\langle g \rangle}$ is of a particularly simple form. For this case $|U_{p^m}| = p^{m-1}(p-1)$. The structure for $\boldsymbol{\Omega}_{p^m}^{\langle g \rangle}$ will be stated for $p \neq 2$ and $p = 2$ in two separate theorems below; both results follow from Theorem 4.10 and Corollary 4.11. These two theorems differ in subtle yet crucial ways: the form of the matrices $\boldsymbol{B}_{p^m}^{\langle g \rangle}$ differs, and there are two degenerate cases $\boldsymbol{B}_{2^1}^{\langle 5 \rangle}$ and $\boldsymbol{B}_{2^0}^{\langle 5 \rangle}$ for $p = 2$, while there is only one degenerate case $\boldsymbol{B}_{p^0}^{\langle g \rangle}$ for $p \neq 2$. In the theorems $\mathbf{1}_t \otimes \boldsymbol{B}$ denotes the stacking of t replications of a matrix $\boldsymbol{B}$.

Theorem 5.1. *The matrix for p prime, $p \neq 2$,*

$$\boldsymbol{\Omega}_{p^m} := \left[\omega\left(\frac{kz \bmod p^m}{p^m} \right) \right]_{\substack{z \in U_{p^m} \\ k \in \mathbb{Z}_{p^m}}},$$

with $\omega(x) = \omega(1-x)$, can be ordered with respect to the divisors of p^m as

$$\boldsymbol{\Omega}_{p^m}^{\langle g \rangle} := \left[\begin{array}{c|c|c|c|c} \mathbf{1}_{p^0} \otimes \boldsymbol{B}_{p^m}^{\langle g \rangle} & \mathbf{1}_{p^1} \otimes \boldsymbol{B}_{p^{m-1}}^{\langle g \rangle} & \cdots & \mathbf{1}_{p^{m-1}} \otimes \boldsymbol{B}_{p^1}^{\langle g \rangle} & \mathbf{1}_{p^{m-1}(p-1)} \otimes \boldsymbol{B}_{p^0}^{\langle g \rangle} \end{array} \right],$$

where g is a generator for U_{p^m} and $\boldsymbol{B}_{p^0}^{\langle g \rangle} := [\omega(0)]$. For $\ell \geq 1$, each matrix $\boldsymbol{B}_{p^\ell}^{\langle g \rangle}$ (associated with a divisor $p^{m-\ell}$) is circulant of the form

$$\boldsymbol{B}_{p^\ell}^{\langle g \rangle} := \left[\begin{array}{c|c} \boldsymbol{M}_{p^\ell}^{\langle g \rangle} & \boldsymbol{M}_{p^\ell}^{\langle g \rangle} \\ \hline \boldsymbol{M}_{p^\ell}^{\langle g \rangle} & \boldsymbol{M}_{p^\ell}^{\langle g \rangle} \end{array} \right],$$

with circulant blocks

$$\boldsymbol{M}_{p^\ell}^{\langle g \rangle} := \left[\omega\left(\frac{kz \bmod p^\ell}{p^\ell} \right) \right]_{\substack{z \in \langle\!\langle g \rangle\!\rangle_{p^\ell} \\ k \in \langle\!\langle g^{-1} \rangle\!\rangle_{p^\ell}}}.$$

Here $\langle\!\langle g \rangle\!\rangle_{p^\ell}$ denotes the ordered set $[g^i \bmod p^\ell : 0 \leq i \leq \varphi(p^\ell)/2 - 1]$, i.e., half of the group U_{p^ℓ} in generator ordering.

Proof. This follows directly from Theorem 4.10 and Corollary 4.11. □

Theorem 5.2. *The matrix*

$$\boldsymbol{\Omega}_{2^m} := \left[\omega\left(\frac{kz \bmod 2^m}{2^m}\right)\right]_{\substack{z\in U_{2^m}\\ k\in\mathbb{Z}_{2^m}}},$$

with $\omega(x) = \omega(1-x)$, can be ordered with respect to the divisors of 2^m as

$$\boldsymbol{\Omega}^{\langle 5\rangle}_{2^m} := \left[\begin{array}{c|c|c|c} \mathbf{1}_{2^0}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^m} & \mathbf{1}_{2^1}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^{m-1}} & \cdots & \mathbf{1}_{2^{m-1}}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^1} \;\big|\; \mathbf{1}_{2^{m-1}(2-1)}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^0}\end{array}\right],$$

where $\boldsymbol{B}^{\langle 5\rangle}_{2^0} := [\omega(0)]$ and $\boldsymbol{B}^{\langle 5\rangle}_{2^1} := [\omega(1/2)]$. For $\ell \geq 2$, each matrix $\boldsymbol{B}^{\langle 5\rangle}_{2^\ell}$ (associated with a divisor $2^{m-\ell}$) is circulant of the form

$$\boldsymbol{B}^{\langle 5\rangle}_{2^\ell} := \left[\begin{array}{c|c} \boldsymbol{M}^{\langle 5\rangle}_{2^\ell} & \boldsymbol{M}^{\langle 5\rangle}_{2^\ell} \\ \hline \boldsymbol{M}^{\langle 5\rangle}_{2^\ell} & \boldsymbol{M}^{\langle 5\rangle}_{2^\ell}\end{array}\right], \tag{5.3}$$

with circulant blocks

$$\boldsymbol{M}^{\langle 5\rangle}_{2^\ell} := \left[\omega\left(\frac{kz \bmod 2^\ell}{2^\ell}\right)\right]_{\substack{z\in\langle 5\rangle_{2^\ell}\\ k\in\langle 5^{-1}\rangle_{2^\ell}}}.$$

Here $\langle 5\rangle_{2^\ell}$ denotes the ordered set $[5^i \bmod 2^\ell : 0 \leq i \leq 2^{\ell-2}-1]$, i.e., half of the group U_{2^ℓ} in generator ordering.

Proof. This follows directly from Theorem 4.10 and Corollary 4.11. □

This theorem will be illustrated with an example shortly. First the effect of the symmetry of ω is discussed. Note that Theorem 5.2 rests upon the partition

$$U_{2^\ell} = \langle 5\rangle_{2^\ell} \cup (-1)\langle 5\rangle_{2^\ell} \qquad \text{for } \ell \geq 2. \tag{5.4}$$

If the symmetry of ω would not be used, formula (5.3) would actually look like

$$\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{2^\ell} := \left[\begin{array}{c|c} \boldsymbol{M}^{\langle 5\rangle}_{2^\ell} & \boldsymbol{M}^{\langle 5\rangle\star}_{2^\ell} \\ \hline \boldsymbol{M}^{\langle 5\rangle\star}_{2^\ell} & \boldsymbol{M}^{\langle 5\rangle}_{2^\ell}\end{array}\right], \text{ with } \left\{\begin{array}{l} \boldsymbol{M}^{\langle 5\rangle}_{2^\ell} = \left[\omega\left(\frac{kz \bmod 2^\ell}{2^\ell}\right)\right]_{\substack{z\in\langle 5\rangle_{2^\ell}\\ k\in\langle 5^{-1}\rangle_{2^\ell}}}, \\ \boldsymbol{M}^{\langle 5\rangle\star}_{2^\ell} = \left[\omega\left(\frac{-kz \bmod 2^\ell}{2^\ell}\right)\right]_{\substack{z\in\langle 5\rangle_{2^\ell}\\ k\in\langle 5^{-1}\rangle_{2^\ell}}}, \end{array}\right.$$

and the stacking of the matrices $\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{2^\ell}$ would be

$$\widehat{\boldsymbol{\Omega}}^{\langle 5\rangle}_{2^m} := \left[\begin{array}{c|c|c|c} \boldsymbol{R}_{2^0}\,\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{2^m} & \cdots & \boldsymbol{R}_{2^{m-2}}\,\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{2^2} & \mathbf{1}_{2^{m-1}}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^1} \;\big|\; \mathbf{1}_{2^{m-1}}\otimes \boldsymbol{B}^{\langle 5\rangle}_{2^0}\end{array}\right], \tag{5.5}$$

where $\boldsymbol{R}_{2^\ell} := \boldsymbol{I}_2 \otimes \boldsymbol{1}_{2^\ell} \otimes \boldsymbol{I}_{2^{m-\ell-2}}$, which stacks the top and bottom half of $\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{2^{m-\ell}}$ separately. The symmetry of ω implies $\boldsymbol{M}^{\langle 5\rangle}_{2^\ell} = \boldsymbol{M}^{\langle 5\rangle\star}_{2^\ell}$ from which the theorem follows.

Now consider the example $n = 2^4 = 16$, initially without using the symmetry of ω. Following (5.4), the first half of U_{16} is generated by 5,

$$[5^i \bmod 16 : 0 \le i \le 3] = [1, 5, 9, 13],$$

where i stays less than $\varphi(n)/2$ since this is only half of the group U_{16}. The second half is a "copy" of this subgroup, but multiplied with $(-1) \equiv 15 \pmod{16}$,

$$[(-1)\,5^i \bmod 16 : 0 \le i \le 3] = [15, 11, 7, 3].$$

Using this row ordering we can associate a column ordering to get the circulant structure. This can be achieved by reversing the row ordering except for the first element. The first submatrix for the divisor $2^0 = 1$ is then

$$\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{16} = \omega\Big(\frac{1}{16}\left[\begin{array}{cccc|cccc} 1 & 13 & 9 & 5 & 15 & 3 & 7 & 11 \\ 5 & 1 & 13 & 9 & 11 & 15 & 3 & 7 \\ 9 & 5 & 1 & 13 & 7 & 11 & 15 & 3 \\ 13 & 9 & 5 & 1 & 3 & 7 & 11 & 15 \\ \hline 15 & 3 & 7 & 11 & 1 & 13 & 9 & 5 \\ 11 & 15 & 3 & 7 & 5 & 1 & 13 & 9 \\ 7 & 11 & 15 & 3 & 9 & 5 & 1 & 13 \\ 3 & 7 & 11 & 15 & 13 & 9 & 5 & 1 \end{array}\right]\Big).$$

The submatrices for the other divisors can be constructed similarly, giving

$$\widehat{\boldsymbol{B}}^{\langle 5\rangle}_{8} = \omega\Big(\frac{1}{8}\left[\begin{array}{cc|cc} 1 & 5 & 7 & 3 \\ 5 & 1 & 3 & 7 \\ \hline 7 & 3 & 1 & 5 \\ 3 & 7 & 5 & 1 \end{array}\right]\Big), \qquad \widehat{\boldsymbol{B}}^{\langle 5\rangle}_{4} = \omega\Big(\frac{1}{4}\left[\begin{array}{c|c} 1 & 3 \\ \hline 3 & 1 \end{array}\right]\Big),$$

$$\boldsymbol{B}^{\langle 5\rangle}_{2} = \omega\Big(\frac{1}{2}\left[\begin{array}{c} 1 \end{array}\right]\Big), \qquad \boldsymbol{B}^{\langle 5\rangle}_{1} = \omega\Big(\left[\begin{array}{c} 0 \end{array}\right]\Big).$$

Note that all these matrices, except for the last two trivial ones, have a block-circulant-with-circulant-blocks structure, which is consistent with the two partitions in their multiplicative groups, see (5.4). Using (5.5), the full matrix looks

like this (where italic font means redundancy)

$$\widehat{\boldsymbol{\Omega}}_{16}^{\langle 5\rangle} = \omega(\frac{1}{16}\left[\begin{array}{cccc|cccc|cccc|cc|cc} 1 & 13 & 9 & 5 & 15 & 3 & 7 & 11 & 2 & 10 & 14 & 6 & 4 & 12 & 8 & 0 \\ 5 & 1 & 13 & 9 & 11 & 15 & 3 & 7 & 10 & 2 & 6 & 14 & \textit{4} & \textit{12} & \textit{8} & \textit{0} \\ 9 & 5 & 1 & 13 & 7 & 11 & 15 & 3 & \textit{2} & \textit{10} & \textit{14} & \textit{6} & \textit{4} & \textit{12} & \textit{8} & \textit{0} \\ 13 & 9 & 5 & 1 & 3 & 7 & 11 & 15 & \textit{10} & \textit{2} & \textit{6} & \textit{14} & \textit{4} & \textit{12} & \textit{8} & \textit{0} \\ \hline 15 & 3 & 7 & 11 & 1 & 13 & 9 & 5 & 14 & 6 & 2 & 10 & 12 & 4 & \textit{8} & \textit{0} \\ 11 & 15 & 3 & 7 & 5 & 1 & 13 & 9 & 6 & 14 & 10 & 2 & \textit{12} & \textit{4} & \textit{8} & \textit{0} \\ 7 & 11 & 15 & 3 & 9 & 5 & 1 & 13 & \textit{14} & \textit{6} & \textit{2} & \textit{10} & \textit{12} & \textit{4} & \textit{8} & \textit{0} \\ 3 & 7 & 11 & 15 & 13 & 9 & 5 & 1 & \textit{6} & \textit{14} & \textit{10} & \textit{2} & \textit{12} & \textit{4} & \textit{8} & \textit{0} \end{array}\right]).$$

Now apply the symmetry of ω by mapping each $k \in \mathbb{Z}_n$ to $\min(n-k, k)$ to obtain

$$\boldsymbol{\Omega}_{16}^{\langle 5\rangle} = \omega(\frac{1}{16}\left[\begin{array}{cccc|cccc|cccc|cc|cc} 1 & 3 & 7 & 5 & \textit{1} & \textit{3} & \textit{7} & \textit{5} & 2 & 6 & \textit{2} & \textit{6} & 4 & \textit{4} & 8 & 0 \\ 5 & 1 & 3 & 7 & \textit{5} & \textit{1} & \textit{3} & \textit{7} & 6 & 2 & \textit{6} & \textit{2} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ 7 & 5 & 1 & 3 & \textit{7} & \textit{5} & \textit{1} & \textit{3} & \textit{2} & \textit{6} & \textit{2} & \textit{6} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ 3 & 7 & 5 & 1 & \textit{3} & \textit{7} & \textit{5} & \textit{1} & \textit{6} & \textit{2} & \textit{6} & \textit{2} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ \hline \textit{1} & \textit{3} & \textit{7} & \textit{5} & 1 & 3 & 7 & 5 & \textit{2} & \textit{6} & \textit{2} & \textit{6} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ \textit{5} & \textit{1} & \textit{3} & \textit{7} & 5 & 1 & 3 & 7 & \textit{6} & \textit{2} & \textit{6} & \textit{2} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ \textit{7} & \textit{5} & \textit{1} & \textit{3} & 7 & 5 & 1 & 3 & \textit{2} & \textit{6} & \textit{2} & \textit{6} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \\ \textit{3} & \textit{7} & \textit{5} & \textit{1} & 3 & 7 & 5 & 1 & \textit{6} & \textit{2} & \textit{6} & \textit{2} & \textit{4} & \textit{4} & \textit{8} & \textit{0} \end{array}\right]),$$

as promised in Theorem 5.2. This matrix can be represented in a reduced form:

$$\boldsymbol{\Omega}_{16}^{\langle 5\rangle} \sim \omega(\frac{1}{16}\left[\begin{array}{cccc|cc|c||c|c} 1 & 3 & 7 & 5 & 2 & 6 & \textit{4} & \textit{8} & 0 \\ 5 & 1 & 3 & 7 & 6 & 2 & \textit{4} & \textit{8} & \textit{0} \\ 7 & 5 & 1 & 3 & \textit{2} & \textit{6} & \textit{4} & \textit{8} & \textit{0} \\ 3 & 7 & 5 & 1 & \textit{6} & \textit{2} & \textit{4} & \textit{8} & \textit{0} \end{array}\right]).$$

5.4.2 Exploiting the embedding structure

In Theorems 5.1 and 5.2 we used implicitly the property that a generator g for the group U_{p^m} also generates U_{p^ℓ} for all $\ell \le m$, with the obvious adjustment for $p = 2$ where 5 generates U_{2^ℓ} in two partitions for all $\ell \le m$. This actually follows from the fact (see, e.g., [1, §24.3.4]) that if r is a generator for U_p, then a generator g for U_{p^ℓ}, $\ell \ge 2$, can be found by taking

$$g = \begin{cases} r + p & \text{if } r^{p-1} \equiv 1 \pmod{p^2}, \\ r & \text{otherwise.} \end{cases} \tag{5.6}$$

It can be checked that if one searches for generators in increasing numerical order starting from 2, then the first p for which the smallest generator of U_p does not generate U_{p^2} is as high as 40487 (and the first p for which one of its generators

does not generate U_{p^2} is $p = 29$). In other words, as long as p is smaller than 40487, the smallest generator of U_p also generates U_{p^ℓ} for all $\ell \geq 1$. Furthermore, the most logical choice in practice is $p = 2$ for which the pseudogenerator 5 can always be used. Since one generator suffices, a matrix for p^m points can easily be extended to a matrix for p^{m+1} points by simply adding one block in front.

For example to extend $\boldsymbol{\Omega}_{16}^{\langle 5\rangle}$ to $\boldsymbol{\Omega}_{32}^{\langle 5\rangle}$:

$$\boldsymbol{\Omega}_{32}^{\langle 5\rangle} = \left[\begin{array}{c|c} \boldsymbol{B}_{32}^{\langle 5\rangle} & \begin{array}{c} \boldsymbol{\Omega}_{16}^{\langle 5\rangle} \\ \hline \boldsymbol{\Omega}_{16}^{\langle 5\rangle} \end{array} \end{array}\right],$$

where the following recursive relations hold

$$\boldsymbol{\Omega}_{16}^{\langle 5\rangle} = \left[\begin{array}{c|c} \boldsymbol{B}_{16}^{\langle 5\rangle} & \begin{array}{c} \boldsymbol{\Omega}_{8}^{\langle 5\rangle} \\ \hline \boldsymbol{\Omega}_{8}^{\langle 5\rangle} \end{array} \end{array}\right], \quad \boldsymbol{\Omega}_{8}^{\langle 5\rangle} = \left[\begin{array}{c|c} \boldsymbol{B}_{8}^{\langle 5\rangle} & \begin{array}{c} \boldsymbol{\Omega}_{4}^{\langle 5\rangle} \\ \hline \boldsymbol{\Omega}_{4}^{\langle 5\rangle} \end{array} \end{array}\right], \quad \boldsymbol{\Omega}_{4}^{\langle 5\rangle} = \left[\begin{array}{c|c} \boldsymbol{B}_{4}^{\langle 5\rangle} & \begin{array}{c} \boldsymbol{\Omega}_{2}^{\langle 5\rangle} \\ \hline \boldsymbol{\Omega}_{2}^{\langle 5\rangle} \end{array} \end{array}\right],$$

with $\boldsymbol{\Omega}_2^{\langle 5\rangle} = [\boldsymbol{B}_2^{\langle 5\rangle} | \boldsymbol{B}_1^{\langle 5\rangle}]$. In reduced form (and a common scaling of 1/32) this reads

$$\boldsymbol{\Omega}_{32}^{\langle 5\rangle} \sim$$

$$\omega(\frac{1}{32}\left[\begin{array}{cccccccc|cccc|cc|c||c|c} 1 & 13 & 9 & 11 & 15 & 3 & 7 & 5 & 2 & 6 & 14 & 10 & 4 & 12 & 8 & 16 & 0 \\ 5 & 1 & 13 & 9 & 11 & 15 & 3 & 7 & 10 & 2 & 6 & 14 & 12 & 4 & 8 & 16 & 0 \\ 7 & 5 & 1 & 13 & 9 & 11 & 15 & 3 & 14 & 10 & 2 & 6 & 4 & 12 & 8 & 16 & 0 \\ 3 & 7 & 5 & 1 & 13 & 9 & 11 & 15 & 6 & 14 & 10 & 2 & 12 & 4 & 8 & 16 & 0 \\ \hline 15 & 3 & 7 & 5 & 1 & 13 & 9 & 11 & 2 & 6 & 14 & 10 & 4 & 12 & 8 & 16 & 0 \\ 11 & 15 & 3 & 7 & 5 & 1 & 13 & 9 & 10 & 2 & 6 & 14 & 12 & 4 & 8 & 16 & 0 \\ 9 & 11 & 15 & 3 & 7 & 5 & 1 & 13 & 14 & 10 & 2 & 6 & 4 & 12 & 8 & 16 & 0 \\ 13 & 9 & 11 & 15 & 3 & 7 & 5 & 1 & 6 & 14 & 10 & 2 & 12 & 4 & 8 & 16 & 0 \end{array}\right]).$$

In Lemma 4.13 it was explained how to sum the result vectors of a matrix-vector multiplication with the matrix $\boldsymbol{\Omega}_n$ so that the complete result could still be calculated in $O(n \log(n))$ time. In fact, since here n is of the form p^m, p prime, any method used to sum the intermediate result vectors would have been sufficient to get at the $O(n \log(n))$ time complexity (see the part just before Lemma 4.13). By accumulating the result vectors from the smallest matrix upward, the intermediate result vectors which are precisely the matrix-vector products with the matrices $\boldsymbol{\Omega}_{p^\ell}$ for ℓ going from 0 up to $m-1$ are also obtained. Thus in an attempt to compute $e_{p^m,s}$, along the way the values of $e_{p^\ell,s}$ for ℓ from 0 to $m-1$ are available at no extra cost.

The availability of all these intermediate results is a nice side effect since then the cost of calculating $X_{m_1,m_2,s}$, which involves $e_{p^\ell,s}$ for all ℓ from m_1 to m_2, is no worse than the cost of calculating just $e_{p^{m_2},s}$. This brings us to the main result of this chapter.

Theorem 5.3. *Construction of embedded lattice rules up to $n = p^{m_2}$ points, with p prime, using Algorithm 5 can be done in $O(sn\log(n))$ time, with $O(n)$ memory for product weights and $O(qn)$ memory for order-dependent weights of order q. The total construction cost including the precalculations is therefore $O(sn(\log(n))^2)$.*

5.5 Embedded lattice rules as sequences

The main advantage of having a sequence of embedded lattice rules $L_0 \subset L_1 \subset L_2 \subset \cdots$ is that the number of points can be increased without having to throw away already computed function values. When the base is b (not necessarily prime anymore), the number of points is increased by a factor of b at each step along the sequence. For $\ell \geq 1$ denote the set of additional points from the point set $L_{\ell-1}$ to L_ℓ with

$$\Delta L_\ell := L_\ell \setminus L_{\ell-1}.$$

Recall from Property 4 that the points in $L_{\ell-1}$ are generated by the indices k which are multiples of b in $\mathbb{Z}_{b^\ell}$. Thus

$$L_m = L_0 \cup \left(\bigcup_{\ell=1}^{m} \Delta L_\ell \right), \tag{5.7}$$

with $L_0 = \{\mathbf{0}\}$ and

$$\Delta L_\ell = \left\{ \left\{ \frac{k\boldsymbol{z}}{b^\ell} \right\} : k \in (\mathbb{Z}_{b^\ell} \setminus b\mathbb{Z}_{b^{\ell-1}}) \right\}. \tag{5.8}$$

Letting $m \to \infty$ in (5.7) leads formally to an extensible lattice sequence L_∞ with an infinite number of points. The construction here considers lattice point sets where the number of points is finite, that is, the union in (5.7) goes only to a predefined upper limit m_2. But, due to the fast construction, this limit can be taken rather large. (In fact, for current practice there is no need to have more than, e.g., 2^{30} points. Such implied limits are then also present in, e.g., implementations of the Sobol' sequence.)

Note that the set of additional points ΔL_ℓ from the point set $L_{\ell-1}$ to L_ℓ actually comprises $b-1$ shifted versions of $L_{\ell-1}$. This can be put forward as another property of embedded lattice rules (see also [46, Theorem 2.3]).

Property 5. The lattice point set L_m can be considered as the union of b shifted versions (dependent on $\boldsymbol{z}$) of the lattice point set L_{m-1}, as well as the union of b^2 shifted versions of L_{m-2}, and so on. Trivially, L_m is the union of b^m shifted versions of the initial point $\mathbf{0}$. Explicitly:

$$L_m = \bigcup_{k_0=0}^{b-1} \cdots \bigcup_{k_\ell=0}^{b-1} \left(L_{m-\ell-1} + \frac{k_\ell \boldsymbol{z}}{b^{m-\ell}} + \cdots + \frac{k_0 \boldsymbol{z}}{b^m} \right) \quad \forall \ell = 0, \ldots, m-1,$$

where the addition is interpreted as in the definition $P_n + \boldsymbol{\Delta} := \{\{\boldsymbol{x} + \boldsymbol{\Delta}\} : \boldsymbol{x} \in P_n\}$ (and the inner braces indicate taking the fractional part).

Using a generating vector from Algorithm 5 for n ranging from b^{m_1} to b^{m_2}, one can start with an approximation of the integral using the point set L_{m_1}, and then gradually add blocks of additional points $\Delta L_{m_1+1}, \Delta L_{m_1+2}, \ldots, \Delta L_{m_2}$ until the error estimate satisfies the accuracy asked for (provided that m_2 is large enough). This usage of the sequence of embedded lattice rules is justified by the knowledge that every lattice point set L_m is good for $m_1 \le m \le m_2$.

From a practical point of view it would be more interesting to be able to use the lattice rule *one point at a time*, i.e., as a low-discrepancy sequence. In a natural ordering of the k indices in (5.8) it is obvious that with our de facto choice of $z_1 = 1$ the first half of the points will all fall in one half of the cube. To generate the points in a different order than the natural ordering, define a permutation ϕ from $\mathbb{Z}_n$ to $\mathbb{Z}_n$ and obtain the kth point as

$$\boldsymbol{x}_k = \left\{ \frac{\phi(k)\,\boldsymbol{z}}{n} \right\}.$$

If the lattice rule has $n = b^m$ points then one can immediately use the radical inverse function in base b for the quantities $\phi(k)/n$. This is the view taken in [46, 47] to define extensible/embedded lattice rules. Recall Definition 1.17.

Definition 5.4. The *radical inverse function in base b* maps integers into b-ary fractions by mirroring around the b-ary radix point:

$$\phi_b(k) = \phi_b((k_{m-1} \ldots k_0)_b) := (0.k_0 \ldots k_{m-1})_b = \sum_{\ell=0}^{m-1} k_\ell\, b^{-(\ell+1)}.$$

Another possible permutation is the Gray ordering. First define the Gray code in an arbitrary base b.

Definition 5.5. The numbers in $\mathbb{Z}_{b^m}$ are ordered in a *generalized Gray code in base b* if the difference between two consecutive numbers in base b is one digit at one position only.

Take the following iterative definition for the Gray code $G_b(k)$ of an integer k in base b (see, e.g., [104])

$$G_b(k) = G_b(k-1) \oplus_b b^{c(k)}, \qquad \text{or equivalently } G_b(k) = k \ominus_b (k \gg_b 1),$$

where $\oplus_b$ is digitwise addition modulo b and $c(k)$ is the position in which the one digit difference is in effect (and likewise $\ominus_b$ is digitwise subtraction modulo b and $\gg_b$ is a shift to the right in base b). The position $c(k)$ is then given by the rightmost digit in k which differs from $b-1$ (and which is easy to calculate).

(Unlike the classical Gray code in base 2, where $\oplus$ and $\ominus$ are the same, this is not a reflection code.)

The Gray ordering was first used in base 2 to generate Sobol' points, see [3]. For arbitrary base this is defined as:

Definition 5.6. The *Gray ordering in base b* is nothing more then the radical inversion of the Gray code in base b:

$$G_{b^{-1}}(k) = \phi_b(G_b(k)).$$

The number $\phi(k)/n$ is not stored as a floating point value, but as an integer scaled by b^m (among others, rounding problems could occur for $b \neq 2$). So suppose $g = G_b(k) = (g_{m-1} \dots g_0)_b$ then the radical inverse Gray code is given as $G_{b^{-1}}(k) = (0.g_0 \dots g_{m-1})_b$ and in actual code it is profitable to compute

$$k' = b^m G_{b^{-1}}(k) = b^m G_{b^{-1}}(k-1) \oplus_b b^{m-(c(k)+1)}, \qquad \text{and then } \boldsymbol{x}_k = \left\{ \frac{k'\mathbf{z}}{n} \right\}.$$

These schemes provide a better ordering of the numbers k/b^ℓ in the unit interval, and this property is preserved under multiplication by $\boldsymbol{z}$ modulo 1 so that the ordering of every one-dimensional projection of the lattice points is good. Both schemes are well-known tools borrowed from the vast literature on digital sequences; see, for example, [76]. The Gray ordering is in general computationally more efficient since the next index can be obtained from the previous one by a simple digit change and the actual radical inversion can be avoided. (In base 2 the bit position can be found with a single assembler instruction on at least i386 and m6800 architectures; but also the radical inversion in base 2 can be done rather efficiently in $O(\log_2(m))$, see [39].)

The use of these schemes can in fact be summarized as a property of embedded lattice rules. Note that they preserve the embedding property $L_0 \subset L_1 \subset \cdots$ of the lattice point sets, while providing a better ordering of the points within each block of additional points ΔL_ℓ. We formulate this as yet another property of embedded lattice rules.

Property 6. The lattice point set L_m can be used as a low-discrepancy sequence if the points are generated in the order

$$\boldsymbol{x}_k = \{\psi_b(k)\, \boldsymbol{z}\} \qquad \text{for} \quad k = 0, 1, \dots, b^m - 1,$$

where $\psi_b(k)$ is a one-dimensional low-discrepancy sequence in base b. E.g., the radical inverse $\phi_b(k)$ of k in base b, or the corresponding Gray code variant.

5.6 Numerical experiments with the construction

This section presents results on good generating vectors of embedded lattice rules which where constructed using Algorithm 5 for n ranging from 2^{10} to 2^{20} and up

Weights	$\max_s X_{10,20,s}(\boldsymbol{z}^\star)$
product 1	1.53
product 0.5	1.48
product 0.3	1.47
product 0.1	1.41
product 0.05	1.37
product 0.01	1.30
product 0.9^j	1.50
product 0.75^j	1.42
product 0.5^j	1.30

Weights	$\max_s X_{10,20,s}(\boldsymbol{z}^\star)$
product $1/j$	1.41
product $1/j^2$	1.31
order-2	1.43
order-3 1	1.42
order-3 0.5	1.55
order-3 0.3	1.45
order-3 0.1	1.47
order-3 0.05	1.45
order-3 0.01	1.43

Table 5.1: Maximum of $X_{10,20,s}(\boldsymbol{z}^\star)$ for various weights.

to 360 dimensions. Apart from the rules for $2^{20} \approx 10^6$ points also one rule of $2^{25} \approx 33 \times 10^6$ points was constructed to be able to compare the degradation.

5.6.1 Experimental values for the error criterion

The search criterion for n from 2^{10} to 2^{20} is

$$X_{10,20,s}(\boldsymbol{z}) = \max_{10 \le m \le 20} \frac{e_{2^m,s}(\boldsymbol{z})}{e_{2^m,s}(\boldsymbol{z}^{(m)})},$$

where $\boldsymbol{z}^{(m)}$ is the near optimal generating vector constructed using the component-by-component algorithm for a 2^m-point rule (which will be referred to as a good "fixed" rule). Let $\boldsymbol{z}^\star$ denote the vector resulting from Algorithm 5. Then the quantity $X_{10,20,s}(\boldsymbol{z}^\star)$ measures how far off the vector $\boldsymbol{z}^\star$ is compared with the vectors $\boldsymbol{z}^{(m)}$ in terms of the maximum (over m) ratio of the worst-case errors.

The numerical experiments consider weighted Sobolev spaces (see §2.4.3) with both product weights and order-dependent weights. In particular equal product weights, decaying product weights, order-2 weights and order-3 weights. The choices of weights together with the maxima of $X_{10,20,s}$ over all $1 \le s \le 360$ are given in Table 5.1.

The product-type weights in the table are self-explanatory: "product 0.5" means $\gamma_j = 0.5$ and "product $1/j$" means $\gamma_j = 1/j$. The choices of order-2 and order-3 weights will now be briefly explained. First, since all one-dimensional projections of our lattice rules are left-rectangle rules with all points distinct, the weight Γ_1 in an order-dependent weight setting has no essential effect on the choice of the rule. Therefore set $\Gamma_1 = 1$. Second, for a rule based on order-2 weights (i.e., if $\Gamma_\ell = 0$ for $\ell > 2$), the rule is unique regardless of how one chooses Γ_2, since in this case Γ_2 rescales only the worst-case error. Therefore without loss of generality also set $\Gamma_2 = 1$. In other words, such a rule is universally applicable for all order-2 problems. For order-3 weights, the rule depends only on the ratio Γ_3/Γ_2. Hence Γ_3 can be used to specify the importance of order-3 interactions compared

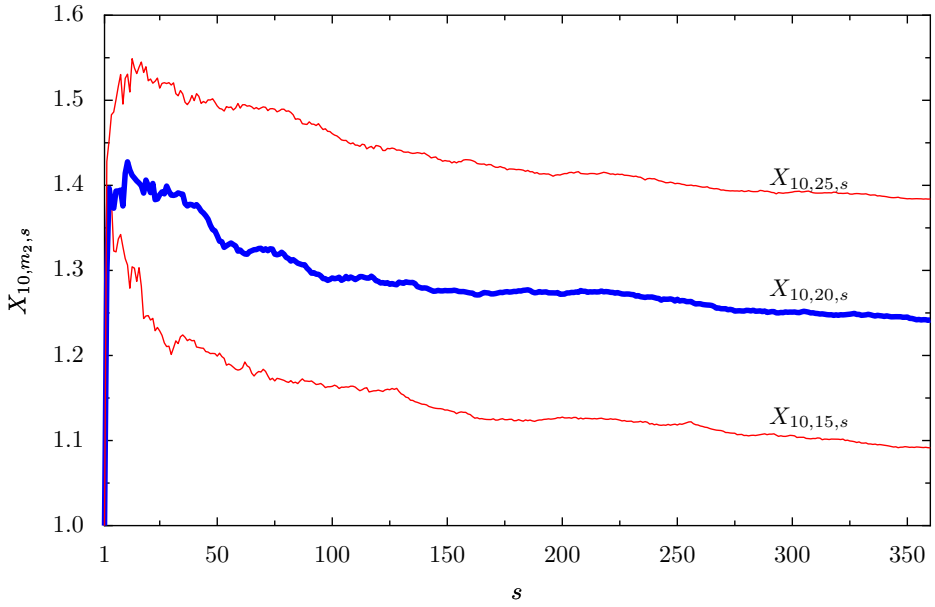


Figure 5.1: Profile of $X_{10,m_2,s}(\boldsymbol{z}^\star)$ for order-2 rules with $m_2 = 15$, 20, and 25 as s increases. There is a peak in the first few 10 dimensions after which the curve decreases.

with order-2 interactions. In the table, "order-3 0.1" corresponds to the choice $\Gamma_1 = \Gamma_2 = 1$ and $\Gamma_3 = 0.1$. Note that having equal product weights $\gamma_j = r$ is equivalent to having order-dependent weights with $\Gamma_\ell = r^\ell$. The generating vector for the order-2 rule is given in Table 5.2. From Table 5.1, the worst-case errors for the order-2 rule in Table 5.2 are within a factor of 1.43 of the worst-case errors obtained with the results from the original component-by-component construction.

From Table 5.1 it can be seen that $X_{10,20,s}(\boldsymbol{z}^\star)$ is substantially less than 1.6 for all our choices of weights. Thus, in our experiments, these rules achieve the optimal rate of convergence but with a constant which is at most 1.6 times larger than the optimal component-by-component one.

5.6.2 Profile of X for a range of m

From the data for Table 5.1 we observed that, starting from $X_{10,20,1}(\boldsymbol{z}^\star) = 1$, the numbers $X_{10,20,s}(\boldsymbol{z}^\star)$ increase initially as s increases, and then they decrease from some dimension onward. Figure 5.1 shows such a profile of $X_{10,m_2,s}(\boldsymbol{z}^\star)$ with increasing s for order-2 rules with $m_2 = 15$, 20, and 25. It is understandable that by allowing the range of n to grow, the quality of the point set is sacrificed. But we observe that the quality remains very good for all feasible values of m_2.

d	z_d	d	z_d	d	z_d	d	z_d	d	z_d	d	z_d
1	1	61	390955	121	151423	181	455605	241	8601	301	83785
2	182667	62	202915	122	492123	182	227317	242	118219	302	11217
3	302247	63	230815	123	166279	183	251285	243	503827	303	163315
4	433461	64	304301	124	335461	184	295275	244	368993	304	293397
5	160317	65	452315	125	295093	185	476155	245	439439	305	386597
6	94461	66	383101	126	98821	186	291155	246	153211	306	338761
7	481331	67	70519	127	118331	187	294443	247	282303	307	88653
8	252345	68	346393	128	371123	188	31913	248	252999	308	337581
9	358305	69	291605	129	166909	189	518445	249	28617	309	230703
10	221771	70	397773	130	369553	190	481917	250	514327	310	140519
11	48157	71	92713	131	310959	191	326981	251	422201	311	421913
12	489023	72	391775	132	130595	192	488711	252	465011	312	416151
13	438503	73	131613	133	417025	193	447541	253	160907	313	197025
14	399693	74	74351	134	264103	194	39629	254	191687	314	350607
15	200585	75	382127	135	453203	195	394681	255	226205	315	262579
16	169833	76	219343	136	497985	196	379411	256	254941	316	510879
17	308325	77	297125	137	200509	197	335309	257	480375	317	451713
18	247437	78	88545	138	269743	198	93541	258	322857	318	8619
19	281713	79	89837	139	461919	199	188491	259	456621	319	50451
20	424209	80	191863	140	45927	200	152371	260	446175	320	193793
21	244841	81	506647	141	394663	201	408829	261	109049	321	155739
22	205461	82	441649	142	299155	202	217957	262	285305	322	23611
23	336811	83	240063	143	81299	203	65145	263	436731	323	437689
24	359375	84	239067	144	112403	204	462013	264	290763	324	100267
25	86263	85	310875	145	447473	205	220627	265	290439	325	439671
26	370621	86	211625	146	480325	206	368989	266	25363	326	211341
27	422443	87	147791	147	105053	207	273585	267	480371	327	22951
28	284811	88	370849	148	328455	208	373297	268	153337	328	73405
29	231547	89	149445	149	513239	209	285793	269	406899	329	220037
30	360239	90	340329	150	322199	210	405807	270	90863	330	169733
31	505287	91	493031	151	47537	211	63693	271	78537	331	408633
32	355195	92	336897	152	485183	212	382573	272	358757	332	64171
33	52937	93	202595	153	490687	213	84291	273	69087	333	44141
34	344561	94	518247	154	214311	214	444801	274	431749	334	122149
35	286935	95	369599	155	61871	215	226471	275	384083	335	258491
36	312429	96	50453	156	359761	216	166195	276	472419	336	215021
37	513879	97	133655	157	509981	217	170689	277	298375	337	11507
38	171905	98	395941	158	192829	218	368423	278	291499	338	295341
39	50603	99	13871	159	17075	219	509169	279	187231	339	41981
40	441451	100	248123	160	486463	220	243221	280	6967	340	378867
41	164379	101	380725	161	463461	221	191447	281	338357	341	261545
42	139609	102	314879	162	438237	222	200867	282	411965	342	56423
43	371213	103	106903	163	444353	223	194633	283	447817	343	445605
44	152351	104	94505	164	381279	224	226469	284	135463	344	362017
45	138607	105	499197	165	425405	225	413665	285	156061	345	504897
46	441127	106	479671	166	179723	226	148007	286	356943	346	335763
47	157037	107	76553	167	203897	227	284505	287	224431	347	417407
48	510073	108	351609	168	300411	228	459795	288	452203	348	58033
49	281681	109	13815	169	35087	229	324557	289	284195	349	80239
50	380297	110	342069	170	501519	230	422149	290	489071	350	372353
51	208143	111	446599	171	172275	231	368087	291	299537	351	116163
52	497641	112	374429	172	278507	232	166133	292	313173	352	287617
53	482925	113	342011	173	473739	233	401441	293	436265	353	483795
54	233389	114	8365	174	141429	234	44077	294	316447	354	310473
55	238553	115	424079	175	117025	235	457535	295	8353	355	75779
56	121499	116	184381	176	516465	236	122611	296	512723	356	141943
57	137783	117	203853	177	204743	237	27489	297	522699	357	418435
58	463115	118	7937	178	505099	238	392389	298	453557	358	305609
59	168681	119	302301	179	359937	239	327231	299	512829	359	152901
60	70699	120	444747	180	359503	240	87841	300	214315	360	129869

Table 5.2: Order-2 embedded rule for $2^{10} \leq n \leq 2^{20}$.

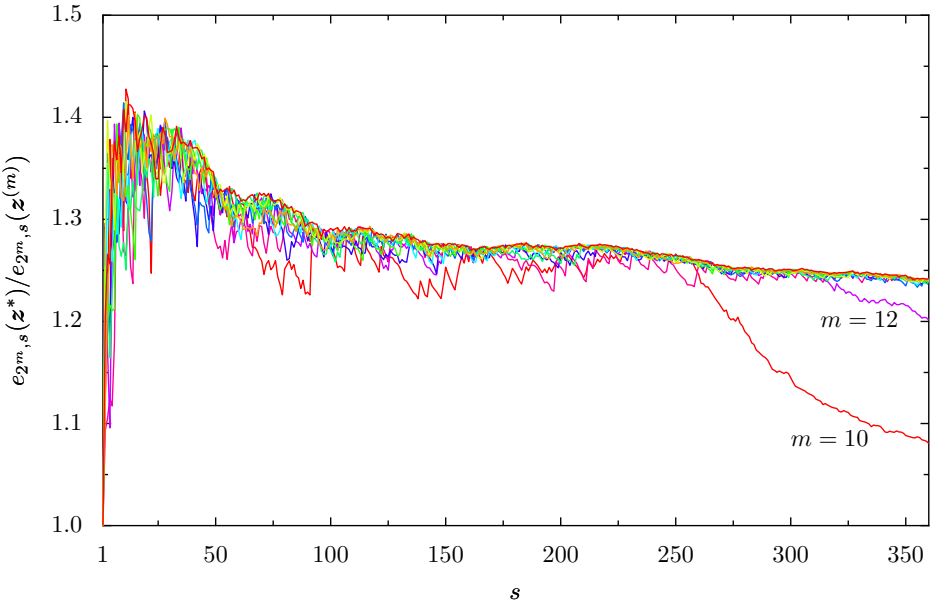


Figure 5.2: Initial segments of a good embedded lattice rule

5.6.3 Using initial segments of a good embedded lattice rule

The criterion $X_{10,20,s}(\boldsymbol{z}^\star)$ ensures that the resulting vector $\boldsymbol{z}^*$ is good for a 2^m-point rule for all values of m between 10 and 20. In other words, every point set in the embedded sequence $L_{10} \subset L_{11} \subset \cdots \subset L_{20}$ is good. (We refer to the smaller lattice point sets L_ℓ with $\ell < 20$ as the "initial segments" of the point set L_{20}.) In Figure 5.2 the worst-case error ratios

$$\frac{e_{2^m,s}(\boldsymbol{z}^\star)}{e_{2^m,s}(\boldsymbol{z}^{(m)})}$$

for the case of the order-2 rule in Table 5.2 and every $m = 10, \ldots, 20$ are plotted.

The maximum of all these graphs corresponds to the graph for $X_{10,20,s}(\boldsymbol{z}^\star)$ in Figure 5.1. For most values of m the embedded rule does better than $X_{10,20,s}(\boldsymbol{z}^\star)$ promises, but note that actually all the individual curves stay close to $X_{10,20,s}(\boldsymbol{z}^\star)$. Also these curves cross each other in an erratic manner, which is an important observation: otherwise the error measure could be redefined to make use of some assumed ordering. That said, in general it can be observed that the curves for higher m tend to be closer to the top.

For high dimensions the curves for $m = 10$ and $m = 12$ depart from the general trend. This might be explained by observing that the differences between

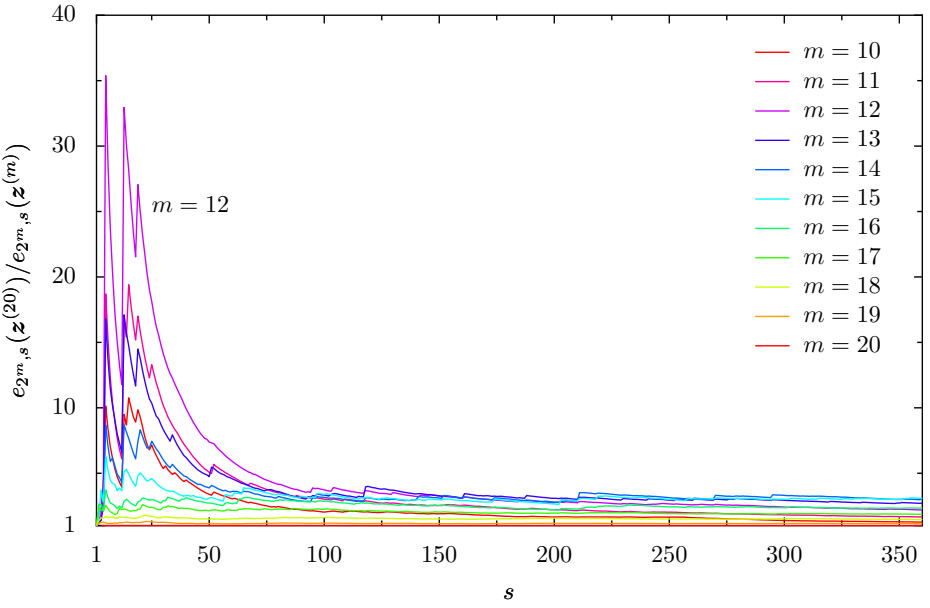


Figure 5.3: Initial segments of a good fixed lattice rule

the worst-case errors of different choices for a new component decrease as the dimension increases. In other words, the different choices of generating vectors are equally bad when there are very few points in very high dimensions.

5.6.4 Using initial segments of a good fixed lattice rule

One could wonder if it would be satisfactory to just use the initial segments from a good 2^{20}-point lattice rule with generating vector $\boldsymbol{z}^{(20)}$ constructed in the standard component-by-component way. (As before, by initial segments we mean the smaller lattice point sets L_ℓ embedded in L_{20} with $\ell < 20$.) If these initial segments would be any good, then there would be little point using Algorithm 5. In Figure 5.3 the worst-case error ratios

$$\frac{e_{2^m,s}(\boldsymbol{z}^{(20)})}{e_{2^m,s}(\boldsymbol{z}^{(m)})}$$

for every $m = 10, \ldots, 20$ are plotted.

It is obvious from the figure that using a large good fixed lattice rule for a maximum number of points $n = 2^{m_2}$ tells nothing about the goodness of the initial segments of this lattice rule. While $X_{10,20,s}(\boldsymbol{z}^\star)$ is bounded by 1.45 for the good embedded lattice rule, for the fixed lattice rule, the corresponding ratio is

35 times worse than the optimal fixed lattice rule for $m = 12$. Once again there is no direct ordering in these curves, but it is obvious that the curves corresponding to higher m are lower on the graph. For $m = 20$ there is a horizontal line at 1, and the line just above this constant line is the one for $m = 19$. It is not surprising that using half of the points of a good fixed lattice rule would also give a reasonable rule. The maximum of all these curves is clearly very bad. We can draw a very clear conclusion from this graph: the embedded lattice rule constructed by Algorithm 5 works very well in comparison with a more simple-minded strategy.

5.7 A Matlab implementation

This section presents two more example implementations for the fast construction of lattice sequences with a prime power of points as given in Algorithm 5. A first routine `fastrank1expt` for product weights and a second routine `fastrank1exod` for order dependent weights. For the calculation of the X criterion they need access to the optimal component-by-component values of the worst-case errors for (fixed) prime power lattice rules. The Matlab scripts for constructing lattice sequences from $n = p^{m_1}$ up to p^{m_2} presented here embed a recursive call to themselves for these precalculations. This means that these scripts can also be used to construct lattice rules with p^m points by taking the input parameters for m_1 and m_2 equal to m.

Listing 5.2 on page 134 contains the code for the fast construction of an embedded lattice sequence for product weights. In comparison with Listing 3.1, page 69, the readability now suffers a bit from explicitly putting the complete code in one file. However, comparing with Listing 3.1 gives a good idea of what is going on. The basic case for the single prime is now repeated $m_2 - m_1 + 1$ times and the vector `q` is now the concatenation of the parts corresponding to the different powers of p. The auxiliary vectors `sidx` and `eidx` are used to easily index `q` for the power of p under consideration, while the vector `phip` contains the sizes of these blocks.

Listing 5.3 on page 136 contains the code for the fast construction of an embedded lattice sequence with order dependent weights. Compare with Listing 3.4, page 74, and the description for order dependent weights in §3.3. Apart from the loops for the different powers of p, there are also loops over the different order parts. This function has an extra output argument `e2s` which is a vector of all squared worst-case errors e^2_{s,b^m}, $m_1 \le m \le m_2$, for the constructed lattice rule.

Both routines use the function `generator` when p is an odd prime. In Chapter 3 a Matlab routine was presented to calculate a generator for U_n with n prime. Here we provide in Listing 5.1 on page 133 a general routine which gives the generator for any suitable n, see Corollary 4.3, and makes use of `generatorp` from Listing 3.2 for the base prime case. To find a generator for some group U_{p^m} it suffices to search a generator for U_{p^2}, see (5.6).

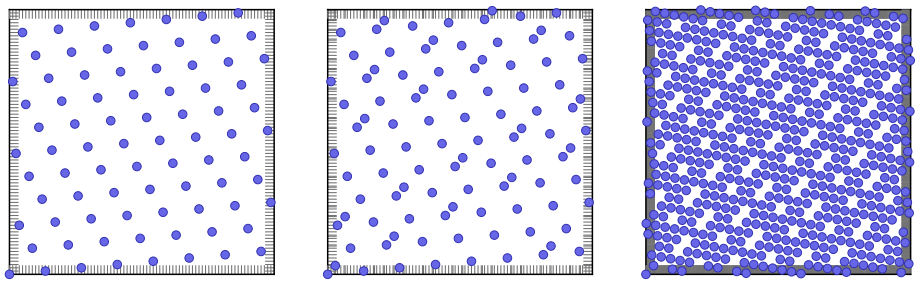


Figure 5.4: A lattice sequence in base 3 with generating vector $\boldsymbol{z} = (1, 140)$. The first plot has 81 points; since this is a power of 3 it is a full lattice. The second plot is stopped at 100 points and the third plot is stopped at 600 points. The points show a nice point by point distribution by using the Gray order in base 3.

Constructing a lattice sequence for powers of 3 from 3^3 up to 3^6 in the unanchored Sobolev space with order-2 weights could be done as follows:

```
[z, e2, x, e2s, CBC_optimal] = ...
  fastrank1exod(3, 3, 6, 100, @(x) x.^2-x+1/6, [1 1], [], 2);
```

Giving the following output for the first 10 dimensions:

```
s=    1, z=      1, e2=3.1361e-07, x=1.0000e+00, mloc=    3
s=    2, z=    140, e2=2.0024e-06, x=1.1581e+00, mloc=    4
s=    3, z=    305, e2=4.8477e-06, x=1.2563e+00, mloc=    5
s=    4, z=    220, e2=9.1841e-06, x=1.1864e+00, mloc=    4
s=    5, z=    311, e2=1.5844e-05, x=1.1318e+00, mloc=    6
s=    6, z=     71, e2=2.3926e-05, x=1.1357e+00, mloc=    6
s=    7, z=     85, e2=3.7140e-05, x=1.1981e+00, mloc=    6
s=    8, z=    236, e2=5.2075e-05, x=1.1756e+00, mloc=    6
s=    9, z=    211, e2=6.8991e-05, x=1.1421e+00, mloc=    6
s=   10, z=    190, e2=8.9898e-05, x=1.1257e+00, mloc=    6
```

The `mloc` variable gives the power of m where the maximum ratio of the worst-case errors occurs. The points in the first two dimensions of this example are plotted in Figure 5.4.

Listing 5.1: Finding a primitive root of n (`generator.m`)

```
function g = generator(n)

f = factor(n);
uf = unique(f); % the unique factors of n, in increasing order
nf = zeros(size(uf)); % in nf we will put the powers of each unique factor
for i=1:length(nf), nf(i) = length(find(f == uf(i))); end;
% we now have that n = prod(uf .^ nf)

if (length(uf) == 2) && (uf(1) == 2) && ...
   (nf(1) == 1) && (mod(uf(2), 2) == 1)
    % n is of the form 2p^k, with p an odd prime
    p = uf(2); k = nf(2);
    g = generatorp(p);
    if mod(g, 2) == 0
        g = g + p^k;
    end;
elseif (length(uf) == 1) && (mod(uf(1), 2) == 1)
    % n is of the form p^k, with p an odd prime
    p = uf(1); k = nf(1);
    g = generatorp(p);
    if k > 1
        if powmod(g, p-1, p^2) == 1
            % This happens first for the prime 40487,
            % that is for n = 40487^2 = 1639197169.
            % The powmod.m routine will then incorrectly
            % calculate 5^(40487-1) mod 1639197169 unless using
            % the arbitrary precision integers from Java.
            g = g + p;
        end;
    end;
elseif n == 2
    g = 1;
elseif n == 4
    g = 3;
else
    error(['n must be 2 or 4 or of the form p^k or 2*p^k, ' ...
           'with p an odd prime']);
end;
```

Listing 5.2: Sequence construction for product weights (`fastrank1expt.m`)

```
function [z, e2, x, CBC_optimal] = fastrank1expt(p, m1, m2, ...
  s_max, omega, gamma, beta, CBC_optimal, verbose)

if (nargin < 9) || isempty(verbose), verbose = 1; end;
if ~isprime(p), error('p must be prime'); end;
if (m1 < 1) || (m1 > m2), error('must have 1 <= m1 <= m2'); end;

if (nargin < 8) || isempty(CBC_optimal) % if precalculations needed
  CBC_optimal = {}; CBC_optimal{m2} = ones(s_max, 1);
  if m1 ~= m2
    for m=m1:m2
      CBC_optimal{m} = CBC_optimal{m2}; % alias using Matlab's lazy copy
      [opt_z, CBC_optimal{m}] = fastrank1expt(p, m, m, s_max, ...
          omega, gamma, beta, CBC_optimal, verbose-1);
      end;
    end;
end;

n = p^m2;                                % total points n = p^m2
if p == 2, g = 5; else g = generator(p*p); end; % pseudo when p = 2
phip0 = 1; phip = zeros(m2, 1);          % calculate φ(p^m)
phip(1) = p - 1; for m=2:m2, phip(m) = p*phip(m-1); end;
phip(phip >= 2) = phip(phip >= 2) / 2; % the size for d = p^m is in phip(m)
perm = zeros(phip(m2), 1); perm(1) = 1;% g^i mod p^m sequence
for i=2:phip(m2), perm(i) = mod(perm(i-1)*g, n); end;
if m2~=1, sidx = [1; cumsum(phip(1:m2-1))+1]; else sidx = [1]; end;
eidx = sidx + phip - 1;                  % start and ending indices

psi0 = omega(0);                         % build the ψ vector
psi = zeros(sum(phip), 1); fft_psi = zeros(sum(phip), 1);
for m=1:m2
  psi(sidx(m):eidx(m)) = omega(mod(perm(1:phip(m)), p^m) / p^m);
  fft_psi(sidx(m):eidx(m)) = fft(psi(sidx(m):eidx(m)));
end;

q0 = 1; q = ones(sum(phip), 1);
cumbeta = cumprod(beta);
z = zeros(s_max, 1); e2 = zeros(s_max, 1); x = zeros(s_max, 1);
```

```
if (verbose >= 1) && (m1 ~= m2)
  fprintf('Calculating rule for %d^%d to %d^%d\n', p, m1, p, m2);
elseif (verbose >= 1)
  fprintf('Calculating optimal CBC for %d^%d\n', p, m1);
end;
for s=1:s_max
  E2 = zeros(phip(m2), 1); X = zeros(phip(m2), m2-m1+1);
  C = beta(s) * q0; % accumulation of the constant parts goes in C
  E2(1) = gamma(s) * psi0 * q0;
  if p == 2, r = 1; else r = 2; end; % p = 2 is special
  v = 1; % number of choices from the previous iteration for m (next loop)
  for m=1:m2  % calculate error for each m
    C = C + beta(s) * r * sum(q(sidx(m):eidx(m)));
    E2(1:phip(m)) = repmat(E2(1:v), phip(m)/v, 1) ...
      + gamma(s) * r * real(ifft(fft_psi(sidx(m):eidx(m)) ...
        .* fft(q(sidx(m):eidx(m)))));
    if m >= m1 % store and calculate X where needed
      X(:, m-m1+1) = repmat(sqrt((-cumbeta(s) + C/p^m ...
        + E2(1:phip(m))/p^m) / CBC_optimal{m}(s)), ...
          phip(m2)/phip(m), 1);
    end;
    r = 2; v = phip(m);
  end;
  [x(s), w] = min(max(X, [], 2)); % pick the good choice
  [dummy, mloc] = max(X(w,:)); mloc = mloc + m1 - 1;
  if s == 1, w = 1; end;
  z(s) = perm(w); z(s) = min(z(s), n-z(s)); % unpermute
  e2(s) = -cumbeta(s) + C/n + E2(w)/n; % calculate real worst-case error
  q0 = (beta(s) + gamma(s) * psi0) * q0; % update q...
  for m=1:m2                                % in turn for each m
    ww = mod(w-1, phip(m)) + 1;
    q(sidx(m):eidx(m)) = (beta(s) + gamma(s) ...
      * psi([ww:-1:1 phip(m):-1:ww+1]+sidx(m)-1)) ...
        .* q(sidx(m):eidx(m));
  end;
  if verbose >= 2
    fprintf('s=%4d, z=%6d, e2=%.4e, x=%.4e, mloc=%4d\n', ...
            s, z(s), e2(s), x(s), mloc);
  end;
end;
```

Listing 5.3: Sequence construction for order dependent weights (`fastrank1exod.m`)

```
function [z, e2, x, e2s, CBC_optimal] = fastrank1exod(p, ...
  m1, m2, s_max, omega, Gamma, CBC_optimal, verbose)

% ... similar part to fastrank1expt.m lines 6 to 36 skipped ...

qstar = length(Gamma);
% q_{s,ell} can be found in q(:, ell+1)
q0 = zeros(1, qstar+1); q = zeros(sum(phip), qstar+1);
q0(0+1) = 1; q(:,0+1) = 1;
z = zeros(s_max, 1); e2 = zeros(s_max, 1); x = zeros(s_max, 1);
prev_e2 = zeros(m2-m1+1, 1);

if (verbose >= 1) && (m1 ~= m2)
  fprintf('Calculating rule for %d^%d to %d^%d\n', p, m1, p, m2);
elseif (verbose >= 1)
  fprintf('Calculating optimal CBC for %d^%d\n', p, m1);
end;
for s=1:s_max
  E2 = zeros(phip(m2), 1); X = zeros(phip(m2), m2-m1+1);
  % calculate S = sum_ell Gamma_ell q_{s-1,ell-1}
  S0 = sum(q0([0:qstar-1]+1) .* Gamma);
  S  = sum(q(:,[0:qstar-1]+1) * diag(Gamma), 2);
  E2(1) = psi0 * S0;
  if p == 2, r = 1; else r = 2; end; % p = 2 is special
  v = 1; % number of choices from the previous iteration for m (next loop)
  for m=1:m2  % calculate error for each m
    E2(1:phip(m)) = repmat(E2(1:v), phip(m)/v, 1) ...
      + r * real(ifft(fft_psi(sidx(m):eidx(m)) ...
        .* fft(S(sidx(m):eidx(m)))));
    if m >= m1 % store and calculate X where needed
      X(:, m-m1+1) = repmat(sqrt((prev_e2(m-m1+1) ...
        + E2(1:phip(m))/p^m) / CBC_optimal{m}(s)), ...
          phip(m2)/phip(m), 1);
    end;
    r = 2; v = phip(m);
  end;
```

```
  [x(s), w(s)] = min(max(X, [], 2)); % pick the good choice
  [dummy, mloc] = max(X(w(s),:)); mloc = mloc + m1 - 1;
  if s == 1, w(s) = 1; end;
  z(s) = perm(w(s)); z(s) = min(z(s), n-z(s)); % unpermute
  e2(s) = prev_e2(end) + E2(w(s))/n; % calculate real worst-case error
  % update q...
  for ell=min(qstar, s+1):-1:1 % for each ℓ...
    q0(ell+1) = q0(ell+1) + psi0 * q0(ell-1+1);
    for m=1:m2                 % and for each m
      ww = mod(w(s)-1, phip(m)) + 1; J = [sidx(m):eidx(m)];
      q(J,ell+1) = q(J,ell+1) ...
        + psi([ww:-1:1 phip(m):-1:ww+1]+sidx(m)-1) ...
          .* q(J,ell-1+1);
    end;
  end;
  % calculate prev_e2 for next iteration
  S0 = sum(q0([1:qstar]+1) .* Gamma);
  S = sum(q(:,[1:qstar]+1) * diag(Gamma), 2);
  prev_e2(1) = S0;
  if p == 2, r = 1; else r = 2; end;
  for m=1:m2
    prev_e2(max(1,m-m1+1)) = prev_e2(max(1,m-m1)) ...
      + r * sum(S(sidx(m):eidx(m)));
    r = 2;
  end;
  prev_e2 = prev_e2 ./ p.^[m1:m2]';
  if verbose >= 2
    fprintf('s=%4d, z=%6d, e2=%.4e, x=%.4e, mloc=%4d\n', ...
            s, z(s), e2(s), x(s), mloc);
  end;
end;
e2s = prev_e2;
```

5.8 Sequence usage of a nonembedded lattice rule

In §5.5 we presented a method to use a lattice with b^m points as a (finite) sequence. This is done by generating the points as

$$\boldsymbol{x}_k = \{\psi_b(k)\,\boldsymbol{z}\} \qquad \text{for} \quad k = 0, 1, \ldots, b^m - 1, \tag{5.9}$$

with $\psi_b(k)$ a one-dimensional low-discrepancy sequence in base b. Obviously this method can be used whether or not the lattice rule was constructed with Algorithm 5. But could a similar method be conceived for lattice rules where the number of points n is not a power of a prime?

Assume that ψ_b is the Gray order, see Definition 5.6. For a composite n which has small prime factors, a mixed radix Gray code could be used to obtain the same effect as (5.9). For large prime factors this becomes less useful, and for n prime the Gray code sequence has no effect. A simple acceptance rejection strategy, while using a Gray code sequence for a small prime base b, could be used, such that $n \leq b^m$ for some m. This strategy delivers a low-discrepancy ordering in the one-dimensional projections since if T is a low-discrepancy ordering over $[0, 1)$ then $T \cap [0, 1 - v)$ is also a low-discrepancy ordering over $[0, 1 - v)$. Algorithm 6 gives then sequence usage for arbitrary lattice rules.

Algorithm 6 Sequence usage of arbitrary lattice rules in Gray ordering

find a small prime b, and m, such that $n \leq b^m$
$k' = 0$, $k = 0$
for $i = 0$ **to** $n - 1$ **do**
  repeat
    if $i = 0$ **then break**; **end if** {to generate the first point $\boldsymbol{x}_0 = \boldsymbol{0}$}
    find $c(k)$, the position of the least significand digit in k differing from $b - 1$
    $k' = k' \oplus_b b^{m-(c(k)+1)}$ {this is $b^m G_{b^{-1}}(k + 1)$}
    $k = k + 1$
  until $k' \geq n$ **and** $i \neq 0$
  use the point $\{k'\boldsymbol{z}/n\}$
end for

In Chapter 8 an example is presented where a lattice rule is used as a sequence but without the rule being specially constructed for this. In doing so, hard guarantees on the convergence are sacrificed in favor of almost complete flexibility in the choice of lattice rule. This also means that the number of points does not need to be specified in advance (that is as long as it does not exceed the total number of points in the lattice rule). Using the lattice as the basis of a randomly shifted lattice rule, as was explained in §1.2.4, still gives a useful statistical estimate of the error, justifying the blind use of any lattice as a sequence. Using this method on lattices constructed in line with the principles of this chapter and [46] of course does provide guarantees for the worst-case error of all the embedded rules.

5.9 A wavelet generated reproducing kernel Hilbert space

From the images in §4.5 and §4.7 for $n = p^m$ it might appear that using a reproducing kernel Hilbert space with a wavelet basis could provide a benefit for the construction. What is then needed is that the $\boldsymbol{\Omega}_{p^\ell}$ matrices are not only embedded, but also lower resolution approximations of each other. In this way, also the subsequent approximations of $I(f)$ could be seen as a multi-resolution approximation. In this section the very simple case of Haar wavelets in base 2 will be considered. In Chapter 6 this space will turn out to be the Walsh space and it follows that one needs to use polynomial lattice rules to make this function space (digital) shift-invariant.

Consider the classical Haar wavelets in base 2 on the interval $[0,1)$

$$\phi(x) := 1,$$
$$\psi_{r,i}(x) := \begin{cases} 0 & \text{if } x \notin 2^{-r}[i, i+1), \\ 2^{r/2} & \text{if } x \in 2^{-r}[i, i+\frac{1}{2}), \\ -2^{r/2} & \text{if } x \in 2^{-r}[i+\frac{1}{2}, i+1). \end{cases}$$

The following is standard. Define $\mathcal{S}^r$ as the space of piecewise constant functions with breakpoints at the points $i\,2^{-r}$, then

$$\mathcal{S}^\infty := \overline{\bigcup_r \mathcal{S}^r} = L_2([0,1)) \qquad \text{and} \qquad \mathcal{S}^{-\infty} := \bigcap_r \mathcal{S}^r = \{0\}.$$

Similarly define $\mathcal{W}^r$ as the resolution which needs to be added to the space $\mathcal{S}^r$ to obtain $\mathcal{S}^{r+1}$, i.e., $\mathcal{W}^r := \mathcal{S}^{r+1} \ominus \mathcal{S}^r$, then

$$L_2([0,1)) = \bigoplus_{r\in\mathbb{Z}} \mathcal{W}^r = \mathcal{S}^0 + \bigoplus_{r=0}^{\infty} \mathcal{W}^r.$$

For any $f \in L_2([0,1))$

$$f(x) = \langle f, \phi\rangle\, \phi(x) + \sum_{r=0}^{\infty} \sum_{i=0}^{2^r-1} \langle f, \psi_{r,i}\rangle\, \psi_{r,i}(x),$$

with $\langle\cdot,\cdot\rangle$ the standard inner product in L_2.

Now using Theorem 2.9 which states that the reproducing kernel of a separable Hilbert space can be written as

$$K(x,y) = \sum_k \varphi_k(x)\, \overline{\varphi_k(y)},$$

the kernel for a one-dimensional space with such a wavelet basis could be defined. Since L_2 is not a reproducing kernel Hilbert space (point evaluation is not bounded), we choose to reduce higher resolution components (i.e., higher frequencies), similar to what is done for the Korobov space. The following kernel uses a smoothness parameter $\alpha > 1$ for this purpose. The kernel for this one-dimensional Haar wavelet generated reproducing kernel Hilbert space is then

$$K(x,y) = 1 + \sum_{r=0}^{\infty} \sum_{i=0}^{2^r-1} 2^{-r\alpha}\, \psi_{r,i}(x)\, \overline{\psi_{r,i}(y)}.$$

(The inner product in this space inserts the weights in the same way as in the Korobov space.) The s-dimensional space is just taken as the tensor-product space and its kernel is thus the product of these one-dimensional kernels.

Although this kernel is not shift-invariant, and the desired effect of having a multi-resolution structure imposed on the embedding of the matrices is thus not obtained, it has very interesting properties. Define

$$\phi(x,y) = \sum_{r=0}^{\infty} \sum_{i=0}^{2^r-1} 2^{-r\alpha}\, \psi_{r,i}(x)\, \overline{\psi_{r,i}(y)}.$$

Due to the limited support of the wavelets this function divides its domain $[0,1)^2$ recursively in squares with sides 2^{-r}, and only the squares on the diagonal are subdivided further (due to the support of the wavelets). Therefore, the function ϕ can be written in terms of one argument which measures the "bitwise distance" between x and y. Using $\ominus_2$ to denote the XOR-operation in base 2, we find:

$$\begin{aligned}
\phi(x,y) &= \phi(x \ominus_2 y, 0) \\
&= \sum_{r=0}^{\infty} \sum_{i=0}^{2^r-1} 2^{-r\alpha}\, \psi_{r,i}(x \ominus_2 y)\, \overline{\psi_{r,i}(0)} \\
&= \sum_{r=0}^{\infty} 2^{-r\alpha} \begin{cases} 2^r & \text{if } 0 \le x \ominus_2 y < 2^{-r+1}, \\ -2^r & \text{if } 2^{-r+1} \le x \ominus_2 y < 2^{-r}, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{5.10}$$

As with shift-invariant functions, the shorthand $\phi(x \ominus y)$ is used instead of the more verbose $\phi(x \ominus_2 y, 0)$. In Chapter 6 it will be shown that this kernel is therefore digital shift-invariant (and also that fast construction is possible). A closed form expression for (5.10) can easily be found by noticing that the function is piecewise

constant. Set $t = -\lfloor \log_2(x) \rfloor$ then

$$\begin{aligned}\phi(x) &= \sum_{r=0}^{\infty} 2^{-r\alpha} \begin{cases} 2^r & \text{if } r < t-1, \\ -2^r & \text{if } r < t, \\ 0 & \text{otherwise} \end{cases} \\ &= \sum_{r=0}^{t-2} 2^{r(1-\alpha)} - 2^{(t-1)(1-\alpha)} \\ &= \frac{1 - 2^{(t-1)(1-\alpha)}}{1 - 2^{(1-\alpha)}} - 2^{(t-1)(1-\alpha)}.\end{aligned}$$

For $\alpha = 2$ this becomes

$$\begin{aligned}\phi(x) &= 2(1 - 2^{1-t}) - 2^{1-t} \\ &= 2 - 2^{-t}\, 6. \end{aligned} \tag{5.11}$$

The worst-case error for this space using product weights is given by

$$e(P_n, K) = \left(-1 + \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} \prod_{j=1}^{s} \left(1 + \gamma_j\, \phi(x_j^{(k)} \ominus_2 y_j^{(\ell)}) \right) \right)^{1/2},$$

with $\boldsymbol{x}_k, \boldsymbol{y}_\ell \in P_n$. In the spirit of Chapter 3 one could store the previous products and write the squared worst-case error for s dimensions as

$$e_s^2(P_n, K) = -1 + \frac{1}{n^2} \sum_{k=0}^{n-1} \sum_{\ell=0}^{n-1} p_{s-1}(k, \ell) \, \left(1 + \gamma_s\, \phi(x_s^{(k)} \ominus_2 y_s^{(\ell)}) \right). \tag{5.12}$$

If for $\boldsymbol{x}, \boldsymbol{y} \in P_n$ also $\boldsymbol{x} \ominus_2 \boldsymbol{y} \in P_n$ (which is the case when P_n is the node set of a lattice rule), then one expects to be able to do the same as in Chapter 3 and obtain fast construction for this space. However, for P_n the node set of a rank-1 lattice rule

$$\begin{aligned} x_s^{(k)} \ominus_2 y_s^{(\ell)} &= \frac{k \cdot z_s}{n} \ominus_2 \frac{\ell \cdot z_s}{n} \\ &\neq \frac{(k \ominus_2 \ell) \cdot z_s}{n}, \end{aligned}$$

where the $\cdot$ denotes multiplication modulo n. As such the double sum in (5.12) can not be replaced by a single sum. In the case of a shift-invariant kernel this is

$$\begin{aligned} x_s^{(k)} - y_s^{(\ell)} &= \frac{k \cdot z_s}{n} - \frac{\ell \cdot z_s}{n} \\ &= \frac{(k - \ell) \cdot z_s}{n}. \end{aligned}$$

It turns out that the multiplication and the subtraction are incompatible for the Haar kernel. This will be solved in the next chapter by using polynomials instead of scalars for which polynomial multiplication will be distributive over the (polynomial) subtraction $\ominus$.

Chapter notes

The fast construction of good embedded lattice rules was published in [15] and was joint work with Frances Y. Kuo. The work in [15] also benefitted from comments by Ian H. Sloan. The acceptance rejection strategy to use arbitrary lattice rules as a sequence appeared in [17]. Fast construction of (polynomial) lattice rules for Haar wavelet spaces was presented in Dundee at NA05 as a side effect of researching the multi-resolution structure in the matrices $\boldsymbol{\Omega}_{p^m}$.

In a recent paper [27], Dick, Pillichshammer and Waterhouse provide an alternative method to construct good extensible lattice rules by ways of a sieve method. As a side effect they also proved that the construction presented in this chapter obtains the optimal rate of convergence. The practical implementation of their method also makes use of the fast component-by-component algorithm.

Chapter 6

Construction of polynomial lattice rules

In [76, §4.4] Niederreiter describes a special construction of (t, m, s)-nets which look very much like lattice rules. In fact the algebraic construction is exactly like that of lattice rules, but instead of working with a ring or a field of scalars, a ring of polynomials is used. The elements of this algebraic structure form an additive abelian group and hence have lattice structure according to Definition 1.8. They are mapped to real vectors in $[0, 1)^s$ to form a low-discrepancy point set which has net-structure. It was shown by Dick, Kuo, Pillichshammer and Sloan [24] that also in this case the component-by-component algorithm can be applied. The nets constructed in this way are then optimized for the worst-case error instead of for the lowest t-value.

6.1 Polynomial lattice rules

Let $\mathbb{F}_q[x]$ denote the ring of polynomials in the formal variable x over the finite field $\mathbb{F}_q$ of order q. Given such a polynomial f, let $\mathbb{F}_q[x]/f$ be the factor group modulo f and $\deg(f)$ the degree of f. If we replace all occurrences of k, $\boldsymbol{z}$ and n in the previous chapters with such polynomials, we obtain polynomial lattice rules. Since now the number of points and the modulus are two separate things, f is used to denote the (polynomial) modulus and n is used to denote the number of points. A simplified definition for polynomial lattice rules for $q = p$, a prime, is given as follows. (For the full definition see, e.g., [24].)

Definition 6.1. A *polynomial lattice rule in base p* with modulus $f \in \mathbb{F}_p[x]$ and

$\deg(f) = m$ has a point set

$$\boldsymbol{x}_k = v_m\left(\frac{k(x)\cdot \boldsymbol{z}(x) \bmod f(x)}{f(x)}\right) \qquad \text{for } k \in \mathbb{F}_p[x]/f.$$

The generating vector $\boldsymbol{z}$ has components $z_j \in (\mathbb{F}_p[x]/f)^\times$ and v_m is a componentwise mapping from the formal Laurent series $\mathbb{F}_p((x^{-1}))$ to the p-adic rationals in $[0,1)$. The jth component of the kth point is then defined as

$$x_j^{(k)} = v_m\left(\frac{k(x)\cdot z_j(x) \bmod f(x)}{f(x)}\right) = v_m\left(\sum_{\ell=1}^{\infty} w_\ell\, x^{-\ell}\right) = \sum_{\ell=1}^{m} w_\ell\, p^{-\ell}, \qquad (6.1)$$

where all $w_\ell \in \mathbb{F}_p$. Note that since p is prime, $\mathbb{F}_p$ is equivalent to $\mathbb{Z}_p$.

The similarities with the scalar rank-1 rules are obvious. Since $k(x) \in \mathbb{F}_p[x]/f$, with $\deg(f) = m$, it follows that k has exactly $n = p^m$ different values. A direct mapping of the scalar $k \in \mathbb{Z}_n$ is given by considering the p-adic expansion

$$k = k_{m-1}\, p^{m-1} + \cdots + k_1\, p + k_0,$$

with all $k_\ell \in \mathbb{Z}_p$, and then using the associated polynomial

$$k(x) = k_{m-1}\, x^{m-1} + \cdots + k_1\, x + k_0.$$

Similarly $(\mathbb{F}_p[x]/f)^\times$ is the exact equivalent of $U_n = \mathbb{Z}_n^\times$ in the scalar case because it contains those elements which have a multiplicative inverse, i.e., those $g \in \mathbb{F}_p[x]/f$ which have $\gcd(g, f) = 1$.

Since we consider p prime and thus $\mathbb{F}_p = \mathbb{Z}_p$, it follows that the actual rational values of the components of the points are exactly the same as for a scalar lattice rule with p^m points, but here a different set of permutations corresponds to each $z_j(x)$. (The mapping (6.1) could also allow the p-adic rationals to have more than m digits while still forming a (t, m, s)-net in base p.)

6.2 Worst-case errors for polynomial lattice rules

In order to define a worst-case error for the use of polynomial lattice rules, a reproducing kernel Hilbert space is needed. In [26] Dick and Pillichshammer make the connection between the theory of lattice rules as presented in Chapter 2 and that of digital nets apparent. Their main tool is a function space based on Walsh functions.

6.2.1 The digital Walsh space

First we define the one-dimensional Walsh functions in arbitrary base b.

Definition 6.2. The *Walsh functions in base* b, $\mathrm{wal}_{b,h} : [0,1) \to \mathbb{C}$, for $h = 0, 1, 2, \ldots$ are given by

$$\mathrm{wal}_{b,h}(x) := \exp(2\pi \mathrm{i}\, (x_1 h_0 + x_2 h_1 + \cdots + x_m h_{m-1})/b),$$

where $h = (h_{m-1} \ldots h_0)_b$, for some m, and $x = (0.x_1 x_2 \ldots)_b \in [0,1)$ in radix b.

The Walsh functions form a complete orthonormal basis for $L_2([0,1))$. An important property of the Walsh functions, following immediately from the definition, is that

$$\mathrm{wal}_{b,k}(x)\, \overline{\mathrm{wal}_{b,k}(y)} = \mathrm{wal}_{b,k}(x \ominus_b y),$$

with $\ominus_b$ digitwise subtraction modulo b.

Multivariate Walsh functions with index $\boldsymbol{h}$ are then defined as the product of the corresponding one-dimensional Walsh functions:

$$\mathrm{wal}_{b,\boldsymbol{h}}(\boldsymbol{x}) := \prod_{j=1}^{s} \mathrm{wal}_{b,h_j}(x_j).$$

The function space setting that will be considered here is a reproducing kernel Hilbert space based on Walsh functions similar to the Fourier series representation of a function as in Chapter 2. Consider now functions $f \in L_2([0,1)^s)$ in their Walsh expansion

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{h} \in \mathbb{Z}_*^s} \hat{f}_b(\boldsymbol{h})\ \mathrm{wal}_{b,\boldsymbol{h}}(\boldsymbol{x}),$$

with $\mathbb{Z}_* = \{0, 1, 2, \ldots\}$ and the Walsh coefficients $\hat{f}_b(\boldsymbol{h})$ defined as

$$\hat{f}_b(\boldsymbol{h}) := \int_{[0,1)^s} f(\boldsymbol{x})\, \overline{\mathrm{wal}_{b,\boldsymbol{h}}(\boldsymbol{x})}\, \mathrm{d}\boldsymbol{x}. \tag{6.2}$$

For simplicity we follow [26] and define a tensor-product space with product weights (the other weighted spaces follow similar to what was done in Chapter 2). We start from the one-dimensional space. For $\alpha > 1$ define the inner product

$$\langle f, g \rangle = \sum_{h=0}^{\infty} r_b(h)^\alpha\, \hat{f}_b(h)\, \overline{\hat{g}_b(h)},$$

with

$$r_b(h) := \begin{cases} 1 & \text{if } h = 0, \\ b^{\lfloor \log_b(h) \rfloor} & \text{otherwise,} \end{cases} \tag{6.3}$$

where $\log_b$ is the logarithm in base b. We thus have that $\lfloor \log_b(h) \rfloor$ is the highest power of b appearing in the b-adic expansion of h. Functions in this one-dimensional space then have

$$\|f\| := \left(\sum_{h=0}^{\infty} r_b(h)^{\alpha} \, |\hat{f}_b(h)| \right)^{1/2} < \infty.$$

Similar to the Fourier series representation in Chapter 2, α must be strictly larger than 1 since here

$$\mu_b(\alpha) := \sum_{h=1}^{\infty} r_b(h)^{-\alpha} = \sum_{h=1}^{\infty} b^{-\alpha \lfloor \log_b(h) \rfloor} = (b-1) \sum_{\ell=1}^{\infty} (b^{-\alpha+1})^{\ell} = \frac{(b-1)\, b^{\alpha}}{b^{\alpha} - b}$$

is positive and finite for $\alpha > 1$ (and the last part is only valid for $\alpha > 1$).

The reproducing kernel is then given by

$$\begin{aligned} K(x,y) &:= \sum_{h=0}^{\infty} r(h)^{-\alpha} \; \mathrm{wal}_{b,h}(x) \overline{\mathrm{wal}_{b,h}(y)} \\ &= 1 + \sum_{h=1}^{\infty} r(h)^{-\alpha} \; \mathrm{wal}_{b,h}(x) \overline{\mathrm{wal}_{b,h}(y)} \\ &= 1 + \phi_{b,\alpha}(x,y), \end{aligned}$$

where from [26] the infinite sum can be replaced with a closed form expression

$$\phi_{b,\alpha}(x,y) = \begin{cases} \mu_b(\alpha) & \text{if } x = y, \\ \mu_b(\alpha) - b^{(1-t)(\alpha-1)}(\mu_b(\alpha)+1) & \text{otherwise, } t = -\lfloor \log_b(x \ominus_b y) \rfloor. \end{cases}$$

(Using $\log_b(0) = -\infty$ and $b^{-\infty} = 0$ the first case could be ignored.)

The kernel of the tensor-product space with product weights for the Walsh space is then given by

$$K(\boldsymbol{x}, \boldsymbol{y}) := \prod_{j=1}^{s} \left(1 + \gamma_j \, \phi_{b,\alpha}(x_j, y_j)\right).$$

This kernel is digital shift-invariant as in the following definition.

Definition 6.3. A reproducing kernel is *digital shift-invariant* in base b if for all $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Delta} \in [0,1)^s$

$$K(\boldsymbol{x} \oplus_b \boldsymbol{\Delta}, \boldsymbol{y} \oplus_b \boldsymbol{\Delta}) = K(\boldsymbol{x}, \boldsymbol{y}),$$

where $\oplus_b$ is digitwise addition modulo b.

Corollary 6.4. *If a reproducing kernel is digital shift-invariant in base b then it can be written in terms of one variable*

$$K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{x} \ominus_b \boldsymbol{y}, \boldsymbol{0}) =: K(\boldsymbol{x} \ominus_b \boldsymbol{y}),$$

where $\ominus_b$ is digitwise subtraction modulo b.

Following from Theorem 2.18, the worst-case error for this space and P_n the node set of a polynomial lattice rule is given by

$$e(P_n, K) = \left(-1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(1 + \gamma_j \, \phi_{b,\alpha}(x_j^{(k)}) \right) \right)^{1/2}, \tag{6.4}$$

where $\phi_{b,\alpha}(x)$ is a shorthand for $\phi_{b,\alpha}(x, 0)$ and has the role of the ω function as in the previous chapters. To derive (6.4) the facts that K is digital shift-invariant and that for P_n the point set of a polynomial lattice rule

$$\forall \boldsymbol{x}, \boldsymbol{y} \in P_n : \boldsymbol{x} \ominus_b \boldsymbol{y} \in P_n$$

was used.

The most interesting case for the Walsh function space is to take the base $b = 2$ (for computational efficiency and programming ease) and $\alpha = 2$. One then obtains the following variable part of the reproducing kernel

$$\begin{aligned} \omega(x) &= \phi_{b,\alpha}(x) \\ &= 2 - 6 \cdot 2^{\lfloor \log_2(x) \rfloor} \qquad (6.5) \\ &= 12 \left(\frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1} \right). \qquad (6.6) \end{aligned}$$

Equation (6.5) is exactly the kernel from §5.9 for the wavelet generated reproducing kernel Hilbert space based on Haar wavelets in base 2 and smoothness $\alpha = 2$, see (5.11). The reason for the second form (6.6) with the 12 in front will become apparent in the next section.

6.2.2 Associated digital shift-invariant Sobolev spaces

Further mirroring the theory from Chapter 2 one can define an associated digital shift-invariant kernel $K^{\mathrm{ds},b}$ if K is not digital shift-invariant.

Definition 6.5. For an arbitrary reproducing kernel K, define the associated digital shift-invariant kernel $K^{\mathrm{ds},b}$ in base b by

$$K^{\mathrm{ds},b}(\boldsymbol{x}, \boldsymbol{y}) := \int_{[0,1)^s} K(\boldsymbol{x} \oplus_b \boldsymbol{\Delta}, \boldsymbol{y} \oplus_b \boldsymbol{\Delta}) \, \mathrm{d}\boldsymbol{\Delta} =: K^{\mathrm{ds},b}(\boldsymbol{x} \ominus_b \boldsymbol{y}).$$

Equivalently Theorem 2.22 holds, but then for digital shifts:

$$E_{\boldsymbol{\Delta}}\left[e^2(P_n \oplus_b \boldsymbol{\Delta}, K)\right] = e^2(P_n, K^{\mathrm{ds},b}).$$

In [26] it is shown that the associated digital shift-invariant kernel for the unanchored Sobolev space with product weights is given by

$$K^{\mathrm{ds},b}(\boldsymbol{x}) = \prod_{j=1}^{s} \left(1 + \gamma_j\, \phi_b(x_j)\right),$$

with

$$\phi_b(x) = \begin{cases} \dfrac{1}{6} & \text{if } x = 0, \\ \dfrac{1}{6} - \dfrac{x_t(b - x_t)}{b^{t+1}} & \text{otherwise, where } t = -\lfloor \log_b(x) \rfloor. \end{cases}$$

For $b = 2$ this simply becomes

$$\phi_2(x) = \frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1},$$

again using $\log_2(0) = -\infty$ and $2^{-\infty} = 0$.

Also for the anchored Sobolev space an associated digital shift-invariant kernel can be obtained. In [26] the following expression is obtained for an anchor $\boldsymbol{a}$

$$K^{\mathrm{ds},b}(\boldsymbol{x}) = \prod_{j=1}^{s} \left(1 + \gamma_j\, \phi_b(x_j)\right),$$

with

$$\phi_b(x) = \begin{cases} a_j^2 - a_j + \dfrac{1}{2} & \text{if } x = 0, \\ a_j^2 - a_j + \dfrac{1}{2} - \dfrac{x_t(b - x_t)}{b^{t+1}} & \text{otherwise, where } t = -\lfloor \log_b(x) \rfloor. \end{cases}$$

For $b = 2$ and all a_j equal to 1 or 0 this simply becomes

$$\phi_2(x) = \frac{1}{2} - 2^{\lfloor \log_2(x) \rfloor - 1}.$$

As such the associated kernel for the anchored space can be written as

$$K(\boldsymbol{x}) = \prod_{j=1}^{s} \left(\left(1 + \frac{\gamma_j}{3}\right) + \gamma_j \left(\frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1}\right)\right).$$

From this example we arrive at a similar form as (2.24) to calculate the worst-case error as

$$e(P_n, K^{\mathrm{ds},b}) = \left(-\prod_{j=1}^{s} \beta_j + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{s} \left(\beta_j + \gamma_j \, \omega(x_j^{(k)}) \right) \right)^{1/2}. \tag{6.7}$$

For the anchored Sobolev space we set $\beta_j = 1 + \gamma_j/3$.

As with the scalar lattice rules, the kernels of the associated digital shift-invariant Sobolev spaces look similar to the kernel of the Walsh space, which takes the role of the Korobov space for the scalar lattice rules. The role of the Bernoulli polynomial of degree 2 is here taken by the function $\frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1}$. The difference between the Korobov space and the unanchored Sobolev space is a factor $2\pi^2$, in this case it is a factor 12.

6.3 Fast construction of polynomial lattice rules

In this section we set $b = p$ prime again. When f is an irreducible polynomial, $\mathbb{Z}_p[x]/f$ is a field and $(\mathbb{Z}_p[x]/f)^\times = \mathbb{Z}_p[x]/f \setminus \{0\}$. Thus there are $p^m - 1$ possible choices for each component of the generating vector.

Now define the $(p^m - 1) \times p^m$ matrix

$$\boldsymbol{\Omega}_{p^m} = \left[\omega\left(v_m \left(\frac{k(x) \cdot z(x)}{f(x)} \right) \right) \right]_{\substack{z(x) \in (\mathbb{Z}_p[x]/f)^\times \\ k(x) \in \mathbb{Z}_p[x]/f}} \tag{6.8}$$

and compare with (3.6) as well as (4.2). The product of k and z is performed in the field $\mathbb{Z}_p[x]/f$, i.e., modulo the irreducible polynomial f. This is the equivalent of the product modulo the prime n in the scalar case. Since the multiplicative group of every finite field is cyclic, a polynomial which generates all of $(\mathbb{Z}_p[x]/f)^\times$ can always be found. Given such a generator g for $(\mathbb{Z}_p[x]/f)^\times$ the matrix can be permuted into the same circulant form as in Chapter 3 and fast construction is obtained in exactly the same way as for the scalar lattice rules.

Theorem 6.6. *Component-by-component construction of a p^m-point polynomial rank-1 lattice in base p modulo f in s dimensions, with f an irreducible polynomial, for a digital shift-invariant reproducing kernel Hilbert space can be done in time $O(s(n \log(n) + tn))$ and memory $O(tn)$, where*

(*i*) $t = 1$ *for product weights;*
(*ii*) $t = q^\star$ *for order dependent weights of order $q^\star \leq s$;*

Proof. This is the exact equivalent of Theorems 3.7 and 3.8. □

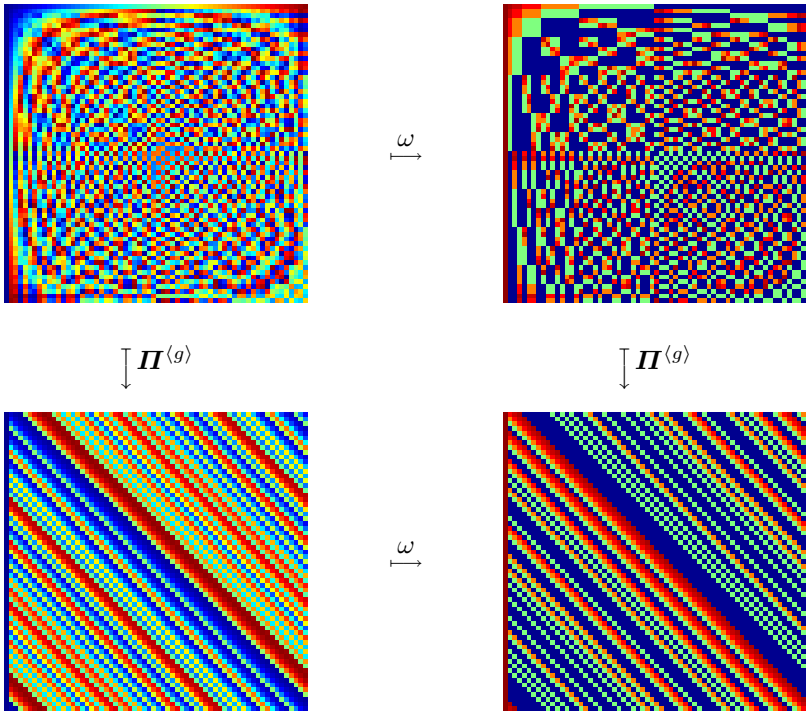


Figure 6.1: The structure of the matrices $\boldsymbol{\Xi}_{p^m}$ and $\boldsymbol{\Omega}_{p^m}$ (top row) for the polynomial lattice rules looks different than for the scalar case. However, the same technique can be applied, getting the matrices $\boldsymbol{\Xi}_{p^m}^{\langle g \rangle}$ and $\boldsymbol{\Omega}_{p^m}^{\langle g \rangle}$ (bottom row) with g a generator for $\mathbb{F}_p[x]/f$.

Obviously, the structure of this matrix is the modulo multiplication structure from Chapter 4, i.e., the matrix $\boldsymbol{\Xi}_{p^m}$, but then for polynomial multiplication. The symmetry $\omega(x) = \omega(1-x)$ is however not present here and thus one can not use Theorem 3.6. It is interesting to note that in this case the ω function only has m different values (and this should be exploitable in one way or another). The matrices $\boldsymbol{\Xi}_{p^m}$ and $\boldsymbol{\Omega}_{p^m}$ are shown in Figure 6.1 together with the permuted circulant form.

6.4 A Matlab implementation for $\mathbb{F}_2[x]$

With respect to the Matlab scripts in Chapter 3, for polynomial lattice rules the initialization of the algorithm becomes only slightly more difficult. After choosing an irreducible polynomial f over $\mathbb{Z}_p$, one has to find a generator for $(\mathbb{Z}_p[x]/f)^\times$ and calculate all its positive powers. Then find the Laurent series expansion for

all g^i/f, $0 \le i < p^m - 1$, and apply the functions v_m and ω. However, the core of the algorithm, which is the matrix-vector multiplication, remains the same.

There is no need to find a generator for $(\mathbb{Z}_p[x]/f)^\times$ if the irreducible polynomial f would be taken as a primitive polynomial since then, by definition, x is always a generator. In [24] the usage of different irreducible polynomials with respect to the worst-case error is compared without noticing much difference. Only in the comparison for 2^{10} one of the irreducible polynomials is not primitive but has similar worst-case error as the primitive ones. (The other nonprimitive polynomials are for the cases 2^8 and 2^{12}, but no other irreducible polynomials are tested.) Taking f a primitive polynomial simplifies the implementation while the quality of the constructed rules is probably not reduced.

From Listing 3.1 we can immediately derive a polynomial version for product weights. We restrict ourselves to the case $p = 2$ since Matlab (and also Octave) provides a Galois class for this case which uses a primitive polynomial as the modulus. This means that only the mapping v_m has to be implemented. An example implementation is given in Listing 6.1 and the function v_m is implemented in Listing 6.2. The primitive polynomial which is implicitly used in Listing 6.1 is the default provided by Matlab, which is the minimal primitive polynomial for a field of order 2^m. By changing the line `g = gf(2, m)` into `g = gf(2, m, f)`, with `f` the decimal representation of a primitive polynomial of degree m, a different primitive polynomial can be used. When `g` is an object of the Galois class, then `g.x` gives the decimal representation of the polynomial and `g.prim_poly` gives the decimal representation of the primitive polynomial used for the field.

An order dependent version for polynomial lattice rules can be obtained with similar changes to Listing 3.4. Therefore, we do not give an example implementation. Just combining Listing 6.1 with Listing 3.4 is enough.

Implementing the mapping function v_m is straightforward, since for calculating the Laurent expansion

$$\frac{k(x)}{f(x)} = w(x)$$

over $\mathbb{F}_p$ we can look at

$$\begin{aligned} & k(x) = w(x)\ f(x) \\ \Leftrightarrow & \sum_{\ell=0}^{m-1} k_\ell\, x^\ell = \sum_{\ell=1}^{\infty} w_\ell\, x^{-\ell} \sum_{\ell=0}^{m} f_\ell\, x^\ell, \end{aligned}$$

where $f_m = 1$ since f is a primitive polynomial. Now matching the powers of x gives the following recursive formula for the w_ℓ, $1 \le \ell \le m$,

$$w_\ell \equiv k_{m-\ell} - \sum_{j=1}^{\ell-1} w_j\, f_{m-(\ell-j)} \pmod p. \tag{6.9}$$

Listing 6.1: Fast construction for $\mathbb{F}_2$ and product weights (`fastpoly2rank1pt.m`)

```
function [z, e2] = fastpoly2rank1pt(m, s_max, omega, gamma, beta)

z = zeros(s_max, 1);
e2 = zeros(s_max, 1);
n = pow2(m);

g = gf(2, m); % generator g = 2 = (10)_2 = x since f is primitive
perm = gf(zeros(n-1, 1), m);
perm(1) = 1; for j=1:n-2, perm(j+1) = perm(j)*g; end;
psi = omega(vm(perm.x, g.prim_poly, m)');
psi0 = omega(0);
fft_psi = fft(psi);

E2 = zeros(n, 1);
cumbeta = cumprod(beta);
q = ones(n-1, 1);
q0 = 1;

for s = 1:s_max
    E2 = real(ifft(fft_psi .* fft(q)));
    [min_E2, w] = min(E2);
    if s == 1, w = 1; noise = abs(E2(1) - min_E2), end;
    z(s) = getfield(perm(w), 'x');
    e2(s) = -cumbeta(s) + ( beta(s) * (q0 + sum(q)) + ...
              gamma(s) * (psi0*q0 + min_E2) ) / n;
    q = (beta(s) + gamma(s) * psi([w:-1:1 n-1:-1:w+1])) .* q;
    q0 = (beta(s) + gamma(s) * psi0) * q0;
    fprintf('s=%4d, z=%6d, w=%6d, e2=%.4e, e=%.4e\n', ...
            s, z(s), w, e2(s), sqrt(e2(s)));
end;
```

Listing 6.2: Mapping Laurent series over $\mathbb{Z}_2$ to $[0,1)$ (`vm.m`)

```
function w = vm(k, f, m)

k = reshape(k, 1, length(k)); % just for vectorization
w = zeros(m, length(k), class(f));
for ell = 1:m
  w(ell,:) = bitget(k, m-ell+1); % bits are indexed from 1
  for j = 1:ell-1
    w(ell,:) = mod(w(ell,:) - w(j,:) * bitget(f, m-(ell-j)+1), 2);
  end;
end;
w = pow2(-[1:m]) * double(w);
```

Chapter notes

Fast construction of polynomial lattice rules modulo irreducible polynomials was published in [84]. Also from the view of random number generators, polynomial lattice rules of Korobov form are of interest, see [69]. In [25] the construction of polynomial lattice rules for the weighted star discrepancy is discussed.

As suggested in [25] the construction for reducible polynomials f should be similar to the derivations in Chapter 4 since then $\mathbb{Z}_p[x]/f$ is not a field anymore but just an abelian ring.

Chapter 7

Copy rules

In the previous chapters only rank-1 lattice rules were considered. However, as explained in Chapter 1, there are more general lattice rules with rank r, $1 \leq r \leq s$, with s the number of dimensions. We show that for the family of so called copy rules and intermediate copy rules fast construction can be easily obtained.

7.1 The canonical form for higher rank rules

In Sloan and Lyness [92] a canonical form for lattice rules is derived from the fundamental theorem of abelian groups. The canonical form expresses a general lattice rule

$$Q(f) = \frac{1}{m} \sum_{k_1=0}^{n_1-1} \cdots \sum_{k_t=0}^{n_t-1} f\left(\left\{ \frac{k_1 \boldsymbol{z}_1}{n_1} + \cdots + \frac{k_t \boldsymbol{z}_t}{n_t} \right\} \right)$$

with $m = n_1 \cdots n_t$ into a minimal sums form. Remember that the number of distinct points n of such a lattice rule needs not be m, although $n \mid m$.

Theorem 7.1. *Every s-dimensional n-point lattice rule, with $n \geq 2$, can be written as*

$$Q(f) = \frac{1}{n} \sum_{k_1=0}^{n_1-1} \cdots \sum_{k_r=0}^{n_r-1} f\left(\left\{ \frac{k_1 \boldsymbol{z}_1}{n_1} + \cdots + \frac{k_r \boldsymbol{z}_r}{n_r} \right\} \right),$$

with r the rank of the lattice rule, $1 \leq r \leq s$, $n = n_1 \cdots n_r$, all $n_{i+1} \mid n_i$, the vectors $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_r$ linearly independent over $\mathbb{Z}^s$, and the components of $\boldsymbol{z}_i$ relatively prime to n_i for each i.

The proof of this theorem is a direct translation of a popular variant of the fundamental theorem of finite abelian groups.

Theorem 7.2. *Let G be a finite abelian group of order n, then there exist r unique integers $n_1, \ldots, n_r$, called the* invariants, *where each $n_{i+1} \mid n_i$, such that*

$$G \simeq \mathbb{Z}/\mathbb{Z}_{n_1} \oplus \mathbb{Z}/\mathbb{Z}_{n_2} \oplus \cdots \oplus \mathbb{Z}/\mathbb{Z}_{n_r}$$

and all $\mathbb{Z}/\mathbb{Z}_{n_i}$ are cyclic.

Proof. Starting from the fundamental theorem of finite abelian groups, which states that G can uniquely (up to reordering) be decomposed into a direct sum of prime power cyclic groups, G can be written as

$$G \simeq \mathbb{Z}/\mathbb{Z}_{p_1^{k_1}} \oplus \mathbb{Z}/\mathbb{Z}_{p_2^{k_2}} \oplus \cdots \oplus \mathbb{Z}/\mathbb{Z}_{p_t^{k^t}}, \tag{7.1}$$

where the p_i are prime numbers (not necessarily distinct) and $n = p_1^{k_1} \cdots p_t^{k_t}$. Now using that for m and ℓ relatively prime

$$\mathbb{Z}/\mathbb{Z}_{m\ell} \simeq \mathbb{Z}/\mathbb{Z}_m \oplus \mathbb{Z}/\mathbb{Z}_\ell,$$

and using that if $\mathbb{Z}/\mathbb{Z}_m$ and $\mathbb{Z}/\mathbb{Z}_\ell$ are cyclic groups only having the zero element in common, then also $\mathbb{Z}/\mathbb{Z}_m \oplus \mathbb{Z}/\mathbb{Z}_\ell$ is cyclic and it is easy to see that the factors in (7.1) can be recombined as needed. □

Sloan and Joe [89] give a convenient way for the recombination process using a table. For example take

$$G \simeq \mathbb{Z}/\mathbb{Z}_{2^2} \oplus \mathbb{Z}/\mathbb{Z}_{2^2} \oplus \mathbb{Z}/\mathbb{Z}_2 \oplus \mathbb{Z}/\mathbb{Z}_3 \oplus \mathbb{Z}/\mathbb{Z}_{3^3} \oplus \mathbb{Z}/\mathbb{Z}_5.$$

Then the canonical form can easily be obtained using the following table.

primes	groups in nonincreasing order		
2	$\mathbb{Z}/\mathbb{Z}_{2^2}$	$\mathbb{Z}/\mathbb{Z}_{2^2}$	$\mathbb{Z}/\mathbb{Z}_2$
3	$\mathbb{Z}/\mathbb{Z}_{3^3}$	$\mathbb{Z}/\mathbb{Z}_3$	
5	$\mathbb{Z}/\mathbb{Z}_5$		
	$\mathbb{Z}/\mathbb{Z}_{2^2 3^3 5}$	$\mathbb{Z}/\mathbb{Z}_{2^2 3}$	$\mathbb{Z}/\mathbb{Z}_2$

Now it follows that

$$G \simeq \mathbb{Z}/\mathbb{Z}_{2^2 3^3 5} \oplus \mathbb{Z}/\mathbb{Z}_{2^2 3} \oplus \mathbb{Z}/\mathbb{Z}_2.$$

Setting $n_1 = 2^2 3^3 5$, $n_2 = 2^2 3$ and $n_3 = 2$, we have that $n_3 \mid n_2 \mid n_1$ and the rank r is 3.

The fundamental theorem for finite abelian groups already made an appearance in Chapter 4 (in Theorem 4.5 and Corollary 4.6) in a different form where it was used to pull apart the multiplicative structure of a rank-1 lattice. Here the fundamental theorem is used to pull apart the additive structure of a general rank lattice.

7.2 Copy rules

To escape the burden to search for r generating vectors for a rank r rule we use an empirical result from Sloan and Walsh [94] that among all such rules, copy rules (to be defined in a moment) apparently are amongst the best (in a limited study), see, e.g., [89] for a discussion which also includes theoretical results. Loosely speaking, a copy rule based on a (small) rank-1 lattice rule, copies scaled down versions of the point set of the rank-1 lattice rule for the first r dimensions. A maximal rank copy rule is obtained when r equals s, having full rank. Copy rules with $1 < r < s$ are called intermediate rank copy rules.

Definition 7.3. The ν^r*-copy rule* of an n-point rank-1 lattice rule with generating vector $\boldsymbol{z}$ and point set $P_n = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}\}$, where $\boldsymbol{x}_k = \{k\boldsymbol{z}/n\}$, is given by

$$Q(f) = \frac{1}{\nu^r n} \sum_{\boldsymbol{v} \in \mathbb{Z}_\nu^r} \sum_{k=0}^{n-1} f\left(\left\{ \frac{\boldsymbol{x}_k}{\nu} + \frac{(v_1, \ldots, v_r, 0, \ldots, 0)^\top}{\nu} \right\} \right), \tag{7.2}$$

with $\gcd(n, \nu) = 1$, $1 < r \le s$ and $\nu \ge 2$.

Note that (7.2) can be written as

$$Q(f) = \frac{1}{\nu^r n} \sum_{\boldsymbol{v} \in \mathbb{Z}_\nu^r} \sum_{k=0}^{n-1} f\left(\left\{ \frac{k\boldsymbol{z}}{n} + \frac{(v_1, \ldots, v_r, 0, \ldots, 0)^\top}{\nu} \right\} \right) \tag{7.3}$$

since

$$\left\{ \left\{ \frac{x}{\nu} + \frac{v}{\nu} \right\} : v \in \mathbb{Z}_\nu \right\} = \left\{ \left\{ x + \frac{v}{\nu} \right\} : v \in \mathbb{Z}_\nu \right\}.$$

Such a ν^r-copy rule based on an n-point lattice rule has in total $\nu^r n$ points. It is useful to compare (7.2) with the product left-rectangle rule (1.15) which is in fact a maximal rank copy rule of the trivial lattice with only one point.

7.2.1 The worst-case error for copy rules

It can be shown that the worst-case error for a ν^r-copy rule can be calculated as the error for only the n-point rank-1 lattice rule where the error representer for the first r dimensions takes a slightly different form. In Joe and Sloan [55] this is proven for the classical lattice criterion P_α (see Definition 2.3). Here we give a minor generalization based on the worst-case error in a shift-invariant tensor-product space with product weights since that is the base case for the fast construction. For simplicity we assume the integral of the kernel is 1.

Theorem 7.4. *Let $\mathcal{H}(K)$ be a shift-invariant tensor-product reproducing kernel Hilbert space equipped with product weights and a kernel given by*

$$K(\boldsymbol{x}) = \prod_{j=1}^{s} \left(1 + \gamma_j\, \omega(x_j)\right), \qquad \text{and assuming } \int_0^1 \omega(x)\,\mathrm{d}x = 0.$$

Further let the variable part ω have the Fourier expansion

$$\omega(x) = \sum_{h \in \mathbb{Z}\setminus\{0\}} \hat{k}_h\, \exp(2\pi \mathrm{i}\, hx), \tag{7.4}$$

Using a ν^r-copy rule Q as given by (7.2), the squared worst-case error can be calculated as

$$e^2(Q,K) = -1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{r} \left(1 + \gamma_j\, \overline{\omega}_\nu(x_j^{(k)})\right) \prod_{j=r+1}^{s} \left(1 + \gamma_j\, \omega(x_j^{(k)})\right),$$

where $\overline{\omega}_\nu$ is given by a ν-coarser Fourier representation of ω

$$\overline{\omega}_\nu(x) = \sum_{h \in \nu\mathbb{Z}\setminus\{0\}} \hat{k}_h\, \exp(2\pi \mathrm{i}\, hx) = \sum_{h \in \mathbb{Z}\setminus\{0\}} \hat{k}_{h\nu}\, \exp(2\pi \mathrm{i}\, h\nu x). \tag{7.5}$$

Proof. Since the kernel K is shift-invariant it follows that the variable kernel part ω has a Fourier representation (7.4). Now using the expression for the worst-case error with product weights, see (2.23), for the ν^r-copy rule Q given by (7.3) and assuming $\boldsymbol{z}$ is the generating vector of the base rank-1 lattice rule, this then gives

$$\begin{aligned}
&e^2(Q,K) \\
&\quad = -1 + \frac{1}{\nu^r n} \sum_{\boldsymbol{v} \in \mathbb{Z}_\nu^r} \sum_{k=0}^{n-1} \prod_{j=1}^{r} \left(1 + \gamma_j\, \omega\left(\frac{kz_j}{n} + \frac{v_j}{\nu}\right)\right) \prod_{j=r+1}^{s} \left(1 + \gamma_j\, \omega\left(\frac{kz_j}{n}\right)\right) \\
&\quad = -1 + \frac{1}{n} \sum_{k=0}^{n-1} \prod_{j=1}^{r} \left(1 + \frac{\gamma_j}{\nu} \sum_{v \in \mathbb{Z}_\nu} \omega\left(\frac{kz_j}{n} + \frac{v_j}{\nu}\right)\right) \prod_{j=r+1}^{s} \left(1 + \gamma_j\, \omega\left(\frac{kz_j}{n}\right)\right).
\end{aligned}$$

Using (7.4) we have

$$\begin{aligned}
\frac{1}{\nu} \sum_{v_j \in \mathbb{Z}_\nu} \omega\left(\frac{kz_j}{n} + \frac{v_j}{\nu}\right) &= \sum_{h \in \mathbb{Z}\setminus\{0\}} \hat{k}_h\, \exp\left(2\pi \mathrm{i}\, h \frac{kz_j}{n}\right) \left(\frac{1}{\nu} \sum_{v_j \in \mathbb{Z}_\nu} \exp\left(2\pi \mathrm{i}\, h \frac{v_j}{\nu}\right)\right) \\
&= \sum_{\substack{h \in \mathbb{Z}\setminus\{0\} \\ h \equiv 0 \,(\mathrm{mod}\, \nu)}} \hat{k}_h\, \exp\left(2\pi \mathrm{i}\, h \frac{kz_j}{n}\right)
\end{aligned}$$

$$= \sum_{h \in \nu\mathbb{Z}\setminus\{0\}} \hat{k}_h \exp\left(2\pi \mathrm{i}\, h \frac{k z_j}{n}\right)$$
$$= \overline{\omega}_\nu\left(\frac{k z_j}{n}\right)$$

from which the result follows. □

7.2.2 Fast construction of copy rules

From the form of the worst-case error formula in Theorem 7.4 it trivially follows that fast component-by-component construction can be achieved. In Kuo and Joe [63] it was already proven that good intermediate rank copy rules can be constructed component-by-component and thus such a construction is justified.

Theorem 7.5. *Component-by-component construction of a ν^r-copy rule based on a rank-1 lattice rule with n points (prime or composite) in an s-dimensional shift-invariant reproducing kernel Hilbert space with product weights can be done in time $O(sn\log(n))$ and memory $O(n)$.*

Proof. See Theorem 3.7 and Theorem 4.12 when n is not prime (or Theorem 5.3, with $m_1 = m_2 = m$, for the special case of n a prime power p^m). □

In the same way also a good embedded lattice sequence using an intermediate rank copy rule could be constructed in a fast way using Theorem 5.3.

Theorem 7.6. *Construction of an embedded ν^r-copy rule with up to $n\nu^r$ points, with $n = p^m$ and p prime, using Algorithm 5 can be done in time $O(sn\log(n))$. The total construction cost including the precalculations is $O(sn(\log(n))^2)$.*

Proof. See Theorem 5.3. □

For the spaces that were discussed in the previous chapters, being the Korobov space, anchored Sobolev space and unanchored Sobolev space, the ω function has always taken the form of a scaled Bernoulli polynomial of degree α for which the Fourier representation is given in (2.4). This means that $\hat{k}_h$ in (7.4) is proportional to $|h|^{-\alpha}$ and thus $\hat{k}_{h\nu}$ in (7.5) is proportional to $\nu^{-\alpha}|h|^{-\alpha}$. As such, the formula for the squared worst-case error in these spaces could be written as

$$e^2(Q, K^{\mathrm{shinv}}) = -1 + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{r}\left(1 + \frac{\gamma_j}{\nu^\alpha}\,\omega(\nu x_j^{(k)})\right)\prod_{j=r+1}^{s}\left(1 + \gamma_j\,\omega(x_j^{(k)})\right).$$

This recovers the form for P_α in [55] and [89].

Obviously only very minor changes have to be made to the Matlab codes presented in Chapter 3 or Chapter 5, i.e., add extra input parameters to specify

$\overline{\omega}$ and r, and then do the calculation in two phases. When the function space is one of the choices above, then no changes to the code are necessary (this is also mentioned in [63]): pre- and post-processing is all that is needed (see the example at the end of this section). Since the changes are this trivial, there is no example code provided. The example codes in Chapters 3 and 5 could be used to derive fast construction of copy rules with n prime or a prime power.

Since the number of points grows exponentially in ν with r, the most practical choice for ν is 2. From the truncation dimension point of view, r should be chosen so that it copies the P_n rule in the first few dimensions which capture almost all of the variance. If this is not naturally so for the integrand under consideration, it might be possible to do a transformation (see the discussion on PCA in Chapter 8). A more elaborate discussion on the use of copy rules is given in [63].

We now provide a short example to construct a copy rule in a Korobov space with a total of $7993 \times 2^3 = 63944$ points using the `fastrank1pt` script from Chapter 3.

```
omega = @(x) 2*pi^2 * (x.^2 - x + 1/6);
n = 7993; s = 10; gamma = 0.9.^[1:s];
nu = 2; r = 3; ggamma = [gamma(1:r)/nu^2 gamma(r+1:end)];
[zz, e2] = fastrank1pt(n, s, omega, ggamma, ones(s, 1));
[g, c, d] = gcd(nu, n); nuinv = mod(c, n);
z = zz; z(1:r) = mod(nuinv * zz(1:r), n); z(1:r)
```

This generates the following output:

```
s=    1, z=      1, w=      1, e2=1.1586e-08, e=1.0764e-04
s=    2, z=   3040, w=    435, e2=2.0206e-07, e=4.4951e-04
s=    3, z=   1429, w=   2302, e2=2.1618e-06, e=1.4703e-03
s=    4, z=   3783, w=   1021, e2=4.7523e-05, e=6.8937e-03
s=    5, z=   2547, w=   2106, e2=5.0772e-04, e=2.2533e-02
s=    6, z=   1991, w=   1086, e2=3.3650e-03, e=5.8008e-02
s=    7, z=   2151, w=    370, e2=1.4906e-02, e=1.2209e-01
s=    8, z=   2956, w=   1671, e2=5.0792e-02, e=2.2537e-01
s=    9, z=   2273, w=   2332, e2=1.4589e-01, e=3.8196e-01
s=   10, z=    990, w=   2315, e2=3.6018e-01, e=6.0015e-01
ans =
         3997
         1520
         4711
```

The generating vector which is returned by the `fastrank1pt` script needs to be multiplied with the multiplicative inverse modulo n of ν for the first r components. The inverse can easily be obtained by using the extended Euclidean algorithm. In Matlab this can be done by using more output parameters for the `gcd` function as was done in the example.

For a normal rule with 64007 points in the same space we get a worst-case error of 6.8794e-01. So in this case the copy-rule is slightly better. A theoretical discussion on this is given in [63] where it is also remarked that this method is unlikely to work for the Sobolev spaces. Loosely speaking, we could say that the factor of $2\pi^2$ is the helping hand for the Korobov space (with $\alpha = 2$) since this is the only difference with the shift-invariant unanchored Sobolev space.

7.3 Polynomial copy rules

The usefulness of copy rules for normal lattice rules might be a point of discussion (see again [63]), but from the algebraic point of view it is interesting to define a polynomial variant of such copy rules. This might be a first step in understanding the algebraic structure needed to develop the equivalent techniques that are used in Chapter 4.

7.3.1 An equivalent definition

Definition 7.3 can also be stated in a polynomial form by carefully matching the operations in (7.2) or (7.3). We then need polynomial operations over the field $\mathbb{F}_p[x]$ or the field of the formal Laurent series $\mathbb{F}_p((x))$ (which may here include positive and negative powers), whichever is appropriate, as well as a modified version of the mapping v_m from (6.1) to get the correct number of bits.

We start by looking at the form (7.3) in the scalar case. The jth component for $j \le r$ is there obtained as

$$\begin{aligned}\left\{\frac{kz_j}{n} + \frac{v_j}{\nu}\right\} &= \left\{\left\{\frac{kz_j}{n}\right\} + \frac{v_j}{\nu}\right\} \\ &= \left\{x_j + \frac{v_j}{\nu}\right\} \qquad \text{with } x_j \in [0,1) \\ &= \left\{\frac{x_j\nu + v_j}{\nu}\right\} \\ &= \frac{(x_j\nu + v_j) \bmod \nu}{\nu}. \end{aligned} \tag{7.6}$$

The modulo ν in (7.6) must be interpreted as a "fractional modulo" like modulo 1 can be interpreted as taking the fractional part. In this way the addition operation in (7.6) is rational addition where the denominator can always be written in the form $n\nu$. In fact this can completely be multiplied through by n to get a more usual definition for the modulo, i.e.,

$$\frac{(x_j\nu + v_j) \bmod \nu}{\nu} = \frac{(x_j\nu n + v_j n) \bmod \nu n}{\nu n} \qquad \text{where now } x_j\nu n \in \mathbb{Z}.$$

Expression (7.6) can now be translated into a polynomial version like

$$\begin{aligned}\frac{k(x)\,z_j(x) \bmod f(x)}{f(x)} + \frac{v_j(x)}{\nu(x)} &= w_j(x) + \frac{v_j(x)}{\nu(x)} \\ &= \frac{(w_j(x)\,\nu(x) + v_j(x)) \bmod \nu(x)}{\nu(x)} \\ &= \sum_{\ell=1}^{\infty} c_\ell\, x^{-\ell} \qquad \in \mathbb{F}_p((x^{-1})).\end{aligned}$$

Translating this formal Laurent series in $\mathbb{F}_p((x^{-1}))$ should now be done up to a precision of p^{-m} where $m = \deg(f) + \deg(\nu)$. In a similar fashion as in Definition 6.1 we arrive at the following definition for polynomial copy rules (for the simplified case with $f(x)$ an irreducible polynomial).

Definition 7.7. Given an n-point rank-1 polynomial lattice rule constructed over $\mathbb{F}_p[x]/f$, i.e., in base p with irreducible modulus $f(x)$, $\deg(f) = m_1$ and $n = p^{m_1}$, then the $\nu(x)^r$*-copy rule*, with $\deg(\nu) = m_2$, is defined by

$$Q(f) = \frac{1}{p^{m_1}(p^{m_2})^r} \sum_{\boldsymbol{v}(x) \in (\mathbb{F}_p[x]/\nu)^r} \sum_{k(x) \in \mathbb{F}_p[x]/f} f\left(v_{m_1+m_2}\left(\frac{k(x)\boldsymbol{z}(x)}{f(x)} + \frac{(v_1(x), \ldots, v_r(x), 0, \ldots, 0)^\top}{\nu(x)}\right)\right),$$

with $\gcd(f(x), \nu(x)) = 1$, $1 < r \le s$ and $\deg(\nu) \ge 1$.

Note that in comparison with the scalar case there is more freedom in choosing how to copy: the degree of the copy polynomial specifies how many times we copy while the coefficients specify "how" we copy.

7.3.2 The worst-case error for polynomial copy rules

The worst-case error for such a polynomial copy rule is given by the next theorem.

Theorem 7.8. *Let $\mathcal{H}(K)$ be a digital shift-invariant reproducing kernel Hilbert space equipped with product weights and kernel*

$$K(\boldsymbol{t}) = \prod_{j=1}^{s}(1 + \gamma_j\,\omega(t_j)), \qquad \textit{and assuming } \int_0^1 \omega(t)\,\mathrm{d}t = 0.$$

Further let

$$\omega(t) = \sum_{h=1}^{\infty} \hat{k}_p(h)\ \mathrm{wal}_{p,h}(t).$$

The squared worst-case error using a $\nu(x)^r$-copy rule Q as given by Definition 7.7 can be calculated as

$$e^2(Q,K) = -1 + \frac{1}{p^{m_1}} \sum_{k=0}^{p^{m_1}-1} \prod_{j=1}^{r} (1 + \gamma_j \, \overline{\omega}_{\nu(x)}(t_j^{(k)})) \prod_{j=r+1}^{s} (1 + \gamma_j \, \omega(t_j^{(k)})),$$

with $\{\boldsymbol{t}_0, \dots \boldsymbol{t}_{p^{m_1}-1}\}$ the points from the basis polynomial lattice rule and $\overline{\omega}_{\nu(x)}$ a filtered version of ω based on the coefficients of $\nu(x)$ (described in the proof).

Proof. This proofs follows a parallel track to the proof of Theorem 7.4. For ease of notation set $m = m_1 + m_2$. For a digital shift-invariant kernel we have

$$e^2(Q,K) = -1 + \frac{1}{p^{m_1}(p^{m_2})^r} \sum_{\boldsymbol{v}(x) \in (\mathbb{F}_p[x]/\nu)^r} \sum_{k(x) \in \mathbb{F}_p[x]/f} \prod_{j=1}^{r} \left(1 + \gamma_j \, \omega\left(v_m\left(\frac{k(x)z_j(x)}{f(x)} + \frac{v_j(x)}{\nu(x)}\right)\right)\right) \prod_{j=r+1}^{s} \left(1 + \gamma_j \, \omega\left(v_m\left(\frac{k(x)z_j(x)}{f(x)}\right)\right)\right).$$

Again as in the proof of Theorem 7.4 the first sum can be brought inside and we have to look at the expression

$$\frac{1}{p^{m_2}} \sum_{v_j(x) \in \mathbb{F}_p[x]/\nu} \omega\left(v_m\left(\frac{k(x)z_j(x)}{f(x)} + \frac{v_j(x)}{\nu(x)}\right)\right) = \frac{1}{p^{m_2}} \sum_{v_j(x) \in \mathbb{F}_p[x]/\nu} \sum_{h=1}^{\infty} \hat{k}_p(h) \, \mathrm{wal}_{p,h}\left(v_m\left(\frac{k(x)z_j(x)}{f(x)} + \frac{v_j(x)}{\nu(x)}\right)\right). \quad (7.7)$$

The Walsh function can be split over the two addends since from (6.1) follows that for $a(x), b(x) \in \mathbb{F}_p((x))$ we have

$$\begin{aligned} v_m(a(x) + b(x)) &= v_m(a(x)) \oplus_p v_m(b(x)) \\ &= a \oplus_p b, \end{aligned}$$

and from Definition 6.2 follows that

$$\mathrm{wal}_{p,h}(a \oplus_p b) = \mathrm{wal}_{p,h}(a) \, \mathrm{wal}_{p,h}(b).$$

Therefore

$$\mathrm{wal}_{p,h}\left(v_m\left(\frac{k(x)z_j(x)}{f(x)} + \frac{v_j(x)}{\nu(x)}\right)\right) = \mathrm{wal}_{p,h}\left(v_m\left(\frac{k(x)z_j(x)}{f(x)}\right)\right) \mathrm{wal}_{p,h}\left(v_m\left(\frac{v_j(x)}{\nu(x)}\right)\right).$$

For a fixed $h = (h_t \dots h_0)_p \neq 0$ (and for some t) now consider

$$\frac{1}{p^{m_2}} \sum_{v(x)\in\mathbb{F}_p[x]/\nu} \mathrm{wal}_{p,h}\left(v_m\left(\frac{v(x)}{\nu(x)}\right)\right) = \frac{1}{p^{m_2}} \sum_{v(x)\in\mathbb{F}_p[x]/\nu} \prod_{i=1}^{m} \exp(2\pi\mathrm{i}\, h_{i-1}\tilde{v}_i/p), \tag{7.8}$$

with non-existent h_i set to zero and the $\tilde{v}_i$ given by

$$\tilde{v} = v_m\left(\frac{v(x)}{\nu(x)}\right) = (0.\tilde{v}_1\tilde{v}_2\dots\tilde{v}_{m_2}\dots\tilde{v}_m)_p, \qquad \text{and } \tilde{v}_i = 0 \text{ for } i > m. \tag{7.9}$$

The $\tilde{v}_i$ are the coefficients of the Laurent expansion of $v(x)/\nu(x)$ and, similar to (6.9), these can be written recursively as

$$\tilde{v}_i \equiv \nu_{m_2}^{-1}\left(v_{m_2-i} - \sum_{j=1}^{m_2} \nu_{m_2-j}\,\tilde{v}_{i-j}\right) \pmod p, \qquad i = 1, 2, \dots, \tag{7.10}$$

where $v_{m_2-i} = 0$ for $i > m_2$ and the values for $i > m$ are formally set to zero in (7.9) and (7.8). Note that for $\forall v(x) \in \mathbb{F}_p[x]/\nu$ the $\tilde{v}_i$ for $1 \le i \le m_2$ just run through all possible elements in $\mathbb{Z}_p$ (since $\nu_{m_2} \in \mathbb{Z}_p \setminus \{0\}$ and thus $\nu_{m_2} \in \mathbb{Z}_p^\times$) and we find that (7.8) can be written as

$$\frac{1}{p}\sum_{\tilde{v}_1\in\mathbb{Z}_p}\left(\exp(2\pi\mathrm{i}\, h_0\tilde{v}_1/p)\cdots\left(\frac{1}{p}\sum_{\tilde{v}_{m_2}\in\mathbb{Z}_p}\exp(2\pi\mathrm{i}\, h_{m_2-1}\tilde{v}_{m_2}/p)\right.\right.$$
$$\left.\left.\times \prod_{\substack{i=m_2+1\\ \text{with } \tilde{v}_i \text{ to match (7.9)}}}^{m} \exp(2\pi\mathrm{i}\, h_{i-1}\tilde{v}_i/p)\right)\cdots\right). \tag{7.11}$$

Since from (7.10) follows that $\tilde{v}_i$ can be written as a linear combination of $\{\tilde{v}_1, \dots, \tilde{v}_{m_2}\}$,

$$\tilde{v}_i \equiv \sum_{j=1}^{m_2} a_{i,j}\tilde{v}_j \pmod p, \tag{7.12}$$

the product for $i = m_2 + 1$ to m can be written as

$$\prod_{\substack{i=m_2+1\\ \text{with } \tilde{v}_i \text{ to match (7.9)}}}^{m} \exp(2\pi\mathrm{i}\, h_{i-1}\tilde{v}_i/p) = \prod_{i=m_2+1}^{m}\prod_{j=1}^{m_2}\exp(2\pi\mathrm{i}\, h_{i-1}a_{i,j}\tilde{v}_j/p).$$

Substituting this in (7.11) gives

$$\prod_{j=1}^{m_2} \frac{1}{p} \sum_{\tilde{v}\in\mathbb{Z}_p} \exp\left(2\pi\mathrm{i}\ \left(h_{j-1} + \sum_{i=m_2+1}^{m} h_{i-1}a_{i,j}\right)\tilde{v}/p\right)$$
$$= \begin{cases} 1 & \text{if } h_{j-1} + \sum_{i=m_2+1}^{m} h_{i-1}a_{i,j} \equiv 0 \pmod{p} \text{ for all } 1 \le j \le m_2, \\ 0 & \text{otherwise.} \end{cases}$$

Denote by $\mathcal{A}_{\nu(x)}$ the values of h which give a non-zero result, i.e.,

$$\mathcal{A}_{\nu(x)} := \left\{h \in \mathbb{Z}_* : h_{j-1} + \sum_{i=m_2+1}^{m} h_{i-1}a_{i,j} \equiv 0 \pmod{p} \text{ for all } 1 \le j \le m_2\right\}.$$

This defines a version of the ω function which uses only a filtered portion of the Walsh coefficients based on the coefficients of $\nu(x)$:

$$\overline{\omega}_{\nu(x)}(t) := \sum_{h\in\mathcal{A}_{\nu(x)}} \hat{k}_p(h)\ \mathrm{wal}_{p,h}(t).$$

It follows that the worst-case error can be calculated as the worst-case error of the uncopied rule using $\overline{\omega}_{\nu(x)}$ instead of ω in the first r dimensions (and mapping with v_m instead of v_{m_1}).

□

When taking $\nu(x)$ to be a monomial the previous theorem simplifies in an obvious way and the exact equivalent of Theorem 7.4 for the scalar case is obtained.

Corollary 7.9. *When $\nu(x) = x^{m_2}$ the filtered version of ω is given by*

$$\overline{\omega}_{\nu(x)}(t) := \sum_{h\in p^{m_2}\mathbb{Z}_*} \hat{k}_p(h)\ \mathrm{wal}_{p,h}(t).$$

Proof. From (7.10) and (7.12) follows that all $a_{i,j} = 0$ for $i > m_2$. □

7.3.3 Fast construction of polynomial copy rules

Since the worst-case error formulas have the desired form, fast construction follows easily.

Theorem 7.10. *Component-by-component construction of a $\nu(x)^r$-copy rule based on a rank-1 polynomial lattice rule, modulo an irreducible polynomial f over $\mathbb{Z}_p$, with $n = p^{\deg(f)}$ points in an s-dimensional digital shift-invariant reproducing kernel Hilbert space with product weights can be done in time $O(sn\log(n))$ and memory $O(n)$.*

Proof. See Theorem 6.6. □

Chapter notes

Since the calculations only operate on vectors of length $n-1$ (in the scalar case or length $p^{m_1}-1$ in the polynomial case), copy rules are an easy way to obtain a rather large number of lattice points, while still having a very manageable calculation cost. E.g., taking $n = 4001$, $\nu = 2$ and $r = 10$ the total number of points is already over 4 million while the cost is only that for the construction of a non-copy rule with 4001 points.

None of the material in this chapter is published since the scalar copy rules are trivial and the part about the polynomial copy rules is still incomplete. The latter is an exercise to get a better understanding of the possibilities of polynomial lattice rules and the underlying algebraic structure.

Chapter 8

Applications

When one constructs lattice rules, one is of course also interested in actual numerical results on real world problems (or almost real world problems). In this chapter, three such examples are shown. Two of them come from financial engineering (rather popular for quasi-Monte Carlo since the results from Paskov and Traub [86]) while the third one is a likelihood estimation in a statistical application. Except for one result, all the numerical tests have been done with the embedded lattice rule given in Table 5.2. This lattice rule is just an order-2 rule, and it is therefore surprising that the numerical results do so well.

8.1 Principal component analysis

To make the application of a low-discrepancy point set much more effective it helps to reorder the variables in a way that maximizes the variance of the first few variables (i.e., making the truncation dimension from Definition 2.28 as low as possible). This is because the first few dimensions of a low-discrepancy point set have in general a better discrepancy than the later ones. Especially for the finance problems two techniques are well known: Brownian bridge and principal component analysis (PCA) [35]. In this section we explain principal component analysis as it is applied in the construction of a Brownian motion.

A standard Brownian motion in one dimension is defined as follows, see, e.g., [35]. (The "standard" means it starts at 0.)

Definition 8.1. A standard one-dimensional *Brownian motion* on a time interval $[0, T]$ is a stochastic process $\{W(t) : 0 \le t \le T\}$ with

(i) $W(0) = 0$;

(ii) for any $0 \le t_1 < t_2 < t_3 < \cdots \le T$ the increments $\{W(t_2) - W(t_1), W(t_3) - W(t_2), \ldots\}$ are independent; and

(iii) the increments are normally distributed with variance given by their time difference, i.e., $W(t_2) - W(t_1) \sim \mathcal{N}(0, t_2 - t_1)$.

In this section we use n to denote the number of steps in the discretized version of the Brownian path. In the next section the n will be changed in an s, since there it will act as the dimension of the integrals.

The PCA technique works by calculating an eigenvalue decomposition of the covariance matrix $\boldsymbol{\Sigma}$ associated with the Brownian motion $W(t)$ and using this decomposition to generate a "square-root" $\boldsymbol{A}$ of $\boldsymbol{\Sigma} = \boldsymbol{A}\,\boldsymbol{A}^\top$. Given a random n-vector $\boldsymbol{z}$ with components $z_j \sim \mathcal{N}(0,1)$ a random sample path from the Brownian motion can be generated by a simple matrix-vector product

$$\boldsymbol{w} = \boldsymbol{A}\,\boldsymbol{z}. \tag{8.1}$$

If the time points are equidistant, $t_i = i\Delta t$, $i = 1, \ldots, n$ and $\Delta t = T/n$, then the eigendecomposition of $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$, where the covariance matrix

$$\big[\Sigma\big]_{i,j} = \min(t_i, t_j), \qquad \text{for } i, j = 1, \ldots, n \tag{8.2}$$

is known in closed form, see [2] (or also [35]).

Lemma 8.2. *The eigenvalues and associated eigenvectors of* (8.2) *are given by*

$$\lambda_i = \frac{\Delta t}{4} \frac{1}{\sin^2\left(\frac{2i-1}{2n+1}\frac{\pi}{2}\right)}, \quad v_i(j) = \frac{2}{\sqrt{2n+1}} \sin\left(\frac{2i-1}{2n+1} j\pi\right), \quad \textit{for } i, j = 1, \ldots, n, \tag{8.3}$$

so that

$$\boldsymbol{\Sigma} = \boldsymbol{V}\,\Lambda\,\boldsymbol{V}^{-1} = \boldsymbol{V}\,\Lambda\,\boldsymbol{V}^\top, \qquad \textit{with } \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n).$$

With this formulation the eigenvalues are automatically sorted in decreasing order, such that

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n.$$

The PCA method chooses this transformation matrix $\boldsymbol{A}$ in the following way

$$\boldsymbol{A}_{\text{PCA}} = \left[\sqrt{\lambda_1}\boldsymbol{v}_1, \sqrt{\lambda_2}\boldsymbol{v}_2, \ldots, \sqrt{\lambda_n}\boldsymbol{v}_n\right] = \boldsymbol{V}\sqrt{\Lambda},$$

with the eigenvalues ordered in decreasing order.

8.2 Pricing an Asian option

We now turn to using s as the number of time steps for the Brownian motion, since it will correspond to the dimension of the integral. The embedded lattice rule from Table 5.2 is used here for pricing an Asian call option. This problem is extensively documented, both in the financial literature and in the quasi-Monte Carlo literature [56, 68, 8]. For an Asian option (also called average option) the payoff is determined by the average value of the underlier (e.g., a stock price) during its lifetime. We will consider a (European style) Asian option with an arithmetic average of s equally spaced stock prices S_j at times $t_j = jT/s$, with expiry time T. The payoff for such a call option can be written as

$$C(T) = \max\left(\frac{1}{s}\sum_{j=1}^{s} S_j - K, 0\right) = \left(\frac{1}{s}\sum_{j=1}^{s} S_j - K\right)^+,$$

where K is the strike price at the expiry time $T = t_s$. We assume the risk-neutral measure of the Black–Scholes model for the path of the stock prices so that the dynamics are defined by

$$\frac{\mathrm{d}S(t)}{S(t)} = r\,\mathrm{d}t + \sigma\,\mathrm{d}W(t),$$

with the risk-free rate r and the volatility σ fixed, and $W(t)$ a one-dimensional Brownian motion. Then

$$S(t) = S(0)\exp\left((r - \sigma^2/2)\,t + \sigma W(t)\right),$$

and thus for a given discretized realization

$$S_j = S_0\exp\left((r - \sigma^2/2)\,t_j + \sigma w_j\right),$$

where S_0 is the initial stock price, r is the risk-free interest rate, σ the volatility of the stock price, and w_j are samples of a Brownian motion at times t_j, i.e., w_j is normally distributed with variance t_j, or $w_j \sim \mathcal{N}(0, t_j)$. This sampled Brownian motion has mean 0 and covariance matrix

$$\boldsymbol{\Sigma} = \Big[\min(t_i, t_j)\Big]_{i,j=1,\ldots,s}.$$

The value of the Asian option at time 0 can then be written as the actualized expected payoff at time T,

$$\begin{aligned} C(0) &= E\left[\exp(-rT)\,C(T)\right] \\ &= \exp(-rT)\int_{\mathbb{R}^s}\left(\frac{1}{s}\sum_{j=1}^{s} S_0\exp\left((r - \sigma^2/2)\,t_j + \sigma w_j\right) - K\right)^+ \mathcal{N}(\boldsymbol{w}; \mathbf{0}, \boldsymbol{\Sigma})\,\mathrm{d}\boldsymbol{w}. \end{aligned}$$

This integral can be transformed to $[0,1)^s$ by using the inverse of the cumulative normal distribution function Φ^{-1}. The correlation in the sample points will be dealt with by using a factorization of the covariance matrix

$$\boldsymbol{\Sigma} = \boldsymbol{A}^\top \boldsymbol{A}.$$

The lattice rule approximation to $C(0)$ is now given by

$$\widetilde{C}(0) = \frac{\exp(-rT)}{n} \sum_{k=0}^{n-1} \left(\frac{1}{s} \sum_{j=1}^{s} S_0 \exp\left((r - \sigma^2/2)\, t_j + \sigma w_{k,j} \right) - K \right)^+,$$

where $\boldsymbol{w}_k = \boldsymbol{A}\, \Phi^{-1}(\boldsymbol{x}_k)$ with lattice point $\boldsymbol{x}_k$ for $k = 0, 1, \ldots, n-1$.

For our particular example we choose the following parameters:

$$S_0 = 100, \qquad r = 0.1, \qquad \sigma = 0.2, \qquad T = 1, \qquad K = 100, \qquad s = 100.$$

We use the principal component analysis method to factorize the matrix $\boldsymbol{\Sigma}$, that is, $\boldsymbol{A} = [\sqrt{\lambda_1}\boldsymbol{v}_1, \ldots, \sqrt{\lambda_s}\boldsymbol{v}_s]$ where $\lambda_1 \geq \cdots \geq \lambda_s$ are the eigenvalues of $\boldsymbol{\Sigma}$ and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_s$ are the corresponding unit-length eigenvectors.

In Table 8.1 the results obtained with the order-2 rule from Table 5.2 and by the Monte Carlo method, both using 10 random shifts, are presented. In Figure 8.1 the graphs of the standard errors obtained by adding one point at a time are plotted. (The values of n in both Table 8.1 and Figure 8.1 correspond to the number of points used from the sequence. Due to the use of 10 random shifts, the actual number of function evaluations in each case is $10n$.) The results for the order-2 rule clearly show a rate of convergence close to $O(n^{-1})$ while the Monte Carlo error decays like $O(n^{-1/2})$. Furthermore, the graph assures us that it is quite all right to use the embedded lattice rules as a sequence, since the standard error goes down nicely (with some minor zigzag behavior, as expected) as the number of points is increased one at a time.

From Wang and Fang [108] it is known that the truncation dimension, Definition 2.28, for the Asian option is $0.8s$ for standard path construction, 8 for Brownian bridge construction and only 2 for PCA construction. It follows that PCA will have a big advantage over the other path constructions. In Figure 8.2 a comparison of the different path constructions and different point sets can be seen for an Asian option where $s = 64$. The horizontal axis shows the number of points used, however, it must be noted that changing this to the effective CPU time does not make any difference for the dimension shown. That is to say, the generation of points using a standard random number generator, a lattice sequence or the Sobol' point set is approximately comparable and the incurred cost to do Brownian bridge or PCA surely pays off in the standard error.

n	order-2 lattice rule		Monte Carlo	
	integral	standard error	integral	standard error
1024	7.11310	5.07e-03	7.13816	8.97e-02
2048	7.10685	1.77e-03	7.10988	5.98e-02
4096	7.10252	1.56e-03	7.12438	4.84e-02
8192	7.10261	6.47e-04	7.11989	2.54e-02
16384	7.10304	4.17e-04	7.12116	2.10e-02
32768	7.10324	2.34e-04	7.12404	1.49e-02
65536	7.10280	1.18e-04	7.11410	1.14e-02
131072	7.10293	4.70e-05	7.11427	9.58e-03
262144	7.10287	1.91e-05	7.10980	5.05e-03
524288	7.10285	1.69e-05	7.10636	5.00e-03
1048576	7.10285	8.68e-06	7.10709	1.69e-03

Table 8.1: Standard errors for the Asian option for Monte Carlo and order-2 lattice rule for $s = 100$ and 10 random shifts.

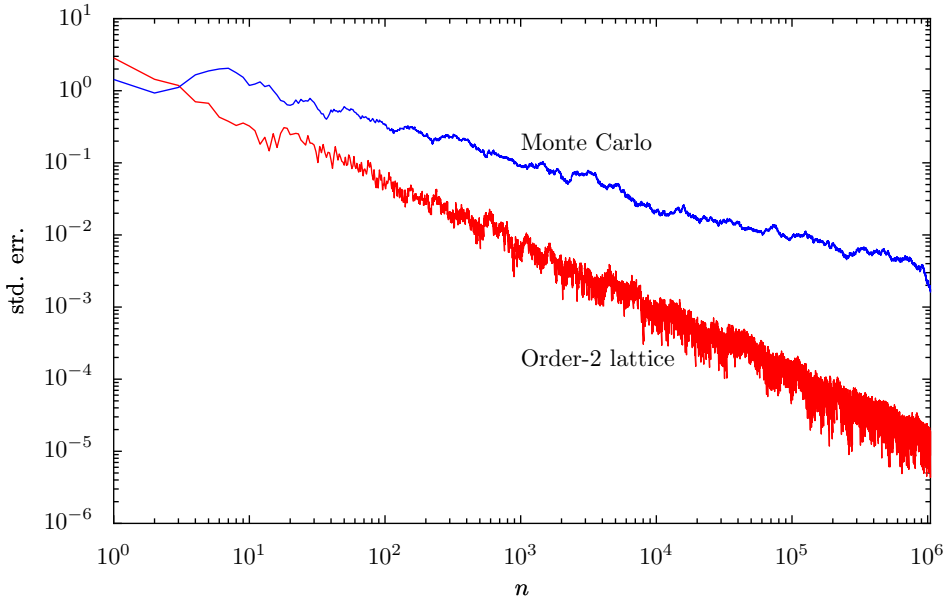


Figure 8.1: Standard errors for the Asian option for Monte Carlo and order-2 lattice rule for $s = 100$ and 10 random shifts.

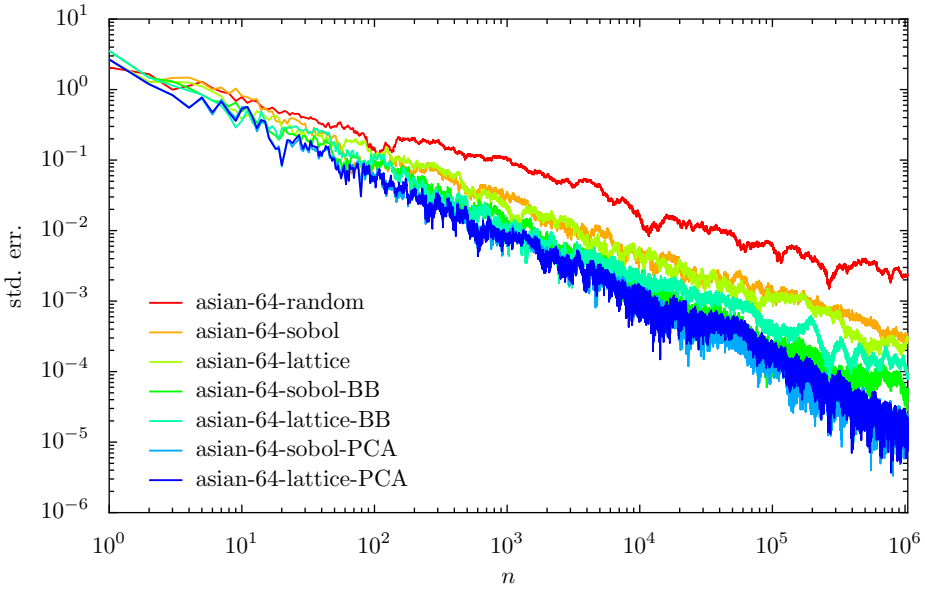


Figure 8.2: Standard errors for Asian option with different path constructions and different point sets for $s = 64$ and 10 random shifts.

8.3 A lookback option

The lookback option sits in the same framework as the Asian option. Only the payoff function is changed to

$$C(T) = \max(\max(S(t_1), \ldots, S(t_s)) - K, 0). \tag{8.4}$$

This is a much harder problem, since this integrand function is very discontinuous. An other example would be a payoff which is the maximum of the end prices of a basket of stocks, a so called outperformance option. In that case the $S(t_i)$ in (8.4) would be changed into $S_i(T)$ with S_i the prices of stock i. However in [8] it is shown that lattice rules perform "dramatically" better than, e.g., Sobol' points on such a problem, and certainly way better than just random points.

There are two reasons for this impressive result:

(i) There is a Black-Scholes kind of analytic solution in terms of the multivariate normal distribution and using work by Genz [34] this integral can be approximated nicely by lattice rules.

(ii) For this function periodization works nicely.

Periodization of integrand functions is a traditional technique that has not been discussed in this text up till now, since it should only be considered for rather low-dimensional integrals. We consider an easy periodizing transformation: using a nonlinear transformation $\varphi(t)$ which is a smooth increasing function from $[0,1)$ to $[0,1)$ and has all derivatives up to α equal at the sides, $\varphi^{(k)}(0) = \varphi^{(k)}(1)$, $1 \le k \le \alpha$. This function is then applied componentwise to the integrand function f. The best known periodizing transforms are the Sidi transforms [88] for which the first three, with increasing smoothness, look like

$$\begin{aligned}
\varphi_1(t) &= \tfrac{1}{2}(1 - \cos(\pi t)),\\
\varphi_2(t) &= \tfrac{1}{2\pi}(2\pi t - \sin(2\pi t)),\\
\varphi_3(t) &= \tfrac{1}{16}(8 - 9\cos(\pi t) + \cos(3\pi t)).
\end{aligned}$$

There is also a similar set of polynomial transforms by Laurie [66]. The integrand is then transformed as

$$\begin{aligned}
&\int_{[0,1)^s} f(x_1, x_2, \ldots, x_s)\,\mathrm{d}\boldsymbol{x}\\
&\qquad = \int_{[0,1)^s} f(\varphi(x_1), \varphi(x_2), \ldots, \varphi(x_s))\,\varphi'(x_1)\varphi'(x_2)\cdots\varphi'(x_s)\,\mathrm{d}\boldsymbol{x}.
\end{aligned}$$

The periodization has also repercussions on the Koksma–Hlawka error bound. From the analysis by Hickernell [45] it follows that the discrepancy for the smooth periodized function space is better, but the variation in that space is worse. So periodizing gives a transformed function space with kernel $K_{\mathrm{per},\alpha}$ and one obtains

$$|(I - Q)(f(\boldsymbol{\varphi}(\boldsymbol{x}))\,\boldsymbol{\varphi}'(\boldsymbol{x}))| \le D(K_{\mathrm{per},\alpha})\;\|f(\boldsymbol{\varphi}(\boldsymbol{x}))\,\boldsymbol{\varphi}'(\boldsymbol{x})\|_{\mathrm{per},\alpha}, \tag{8.5}$$

in comparison with

$$|(I - Q)(f(\boldsymbol{x}))| \le D(K_\alpha)\;\|f(\boldsymbol{x})\|_\alpha,$$

where $\|f(\boldsymbol{\varphi}(\boldsymbol{x}))\,\boldsymbol{\varphi}'(\boldsymbol{x})\|_{\mathrm{per},\alpha}$ will be much worse than $\|f(\boldsymbol{x})\|_\alpha$.

We now return to the pricing problem. The value of the lookback option can be expressed in its analytic form as

$$V(S_0, K, r, \sigma, T) = \exp(-rT)\,S_0\left(\sum_{j=1}^{s} \exp(rt_j)\,H_j\,I_{s-j} - (1 - L_s)\,K\right),$$

where H_j, I_{s-j} and L_s are all multivariate normal probabilities

$$\frac{1}{\sqrt{|\boldsymbol{\Sigma}|(2\pi)^s}} \int_{-\infty}^{a_1} \cdots \int_{-\infty}^{a_s} \exp(-\tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{x})\,\mathrm{d}\boldsymbol{x},$$

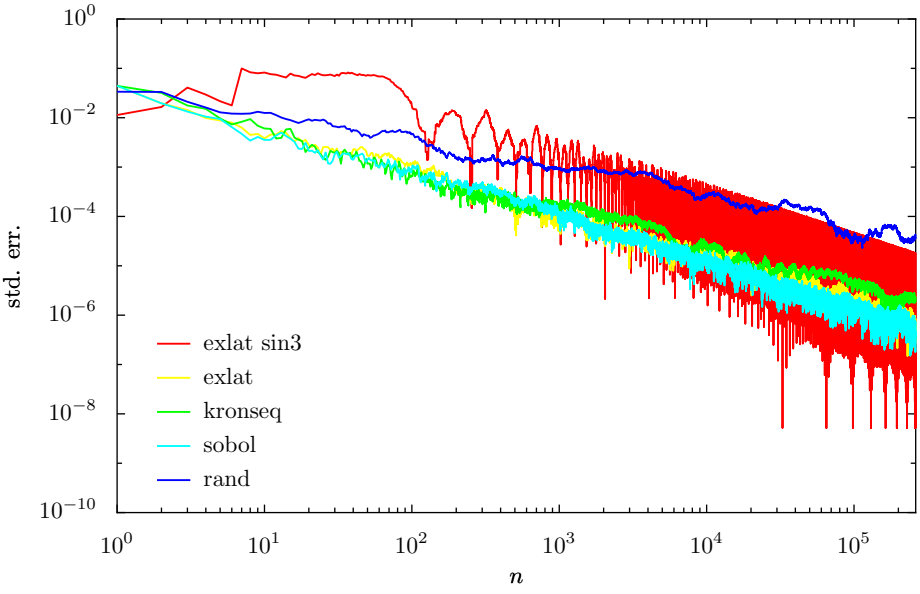


Figure 8.3: Effect of periodization on the lookback option in 5 dimensions.

with different dimensions and covariance matrices, see [8] for the detailed expressions.

We tested this problem numerically in 5 dimensions to see what actually happens in the paper [8]. The main question is: do they get an improved rate of convergence, corresponding to the smoothness of the space after periodization? In [8] fixed lattice rules are used. Here we use a sequence given in Table 5.2 (the order-2 rule). In this way we can see what happens along the way.

Figure 8.3 shows the standard error, using 10 random shifts, for random points, Sobol' points, the Kronecker sequence (with irrational vector as given in Chapter 1), the extensible lattice and the extensible lattice with periodization. It can clearly be seen that the quasi-Monte Carlo methods beat the Monte Carlo method.

However the interesting action is for the extensible lattice rule with periodization. Its convergence curve is dramatically shifted up! This is in accordance with (8.5). More importantly, whenever there is a glitch downwards it performs way better than the standard methods (from 1000 points and on). This is in accordance with the results in [8], since there is always a glitch downwards when the extensible rule is stopped at a complete lattice.

At the end of the graph the glitches do not go down consistently anymore, but stop at $\sqrt{\epsilon_{\text{mach}}}$, with ϵ_{mach} the machine epsilon for double precision arithmetic.

This is a consequence of the calculations being done in double precision, where for the periodization to work effectively for many points, a larger precision is needed. In other (lower dimensional examples) it was noted that the downward glitches then consistently go down with an improved rate of convergence.

8.4 A statistics example

One place where multivariate integrals appear is in statistics where one wants to estimate the parameters of some assumed model by means of a maximum likelihood method. In such a problem, approximating the integral is at the inside of the optimization loop, and so gets executed many times. We present here such a prototype integral, borrowed from [37],

$$T = \int_{\mathbb{R}^s} \frac{1}{1 + \exp\left(-\sum_{j=1}^{s} \theta_j\right)} \mathcal{N}_s(\boldsymbol{\theta}; -\boldsymbol{\beta}, \boldsymbol{\Sigma}) \, \mathrm{d}\boldsymbol{\theta},$$

where $\mathcal{N}_s(\boldsymbol{\theta}; -\boldsymbol{\beta}, \boldsymbol{\Sigma})$ is the multivariate normal distribution in $\boldsymbol{\theta}$ with mean $-\boldsymbol{\beta}$ and covariance matrix $\boldsymbol{\Sigma}$. (For verification purposes this integral can easily be transformed into a one-dimensional integral.) The integration region $\mathbb{R}^s$ is transformed into the unit cube using the inverse of the cumulative normal distribution and a principal component analysis (PCA) factorization of $\boldsymbol{\Sigma}$.

We will solve this integral in 5 dimensions, and choose to calculate this for $\boldsymbol{\beta} = (1, 1, 1, 1, 1)^\top$ and a covariance matrix

$$\bigl[\Sigma\bigr]_{i,j} = \begin{cases} \sigma_i \sigma_j = \sigma_i^2 & \text{if } i = j, \\ \rho\, \sigma_i \sigma_j & \text{if } i \neq j, \end{cases}$$

where $\boldsymbol{\sigma} = (2.0, 1.5, 1.0, 0.5, 0.3)^\top$ and correlation $\rho = 0.3$.

We used one lattice rule with trigonometric degree 37 and $n = 1\,044\,808$ points and the lattice rule from Table 5.2 constructed component-by-component optimized for order 2 interactions with $n = 2^{20} = 1\,048\,576$ points. Using the technique of randomly shifted lattice rules, with 10 shifts, the estimated standard error was plotted in Figure 8.4.

From the plots it can be seen that both lattice rules have almost a comparable performance and that there is indeed no reason not to use a lattice rule as a sequence. The lattice rule in the left plot has a number of points which in factorized form is $n = 2^3 \times 61 \times 2141$ while the one on the right has a number of points $n = 2^{20}$. Although the lattice rule on the left was not constructed with sequence usage in mind, it performs reasonably well, and through the usage of the standard error the user is able to get an idea of its performance. (That is, if it would go as bad as the example in Figure 5.3, then the user would notice this.) The lattice rule on the right was optimized for 2^ℓ, $\ell = 10, \ldots, 20$ points, where at intermediate powers, the points form a lattice rule.

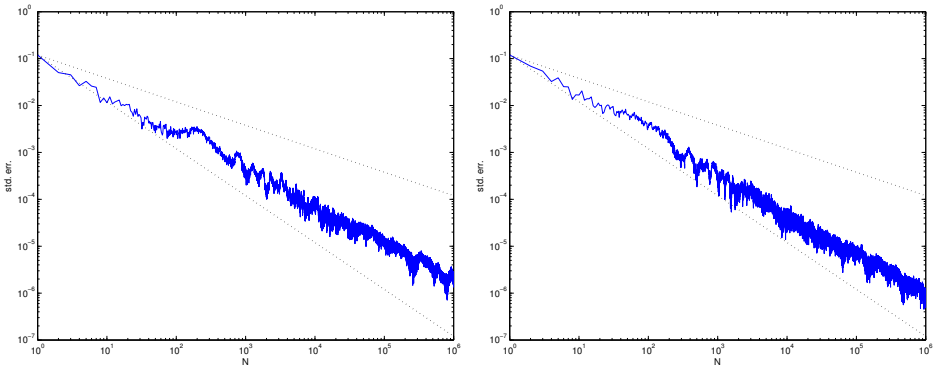


Figure 8.4: Actual convergence of the estimated standard error for T with 10 random shifts (so the actual number of points is times 10). Left: using a lattice rule with trigonometric degree 37 constructed with methods described in [14]. Right: using a lattice sequence optimized for order 2 interactions, constructed component-by-component as described in Chapter 5. Both lattices are used as a sequence, described in §5.8 and §5.5 respectively. The dotted lines are $O(1/n)$ (the optimal rate of convergence) and $O(1/\sqrt{n})$ (the Monte Carlo rate of convergence).

Chapter notes

The 100-dimensional Asian option example was published in [15]. The results for the different path constructions were presented at the SIAM Financial Mathematics and Engineering conference and at the Dagstuhl "Algorithms and Complexity for Continuous Problems" 2006 workshop. At the last workshop also the result on the periodization for the lookback option was presented. The statistics example appeared in [17].

In the finance literature the PCA method is mostly not the preferred method due to the cost of the matrix-vector product which is $O(s^2)$. Instead they use the Brownian bridge which has a cost linear in the number of dimensions, i.e., $O(s)$. However, recent developments by Mike Giles (and previously already by Klaus Scheicher) have shown that the matrix-vector product for the PCA can in fact be calculated using a fast sine transform, obtaining $O(s \log(s))$.

Being able to do PCA with a minor cost of $O(s \log(s))$ is an important result since the PCA method in general performs better than Brownian bridge. Comparing the type I discrete sine transform (DST-I) with Lemma 8.2 gives a direct way of doing the calculation by means of one application of the Danielson-Lanczos lemma to get rid of inserted zero values (using an FFT, since DST implementations are rare). (In fact the transform needed is not really a DST-I, but a more exotic type DST-VII.)

Chapter 9

Future work and open problems

At the end of a PhD thesis one must look back and forward. Some work has been done, but much more remains. In this final chapter some possible projects for the future are briefly discussed. But first the contributions presented in this text are stated.

9.1 Contributions

In this text we presented a fast construction algorithm for rank-1 lattice rules. The fast algorithm is in fact a matrix-vector formulation of the original component-by-component algorithm from Sloan and his collaborators. To get the matrix into a nice form, we used number theoretic permutations.

It is a fact that most of the papers using component-by-component construction which predate the fast algorithm could have made use of the fast algorithm. Luckily also after its conception the component-by-component construction remains a crucial part of many papers. In this way many new papers are using or referencing the fast construction, see, e.g., [25, 27, 64, 65].

The following contributions were presented in this text.

(i) Fast construction is possible for lattice rules with a prime number of points [83].

(ii) Fast construction is possible for lattice rules with a composite number of points n [85]. The performance degrades when n is a product of many distinct primes.

(iii) Fast construction of good lattice sequences, where the number of points is a prime power p^m, is possible [15]. These rules behave like (t, s)-sequences in

that the complete lattice rule consists of good shifted smaller lattice rules at intervals anchored at and spanning powers of the base p.

(iv) A good lattice sequence can be used as a low-discrepancy sequence [17].

(v) Any lattice rule can be reliably used as a sequence when using the lattice rule as the basis for a randomly shifted rule and the number of points of such a rule need not be a prime power [17].

(vi) Fast construction of polynomial lattice rules modulo an irreducible polynomial is possible [84].

(vii) Fast construction of copy rules is trivially possible.

(viii) The order-2 lattice sequence from Table 5.2 performs extremely well on some real life applications [15].

(ix) For most of the fast constructions we provided sample Matlab code in the spirit of Trefethen [106]: high-level code, not necessarily easy to read, but fitting on one page to make the code easy to study; and fast enough to enable experimentation.

9.2 Possible future research projects

9.2.1 Fast Korobov type construction

Apart from component-by-component construction, also Korobov-type construction gives good results, see §2.5.2. Although the construction complexity of the Korobov form was the same as the original component-by-component construction, in practice it was the preferred method for large n.

After the fast algorithm however, there isn't really any reason anymore to do Korobov-type construction. From a distance the Korobov-type construction has a much more simple form and the search space looks easier, therefore there is the question if it is possible to have a fast Korobov-type construction.

9.2.2 Optimal weights

A large part in Chapter 2 was about weighted function spaces and tractability. However, from the practical point of view this does not help to find optimal weights given an integrand function. In most cases it is unknown in what function space an integrand is.

The holy grail for the weighted function spaces would be to find a cheap algorithm which can estimate optimal weights given a function and then, using the fast construction algorithm, construct a tailored lattice rule for the given problem.

However, we do not really understand the interactions between the iterative component-by-component algorithm and the weights. Clearly, the first dimensions have the most freedom and so the projected discrepancies over the first few dimensions are clearly better. However, when using order dependent weights of, say,

order 2, one wants all order 2 projections to be acceptable. The numerical results with the order 2 rule from Table 5.2 show that the constructed rule is acceptable, but what happens if the order is increased?

9.2.3 Even faster constructions

The question could be asked if $O(sn\log(n))$ is the lower bound for the component-by-component construction cost. Intuitively, we think the answer is yes for general shift-invariant function spaces. However, for the trivial example of a function space with only constant functions no construction is needed. Are there other nontrivial and useful function spaces which have a lower construction cost than $O(sn\log(n))$?

Here, we think the answer is yes again. Preliminary testing with the unanchored Sobolev space with order 2 weights has shown some interesting results. These were presented as "observations" at the recent Monte Carlo and Quasi-Monte Carlo methods 2007 conference. The observations hint at a deeper algebraic structure, where the number theoretic properties of n dictate the order of the choices for the components of the generating vector.

Two such observations for n a prime number of points p are the following.

Observation 9.1. *There are (many) primes which behave like a prime Fibonacci number for component-by-component construction in the unanchored Sobolev space with order-2 weights: the optimal choice for z_2 is a solution of*

$$z \equiv \sqrt{-1} \pmod{p}. \tag{9.1}$$

Moreover, if $\mathcal{Z}_2$ is the set of equivalent choices for dimension 2 then

$$\frac{1}{|\mathcal{Z}_2|}\sum_{z\in\mathcal{Z}_2} \operatorname{ind}_g(z) \pmod{p} = \varphi(p)/4, \tag{9.2}$$

where here $\operatorname{ind}_g(z) \pmod{p}$ is the discrete logarithm of z to the base g modulo p.

It can easily be proven that for a prime Fibonacci number F_m the solutions to (9.1) are given by F_{m-1} and F_{m-2}, i.e., a Fibonacci lattice rule with a prime number of points has this structure and so do many 2-dimensional lattice rules.

Observation 9.2. *In addition to Observation 9.1 there are (many) primes for which*

$$\operatorname{ind}_g(z_{2k}) \equiv \operatorname{ind}_g(z_{2k-1}) + \varphi(p)/4 \pmod{p}.$$

Thus if $z_{2k-1} \equiv g^\beta$ then $z_{2k} \equiv g^{\beta+\varphi(p)/4}$.

This observation hints at a recurrence relation between the components of the generating vector $\boldsymbol{z}$ in the case of the unanchored Sobolev space with order 2 weights. This observation can even be generalized to the observation of periodic sequences for all k-th sequential differences of the powers of a generator g.

9.3 Final sentence

As can be seen in the previous section, interesting things, triggered by the fast algorithm, are still waiting to be uncovered.

Bibliography

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables*, volume 55 of *National Bureau of Standards Applied Mathematics Series*. U.S. Government Printing Office, Washington, D.C., 1964. Cited on pp. 22, 121.

[2] F. Åkesson and J. P. Lehoczky. Discrete eigenfunction expansion of multi-dimensional Brownian motion and the Ornstein-Uhlenbeck process. Working paper, 1998. Cited on pp. 168.

[3] I. A. Antonov and V. M. Saleev. An economic method of computing LP_τ-sequences. *U.S.S.R. Comput. Math. and Math. Phys.*, 19:256–259, 1979. Cited on pp. 125.

[4] N. Aronszajn. Theory of reproducing kernels. *Trans. Amer. Math. Soc.*, (68):337–404, 1950. Cited on pp. 26.

[5] N. S. Bahvalov. Approximate computation of multiple integrals. *Vestnik Moskov. Univ. Ser. Mat. Meh. Astr. Fiz. Him.*, (4):3–18, 1959. In Russian. Cited on pp. 3, 51.

[6] M. Beckers and R. Cools. A relation between cubature formulae of trigonometric degree and lattice rules. In Brass and Hämmerlin [9], pages 13–24. Cited on pp. 53.

[7] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Kluwer Academic, 2004. Cited on pp. 36.

[8] P. P. Boyle, T. Lai, and K. S. Tan. Pricing options using lattice rules. *North Amer. Act. J.*, pages 50–76, 2005. Cited on pp. 169, 172, 174.

[9] H. Brass and G. Hämmerlin, editors. *Numerical integration IV (Oberwolfach, 1992)*. Birkhäuser Verlag, 1993. Cited on pp. 181, 187.

[10] R. E. Caflisch, W. Morokoff, and A. B. Owen. Valuation of mortgage backed securities using brownian bridges to reduce effective dimension. *J. Comput. Finance*, (1):27–46, 1997. Cited on pp. 39.

[11] H. Cohen. *A Course in Computational Algebraic Number Theory*. Graduate Texts in Mathematics. Springer-Verlag, 3rd edition, 1996. Cited on pp. 70, 85, 87.

[12] H. Cohn. *Advanced Number Theory*. Dover, 1980. Cited on pp. 8.

[13] R. Cools. Constructing cubature formulae: The science behind the art. *Acta Numer.*, 6:1–54, 1997. Cited on pp. 26.

[14] R. Cools and H. Govaert. Five-and six-dimensional lattice rules generated by structured matrices. *J. Complexity*, 19(6):715–729, 2003. Cited on pp. 53, 176.

[15] R. Cools, F. Y. Kuo, and D. Nuyens. Constructing embedded lattice rules for multivariate integration. *SIAM J. Sci. Comput.*, 28(6):2162–2188, 2006. Cited on pp. 82, 142, 176, 177, 178.

[16] R. Cools and J. N. Lyness. Three-and four-dimensional K-optimal lattice rules of moderate trigonometric degree. *Math. Comp.*, 70(236):1549–1567, 2001. Cited on pp. 53.

[17] R. Cools and D. Nuyens. The role of structured matrices for the construction of integration lattices. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 1(3):257–272, 2006. Cited on pp. 53, 142, 176, 178.

[18] R. Cools and A. V. Reztsov. Different quality indexes for lattice rules. *J. Complexity*, 13(2):235–258, 1997. Cited on pp. 26.

[19] R. Cranley and T. N. L. Patterson. Randomization of number theoretic methods for multiple integration. *SIAM J. Numer. Anal.*, 13(6):904–914, 1976. Cited on pp. 14, 15.

[20] P. J. Davis. *Circulant Matrices*. Wiley, 1979. Cited on pp. 60.

[21] J. Dick. On the convergence rate of the component-by-component construction of good lattice rules. *J. Complexity*, 20(4):493–522, Aug. 2004. Cited on pp. 52, 57, 104, 115.

[22] J. Dick and F. Y. Kuo. Constructing good lattice rules with millions of points. In Niederreiter [77], pages 181–197. Cited on pp. 52, 79, 80, 104.

[23] J. Dick and F. Y. Kuo. Reducing the construction cost of the component-by-component construction of good lattice rules. *Math. Comp.*, 73(248):1967–1988, 2004. Cited on pp. 52, 79, 104.

[24] 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. Cited on pp. 143, 151.

[25] J. Dick, G. Leobacher, and F. Pillichshammer. Construction algorithms for digital nets with small weighted star discrepancy. *Finite Fields Appl.*, 2007. To appear. Cited on pp. 153, 177.

[26] J. Dick and F. Pillichshammer. Multivariate integration in weighted Hilbert spaces based on Walsh functions and weighted Sobolev spaces. *J. Complexity*, 21(2):149–149, 2005. Cited on pp. 144, 145, 146, 148.

[27] J. Dick, F. Pillichshammer, and B. Waterhouse. The construction of good extensible rank-1 lattices. *Math. Comp.* To appear. Cited on pp. 142, 177.

[28] J. Dick, I. H. Sloan, X. Wang, and H. Woźniakowski. Liberating the weights. *J. Complexity*, 20(5):593–623, Oct. 2004. Cited on pp. 42.

[29] J. Dick, I. H. Sloan, X. Wang, and H. Woźniakowski. Good lattice rules in weighted Korobov spaces with general weights. *Numer. Math.*, 103(1):63–97, Mar. 2006. Cited on pp. 43.

[30] M. Drmota and R. F. Tichy. *Sequences, Discrepancies and Applications*. Lecture Notes in Mathematics. Springer-Verlag, 1997. Cited on pp. 7.

[31] H. Faure. Discrépences de suites associées à un systéme de numbération (en dimension s). *Acta Arith.*, 41:337–351, 1982. Cited on pp. 4.

[32] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing, Vol. 3*, pages 1381–1384. IEEE, 1998. Cited on pp. 62, 75, 76.

[33] J. A. Gallian. *Contemporary Abstract Algebra*. Houghton Mifflin, 4th edition, 1998. Cited on pp. 85, 87.

[34] A. Genz. Numerical computation of multivariate normal probabilities. *J. Comp. Graph Stat.*, 1:141–149, 1992. Cited on pp. 172.

[35] P. Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer-Verlag, 2003. Cited on pp. 167, 168.

[36] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Studies in the Mathematical Sciences. Johns Hopkins University Press, 3rd edition, 1996. Cited on pp. 61.

[37] J. González, F. Tuerlinckx, P. De Boeck, and R. Cools. Numerical integration in logistic-normal models. *Comput. Statist. Data Anal.* To appear. Cited on pp. 175.

[38] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 2nd edition, 1994. Cited on pp. 87.

[39] R. Gutman. Algorithm alley: Exploiting 64-bit parallelism. *Dr Dobb's J*, 25(9):133–134,136, Sept. 2000. Online at `http://www.ddj.com` with date July 22, 2001. Cited on pp. 125.

[40] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer. Math.*, 2:84–90, 1960. Cited on pp. 4.

[41] P. Hellekalek and G. Larcher, editors. *Random and Quasi-Random Point Sets*. Springer-Verlag, 1998. Cited on pp. 184, 187.

[42] F. J. Hickernell. Quadrature error bounds with applications to lattice rules. *SIAM J. Numer. Anal.*, 33(5):1995–2016, Oct. 1996. Cited on pp. 41.

[43] F. J. Hickernell. A generalized discrepancy and quadrature error bound. *Math. Comp.*, 67(221):299–322, Jan. 1998. Cited on pp. 17, 23, 28, 41.

[44] F. J. Hickernell. Lattice rules: How well do they measure up? In Hellekalek and Larcher [41], pages 109–166. Cited on pp. 7, 17, 23, 28, 41, 43, 53.

[45] F. J. Hickernell. Obtaining $O(n^{-2+\epsilon})$ convergence for lattice quadrature rules. In K. T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 274–289. Springer-Verlag, 2002. Cited on pp. 173.

[46] F. J. Hickernell, H. S. Hong, P. L'Écuyer, and C. Lemieux. Extensible lattice sequences for quasi-Monte Carlo quadrature. *SIAM J. Sci. Comput.*, 22:1117–1138, 2001. Cited on pp. 113, 114, 115, 123, 124, 138.

[47] F. J. Hickernell and H. Niederreiter. The existence of good extensible rank-1 lattices. *J. Complexity*, 19(3):286–300, June 2003. Cited on pp. 113, 124.

[48] F. J. Hickernell and X. Wang. The error bounds and tractability of quasi-Monte Carlo algorithms in infinite dimensions. *Math. Comp.*, 71(240):1641–1661, 2002. Cited on pp. 42.

[49] F. J. Hickernell and H. Woźniakowski. Integration and approximation in arbitrary dimensions. *Adv. Comput. Math.*, 12(1):25–28, 2000. Cited on pp. 42.

[50] F. J. Hickernell and H. Woźniakowski. Tractability of multivariate integration for periodic functions. *J. Complexity*, 17(4):660–682, Dec. 2001. Cited on pp. 48.

[51] E. Hlawka. Über die diskrepanz mehrdimensionaler folgen mod. 1. *Math. Z.*, 77(1):273–284, Dec. 1961. Cited on pp. 8.

[52] J. Imai and K. S. Tan. Minimizing effective dimension using linear transformation. In Niederreiter [77], pages 275–292. Cited on pp. 40.

[53] S. Joe. Component by component construction of rank-1 lattice rules having $O(n^{-1}(\ln(n))^d)$ star discrepancy. In Niederreiter [77], pages 293–298. Cited on pp. 57.

[54] S. Joe and F. Y. Kuo. Remark on Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Trans. Math. Software*, 29(1):49–57, Mar. 2003. Cited on pp. 2.

[55] S. Joe and I. H. Sloan. Embedded lattice rules for multidimensional integration. *SIAM J. Numer. Anal.*, 29:1119–1154, 1992. Cited on pp. 157, 159.

[56] C. Joy, P. P. Boyle, and K. S. Tan. Quasi-Monte Carlo methods in numerical finance. *Management Science*, pages 926–938, 1996. Cited on pp. 169.

[57] D. E. Knuth. *The Art of Computer Programming – Seminumerical Algorithms*, volume 2. Addison-Wesley, 3rd edition, 1998. Cited on pp. 70.

[58] 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. Cited on pp. 19, 21.

[59] N. M. Korobov. Properties and calculation of optimal coefficients. *Dokl. Akad. Nauk SSSR*, 132:1009–1012, 1960. In Russian. English translation see [60]. Cited on pp. 19, 51.

[60] N. M. Korobov. Properties and calculation of optimal coefficients. *Soviet Math. Dokl.*, 1:696–700, 1960. Cited on pp. 185.

[61] N. M. Korobov. *Number-Theoretic Methods in Approximate Analysis*. Goz. Izdat. Fiz.-Math., 1963. In Russian. English translation of results on optimal coefficients in [103]. Cited on pp. 51.

[62] F. Y. Kuo. Component-by-component constructions achieve the optimal rate of convergence for multivariate integration in weighted Korobov and Sobolev spaces. *J. Complexity*, 19:301–320, June 2003. Cited on pp. 49, 52, 57, 78, 79, 115.

[63] F. Y. Kuo and S. Joe. Component-by-component construction of good intermediate-rank lattice rules. *SIAM J. Numer. Anal.*, 41(4):1465–1486, 2003. Cited on pp. 159, 160, 161.

[64] F. Y. Kuo and I. H. Sloan. Lifting the curse of dimensionality. *Notices Amer. Math. Soc.*, 52(11):1320–1328, Dec. 2005. Cited on pp. 47, 53, 112, 177.

[65] F. Y. Kuo, I. H. Sloan, and H. Woźniakowski. Lattice rule algorithms for multivariate approximation in the average case setting. To appear. Cited on pp. 177.

[66] D. P. Laurie. Periodizing transformations for numerical integration. *J. Comput. Appl. Math.*, 66(1-2):337–344, 1996. Cited on pp. 173.

[67] P. L'Écuyer and C. Lemieux. Recent advances in randomized quasi-Monte Carlo methods. In M. Dror, P. L'Écuyer, and F. Szidarovszki, editors, *Modeling Uncertainty: An Examination of Its Theory, Methods, and Applications*, pages 419–474. Kluwer Academic, 2002. Cited on pp. 14.

[68] C. Lemieux and P. L'Écuyer. Efficiency improvement by lattice rules for pricing Asian options. In D. J. Medeiros, E. F. Watson, J. Carson, and M. Manivannan, editors, *Proceedings of the 30th conference on Winter simulation*, pages 579–586. IEEE, 1998. Cited on pp. 169.

[69] C. Lemieux and P. L'Écuyer. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM J. Sci. Comput.*, 24(5):1768–1789, 2002. Cited on pp. 153.

[70] J. N. Lyness. An introduction to lattice rules and their generator matrices. *IMA J. Numer. Anal.*, 9:405–419, 1989. Cited on pp. 9, 26.

[71] J. N. Lyness and T. Sørevik. Four-dimensional lattice rules generated by skew-circulant matrices. *Math. Comp.*, 73(245):279–295, 2004. Cited on pp. 53.

[72] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. Math. Software*, 8(1):3–30, 1998. Cited on pp. 2.

[73] R. E. Moore. *Computational Functional Analysis*. Mathematics and its Applications. Ellis Horwood, 1985. Cited on pp. 23.

[74] H. Niederreiter. On a number-theoretical integration method. *Aequationes Math.*, 8(3):304–311, Oct. 1972. Cited on pp. 5.

[75] H. Niederreiter. Quasi-Monte Carlo methods and pseudo-random numbers. *Bull. Amer. Math. Soc.*, 84(6):957–1041, Nov. 1978. Cited on pp. 52.

[76] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. Number 63 in Regional Conference Series in Applied Mathematics. SIAM, 1992. Cited on pp. 4, 5, 7, 12, 17, 51, 53, 125, 143.

[77] H. Niederreiter, editor. *Monte-Carlo and Quasi-Monte Carlo Methods - 2002*. Springer-Verlag, Jan. 2004. Cited on pp. 182, 185.

[78] H. Niederreiter and C. Xing. Nets, (t, s)-sequences, and algebraic geometry. In Hellekalek and Larcher [41], pages 267–302. Cited on pp. 4.

[79] E. Novak and H. Woźniakowski. Intractability results for integration and discrepancy. *J. Complexity*, 17(2):388–441, June 2001. Cited on pp. 37.

[80] E. Novak and H. Woźniakowski. When are integration and discrepancy tractable? In R. A. DeVore, A. Iserles, and E. Suli, editors, *Foundations of Computational Mathematics*, pages 211–266. Cambridge University Press, 2001. Cited on pp. 3, 37, 47.

[81] D. Nuyens. Fast construction of a good lattice rule / About the cover. *Notices Amer. Math. Soc.*, 52(11):1329 + cover, Dec. 2005. Cited on pp. 104, 112.

[82] D. Nuyens and R. Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. Report TW392, Dept. of Computer Science, K.U.Leuven, May 2004. Cited on pp. 78.

[83] 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(2):903–920, 2006. Cited on pp. 56, 82, 104, 177.

[84] 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. Cited on pp. 82, 153, 178.

[85] D. Nuyens and R. Cools. Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points. *J. Complexity*, 22(1):4–28, 2006. Cited on pp. 104, 112, 177.

[86] S. Paskov and J. F. Traub. Faster evaluation of financial derivatives. *J. Portfolio Management*, 22(1):113–120, 1995. Cited on pp. 40, 167.

[87] C. M. Rader. Discrete Fourier transforms when the number of data samples is prime. *Proc. IEEE*, 5:1107–1108, 1968. Cited on pp. 62, 63, 116.

[88] A. Sidi. A new variable transformation for numerical integration. In Brass and Hämmerlin [9], pages 359–373. Cited on pp. 173.

[89] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Oxford Science Publications, 1994. Cited on pp. 9, 14, 17, 50, 51, 53, 156, 157, 159.

[90] I. H. Sloan and P. J. Kachoyan. Lattice methods for multiple integration: Theory, error analysis and examples. *SIAM J. Numer. Anal.*, 24(1):116–128, 1987. Cited on pp. 9, 20.

[91] I. H. Sloan, F. Y. Kuo, and S. Joe. Constructing randomly shifted lattice rules in weighted Sobolev spaces. *SIAM J. Numer. Anal.*, 40(5):1650–1665, 2002. Cited on pp. 32, 78.

[92] I. H. Sloan and J. N. Lyness. The representation of lattice quadrature rules as multiple sums. *Math. Comp.*, 52(185):81–94, 1989. Cited on pp. 9, 10, 155.

[93] I. H. Sloan and A. V. Reztsov. Component-by-component construction of good lattice rules. *Math. Comp.*, 71(237):263–273, 2002. Cited on pp. 11, 52.

[94] I. H. Sloan and L. Walsh. A computer search of rank 2 lattice rules for multidimensional quadrature. *Math. Comp.*, 54:281–302, 1990. Cited on pp. 157.

[95] I. H. Sloan, X. Wang, and H. Woźniakowski. Finite-order weights imply tractability of multivariate integration. *J. Complexity*, 20(1):46–74, Feb. 2004. Cited on pp. 42, 49.

[96] I. H. Sloan and H. Woźniakowski. An intractability result for multiple integration. *Math. Comp.*, 66(219):1119–1124, July 1997. Cited on pp. 35.

[97] I. H. Sloan and H. Woźniakowski. When are quasi-Monte Carlo algorithms efficient for high dimensional integrals. *J. Complexity*, 14(1):1–33, Mar. 1998. Cited on pp. 27, 41, 48.

[98] I. H. Sloan and H. Woźniakowski. Tractability of multivariate integration for weighted Korobov classes. *J. Complexity*, 17(4):697–721, Dec. 2001. Cited on pp. 27, 48, 49.

[99] I. H. Sloan and H. Woźniakowski. Tractability of integration in non-periodic and periodic weighted tensor product Hilbert spaces. *J. Complexity*, 18(2):479–499, 2002. Cited on pp. 36, 37, 44.

[100] I. M. Sobol'. Distribution of points in a cube and the approximate evaluation of integrals. *Ž. Vyčisl. Mat. i Mat. Fiz.*, 7:784–802, 1967. In Russian. English translation see [101]. Cited on pp. 4, 13.

[101] I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Comput. Math. and Math. Phys.*, 7:86–112, 1967. Cited on pp. 188.

[102] I. M. Sobol'. Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. *Math. Comput. Simulation*, 55(1-3):271–280, 2001. Cited on pp. 38, 40.

[103] A. H. Stroud. *Approximate Calculation of Multiple Integrals*. Automatic Computation. Prentice-Hall, 1971. Cited on pp. 185.

[104] S. Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer Academic, 1995. Cited on pp. 124.

[105] J. F. Traub, G. W. Wasilkowski, and H. Woźniakowski. *Information-Based Complexity*. Computer Science and Scientific Computing. Academic Press, Inc., 1988. Cited on pp. 27.

[106] L. N. Trefethen. Ten digit algorithms – "Ten digits, five seconds, and just one page". Technical report, Oxford University, June 2005. Mitchell Lecture, NA05, Dundee. Cited on pp. 178.

[107] C. F. Van Loan. *Computational Frameworks for the Fast Fourier Transform*, volume 10 of *Frontiers in Applied Mathematics*. SIAM, 1992. Cited on pp. 62, 63, 112, 116.

[108] X. Wang and K.-T. Fang. The effective dimension and quasi-Monte Carlo integration. *J. Complexity*, 19(2):101–124, Apr. 2003. Cited on pp. 170.

[109] X. Wang and I. H. Sloan. Why are high-dimensional finance problems often of low effective dimension? *SIAM J. Sci. Comput.*, 27(1):159–183, 2005. Cited on pp. 40.

[110] T. T. Warnock. Computational investigations of low-discrepancy point sets. In S. K. Zaremba, editor, *Applications of Number Theory to Numerical Analysis*, pages 319–343. Academic Press, 1972. Cited on pp. 4.

[111] C. Xing and H. Niederreiter. A construction of low-discrepancy sequences using global function fields. *Acta Arith.*, 73(1):87–102, 1995. Cited on pp. 4.

[112] S. K. Zaremba. Some applications of multidimensional integration by parts. *Ann. Polon. Math.*, 21:85–96, 1968. Cited on pp. 8.

Snelle constructie van goede roosterregels

Dirk Nuyens
Departement Computerwetenschappen, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, België

Samenvatting

We ontwikkelen een snel algoritme voor de constructie van goede rang-1-roosterregels voor het benaderen van multivariate integralen. Een gekende methode voor de constructie van deze roosterregels is de component-per-component methode. De zo bekomen roosterregels hebben een optimale orde van convergentie. Hun constructiekost is $O(s^2n^2)$ voor een regel in s dimensies met n punten.

We tonen aan hoe we goede rang-1-roosterregels kunnen construeren in tijd $O(sn\log(n))$ en geheugen $O(n)$ met een nieuw algoritme dat we snelle component-per-component constructie noemen. We tonen dit eerst aan in het geval dat n een priemgetal is en de roosterregel wordt opgesteld voor een gewogen, verschuivingsinvariante en tensorproductvorm Hilbertruimte met reproducerende kern. We tonen dit ook aan voor het geval de gewichten orde-afhankelijk zijn en de ruimte hierdoor geen tensorproductvorm meer heeft.

Ook indien n geen priemgetal is, blijkt de snelle constructie mogelijk te zijn. De constructie wordt wel moeilijker indien n uit veel verschillende priemfactoren bestaat. Een interessant geval doet zich voor bij machten van een priemgetal. In dat geval kunnen we op een snelle manier roosterrijen construeren die punt per punt gebruikt kunnen worden.

Twee natuurlijke uitbreidingen van het component-per-component algoritme zijn de constructie van veeltermroosterregels en de constructie van stempelregels. We tonen aan dat ook in deze gevallen een snelle constructie mogelijk is. De kwaliteit van de roosterregels wordt gedemonstreerd aan de hand van enkele praktijkvoorbeelden.

Snelle constructie van goede roosterregels

Inhoudsopgave

1 Inleiding

Voor het benaderen van hoogdimensionale integralen zijn er niet veel alternatieven. Eén van de mogelijkheden zijn kubatuurformules met gelijke gewichten die gebruik maken van puntenverzamelingen met kleine discrepantie. In deze inleiding tonen we aan dat er heel wat redenen zijn voor het gebruik van deze methoden in plaats van de traditionele Monte Carlo methode. Het construeren van puntenverzamelingen met kleine discrepantie is daarom belangrijk.

1.1 Multivariate integratie

Het onderliggende probleem in deze tekst is het benaderen van een s-dimensionale integraal over de eenheidskubus

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

We gebruiken hiervoor een eenvoudige n-punts kubatuurformule Q die de functie f evalueert in de punten $\boldsymbol{x}_k$ en de integraal benadert als een gewogen gemiddelde

$$Q(f;P_n) := \sum_{k=0}^{n-1} w_k\, f(\boldsymbol{x}_k).$$

Als men dit met een klassieke productregel wil doen, dan is het aantal dimensies waarin dit praktisch werkbaar is zeer beperkt. Nemen we bijvoorbeeld een product van eendimensionale kwadratuurformules die elk m punten gebruiken,

$$\int_0^1 \cdots \int_0^1 f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \approx \sum_{k_1=0}^{m-1} w_{k_1}^{(1)} \cdots \sum_{k_s=0}^{m-1} w_{k_s}^{(s)}\, f(x_{k_1}^{(1)},\ldots,x_{k_s}^{(s)}),$$

dan is het totaal aantal punten $n = m^s$ exponentieel in het aantal dimensies. Als we veronderstellen dat de fout voor de eendimensionale kwadratuurformules daalt met $O(m^{-r})$, $r > 0$, dan daalt de fout in s dimensies echter maar als $O(n^{-r/s})$. Dit noemt men *de vloek van de dimensie*: er is een exponentieel aantal punten n nodig om een bepaalde nauwkeurigheid te behalen in de benadering.

Een alternatieve oplossing is het gebruik van een kubatuurformule met gelijke gewichten van de vorm

$$Q(f;P_n) := \frac{1}{n}\sum_{k=1}^{n} f(\boldsymbol{x}_k),$$

waarbij de punten $\boldsymbol{x}_k \in P_n$ uniform verdeeld zijn over de eenheidskubus. Wanneer deze punten willekeurig en onafhankelijk van elkaar gekozen worden, dan noemt met deze techniek *Monte Carlo integratie*. Wanneer deze punten echter deterministisch en "beter dan willekeurig" gekozen worden, dan noemt men deze techniek *quasi-Monte Carlo integratie*.

De Monte Carlo methode en de quasi-Monte Carlo methode bekomen hun benadering door een eenvoudig gemiddelde. Voor de Monte Carlo methode kunnen we gebruik maken van de wet van de grote getallen om aan te tonen dat we de exacte waarde beter en beter benaderen wanneer we meer en meer punten gebruiken. Indien $f \in L_2$ dan vinden we

$$\int_{[0,1)^{ns}} \left(\frac{1}{n}\sum_{k=0}^{n-1} f(\boldsymbol{x}_k) - I(f)\right)^2 \mathrm{d}\boldsymbol{x}_1 \cdots \mathrm{d}\boldsymbol{x}_n = \frac{\sigma^2(f)}{n},$$

met $\sigma^2(f)$ de variantie van de functie. De fout gedraagt zich dus gemiddeld genomen als $O(\sigma(f)\, n^{-1/2})$. We kunnen besluiten dat de Monte Carlo methode een probabilistische foutenschatting toelaat van grootte $O(n^{-1/2})$. Het lijkt er dus op dat we de vloek van de dimensie ontlopen hebben. Dit is echter niet het geval: we hebben enkel de maat waarmee we de fout meten veranderd. Voor de productregel was deze deterministisch, terwijl we hier een verwachte waarde hebben over alle mogelijk selecties van n willekeurige punten.

De quasi-Monte Carlo methode ziet er exact hetzelfde uit als de Monte Carlo methode, alleen gebruiken we nu puntenverzamelingen met kleine discrepantie in plaats van willekeurige punten. Als de fout bij Monte Carlo gemiddeld gezien over alle mogelijke puntenverzamelingen van grootte $O(n^{-1/2})$ is, dan moeten er uiteraard puntenverzamelingen bestaan die minstens even goed zijn. Voor puntenverzamelingen met kleine discrepantie is de fout van grootte $O(n^{-1}\log(n)^s)$. Dit is asymptotisch beter dan Monte Carlo, maar het asymptotisch regime treedt theoretisch pas op bij een aantal punten $n = e^s$ dat exponentieel is in het aantal dimensies. In de praktijk nemen we echter vaak een convergentie van $O(n^{-1})$ waar, zelfs voor zeer grote waarden van s.

Een zeer actief onderzoeksdomein is dan ook het specificeren voor welke functies de quasi-Monte Carlo integratie een effectieve oplossing is. Het blijkt dat zelfs voor schijnbaar zeer redelijke klassen van functies, de numerieke benadering van de multivariate integraal praktisch onmogelijk is wanneer we een deterministische maat willen voor de fout. In §2.2 zullen we functieruimten definiëren waarvoor aangetoond kan worden dat multivariate integratie praktisch mogelijk is onder bepaalde voorwaarden.

1.2 Puntenverzamelingen met kleine discrepantie

De discrepantie is een maat voor de afwijking die een puntenverzameling heeft ten opzichte van de uniforme verdeling. Voor een puntenverzameling P_n met n punten in de s-dimensionale eenheidskubus en waarbij J een willekeurig gebied is in deze kubus, kunnen we de *lokale discrepantie* definiëren als

$$\operatorname{discr}(J, P_n) := \frac{|P_n \cap J|}{n} - \operatorname{vol}(J).$$

Dit is het verschil tussen het aantal punten in het gebied J en het verwachte aantal punten in dit gebied. Op basis van deze lokale discrepantie kunnen we een maat definiëren over een verzameling van gebieden, die dan de discrepantie van de puntenverzameling over de hele s-dimensionale kubus geeft.

Een veel gebruikte discrepantie is de *sterdiscrepantie* $D^\star$ waarbij de gebieden J gekozen worden als rechthoekige gebieden met één hoekpunt gelijk aan de oorsprong:

$$J = [\mathbf{0}, \boldsymbol{x}) = [0, x_1) \times \cdots \times [0, x_s).$$

De klassieke definities gebruiken doorgaans een L_∞-norm, maar eender welke L_p-norm is mogelijk. De L_p-sterdiscrepanties zijn gedefinieerd als

$$D_p^\star(P_n) := \left(\sum_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \int_{[0,1)^{|\mathfrak{u}|}} \Big| \operatorname{discr}([\mathbf{0}, \boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u})) \Big|^p \, \mathrm{d}\boldsymbol{x}_\mathfrak{u} \right)^{1/p}, \qquad 1 \leq p < \infty,$$

$$D_\infty^\star(P_n) := \max_{\emptyset \neq \mathfrak{u} \subseteq \mathcal{D}_s} \sup_{\boldsymbol{x}_\mathfrak{u} \in [0,1)^{|\mathfrak{u}|}} \left| \operatorname{discr}\left([\mathbf{0}, \boldsymbol{x}_\mathfrak{u}), P_n(\mathfrak{u})\right) \right| = \sup_{\boldsymbol{x} \in [0,1)^s} \left| \operatorname{discr}\left([\mathbf{0}, \boldsymbol{x}), P_n\right) \right|,$$

waarbij met $P_n(\mathfrak{u})$ de projectie van de puntenverzameling op de coördinaten in $\mathfrak{u}$ bedoeld wordt en $\boldsymbol{x}_\mathfrak{u}$ gedefinieerd is als de vector met componenten uit $\boldsymbol{x}$ voor de coördinaten in $\mathfrak{u}$.

Door de discrepantie over alle mogelijke projecties van de puntenverzameling te bekijken (zoals in bovenstaande definities), is het mogelijk om een algemene Koksma–Hlawka-foutengrens op te stellen. De fout wordt dan begrensd als

$$|Q(f; P_n) - I(f)| \leq D_p(P_n)\, V_q(f), \qquad \frac{1}{p} + \frac{1}{q} = 1, \quad p \geq 1, \tag{1}$$

waarbij $V_q(f)$ de *variatie* van de functie f is. De variatie is een maat voor de ruwheid van een functie en we kunnen ze uitdrukken als een (semi-)norm in een functieruimte. We slaan de klassieke definities van de s-dimensionale variatie dan ook over en bestuderen de fout van de kubatuurformule $Q(f; P_n)$ in een genormeerde ruimte, zie §2.1.

De Koksma–Hlawka-foutengrens, zie formule (1), toont duidelijk dat de fout van de kubatuurformule als twee afzonderlijke delen bestudeerd kan worden. Eén

deel heeft enkel te maken met de functie, het andere deel heeft enkel te maken met de puntenverzameling. In normale omstandigheden weten we niet voor welke functies de kubatuurformule gebruikt zal worden; de puntenverzameling daarentegen wordt wel op voorhand geconstrueerd. We concluderen daarom dat het cruciaal is om puntenverzamelingen met kleine discrepantie te kunnen construeren. Constructie van zulke puntenverzamelingen is net het onderwerp van deze thesis.

2 Roosterregels en functieruimten

Een roosterregel gebruikt een puntenverzameling die geïnterpreteerd kan worden als een eindige cyclische additieve subgroep van $\mathbb{R}^s/\mathbb{Z}^s$. Met andere woorden, indien $\boldsymbol{x}$ en $\boldsymbol{y}$ punten zijn van de puntenverzameling P_n van een roosterregel, dan hebben we de volgende eigenschappen:

(i) $\boldsymbol{x}+\boldsymbol{y} \in P_n$ (gesloten);
(ii) $(\boldsymbol{x}+\boldsymbol{y})+\boldsymbol{z} = \boldsymbol{x}+(\boldsymbol{y}+\boldsymbol{z})$ (associatief);
(iii) $\exists \mathbf{0} \in P_n : \boldsymbol{x}+\mathbf{0} = \boldsymbol{x}$ (neutraal element);
(iv) $\exists(-\boldsymbol{x}) \in P_n : \boldsymbol{x}+(-\boldsymbol{x}) = \mathbf{0}$ (invers);
(v) $\boldsymbol{x}+\boldsymbol{y} = \boldsymbol{y}+\boldsymbol{x}$ (commutatief).

De operatie in de bovenstaande lijst van eigenschappen is de componentsgewijze optelling modulo 1. Doordat P_n een deelgroep is van de quotientgroep $\mathbb{R}^s/\mathbb{Z}^s$ liggen de punten in de halfopen s-dimensionale eenheidskubus $[0,1)^s$ en is de oorsprong $\mathbf{0}$ steeds een element van P_n.

We kunnen nu de definitie van een roosterregel geven.

Definitie 1. Een *roosterregel* is een kubatuurformule Q met gelijke gewichten,

$$Q(f) = \frac{1}{n} \sum_{\boldsymbol{x}_k \in P_n} f(\boldsymbol{x}_k),$$

waarbij de n punten van de puntenverzameling P_n een eindige cyclische additieve subgroep van $\mathbb{R}^s/\mathbb{Z}^s$ vormen onder de operatie van componentsgewijze optelling modulo 1.

Doordat P_n een eindige groep is met n elementen, volgt dat we een verzameling van generatoren $G = \{\boldsymbol{z}_i/n_i\}_{i=1}^t$ kunnen vinden zodanig dat $P_n = \langle G \rangle$. Met andere woorden, elk punt $\boldsymbol{x} \in P_n$ kan geschreven worden als een lineaire combinatie van de generatoren:

$$\boldsymbol{x} = \sum_{i=1}^{t} \lambda_i \boldsymbol{z}/n_i \bmod 1, \qquad \boldsymbol{\lambda} \in \mathbb{Z}^s.$$

De generatoren zijn niet uniek bepaald. Het minimum aantal generatoren nodig om de verzameling P_n te genereren wordt de *rang* van de roosterregel genoemd. Uiteraard is de rang een geheel getal tussen 1 en s.

In deze tekst zullen we hoofdzakelijk roosterregels met rang 1 beschouwen. We geven daarom een specifieke definitie voor rang-1-regels. Het is gebruikelijk om een kortere notatie te gebruiken in plaats van de modulo 1, daarvoor gebruiken we gekrulde haken:

$$\{\boldsymbol{x}\} := \boldsymbol{x} \bmod 1.$$

Definitie 2. Een *rang*-1-*roosterregel* is een kubatuurformule Q met gelijke gewichten,

$$Q(f) = \frac{1}{n} \sum_{k=0}^{n-1} f\left(\left\{\frac{k\boldsymbol{z}}{n}\right\}\right),$$

waarbij $\boldsymbol{z} \in \mathbb{Z}^s$ de *genererende vector* genoemd wordt. De roosterregel heeft n verschillende punten indien $\gcd(z_1, \ldots, z_s, n) = 1$ en elke eendimensionale projectie heeft ook n verschillende punten indien $\gcd(z_j, n) = 1$ voor alle $j = 1, \ldots, s$.

Een "goede" roosterregel is een puntenverzameling met een kleine discrepantie. Hiervoor is het noodzakelijk om voor een gekozen n een goede keuze te maken van de genererende vector $\boldsymbol{z} \in \mathbb{Z}^s$. Het zoeken naar zo een "goede" vector is het onderwerp van deze thesis. In plaats van naar de discrepantie te kijken, zullen we vanaf nu werken met de ergst-mogelijke fout in een bepaalde functieruimte voor functies met norm kleiner of gelijk aan 1 (zoals hogerop vermeld is dit een equivalente visie).

2.1 Hilbertruimten met reproducerende kern

We beschouwen hier een Hilbertruimte $\mathcal{H}$ van reële functies $f : [0,1)^s \to \mathbb{R}$. Deze Hilbertruimte heeft een reproducerende kern $K : [0,1)^s \times [0,1)^s \to \mathbb{R}$ indien deze functie K de *reproducerende eigenschap* heeft:

$$f(\boldsymbol{y}) = \langle f, K(\cdot, \boldsymbol{y})\rangle_{\mathcal{H}}, \qquad \forall f \in \mathcal{H},\ \forall \boldsymbol{y} \in [0,1)^s.$$

Door gebruik te maken van zo een functieruimte kunnen we een formule afleiden die equivalent is aan (1). Hiervoor steunen we op de Rieszrepresentatie-eigenschap voor lineaire functionalen. Dit is samengevat in de volgende stelling.

Stelling 3. *In een Hilbertruimte $\mathcal{H}(K)$ met reproducerende kern K bestaat er een foutvertegenwoordiger $\xi_{K,Q} \in \mathcal{H}(K)$ voor de fout van de kubatuurformule Q zodanig dat $I(f) - Q(f) = \langle f, \xi_{K,Q}\rangle_{\mathcal{H}}$ voor alle functies $f \in \mathcal{H}(K)$. De fout kan hierdoor begrensd worden door*

$$|I(f) - Q(f)| \leq \|f\|_{\mathcal{H}}\, \|\xi_{K,Q}\|_{\mathcal{H}}.$$

We definiëren nu een maat voor de integratiefout in een ruimte $\mathcal{H}$.

Definitie 4. De *ergst-mogelijke fout* van een kubatuurformule Q voor het benaderen van de integraal voor functies uit de eenheidsbol van de functieruimte $\mathcal{H}$, is gedefinieerd als

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

Uit Stelling 3 en Definitie 4 kunnen we afleiden dat voor $\mathcal{H}$ een Hilbertruimte met reproducerende kern K, de ergst-mogelijke fout gegeven wordt als de norm van de foutvertegenwoordiger:

$$e(Q,\mathcal{H}) = e(Q,K) = \|\xi_{K,Q}\|_{\mathcal{H}}.$$

Doordat we werken in een Hilbertruimte met reproducerende kern, kunnen we de ergst-mogelijke fout neerschrijven in een expliciete formule in functie van de reproducerende kern.

Stelling 5. *Het kwadraat van de ergst-mogelijke fout voor integratie van functies in $\mathcal{H}(K)$ met de kubatuurformule Q (met gelijke gewichten) is gelijk aan*

$$e^2(Q,K) = \int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} - \frac{2}{n}\sum_{k=0}^{n-1}\int_{[0,1)^s} K(\boldsymbol{x}_k,\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} + \frac{1}{n^2}\sum_{k,\ell=0}^{n-1} K(\boldsymbol{x}_k,\boldsymbol{x}_\ell),$$

met $\boldsymbol{x}_k,\boldsymbol{x}_\ell \in P_n$ voor $k,\ell = 0,\ldots,n-1$ de punten van de puntenverzameling P_n gebruikt door Q.

De reproducerende kernen die we in deze tekst beschouwen zijn *verschuivingsinvariant* (eventueel na een uitmiddelingstransformatie over alle mogelijke verschuivingen). Dit betekent dat voor $\forall \boldsymbol{x},\boldsymbol{y},\boldsymbol{\Delta} \in [0,1)^s$

$$K(\{\boldsymbol{x}+\boldsymbol{\Delta}\},\{\boldsymbol{y}+\boldsymbol{\Delta}\}) = K(\boldsymbol{x},\boldsymbol{y}). \tag{2}$$

Een onmiddellijk gevolg hiervan is dat de kern kan geschreven worden in functie van één argument:

$$K(\boldsymbol{x},\boldsymbol{y}) = K(\{\boldsymbol{x}-\boldsymbol{y}\},\boldsymbol{0}) =: K(\{\boldsymbol{x}-\boldsymbol{y}\}).$$

Het eenvoudigste geval voor de snelle constructie die in deze tekst wordt behandeld, beschouwt verder enkel maar tensorproductruimten. Daarbij is de reproducerende kern in een s-dimensionale ruimte het product van s eendimensionale kernen:

$$K(\boldsymbol{x}) = \prod_{j=1}^{s} \eta_j(x_j).$$

Indien de reproducerende kern deze eigenschappen heeft en we met een roosterregel werken, dan vereenvoudigt de formule voor de ergst-mogelijke fout.

Gevolg 6. *Als de reproducerende kern verschuivingsinvariant is, de tensorproductvorm heeft en Q is een roosterregel, dan is*

$$e^2(Q,K) = -\prod_{j=1}^{s}\int_0^1 \eta_j(x)\,\mathrm{d}x + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\eta_j(x_j^{(k)}), \tag{3}$$

met η_j de kernen van de eendimensionale Hilbertruimten met reproducerende kern, zodat $K(\boldsymbol{x}) = \prod_{j=1}^{s}\eta_j(x_j)$.

Formule (3) is de basisformule voor de theoretische analyse van het component-per-component algoritme in het vervolg van de tekst.

2.2 Gewogen Hilbertruimten met reproducerende kern

Een s-dimensionale functie kan steeds geschreven worden als een som van functies over alle 2^s mogelijke projecties

$$f(\boldsymbol{x}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} f_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}}),$$

waarbij $\mathcal{D}_s = \{1,\ldots,s\}$. Een mogelijke decompositie wordt gegeven door de ANOVA- of variantieanalysedecompositie. De projectie over $\mathfrak{u}\subseteq\mathcal{D}_s$ wordt hierbij gegeven door

$$f_{\mathfrak{u}}(\boldsymbol{x}) := \int_{[0,1)^{s-|\mathfrak{u}|}} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}} - \sum_{\mathfrak{v}\subset\mathfrak{u}} f_{\mathfrak{v}}(\boldsymbol{x}).$$

Aan de hand van deze ontbinding kunnen we het belang van verschillende projecties van een functie onderzoeken. Voor een hoogdimensionale functie is het zeer onwaarschijnlijk dat alle 2^s projecties even belangrijk zijn. Misschien zijn enkel de eerste t variabelen belangrijk, of misschien zijn enkel de tweedimensionale projecties belangrijk...

Dit inzicht kunnen we gebruiken om gewogen functieruimten te definiëren. Voor positieve gewichten $\gamma_{\mathfrak{u}} \geq 0$ beschouwen we een Hilbertruimte met reproducerende kern gegeven door

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} K_{\mathfrak{u}}(\boldsymbol{x},\boldsymbol{y}),$$

waarbij de reproducerende kernen $K_{\mathfrak{u}}$ gegeven worden door

$$K_{\mathfrak{u}}(\boldsymbol{x},\boldsymbol{y}) = K_{\mathfrak{u}}(\boldsymbol{x}_{\mathfrak{u}},\boldsymbol{y}_{\mathfrak{u}}) = \prod_{j\in\mathfrak{u}}\eta(x_j,y_j).$$

Met elke $K_{\mathfrak{u}}$ kunnen we een verschuivingsinvariante kern $K_{\mathfrak{u}}^{\text{shinv}}$ associëren,

$$K_{\mathfrak{u}}^{\text{shinv}}(\boldsymbol{x}) = \prod_{j\in\mathfrak{u}} \omega(x_j) \qquad \text{met} \qquad \omega(x) := \int_0^1 \eta\left(\{x+\Delta\},\Delta\right) \mathrm{d}\Delta, \tag{4}$$

en dus

$$K^{\text{shinv}}(\boldsymbol{x}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} K_{\mathfrak{u}}^{\text{shinv}}(\boldsymbol{x}).$$

We kiezen $K_\emptyset = 1$ en $\gamma_\emptyset = 1$. Hierdoor hebben we de volgende equivalente eigenschappen

$$\int_{[0,1)^{2s}} K(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} = 1, \quad \int_{[0,1)^s} K^{\text{shinv}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = 1 \quad \text{en} \quad \int_0^1 \omega(x)\,\mathrm{d}x = 0.$$

Het specificeren van 2^s gewichten is uiteraard een onbegonnen zaak, zelfs wanneer s relatief klein is. Daarom wordt er een structuur opgelegd aan de gewichten. We geven twee interessante mogelijkheden om het aantal gewichten te beperken.

(i) *Productgewichten.* Hierbij kiezen we s gewichten

$$\gamma_{\{1\}} \geq \cdots \geq \gamma_{\{s\}} \geq 0, \tag{5}$$

één per dimensie, en stellen we

$$\gamma_{\mathfrak{u}} = \prod_{j\in\mathfrak{u}} \gamma_{\{j\}}.$$

Verkort noteren we γ_j in plaats van $\gamma_{\{j\}}$. De ordening in (5) is een reflectie van de veronderstelling dat de eerste dimensies belangrijker zijn dan de latere dimensies. Een ruimte gewogen met productgewichten is een tensorproductruimte omdat

$$\begin{aligned} K(\boldsymbol{x},\boldsymbol{y}) &= \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \prod_{j\in\mathfrak{u}} \gamma_j\,\eta(x_j,y_j) \\ &= \prod_{j=1}^{s} \left(1+\gamma_j\,\eta(x_j,y_j)\right). \end{aligned}$$

De ergst-mogelijke fout voor een roosterregel heeft hierbij de vorm

$$e(P_n,K^{\text{shinv}}) = \left(-1+\frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\left(1+\gamma_j\,\omega(x_j)\right)\right)^{1/2}. \tag{6}$$

(ii) *Orde-afhankelijke gewichten.* Een andere mogelijkheid is om alle orde-ℓ interacties op dezelfde manier te wegen:

$$\gamma_{\mathfrak{u}} = \Gamma_{|\mathfrak{u}|}, \qquad \text{met } \Gamma_{\emptyset} = \gamma_{\emptyset} = 1.$$

Hierdoor is het mogelijk om bijvoorbeeld enkel eendimensionale en tweedimensionale projecties te hebben door de gewichten $\Gamma_\ell = 0$ te stellen voor $\ell > 2$. Dit noemen we dan orde-afhankelijke gewichten van orde 2. De ergst-mogelijke fout voor een roosterregel heeft hier de vorm

$$e(P_n, K^{\text{shinv}}) = \left(\frac{1}{n} \sum_{k=0}^{n-1} \sum_{\ell=1}^{s} \Gamma_\ell \sum_{\substack{\mathfrak{u} \subseteq \mathcal{D}_s \\ |\mathfrak{u}|=\ell}} \prod_{j \in \mathfrak{u}} \omega(x_j^{(k)}) \right)^{1/2}. \tag{7}$$

Twee belangrijke functieruimten zijn de Korobovruimte en de Sobolevruimte.

Definitie 7. De *gewogen Korobovruimte* heeft een reproducerende kern

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\mathfrak{u} \subseteq \mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j \in \mathfrak{u}} \sum_{0 \neq h \in \mathbb{Z}} r(h)^{-\alpha} \exp(2\pi \mathrm{i}\, h(x_j - y_j)),$$

inwendig product

$$\langle f, g \rangle = \sum_{\boldsymbol{h} \in \mathbb{Z}^s} \gamma_{\mathfrak{u}_{\boldsymbol{h}}}^{-1} r(\boldsymbol{h})^{\alpha}\, \hat{f}(\boldsymbol{h})\, \overline{\hat{g}(\boldsymbol{h})},$$

en norm

$$\|f\| = \left(\sum_{\boldsymbol{h} \in \mathbb{Z}^s} \gamma_{\mathfrak{u}_{\boldsymbol{h}}}^{-1} r(\boldsymbol{h})^{\alpha}\, |\hat{f}(\boldsymbol{h})|^2 \right)^{1/2}.$$

Hierbij is $\mathfrak{u}_{\boldsymbol{h}} := \{j \in \mathfrak{u} : h_j \neq 0\}$ en bepaalt $\alpha > 1$ de zachtheid van de functies. De functie $r(\boldsymbol{h})$ is gedefinieerd als

$$r(\boldsymbol{h}) := \prod_{j=1}^{s} r(h_j) \qquad \text{met } r(h) = \max(1, |h|).$$

De Korobovruimte is van nature verschuivingsinvariant; de functie ω uit (4) is hier gegeven als

$$\omega(x) = 2\pi^2 \left(x^2 - x + \frac{1}{6} \right).$$

Definitie 8. De *gewogen Sobolevruimte zonder ankerpunt* heeft een reproducerende kern

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}} \prod_{j\in\mathfrak{u}} \tfrac{1}{2}B_2(\{x_j - y_j\}) + (x_j - \tfrac{1}{2})(y_j - \tfrac{1}{2}),$$

inwendig product $\langle f, g\rangle =$

$$\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1)^{|\mathfrak{u}|}} \left(\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}} \,\mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right)\left(\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} g(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}} \,\mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right) \mathrm{d}\boldsymbol{x}_{\mathfrak{u}},$$

en norm

$$\|f\| = \left(\sum_{\mathfrak{u}\subseteq\mathcal{D}_s} \gamma_{\mathfrak{u}}^{-1} \int_{[0,1)^{|\mathfrak{u}|}} \left|\int_{[0,1)^{s-|\mathfrak{u}|}} \frac{\partial^{|\mathfrak{u}|} f(\boldsymbol{x})}{\partial \boldsymbol{x}_{\mathfrak{u}}} \,\mathrm{d}\boldsymbol{x}_{\mathcal{D}_s\setminus\mathfrak{u}}\right|^2 \mathrm{d}\boldsymbol{x}_{\mathfrak{u}}\right)^{1/2}.$$

De Sobolevruimte maken we verschuivingsinvariant door het gemiddelde te nemen over alle mogelijke verschuivingen. De ω-functie uit (4) heeft hier de volgende vorm

$$\omega(x) = x^2 - x + \frac{1}{6}.$$

Men kan aantonen dat voor de ergst-mogelijke fout in ongewogen Korobov- of Sobolevruimten (m.a.w. alle gewichten $\gamma_{\mathfrak{u}}$ gelijk aan 1) er een exponentieel aantal punten (in het aantal dimensies) nodig is om de kubatuurbenadering te laten convergeren. Dit is de gekende vloek van de dimensie en het benaderen van integralen is dan uiteraard niet praktisch haalbaar. Indien de gewichten echter aan bepaalde voorwaarden voldoen, dan bekomt men wel convergentie met een aanvaardbaar aantal punten. Het aantal punten is dan polynomiaal in het aantal dimensies of zelfs totaal niet afhankelijk van het aantal dimensies, maar enkel van de gevraagde nauwkeurigheid.

2.3 Component-per-component constructie

Enkel in twee dimensies is er een expliciete formule gekend die de optimale genererende vector geeft. Daarom is men voor de constructie van roosterregels in hogere dimensies aangewezen op een exhaustieve zoekoperatie. Uit Definitie 2 volgt dat de componenten van de genererende vector beperkt kunnen worden tot $\mathbb{Z}_n = \{0,\ldots,n-1\}$. Indien we ook eisen dat $\gcd(z_j, n) = 1$, dan beperkt de keuze zich tot de multiplicatieve groep modulo n:

$$U_n := \{v \in \mathbb{Z}_n : \gcd(v,n) = 1\} = \mathbb{Z}_n^{\times}, \qquad |U_n| = \varphi(n).$$

Voor het zoeken van de s-dimensionale vector $\boldsymbol{z}$ zijn er dus $O(n^s)$ mogelijkheden. (De Euler totient functie $\varphi(n)$ is van orde grootte $O(n)$.)

Een manier om deze zoekruimte te beperken is het component-per-component algoritme van Sloan en Reztsov [93]. Hierbij zoekt men eerst z_1 door het minimaliseren van de ergst-mogelijke fout voor een eendimensionale roosterregel, vervolgens zoekt men z_2 en minimaliseert men de fout voor een tweedimensionale roosterregel terwijl men de keuze van z_1 behoudt, enzovoorts... Door middel van een inductief bewijs kan aangetoond worden dat, indien men een goede roosterregel heeft in s dimensies, men op deze manier een goede roosterregel kan vinden in $s+1$ dimensies. Dit proces is neergeschreven in Algoritme 1 waarbij $s_{\max}$ een open parameter is. Op deze manier moet men maar s keer de ergst-mogelijke fout berekenen in plaats van een exponentieel aantal keren.

Algoritme 1 Component-per-component constructie

for $s = 1$ **to** $s_{\max}$ **do**
  for all $z_s \in U_n$ **do**
    bereken $e_s^2(z_1, \ldots, z_{s-1}, z_s)$
  end for
  $z_s = \underset{z \in U_n}{\operatorname{argmin}}\, e_s^2(z_1, \ldots, z_{s-1}, z)$
end for

De constructiekost voor dit algoritme is $O(s^2 n^2)$ voor de ergst-mogelijke fout in een verschuivingsinvariante Hilbertruimte met reproducerende kern en productgewichten, zie (3) en (6). Kuo [62] toonde aan dat dit algoritme optimale roosterregels construeert in Korobov- en Sobolevruimten voor gepaste gewichten (zie ook Dick [21]). Het is dit algoritme dat we in de rest van de tekst zullen versnellen tot $O(sn\log(n))$.

3 Constructies met een priem aantal punten

De puntenverzameling van een rang-1-roosterregel kunnen we noteren met een modulaire notatie als

$$P_n := \left\{ \frac{k \cdot \boldsymbol{z}}{n} : 0 \leq k < n \right\},$$

waarbij we met $k \cdot \boldsymbol{z}$ componentsgewijze vermenigvuldiging modulo n bedoelen. We leggen met deze notatie de nadruk op de structuur van de multiplicatieve groep modulo n. Het uitbuiten van deze structuur zal ons het snelle component-per-component algoritme opleveren.

Voor de component-per-component constructie in een verschuivingsinvariante Hilbertruimte met reproducerende kern met productgewichten moeten we formule (5) uitrekenen voor iedere mogelijke keuze van $z_s \in U_n$, en dat voor elke dimensie

s. We kunnen de formule voor het kwadraat van de ergst-mogelijke fout eenvoudig in twee delen splitsen om het iteratieve karakter te benadrukken. Hiervoor splitsen we het product over de s dimensies als volgt:

$$\prod_{j=1}^{s}\left(1+\gamma_j\,\omega\!\left(\frac{k\cdot z_j}{n}\right)\right)=\prod_{j=1}^{s-1}\left(1+\gamma_j\,\omega\!\left(\frac{k\cdot z_j}{n}\right)\right)\left(1+\gamma_s\,\omega\!\left(\frac{k\cdot z_s}{n}\right)\right)$$
$$=p_{s-1}(k)\left(1+\gamma_s\,\omega\!\left(\frac{k\cdot z_s}{n}\right)\right).$$

Hierin bevat de vector $\boldsymbol{p}_{s-1}\in\mathbb{R}^n$ alle vorige producten. De ergst-mogelijke fout kunnen we dan schrijven als

$$e_s^2(z)=-1+\frac{1}{n}\sum_{k=0}^{n-1}p_{s-1}(k)\;\left(1+\gamma_s\,\omega\!\left(\frac{k\cdot z}{n}\right)\right),\quad\forall z\in U_n. \tag{8}$$

Deze formule moeten we berekenen voor elke mogelijke keuze van $z_s\in U_n$ en per iteratiestap hebben we dan een kost van $O(n^2)$. De kost tot s dimensies is dus $O(sn^2)$ met het gebruik van $O(n)$ geheugen voor de vector $\boldsymbol{p}$.

3.1 De matrix $\boldsymbol{\Xi}_n$

We kunnen eenvoudig inzien dat (8) in feite een matrix-vector-vermenigvuldiging is. We definiëren de matrix

$$\boldsymbol{\Omega}_n:=\left[\omega\!\left(\frac{k\cdot z}{n}\right)\right]_{\substack{z=1,\ldots,n-1\\k=0,\ldots,n-1}}\qquad(n\text{ priem}),$$

en we bekomen de gevectoriseerde uitdrukking

$$\boldsymbol{e}_s^2=-1+\frac{1}{n}(\mathbf{1}_{n-1\times n}+\gamma_s\,\boldsymbol{\Omega}_n)\,\boldsymbol{p}_{s-1},$$

met $\mathbf{1}_{n-1\times n}$ een $n-1\times n$ matrix volledig gevuld met het cijfer één. De vector $\boldsymbol{p}$ wordt op het einde van elke iteratiestap aangepast als volgt

$$\boldsymbol{p}_s=(\mathbf{1}_{n\times 1}+\gamma_s\,\boldsymbol{\Omega}_n(z_s,:))\;.*\;\boldsymbol{p}_{s-1},$$

waarbij $\boldsymbol{\Omega}_n(z_s,:)$ de rij is corresponderende met z_s en de notatie $.*$ een elementsgewijze vermenigvuldiging voorstelt.

De matrix $\boldsymbol{\Omega}_n$ wordt enkel geëvalueerd in n verschillende punten en kan dus maximaal n verschillende waarden bevatten. De structuur van deze matrix wordt volledig gedicteerd door de modulo n structuur; de ω-functie kunnen we totaal willekeurig kiezen. Daarom definiëren we een gelijkaardige matrix:

$$\boldsymbol{\Xi}_n:=\big[k\cdot z\big]_{\substack{z=1,\ldots,n-1\\k=0,\ldots,n-1}}\qquad(n\text{ priem}). \tag{9}$$

We kunnen het grootste stuk van deze matrix nu interpreteren als de vermenigvuldigingstabel van de groep U_n: enkel de kolom voor $k = 0$ moeten we weglaten, omdat voor n priem $U_n = \{1, \ldots, n-1\} = \mathbb{Z}_n \setminus \{0\}$. Het is duidelijk dat als we een snelle methode kunnen vinden voor het matrix-vector-product met matrices van deze structuur, we het component-per-component algoritme drastisch zullen kunnen versnellen.

3.2 Een snel algoritme

Doordat we n priem gekozen hebben, is de groep U_n cyclisch. Dat wil zeggen dat we een *generator* $g \in U_n$ kunnen vinden waarvan de machten de volledige groep U_n vormen:

$$\begin{aligned} \langle g \rangle &:= \{g^k \bmod n : k \in \mathbb{Z}\} \\ &= U_n \qquad\qquad (g \text{ een generator van } U_n). \end{aligned}$$

Omdat $|U_n| = \varphi(n)$ weten we dat we de machten ook modulo $\varphi(n)$ mogen schrijven en dus, indien g een generator is, hebben we $U_n = \{g^k \bmod n : k \in \mathbb{Z}_{\varphi(n)}\}$.

We definiëren nu eerst wat een circulante matrix is.

Definitie 9. Een *circulante matrix* $\boldsymbol{C}_m = \operatorname{circ}(\boldsymbol{c})$ van orde m is een matrix gedefinieerd door de m elementen van de vector $\boldsymbol{c}$, geïndexeerd vanaf 0, als

$$\big[\boldsymbol{C}_m\big]_{k,\ell} = c_{k-\ell \bmod m}.$$

Een zeer belangrijke eigenschap van circulante matrices is dat het matrix-vector-product op een snelle manier kan uitgevoerd worden door gebruik te maken van de snelle Fouriertransformatie, zie bijvoorbeeld [36, p. 201].

Stelling 10. *Een matrix-vector-product van een vector $\boldsymbol{x}$ met een circulante matrix $\boldsymbol{C}_m$ kan uitgevoerd worden in tijd $O(m \log(m))$ door gebruik te maken van de eigenwaardenontbinding van $\boldsymbol{C}_m$:*

$$\boldsymbol{C}_m\, \boldsymbol{x} = \boldsymbol{F}_m^{-1}\, \operatorname{diag}(\boldsymbol{F}_m\, \boldsymbol{c})\, \boldsymbol{F}_m\, \boldsymbol{x}.$$

Hierbij is $\boldsymbol{F}_m^{-1}\, \operatorname{diag}(\boldsymbol{F}_m\, \boldsymbol{c})\, \boldsymbol{F}_m$ de eigenwaardenontbinding van $\boldsymbol{C}_m$, met

$$\boldsymbol{F}_m := \big[\exp(-2\pi \mathrm{i}\, k\ell/m)\big]_{\substack{k=0,\ldots,m-1 \\ \ell=0,\ldots,m-1}}$$

de discrete Fouriertransformatie van orde m.

Indien de Fouriertransformaties uitgevoerd worden door het snelle Fourieralgoritme, dan is de totale rekentijd $O(m \log(m))$. De snelle Fouriertransformatie is mogelijk voor eender welke m.

We kunnen nu het verband leggen tussen een cyclische groep van orde m en een circulante matrix van orde m. Uit Definitie 9 volgt dat indien we de elementen in de vermenigvuldigingstabel van U_n in verticale richting schrijven in volgorde van de machten van een generator, en in horizontale richting in negatieve machten van diezelfde generator, we een circulante matrix bekomen van orde $\varphi(n)$. We illustreren dit hieronder waarbij er links in beide richtingen de volgorde van een generator gebruikt werd en rechts de volgorde zoals hierboven beschreven werd.

$$\begin{array}{c|ccccc} \cdot & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ \hline g^0 & g^0 & g^1 & g^2 & \cdots & g^{-1} \\ g^1 & g^1 & g^2 & g^3 & \iddots & g^0 \\ g^2 & g^2 & g^3 & \iddots & \iddots & g^1 \\ \vdots & \vdots & \iddots & \iddots & \iddots & \vdots \\ g^{-1} & g^{-1} & g^0 & g^1 & \cdots & g^{-2} \end{array} \simeq \begin{array}{c|ccccc} \cdot & g^0 & g^{-1} & g^{-2} & \cdots & g^1 \\ \hline g^0 & g^0 & g^{-1} & g^{-2} & \dots & g^1 \\ g^1 & g^1 & g^0 & g^{-1} & \ddots & \vdots \\ g^2 & g^2 & g^1 & \ddots & \ddots & g^{-2} \\ \vdots & \vdots & \ddots & \ddots & \ddots & g^{-1} \\ g^{-1} & g^{-1} & \dots & g^2 & g^1 & g^0 \end{array}$$

Daar het component-per-component algoritme in essentie een matrix-vector-product van deze vorm bevat, bekomen we een snel algoritme.

Stelling 11. *Component-per-component constructie van een rang-1-roosterregel met n punten in s dimensies, met n priem, in een verschuivingsinvariante en tensorproduct Hilbertruimte met reproducerende kern, uitgerust met productgewichten, kost $O(sn\log(n))$ tijd en $O(n)$ geheugen.*

Ook voor een ruimte met orde-afhankelijke gewichten is dit mogelijk. Opnieuw moeten we hiervoor de formule voor het kwadraat van de ergst-mogelijke fout iteratief schrijven. We vertrekken nu van formule (7) en vinden

$$\begin{aligned} e_s^2(z_s) &= \frac{1}{n}\sum_{k=0}^{n-1}\sum_{\ell=1}^{s} \Gamma_\ell \left(\underbrace{\sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_{s-1}\\ |\mathfrak{u}|=\ell}} \prod_{j\in\mathfrak{u}} \omega\left(\frac{k\cdot z_j}{n}\right)}_{p_{s-1,\ell}(k)} \right. \\ &\qquad \left. + \omega\left(\frac{k\cdot z_s}{n}\right) \underbrace{\sum_{\substack{\mathfrak{u}\subseteq\mathcal{D}_{s-1}\\ |\mathfrak{u}|=\ell-1}} \prod_{j\in\mathfrak{u}} \omega\left(\frac{k\cdot z_j}{n}\right)}_{p_{s-1,\ell-1}(k)} \right) \\ &= e_{s-1}^2 + \frac{1}{n}\sum_{k=0}^{n-1} \omega\left(\frac{k\cdot z_s}{n}\right) \left(\sum_{\ell=1}^{s} \Gamma_\ell\, p_{s-1,\ell-1}(k)\right). \end{aligned}$$

Opnieuw is de essentie van deze berekening een matrix-vector-vermenigvuldiging met een circulante matrix van orde $\varphi(n)$:

$$\begin{aligned} \boldsymbol{e}_s^2 &= e_{s-1}^2 \mathbf{1}_{\varphi(n)\times 1} + \frac{1}{n}\,\boldsymbol{\Omega}_n \left(\sum_{\ell=1}^{s} \Gamma_\ell\, \boldsymbol{p}_{s-1,\ell-1}\right), \\ \boldsymbol{p}_{s,\ell} &= \boldsymbol{p}_{s-1,\ell} + \boldsymbol{\Omega}_n(z_s,:) \mathbin{.*} \boldsymbol{p}_{s-1,\ell-1}. \end{aligned}$$

We bekomen dus een snelle constructie voor orde-afhankelijke gewichten. We geven de stelling in functie van een ruimte met orde $q^\star$ gewichten, $2 \le q^\star \le s$, waarbij $\Gamma_\ell = 0$ voor $\ell > q^\star$.

Stelling 12. *Component-per-component constructie van een rang-1-roosterregel met n punten in s dimensies, met n priem, in een verschuivingsinvariante Hilbertruimte met reproducerende kern, uitgerust met orde-afhankelijke gewichten van orde $q^\star$, kost $O(s(n\log(n) + nq^\star))$ tijd en $O(nq^\star)$ geheugen.*

Voor $q^\star \ll s$ is de complexiteit vergelijkbaar met de $O(sn\log(n))$ complexiteit bij productgewichten.

Het nut van deze snelle algoritmen wordt geïllustreerd in Figuur 3.2, op pagina 76 in het Engelstalige deel, waarin meetresultaten voor vier versies van het component-per-component algoritme getoond worden. De routine *slowrank1* heeft een complexiteit van $O(s^2n^2)$, de routines *rank1* en *spmvrank1* zijn beide implementaties van de $O(sn^2)$ versie en de routine *fastrank1* is het nieuwe algoritme op basis van de circulante matrix met $O(sn\log(n))$. Het is duidelijk dat het nieuwe algoritme steeds te prefereren is. (De afwijkingen in het begin van de grafiek voor *fastrank1* liggen aan de initialisatie van de snelle Fourierroutines.)

4 Constructies met een niet-priem aantal punten

De snelle constructie uit §3 steunt op het feit dat bijna de volledige matrix $\boldsymbol{\Xi}_n$, gedefinieerd in (9), door middel van permutaties in een circulante vorm gebracht kan worden op voorwaarde dat n priem is. Wanneer n niet priem is kunnen we uiteraard ook matrices $\boldsymbol{\Omega}_n$ en $\boldsymbol{\Xi}_n$ definiëren. De rijen van deze matrices komen overeen met de keuzes voor $z \in U_n$, de kolommen komen overeen met de sommeringsindex $k \in \mathbb{Z}_n$ in (3). We bekomen dus een matrix $\boldsymbol{\Xi}_n$ (en gelijkaardige matrix $\boldsymbol{\Omega}_n$) van grootte $\varphi(n) \times n$, gedefinieerd als

$$\boldsymbol{\Xi}_n := \big[k \cdot z\big]_{\substack{z\in U_n \\ k\in\mathbb{Z}_n}},$$

waarbij de vermenigvuldiging modulo n gebeurt.

De iteratieve formules voor het kwadraat van de ergst-mogelijke fout uit §3 blijven uiteraard hetzelfde, enkel de matrix $\boldsymbol{\Omega}_n$ verandert. Om een snelle constructie

voor willekeurige n te bekomen volstaat het dus om een snel algoritme voor matrix-vector-vermenigvuldiging op te stellen voor matrices van de vorm $\boldsymbol{\Xi}_n$. We zullen een dergelijk algoritme bekomen door het uitbuiten van de algebraïsche structuur van de matrix.

4.1 Cyclische groepen en circulante matrices

Door gebruik te maken van enkele klassieke stellingen uit de algebra, kunnen we de groep U_n ontbinden in verschillende cyclische groepen. Uit de hoofdstelling van Abelse groepen besluiten we het volgende.

Lemma 13. *Elke multiplicatieve groep U_n, met $n = n_1 n_2 \dots n_r$ de priemfactorisatie van n en alle n_i relatief priem, kunnen we schrijven als*

$$U_n \simeq \begin{cases} U_{n_1} \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{als } 8 \nmid n, \\ \left(\langle 2^{k-1}-1\rangle_{2^k} \oplus \langle 5\rangle_{2^k}\right) \oplus U_{n_2} \oplus \cdots \oplus U_{n_r} & \text{als } 8 \mid n,\ n_1 = 2^k,\ k \geq 3, \end{cases}$$

waarbij elke subgroep een cyclische multiplicatieve groep is.

Het opsommen van elementen van U_n kunnen we nu doen in functie van de generatoren van de cyclische subgroepen waarna we met behulp van de Chinese reststelling de tupels converteren naar elementen van U_n.

Nemen we bijvoorbeeld $n = p_1 p_2$, een product van twee verschillende priemgetallen, dan kunnen we de vermenigvuldigingstabel schrijven in functie van hun twee generatoren g_1 en g_2 als

$\cdot$	$(g_1^0, \boldsymbol{g_2^{-1}})$	$(g_1^{-1}, \boldsymbol{g_2^{-1}})$	$\cdots$	$(g_1^1, \boldsymbol{g_2^{-1}})$
$(g_1^0, \boldsymbol{g_2})$	$(g_1^0, \boldsymbol{C}_2)$	$(g_1^{-1}, \boldsymbol{C}_2)$	$\cdots$	$(g_1^1, \boldsymbol{C}_2)$
$(g_1^1, \boldsymbol{g_2})$	$(g_1^1, \boldsymbol{C}_2)$	$(g_1^0, \boldsymbol{C}_2)$	$\cdots$	$(g_1^2, \boldsymbol{C}_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(g_1^{-1}, \boldsymbol{g_2})$	$(g_1^{-1}, \boldsymbol{C}_2)$	$(g_1^{-2}, \boldsymbol{C}_2)$	$\cdots$	$(g_1^0, \boldsymbol{C}_2)$

$$\text{met} \quad \boldsymbol{C}_2 = \begin{bmatrix} g_2^0 & g_2^{-1} & \cdots & g_2^1 \\ g_2^1 & g_2^0 & \cdots & g_2^2 \\ \vdots & \vdots & \ddots & \vdots \\ g_2^{-1} & g_2^{-2} & \cdots & g_2^0 \end{bmatrix}.$$

Met $\boldsymbol{C}_2$ bedoelen we de circulante vorm van de vermenigvuldigingstabel van U_{p_2} en $(g_1^k, \boldsymbol{C}_2)$ is de combinatie van g_1^k met elk element in $\boldsymbol{C}_2$. We bekomen dus een blok-circulante matrix (van twee niveaus). De elementen in de uiteindelijke

matrix kunnen we bekomen door gebruik te maken van de Chinese reststelling. Het spreekt voor zich dat we dit ook voor meer dan twee factoren kunnen, waarbij we dan een geneste blok-circulante matrix krijgen met r niveaus voor een getal n met r factoren.

4.2 Opdelen van de index matrix in circulante blokken

De matrix $\boldsymbol{\Xi}_n$ bevat in de horizontale richting meer dan alleen de elementen uit U_n, namelijk alle elementen uit $\mathbb{Z}_n$. We zullen nu gebruik maken van een ontbinding voor $\mathbb{Z}_n$ op basis van de delers van n. (In de volgende stelling hebben we het begrip *nevenklasse* nodig: een nevenklasse van U_n gegeven door d is gedefinieerd als $d\,U_n := \{d \cdot u : u \in U_n\}$.)

Stelling 14. *De unie van alle nevenklassen van U_n, gegeven door de delers van n, vormt een partitie voor $\mathbb{Z}_n$*

$$\begin{aligned}\mathbb{Z}_n &= \bigcup_{d|n} d\,U_n \\ &= \bigcup_{d|n} d\,U_{n/d} \bmod n.\end{aligned}$$

De partitie gegeven door de deler d heeft grootte $|\,d\,U_n| = \varphi(n/d)$ en dus

$$n = \sum_{d|n} |\,d\,U_n| = \sum_{d|n} \varphi(n/d).$$

Met deze partitionering bekomen we een opdeling van de matrix $\boldsymbol{\Xi}_n$ in geneste blok-circulante matrices per deler van n.

Stelling 15. *De matrix*

$$\boldsymbol{\Xi}_n = \Big[k \cdot z\Big]_{\substack{z \in U_n \\ k \in \mathbb{Z}_n}}$$

kunnen we opdelen in verticale partities $\boldsymbol{A}_d$, met een partitie per deler $d^{(1)}, \ldots, d^{(\nu)}$ van n (met ν het aantal delers van n) als

$$\boldsymbol{\Xi}_n \simeq [\boldsymbol{A}_{d^{(1)}} | \boldsymbol{A}_{d^{(2)}} | \cdots | \boldsymbol{A}_{d^{(\nu)}}], \qquad \boldsymbol{A}_d = \Big[k \cdot z\Big]_{\substack{z \in U_n \\ k \in d\,U_{n/d}}}.$$

De grootte van $\boldsymbol{A}_d$ is $\varphi(n) \times \varphi(n/d)$. Deze partities $\boldsymbol{A}_d$ kunnen we dan, elk afzonderlijk*, horizontaal opdelen in t_d identieke vierkante blokken $\boldsymbol{B}_d$ van grootte $\varphi(n/d) \times \varphi(n/d)$, waarbij*

$$\boldsymbol{A}_d \simeq \left[\begin{array}{c} \boldsymbol{B}_d \\ \hline \vdots \\ \hline \boldsymbol{B}_d \end{array}\right] = \mathbf{1}_{t_d \times 1} \otimes \boldsymbol{B}_d, \qquad \boldsymbol{B}_d = \Big[k \cdot z\Big]_{\substack{z \in U_{n/d} \\ k \in d\,U_{n/d}}} = \Big[d \cdot k \cdot z\Big]_{\substack{z \in U_{n/d} \\ k \in U_{n/d}}},$$

met $t_d = \varphi(n)/\varphi(n/d)$. *De blokken* $\boldsymbol{B}_d$ *zijn isomorf met de vermenigvuldigingstabel van* $U_{n/d}$.

Verder kunnen de permutaties op de matrix $\boldsymbol{\Xi}_n$ *zo gekozen worden dat de verticale partities* $\boldsymbol{A}_d$ *de* t_d *identieke geneste blok-circulante voorstelling van de vermenigvuldigingstabel van* $U_{n/d}$ *op een in elkaar geschoven manier bevatten. Stellen we met* $\boldsymbol{C}_{n/d}$ *de geneste blok-circulante vorm voor dan hebben we*

$$\boldsymbol{A}_d = (\otimes_i \boldsymbol{R}_{d,i})\; d\, \boldsymbol{C}_{n/d} \bmod n,$$

met

$$\boldsymbol{R}_{d,i} = \mathbf{1}_{t_{d,i}\times 1} \otimes \boldsymbol{I}_{\varphi(n_i/g_i)},$$

waarbij $d_i = d \bmod n_i$, $g_i = \gcd(d_i, n_i)$, $t_{d,i} = \phi(n_i)/\phi(n_i/g_i)$ *en met* $\boldsymbol{I}_{\varphi(n_i/g_i)}$ *de identiteitsmatrix van grootte* $\varphi(n_i/g_i)$ *(met de* i *indices gaande over de factorisatie van* n *als in Lemma* 13*).*

In het Engelstalige deel werden verschillende illustraties opgenomen om de geneste blok-circulante structuur voor verschillende n te tonen. Zie Figuren 4.1–4.3, vanaf pagina 99, en ook Figuren 4.6–4.8, vanaf pagina 106, waarop expliciet de permutaties aangegeven zijn.

4.3 Een snel matrix-vector algoritme voor willekeurige n

Met behulp van Stelling 15 en twee technische lemma's (die we in deze samenvatting weglaten) bekomen we nu een snel matrix-vector algoritme voor matrices die de structuur van $\boldsymbol{\Xi}_n$ hebben.

Stelling 16. *Een matrix-vector-vermenigvuldiging met een matrix die homomorf is met de matrix* $\boldsymbol{\Xi}_n$ *kan gebeuren in tijd* $O(n\log(n))$ *en met geheugen* $O(n)$.

Het eerste technische lemma nodig voor het bewijs van deze stelling stelt dat een matrix-vector-product met een k keer geneste blok-circulante matrix van orde m kan gebeuren in tijd $O(km\log(m))$. Het tweede technische lemma stelt dat het samenstellen van de verschillende deelresultaten (over alle delers van n) in het matrix-vector-product kan gebeuren in $O(\kappa(n)\, n)$ met $\kappa(n)$ het aantal verschillende priemfactoren in n.

Een extra veronderstelling die we maken om Stelling 16 te bekomen is dat het aantal priemfactoren van n afgeschat kan worden door een constante. Voor de praktische gevallen waarin we deze stelling willen gebruiken, is dit een zeer redelijke veronderstelling omdat voor $n \leq 2^{32}$ we altijd $\kappa(n) \leq 9$ hebben.

Stellingen 15 en 16 zijn de belangrijkste theoretische resultaten in deze tekst. In de praktijk zijn roosterregels met een priem aantal punten echter te prefereren. Enerzijds omdat de implementatie van de snelle constructie veel eenvoudiger en sneller is. Anderzijds omdat het bekend is dat de ergst-mogelijke fout groter wordt

naarmate het aantal priemfactoren in n stijgt, zie [21]. Vanuit dit standpunt is het resultaat van §3 dus veel interessanter.

De structuur die echter door deze stelling is blootgelegd, stelt ons wel in staat om "roosterrijen" te construeren. Het belangrijkste nadeel van klassieke roosterregels is namelijk dat we steeds alle punten tegelijk moeten gebruiken. Indien de benadering niet goed genoeg is, dan moeten we een grotere roosterregel kiezen en helemaal opnieuw beginnen. Het is interessanter om het vorige resultaat te behouden en nieuwe punten toe te voegen totdat we een resultaat bekomen dat we nauwkeurig genoeg vinden. Dit wordt mogelijk gemaakt door het gebruik van roosterrijen.

5 Roosterrijen

De structuur van $\boldsymbol{\Xi}_n$ die we hebben ontrafeld in §4 neemt een heel bijzondere vorm aan indien n een macht is van een priemgetal (we behandelen dit in §5.2). Ook roosterregels met n een macht van een (priem)getal hebben een bijzondere eigenschap: ze zijn namelijk genest. Stel dat L_m de puntenverzameling van een roosterregel is van grootte p^m

$$L_m = L_m(\boldsymbol{z}) := \left\{ \left\{ \frac{k\boldsymbol{z}}{p^m} \right\} : 0 \le k \le p^m - 1 \right\}, \quad m = 0, 1, \ldots,$$

voor een vaste $\boldsymbol{z} \in \mathbb{Z}^s$, dan hebben we een rij van geneste roosterregels, m.a.w. een *roosterrij* waarvoor

$$L_0 \subset L_1 \subset \cdots \subset L_m \subset L_{m+1} \subset \cdots .$$

5.1 Een algoritme voor goede roosterrijen

We stellen nu een algoritme voor om een rij van roosterregels op te stellen met een aantal punten p^m en die "goed" zijn voor $m_1 \le m \le m_2$. We definiëren hiervoor een criterium dat de maximale afwijking meet voor de ergst-mogelijke fout ten opzichte van de voor het component-per-component algoritme optimale ergst-mogelijke fout:

$$X_{m_1,m_2,s}(\boldsymbol{z}) := \max_{m_1 \le m \le m_2} \frac{e_{p^m,s}(\boldsymbol{z})}{e_{p^m,s}(\boldsymbol{z}^{(m)})}.$$

Doordat we weten dat het component-per-component algoritme in gewogen Korobovruimten en Sobolevruimten optimale roosterregels kan construeren, weten we ook dat, als we $X_{m_1,m_2,s}(\boldsymbol{z})$ kunnen afschatten door een (kleine) constante, we een optimale roosterrij hebben voor de gegeven vector $\boldsymbol{z}$.

Algoritme 2 geeft schematisch weer hoe dit nieuwe algoritme in zijn werk gaat; het algoritme is in feite een component-per-component constructie op basis van het

nieuwe criterium $X_{m_1,m_2,s}$. Uit numerieke resultaten blijkt dat we inderdaad optimale roosterrijen construeren. Voor de testen die we uitvoerden in verschillende functieruimten, was het nieuwe criterium steeds kleiner dan $1,60$.

Algoritme 2 Component-per-component constructie voor roosterrijen

for $s = 1$ **to** $s_{\max}$ **do**
  for all $z_s \in U_{p^{m_2}}$ **do**
    $$X_{m_1,m_2,s}(z_s) = \max_{m_1 \le m \le m_2} \frac{e_{p^m,s}(\boldsymbol{z})}{e_{p^m,s}(\boldsymbol{z}^{(m)})}$$
  end for
  $$z_s = \operatorname*{argmin}_{z \in U_{p^{m_2}}} X_{m_1,m_2,s}(z)$$
end for

5.2 De structuur van $\boldsymbol{\Omega}_{p^m}$

Wanneer n een macht van een priemgetal is, $n = p^m$, dan heeft de matrix $\boldsymbol{\Xi}_{p^m}$ een zeer specifieke vorm. Dit komt namelijk omdat de groep U_{p^m} op zich reeds cyclisch is (zie Lemma 13, met een speciaal geval voor $p = 2$). We hebben dus maar één generator g nodig. Deze generator kan zelfs zo gekozen worden dat hij bruikbaar is als generator voor alle groepen U_{p^ℓ} (zie bijvoorbeeld [1, §24.3.4]). Indien r een generator is voor U_p, dan is g een generator voor U_{p^ℓ}, $\ell \ge 2$, als

$$g = \begin{cases} r + p & \text{als } r^{p-1} \equiv 1 \pmod{p^2}, \\ r & \text{anders.} \end{cases}$$

We moeten met andere woorden enkel maar een generator zoeken voor U_{p^2} (omdat $r + p \equiv r \pmod{p}$).

De exacte stellingen betreffende de structuur van $\boldsymbol{\Xi}_{p^m}$ laten we hier achterwege, maar we schetsen het idee. Vereenvoudigd voorgesteld bekomen we de volgende structuur na de permutaties op basis van g:

$$\boldsymbol{\Xi}_{p^m}^{\langle g \rangle} = \left[\; \mathbf{1}_{p^0} \otimes \boldsymbol{B}_{p^m}^{\langle g \rangle} \;\middle|\; \mathbf{1}_{p^1} \otimes \boldsymbol{B}_{p^{m-1}}^{\langle g \rangle} \;\middle|\; \cdots \;\middle|\; \mathbf{1}_{\varphi(p^m)} \otimes \boldsymbol{B}_{p^0}^{\langle g \rangle} \; \right].$$

Met andere woorden, de matrix $\boldsymbol{\Xi}_{p^m}$ is een aaneenschakeling van onder elkaar geplaatste identieke blokken. In het eerste vak (voor $d = 1$) bevindt zich maar één blok, in het tweede vak (voor $d = p$) bevinden zich p identieke blokken op elkaar, enzovoorts, tot we in het laatste vak (voor $d = p^m$) een opeenstapeling hebben van één enkele waarde (m.a.w. een constante kolom). Wanneer $p = 2$ hebben we iets gelijkaardig, met de complicatie dat U_{2^m} voor $m > 2$ geen cyclische groep is en we moeten werken met een pseudogenerator ($g = 5$) en een blok-circulante matrix als basis-structuur.

Door deze specifieke structuur voor priemmachten kunnen we gelijktijdig de waarden voor $e_{p^m,s}(z_s)$ uit het nieuwe criterium $X_{m_1,m_2,s}$ berekenen voor alle $1 \leq m \leq m_2$ tegelijk en voor de kost van $e_{p^{m_2},s}(z_s)$ alleen, m.a.w. met kost $O(sn\log(n))$ waarbij $n = p^{m_2}$. Het berekenen van de optimale ergst-mogelijke fouten volgens het component-per-component algoritme voor alle $m_1 \leq m \leq m_2$ die we nodig hebben voor het berekenen van X, kunnen we uiteraard ook met het snelle algoritme doen en kost $O(s(n\log(n))^2)$ waarbij $n = p^{m_2}$. We bekomen dus het volgende resultaat.

Stelling 17. *Constructie van roosterrijen met Algoritme* 2 *voor een maximum van* $n = p^{m_2}$ *punten, met* p *priem, kost* $O(sn\log(n))$ *met* $O(n)$ *geheugen voor productgewichten en* $O(qn)$ *geheugen voor orde-afhankelijke gewichten van orde* q*. De totale constructiekost met inbegrip van de voorbereidende berekeningen van de optimale ergst-mogelijke fouten voor* $m_1 \leq m \leq m_2$ *bedraagt* $O(sn(\log(n))^2)$*.*

5.3 Het gebruik van roosterrijen

Indien we een roosterrij geconstrueerd met Algoritme 2 gebruiken voor de machten van p tussen p^{m_1} en p^{m_2} dan garandeert de constructie ons een a posteriori foutengrens die bijvoorbeeld een constante factor $1,60$ van de optimale convergentie verwijderd is. We kunnen een dergelijke roosterrij echter ook punt voor punt gebruiken. Hiervoor gebruiken we een permutatie ϕ van $\mathbb{Z}_n$ naar $\mathbb{Z}_n$ waarbij het $k$de punt nu gegeven is door

$$\boldsymbol{x}_k = \left\{ \frac{\phi(k)\,\boldsymbol{z}}{n} \right\}.$$

Door een goede permutatie te kiezen, zorgen we ervoor dat de puntenrij zo goed mogelijk uniform verdeeld is voor een stijgend aantal punten. Goede permutaties zijn bekend uit de literatuur omtrent (t,s)-rijen (een andere vorm van puntenverzamelingen met kleine discrepantie). We kunnen voor $\phi(k)/n$ bijvoorbeeld de radix-inversie functie ϕ_p in basis p gebruiken, of de Graycode-variant hiervan. Dit zijn beiden eendimensionale rijen met kleine discrepantie.

Ook voor roosterregels die niet met Algoritme 2 geconstrueerd werden, kunnen we de roosterregel punt voor punt genereren. De garantie dat we goede geneste roosterregels tegenkomen is dan echter wel verdwenen; we weten enkel dat de uiteindelijke volledige roosterregel goed is. Door gebruik te maken van een stochastische foutenschatter (die werkt op basis van een klein aantal randomisaties van de kubatuurregel), bekomen we wel een goed idee van de performantie. Zelfs wanneer het aantal punten van de roosterregel geen macht van een (priem)getal is, kunnen we de regel op deze manier punt voor punt gebruiken. We gebruiken dan een eenvoudige verwerpingsmethode die een eendimensionale rij met kleine discrepantie in bijvoorbeeld basis 2 gebruikt. De technische details laten we hier achterwege.

6 Constructies van veeltermroosterregels

In plaats van met een veld of ring van gehele getallen te werken, zoals in §3 en §4, is het ook mogelijk om roosters over veeltermen te beschouwen. We vervangen hiervoor eenvoudigweg alle $\boldsymbol{z} \in (U_n)^s$ door een veeltermequivalent $\boldsymbol{z}(x) \in ((\mathbb{F}_q[x]/f(x))^\times)^s$ met $\mathbb{F}_q$ het Galoisveld van orde q. Hierbij neemt $f(x) \in \mathbb{F}_q[x]$ de plaats in van de scalaire modulus n. Voor de eenvoud beschouwen we hier enkel maar Galoisvelden van priemgrootte. In dit geval is $\mathbb{F}_p$ equivalent met $\mathbb{Z}_p$.

De notatie voor veeltermroosterregels is een heel stuk complexer dan voor normale roosterregels. Echter, de theorie van de normale roosterregels kan bijna rechtstreeks vertaald worden naar het veeltermequivalent. De complexiteit van de bewijzen neemt echter toe omdat er bij veeltermroosterregels meer mogelijkheden zijn dan voor gewone roosterregels. Belangrijk om op te merken is dat de puntenverzameling van een veeltermroosterregel geen roosterregel is. De roosterstructuur van de veeltermen wordt namelijk niet behouden wanneer ze vertaald worden naar de eenheidskubus $[0,1)^s$. De puntenverzameling die bekomen wordt, is een (t,m,s)-net, dit is een ander type van puntenverzameling met lage discrepantie.

6.1 Veeltermroosterregels

We geven nu een vereenvoudigde definitie van veeltermroosterregels voor het geval waarin we veeltermen beschouwen over $\mathbb{F}_p$ met p priem (zie [24] voor de volledige definitie).

Definitie 18. Een *veeltermroosterregel in basis p* met modulus $f \in \mathbb{F}_p[x]$ en graad $\deg(f) = m$ heeft punten

$$\boldsymbol{x}_k = v_m\left(\frac{k(x)\cdot \boldsymbol{z}(x) \bmod f(x)}{f(x)}\right) \qquad \text{voor } k \in \mathbb{F}_p[x]/f.$$

De genererende vector $\boldsymbol{z}$ heeft componenten $z_j \in (\mathbb{F}_p[x]/f)^\times$ waarbij v_m een functie is die componentsgewijs de formele Laurentreeksen $\mathbb{F}_p((x^{-1}))$ vertaalt naar radix-p rationale getallen in $[0,1)$. De $j$de component van het $k$de punt is dan gedefinieerd als

$$x_j^{(k)} = v_m\left(\frac{k(x)\cdot z_j(x) \bmod f(x)}{f(x)}\right) = v_m\left(\sum_{\ell=1}^{\infty} w_\ell\, x^{-\ell}\right) = \sum_{\ell=1}^{m} w_\ell\, p^{-\ell},$$

met alle $w_\ell \in \mathbb{F}_p$.

Dick, Kuo, Pillichshammer en Sloan [24] toonden aan dat het ook voor deze veeltermroosterregels mogelijk is om het component-per-component algoritme te gebruiken voor de constructie. Hiervoor werd er gesteund op onderliggende theorie over digitale functieruimten door Dick en Pillichshammer [26].

Belangrijk voor de snelle constructie zijn reproducerende kernen die digitaal verschuivingsinvariant zijn. Dit wil zeggen dat voor $\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Delta} \in [0,1)^s$

$$K(\boldsymbol{x} \oplus_p \boldsymbol{\Delta}, \boldsymbol{y} \oplus_p \boldsymbol{\Delta}) = K(\boldsymbol{x}, \boldsymbol{y}),$$

met $\oplus_p$ de componentsgewijze optelling modulo p van de afzonderlijke getallen in een radix-p voorstelling (m.a.w. een veeltermoptelling over $\mathbb{F}_p$ met als coëfficiënten de radix-p expansie van x en y). Net als voor de gewone verschuivingsinvariante kern, zie (2), kunnen we de kern ook hier in functie van één parameter schrijven:

$$K(\boldsymbol{x}, \boldsymbol{y}) = K(\boldsymbol{x} \ominus_p \boldsymbol{y}, \boldsymbol{0}) =: K(\boldsymbol{x} \ominus_p \boldsymbol{y}).$$

De elementaire functieruimte die hier aan voldoet, is een functieruimte gebaseerd op Walshfuncties. In basis 2 kunnen we deze functieruimte bekijken als een functieruimte met de klassieke Haarwavelets als basisfuncties, maar zo genormaliseerd dat de functieruimte een reproducerende kern heeft. De ω-functie in basis 2 voor het berekenen van de ergst-mogelijke fout (6) en (7) is voor deze functieruimte gegeven door

$$\omega(x) = 12 \left(\frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1} \right).$$

Ook de Sobolevruimte zonder ankerpunt, Definitie 8, kan digitaal verschuivingsinvariant gemaakt worden. De ω-functie in basis 2 is hiervoor

$$\omega(x) = \frac{1}{6} - 2^{\lfloor \log_2(x) \rfloor - 1}.$$

Met behulp van deze ω-functies kunnen we opnieuw op zoek gaan naar een algoritme voor snelle constructie.

6.2 Snelle constructie van veeltermroosterregels

Indien de veelterm modulus f in Definitie 18 gekozen wordt als een irreduceerbare veelterm over $\mathbb{Z}_p$, dan is $\mathbb{Z}_p[x]/f$ een veld en is $(\mathbb{Z}_p[x]/f)^\times = \mathbb{Z}_p[x]/f \setminus \{0\}$. Een irreduceerbare veelterm is namelijk het letterlijke equivalent van een priemmodulus.

We kunnen nu een matrix van grootte $(p^m - 1) \times p^m$ definiëren

$$\boldsymbol{\Omega}_{p^m} = \left[\omega\left(v_m \left(\frac{k(x) \cdot z(x)}{f(x)} \right) \right) \right]_{\substack{z(x) \in (\mathbb{Z}_p[x]/f)^\times \\ k(x) \in\ \mathbb{Z}_p[x]/f}}.$$

Zoals in het scalaire geval kunnen we ook hier een generator vinden voor de multiplicatieve groep $(\mathbb{Z}_p[x]/f)^\times$ en kunnen we de matrix $\boldsymbol{\Omega}_{p^m}$ in dezelfde circulante vorm brengen als in §3. We bekomen dus een snelle constructie op exact dezelfde manier als voor de gewone roosterregels.

Stelling 19. *Component-per-component constructie van een veeltermroosterregel in basis p met p^m punten in s dimensies in een digitaal verschuivingsinvariante Hilbertruimte met reproducerende kern kost $O(sn\log(n))$ en $O(n)$ geheugen voor productgewichten en $O(qn)$ geheugen voor orde-afhankelijke gewichten van orde q.*

7 Stempelregels

Stempelregels zijn een eenvoudige manier om roosterregels met hogere rang te bekomen. We starten hiervoor met een rang-1-roosterregel met n punten over $[0,1)^s$ en verkleinen deze zo dat we een roosterregel over $[0,\nu^{-1})^s$ bekomen. De volledige eenheidskubus vullen we nu met ν^s kopieën hiervan. Indien ν en n relatief priem zijn, dan bekomen we een roosterregel met $n\nu^s$ punten en van volle rang, namelijk rang s.

Het is duidelijk dat deze procedure enkel mogelijk is voor kleine ν en s omdat we een exponentieel aantal punten $n\nu^s$ bekomen. Voor hogere dimensies kunnen we er echter ook voor kiezen om niet over alle assen te kopiëren, maar enkel maar over de eerste r.

Definitie 20. Een ν^r-stempelregel van een rang-1-roosterregel met genererende vector $\boldsymbol{z}$ en met n punten heeft de vorm

$$Q(f) = \frac{1}{\nu^r n} \sum_{\boldsymbol{v}\in\mathbb{Z}_\nu^r} \sum_{k=0}^{n-1} f\left(\left\{\frac{k\boldsymbol{z}/n}{\nu} + \frac{(v_1,\dots,v_r,0,\dots,0)^\top}{\nu}\right\}\right),$$

waarbij $\gcd(n,\nu)=1$, $1<r\le s$ en $\nu\ge 2$.

7.1 Snelle constructie van stempelregels

Joe en Sloan [55] tonen aan dat het berekenen van de ergst-mogelijke fout voor een stempelregel in een Korobovruimte kan berekend worden met dezelfde kost als voor de rang-1-basisregel waarop de stempelregel gebaseerd is. In [63] wordt deze stelling lichtjes uitgebreid voor de gewogen Korobovruimte en een gewogen verschuivingsinvariante Sobolevruimte (beiden met productgewichten). We geven hier nog een eenvoudige generalisatie voor eender welke verschuivingsinvariante functieruimte.

Stelling 21. *Gegeven een verschuivingsinvariante Hilbertruimte met reproducerende kern*

$$K(\boldsymbol{x}) = \prod_{j=1}^{s} \left(1+\gamma_j\,\omega(x_j)\right), \qquad \textit{waarbij we veronderstellen dat } \int_0^1 \omega(x)\,\mathrm{d}x = 0,$$

en waarvoor de Fourierexpansie van ω gegeven is als

$$\omega(x) = \sum_{h\in\mathbb{Z}\setminus\{0\}} \hat{k}_h \exp(2\pi\mathrm{i}\, hx).$$

Dan kunnen we de ergst-mogelijke fout van een ν^r-stempelregel in deze ruimte berekenen als de ergst-mogelijke fout voor de rang-1-roosterregel waarop de stempelregel gebaseerd is, maar met een aangepaste ω-functie voor de eerste r dimensies, gegeven als

$$\overline{\omega}_\nu(x) = \sum_{h\in\nu\mathbb{Z}\setminus\{0\}} \hat{k}_h \exp(2\pi\mathrm{i}\, hx) = \sum_{h\in\mathbb{Z}\setminus\{0\}} \hat{k}_{h\nu} \exp(2\pi\mathrm{i}\, h\nu x).$$

Opnieuw bekomen we een snelle constructie doordat de formule voor de ergst-mogelijke fout van de correcte vorm is. Het algoritme werkt dan eerst met een matrix gebaseerd op de functie $\overline{\omega}_\nu$ en na r iteratiestappen wordt er overgeschakeld op de normale matrix gebaseerd op de functie ω. Meer nog, voor de eerder besproken Korobov- en Sobolevruimten is er zelfs geen aanpassing nodig aan de implementatie van het algoritme; de wijziging kan volledig worden verwerkt door voor- en naverwerking.

Stelling 22. *Component-per-component constructie van een stempelregel gebaseerd op een rang-1-roosterregel met n punten (niet noodzakelijk priem) in s dimensies in een verschuivingsinvariante Hilbertruimte met reproducerende kern en uitgerust met productgewichten kost $O(sn\log(n))$ tijd en $O(n)$ geheugen.*
Onder gelijkaardige voorwaarden kunnen we ook roosterrijen genereren van stempelregels.

7.2 Veeltermstempelregels

Zoals reeds vermeld, verloopt de theorie van veeltermroosterregels veelal parallel met die van de gewone roosterregels. Als theoretische oefening hebben we daarom ook veeltermstempelregels gedefinieerd. De definitie loopt gelijk met Definitie 20, met de aanpassingen dat de basisregel een rang-1-veeltermroosterregel is en dat ν nu een veelterm is.

Definitie 23. Gegeven een rang-1-veeltermroosterregel over $\mathbb{F}_p[x]/f$ in basis p en met modulus f, $\deg(f) = m_1$, dan is de *$\nu(x)^r$-stempelregel*, met $\deg(\nu) = m_2$, gedefinieerd als

$$Q(f) = \frac{1}{p^{m_1}(p^{m_2})^r} \sum_{\boldsymbol{v}(x)\in(\mathbb{F}_p[x]/\nu)^r} \sum_{k(x)\in\mathbb{F}_p[x]/f} f\left(v_{m_1+m_2}\left(\frac{k(x)\boldsymbol{z}(x)}{f(x)} + \frac{(v_1(x),\ldots,v_r(x),0,\ldots,0)^\top}{\nu(x)}\right)\right),$$

waarbij $\gcd(f(x),\nu(x)) = 1$, $1 < r \leq s$ en $\deg(\nu) \geq 1$.

We kunnen hier een gelijkaardig resultaat als Stelling 21 bekomen.

Stelling 24. *Gegeven een digitaal verschuivingsinvariante Hilbertruimte met reproducerende kern*

$$K(\boldsymbol{t}) = \prod_{j=1}^{s}(1+\gamma_j\,\omega(t_j)), \qquad \textit{waarbij we veronderstellen dat } \int_0^1 \omega(t)\,\mathrm{d}t = 0,$$

en waarvoor de Walshexpansie van ω gegeven is als

$$\omega(t) = \sum_{h=1}^{\infty} \hat{k}_p(h)\ \mathrm{wal}_{p,h}(t).$$

Dan kunnen we de ergst-mogelijke fout van een $\nu(x)^r$-stempel regel in deze ruimte berekenen als de ergst-mogelijke fout voor de rang-1-veeltermroosterregel waarop de stempelregel gebaseerd is, maar met een aangepaste ω-functie voor de eerste r dimensies.

Uiteraard is het ook hier weer mogelijk om een snelle constructie te bekomen.

8 Toepassingen

Verschillende numerieke testen werden uitgevoerd met roosterregels die geconstrueerd werden met de technieken die we beschreven hebben. Doordat de onderliggende theorie voor component-per-component constructie voorspelt dat we optimale roosterregels bekomen, verwachten we natuurlijk steeds goede resultaten.

In [15] berekenden we een roosterrij in een Sobolevruimte zonder ankerpunt met orde 2 gewichten. Door de specifieke vorm van de formule voor de ergst-mogelijke fout bij orde-afhankelijke gewichten (7) maakt het niet uit hoe de twee gewichten, Γ_1 en Γ_2, gekozen worden. Hierdoor krijgen we in feite een roosterregel die universeel is voor alle orde 2 problemen.

Verschillende grafieken, Figuren 8.1–8.4 in het Engelstalige deel, zie pagina 171 en verder, tonen inderdaad aan dat deze roosterrij zeer goed werkt. Er werden twee problemen uit de financiële wiskunde en één probleem uit de statistiek getest.

9 Resultaten

De volgende lijst van onderzoeksresultaten kwam in deze tekst aan bod.

(i) We bekwamen een snelle constructie voor roosterregels met een priem aantal punten [83].

(ii) We bekwamen een snelle constructie voor roosterregels met een niet-priem aantal punten [85]. De kost stijgt lichtjes voor een toenemend aantal priemfactoren.

(iii) We bekwamen een snelle constructie van goede roosterrijen [15].

(iv) Goede roosterrijen kunnen punt per punt gebruikt worden [15].

(v) Eender welke roosterregel kunnen we in feite op een betrouwbare manier punt per punt gebruiken, zelfs indien het aantal punten geen macht van een priemgetal is [17]. De effectiviteit kunnen we controleren door gebruik te maken van een stochastische foutenschatter.

(vi) We bekwamen een snelle constructie van veeltermroosterregels modulo een irreduceerbare veelterm [84].

(vii) We bekwamen een snelle constructie van stempelregels.

(viii) Numerieke testen met een roosterrij voor orde 2 gewichten toonden aan dat deze roosterrij zeer goed presteert [15].

(ix) De (Engelstalige) tekst bevat voorbeeldcode voor bijna alle snelle constructies die besproken werden. We volgden hierbij de principes van Trefethen [106]: hoog-niveau code, niet noodzakelijk eenvoudig leesbaar, maar wel passend op ongeveer één pagina, zodat ze makkelijk te bestuderen is; en snel genoeg om mee te kunnen experimenteren.

Bibliografie

Zie de Engelstalige bibliografie op pagina 181.

Curriculum vitae

- 1994-1998: Industrieel ingenieur elektriciteit elektronica, K.H.K. Kempen.
- 1999-2002: Burgerlijk ingenieur computerwetenschappen, K.U.Leuven.

Publicatielijst

- D. Nuyens and R. Cools. Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. Report TW392, Dept. of Computer Science, K.U.Leuven, May 2004.
- D. Nuyens. Fast construction of a good lattice rule / About the cover. *Notices Amer. Math. Soc.*, 52(11):1329 + cover, Dec. 2005.
- R. Cools and D. Nuyens. Structured matrices as a blueprint for integration lattices. In T. E. Simos, G. Psihoyios and Ch. Tsitouras, editors *ICNAAM 2005 – International Conference on Numerical Analysis and Applied Mathematics 2005*, pages 12–15. Wiley-VCH Verlag, ISBN 3-527-40652-2, 2005.
- 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(2):903–920, 2006.
- R. Cools and D. Nuyens. The role of structured matrices for the construction of integration lattices. *JNAIAM J. Numer. Anal. Ind. Appl. Math.*, 1(3):257–272, 2006.
- 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, ISBN 3-540-25541-9, 2006.
- D. Nuyens and R. Cools. Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points. *J. Complexity*, 22(1):4–28, 2006.
- R. Cools, F. Y. Kuo, and D. Nuyens. Constructing embedded lattice rules for multivariate integration. *SIAM J. Sci. Comput.*, 28(6):2162–2188, 2006.