Multilevel Rank-1 Lattice Rules for Infinite-dimensional Integration Problems

Multilevel-Rang-1-Gitterformeln für unendlichdimensionale Integrationsprobleme
Diplomarbeit von Sebastian Mayer
August 2011

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Mathematik
Arbeitsgruppe Stochastik

Multilevel Rank-1 Lattice Rules for Infinite-dimensional Integration Problems
Multilevel-Rang-1-Gitterformeln für unendlichdimensionale Integrationsprobleme

Vorgelegte Diplomarbeit von Sebastian Mayer

1. Gutachten: Prof. Dr. Klaus Ritter, TU Kaiserslautern
2. Gutachten: Prof. Dr. Michael Kohler, TU Darmstadt

Tag der Einreichung:

Erklärung zur Diplomarbeit

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den

(Sebastian Mayer)

Notations and Conventions

- $\mathbb{N}_0 := \{0, 1, 2, 3, ...\}$; $\mathbb{N} := \mathbb{N}_0 \setminus \{0\}$
- For $A \subseteq \mathbb{N}$ we denote by $P_0(A)$ the set of all finite subsets of A.
- For $a, b \in \mathbb{N}$ with $a < b$ we write $a : b$ for the set of numbers $\{a, ..., b\}$.
- By bold letters $\mathbf{a}, \mathbf{b}, \mathbf{c}, ...$ we usually denote elements in $\mathbb{R}^{\mathscr{I}}$ where $\mathscr{I} \subseteq \mathbb{N}$ follows from the context (most of the time $\mathscr{I} = \mathbb{N}$). Especially when a vetor $\mathbf{x}$ occurs in context of algorithmic issues we use additionally the notation $\mathbf{x}[i] = x_i$ to denote the i-the element.
- Let $\mathbf{x} \in \mathbb{R}^{\mathscr{I}}$ with $\mathscr{I} \subseteq \mathbb{N}$. For $\nu \subseteq \mathscr{I}$ we denote by $\mathbf{x}_\nu$ the $|\nu|$-dimensional subvector of $\mathbf{x}$ whose components have an index in ν. Moreover we use the shorthand $\mathbf{x}_{-\nu} = \mathbf{x}_{\mathscr{I} \setminus \nu}$.
- Landau symbols: Let $f, g : \mathbb{R} \to \mathbb{R}$. Then we write
  - $f = O(g)$ if there exists $c > 0$ and $x_0 \in \mathbb{R}$ such that $|f(x)| \leq c|g(x)|$ for all $x > x_0$.
  - $f = \Omega(g)$ if there exists $c > 0$ and $x_0 \in \mathbb{R}$ such that $|f(x)| \geq c|g(x)|$ for all $x > x_0$.
  - $f = \Theta(g)$ if both $f = O(g)$ and $f = \Omega(g)$.
- Alternatively, we use the following notations for asymptotic behaviour:
  - $f \preccurlyeq g$ iff $f = O(g)$.
  - $f \succcurlyeq g$ iff $f = \Omega(g)$.
  - $f \asymp g$ iff $f = \Theta(g)$.
- Let $\mathbf{x} \in \mathbb{R}^n$. We use the notation $\{\mathbf{x}\}$ to indicate that we take componentwise the fractional part, that is

$$\{\mathbf{x}\}_j = x_j - \lfloor x_j \rfloor$$

Contents

1 Introduction

In this thesis we study *infinite-dimensional integration problems* over $\mathbb{R}^{\mathbb{N}}$ where one seeks to determine an integral of the form

$$I(f) = \int_{\mathfrak{X}} f(\mathbf{x})\mu(\mathrm{d}\mathbf{x})$$

for a product probability measure μ over $\mathfrak{X} \subseteq \mathbb{R}^{\mathbb{N}}$ and the integrand f originating from some class F of functions on $\mathfrak{X}$. The practical application which motivates our examinations arises in Mathematical Finance. In the Black-Scholes model we are able to describe the possible price histories (paths) of an asset S on a time horizon $[0,T]$ in terms of i.i.d. standard normally distributed random variables $X_0, X_1, X_2, \ldots$. Thereby X_0 determines the asset price at time T, X_0 and X_1 determine the price at time $T/2$, X_0, X_1, and X_2 at $T/4$, X_0, X_1, and X_3 at $3/4T$, and so on. Furthermore, payoffs of option contracts with underlying S can then be described in terms of functions $f(X_0, X_1, \ldots)$ and the option price is modelled by the expectation $\mathrm{E}[f(X_0, X_1, \ldots)]$. We obtain a formulation as infinite-dimensional integration problem if we let $\mathfrak{X} = \mathbb{R}^{\mathbb{N}}$, $\mu = \otimes_{j\in\mathbb{N}} N(0,1)$, and F be the class of payoffs that are of interest.

In general, the problem instances we address within this thesis share the property that there are no closed-form solutions available to determine the integral, giving rise to the need for numerical methods which at least approximate $I(f)$ for given $f \in F$. For this purpose we consider *quadrature rules* which approximate $I(f)$ by a weighted sum

$$\sum_{i=1}^{n} w_i f(\mathbf{x}^{(i)}), \quad (w_1, \ldots, w_n) \in \mathbb{R}^n$$

over the integrand f evaluated at the points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)} \in \mathfrak{X}$. However, a feasible quadrature rule will generally be able to process only a finite number of components of each integration point $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$. For the considered problem instances we find a remedy through the occurring integrands being anchored. This means that there is an element $\mathbf{a} \in \mathfrak{X}$ such that any integrand $f \in F$ can be evaluated in the components of $\mathbf{a}$ without the need for computation. In consequence, we get a feasible quadrature rule if the points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ are chosen such that they differ only in the first d components from $\mathbf{a}$. Of course, we actually approximate the integral $I(\Psi_{1:d,\mathbf{a}} f)$ then, with

$$(\Psi_{1:d,\mathbf{a}} f)(\mathbf{x}) = f(\mathbf{x}_{1:d}, \mathbf{a}_{-1:d}).$$

The particular quadrature rules which we focus on in this thesis are *multilevel rank-1 lattice rules*. These comprise two different concepts. The first is the *multilevel idea*. One chooses dimensions $d_1 < \ldots < d_L$ and writes $I(\Psi_{1:d_L,\mathbf{a}} f)$ as the telescope sum

$$I(\Psi_{1:d_L,\mathbf{a}} f) = I(\Psi_{1:d_1,\mathbf{a}} f) + \sum_{l=2}^{L} I(\Psi_{1:d_l,\mathbf{a}} f - \Psi_{1:d_{l-1},\mathbf{a}} f).$$

Then for each integral appearing on the right hand side, we apply an extra quadrature rule. Experience shows that it is often easier to approximate $I(\Psi_{1:d_L,\mathbf{a}}f)$ to some precision in this way than by using only one quadrature rule which we apply directly to $\Psi_{1:d_L,\mathbf{a}}f$.

The second concept which is involved in multilevel rank-1 lattice rules is that each quadrature rule takes deterministic integration points that form a *rank-1 lattice*. Such lattices are constructed on some multidimensional unit cube $[0,1]^d$ in the first instance, always consisting of a prime number of points. Then they may be transformed to the domain given by the problem specification. For instance, regarding option pricing we would transform the lattices to the domain $\mathbb{R}^d$.

Illustrating a rank-1 lattice is the easiest on the 2-dimensional unit square. For this purpose, fix a prime n. Initially we reside in the origin, then cover a lattice specific distance in a lattice specific direction to reach the next point, and repeat this until we reach the origin again, which will be after n times. If we thereby hit the boundary of the unit square, say in the point $\mathbf{y}$, we enter the square again in a boundary point on the opposite side. We obtain this point either by reflection over the horizontal line through the point $(\frac{1}{2},\frac{1}{2})$ if $\mathbf{y}$ hits the bottom or top boundary, or by reflection over the corresponding vertical line if $\mathbf{y}$ hits the right and left boundary, respectively. For a graphical illustration, see the left picture in Figure 1.1.

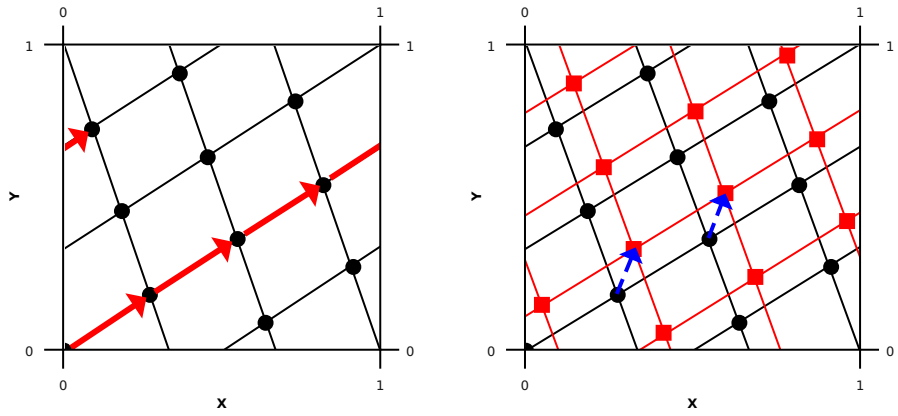

Figure 1.1: Rank-1 lattice and shifted rank-1 lattice with $n = 11$ points.

When we are given a rank-1 lattice we may additionally apply a shift as illustrated in the right picture in Figure 1.1. Beside the specific distance and direction, the shift turns out to be a crucial quantity to affect the quality of a rank-1 lattice for numerical integration.

Exploring the potential of multilevel rank-1 lattice rules, we structure the thesis into two parts. In a first theoretical part we seek to identify instances of the infinite-dimensional integration problem for which we can prove multilevel rank-1 lattice rules to perform well. More specifically, we assume a *worst-case setting*, measuring the error $e(Q,F)$ and $\mathrm{cost}(Q,F)$ of a quadrature rule Q with regard to F in terms of the greatest error and largest cost over all $f \in F$, respectively.

In this setting we use a cost model reflecting the fact that we are primarily interested in understanding how well a given quadrature rules utilizes the available information.

This information is given in form of f evaluated at the integration points. Hence the model measures computational cost only in terms of the sum over all cost these function evaluations produce. In turn, we assume for all $f \in F$ that computing $f(\mathbf{x})$ has cost increasing linearly in the maximal dimension in which $\mathbf{x}$ differs from the anchor $\mathbf{a}$.

Then, based on the worst-case setting we present a result by Niu et al.[NHMR10]. Considering integration with respect to the uniform distribution over $[0,1]^{\mathbb{N}}$ these authors show, for F being the unit ball of a *weighted Sobolev space*, that the *n-th minimal error*

$$e_N(F) = \inf\{e(Q,F) : Q \text{ deterministic quadrature rule with } \operatorname{cost}(Q,F) \leq N\}$$

satisfies the asymptotic bound

$$e_N(F) = \Theta(N^{-1})$$

for properly chosen weights. This matching bound is established by means of multilevel rank-1 lattice rules, each involved rank-1 lattice rule using an appropriate deterministic shift. Consequently, multilevel rank-1 lattice rules yield optimal algorithms for the specific problem setting. However, we obtain the proper shifts only by an existential statement and it is still an open problem whether it is possible to construct the shifts explicitly.

In the practical part we focus again on the application in Mathematical Finance we mentioned in the beginning. Unfortunately, it is not possible to carry over the results given in the theoretical part to payoffs studied for option pricing in the Black-Scholes model. Further there are no theoretical results yet on the efficiency of multilevel rank-1 lattice rules for path dependent option pricing. Hence, for the time being, we are restricted to numerical experiments.

We content ourself with running experiments for the *Asian call option* which is defined by the discounted payoff

$$\varphi_{AC}(w) = e^{-rT} \max\left(\frac{1}{T}\int_0^T w(t)\mathrm{d}t - K, 0\right),$$

where w denotes a specific price history of the underlying asset. As we have no indication how to choose shifts, it is not reasonable to focus on deterministic multilevel rank-1 quadrature rules here. Instead we randomize the shifts, leading to randomized algorithms which we refer to as *multilevel shifted rank-1 lattice rules*.

Now, by experimentally measuring orders of convergence, we find for e, the rooted mean square integration error of the multilevel shifted rank-1 lattice rule applied to the Asian call option, the relation

$$e \propto (\log N)^{0.94} \cdot N^{-0.71}$$

where N denotes recorded cost in terms of the previously mentioned cost model. For comparison, we run the same experiments also for shifted rank-1 lattice rules, omitting the multilevel technique, as well as Monte Carlo rules where the integration points are sampled according to a multivariate normal distribution. In all cases we observe convergence

of the rooted mean square error being significantly slower than in case of the multilevel shifted rank-1 lattice rules.

Additionally we perform a line of experiments which explore the impact of deterministically chosen shifts on the integration error observed at given cost. Thereby we consider only shifts that are equal in each dimension. Our results document that the shift heavily affects the error. To give an impression, if one has somehow found the right shift, it is possible to compute the option price correctly up to the fifth digit after the decimal point, using only 17 integration points in 16 dimensions. This result suggests that it is sufficient to search for good shifts on the diagonal. However, it is still unclear how to do this search.

The thesis is organized as follows. In Chapter 2 we introduce the necessary background. In particular, we show thoroughly how to formulate option pricing as an infinite-dimensional integration problem over $\mathbb{R}^{\mathbb{N}}$.

Then, in Chapter 3 we present the results by Niu et al.[NHMR10]. We particularly emphasize the construction and characterization of the function space, adding a proof that verifies the function space to exist in the form as stated by the authors.

In Chapter 4 we describe the implementations we used to carry out the numerical experiments. We provide rather detailed descriptions since an important contribution of this thesis is the development of a fast and flexible framework which is easily extensible for further experimental designs, integrands and methods of integration point generation. In particular, we demonstrate that the assumptions on the cost model made for the theoretical part are well-founded by the real implementations.

Subsequently, we present the results of our numerical experiments in Chapter 5. Finally, in Chapter 6 we briefly name some open problems and give a short overview on open work regarding our implementational framework.

2 Fundamentals

2.1 Pricing of Path Dependent Options in the Black-Scholes Model

Let $T > 0$ and assume W to be a Brownian motion on the time interval $[0, T]$ taking values in $\mathbb{R}$. In the Black-Scholes model the price process of an asset is given by a *geometric Brownian motion* $S = \Gamma(W)$ with Γ defined by

$$\Gamma(f)(t) := s_0 \exp((r - \sigma^2/2)t + \sigma f(t)) \tag{2.1}$$

and r denoting the fixed income rate, σ the volatility of the asset and s_0 the initial value at time $t = 0$.

Path dependent option contracts are now modelled by P_S-integrable discounted payoffs $\varphi_p : C([0, T]) \to \mathbb{R}$. The prize of the option is set to be the expectation $\mathrm{E}[\varphi_p(S)]$.

A particular option is the *Asian Call Option* with payoff $\varphi_p = \varphi_{AC}$ given by

$$\varphi_{AC}(f) = e^{-rT} \max\left(\frac{1}{T}\int_0^T f(t)\mathrm{d}t - K, 0\right). \tag{2.2}$$

That is, the holder realizes profit if the average price of the underlying asset is greater than a strike prize K agreed upon beforehand.

To obtain the prize of the option one has to determine the expectation $\mathrm{E}[\varphi_p(S)]$. In case of path dependent payoffs φ_p closed-form solutions for this purpose are typically not available. Consequently we need numerical methods to compute the value at least approximately. The *classical Monte Carlo (MC) method* would be to estimate the expectation $\mathrm{E}[\varphi_p(S)]$ by taking the arithmetic mean of φ_p applied to a fixed number of i.i.d. samples distributed according to P_S. But this is not practicable as we can not simulate paths of S exactly.

Instead we have to find some approximating process $\hat{S}$ with paths we are able to sample such that φ_p is $P_{\hat{S}}$-integrable and

$$\mathrm{E}[\varphi_p(S)] \approx \mathrm{E}[\varphi_p(\hat{S})].$$

We make this more precise below.

We assume $T = 1$ in the following. This is no restriction as for $T' > 0$ the process $\tilde{W}$ defined by $\tilde{W}_t = \sqrt{T'} W_{t/T'}$ yields a Brownian motion on $[0, T']$.

2.1.1 Approximation of Stochastic Differential Equations

The option pricing scenario is a special case of the following situation: Consider a scalar, autonomous Stochastic Differential Equation (SDE)

$$\begin{aligned} \mathrm{d}X_t &= a(X_t)\mathrm{d}t + b(X_t)\mathrm{d}W_t, \\ X_0 &= x_0 \in \mathbb{R}, \end{aligned} \tag{2.3}$$

on the time interval $[0,1]$. The drift coefficient $a : \mathbb{R} \to \mathbb{R}$ and the diffusion coefficient $b : \mathbb{R} \to \mathbb{R}$ are assumed to satisfy the global Lipschitz condition

$$|a(x) - a(y)| + |b(x) - b(y)| \leq C_1|x - y|,$$

where C_1 is a positive constant and $x, y \in \mathbb{R}$. In the case of autonomous SDEs the global Lipschitz condition implies the linear growth condition

$$a(x)^2 + b(x)^2 \leq C_2(1 + x^2),$$

where $C_2 > 0$ is constant and $x \in \mathbb{R}$. These conditions guarantee the existence of a strong solution X of (2.3), see [KS98, Section 5.2]. Then determine the expectation $\mathrm{E}[\varphi(X)]$ of a P_X-integrable functional $\varphi : C([0,1]) \to \mathbb{R}$ with respect to X

We can recover the option pricing situation in this general SDE setting in two different ways:

R1: Either we consider the driving Brownian motion W as the solution of the trivial variant of (2.3) where $a = 0$, $b = 1$ and $x_0 = 0$. Then we consider functionals given by

$$\varphi = \varphi_p \circ \Gamma.$$

R2: Or we recover the geometric Brownian motion S as the solution of (2.3) where

$$\begin{aligned} a(x) &= rx, \\ b(x) &= \sigma x, \\ x_0 &= s_0. \end{aligned}$$

Then we consider $\varphi = \varphi_p$.

Consider $C([0,1]) \hookrightarrow L_2([0,1])$ and denote by $\|\cdot\|$ the norm on $L_2([0,1])$. In the next subsection we will introduce a construction scheme which yields a sequence of processes $(\hat{X}^{(l)})_{l\in\mathbb{N}}$ such that

$$\mathrm{E}[\|X - \hat{X}^{(l)}\|^2]^{\frac{1}{2}} \to 0 \text{ for } l \to \infty$$

and for each $l \in \mathbb{N}$ the process $\hat{X}^{(l)}$ has paths we can simulate. To obtain approximations for $\mathrm{E}[\varphi(X)]$ from $(\hat{X}^{(l)})_{l\in\mathbb{N}}$ we have to ensure that

(i) φ is $P_{\hat{X}^{(l)}}$-integrable and

(ii) $\lim_{l\to\infty} \mathrm{E}[\varphi(\hat{X}^{(l)})] = \mathrm{E}[\varphi(\hat{X})]$.

Example 2.1. In setting R2 it suffices to have φ_p Lipschitz continuous in order to guarantee (i),(ii). In setting R1, given Lipschitz continuity of φ_p, we additionally have to ensure that $(\varphi(\hat{X}^{(l)}))_{l\in\mathbb{N}}$ is uniformly integrable.

2.1.2 Euler Monte Carlo Method

The *Euler-Maruyama scheme* is a classical and simple method to construct approximations $\hat{X}^{(l)}$ of a solution X of (2.3) in the strong sense

$$\mathrm{E}[\|X - \hat{X}^{(l)}\|^2]^{\frac{1}{2}} \to 0$$

for $l \to \infty$. For a detailed introduction see e.g. [KP92, Chapter 9]. Here we consider only the case of equidistant time discretizations.

Definition 2.2 (Euler-Maruyama scheme)**.** Choose $l \in \mathbb{N}_0$ and construct the *Euler approximation* $\hat{X}^{(l)}$ using the iterative scheme

$$\begin{aligned} \hat{X}_0^{(l)} &= x_0 \\ \hat{X}_{t_{i+1}}^{(l)} &= \hat{X}_{t_i}^{(l)} + a(\hat{X}_{t_i}^{(l)})(t_{i+1} - t_i) + b(\hat{X}_{t_i}^{(l)}) \cdot (W_{t_{i+1}} - W_{t_i}) \end{aligned}$$

where $t_i = i \cdot 2^{-l}$, $i = 0, ..., 2^l$. For $t \notin \{t_0, ..., t_{2^l}\}$ use piecewise linear interpolation:

$$\hat{X}_t^{(l)} = \hat{X}_{t_i}^{(l)} + \frac{t - t_i}{t_{i+1} - t_i}(\hat{X}_{t_{i+1}}^{(l)} - \hat{X}_{t_i}^{(l)}), \quad t \in [t_i, t_{i+1}).$$

Paths of $\hat{X}^{(l)}$ are easy to sample as we only need realizations of the independent Brownian increments

$$\Delta W^{(i)} = W_{t_{i+1}} - W_{t_i}, \quad i = 0, ..., 2^l - 1 \tag{2.4}$$

which are normally distributed with mean 0 and variance δ_l.

Remark 2.3. Alternatively, we may define the Euler-Maruyama scheme on the path space for given $l \in \mathbb{N}$ via the map $\zeta^{(l)} : C([0,1]) \to C([0,1])$ given by

$$\begin{aligned} \zeta^{(l)}(f)(0) &= x_0 \\ \zeta^{(l)}(f)(t_{i+1}) &= \zeta^{(l)}(f)(t_i) + a(\zeta^{(l)}(f)(t_i))(t_{i+1} - t_i) + b(\zeta^{(l)}(f)(t_i)) \cdot (f(t_{i+1}) - f(t_i)) \end{aligned}$$

for $t_i = i \cdot 2^{-l}$, $i = 0, ..., 2^l$ and piecewise linear interpolation for $t \notin \{t_0, ..., t_{2^l}\}$.

Then we have

$$\hat{X}^{(l)} = \zeta^{(l)}(W).$$

Imposing sufficient conditions on φ we now get a practicable numerical method to estimate $\mathrm{E}[\varphi(X)]$ by using the classical Monte Carlo method with samples drawn under the distribution $P_{\hat{X}^{(l)}}$.

Definition 2.4 (Euler Monte Carlo Method)**.** Let $l \in \mathbb{N}$ and $n \in \mathbb{N}$. The *Euler MC method* $A_{l,n}^{MC} = A_{l,n}^{MC}(\varphi, a, b)$ is defined by

$$A_{l,n}^{MC} = \frac{1}{n} \sum_{j=1}^{n} \varphi(\hat{X}_j^{(l)}) \tag{2.5}$$

with $\hat{X}_1^{(l)}, ..., \hat{X}_n^{(l)}$ denoting n independent copies of $\hat{X}^{(l)}$.

Remark 2.5. Note that $A_{l,n}^{MC}$ uses realizations of $n \cdot 2^{l-1}$ independent random variables with mean 0 and variance 2^{-l}.

Remark 2.6. For an estimator A of $\mathrm{E}[\varphi(X)]$ the *mean squared error*

$$e^2(A) = \mathrm{E}\left[(\mathrm{E}[\varphi(X)] - A)^2\right] \tag{2.6}$$

splits into two sources of error,

$$e^2(A) = b^2(A) + \mathrm{Var}(A).$$

Here $b(A)$ denotes the *bias*

$$b(A) = \left|\mathrm{E}[\varphi(X)] - \mathrm{E}[A]\right|. \tag{2.7}$$

For the Euler MC method we usually have $b(A_{l,n}^{MC}) \neq 0$ as we only approximate the paths of X. That is, $A_{l,n}^{MC}$ is an biased estimator of $\mathrm{E}[\varphi(X)]$. The second error source $\mathrm{Var}(A_{l,n}^{MC})$ arises from the fact that we have only finitely many sample paths of $X^{(l)}$ at hand to estimate the expectation $\mathrm{E}[\varphi(\hat{X}^{(l)})]$. We have

$$\mathrm{Var}(A_{l,n}^{MC}) = \frac{1}{n^2}\mathrm{Var}(\varphi(\hat{X}^{(l)})).$$

2.1.3 Multilevel Euler Monte Carlo Method

The following concept was first developed in [H98], [HS99], and [H01]. In the context of SDE it was brought up by [G08]. Let $L \in \mathbb{N}$ and $\hat{X}^{(l)}$ be Euler approximations of X for $l = 1, ..., L$. Then it clearly holds true that

$$\mathrm{E}[\varphi(\hat{X}^{(L)})] = \mathrm{E}[\varphi(\hat{X}^{(1)})] + \sum_{l=2}^{L} \mathrm{E}[\varphi(\hat{X}^{(l)}) - \varphi(\hat{X}^{(l-1)})]. \tag{2.8}$$

Given sufficient conditions on φ it is often the case that the difference $\varphi(\hat{X}^{(l)}) - \varphi(\hat{X}^{(l-1)})$ converges in L_2 with limit 0. Hence variances decline for increasing l and in particular, assumptions (i) and (ii) are fulfilled then. The *multilevel idea* now is to exploit the declining variances by estimating each expectation on the right hand side of (2.8) independently using modified Euler MC methods. Thereby we allow the less samples to be used the greater l gets. We hope that we reach a better estimate but with less realizations of Brownian increments needed as if we had approximated $\mathrm{E}[\varphi(\hat{X}^{(L)})]$ directly with the Euler Monte Carlo method.

The modification requires the Euler-Maruyama scheme to be altered such that for a realization of $\varphi(\hat{X}^{(l)}) - \varphi(\hat{X}^{(l-1)})$, $l \geq 2$, both $\hat{X}^{(l)}$ and $\hat{X}^{(l-1)}$ use the same realization of Brownian increments.

This is easily accomplished because of the fact that for $\Delta W_1, \Delta W_2$ being normally distributed with mean 0 and variance 2^{-l}, the sum

$$\Delta W_1 + \Delta W_2$$

is normally distributed with mean 0 and variance 2^{-l+1}. Consequently, if we are given a realization $(\Delta w_i)_{i=0,\dots,2^{l-1}-1}$ of Brownian increments for $\hat{X}^{(l)}$, we obtain a realization of Brownian increments for $\hat{X}^{(l-1)}$ by

$$(\Delta w_i + \Delta w_{i+1})_{i=0,\dots,2^{l-2}-1}.$$

We refer to this modified version in the following as the *multilevel Euler-Maruyama scheme*.

Definition 2.7 (Multilevel Euler Monte Carlo Method)**.** Choose $L \in \mathbb{N}$, $L > 0$ and numbers $\mathbf{n} = (n_1, \dots, n_L)$. The *multilevel Euler MC method* $A^{ML}_{L,\mathbf{n}} = A^{ML}_{L,\mathbf{n}}(\varphi, a, b)$ is defined by

$$A^{ML}_{L,\mathbf{n}} = \frac{1}{n_1}\sum_{j=1}^{n_1} \varphi(\hat{X}^{(1)}_j) + \sum_{l=2}^{L} \frac{1}{n_l}\sum_{j=1}^{n_l} \left(\varphi(\hat{X}^{(l)}_j) - \varphi(\hat{X}^{(l-1)}_j)\right).$$

Here $(\hat{X}^{(1)}_j)_{j=1,\dots,n_1}$ denote n_1 independent copies of $\hat{X}^{(1)}$. For $l = 2, \dots, L$ the tuples $(\hat{X}^{(l)}_j, \hat{X}^{(l-1)}_j)_{j=1,\dots,n_l}$ are n_l independent copies of the coupled Euler approximations $(\hat{X}^{(l)}, \hat{X}^{(l-1)})$ constructed according to the multilevel Euler-Maruyama scheme.

Remark 2.8. The multilevel Euler MC method $A^{ML}_{L,\mathbf{n}}$ is a biased estimator for $\mathrm{E}[\varphi(X)]$. We have

$$\mathrm{Var}(A^{ML}_{L,\mathbf{n}}) = \frac{1}{n_1}\mathrm{Var}(\varphi(\hat{X}^{(1)})) + \sum_{l=2}^{L}\frac{1}{n_l}\mathrm{Var}(\varphi(\hat{X}^{(l)}) - \varphi(\hat{X}^{(l-1)}))$$

and typically $b(A^{ML}_{L,\mathbf{n}}) \neq 0$.

Moreover, the multilevel Euler MC method has the advantage over the classical Euler MC method that it allows to estimate the bias, see Paragraph S3 in Section 4.4 below.

2.1.4 Option Pricing as Infinite-dimensional Integral

So far we have considered the option pricing scenario from an SDE perspective. Now we establish a link between infinite-dimensional integration over $\mathbb{R}^{\mathbb{N}}$ and option pricing in the SDE setting R1. This will basically result from describing Brownian motion in a pertinent form.

Brownian motion can be written in form of a series representation

$$W = \sum_{j=0}^{\infty} X_j e_j$$

where $(X_j)_{j\in\mathbb{N}_0}$ denotes a sequence of independent standard normally distributed random variables and $(e_j)_{j\in\mathbb{N}_0}$ a sequence of functions from $C([0,1])$. We name two well-known series representations and make precise in which sense convergence of the partial sums $\sum_{j=0}^{n} X_j e_j$ is given.

With the *Karhunen-Loève expansion* we actually reconstruct a given Brownian motion W, see e.g. [R00, Example 28, p. 54/55]. The space $C([0,1])$ is considered as an embedding in $L_2([0,1])$. For the functions e_j one has

$$e_j(t) = \sqrt{2}\frac{sin(\pi(j+1/2)t)}{\pi(j+1/2)} \tag{2.9}$$

and the sequence of independent normally distributed random variables $(X_j)_{j\in\mathbb{N}}$ is recovered from the given Brownian motion W by

$$X_j = \left\langle W, e_j \right\rangle_{L_2}.$$

Then $W = \sum_{j=0}^{\infty} X_j e_j$ in the sense that

$$\lim_{n\to\infty} \mathrm{E}[\|W - \sum_{j=0}^{n} X_j e_j\|_{L_2}^2]^{1/2} = 0.$$

The sequence $(e_j)_{j\in\mathbb{N}}$ forms an orthogonal basis of $L_2([0,1])$. Consequently, orthogonality and independence of $(X_j e_j)_{j\in\mathbb{N}}$ coincide. Because of $|sin(\pi(j+1/2)t)| \leq 1$ the terms $X_j e_j(t)$ contribute less and less to the variance of W_t with increasing j. In particular, among all linear methods which approximate W by means of a certain number k of i.i.d. standard normally distributed variables, the Karhunen-Loève partial sum up to term k explains most of the variance of W.

Remark 2.9. Reconstructing a Brownian motion by means of the Karhunen-Loève expansion is also known as *principal component construction*.

The second series representation is the *Lévy-Ciesielski expansion*, see e.g. [K06, Section 21.5]. Here we consider $C([0,1])$ equipped with the usual supremum norm $\|\cdot\|_\infty$. Define

$$J(m) := \max(2^{m-1} - 1, 0)$$

for $m \in \mathbb{N}_0$ and let $(X_{m,j})_{m\in\mathbb{N}_0, j=0,\ldots,J(m)}$ be a sequence of independent standard normally distributed random variables on a common probability space $(\Omega, \mathfrak{A}, P)$. Further choose $(e_{m,j})_{m,j}$ to be the *Schauder functions* which are given by $e_{0,0}(t) = t$ for $m = j = 0$ and otherwise

$$e_{m,j}(t) = 2^{\frac{(m-1)}{2}} \begin{cases} t - \frac{k-1}{2^m} & \text{if } t \in I(k-1,m) \\ \frac{k+1}{2^m} - t & \text{if } t \in I(k,m) \\ 0, & \text{else} \end{cases} \tag{2.10}$$

with $k = 2j+1$ and $I(m,k) := [\frac{k}{2^m}, \frac{k+1}{2^m}[$. The Schauder functions have the form of hat functions, see Figure 2.1 for a graphical illustration.

Now the partial sums

$$L^{(n)} := \sum_{m=0}^{n} \sum_{j=0}^{J(m)} X_{m,j} e_{m,j} \tag{2.11}$$

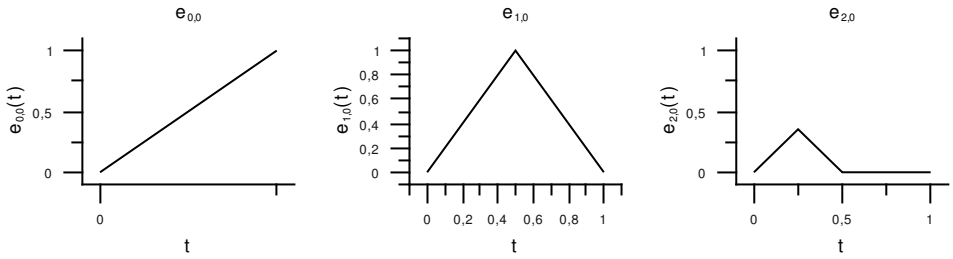

Figure 2.1: The first Schauder functions for $m = 0, 1, 2$.

P-a.s. converge uniformly in t to a process W which is a Brownian motion. Moreover, we even have

$$\lim_{n\to\infty} \mathrm{E}[\|W - L^{(n)}\|_\infty^2]^{1/2} = 0.$$

For the partial sum $L^{(n)}$ we can easily observe that it is a process with continuous, piecewise linear paths. In particular, on each interval $I(n,i)$, $i \in \{0,\dots,2^n - 1\}$ we have $L^{(n)}$ given by

$$L_t^{(n)} = c_1(n,i) \cdot t + c_2(n,i) \tag{2.12}$$

where the coefficients $c_1(n,i)$ and $c_1(n,i)$ are linear combinations of the $X_{m,j}$ which depend on the interval. For a graphical illustration how $L^{(n)}$ is generated from the Schauder functions see Figure 2.2. A detailed description of the coefficients $c_1(n,i)$, $c_2(n,i)$ and their efficient calculation is given in Subsection 4.2.2 later in this work.

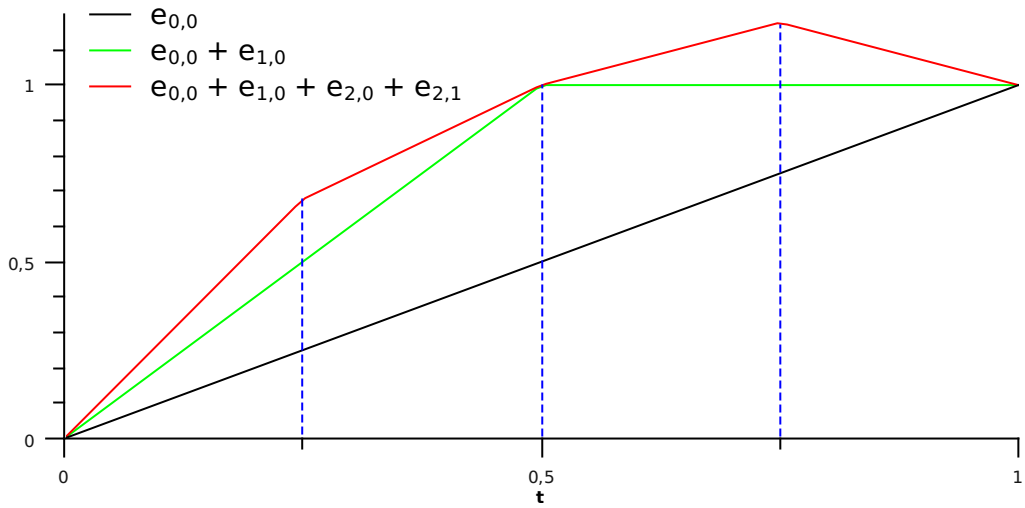

Figure 2.2: Illustration of $L^{(2)}$ assuming $X_{m,j} = 1$.

The partial sum $L^{(n)}$ exactly reproduces W at the points $t_i = i/2^n$, $i = 0, ..., 2^n$. For increasing m, j the terms $X_{m,j}e_{m,j}(t_i)$ contribute less and less to the variance of W_{t_i}, demonstrated the best by an example: For $i = 1/4, 1/2, 3/4, 1$ we have

$$\begin{aligned}
\mathrm{Var}(W_1) &= 1 = \mathrm{Var}(X_{0,0}),\\
\mathrm{Var}(W_{1/4}) &= \frac{1}{4} = \mathrm{Var}(\frac{1}{4}X_{0,0} + \frac{1}{4}X_{1,0} + \frac{1}{\sqrt{8}}X_{2,0}),\\
\mathrm{Var}(W_{1/2}) &= \frac{1}{2} = \mathrm{Var}(\frac{1}{2}X_{0,0} + \frac{1}{2}X_{1,0}),\\
\mathrm{Var}(W_{3/4}) &= \frac{3}{4} = \mathrm{Var}(\frac{3}{4}X_{0,0} + \frac{1}{4}X_{1,0} + \frac{1}{\sqrt{8}}X_{2,1}).
\end{aligned}$$

Remark 2.10. Constructing Brownian motion by means of the Lévy-Ciesielski expansion is a special case of the *Brownian bridge construction*.

Remark 2.11. Partial sums of the Lévy-Ciesielski expansion are closely related to the Euler-Maruyama scheme in the SDE setting R1. For fixed $n \in \mathbb{N}$ the distributions of $L^{(n)}$ and of the Euler approximation $\hat{X}^{(n)}$ on the path space $C([0,1])$ coincide,

$$P_{L^{(n)}} = P_{\hat{X}^{(n)}}.$$

In particular, let $\mathbf{x}_{1:2^n} = (x_i)_{i=1,...,2^n}$ denote a realization of 2^n standard normally distributed random variables. Then we obtain on the one hand a sample path of $\hat{X}^{(n)}$ if we take $(\sqrt{2^{-n}}x_i)_{i=1,...,2^n}$ as Brownian increments, on the other hand we obtain a sample path of $L^{(n)}$ if we take $\mathbf{x}_{1:2^n}$ as a realization of $(X_{m,j})_{m=0,...,n;j=0,...,J(m)}$.

Now let W be a Brownian motion given by one of the previously introduced series representations. For the sake of simplicity we use in either case the same indexing for the moment, writing $W = \sum_{j=0}^{\infty} X_j e_j$. Put $\mu = \otimes_{j\in\mathbb{N}} N(0,1)$. Then there is a subset $\mathfrak{X} \subseteq \mathbb{R}^{\mathbb{N}}$ with $\mu(\mathfrak{X}) = 1$ such that for a feasible payoff φ_p the map f defined by

$$f(\mathbf{x}) := \varphi_p \circ \Gamma(\sum_{j=0}^{\infty} x_j e_j) \tag{2.13}$$

is a real valued function on $\mathfrak{X}$. For this f we have $\mathrm{E}[\varphi_p \circ \Gamma(W)] = I(f)$ where

$$I(f) = \int_{\mathbb{R}^{\mathbb{N}}} f(\mathbf{x})\mu(\mathrm{d}\mathbf{x}).$$

2.2 Information-based Complexity

In the previous section we saw two particular randomized algorithms for option pricing in the Black-Scholes model. Now there is a number of questions which naturnally arise:

- What are quantities to measure the performance of these algorithms?
- How good are these algorithms, are there better ones?

- How hard is the considered problem? That is, is there a lower bound on the performance which algorithms can achieve?

This section is devoted to introducing quantities and a formal setting from information-based complexity theory to treat questions of this kind. Thereby we stick to the perspective of infinite-dimensional integration we developed in Subsection 2.1.4.

The abstract problem we study has the following form. Let D be a Borel subset of $\mathbb{R}$ and let ρ be a probability measure on $\mathfrak{B}(D)$. We consider the product measure $\mu := \otimes_{j\in\mathbb{N}}\rho$ and a measurable subset $\mathfrak{X} \subseteq D^{\mathbb{N}}$ with $\mu(\mathfrak{X}) = 1$. Then let F be a class of μ-integrable functions $f : \mathfrak{X} \to \mathbb{R}$. The *infinite-dimensional integration problem over* $\mathbb{R}^{\mathbb{N}}$ is now to calculate the integral

$$I(f) := \int_{\mathfrak{X}} f(\mathbf{x})\mu(\mathrm{d}\mathbf{x}) \tag{2.14}$$

for $f \in F$.

Definition 2.12. We denote the infinite-dimensional integration problem as described above by the tupel $(D, \rho, \mathfrak{X}, F)$.

Remark 2.13. The class F embodies global information which is available about the integrands, e.g. smoothness constraints.

Example 2.14. Let W be a Brownian motion given by the Lévy-Ciesielski expansion. Then option pricing for Lipschitz continuous payoffs in the Black-Scholes model according to the previous definition is the tupel $(\mathbb{R}, \mathfrak{X}, N(0,1), F)$ where

$$\mathfrak{X} = \{\mathbf{x} \in \mathbb{R}^{\mathbb{N}} : \lim_{n\to\infty} \| \sum_{j=0}^{n} x_j e_j \|_\infty < \infty\}$$

and F denotes the set of all functions given by (2.13), the payoffs φ_p being Lipschitz continuous now.

In the scope of this thesis we will consider only classes F of integrands which are symmetric, convex subsets of a vector space over $\mathbb{R}$. We recall:

Definition 2.15. Let F be a subset of some vector space over $\mathbb{R}$. Then we call F

(i) *symmetric* if for all $f \in F$ it holds true that $-f \in F$.

(ii) *convex* if for $f, g \in F$ and $\lambda \in [0,1]$ the vector $\lambda f + (1-\lambda)g$ is in F, too.

If we think of a method to calculate the integral $I(f)$ running on a real existent machine and which is given some subroutine that computes the evaluation of f at a given point $\mathbf{x}$, this subroutine can certainly take only a finite number of components of $\mathbf{x}$ into account.

We address this limitation in our abstract setting as follows: Fix some $\mathbf{a} \in \mathfrak{X}$ and define for each finite subset $\upsilon \subset \mathbb{N}$ the set

$$\mathfrak{X}_{\upsilon,\mathbf{a}} = \left\{\mathbf{x} \in \mathfrak{X} : \mathbf{x}_{-\upsilon} = \mathbf{a}_{-\upsilon}\right\}.$$

Then for a particular integrand $f \in F$ the available information is given by

$$f(\mathbf{x}), \quad \mathbf{x} \in \bigcup_{\upsilon \in P_0(\mathbb{N})} \mathfrak{X}_{\upsilon,\mathbf{a}}.$$

Assume information is only gained using points in $\mathfrak{X}_{\upsilon,\mathbf{a}}$ for some $\upsilon \in P_0(\mathbb{N})$. Further define $\Psi_{\upsilon,\mathbf{a}} : \mathbb{R}^{(\mathbb{R}^{\mathbb{N}})} \to \mathbb{R}^{(\mathbb{R}^{\mathbb{N}})}$ by

$$(\Psi_{\upsilon,\mathbf{a}} f)(\mathbf{x}) := f(\mathbf{x}_\upsilon, \mathbf{a}_{-\upsilon}). \tag{2.15}$$

Then any method using only this information to compute approximative solutions to the infinite-dimensional integration problem actually only produces solutions to the finite-dimensional *subproblem* given by

$$\int_{\mathfrak{X}_{\upsilon,\mathbf{a}}} f(\mathbf{x})\mu(\mathrm{d}\mathbf{x}), \quad f \in \Psi_{\upsilon,\mathbf{a}}(F).$$

2.2.1 Quadrature Rules and General Multilevel Algorithm

The algorithms we consider for numerical integration in this thesis are *quadrature rules*:

Definition 2.16. Let $n \in \mathbb{N}$ and let $\mathbf{w} = (w_1, ..., w_n) \in \mathbb{R}^n$. Further let $\mathscr{X} = (\mathbf{X}_1, ..., \mathbf{X}_n)$ denote a family of random vectors defined over a common probability space, the $\mathbf{X}_i$ taking values in $\bigcup_{\upsilon \subset \mathbb{N}} \mathfrak{X}_{\upsilon,\mathbf{a}}$. Then $Q(\cdot; \mathscr{X}, \mathbf{w})$ given by

$$Q(f; \mathscr{X}, \mathbf{w}) = \sum_{i=1}^{n} w_i f(\mathbf{X}_i), \ f \in F$$

is called a *randomized quadrature rule with deterministic weights*.

If all $w_i = 1/n$ we write $Q(\cdot; \mathscr{X}) = Q(\cdot; \mathscr{X}, \mathbf{w})$. If $\mathscr{X}$ consists of i.i.d. random vectors we call the quadrature rule a *classical Monte-Carlo rule*. If each $\mathbf{X}_i$ is constant we call it a *deterministic* quadrature rule and prefer the notation $\mathscr{X} = P_n$ to emphasize that we have a set of n points now.

Remark 2.17. Generally, if we want to emphasize that we generate integration points randomly according to some target distribution then we use the terminology *Monte Carlo integration*.

Example 2.18. We continue Example 2.14. In terms of SDE setting R1 we recover the Euler Monte Carlo method as classical Monte Carlo rules applied to certain integrands. Therefor let $\mathbf{a} = \mathbf{0} = (0, 0, ...)$. Furthermore recall the partial sums of the Lévy-Ciesielski expansion $L^{(l)}$, $l \in \mathbb{N}$, see (2.11). Now let $l \in \mathbb{N}$ and put $\upsilon = 1 : 2^l$. By Remark 2.11 the Euler approximation $\hat{X}^{(l)}$ has the same distribution on the path space $C([0,1])$ as the partial sum $L^{(l)}$ of the Lévy-Ciesielski expansion. Consequently,

$$\mathrm{E}[\varphi(\hat{X}^{(l)})] = I(\Psi_{\upsilon,\mathbf{0}} f)$$

for the integrand f defined by (2.13). Moreover, the quadrature rule $Q(f; \mathscr{X})$ corresponds to the Euler Monte Carlo method if $\mathscr{X}$ consists of i.i.d. random vectors with distribution $\mu_\upsilon = \otimes_{j=1}^{2^l} N(0,1)$.

Now we define the general multilevel algorithm for integration over $\mathbb{R}^{\mathbb{N}}$ which uses quadrature rules as building blocks:

Definition 2.19 (Multilevel Algorithm)**.** Let $L \in \mathbb{N}$, $L > 0$ and further $\mathbf{d} = (d_1, ..., d_L) \in \mathbb{N}^L$ as well as $\mathbf{n} = (n_1, ..., n_L) \in \mathbb{N}^L$. Further, let $\mathscr{X}_{n_1,d_1}, ..., \mathscr{X}_{n_L,d_L}$ be independent families of random vectors defined over a common probability space such that $\mathscr{X}_{n_i,d_i}$ consists of n_i random vectors each of which takes values in $\mathfrak{X}_{1:d_i,\mathbf{a}}$. Moreover, for $i = 1, ..., L$ let $\mathbf{w}^{(i)} \in \mathbb{R}^{n_i}$. Then the *multilevel algorithm* is defined by

$$Q^{ml}_{\mathbf{n},\mathbf{d}}(f) := Q(\Psi_{1:d_1,a} f; \mathscr{X}_{n_1,d_1}, \mathbf{w}^{(1)}) + \sum_{l=2}^{L} Q(\Psi_{1:d_l,a} f - \Psi_{1:d_{l-1},a} f; \mathscr{X}_{n_l,d_l}, \mathbf{w}^{(l)}). \quad (2.16)$$

We call the numbers $l = 1, ..., L$ the *levels* of the algorithms.

In particular, if the building blocks $Q(\cdot; \mathscr{X}_{n_i,d_i}, \mathbf{w}^{(i)})$ are classical Monte Carlo rules, we refer to $Q^{ml}_{\mathbf{n},\mathbf{d}}$ as *multilevel Monte Carlo rule*.

Remark 2.20. Let $d^* = \max\{d_1, ..., d_L\}$. Then the multilevel algorithm $Q^{ml}_{\mathbf{n},\mathbf{d}}$ is a weighted quadrature rule using random vectors which take values in $\mathfrak{X}_{1:d^*,\mathbf{a}}$.

2.2.2 Cost and Error Models

To draw conclusions on the intrinsic difficulty of a problem and the performance of particular quadrature algorithms we have to specify what we consider to be cost and error of an algorithm. We follow the models used in [HMNR10].

We are concerned both with randomized and deterministic algorithms, hence the following error definition w.r.t. a particular integrand is obvious:

Definition 2.21 (Integration Error)**.** Let $Q = Q(\cdot; \mathscr{X}, \mathbf{w})$ be a quadrature rule. We define the *integration error* $e(Q, f)$ which the algorithm Q makes for a particular $f \in F$ as the root mean squared error

$$e(Q, f) = \sqrt{\mathrm{E}[((I(f) - Q(f))^2]}. \quad (2.17)$$

Typically, a quadrature rule Q takes only integration points from $\mathfrak{X}_{\upsilon,\mathbf{a}}$ for some finite $\upsilon \subset \mathbb{N}$. This implies $Q(f) = Q(\Psi_{\upsilon,\mathbf{a}} f)$. Then the integration error $e(Q, f)$ comprises two sources of error. Namely a *truncation error*

$$|I(f) - I(\Psi_{\upsilon,\mathbf{a}} f)|$$

which is because Q actually approximates solutions to a finite-dimensional subproblem, and the integration error

$$\sqrt{\mathrm{E}[(I(\Psi_{\upsilon,\mathbf{a}} f) - Q(\Psi_{\upsilon,\mathbf{a}} f))^2]}$$

which the rule Q produces for the integrand $\Psi_{\upsilon,\mathbf{a}} f$. Using

$$e(Q, f) \leq |I(f) - I(\Psi_{\upsilon,\mathbf{a}} f)| + \sqrt{\mathrm{E}[(I(\Psi_{\upsilon,\mathbf{a}} f) - Q(\Psi_{\upsilon,\mathbf{a}} f))^2]}$$

we may seperate the analysis of truncation and finite-dimensional integration error. This provides a way to obtain at least upper bounds on the integration error.

Regarding not only the error with respect to a single function $f \in F$ but the whole class we take a pessimistic perspective and jugde quadrature rules by means of the greatest error they make over all integrands in the class F. If only deterministc algorithms are allowed this setting is called *worst-case setting* in the literature. If randomized algorithms are considered, as well, usually the term *randomized setting* is used.

Definition 2.22 (Worst-case Integration Error)**.** Given a class of integrands F and a quadrature algorithm Q we define the *(randomized) worst-case error* of Q to be

$$e(Q,F) := \sup_{f\in F} e(Q,f). \tag{2.18}$$

Remark 2.23. If Q denotes a deterministic algorithm the worst-case error reduces to

$$e(Q,F) = \sup_{f\in F} |I(f) - Q(f)|.$$

Now we turn to the cost models. We assume that for each possible integrand $f \in F$ there is a subroutine which calculates the value $f(\mathbf{x})$ for a given point $\mathbf{x} \in \mathfrak{X}$. Then we consider two distinct cost models for evaluating f at a point $\mathbf{x}$:

Definition 2.24 (Fixed Subspace Sampling (fix))**.** Choose a finite $\emptyset \neq \upsilon \subset \mathbb{N}$ and $\mathbf{a} \in \mathfrak{X}$. Then evaluations are only allowed at points $\mathbf{x} \in \mathfrak{X}_{\upsilon,\mathbf{a}}$. The cost for evaluation at $\mathbf{x}$ coincides with the dimension of $\mathfrak{X}_{\upsilon,\mathbf{a}}$, that is it is modelled by a function

$$c_{\upsilon,\mathbf{a}}(\mathbf{x}) = \begin{cases} \dim(\mathfrak{X}_{\upsilon,\mathbf{a}}) & , \quad \text{if } \mathbf{x} \in \mathfrak{X}_{\upsilon,\mathbf{a}} \\ \infty & , \quad \text{otherwise.} \end{cases}$$

Definition 2.25 (Variable Subspace Sampling (var))**.** Choose $\mathbf{a} \in \mathfrak{X}$ and an increasing sequence $\mathbf{v} = (\upsilon_i)_{i\in\mathbb{N}}$ of finite nonempty subsets such that

$$\mathfrak{X}_{\upsilon_1,\mathbf{a}} \subset \mathfrak{X}_{\upsilon_2,\mathbf{a}} \subset \dots .$$

Now points for evaluation are allowed which are contained in one of the finite-dimensional subspaces $\mathfrak{X}_{\upsilon_i,\mathbf{a}}$. The cost is thereby set to be

$$c_{\mathbf{v},\mathbf{a}}(\mathbf{x}) = \inf\{\dim(\mathfrak{X}_{\upsilon_i,\mathbf{a}}) : \mathbf{x} \in \mathfrak{X}_{\upsilon_i,\mathbf{a}}\}$$

where we put $\inf\emptyset = \infty$.

For the computational cost of a quadrature algorithm we accommondate only function evaluation and omit any cost which arises from arithmetical operations. Thus, for a given cost function c we set $\mathrm{cost}_c(Q,f)$ to be the sum of the evaluation costs over all integration points used by Q. Note that it is possible that Q chooses the integration points adaptively depending on the integrand f and hence the cost depend on f, too. For randomized algorithms $\mathrm{cost}_c(Q,f)$ is a random variable.

According to the worst-case and randomized setting respectively, we define the (randomized) wort-case cost as follows:

Definition 2.26 (Worst-case cost)**.** Let $\Delta \in \{\text{fix}, \text{var}\}$ and define

$$C_{\text{fix}} := \{c_{\upsilon,\mathbf{a}} : \upsilon \subset \mathbb{N} \text{ finite}, \ \mathbf{a} \in \mathfrak{X}\},$$
$$C_{\text{var}} := \{c_{\mathbf{v},\mathbf{a}} : \mathbf{v} = (\upsilon_i)_{i\in\mathbb{N}} \text{ as above}, \ \mathbf{a} \in \mathfrak{X}\}.$$

Then the *worst-case cost* $\text{cost}_\Delta(Q, F)$ of a quadrature algorithm Q is defined to be

$$\text{cost}_\Delta(Q, F) := \inf_{c\in C_\Delta} \sup_{f\in F} \mathrm{E}[\text{cost}_c(Q, f)].$$

Remark 2.27. The infimum construction "$\inf_{c\in C_\Delta}$ ..." assures that we deal fairly with the algorithm on the basis of a cost function which fits to the points at which the algorithm evaluates the integrand.

Example 2.28. For the multilevel algorithm given by (2.19) we have

$$\text{cost}_{\text{var}}(Q^{ml}_{\mathbf{n},\mathbf{d}}, F) \le n_1 d_1 + \sum_{l=2}^{L} 2n_l d_l.$$

The concepts we have seen so far allow to draw conclusion on the performance of a particular algorithm. To examine the hardness of a problem we need quantities which give information about the performance of a whole class of algorithms for the integration problem.

Denote by $\mathcal{Q}^{\text{ran}}(F)$ the class of randomized quadrature rules for functions in F and by $\mathcal{Q}^{\text{det}}(F)$ the class of deterministic quadrature rules.

Definition 2.29 (n-th Minimal Errors and Information Complexity)**.** For $\Diamond \in \{\text{det}, \text{ran}\}$ and $\Delta \in \{\text{fix}, \text{var}\}$ the *n-th minimal worst-case error* is given by

$$e^{\Diamond}_{N,\Delta}(F) := \inf\{e(Q, F) : Q \in \mathcal{Q}^{\Diamond}(F) \wedge \text{cost}_\Delta(Q, F) \le N\}. \tag{2.19}$$

The *information complexity* is given by

$$\text{comp}^{\Diamond}_{\varepsilon,\Delta}(F) := \inf\{\text{cost}_\Delta(Q, F) : Q \in \mathcal{Q}^{\Diamond}(F) \wedge e(Q, F) \le \varepsilon\}. \tag{2.20}$$

Remark 2.30. As each deterministic algorithm is in particular a randomized algorithm we have $\mathcal{Q}^{\text{det}}(F) \subset \mathcal{Q}^{\text{ran}}(F)$. Consequently,

$$e^{\text{ran}}_{N,\Delta}(F) \le e^{\text{det}}_{N,\Delta}(F)$$

In the literature the n-th minimal errors and the information complexity are usually defined with respect to a broader class of algorithms. But in the case of deterministic quadrature rules it suffices to introduce the quantities with respect to $\mathcal{Q}^{\text{det}}(F)$ as we only consider classes F of integrands which are symmetric, convex subsets of some vector space, see e.g. [MNR11, Prop. 7.24, p. 253f].

However, in the case of randomized quadrature rules the n-th minimal error and the information complexity as defined here yield merely upper and lower bounds, respectively, for these quantities defined in the usual way.

Definition 2.31 (Optimal Quadrature Rule)**.** We call an quadrature rule Q optimal if we have $e(Q,F) = e^{\Diamond}_{N,\Delta}(F)$ for $\mathrm{cost}_{\Delta}(Q,F) \leq N$.

Definition 2.32. The infinite-dimensional integration problem is said to be *tractable* if there exists $p > 0$ such that

$$e^{\Diamond}_{N,\Delta}(F) = O(N^{-p}). \tag{2.21}$$

The supremum p^* over all $p > 0$ such that (2.21) is satisfied is called the *exponent* of the problem.

Remark 2.33. Equivalently a problem is tractable if there exists $q > 0$ such that

$$\mathrm{comp}^{\Diamond}_{\varepsilon,\Delta}(F) = O(\varepsilon^{-q}). \tag{2.22}$$

For the infimum q^* over all $q > 0$ such that (2.22) is satified, we have $q* = 1/p*$ where $p*$ is the exponent of the problem.

Remark 2.34. We introduced key notions from information based complexity theory in a way which is suitable for our SDE perspective R1. We conclude this section noting that all notions and problem definitions can be introduced analogously in a way suited for perspective R2. Then one usually studies $\mathfrak{X} = (C([0,1]), \|\cdot\|_{\infty})$ or $\mathfrak{X} = L_p([0,1])$ with $1 \leq p < \infty$, and μ is the distribution of the geometric Brownian motion on the path space $C([0,1])$.

In particular, we are then given a theoretical result on pricing of the Asian Call option in the Black Scholes model. Let F denote the class of all Lipschitz continuous functionals on $C([0,1])$ with Lipschitz bound at most 1. Then we have $\varphi_{\mathrm{AC}} \in F$. Now, for fixed subspace sampling it is known that

$$e^{\mathrm{ran}}_{N,\mathrm{fix}}(F) \preccurlyeq (N^{-1/4}). \tag{2.23}$$

The bound is derived by means of the Euler Monte Carlo method. For the variable subspace sampling regime it is known that

$$N^{-1/2} \preccurlyeq e^{\mathrm{ran}}_{N,\mathrm{var}}(F) \preccurlyeq N^{-1/2}\log N. \tag{2.24}$$

Here the bound is derived by means of the Multilevel Euler Monte Carlo method. For both results see [CDMR08, Theorem 11].

2.3 Reproducing Kernel Hilbert Spaces (RKHS)

The function spaces we will present in Chapter 3 are Hilbert spaces whose elements are functions from some domain E into the reals and on which point evaluations are continuous linear functionals. In this section we introduce the general theory of such Hilbert spaces as far as necessary for Chapter 3. A comprehensive treatise of this topic is found in [A50].

Note that we consider only real valued kernels in the following. However, all result are also true for kernels with complex values.

2.3.1 Definition and Existence

Let E be some set and H a Hilbert space with inner product denoted by $\langle \cdot, \cdot \rangle$ whose elements are functions $f : E \to \mathbb{R}$ and which has continuous point evaluations. By Riesz's representation theorem the evaluation at point $t \in E$ has a representative δ_t. We define a map $K : E \times E \to \mathbb{R}$ by

$$K(s,t) := \langle \delta_s, \delta_t \rangle = \delta_t(s) = \delta_s(t).$$

By definition, K has the properties that

(H1) $K(\cdot, t)$ is an element of H for each $t \in E$,

(H2) $\langle h, K(\cdot, t) \rangle = h(t)$ holds for every $h \in H$ (reproducing property).

Because each point evaluation has a unique representative, K is the only map having both properties with regard to H.

Definition 2.35. Let H be a Hilbert space of functions $f : E \to \mathbb{R}$ and the point evaluations being continuous functionals on H. Then the uniquely determined map K which fulfills (H1) and (H2) is called the *reproducing kernel*. The space H is called a *reproducing kernel Hilbert space* (RKHS).

We recall the following definition:

Definition 2.36. A symmetric map $K : E \times E \to \mathbb{R}$ is called *positive semi-definite* if for all $n \in \mathbb{N}$, $t_1, ..., t_n \in E$ and $c_1, ..., c_n \in \mathbb{R}$ it holds true that

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(t_i, t_j) \geq 0.$$

Clearly a reproducing kernel is symmetric. That it is positive semi-definite is easy to see, as well: For $c_1, ..., c_n \in \mathbb{R}$ and $t_1, ..., t_n \in E$ we have

$$\sum_{i,j=1}^{n} c_i c_j K(t_i, t_j) = \sum_{i,j=1}^{n} c_i c_j \left\langle K(\cdot, t_i), K(\cdot, t_j) \right\rangle = \left\langle \sum_{i=1}^{n} c_i K(\cdot, t_i), \sum_{j=1}^{n} c_j K(\cdot, t_j) \right\rangle \geq 0.$$

This leads to the question if each positive semi-definite map is the reproducing kernel of some Hilbert space with continuous point evaluations. And indeed we can answer this question positively and consequently have a one-to-one relation between positive semi-definite mappings and reproducing kernel Hilbert spaces.

Proposition 2.37. *Let $K : E \times E \to \mathbb{R}$ be positive semi-definite. Then there exists a unique Hilbert space $(H(K), \langle \cdot, \cdot \rangle_K)$ with $H(K) \subseteq \mathbb{R}^E$ such that (H1) and (H2) hold true for $H = H(K)$.*

Proof sketch. See e.g. [R00, Prop. III.1, p. 34] for a complete proof. Here we only outline the key points to proof existence. One begins by considering the space

$$H_0(K) := \operatorname{span}\{K(\cdot, t) : t \in E\}$$

of finite linear combinations and defining a bilinear form

$$\left\langle \sum_{i=1}^{n} \lambda_i K(\cdot, t_i), \sum_{j=1}^{n} \mu_j K(\cdot, s_j) \right\rangle_0 := \sum_{i,j=1}^{n} \lambda_i \mu_j K(t_i, s_j)$$

on $H_0(K)$.

For the vector space $(H_0(K), \langle \cdot, \cdot \rangle_0)$ one easily verifies (H1) and (H2), but $H_0(K)$ will in general not be complete. In order to reach $H(K) \subseteq \mathbb{R}^D$ it is not sufficient to argue that every normed vector space can be completed. Instead one constructs $H(K)$ explicitly as the set of pointwise limits of Cauchy sequences from $H_0(K)$. Then one can verify the space $H(K)$ to be complete w.r.t to the bilinear form given by

$$\langle h, g \rangle_K := \lim_{n \to \infty} \langle h_n, g_n \rangle_0$$

where $(g_n)_{n \in \mathbb{N}}$ and $(h_n)_{n \in \mathbb{N}}$ denote Cauchy sequences from $H_0(K)$ and g, h their pointwise limits. That (H1) and (H2) hold true for $H(K)$ now is easy to see. □

Example 2.38. An easy example to familiarize oneself with the concept of reproducing kernel Hilbert spaces provides the Euclidean space $\mathbb{R}^n$, which can be recovered as a RKHS of functions $f : E \to \mathbb{R}$ with domain $E = \{1, ..., n\}$ for the kernel

$$K : E \times E \to \mathbb{R}, \quad (i, j) \mapsto \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}.$$

2.3.2 Scaled Sums of RKHS

In the following we consider sums and rescalings of reproducing kernels and examine how the Hilbert spaces we obtain this way are related to the original ones.

We begin by considering for a reproducing kernel $K : E^2 \to \mathbb{R}$ the rescaled variant γK with $\gamma > 0$. Obviously the sets $H_0(K)$ and $H_0(\gamma K)$ are equal. A simple calculation reveals the scalar products, which we obtain in accordance to the proof of Proposition 2.37, to be related as follows:

$$\langle \cdot, \cdot \rangle_{\gamma K} = \frac{1}{\gamma} \langle \cdot, \cdot \rangle_K .$$

We derive

Lemma 2.39. *$H(\gamma K) = H(K)$ as sets and $(H(\gamma K), \gamma^{-1} \langle \cdot, \cdot \rangle_K) \simeq (H(K), \langle \cdot, \cdot \rangle_K)$ via the isometry $f \mapsto \gamma^{-1/2} f$.*

Proof. The set $H_0(K)$ has the same completion w.r.t $\langle\cdot,\cdot\rangle_{\gamma K}$ and $\langle\cdot,\cdot\rangle_K$, i.e. $H(\gamma K) = H(K)$ as sets. That $f \mapsto \gamma^{-1/2} f$ is an isometry is easy to see. □

Assume now we are given reproducing kernels $K_1, ..., K_n$ defined on the common domain E^2. Obviously, the following holds true:

Lemma 2.40. *The map $K(x,y) := \sum_{i=1}^n K_i(x,y)$ is positive semi-definite, as well.*

The Hilbert space we obtain from K as in Lemma 2.40 is closely related to the direct sum of the RKHS that correspond to the kernels K_i. We put this more precisely with the next proposition. For this purpose we define the mapping

$$\Phi : \bigoplus_{i=1}^n H(K_i) \to \sum_{i=1}^n H(K_i) \subseteq \mathbb{R}^E, \quad (h_1, ..., h_n) \mapsto \sum_{i=1}^n h_i. \tag{2.25}$$

Note that the notation $\sum_{i=1}^n H(K_i)$ stands for the vector space of functions from E into the reals which we obtain if we take pointwise sums of functions from the spaces $H(K_i)$.

Proposition 2.41 (Sums of RKHS)**.** *Let $K_1, ..., K_n$ be reproducing kernels on a common domain E^2 and $K = \sum_{i=1}^n K_i$. Then we have*

$$H(K) = \sum_{i=1}^n H(K_i)$$

as sets. The norm of $f \in H(K)$ is given by

$$\|f\|_K^2 = \sum_{i=1}^n \|f_i'\|_{K_i} = \min\{\sum_{i=1}^n \|f_i\|_{K_i}^2 : f = \sum_{i=1}^n f_i \text{ and } f_i \in H(K_i)\}$$

where $(f_i')_{i=1,...,n} \in (\ker\Phi)^\perp$ with $f = \sum_{i=1}^n f_i'$.

Proof. For the case $n = 2$ see e.g. [R00, Lem. III.2, p. 35]. For arbitrary $n \in \mathbb{N}$ the assertion follows by induction. □

Remark 2.42. If $\ker\Phi = \{0\}$ every $f \in H(K)$ has a unique representation as a sum $f = \sum_{i=1}^n f_i$ with $f_i \in H(K_i)$. Consequently, we have

$$\|f\|_K^2 = \sum_{i=1}^n \|f_i\|_{K_i}^2$$

and each $H(K_i)$ is a closed subspace of $H(K)$. Hence we may identify $H(K)$ with $\bigoplus_{i=1}^n H(K_i)$.

Remark 2.43. In the situation of Proposition 2.41, assume $\ker\Phi = \{0\}$ and consider the kernel given by $\widetilde{K}(x,y) := \sum_{i=1}^n \gamma_i K_i(x,y)$ with $\gamma_i > 0$. Then

$$H(K) = \bigoplus_{i=1}^n H(K_i) \simeq \bigoplus_{i=1}^n H(\gamma_i K_i) = H(\widetilde{K})$$

where isomorphy is induced by the isometry

$$\iota : H(K) \to H(\widetilde{K}), \sum_{i=1}^{n} h_i \mapsto \sum_{i=1}^{n} \sqrt{\gamma_i} h_i.$$

That is, depending on how we rescaled each summand, we obtain a distorted version of $H(K)$. Moreover, as $H(K_i)$ and $H(\gamma_i K_i)$ coincide as sets, we can immediately conclude that $H(K)$ and $H(\widetilde{K})$ are equal as sets, as well. We even get that $H_0(\widetilde{K})$ is also dense in $H(K)$.

Corollary 2.44. *Let K, L be reproducing kernels on E^2. If $cL - K$ is positive semi-definite for some $c > 0$ we have*

$$H(K) \subseteq H(L)$$

and on $H(K)$ the norms fulfill

$$\|\cdot\|_L \leq c\|\cdot\|_K.$$

Proof. Follows immediately from Proposition 2.41 since $cL = K + M$ where $M = cL - K$. □

2.3.3 Tensor Products of RKHS

We begin this subsection with a general result:

Lemma 2.45 (Schur's lemma)**.** *Let $K_1, ..., K_n$ be reproducing kernels on a common domain E^2. Then the map K given by $K(x, y) = \prod_{i=1}^{n} K_i(x, y)$ is positive semi-definite, as well.*

Proof. A proof of 2.45 is given in [A50, Section VIII]. □

Within this thesis we only deal with products of reproducing kernels where each kernel actually depends on a different set of variables. For the sake of simplicity we treat only products of two kernels below, but by induction these results carry over to products of arbitrary finite numbers $n \in \mathbb{N}$ of kernels.

Definition 2.46. Let $E = E_1 \times E_2$ where E_1, E_2 are arbitrary non-empty sets. Furthermore assume we are given reproducing kernels $K_1 : E_1^2 \to \mathbb{R}$ and $K_2 : E_2^2 \to \mathbb{R}$. Then we write

$$K = K_1 \otimes K_2$$

for the kernel $K : E^2 \to \mathbb{R}$ given by

$$K((x_1, x_2), (y_1, y_2)) = K_1(x_1, y_1) \cdot K_2(x_2, y_2).$$

Lemma 2.47. *Let K_1, K_2 as above. Further let $(e_i)_{i\in\mathbb{N}}$, $(f_j)_{j\in\mathbb{N}}$ denote orthonormal bases (ONB) of $H(K_1)$ and $H(K_2)$ respectively. Then $(e_i \otimes f_j)_{(i,j)\in\mathbb{N}^2}$ yields an ONB of $H(K_1 \otimes K_2)$.*

Proof. For $i,j \in \mathbb{N}$ let $\alpha_{i,j} \in \mathbb{R}$ such that $\sum_{i,j} |\alpha_{i,j}|^2 < \infty$. Since

$$e_i \otimes f_j(x_1,x_2) = \langle e_i, K_1(\cdot,x)\rangle_{K_1} \left\langle f_j, K_2(\cdot,y)\right\rangle_{K_2}$$

the Hölder inequality yields for every $(x_1,x_2) \in E$ that

$$\begin{aligned}\left(\sum_{i,j} |\alpha_{i,j} \cdot e_i \otimes f_j(x_1,x_2)|\right)^2 &\leq \sum_{i,j} \alpha_{i,j}^2 \sum_{i,j} \langle e_i, K_1(\cdot,x_1)\rangle_{K_1}^2 \left\langle f_j, K_2(\cdot,x_2)\right\rangle_{K_2}^2 \\ &= \sum_{i,j} \alpha_{i,j}^2 K_1(x_1,x_1)K_2(x_2,x_2).\end{aligned}$$

Consequently $\sum_{i,j} \alpha_{i,j} \cdot e_i \otimes f_j$ is a real valued function.

From

$$\sum_{i,j} \alpha_{i,j} \cdot e_i \otimes f_j(x_1,x_2) = \sum_i e_i(x_1) \cdot \sum_j \alpha_{i,j} \cdot f_j(x_2)$$

and

$$\sum_i \left(\sum_j \alpha_{i,j} \cdot f_j(x_2)\right)^2 \leq \sum_i \sum_j \alpha_{i,j}^2 K_2(x_2,x_2)$$

we derive that $\sum_{i,j} \alpha_{i,j} \cdot e_i \otimes f_j = 0$ implies $\alpha_{i,j} =$ for all $i,j \in \mathbb{N}$.

Hence the set

$$H := \left\{ \sum_{i,j} \alpha_{i,j} e_i \otimes f_j : \alpha_{i,j} \in \mathbb{R} \text{ and } \sum_{i,j} |\alpha_{i,j}|^2 < \infty \right\}$$

equipped with the scalar product given by

$$\left\langle \sum_{i,j} \alpha_{i,j} \cdot e_i \otimes f_j, \sum_{i,j} \beta_{i,j} \cdot e_i \otimes f_j \right\rangle = \sum_{i,j} \alpha_{i,j} \cdot \beta_{i,j}$$

yields a Hilbert space of real valued functions with ONB $(e_i \otimes f_j)_{(i,j)\in\mathbb{N}^2}$. It remains to verify that H is the reproducing kernel Hilbert space of $K_1 \otimes K_2$. But this is easy to see: For $(x,y) \in E$ the choice $\alpha_{i,j} = e_i(x)f_j(y)$ shows $K_1 \otimes K_2(\cdot,(x,y)) \in H$. Furthermore for $h = \sum_{i,j} \beta_{i,j} e_i \otimes f_j$ we have

$$\langle h, K_1 \otimes K_2(\cdot,(x,y))\rangle = \sum_{i,j} \beta_{i,j} e_i(x) f_j(y) = h((x,y)).$$

Hence the reproducing property follows for $K_1 \otimes K_2$.

□

In our particular situation the following holds true:

Proposition 2.48. *Let K_1, K_2 be reproducing kernels on E_1^2 and E_2^2 respectively. Then for $K = K_1 \otimes K_2$ we have*

$$H(K) = H(K_1) \otimes H(K_2).$$

For elementary tensors $f = f_1 \otimes f_2$ the norm is given by $\|f\|_K = \|f_1\|_{K_1} \|f_2\|_{K_2}$.

Proof. See [A50, Section VIII]. There the interested reader does also find results for general products of reproducing kernels. □

2.3.4 Restrictions of Reproducing Kernels

Proofs for the following are found in [HMNR10, Appendix A]. Let $E = E_1 \times E_2$ as in the previous subsection. Further fix $a \in E_2$ and let $\Psi : \mathbb{R}^E \to \mathbb{R}^E$ denote the linear mapping given by

$$(\Psi f)(x_1, x_2) = f(x_1, a)$$

for $x_1 \in E_1, x_2 \in E_2$.

Now assume we are given a reproducing kernel $K : E^2 \to \mathbb{R}$ and consider the kernels $J : E^2 \to \mathbb{R}$ and $L : E_1^2 \to \mathbb{R}$ given by

$$J((x_1, x_2), (y_1, y_2)) = K((x_1, a), (y_1, a))$$

and

$$L(x_1, y_1) = K((x_1, a), (y_1, a))$$

respectively. Then the following holds true:

Lemma 2.49. *We have*

$$\{\Psi f : f \in H(K) \wedge \|f\|_K \leq 1\} = \{g \in H(J) : \|g\|_J \leq 1\}.$$

Lemma 2.50. *We have*

$$H(J) = \{f : E \to \mathbb{R} : \exists g \in H(L)\ \forall x_2 \in E_2 : f(\cdot, x_2) = g\}.$$

In particular, for $f \in H(J)$ and $g(\cdot) := f(\cdot, x_2)$ we have $g \in H(L)$ and $\|g\|_L = \|f\|_J$.

2.4 Rank-1 Lattice Rules

Rank-1 lattice rules are deterministic quadrature rules which have been widely studied for finite-dimensional integration problems of the form

$$\int_{[0,1]^d} f(\mathbf{x}) \mathrm{d}\mathbf{x}, \quad f \in F$$

for different classes F of functions on the d-dimensional unit cube $[0,1]^d$ and integration w.r.t. the uniform distribution. Their naming is due to their point set forming a *rank-1 lattice*.

Let us fix $D = [0,1]$ for this section.

2.4.1 Rank-1 Lattices

We give a brief introduction to rank-1 lattices in the following and refer the interested reader to [SJ94] for a more detailed treatise of construction and properties of rank-1 lattices. Also classical analysis of error bounds can be found there.

Definition 2.51 (Rank-1 lattice)**.** Let $n \in \mathbb{N}$ be a prime number. A *rank-1 lattice* is a point set $P_n = \{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$ in the d-dimensional unit cube $[0,1]^d$ with

$$\mathbf{x}_j^{(i)} = \frac{i \cdot z_j \bmod n}{n}$$

for a vector $\mathbf{z}$ with $z_j \in \{1, ..., n-1\}$. The vector $\mathbf{z}$ is called the *generator* of the lattice.

We use the notation $\{\cdot\}$ below to indicate that we take (component-wise) the fractional part of the argument. This mapping composed with ordinary addition turns a rank-1 lattice P_n into a finite Abelian group: For $\mathbf{x}^{(k)}, \mathbf{x}^{(l)} \in P_n$ we have

$$\{\mathbf{x}^{(k)} + \mathbf{x}^{(l)}\} = \mathbf{x}^{(k+l \bmod n)} \in P_n.$$

Remark 2.52. The term *rank-1* refers to the fact that P_n can be decomposed into only *one* cyclic subgroup, namely P_n itself.

Definition 2.53 (Shifted rank-1 lattice)**.** Let P_n denote a rank-1 lattice and let $\Delta \in [0,1]^d$. Then we obtain a *shifted rank-1 lattice* $\widetilde{P}_n = \{\tilde{\mathbf{x}}^{(1)}, ..., \tilde{\mathbf{x}}^{(n)}\}$ by

$$\tilde{\mathbf{x}}_j^{(i)} = \{\mathbf{x}_j^{(i)} + \Delta\}.$$

The lattice points of P_n and $\widetilde{P}_n$ differ only in their absolute position. The relative positions the points have to each other are equal modulo 1.

The classical motivation to study rank-1 lattices as point sets for quadrature rules is that they have a so called *low discrepancy* for certain generators. The discrepancy of a point set is a quantity that measures the deviation from the uniform distribution. The lower the discrepancy is the more the point set looks like being uniformly distributed.

There are a number of different discrepancies defined in the literature. One important variant is given by the so called L_2 *variant of the star discrepancy*. It is defined as follows: Let for a point set P_n and $A \subseteq [0,1]^d$ the local discrepancy be given by

$$\operatorname{discr}(A, P_n) := \frac{|P_n \cap A|}{n} - \operatorname{vol}(A).$$

Then the L_2 variant of the star discrepancy is defined by

$$D_2^\star(P_n) := \left(\int_{[0,1]^s} |\operatorname{discr}([\mathbf{0}, \mathbf{x}], P_n)|^2 \mathrm{d}\mathbf{x} \right)^{1/2},$$

where $[\mathbf{0}, \mathbf{x}] = [0, x_1] \times ... \times [0, x_d]$.

If $P_n = \{\mathbf{X}_1, ..., \mathbf{X}_n\}$ with $(\mathbf{X}_i)_{i=1,..,n}$ a family of i.i.d. random variables and $\mathbf{X}_1$ uniformly distributed on $[0,1]^d$, then its discrepancy is at most of the order $1/2$ in n on the average, that is $\mathrm{E}[D_2^\star(P_n)] = O(n^{-1/2})$. However, if $P_n = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ is a low-discrepancy point set we have $D_2^\star(P_n) = O(n^{-1}(\log n)^d)$, which is now even a deterministic bound. If the class F of integrands is now chosen properly these bounds become upper bounds for the randomized worst-case error.

Since low-discrepancy point sets are deterministic but behave like uniformly distributed point sets to some extent, quadrature rules using such point sets are also referred to as *quasi-Monte Carlo rules*.

Remark 2.54. Generally, if we want to emphasize that we generate integration points by means of a low-discrepancy point set we use the terminology *quasi-Monte Carlo integration*.

2.4.2 Rank-1 Lattice Rules for Weighted Korobov Spaces

A more recent line of research studies rank-1 lattice rules for finite-dimensional integration problems as above where the class F of integrands is the unit ball of weighted Korobov spaces. The facts given in this subsection are mainly found in [NW10] which provides a comprehensive survey of this topic.

Recall that the *Fourier coefficients* of an integrable function $f : [0,1]^d \to \mathbb{C}$ are given by

$$\hat{f}(\mathbf{h}) = \int_{[0,1]^d} \exp(-2\pi i \mathbf{h}^T \mathbf{x}) f(\mathbf{x}) \mathrm{d}\mathbf{x}, \quad \mathbf{h} \in \mathbb{Z}^d.$$

Furthermore, for $\boldsymbol{\gamma} = (\gamma_{d,u})_{d \in \mathbb{N}, u \subseteq 1:d}$ with $\gamma_{d,u} > 0$ and $\alpha > 1/2$ we put

$$\varrho_{d,\alpha,\boldsymbol{\gamma}}(\mathbf{h}) = \frac{1}{\gamma_{d,u_\mathbf{h}}} \prod_{j \in u_\mathbf{h}} |2\pi h_j|^{2\alpha}$$

for $\mathbf{h} \in \mathbb{Z}^d$ and $u_\mathbf{h} = \{j \in 1:d \mid h_j \neq 0\}$.

Now consider the set

$$H_{d,\alpha,\boldsymbol{\gamma}} := \left\{ f : [0,1]^d \to \mathbb{C} : f \text{ is periodic and } \sum_{\mathbf{h} \in \mathbb{Z}^d} \varrho_{d,\alpha,\boldsymbol{\gamma}}(h) |\hat{f}(\mathbf{h})|^2 < \infty \right\}.$$

Then we have

Proposition 2.55. *The set $H_{d,\alpha,\boldsymbol{\gamma}}$ as given above is a Hilbert space with norm given by*

$$\|f\|^2_{H_{d,\alpha,\boldsymbol{\gamma}}} = \sum_{\boldsymbol{h} \in \mathbb{Z}^d} \varrho_{d,\alpha,\boldsymbol{\gamma}}(h) |\hat{f}(\boldsymbol{h})|^2.$$

Moreover, the space $H_{d,\alpha,\boldsymbol{\gamma}}$ is a reproducing kernel Hilbert space with kernel $K_{d,\alpha,\boldsymbol{\gamma}}$ on D^{2d} given by

$$K_{d,\alpha,\boldsymbol{\gamma}}(\mathbf{x},\boldsymbol{y}) = \sum_{u \subseteq 1:d} \gamma_{d,u} \prod_{j \in u} \left(\frac{2}{(2\pi)^{2\alpha}} \sum_{h=1}^{\infty} \frac{\cos(2\pi h(x_j - y_j))}{h^{2\alpha}} \right). \tag{2.26}$$

Definition 2.56 (Weighted Korobov Space)**.** The space $H_{d,\alpha,\gamma}$ as given in the previous proposition is called the *weighted Korobov space* with weights $\boldsymbol{\gamma}$ and parameter α. Furthermore, we call $K_{d,\alpha,\gamma}$ the *Korobov kernel* associated with $H_{d,\alpha,\gamma}$.

Remark 2.57. For $x \in [0,1]$ it holds true that

$$B_2(x) := x^2 - x + \frac{1}{6} = \frac{1}{\pi^2} \sum_{h=1}^{\infty} \frac{\cos(2\pi x)}{h^2}. \tag{2.27}$$

Hence in the case $\alpha = 1$ we may write $K_{d,1,\gamma}$ also in the form

$$K_{d,1,\gamma}(\mathbf{x},\mathbf{y}) = \sum_{u \subseteq 1:d} \frac{\gamma_{d,u}}{2^{|u|}} \prod_{j \in u} B_2(\{x_j - y_j\}).$$

If we take the unit ball $B(K_{d,\alpha,\gamma})$ as the class F for the integration problem on the unit cube $[0,1]^d$ the following proposition reveals that there are deterministic quadrature rules whose worst-case error declines with order arbitrarily close to α in the number of integration points n.

Proposition 2.58. *Let $F = B(H_{d,\alpha,\gamma})$ and denote by $P_n(\mathbf{z})$ a rank-1 lattice with generator $\mathbf{z}$. Then for all $d \in \mathbb{N}$ and for all $\tau \in [1/2, \alpha)$ there exists a constant $C(d,\tau) > 0$ such that for all $n \in \mathbb{N}$ there is a generator $\mathbf{z}^*$ such that the corresponding quadrature rule satisfies*

$$e(Q(\cdot; P_n(\mathbf{z}^*)), F) \leq C(d,\tau)(n-1)^{-\tau}.$$

With $\mathbf{z}^*$ as in the previous proposition we refer to rank-1 lattices $P_n(\mathbf{z}^*)$ as *good* rank-1 lattices.

Remark 2.59. With the weights $(\gamma_u)_{d \in \mathbb{N}, u \subseteq 1:d}$ properly chosen the constant $C(d,\tau)$ in Proposition 2.58 can be shown to be bounded from above for all d and τ by some positive constant.

The kernel $K_{d,\alpha,\gamma}$ obviously has the property to be *shift-invariant* which we introduce in the next definition.

Definition 2.60. A reproducing kernel $K : [0,1]^d \times [0,1]^d \to \mathbb{R}$ is said to be *shift-invariant* if

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

for all $\Delta \in [0,1]^d$.

Remark 2.61. As the particular choice $\Delta = \{-\mathbf{y}\}$ reveals a shift-invariant kernel K can be written in terms of one variable:

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

The shift-invariance of $K_{d,\alpha,\gamma}$ make the results of Proposition 2.58 especially interesting as we do not only have existence of the generator $\mathbf{z}^*$ but we are actually able to calculate it efficiently. This is due to an approach first brought up by Sloan and Reztsov [SR02] which chooses the components of $\mathbf{z}^*$ one by one. A notably efficient advancement of this approach is the *fast component-by-component (CBC) algorithm* developed by Nuyens and Cools (see [Ny07], but also [NC06a] and [NC06b]). If we take n to be the number of lattice points (where n is prime) and d the dimension of the integration points then the fast CBC algorithm allows to compute $\mathbf{z}^*$ at cost $O(dn\log(n))$ using $O(n)$ memory. We elaborate the algorithm in Subsection 4.3.2 when we discuss our C++ library.

The previous reasoning shows that shift-invariance is crucial from a practical point of view. However, as we see in the following it is possible to utilize the results gained in the case of shift-invariance to some extent in a non-shift-invariant setting, as well.

Lemma 2.62. *Let $K : [0,1]^{2d} \to \mathbb{R}$ be a reproducing kernel. Then the map K^{shinv} defined by*

$$K^{shinv}(\boldsymbol{x},\boldsymbol{y}) := \int_{[0,1)^d} K(\{\boldsymbol{x}+\Delta\},\{\boldsymbol{y}+\Delta\})\mathrm{d}\Delta. \tag{2.28}$$

yields a reproducing kernel, as well. Furthermore K^{shinv} is shift-invariant.

Definition 2.63. Let K be a reproducing kernel. Then we call the kernel K^{shinv} which we obtain by the previous lemma the *associated shift-invariant kernel* of K.

Now, given a kernel K and its associated shift-invariant kernel K^{shinv}, we obtain the following relation for quadrature rules using rank-1 lattices and randomly shifted rank-1 lattices, respectively, on the spaces $H(K)$ and $H(K^{shinv})$.

Proposition 2.64. *Let P_n be a rank-1 lattice. Furthermore, let P_n^Δ denote the shifted version of P_n with shift $\Delta \in [0,1]^d$. If Δ is a uniformly distributed random variable then*

$$\mathrm{E}[e^2\left(Q(\cdot,P_n^\Delta),B(K)\right)] = e^2\left(Q(\cdot,P_n),B(K^{shinv})\right).$$

Proof. See [Ny07, Theorem 2.22]. □

If we have a good lattice rule for $B(K^{shinv})$ the previous result gives us at least the existence of a good lattice rule for $B(K)$ reaching the same order of worst-case error decay through an averaging argument. From a practical perspective such a pure existential statements is useless. However, we may also interpret Proposition 2.64 in the way that we can construct a randomized quadrature rule using a rank-1 lattice with random shift for $B(K)$ which is on the average at least as good as the deterministic rule using the same unshifted rank-1 lattice on $B(K^{\mathrm{shinv}})$. This motivates the following definition:

Definition 2.65 (Shifted Rank-1 Lattice Rule)**.** Let P_n denote a shifted rank-1 lattice where the shift is a random variable uniformly distributed on $[0,1]^d$. Then we call the randomized quadrature rule $Q(\cdot;P_n)$ a *shifted rank-1 lattice rule*.

Definition 2.66 (Multilevel Shifted Rank-1 Lattice Rule)**.** If we provide the multilevel algorithm (2.16) with shifted rank-1 lattice rules as building blocks then we call it a *multilevel shifted rank-1 lattice rule*.

Remark 2.67. We will explicitly exploit the results above in Section 3.2 in the situation of *anchored weighted Sobolev spaces*. These are reproducing kernel Hilbert spaces with kernels such that their associated shift-invariant kernels are precisely Korobov kernels.

3 Multilevel Rank-1 Lattice Rules as Optimal QMC Algorithms

Niu et al.([NHMR10]) specify a class of reproducing kernel Hilbert spaces $H(K)$ with product weights whose unit balls yield suitable function spaces such that the integration problem 2.12 becomes tractable in the worst-case setting. As cost models both the fixed subspace sampling regime 2.24 and the variable subspace sampling regime 2.25 are analyzed. In the latter case quasi-Monte Carlo rules combined with the multilevel technique outperform classical QMC rules.

In this chapter we firstly present the construction idea of these Hilbert spaces in detail. Thereby we follow Hickernell et al.([HMNR10]) and add a proof that the considered Hilbert spaces feature the properties stated by Hickernell et al. in the case of functions in infinitely many variables. Next we put emphasis on a particular application where $H(K)$ is recovered as an anchored weighted Sobolev space. Finally, we briefly summarize the tractability and optimality results given by [NHMR10] for the specific situation of $H(K)$ being the anchored weighted Sobolev space.

3.1 Construction of the Function Space

Except for subsection 3.1.2 the following is basically a reformulation of [HMNR10, Section 2].

Let us briefly recall the setting of our integration problem. We have a Borel measure ρ over some Borel set $D \subset \mathbb{R}$ and consider the product measure $\mu = \otimes_{j\in\mathbb{N}}\rho$. Then integration is with respect to this product measure.

Now assume we are given a reproducing kernel $k : D \times D \to \mathbb{R}$ which the following is fulfilled for:

(A1) $k \neq 0$ is measurable w.r.t. $\mathfrak{B}(D^2)$

(A2) the only constant function in $H(k)$ is the zero function: $H(k) \cap H(1) = \{0\}$

(A3) we have $\int_D k(x,x)\rho(\mathrm{d}x) < \infty$

As k is positive semi-definite, (A3) ensures k to be $\rho \otimes \rho$-integrable.

Besides the kernel we assume that we are given a sequence of weights $(\gamma_j)_{j\in\mathbb{N}}$ with

(A4) $\gamma_1 \geq \gamma_2 \geq \ldots > 0, \quad \sum_{j=1}^{\infty} \gamma_j < \infty.$

For finite sets $u \subset \mathbb{N}$ we set $\gamma_u := \prod_{j\in u} \gamma_j$. This means in particular $\gamma_\emptyset = 0$. The role of the weights becomes clear in Subsection 3.2.2 where we treat tractability issues.

In order to construct the function space our first step is the choice of an appropriate domain. We will see in 3.1.2 why we choose

$$\mathfrak{X} = \left\{ x \in D^{\mathbb{N}} : \sum_{j=1}^{\infty} \gamma_j k(x_j, x_j) < \infty \right\}. \tag{3.1}$$

instead of the whole set $D^{\mathbb{N}}$.

Lemma 3.1. *It holds true that* $\mu(\mathfrak{X}) = 1$.

3.1.1 Construction for Finitely Many Variables

For $u \in P_0(\mathbb{N})$ we define

$$k_u : \mathfrak{X} \times \mathfrak{X} \to \mathbb{R},\ (\mathbf{x}, \mathbf{y}) \mapsto \prod_{j \in u} k(x_j, y_j). \tag{3.2}$$

That is, $k_u = (\bigotimes_{j \in u} k) \otimes m_c$ where m_c is the constant kernel on $(D^{\mathbb{N}\setminus u})^2$. With Proposition 2.48 and Lemma 2.50 we conclude

$$H(k_u) = \{ f : \mathfrak{X} \to \mathbb{R} \mid \exists g \in \bigotimes_{j \in u} H(k)\ \forall \mathbf{x} \in \mathfrak{X} :\ f(\cdot, \mathbf{x}_{-u}) = g \}.$$

We introduce the shorthand notation $H_u := H(k_u)$, which means in particular $H_\emptyset = H(1)$. Obviously, functions in H_u depend only on the variables x_u.

Remark 3.2. Generally, if κ is a kernel on $\mathfrak{X}^2$ actually depending only on variables with index $u \in P_0(\mathbb{N})$, this property carries over to the functions in $H(\kappa)$. It is a immediate consequence of the reproducing property. For $\mathbf{x}, \mathbf{y} \in \mathfrak{X}$ with $\mathbf{x}_u = \mathbf{y}_u$ and $f \in H(\kappa)$ we find

$$f(\mathbf{x}) = \langle f, \kappa(\cdot, (\mathbf{x}_u, \mathbf{x}_{-u})) \rangle_\kappa = \langle f, \kappa(\cdot, (\mathbf{x}_u, \mathbf{y}_{-u}) \rangle_\kappa = f(\mathbf{x}_u, \mathbf{y}_{-u}) = f(\mathbf{y}).$$

For finite $v \subset \mathbb{N}$ Lemma 2.40 yields that the map K_v given by

$$K_v(\mathbf{x}, \mathbf{y}) := \sum_{u \subseteq v} \gamma_u k_u(\mathbf{x}, \mathbf{y}) \tag{3.3}$$

defines a reproducing kernel. The so obtained Hilbert space $H(K_v)$ consists of functions in the variables x_v. We even have that each function in $H(K_v)$ has a unique orthogonal decomposition into functions from the spaces H_u.

Lemma 3.3. *Given distinct finite subsets* $u_1, \ldots, u_n \subset \mathbb{N}$, *let* $f_1 \in H_{u_1}, \ldots, f_n \in H_{u_n}$ *such that* $\sum_{i=1}^n f_i = 0$. *Then we have* $f_i = 0$ *for each* $i = 1, \ldots, n$.

Proof. See [HMNR10, Section 2, Lem. 3]. We note that assumption (A2) is crucial here. The key argument is that for fixed $l \in \bigcup_{i=1}^n u_i \setminus \bigcap_{i=1}^n u_i$ the assumption $\sum_{i=1}^n f_i = 0$ implies

$$\sum_{i \in I_1} f_i = \sum_{i \in I_2} f_i = 0 \tag{3.4}$$

with $I_1 = \{i : l \in u_i\}$ and $I_2 = \{i : l \notin u_i\}$. This allows to conclude by induction. □

Proposition 3.4. *For the kernel K_v it holds true that*

$$H(K_v) = \bigoplus_{u \subseteq v} H_u \tag{3.5}$$

where the norm is given by

$$\|f\|_{K_v}^2 = \sum_{u \subseteq v} \gamma_u^{-1} \|f_u\|_u^2. \tag{3.6}$$

Note that following the shorthands introduced above we wrote $\|\cdot\|_u$ for $\|\cdot\|_{k_u}$.

Proof. Lemma 3.3 assures that the map Φ given by (2.25) with $K_i = k_{u_i}$ is injective. Now Proposition 2.41 yields the desired result. □

The space $H(K_v)$ has also a tensor product structure in the sense that

$$H(K_v) = \{f : \mathfrak{X} \to \mathbb{R} \mid \exists g \in \bigotimes_{j \in v} H(1 + \gamma_j k) \; \forall \mathbf{x} \in \mathfrak{X} : \; f(\cdot, \mathbf{x}_{-v}) = g\}, \tag{3.7}$$

the norm for fundamental tensors $f(\cdot, \mathbf{x}_{-v}) = \otimes_{j \in v} g_j$ being given by

$$\|f\|_{K_v} = \prod_{j \in v} (1 + \gamma_j^{-1} \|g_j\|_k).$$

With Proposition 2.48 and Lemma 2.50, the above is a direct consequence of

$$\sum_{u \subseteq v} \gamma_u k_u(\mathbf{x}, \mathbf{y}) = \prod_{j \in v} (1 + \gamma_j k(x_j, y_j))$$

for arbitrary $\mathbf{x}, \mathbf{y} \in \mathfrak{X}$, which can be shown by an easy calculation.

3.1.2 Extension to Countably Many Variables

We want to extend the construction principle from the previous subsection to countably many variables. This turns out to be none-straightforward.

At first we have to show that the map K given by

$$K(\mathbf{x}, \mathbf{y}) := \sum_{u \in P_0(\mathbb{N})} \gamma_u k_u(\mathbf{x}, \mathbf{y}) = \lim_{d \to \infty} \sum_{u \subseteq 1:d} \gamma_u k_u(\mathbf{x}, \mathbf{y}) \tag{3.8}$$

converges pointwise for $\mathbf{x}, \mathbf{y} \in \mathfrak{X}$. This follows from the lemma below.

Lemma 3.5. *For $\boldsymbol{x}, \boldsymbol{y} \in \mathfrak{X}$ holds true that $\sum_{u \in P_0(\mathbb{N})} \gamma_u |k_u(\boldsymbol{x}, \boldsymbol{y})| < \infty$.*

Proof. See [HMNR10, Section 2, Lemma 5]. □

Positive semi-definiteness of K is now easy to verify. Consequently we actually obtain a reproducing kernel.

In a next step we would like to identify $H(K)$ and the direct Hilbert sum $\bigoplus_{u\in P_0(\mathbb{N})} H(\gamma_u k_u)$ via the mapping

$$\varsigma : \bigoplus_{u\in P_0(\mathbb{N})} H(\gamma_u k_u) \to \sum_{u\in P_0(\mathbb{N})} H(\gamma_u k_u), \quad (f_u)_u \mapsto \sum_u f_u. \tag{3.9}$$

Comparing our approach in the finite case we encounter two obstacles now. First, it is not immediately clear if Proposition 2.41 carries over to countable sums of kernels. Second, the more severe shortcoming is that Lemma 3.3 can not be carried over to countably many distinct sets $u \in P_0(\mathbb{N})$. This would be necessary to obtain injectivity of ς in the way as we did in the finite case. That Lemma 3.3 can not be applied here is due to the crucial argument in its proof being inductive.

Remark 3.6. In the proof of Lemma 3.3 one might be tempted to establish a specific well-ordering on the set $\mathscr{U}$ of all $(u_i)_{i\in\mathscr{I}}$ of the form such that $\mathscr{I} \subseteq \mathbb{N}$ and the u_i's are distinct finite subsets of $\mathbb{N}$. This would allow to carry out the argument (3.4) also for countable I_1 and I_2 respectively.

But the set $\mathscr{U}$ is easily seen to be equivalent to the the set of real numbers $\mathbb{R}$. It is a well-known result from set theory (at least within the Zermo-Fraenkel axioms and the axiom of choice, otherwise it is only a conjecture) that on $\mathbb{R}$ it is impossible to construct a well-ordering explicitly.

The specific tensor product structure of K gives rise to another approach that gets by without coming back to Lemma 3.3. The idea is to utilize the results we already established for finitely many variables by recovering the spaces $H(K_{1:d})$ as *closed* subspaces of $H(K)$. Note that because of

$$K = K_{1:d} + \sum_{\substack{w\subseteq 1:d,\\ \emptyset\neq v\in P_0(\mathbb{N}\setminus 1:d)}} \gamma_{w\cup v} k_{w\cup v}$$

and Corollary 2.44 we already have $H(K_{1:d}) \subset H(K)$.

Lemma 3.7. *For $d \in \mathbb{N}$ the Hilbert space $H(K_{1:d})$ is a closed subspace of $H(K)$. In particular, there is a constant $c > 0$ such that*

$$\|\cdot\|_{K_{1:d}} = c\|\cdot\|_K$$

on $H(K_{1:d})$. This constant is independent of the chosen d.

Proof. Let $E_1 = D^{1:d}$ and $E_2 = \{\mathbf{x}_{-1:d} \in D^{-1:d} : \mathbf{x} \in \mathfrak{X}\}$. Further fix $\mathbf{a}_{-1:d} \in E_2$. We consider the kernel $L = \bigotimes_{j\in 1:d}(1+\gamma_j k)$ on $E_1 \times E_1$ and the kernel

$$M = \bigotimes_{j\notin 1:d}(1+\gamma_j k) = \sum_{u\in P_0(\mathbb{N}\setminus 1:d)} \gamma_u k_u$$

on $E_2 \times E_2$. Note that $K = L \otimes M$ and $K_{1:d}(\mathbf{x},\mathbf{y}) = L(\mathbf{x}_{1:d},\mathbf{y}_{1:d})$. The choice $u = \emptyset$ shows $1 \in H(M)$.

Let $f \in H(K_{1:d})$ and put $g(\mathbf{x}_{1:d}) := f(\mathbf{x}_{1:d}, \mathbf{a}_{-1:d})$. By Lemma 2.50 (with $K = J = K_{1:d}$) we have $g \in H(L)$ and $\|g\|_L = \|f\|_{K_{1:d}}$.

Now let $(e_i)_{i\in\mathbb{N}}$ and $(f_j)_{j\in\mathbb{N}}$ denote an ONB of $H(L)$ and $H(M)$ respectively, according to Lemma 2.47. W.l.o.g. assume $f_1 = c_d$ with c_d being some positive constant. We define a function h by letting $h(\mathbf{x}) = c_d g(\mathbf{x}_{1:d})$. Then, if we choose $\alpha_{i,1} = \langle g, e_i \rangle_L$ and $\alpha_{i,j} = 0$ for $j > 1$ we get

$$h = \sum_{i,j\in\mathbb{N}} \alpha_{i,j} e_i \otimes f_j.$$

Consequently, we have $h \in H(K)$ and furthermore $\|h\|_K = \|g\|_L$. With

$$K_{1:d}(\cdot, (\mathbf{x}_{1:d}, \mathbf{a}_{-1:d})) = K_{1:d}(\cdot, \mathbf{x})$$

we find that

$$h(\mathbf{x}) = \langle g \otimes f_1, L \otimes M(\cdot, \mathbf{x}) \rangle_K = c_d g(\mathbf{x}_{1:d}) = c_d f(\mathbf{x}_{1:d}, \mathbf{a}_{-1:d}) = c_d \langle f, K_{1:d}(\cdot, \mathbf{x}) \rangle_{K_{1:d}} = c_d f(\mathbf{x})$$

and thus $\|h\|_K = c_d \|f\|_K$.

For every $d \in \mathbb{N}$ we already have the direct Hilbert sum character of $H(K_{1:d})$, see Proposition 3.4. For the particular choice $f = 1$ we can therefore immediately conclude that $\|f\|_{K_{1:d}} = 1$. Hence $c_d = 1/\|f\|_K$ is independent of d. □

Equipped with the previous lemma we are now able to establish the desired structure for $H(K)$.

Theorem 3.8. *For the kernel $K = \sum_{u\in P_0(\mathbb{N})} \gamma_u k_u$ we have*

$$H(K) = \bigoplus_{u\in P_0(\mathbb{N})} H(\gamma_u k_u),$$

where the norm is given by

$$\|f\|_K^2 = \sum_{u\in P_0(\mathbb{N})} \gamma_u^{-1} \|f_u\|_u^2$$

with $f \in H(K) = \sum_{u\in P_0(\mathbb{N})} f_u$ and $f_u \in H(k_u)$.

Proof. Let $H = \bigoplus_{u\in P_0(\mathbb{N})} H(\gamma_u k_u)$ and let $\|\cdot\|$ denote the canonical norm on H. We prove the map ς defined in (3.9) to be injective (and hence bijective). Then the image $\varsigma(H)$ is shown to be a Hilbert space of real valued functions which moreover has the reproducing kernel K.

Let $f = (f_u)_u \in H$. Then for $\mathbf{x} \in \mathfrak{X}$ the inequality

$$\sum_u |f_u(\mathbf{x})| \le \sum_u \|f_u\|_u k_u(\mathbf{x}, \mathbf{x})^{1/2} \le \sum_u \gamma_u^{-1} \|f_u\|_u^2 \cdot \sum_u \gamma_u k_u(\mathbf{x}, \mathbf{x}) < \infty$$

yields that $\varsigma(f)$ is a real valued function on $\mathfrak{X}$.

For $d \in \mathbb{N}$ we define $f^{(d)} \in H$ by letting $f_u^{(d)} = f_u$ if $u \subseteq 1:d$ and $f_u^{(d)} = 0$ otherwise. Then $\lim_{d\to\infty} f^{(d)} = f$. According to Lemma 3.7 we have

$$\|f^{(d)}\| = \|\varsigma(f^{(d)})\|_{K_{1:d}} = c\|\varsigma(f^{(d)})\|_K.$$

Consequently, $(\varsigma(f^{(d)}))_{d\in\mathbb{N}}$ is a Cauchy sequence in $H(K)$. Since $\lim_{d\to\infty} \varsigma(f^{(d)})(\mathbf{x}) = \varsigma(f)(\mathbf{x})$ for all $\mathbf{x} \in \mathfrak{X}$ we conclude that $\varsigma(f) \in H(K)$ and $\|f\| = c\|\varsigma(f)\|_K$. In other words, $\varsigma(H)$ is a closed subspace of $H(K)$ and ς is injective.

It remains to show that K is the reproducing kernel of $\varsigma(H)$. For the choice $f_u = \gamma_u k_u(\cdot, \mathbf{x})$ we get $K(\cdot, \mathbf{x}) = \varsigma(f) \in \varsigma(H)$. Moreover, for $g \in H$ we obtain

$$\langle \varsigma(g), K(\cdot,\mathbf{x})\rangle_{\varsigma(H)} = \langle g, f\rangle = \sum_{u\in P_0(\mathbb{N})} \gamma_u^{-1} \langle g_u, f_u\rangle_u = \sum_{u\in P_0(\mathbb{N})} g_u(\mathbf{x}) = \varsigma(g)(\mathbf{x}).$$

□

Integration as Continuous Linear Functional

Having assumption (A3), the continuity of point evaluations makes integration

$$I(f) = \int_{\mathfrak{X}} f(\mathbf{x})\mu(\mathrm{d}\mathbf{x}).$$

a non-trivial, continuous linear functional on $H(K)$, as well. We derive this from the following lemma:

Lemma 3.9. *We have*

$$\int_{\mathfrak{X}} \|K(\cdot,\mathbf{x})\|_K \mu(\mathrm{d}\mathbf{x}) \le \int_{\mathfrak{X}} K(\mathbf{x},\mathbf{x})\mu(\mathrm{d}\mathbf{x}) < \infty$$

Proof. Put $m := \int_D k(x,x)\rho(dx)$. Due to (A3) we have $m < \infty$. Use Cauchy-Schwarz's and Hölder's inequality to get

$$\begin{aligned}\int_{\mathfrak{X}} \sqrt{K(\mathbf{x},\mathbf{x})}\mu(\mathrm{d}\mathbf{x}) &\le \int_{\mathfrak{X}} K(\mathbf{x},\mathbf{x})\mu(\mathrm{d}\mathbf{x}) \\ &= \sum_{u\in P_0(\mathbb{N})} \gamma_u m^{|u|} = \prod_{j\in\mathbb{N}} (1+\gamma_j m) \\ &\le \prod_{j\in\mathbb{N}} \exp(\gamma_j m) = \exp(m\sum_{j\in\mathbb{N}} \gamma_j) < \infty.\end{aligned}$$

□

Corollary 3.10. *Integration $I(\cdot)$ is a non-trivial, continuous linear functional on $H(K)$.*

Proof. Using Lemma 3.9 and

$$I(f) \le \|f\|_K \int_{\mathfrak{X}} \|K(\cdot,\mathbf{x})\|_K \mu(\mathrm{d}\mathbf{x})$$

we conclude that integration is continuous. Hence there is a unique representative $h \in H(K)$ with $I(f) = \langle f, h\rangle_K$. It is given by

$$h(\mathbf{x}) = \langle h, K(\cdot,\mathbf{x})\rangle_K = \int_{\mathfrak{X}} K(\mathbf{x},\mathbf{y})\mu(\mathrm{d}\mathbf{y}).$$

As $1 \in H(K)$ and $\langle 1, h\rangle_K = \mu(\mathfrak{X}) = 1$ we conclude that h is non-trivial. □

3.2 Results for the Minimum-Kernel

Let $\mathfrak{X}$ as in (3.1) and let ρ denote the uniform distribution on $D = [0,1]$. We study the particular choice of k being the *minimum kernel*

$$k(x,y) = \min(x,y), \quad x,y \in D.$$

Before we address tractability issues we verify the assumptions (A1) - (A3) for k and characterize the Hilbert spaces we obtain for the kernels k_u, $K_{1:d}$ and K as given by (3.2), (3.3) and (3.8), respectively.

3.2.1 Anchored Weighted Sobolev Spaces

Clearly, k fulfills (A1) and (A3). To show (A2) we have to characterize the space $H(k)$. It is well known that the *Sobolev space*

$$W_2^1(D) = \{f \in C(D) : f \text{ absolutely continuous}, f' \in L_2(D)\}$$

equipped with the inner product

$$\langle f_1, f_2 \rangle = f_1(0)f_2(0) + \int_0^1 f_1'(t)f_2'(t)\mathrm{d}t$$

has the reproducing kernel $R = 1 + k$, see e.g. [R00, Example 5, p. 38/39]. Note that f' means the weak derivative.

Consider now the closed subspace

$$H = \{f \in W_2^1(D) : f(0) = 0\}.$$

We immediately see that $k(\cdot, x) \in H$. For $f \in H$ we observe

$$\langle f, k(\cdot,x)\rangle = \int_0^1 f'(t)\min(t,x)'\mathrm{d}t = \int_0^x f'(t)\mathrm{d}t = f(x).$$

Hence we have $H(k) = H$. But then (A2) obviously holds true for k.

Now that we verified the minimum kernel k to fulfill (A1) - (A3), let us have a closer look at the spaces $H_u, H(K_{1:d})$ and $H(K)$.

Lemma 3.11. *The space $H_u = H(k_u)$ consists of all continuous functions f such that*

(i) $f(\boldsymbol{x})$ depends only on $\boldsymbol{x}_u$,

(ii) if $x_j = 0$ for some $j \in u$ then $f(\boldsymbol{x}) = 0$,

(iii) the weak derivative $f^{(u)} = \frac{\partial^{|u|} f}{\partial \boldsymbol{x}_u}$ exists and is square-integrable.

The norm on H_u is given by

$$\|f\|_u^2 = \int_{\mathfrak{X}} f^{(u)}(\mathbf{x})^2 \mathrm{d}\mathbf{x}.$$

Proof. The properties (i) and (ii) are obvious. To prove (iii), let $(e_i)_{i\in\mathbb{N}}$ be an ONB of H_u. By Lemma 2.47 we may assume each e_i to be a fundamental tensor $e_i = \otimes_{j\in u} g_j^i$ with $g_j^i \in H(k)$ and $\|g_j^i\| = 1$. But then for each e_i property (iii) clearly holds true, in particular

$$\int_{\mathfrak{X}} e_i^{(u)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} = 1.$$

Now let $f = \sum_{i=1}^{\infty} \alpha_i e_i$ and $\varphi \in C_c^\infty(\mathfrak{X})$ be a test function. Then

$$\begin{aligned}\int_{\mathfrak{X}} f(\mathbf{x})\varphi^{(u)}(\mathbf{x})\mathrm{d}\mathbf{x} &\overset{(1)}{=} \sum_{i\in\mathbb{N}} \alpha_i \int_{\mathfrak{X}} e_i(\mathbf{x})\varphi^{(u)}(\mathbf{x})\mathrm{d}\mathbf{x} \\ &= -\sum_{i\in\mathbb{N}} \alpha_i \int_{\mathfrak{X}} e_i^{(u)}(\mathbf{x})\varphi(\mathbf{x})\mathrm{d}\mathbf{x} \overset{(2)}{=} -\int_{\mathfrak{X}} \sum_{i\in\mathbb{N}} \alpha_i e_i^{(u)}(\mathbf{x})\varphi(\mathbf{x})\mathrm{d}\mathbf{x}\end{aligned} \tag{3.10}$$

and hence $f^{(u)} = \sum_{i=1}^{\infty} \alpha_i e_i^{(u)}$. Thereby (1), (2) hold true since

$$|\sum_{i=1}^{n} \alpha_i e_i(\mathbf{x})| \le \|f\|_u \sqrt{k_u(\mathbf{x},\mathbf{x})} \tag{3.11}$$

$$|\sum_{i=1}^{n} \alpha_i e_i^{(u)}(\mathbf{x})| \le \sum_{i=1}^{n} |\alpha_i||e_i^{(u)}(\mathbf{x})| \le \sum_{i=1}^{\infty} |\alpha_i||e_i^{(u)}(\mathbf{x})| \tag{3.12}$$

together with the Hölder inequality yield dominating functions which allow to interchange infinite series and integral.

Finally, we conclude for the norm:

$$\begin{aligned}\int_{\mathfrak{X}} f^{(u)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} &= \int_{\mathfrak{X}} \sum_{i,j} \alpha_i \alpha_j e_i^{(u)}(\mathbf{x}) e_j^{(u)}(\mathbf{x})\mathrm{d}\mathbf{x} \\ &\overset{(3)}{=} \sum_{i,j} \alpha_i \alpha_j \int_{\mathfrak{X}} e_i^{(u)}(\mathbf{x}) e_j^{(u)}(\mathbf{x}) d\mathbf{x} = \sum_{i=1}^{\infty} \alpha_i^2 = \|f\|_u.\end{aligned}$$

For (3) we use the inequality

$$|\sum_{i=1}^{n} \alpha_i \alpha_j e_i^{(u)}(\mathbf{x}) e_j^{(u)}(\mathbf{x})| \le \sum_{i=1}^{\infty} |\alpha_i| \cdot |\alpha_j| \cdot |e_i^{(u)}(\mathbf{x})| \cdot |e_j^{(u)}(\mathbf{x})|$$

and apply dominated convergence again. □

For $v \subseteq u$ and $f \in H_u$ the weak derivative $f^{(v)}$ necessarily exists, as well. Moreover, it is even square-integrable and, in particular, we have the following useful inequality:

Lemma 3.12. *For $f \in H_u$ and $v \subseteq u$ we have*

$$\int_{\mathfrak{X}} f^{(v)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} \leq \|f\|_u^2 \int_{\mathfrak{X}} k_{u\setminus v}(\mathbf{x},\mathbf{x}) \mathrm{d}\mathbf{x}.$$

Proof. Let $(e_i \otimes f_j)_{i,j\in\mathbb{N}}$ be an ONB of $H_u = H(k_v \otimes k_{u\setminus v})$ according to Lemma 2.47. Further let $(\alpha_{i,j})_{i,j\in\mathbb{N}}$ such that $f = \sum_{i,j\in\mathbb{N}} \alpha_{i,j} e_i \otimes f_i$. According to the proof of Lemma 2.47 we have $f = \sum_{i=1}^{\infty} e_i h_i$ with $h_i = \sum_{j=1}^{\infty} \alpha_{i,j} f_j \in H(k_{u\setminus v})$. Then

$$\begin{aligned}
\int_{\mathfrak{X}} f^{(v)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} &= \int_{\mathfrak{X}} \sum_{i,j} e_i^{(v)}(\mathbf{x}_v) e_j^{(v)}(\mathbf{x}_v) h_i(\mathbf{x}_{u\setminus v}) h_j(\mathbf{x}_{u\setminus v}) \mathrm{d}\mathbf{x} \\
&= \sum_{i,j} \underbrace{\int_{\mathfrak{X}} e_i^{(v)}(\mathbf{x}_v) e_j^{(v)}(\mathbf{x}_v) \mathrm{d}\mathbf{x}}_{=1 \text{ if i=j, 0 otherwise}} \int_{\mathfrak{X}} h_i(\mathbf{x}_{u\setminus v}) h_j(\mathbf{x}_{u\setminus v}) \mathrm{d}\mathbf{x} \\
&= \sum_i \int_{\mathfrak{X}} h_i(\mathbf{x}_{u\setminus v})^2 \mathrm{d}\mathbf{x} \leq \sum_i \|h_i\|_{k_{u\setminus v}}^2 \int_{\mathfrak{X}} k_{u\setminus v}(\mathbf{x},\mathbf{x}) \mathrm{d}\mathbf{x}.
\end{aligned}$$

With $\|h_i\|_{u\setminus v}^2 = \sum_{j=1}^{\infty} \alpha_{i,j}^2$ and $\|f\|_u^2 = \sum_{i,j} \alpha_{i,j}^2$ the assertion follows. Note that we used dominated convergence similarly as in (3.10), (3.11) and (3.12). □

Recall from Section 2.2 the operator $\Psi_{v,\mathbf{a}} : f \mapsto f(\cdot, \mathbf{a}_{-v})$ that we used to restrict the set of variables a function depends on to the set v. With Lemma 3.11(ii) we see that for the specific choice $\mathbf{a} = \mathbf{0} = (0, 0, \ldots) \in \mathfrak{X}$ it holds true that

$$\Psi_{v,\mathbf{0}}(f) = \sum_{u\subseteq 1:d} f_u(\cdot, \mathbf{0}_{-v}) = \sum_{u\subseteq v} f_u,$$

that is, $\Psi_{v,\mathbf{0}}$ acts as the orthogonal projection onto $H(K_v)$. This insight allows to derive a meaningful characterization of the norms of $H(K_{1:d})$ and $H(K)$ in the following and furthermore plays a significant role in the upcoming tractability analysis in Subsection 3.2.2.

Let us now characterize the space $H(K_{1:d})$.

Proposition 3.13. *For $d \in \mathbb{N}$ the space $H(K_{1:d})$ is the space of all continuous functions $f : \mathfrak{X} \to \mathbb{R}$ such that $f(\mathbf{x})$ depends only on $\mathbf{x}_{1:d}$ and the weak derivative $f^{(u)}$ exists and is square-integrable for each $u \subseteq 1:d$. On $H(K_{1:d})$ the norm is given by*

$$\|f\|_{K_{1:d}}^2 = \sum_{u\subseteq 1:d} \gamma_u^{-1} \int_{D^{|u|}} f^{(u)}(\mathbf{x}_u, \boldsymbol{a}_{-u}) \mathrm{d}\mathbf{x}_u.$$

Proof. To begin with, we prove the asserted for the norm. Let $f = \sum_{u\subseteq 1:d} f_u \in H(K_{1:d})$. Each f_u has a weak derivative, thus f has a weak derivative and $f^{(v)} = \sum_{u:v\subseteq u\subseteq 1:d} f_u^{(v)}$. Further, with $\Psi_{v,\mathbf{0}}$ being the orthogonal projection onto $H(K_v)$, we have $\Psi_{v,\mathbf{a}}(f^{(v)}) = f_v^{(v)}$ and

$$\|f\|_{K_{1:d}}^2 = \sum_{u\subseteq 1:d} \gamma_u^{-1} \|f_u\|_u^2 = \sum_{u\subseteq 1:d} \gamma_u^{-1} \int_{\mathfrak{X}} f_u^{(u)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} = \sum_{u\subseteq 1:d} \gamma_u^{-1} \int_{\mathfrak{X}} f^{(u)}(\mathbf{x}_u, \mathbf{a}_{-u})^2 \mathrm{d}\mathbf{x}_u.$$

Now denote by $G_{1:d}$ the space of continuous functions $f : \mathfrak{X} \to \mathbb{R}$ such that $f(\mathbf{x})$ depends only on $x_{1:d}$ and the weak derivative $f^{(u)}$ exists and is square-integrable for each $u \subseteq 1:d$. With Lemma 3.12 we get

$$\int_{\mathfrak{X}} f^{(v)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} \leq d \sum_{u:v\subseteq u\subseteq 1:d} \int_{\mathfrak{X}} f_u^{(v)}(\mathbf{x})^2 \mathrm{d}\mathbf{x} \leq d \sum_{u:v\subseteq u\subseteq 1:d} \gamma_u^{-1} \|f_u\|_u^2 \int_{\mathfrak{X}} k_{u\setminus v}(\mathbf{x},\mathbf{x}) \mathrm{d}\mathbf{x} < \infty \tag{3.13}$$

for $f \in H(K_{1:d})$, hence $H(K_{1:d}) \subseteq G_{1:d}$. It remains to establish the opposite set inclusion.

Therefor, let $f \in G_{1:d}$ and consider the functions f_v defined recursively for $v \subseteq 1:d$ by

$$\begin{aligned} f_\emptyset &:= \Psi_{\emptyset,\mathbf{a}}(f) \\ f_v &:= \Psi_{v,\mathbf{a}}(f) - \sum_{u\subsetneq v} f_u. \end{aligned}$$

Then $f = \sum_{v\subseteq 1:d} f_v$ and for $v \subseteq 1:d$ we clearly have that f_v depends only on x_v and that $f_v^{(v)}$ is square-integrable. Consequently, if we verified $f_v(\mathbf{x}) = 0$ in case that $x_j = 0$ for some $j \in v$ then we would have $f_v \in H_v$ and consequently $f \in H(K_{1:d})$.

So let $x \in \mathfrak{X}$ such that $x_j = 0$ for some $j \in v$. Furthermore put $w = v \setminus \{j\}$. Then we have

$$\begin{aligned} f_v(\mathbf{x}) \overset{\text{def.}}{=} f(\mathbf{x}_v, \mathbf{a}_{-v}) - \sum_{u\subsetneq v} f_u(\mathbf{x}) &= f(\mathbf{x}_w, \mathbf{a}_{-w}) - \sum_{u\subseteq w} f_u(\mathbf{x}) \\ &\overset{\text{induction}}{=} \sum_{u\subseteq w} f_u(\mathbf{x}) - \sum_{u\subseteq w} f_u(\mathbf{x}) = 0. \end{aligned}$$

□

In the specific situation of k being the minimum kernel, the space $H(K_{1:d})$ is well known in the literature as *anchored weighted Sobolev space* with anchor $\mathbf{a} = (0,0,\ldots)$ and product weights $(\gamma_j)_{j\in 1:d}$. To stress the importance we give a definition:

Definition 3.14. The space $H(K_{1:d})$ is called the *anchored weighted Sobolev space* with anchor $\mathbf{a} = (0,0,\ldots)$ and product weights $(\gamma_j)_{j\in 1:d}$.

Remark 3.15. Anchored weighted Sobolev spaces are also studied for general weights $(\gamma_u)_{u\subseteq 1:d}$ and anchors $\mathbf{a}$ with $a_j \in [0,1]$. Then the kernels $k_{\{j\}}$ are not always the minimum kernel, but

$$k_{\{j\}}(\mathbf{x},\mathbf{y}) = \begin{cases} \min(|x_j - a_j|, |y_j - a_j|) & \text{if } (x_j - a_j)(y_j - a_j) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

See e.g. [NW10] for further details.

The next proposition establishes a link between anchored weighted Sobolev spaces and the weighted Korobov spaces we encountered in Section 2.4. This link is a crucial foundation for the tractability results we present in the consecutive subsection.

Proposition 3.16. *For $j \in \mathbb{N}$ let $\beta_j = 1 + \frac{\gamma_j}{3}$. Further, let $\tilde{\boldsymbol{\gamma}} = (\tilde{\gamma}_{d,u})_{d\in\mathbb{N},u\subseteq 1:d}$ with*

$$\tilde{\gamma}_{d,u} = 2^{|u|}\gamma_u \prod_{j\in 1:d\setminus u} \beta_j.$$

Then the Korobov kernel $K_{d,1,\tilde{\gamma}}$ given by (2.26) *with the choice $\alpha = 1$ and weights $\tilde{\boldsymbol{\gamma}}$ is the associated shift-invariant kernel of $K_{1:d}$.*

Furthermore, for the particular choice of α and the weights $\tilde{\boldsymbol{\gamma}}$ the Korobov kernel $K_{d,1,\tilde{\gamma}}$ has the form

$$K_{d,1,\tilde{\gamma}}(\mathbf{x},\mathbf{y}) = \prod_{j\in 1:d} (\beta_j + \gamma_j B_2(\{x_j - y_j\}))$$

where B_2 is given by (2.27).

Proof. See [NW10, Lemma 16.20] for the relation between Sobolev and Korobov space. Then note that we already have

$$K_{d,1,\gamma}(\mathbf{x},\mathbf{y}) = \sum_{u\subseteq 1:d} \frac{\tilde{\gamma_{d,u}}}{2^{|u|}} \prod_{j\in u} B_2(\{x_j - y_j\}).$$

Hence the latter assertion follows by the following calculation:

$$\sum_{u\subseteq 1:d} \frac{\tilde{\gamma_{d,u}}}{2^{|u|}} \prod_{j\in u} B_2(\{x_j - y_j\}) = \sum_{u\subseteq 1:d} \gamma_u \left(\prod_{j\in 1:d\setminus u} \beta_j\right) \prod_{j\in u} B_2(\{x_j - y_j\})$$
$$= \prod_{j\in 1:d} \beta_j \sum_{u\subseteq 1:d} \prod_{j\in u} \frac{\gamma_j}{\beta_j} B_2(\{x_j - y_j\}) = \prod_{j\in 1:d} \beta_j \prod_{j\in 1:d} (1 + \frac{\gamma_j}{\beta_j} B_2(\{x_j - y_j\})).$$

□

As stated by Hickernell et al.[HMNR10] the characterization we verified for $H(K_{1:d})$ is also true for the space $H(K)$. Hence we obtain

Proposition 3.17. *The space $H(K)$ is the space of all continuous functions $f : \mathfrak{X} \to \mathbb{R}$ having square-integrable weak derivatives $f^{(u)}$ for each $u \in P_0(\mathbb{N})$. The norm is given by*

$$\|f\|_K^2 = \sum_{u\in P_0(\mathbb{N})} \gamma_u^{-1} \int_{D^{|u|}} f^{(u)}(\mathbf{x}_u, \boldsymbol{a}_{-u}) \mathrm{d}\mathbf{x}_u.$$

3.2.2 Complexity Results

Recall from Section 2.2 the deterministic N-th minimal error

$$e^{\mathrm{det}}_{N,\Delta}(F) = \inf\{e(Q,F) : Q \in \mathcal{Q}^{\mathrm{det}}(F) \wedge \mathrm{cost}_\Delta(Q,F) \leq N\}$$

with $\Delta \in$ {fix,var} and in the given situation $F = B(K)$. We present the upper and lower bounds on the N-th minimal error the authors Niu et al. derive in [NHMR10], particularly

showing the superiority of the multilevel approach in the given setting. Thereby we also summarize the reasoning leading to the results.

Let us at first have a brief glance at the form of the worst-case error for a quadrature rule $Q(\cdot; P_n, \mathbf{w})$ using points $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathfrak{X}_{1:d,\mathbf{0}}$. Due to integration and point evaluation being continuous linear functionals on $H(K)$ we have representatives h and $g = \sum_{i=1}^{n} w_i K(\cdot, \mathbf{x}_i)$ for integration and $Q(\cdot; P_n, \mathbf{w})$, respectively. Thereby g is an element of $H(K_{1:d})$.

We mentioned the truncation error in Section 2.2 which expresses the error arising from approximating $I(\Psi_{1:d,\mathbf{a}} f)$ instead of $I(f)$. We fix the notation

$$b_{1:d}(F) = \sup_{f \in F} |I(f) - I(\Psi_{1:d,\mathbf{a}} f)|.$$

Now, using that $\Psi_{1:d,\mathbf{0}}$ is the orthogonal projection onto $H(K_{1:d})$, we obtain

$$\|h - g\|_K^2 = \|h - \Psi_{1:d,\mathbf{0}} h\|_K^2 + \|\Psi_{1:d,\mathbf{0}} h - g\|_K^2.$$

Consequently, the squared worst-case error $e^2(Q(\cdot; P_n), B(K))$ splits exactly into the truncation error $b_{1:d}^2(B(K))$ and the worst-case error of Q w.r.t. $B(K_{1:d})$:

$$e^2(Q(\cdot; P_n), B(K)) = b_{1:d}^2(B(K)) + e^2(Q(\cdot; P_n), B(K_{1:d})).$$

Considering particularly the multilevel algorithm $Q_{\mathbf{n},\mathbf{d}}^{ml}$ given by Definition 2.19, we have the following: Let $\mathbf{n} = (n_1, ..., n_L)$, $\mathbf{d} = (d_1, ..., d_L)$ and further let $Q_1, ..., Q_L$ denote the quadrature rules used by the multilevel algorithm as building blocks. Then the worst-case error fulfills

$$\begin{aligned} e^2(Q_{\mathbf{n},\mathbf{d}}^{ml}, B(K)) = b_{1:d_L}^2(B(K)) &+ e^2(Q_1, B(K_{1:d_1})) \\ &+ \sum_{l=2}^{L} e^2(Q_l, B(K_{1:d_l})) - e^2(Q_l, B(K_{1:d_{l-1}})), \end{aligned} \tag{3.14}$$

see [NHMR10, Theorem 1].

In the following the sequence of weights $(\gamma_j)_{j \in \mathbb{N}}$ is assumed to satisfy

$$\gamma_j \asymp j^{-1-2q}$$

for some $q > 0$.

Upper Bounds

In order to yield upper bounds the multilevel algorithm $Q_{\mathbf{n},\mathbf{d}}^{ml}$ with rank-1 lattice rules as building blocks is used. As it is difficult to work directly with (3.14) Niu et al. introduce auxiliary weights $(\gamma'_j)_{j \in \mathbb{N}}$ with corresponding kernels $K'_{1:d}$ such that $\gamma'_j = j^{2(q-q')}\gamma_j$ with $q \geq q' > 0$. This allows to modify (3.14) in the way that

$$e^2(Q_{\mathbf{n},\mathbf{d}}^{ml}, B(K)) \leq b_{1:d_L}^2(B(K)) + \sum_{l=1}^{L} \kappa_l e^2(Q_l, B(K'_{1:d_l})) \tag{3.15}$$

where $\kappa_l := \max_{d_{l-1} \le l \le d_l} \frac{\gamma_j}{\gamma'_j}$. For the truncation error it can be shown that

$$b_{1:d}^2(B(K)) \asymp \sum_{j=d+1}^{\infty} \gamma_j \preceq d^{-2q}.$$

Further, following an analysis in [HSW04] it can be assumed

$$\sup_{d\in\mathbb{N}} \inf\{e(Q(\cdot;P_n), B(K'_{1:d})) : P_n \text{ is shifted rank-1 lattice }\} \preceq n^{-p'} \tag{3.16}$$

with $0 < p' < \min(1, q'+1/2)$. Basically, the argumentation here is that given the Sobolev kernel $K'_{1:d}$ the weights of the associated Korobov kernel (cf. Proposition 3.16) are such that we find good rank-1 lattice rules whose worst-case error declines with order at most p' in n independent of the dimension d (cf. Proposition 2.58). Then there are shifts such that (3.16) holds true (cf. Proposition 2.64).

Given this Niu et al. conclude by means of the multilevel algorithm that

$$e^{\det}_{N,\mathrm{var}}(B(K)) \preceq N^{-p' \min(1, q/p'+q')}, \tag{3.17}$$

see [NHMR10, Theorem 3]. The crucial insight in terms of to the multilevel algorithm is that the upper bound is reached if the components of $\mathbf{n}$ and $\mathbf{d}$ are chosen to decline and increase geometrically in the level l, respectively.

Lower Bounds

We saw that for each quadrature rule $Q \in \mathcal{Q}^{\det}(B(K))$ the worst-case error splits into the truncation error $b_{1:d}(B(K))$ and the worst-case error $e(Q, B(K_{1:d}))$ of the corresponding d-dimensional subproblem.

The worst-case error of the d-dimensional subproblem can be shown to be bounded from below by $e(Q, B(k_{\{1\}}))$, that is the error of the the extremal case that we considered the one-dimensional subproblem. It is well-known that $e^{\det}_{N,\Delta}(B(k_{\{1\}})) \asymp N^{-1}$.

This leads to the conclusion that

$$e^{\det}_{N,\mathrm{fix}}(B(K)) \succeq \inf_{nd\le N} \left\{ (\sum_{j=d+1}^{\infty} \gamma_j)^{1/2} + e^{\det}_{N,\mathrm{fix}}(B(k)) \right\} \asymp \inf_{nd\le N} \left\{ (\sum_{j=d+1}^{\infty} j^{-1-2q})^{1/2} + N^{-1} \right\} \tag{3.18}$$

$$e^{\det}_{N,\mathrm{var}}(B(K)) \succeq (\sum_{j=N+1}^{\infty} \gamma_j)^{1/2} + e^{\det}_{N,\mathrm{var}}(B(k)) \asymp (\sum_{j=N+1}^{\infty} j^{-1-2q})^{1/2} + N^{-1}, \tag{3.19}$$

see [NHMR10, Theorem 4].

Conclusion

In order to simplify the presentation we introduce for $\Delta \in \{\mathrm{fix}, \mathrm{var}\}$ the exponent

$$\lambda^{\Delta}(B(K)) := \sup\{\chi > 0 : \sup_{N\in\mathbb{N}} e^{\det}_{N,\Delta}(B(K)) \cdot N^{\chi} < \infty\}.$$

Niu et al. derive now from (3.17), (3.18) and (3.19) the following, see [NHMR10, Corollaries 1,2]:

$$\begin{aligned} \min\left(\frac{q}{q+1}, \frac{q}{2q/(2q+1)+1}\right) &\leq \lambda^{\mathrm{fix}}(B(K)) && \leq \frac{q}{q+1} \\ \min\left(q, \frac{q+1/2}{2}, 1\right) &\leq \lambda^{\mathrm{var}}(B(K)) && \leq \min(q,1). \end{aligned}$$

Hence, under the fixed subspace sampling regime, rank-1 lattice rules yield optimal algorithms for most values of q. But under the variable subspace sampling regime, multilevel rank-1 lattice rules always outperform standard rank-1 lattice rules and moreover, the multilevel variant always yields an optimal algorithm except for $1/2 \leq q \leq 3/2$.

4 C++ Library for Numerical Experiments

This chapter is devoted to describing our implementations we used to carry out the numerical experiments presented in Chapter 5. We tried to design the implementations in a way which is easy extensible for further experiments, integrands and approaches to generate integration points. Though the focus is currently on numerical experiments we aimed at fast and efficient implementations which might be the basis for real applications.

Except for Section 4.1, where we describe some design elements, we keep our explanations in a mathematical style. Technical aspects and details concerning programming techniques are avoided as far as possible.

Remark 4.1. When we analyze computational cost in this chapter we refer to the *real number model* which comprises arithmetical and access operations as opposed to the coarser cost model introduced in Subsection 2.2.2.

4.1 General Features and Core Components

We recall Definition 2.19 which introduced the multilevel algorithm in the form

$$Q^{ml}_{\mathbf{n},\mathbf{d}}(f) := Q(\Psi_{1:d_1,a}f; \mathscr{X}_{n_1,d_1}, \mathbf{w}^{(1)}) + \sum_{l=2}^{L} Q(\Psi_{1:d_l,a}f - \Psi_{1:d_{l-1},a}f; \mathscr{X}_{n_l,d_l}, \mathbf{w}^{(l)}).$$

Currently our implementation handles only the case of equal weighted quadrature rules as building blocks, that is $w_i^{(l)} = 1/n_l$.

4.1.1 Design of the Library

For a flexible implementation it is useful to divide the multilevel algorithm into components which can be implemented independently of each other or vary with each multilevel algorithm run. For this purpose let us change the signature of the algorithm as follows:

$$Q^{ml}_{\mathbf{n},\mathbf{d}}(f) = Q^{ml}(f, P, \vartheta, \varepsilon).$$

The arguments now refer to

(1) f: a subroutine representing the integrand.

(2) P: a component representing the method used to generate integration points. Note that this method can be both deterministic and randomized.

(3) ϑ: a component representing the strategy which determines the choice of the multilevel parameters $\mathbf{n}$ and $\mathbf{d}$,

(4) ε: a user-specified bound on the integration error.

Since the choice of the multilevel parameters is assigned to ϑ it is the component which determines whether an algorithm run is to be finished. Consequently, it is reasonable to put the handling of the error bound ε into its responsibility.

It depends on P and ϑ whether ε actually acts as an error bound. For instance, if the integration points are generated randomly, we might implement a strategy based on variance and bias estimates which turns ε into error bound the algorithm hits on average. On the other hand it is likewise possible that ε is in fact just a heuristic with no guaranteed quality.

Monitoring Mechanism

One focus of the library is to provide tools for numerical experiments. Such experiments require a mechanism to monitor the multilevel algorithm's activities. Since we do not know the design of all experiments one might carry out with the multilevel algorithm, it is a bad idea to code the specific monitoring directly into the implementation of Q^{ml}.

Instead we need a flexible mechanism which prescribes what information has to be provided for monitoring but makes no assumptions on the usage of the monitored data.

We address these requirements by designing our monitoring mechanism according to the *observer-observable pattern*. The core implementation of Q^{ml} is an observable which has a list of observers, that is a list of components which want to be informed whenever some change happened. In view of our multilevel computation this means that each of the listed observers gets notified by Q^{ml} as soon as a crucial computational step has been carried out. The smallest crucial computational step is the evaluation of f at some integration point.

See the next subsection to have a more detailed illustration when notifications take place.

4.1.2 Basic Variant of the Multilevel Algorithm

Let f, P and ϑ be components as defined above. Moreover, let $\mathbf{d}^{\vartheta}$, $\mathbf{n}^{\vartheta}$ denote the dimensions and numbers of integration points the strategy ϑ generates. To follow previous notations we use $P_{n,d}$ to indicate that the component P is configured such that it represents a point set of n different points in dimension d.

We implement the building blocks of the multilevel algorithm (2.19) as a subroutine

$$Q(l, f, P, \vartheta).$$

Here l denotes the level which we currently wish to compute. If we identify f and the function represented by it, the subroutine's outcome is such that

$$\begin{aligned} Q(l, f, P, \vartheta) &= Q(\Psi_{1:d_1^{\vartheta},\mathbf{a}} f\,; P_{n_1^{\vartheta},d_1^{\vartheta}}) && \text{for } l = 1 \\ Q(l, f, P, \vartheta) &= Q(\Psi_{1:d_l^{\vartheta},\mathbf{a}} f - \Psi_{1:d_{l-1}^{\vartheta},\mathbf{a}} f\,; P_{n_l^{\vartheta},d_l^{\vartheta}}) && \text{for } l > 1. \end{aligned}$$

Note that the vector $\mathbf{a}$ is implicitly given by the subroutine f.

To allow for adaptive strategies we manage the levels in a queue. A queue is a first-in first-out (FIFO) data structure, that is new elements can only be added to its end and the next element to read comes always from its front. We refer to this queue in the following as *level queue*.

We desire the core implementation of the multilevel algorithm to be compatible with both deterministic and randomized methods of integration point generation. In the randomized case it might be of interest to compute the same level several times in order to reach more stable outcomes by averaging. The strategy given in paragraph S3 of Section 4.4 uses multiple level computation, as well. However, from the core implementation's point of view it actually does not matter how we generate the integration points, it only must know how often it has to repeat the computation of the current level.

Because the strategy ϑ determines the number of levels and on each level the evaluation dimension and the number of integration points, it has to fill the level queue. Consequently, it already acts pretty much like a controller and we choose the strategy to determine the number of *level repetitions*, as well. If it sets the number of level repetitions to be M we denote the different outcomes by $Q_l^{(i)}$ for $i = 1, ..., M$.

As the core implementation does not know the meaning of the results $Q_l^{(i)}$, it can not decide how to map the outcomes to the actual level outcome Q_l^{res} being incorporated in the total multilevel algorithm return value. In accordance with the previous approach we let ϑ decide how to map the repeated outcomes to the level outcome.

The decisions made so far lead to Algorithm 4.1.

Algorithm 4.1 Basic Multilevel Algorithm Implementation

Input: function representer f, point generator P, strategy ϑ, error bound ε
Output: an approximation of $I(f)$

1: ϑ initializes level queue;
2: **while** level queue not empty **do**
3: $l \leftarrow$ first element in level queue; remove first element from level queue
4: ϑ specifies number of level repetitions M;
5: **for** $i = 1, ..., M$ **do**
6: compute $Q_l^{(i)} \leftarrow Q(l, f, P, \vartheta)$;
7: **end for**
8: ϑ maps $(Q_l^{(i)})_{i=1,...,M}$ to level result Q_l^{res};
9: ϑ updates level queue;
10: **end while**
11: **return** $\sum_{l=1}^{L} Q_l^{res}$;

We omitted in Algorithm 4.1 those lines implementing the monitoring mechanism. These involve the following subroutines:

- The subroutine notifyStart(f,P,ϑ,ε) is invoked before the multilevel algorithm actually has started. It tells the listeners which integrand, point generation, parameter strategy and user-specific error bound the algorithm is currently running with.

- The subroutine notifyBeforeLevel(l,i,d_l,n_l) is invoked within $Q(l, f, P, \vartheta)$ right before the current level has been computed. It tells the listeners the current level l, the level

repetition i, the current dimension d_l and the current number of integration points n_l.

- The subroutine notify(f_l,f_{l-1}) is called within $Q(l, f, P, \vartheta)$ every time the algorithm has evaluated the integrands in the currently considered dimensions d_l and d_{l-1}. The listeners are told the evaluation outcomes f_l and f_{l-1} for dimension d_l and d_{l-1}, respectively.
- The subroutine notifyAfterLevel(l,i,Q_l^{res}) is invoked within $Q(l, f, P, \vartheta)$ right after a level computation has finished. It tells the listeners the current level l, the level repetition i and the level outcome Q_l^{res}.
- The subroutine notifyStop(R) is called just after the multilevel algorithm has finished and tells the listeners the output $R = \sum_{l=1}^{L} Q_l^{res}$.

4.1.3 Multithreading

Multi-core processors are nowadays standard components of personal computers. Even portable notebooks often come along with two cores. These multi-core architectures offer the opportunity to save real computation time. Namely by explicitly delegating portions of work involved with a multilevel algorithm run or a numerical experiment to different programme threads that are executed in parallel.

Of course, we have to analyze carefully the dependencies between different computational steps, identifying those parts which can be computed independently of each other. We briefly discuss two different approaches how to introduce multiple threads to our code.

The simplest idea is to stick to the basic core implementation given by Algorithm 4.1 and put only different calls to this routine into parallel programme threads. This approach seems suitable due to our focus being currently on numerical experiments where we have to repeat the same computation many times. However, we meet a problem implementing this straightforward. Our numerical experiments involve pseudorandom number generation. Thereby we are not offhand guaranteed that the pseudorandom numbers generated for the different threads do not show any form of unintended correlation.

Having practical applications in mind another approach is introducing multi-threading to Algorithm 4.1 itself. The first idea is, of course, to parallelize the computation of the different levels. However, this might be tricky or impossible if we use adaptive strategies for the choice of the multilevel parameters. A simple and possible way is to split each level computation up into an number N of parts that corresponds to the number of available CPU kernels. More precisely, if we are given n_l integration points in level l we run N threads each computing $\lfloor \frac{n_l}{N} \rfloor$ integrand evaluations (if n_l is not a multiple of N we divide the remaining integrand evaluations to the threads as equally as possible). If $n < N$ we run only one single thread. As the integration points are already generated when we split the computation we avoid the problem with pseudorandom numbers as sketched above.

Currently we implemented the latter approach, reaching time savings of a factor up to 1.6 on a machine with two Intel P8700 2.5 GHz CPU kernels. But we consider this rather as a workaround. It works badly if the computational effort for integrand evaluation is not significantly higher than for pseudorandom number generation. Then not including

the pseudorandom number generation with our multithreading approach may eliminate a significant amount if not most of the time savings we intended to realize through multithreading.

4.2 Pathwise Computation of Asian Call Payoff

We recall from Section 2.1 that the prize of the Asian Call Option in the Black-Scholes model on the time interval $[0,T]$ is given by $E[\varphi_{\mathrm{AC}}(S)]$ where

$$S_t = \Gamma(W)_t = \exp((r-\sigma^2/2)\cdot t + \sigma W_t)$$

is a geometric Brownian motion and the payoff φ_{AC} is given by

$$\varphi_{\mathrm{AC}}(f) = e^{-rT}\max\left(\frac{1}{T}\int_0^T f(t)\mathrm{d}t - K, 0\right).$$

In Section 2.1 we also learned a number of approaches to compute the option prize approximately. This involved distinct possibilities to approximate $\varphi_{\mathrm{AC}}(S)$ pathwise. In this section we now discuss these from an implementational point of view and compare their computational costs.

With regard to our multilevel algorithm implementation introduced in the previous section we have to implement the pathwise approximations as subroutines f which behave as follows: Given a vector $\mathbf{x}$ (integration point) and an integer d (dimension) the component f uses the first d components of $\mathbf{x}$ to compute a pathwise approximation of $\varphi_{\mathrm{AC}}(S)$. We denote the result by $f(\mathbf{x}_{1:d})$.

It is important to note here that we assume the component not to care how $\mathbf{x}$ is generated but to treat it always as a vector of numbers each of which is sampled independently according to the standard normal distribution.

4.2.1 Comparison of Distinct Approaches

Put

$$i(m,j) = \begin{cases} 0 & \text{if } m = j = 0, \\ 2^{m-1} + j & \text{otherwise.} \end{cases}$$

and $d_l = 2^l$.

Then for the Lévy-Ciesielski expansion let

$$L^{(l)}(\mathbf{x}_{1:d_l}) = \sum_{m=0}^{l}\sum_{j=0}^{J(m)} x_{i(m,j)} e_{m,j}$$

denote the l-th partial sum of the Lévy-Ciesielski expansion where the realization of $X_{m,j}$ took the value $x_{i(m,j)}$. Analogously, for the Karhunen-Loève expansion let

$$K^{(l)}(\mathbf{x}_{1:d_l}) = \sum_{j=1}^{d_l} x_j e_j.$$

Furthermore, denote by $\hat{W}^{(l)}(\mathbf{x}_{1:d_l})$ the path of the Euler approximation in setting R1 which we obtain if the Brownian increments took the values $2^{-l/2}x_j$ for $j = 1, ..., d_l$.

By $\zeta_{R2}^{(l)}$ we denote the map given by the Euler-Maruyama on the path space in case of setting R2, see Remark 2.3.

Now, according to Section 2.1, we may implement components $f_1, ..., f_5$ which compute pathwise approximations of $\varphi_{\mathrm{AC}}(S)$ in the following five different ways. In relation to setting R1 we have:

F1: A subroutine computing an approximative path of Brownian motion by means of the Lévy-Ciesielski partial sum:

$$f_1(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(L^{(l)}(\mathbf{x}_{1:d_l})). \tag{4.1}$$

F2: A subroutine computing an approximative path of Brownian motion by adding up Brownian increments:

$$f_2(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(\hat{W}^{(l)}(\mathbf{x}_{1:d_l})). \tag{4.2}$$

F3: A subroutine computing an approximative path of Brownian motion by means of the Karhunen-Loève partial sum:

$$f_3(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(K^{(l)}(\mathbf{x}_{1:d_l})). \tag{4.3}$$

In relation to setting R2 we have:

F4: A subroutine computing an approximative path of geometric Brownian motion using the Euler-Maruyama scheme where Brownian increments are determined from the Lévy-Ciesielski partial sum:

$$f_4(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \zeta_{R2}^{(l)}(L^{(l)}(\mathbf{x}_{1:d_l})). \tag{4.4}$$

F5: A subroutine computing an approximative path of geometric Brownian motion using the Euler-Maruyama scheme where Brownian increments are determined directly from $\mathbf{x}_{1:d_l}$:

$$f_5(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \zeta_{R2}^{(l)}(\hat{W}^{(l)}(\mathbf{x}_{1:d_l})). \tag{4.5}$$

We analyze the cost to compute $f_k(\mathbf{x}_{1:d_l})$ for $k = 1, ..., 5$. Thereby we focus on clarifying how the cost depend on d_l. In the following denote in by w in either case the path we obtain according to the given approach.

Cost analysis for F4 and F5

The cost to evaluate $\varphi_{\mathrm{AC}}(w)$ basically result from calculating the one-dimensional integral

$$\int_0^T w(t)\mathrm{d}t. \tag{4.6}$$

The path w is piecewise linear between the points $w(t_i)$, $t_i = i2^{-l}$ for $i = 0,...,2^{-l}$. Hence we have

$$\int_0^T w(t)\mathrm{d}t = \sum_{i=0}^{2^l-1} \int_{t_i}^{t_{i+1}} w(t)\mathrm{d}t = \sum_{i=0}^{2^l-1} \frac{1}{2}(w(t_i) + w(t_{i+1})) \cdot (t_{i+1} - t_i).$$

As we use the Euler-Maruyama scheme the values $w(t_i)$, $i = 0,...,2^l$ are successively determined by

$$w(t_{i+1}) = (1 + r2^{-l} + \sigma 2^{-l/2} x_i) \cdot w(t_i)$$

with $w(0) = s_0$ in case of the fifth approach. Hence we basically iterate once over the elements of $\mathbf{x}_{1:d_l}$.

For the fourth approach we proceed analogously but have to determine the Brownian increments from the Lévy-Ciesielski partial sum. This requires to access the components of $\mathbf{x}_{1:d_l}$ in a more complicated order but still can be done in a number of operations which increases linear in d_l, see Section 4.2.2 below.

We conclude that for $k = 4,5$ we can compute $f_k(\mathbf{x}_{1:d_l})$ in $O(d_l)$ using only simple arithmetic and access operations. For $k = 4$ the number of required operations is approximately 5 times higher than for $k = 5$ due to the more complicated usage of $\mathbf{x}_{1:d_l}$ with the Lévy-Ciesielski partial sum.

Cost analysis for F1 and F2

The cost analysis here is basically analogue as in case $k = 4,5$ except that now we have additional calls to the exponential function due to the integral to be computed being now

$$\int_0^T \Gamma(w)(t)\mathrm{d}t. \tag{4.7}$$

The path w is again piecewise linear. Having

$$w(t) = w(t_i) + \frac{t - t_i}{t_{i+1} - t_i}(w(t_{i+1}) - w(t_i))$$

for $t \in [t_i, t_{i+1})$ we rewrite it as $w(t) = c_1(i) \cdot t + c_2(i)$ with

$$c_1(i) = (t_{i+1} - t_i)^{-1}(w(t_{i+1}) - w(t_i))$$

and $c_2(i) = w(t_i) - c_1(i)t_i$. Put $\tilde{r} := r - \sigma^2/2$. Then

$$\begin{aligned}\int_0^T \Gamma(w)(t)\mathrm{d}t &= \sum_{i=0}^{2^l-1} (\tilde{r} + \sigma c_1(i))^{-1} \left[\exp\left((\tilde{r} + \sigma c_1(i))t + \sigma c_2(i)\right)\right]_{t_i}^{t_{i+1}} \\ &= \sum_{i=0}^{2^l-1} (\tilde{r} + \sigma c_1(i))^{-1} \left[\exp(\tilde{r}t + \sigma w(t))\right]_{t_i}^{t_{i+1}}.\end{aligned}$$

Consequently, for $k = 1, 2$ computing $f_k(\mathbf{x}_{1:d_l})$ requires a number of operations which increases linear in d_l. Compared to the case $k = 4, 5$ we now have additional $2d_l$ calls to the exponential function.

Cost analysis for F3

Approach 3 uses the Karhunen-Loève expansion. Since

$$e_j(t) = \sqrt{2}\frac{sin(\pi(j - 1/2)t)}{\pi(j - 1/2)}$$

the path w is now not piecewise linear as it has been the case before. Hence we can not determine

$$\int_0^T w(t)\mathrm{d}t$$

exactly anymore. In addition, approximating the integral requires to use some quadrature rule U, e.g. the trapezoidal rule. The computational cost for this quadrature rule rises proportionally to the number N_U of interpolation points it uses.

It is natural here to expect the output $f_3(\mathbf{x}_{1:d_l})$ to have a higher precision the greater d_l gets. If we use a fixed N_U it is unlikely to reach this demand for arbitrary large dimensions d_l. So, taking the demand for increasing precision fully into account would require to raise N_U in some way if d_l is increased. However, from a practical point of view, when the range of used dimensions d_l is bounded, it might be reasonable to fix some N_U or to limit N_U to some maximal value. In our current implementation N_U grows quadratically in d_l.

Besides the fact that the form of e_j makes the integral calculation more complicated, it also has the consequence that we have to perform $N_U \cdot d_l$ expensive sine computations.

We conclude that the third approach is quite expensive. If we incorporate the demand for precision increasing with the dimension, then the cost do not grow linear in d_l. But it might be legitimate to assume N_u fixed or bounded by some maximal value. Moreover, practical experience shows that we struggle to increase the precision of the output $f_3(\mathbf{x}_{1:d_l})$ reliably. We observed the problem for $d_l > 1000$. Then for some values of $\mathbf{x}_{1:d_l}$ summing up a large number of expression of the form

$$s_0 \exp((r - \sigma^2/2)t + \sigma \sum_{j=1}^{d_l} x_j e_j(t))$$

within the quadrature rule U temporarily requires more than double floating point precision.

Remark 4.2. The results on the cost of the approaches F1,F2,F4 and F5 justify the cost model in Subsection 2.2.2 as a reasonable abstraction for theoretical analysis of integration problems.

4.2.2 Implementation of the Lévy-Ciesielski Expansion

We recall from Subsection 2.1.4 that the Schauder functions $(e_{m,j})_{m\in\mathbb{N},j=0,\dots,J(m)}$ on the time interval $[0,1]$ are given by $e_{0,0}(t)=t$ for $m=j=0$ and for $m>0$ by

$$e_{m,j}(t) = 2^{\frac{(m-1)}{2}} \begin{cases} t-\frac{k-1}{2^m} & \text{if } t\in I(k-1,m) \\ \frac{k+1}{2^m}-t & \text{if } t\in I(k,m) \\ 0, & \text{else} \end{cases}$$

with $k=2j+1$ and $I(m,k)=[\frac{k}{2^m},\frac{k+1}{2^m}[$. To treat the case of an time interval $[0,T]$ with $T\neq 1$ we simply have to consider the transformed Schauder functions given by $\sqrt{T}e_{m,j}(t/T)$. Hence we may w.l.o.g. content ourself with assuming $T=1$.

For fixed $n\in\mathbb{N}$ the Lévy-Ciesielski partial sum

$$L^{(n)} = \sum_{m=0}^{n}\sum_{j=0}^{J(m)} X_{m,j}e_{m,j}$$

is affine linear on each interval $I(n,i)$, $i=0,\dots,2^n-1$, and hence

$$L_t^{(n)} = c_1(n,i)\cdot t + c_2(n,i)$$

for $t\in I(n,i)$ as well as coefficients $c_1(n,i)$ and $c_2(n,i)$ being linear combinations of the $X_{m,j}$ that depend on the given interval. We explicitly determine the linear combinations and derive an recursive scheme to calculate c_1 and c_2.

Recursive Scheme

In the following we denote $(e_{m,j})_{j=0,\dots,J(M)}$ as the *m-th level* of Schauder functions. Fix $i_n\in\{0,\dots,2^n-1\}$. Further let for $m<n$ the integer $i_m\in\{0,\dots,2^m-1\}$ be such that $I(n,i_n)\subseteq I(m,i_m)$. From the definition of the Schauder functions we see immediately that on each level $m=0,\dots,n$ there is precisely one Schauder function which takes values different from zero on $I(m,i_m)$. We denote by $j(m,i_m)$ the index of the corresponding Schauder function in the m-th level. Then on $I(n,i_n)$ we have for $s=1,2$ that

$$c_s(n,i_n) = \sum_{m=0}^{n}\lambda_s(m,i_m)X_{m,j(m,i_m)}$$

with $\lambda_s(m,i_m)\in\mathbb{R}$.

For $m=0$ we immediately see that $\lambda_1(0,0)=1$ and $\lambda_2(0,0)=0$. To determine the coefficients $\lambda_s(m,i_m)$ on level $m\in\{1,\dots,n\}$ we put $k(m,i_m):=2j(m,i_m)+1$ and observe

$$I(m,i_m) = \begin{cases} I(m,k(m,i_m)-1) & \text{if } i_m \text{ is even,} \\ I(m,k(m,i_m)) & \text{if } i_m \text{ is odd.} \end{cases} \tag{4.8}$$

We conclude for $t\in I(m,i_m)$ that

$$e_{m,j(m,i_m)}(t) = \left[(-1)^{i_m}\cdot 2^{\frac{m-1}{2}}\right]\cdot t + (-1)^{i_m+1}\cdot 2^{\frac{m-1}{2}}\cdot b(m,i_m) \tag{4.9}$$

with

$$b(m,i_m) = \begin{cases} \frac{k(m,i_m)-1}{2^m} & \text{if } i_m \text{ is even,} \\ \frac{k(m,i_m)+1}{2^m} & \text{if } i_m \text{ is odd.} \end{cases}$$

So we have

$$\begin{aligned} \lambda_1(m,i_m) &= (-1)^{i_m} \cdot 2^{\frac{m-1}{2}}, \\ \lambda_2(m,i_m) &= (-1)^{i_m+1} \cdot 2^{\frac{m-1}{2}} \cdot b(m,i_m). \end{aligned}$$

Though we have characterized $j(m,i_m)$ and $\lambda_s(m,i_m)$ we are yet lacking formulae which allow an efficient computation. Fortunately, we can obtain them easily. From (4.8) and $k(m,i_m) = 2j(m,i_m)+1$ we derive

$$\begin{aligned} j(m,i_m) &= \lfloor i_m/2 \rfloor, \\ b(m,i_m) &= \frac{2\lfloor (i_m+1)/2 \rfloor}{2^m}. \end{aligned} \tag{4.10}$$

Now that we are able to compute $j(m,i_m)$ and $\lambda_s(m,i_m)$ efficiently it remains to derive explicit formulae which describe the relation between the numbers $i_0,...,i_n$. The subset relation $I(m,i_m) \subseteq I(m-1,i_{m-1})$ implies

$$i_m = 2i_{m-1} \quad \vee \quad i_m = 2i_{m-1}+1.$$

So, in either case, if we are given i_m we have $i_{m-1} = \lfloor i_m/2 \rfloor$.

Combining the formulae derived so far gives rise to a recursive scheme to compute $c_s(m,i_m)$ for the interval $I(m,i_m)$: first we compute $\lambda_s(m,i_m)$, then we recursively determine $c_s(m-1,\lfloor i_m/2 \rfloor)$ for the interval $I(m-1,\lfloor i_m/2 \rfloor)$. Finally, we sum both parts up:

$$\begin{aligned} c_1(m,i_m) &= \lambda_1(m,i_m) \cdot X_{m,j(m,i_m)} + c_1(m-1,\lfloor i_m/2 \rfloor) \\ c_2(m,i_m) &= \lambda_2(m,i_m) \cdot X_{m,j(m,i_m)} + c_2(m-1,\lfloor i_m/2 \rfloor) \end{aligned} \tag{4.11}$$

Now if we are given a realizations of $X_{m,j}$ with $m = 0,...,n$ and $j = 0,...,J(m)$ in form of a vector $\mathbf{x}_{1:2^n}$ the recursive scheme involves n access operations to the vector. Each coefficient $\lambda_s(m,i_m)$ can be calculated in $O(1)$. Thereby note that powers of 2 can be computed in constant time using bit shifts. We conclude that the recursive scheme has computational cost in $O(n)$.

Traversing the Lévy-Ciesielski Binary Tree

Let $d_n = 2^n$. When using the Lévy-Ciesielski partial sum $L^{(n)}$ within $f_2(\mathbf{x}_{1:d_n})$ and $f_4(\mathbf{x}_{1:d_n})$ as given in the previous subsection we have to compute the coefficients $c_s(n,i_n)$, $s = 1,2$, successively for each interval $I(n,i_n)$, $i_n = 0,...,2^n-1$.

Using the recursive scheme given by (4.11) the cost to compute $c_s(n,i_n)$ is in $O(\log(d_n))$. So, overall we would have cost in $O(d_l \log(d_l))$ for computing all coefficients. However, the recursive scheme is not the optimal way for this purpose yet. When we are given

$$c_s(n,i_n) = \sum_{m=0}^{n} \lambda_s(m,i_m) X_{m,j(m,i_m)}$$

on the interval $I(n,i_n)$ and go from i_n to i_n+1 not all numbers i_m change for $m<n$. Consequently, it is not necessary to recompute every summand in the linear combination every time.

The previously mentioned relation

$$I(m,i_m)\subseteq I(m-1,i_{m-1}) \quad\Longrightarrow\quad i_m=2i_{m-1} \quad\vee\quad i_m=2i_{m-1}+1$$

suggests to imagine the intervals

$$I(m,i),\quad m=0,...,n,\ i=0,...,J(m+1)$$

forming a binary tree with root node $I(0,0)$. We refer to this tree representation as *Lévy-Ciesielski binary tree*.

We use the picture of the Lévy-Ciesielski binary tree to derive an algorithm which computes all the $c_s(n,i_n)$ together with cost in $O(d_n)$ using $O(\log d_n)$ extra memory. Therefor let us turn around the level order in the computation and start with computing $\lambda_s(0,0)X_{0,0}$. Furthermore introduce vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ of size $n+1$ which store for each level $m=0,...,n$ respectively the value of $c_1(m,i)$ and $c_2(m,i)$ we computed at last.

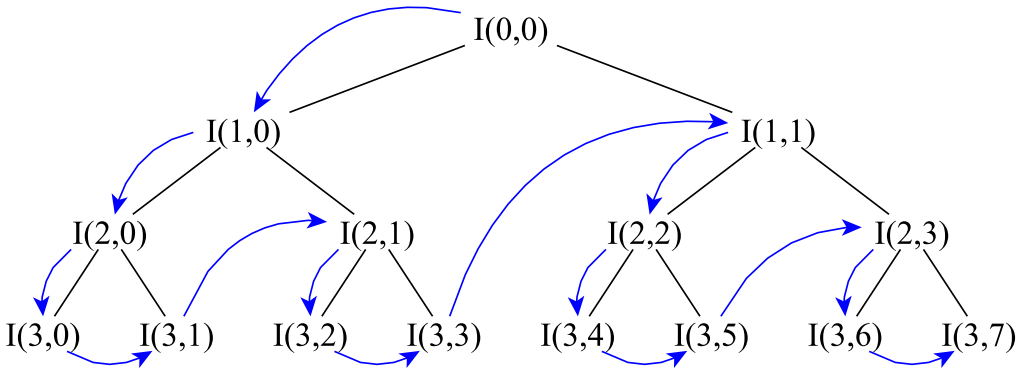

Figure 4.1: Traversing the Lévy-Ciesielski binary tree

Then, if we identify visiting the node $I(m,i)$ with computing $\lambda_s(m,i)X_{m,j(m,i)}$ and updating

$$\mathbf{c}_s[m]:=\mathbf{c}_s[m-1]+\lambda_s(m,i)X_{m,j(m,i)},$$

we obtain an algorithm which computes each $\lambda_s(m,i)X_{m,j(m,i)}$ exactly one time by walking through the Lévy-Ciesielski binary tree in a suitable order. For $n=3$ Figure 4.1 gives a graphical demonstration. The cost to traverse a tree is linear in the number of nodes. As the Lévy-Ciesielski binary tree has 2^{n+1} nodes we conclude that we can compute all the coefficients $c_s(n,i_n)$ together with cost in $O(d_n)$ using $O(\log d_n)$ extra memory.

We reach this by implementing the traversal with a stack based approach. A stack is a data container following the last-in-first-out principle. This means that elements are taken from the container in exact the opposite order in which they were stored to the container. Let $\mathbf{s}$ denote a stack of maximal size $2n+1$ and $\mathbf{i}$ a vector of size $n+1$. The stack $\mathbf{s}$ stores the levels m which have to be proceeded yet and the vector $\mathbf{i}$ for each level m the number i_m we currently consider. Then Algorithm 4.2 yields the desired traversal.

Algorithm 4.2 Computing Coefficients by Traversing the Lévy-Ciesielski Binary Tree

Input: realization $\mathbf{x} = \mathbf{x}_{1:2^n}$ of $(X_{m,j})_{m=0,\dots,n,j=0,\dots,J(m)}$.

```
c_1[0] ← x[0];
c_2[0] ← 0;
store m = 1 to s twice;
for m = 0, ..., n do
    i[m] ← 0;
end for
while s not empty do
    take m from s;
    compute: λ_s(m, i[m]);
    update: c_s[m] ← c_s[m − 1] + λ_s(m, i[m]) · x [2^{m−1} + ⌊i[m]/2⌋];
    if m ≠ n then
        store m + 1 to s twice;
    else
        use c_1[n], c_2[n] to compute respectively (4.6) and (4.7) on I(n, i[m]).
    end if
    i[m] ← i[m] + 1;
end while
```

Remark 4.3. Algorithm 4.2 can be slightly improved by exploiting the following. Since $L^{(n)}$ has continuous paths we have

$$c_1(n, i_n - 1) \cdot t_{i_n} + c_2(n, i_n - 1) = c_1(n, i_n) \cdot t_{i_n} + c_2(n, i_n).$$

Hence the coefficient $c_2(n, i_n)$ is uniquely determined by

$$c_2(n, i_n) = c_2(n, i_n - 1) + t_{i_n} \left(c_1(n, i_n - 1) - c_1(n, i_n) \right)$$

and the vector $\mathbf{c}_2$ can be replaced by a single variable which stores the value of c_2 on the previously considered interval.

Remark 4.4. With the multilevel algorithm we have to compute differences of the form $f(\mathbf{x}_{1:d_l}) - f(\mathbf{x}_{1:d_{l-1}})$ with $d_l = 2^k d_{l-1}$ for $k \in \mathbb{N}$. Therefor we do not have to execute the traversal of the Lévy-Ciesielski binary tree twice, but while doing the traversal for $n = d_l$ all values needed for $n = d_{l-1}$ are computed "on the way".

4.3 Methods of Integration Point Generation

Components for generation of integration points basically share the following interface in our library:

- A method prepare(l,d,n) which allows to tell the component that we expect it to compute a number of n integration points in dimension d on level l in the multilevel algorithm.
- A method getPoint(**p**, i) which saves the i-th integration point to the vector **p**.

Currently the library contains two such components. The first simply generates pseudo-random integration points. We refer to this component as P_{rand}. The second component implements (shifted) rank-1 lattices where the lattice generator is computed using the fast CBC algorithm of [Ny07]. We refer to this component as P_{lattice}.

4.3.1 Mersenne Twister

The component P_{rand} generates integration points $\mathbf{p}$ which are element-wise uniformly distributed on $[0,1]$. The pseudo random numbers are generated using a Mersenne Twister in the version MT19937 provided by the GNU Scientific Library [GSL11]. At the beginning we once initialize the Mersenne Twister with the default seed 4357.

The method prepare contains no instructions for this component. When invoking getPoint($\mathbf{p}$, i) the value i has no effect but for each component of $\mathbf{p}$ simply the Mersenne Twister is called.

We stress that the current implementation does not support multiple threads. If the component is used with multiple threads the access has to be synchronized between the threads. In particular, this means that all pseudo random numbers originate from one sequence.

4.3.2 (Shifted) Rank-1 Lattices

The component P_{lattice} generates integration points forming from a rank-1 lattice which can be either unshifted, with user-specified shift or randomly shifted. The core of this component is an implementation of the fast CBC algorithm which is used to compute the rank-1 lattice generator. Note that we currently implemented only the version of the fast CBC algorithm which accepts numbers of integration points that are prime, but there are also versions which accept more general numbers of integration points and hence compute generators for lattices other than rank-1 lattices, see [Ny07, Chapter 4].

We begin with discussing the usage of the P_{lattice}. On creation the component requires the following parameters to be specified:

- a function describing a shift-invariant kernel ω on $[0,1] \times [0,1]$. The kernel is expected to be given in terms of one variable, cf. Remark 2.61.
- functions describing sequences of weights $\boldsymbol{\beta} = (\beta_i)_{i\in\mathbb{N}}$ and $\boldsymbol{\gamma} = (\gamma_i)_{i\in\mathbb{N}}$, respectively.
- a flag indicating how random shifts are introduced. The component supports two distinct modes:
  1. No random shifts. Then the user may manually specify a shift.
  2. Independent random shifts. That is, for each level and each repetition the corresponding shift is sampled independently, the shift being in each component distributed according to the uniform distribution on $[0,1]$.

Given ω, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, invoking prepare(l,d,n) the generator $\mathbf{z} = (z_1, \ldots, z_d) \in \{1, \ldots, n-1\}^d$ of a rank-1 lattice $P_n(\mathbf{z})$ is computed by determing $\mathbf{z}$ component by component. Thereby each component z_s is chosen such that

$$e^2(Q(\cdot, P_n(\mathbf{z})), B(K_{d,\boldsymbol{\beta},\gamma}))$$

is minimized while keeping the previously determined components $z_1, ..., z_{s-1}$ fixed. Here $Q(\cdot, P_n(\mathbf{z}))$ denotes the rank-1 lattice rule using the lattice generated by $\mathbf{z}$. The class of integrands is the unit ball $B(K_{d,\boldsymbol{\beta},\boldsymbol{\gamma}})$ of the RKHS which has the shift-invariant reproducing kernel

$$K_{d,\boldsymbol{\beta},\boldsymbol{\gamma}}(\mathbf{x},\mathbf{y}) = \prod_{j=1}^{d}(\beta_j + \gamma_j \omega(\{x_j - y_j\}).$$

Example 4.5. Let $\boldsymbol{\gamma} = (\gamma_j)_{j\in\mathbb{N}}$ be a sequence of weights and let $K_{1:d}$ denote the kernel of the anchored weighted Sobolev space with product weights given by $\boldsymbol{\gamma}$. Then with

$$\begin{aligned}\beta_j &= 1 + \frac{\gamma_j}{3},\\ \omega(\{\cdot\}) &= B_2(\{\cdot\})\end{aligned}$$

the kernel $K_{d,\boldsymbol{\beta},\boldsymbol{\gamma}}$ is the associated shift-invariant kernel of $K_{1:d}$, see Proposition 3.16.

Once a lattice generator is given, the calculation of the integration points is trivial, cf. Section 2.4. After invoking getPoint(**p**, i) the elements of **p** are

$$p_j = \left\{\frac{i\cdot z_j}{n} + \Delta_j\right\}, \quad j = 1, ..., d$$

where the method prepare generated the shift Δ according to the selected mode. In particular, the shift may be $\Delta = \mathbf{0}$.

Basic Idea of the Fast CBC Algorithm

We briefly summarize [Ny07, Chapter 3] to outline the crucial idea behind the fast CBC algorithm. Algorithm 4.3 demonstrates a first naive approach to perform the component-by-component search. There we use that the squared worst case error is given by

$$e^2(Q(\cdot, P_n(\mathbf{z})), B(K_{d,\boldsymbol{\beta},\boldsymbol{\gamma}})) = e^2_{d,\boldsymbol{\gamma},\boldsymbol{\beta}}(\mathbf{z})$$

with

$$e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_1, ..., z_s) := -\prod_{j=1}^{s}\beta_j + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\left(\beta_j + \gamma_j\omega\left(\left\{\frac{k\cdot z_j}{n}\right\}\right)\right). \tag{4.12}$$

To emphasize that in the s-th iteration only the component z_s is changed we rather write $e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_s)$ instead of $e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_1, ..., z_s)$ omitting the preceding components in the argument list.

We easily see that a direct implementation of Algorithm 4.3 would require a number of arithmetical operations in $O(d^2n^2)$. But in iteration s the product

$$\prod_{j=1}^{s}(\beta_j + \gamma_j\omega(\{(k\cdot z_j)/n\}))$$

Algorithm 4.3 First naive component-by-component construction

1: **for** $s = 1, ..., d$ **do**
2: **for** $z_s = 1, ..., n-1$ **do**
3: calculate $e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_s) = -\prod_{j=1}^{s} \beta_j + \frac{1}{n}\sum_{k=0}^{n-1}\prod_{j=1}^{s}\left(\beta_j + \gamma_j \omega\left(\left\{\frac{k \cdot z_j}{n}\right\}\right)\right)$
4: **end for**
5: $z_s = \operatorname{argmin}_{z=1,...,n-1} e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_1, ..., z_s)$
6: **end for**

is fix up to the s-th factor. Hence, if we introduce a vector $\mathbf{p}_{s-1} \in \mathbb{R}^n$ storing the result of multiplying the factors $1, ..., s-1$ for each k,

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

we can immediately reduce the number of arithmetical operations to be in $O(dn^2)$. The prize we pay is that we now need additional memory in $O(n)$ for the vectors $\mathbf{p}_{s-1}$ (we can overwrite the same memory for each iteration s).

Defining the matrix

$$\boldsymbol{\Omega}_n := \left[\omega\left(\left\{\frac{k \cdot z}{n}\right\}\right)\right]_{\substack{z=1,...,n-1 \\ k=0,...,n-1}},$$

the shorthand $\bar{\beta}_s = \prod_{j=1}^{s} \beta_j$ and the worst-case error vector

$$\mathbf{e}^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}} := (e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(i))_{i=1,...,n-1}$$

we may write

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

The crucial discovery of the inventors of the fast CBC algorithm is now that there is a $n \times n$ permutation matrix Π such that, for Π and the submatrix Π' we obtain by deleting the first row and the first column, we get the following:

- the permuted worst-case error vector $\mathbf{E}^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}} := \Pi'^{\top} e^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}}(z_1, ..., z_s)$ takes the form

$$\mathbf{E}^2_{s,\boldsymbol{\gamma},\boldsymbol{\beta}} = -\bar{\beta}_s \mathbf{1}_{n-1\times 1} + \frac{1}{n}(\beta_s \mathbf{1}_{n-1\times n} + \gamma_s \boldsymbol{\Omega}'_n)\mathbf{q}_{s-1}$$

  Here $\boldsymbol{\Omega}'_n$ and $\mathbf{q}_{s-1}$ denote basically the permuted forms of $\boldsymbol{\Omega}_n$ and $\mathbf{p}_{s-1}$, respectively. The matrix $\boldsymbol{\Omega}'_n$ is of the form

$$\boldsymbol{\Omega}'_n = [\omega_0 \mathbf{1}_{n-1\times 1} \quad C_{n-1}]$$

  where C_{n-1} is a circulant matrix. For such matrices matrix-vector multiplications can be done in $O(n\log(n))$ arithmetical operations instead of $O(n^2)$ when using a *Fast Fourier Transform (FFT)*.

- the components $z_1, ..., z_s$ of the lattice generator in the unpermuted space can be computed in constant time from the components $w_1, ..., w_s$ of the lattice generator in the permuted space if we use $O(n)$ memory.

Overall, this leads to a component-by-component search which needs $O(dn\log(n))$ arithmetical operations using $O(n)$ extra memory.

Remark 4.6. For $k \in \{1, ..., \frac{n-1}{2}\}$ the shift-invariance of ω implies

$$\omega\left(\left\{\frac{n-k}{n}\right\}\right) = \omega\left(\left\{\frac{k}{n}\right\}\right).$$

Thus it suffices to search the lattice generator components z_j in the set $\{1, ..., \frac{n-1}{2}\}$ reducing the cost of the fast CBC algorithm by a factor 2.

Implementation

Our implementation of the fast CBC algorithm is basically a translation of the Matlab code provided by [Ny07, Listings 3.1, 3.2 and 3.3, p. 69/70] into C++ code. Thus we content ourself with discussing two pieces of code where translation into C++ code meant more than simple copy and paste.

Line 9 of [Ny07, Listing 3.1] involves the multiplication of two integers. This demands particular attention as the product may become very large, even larger than the capacity of the primitive datatype long (at least on a 32 bit machine).

In Matlab the programmer does not have to bother since Matlab itself is capable of handling arbitrarily large integers. C++ does not provide this capability by default. If one is not aware of this circumstance in advance it is a source of undetected false lattice generator calculation since potential integer overflows are unrecognizable at compile time.

Fortunately, there are numerous implemented solutions to this problem available. We decided to use the small library by McCutchen [MM10] which provides us with proper datatypes.

The second part in [Ny07, Listing 3.1] where we had to do extra work were those pieces of code dealing with fast Fourier transformations. The concerned lines are 15 and 17. First of all, the C++ standard library has no FFT implementations. So again, we had to seek out for existent implementations. Though certainly not the best freely available solution we opt for the FFT routines of the GNU Scientific Library (GSL) [G97]. We decided so as they seemed the easiest to use and required the least practice to get it running.

Abstractly speaking, lines 15 and 17 in [Ny07, Listing 3.1] now do the following: For $m = (n-1)/2$ and a given real valued vector $\mathbf{x} = \mathbf{x}_{0:m-1}$ the fast Fourier transform FFT is utilized to compute the discrete Fourier transform (DFT) of $\mathbf{x}$,

$$\tilde{\mathbf{x}} = \mathrm{FFT}(\mathbf{x}).$$

As $\mathbf{x}$ is real valued the result $\tilde{\mathbf{x}}$ is a *conjugate symmetric* complex valued vector, that is it obeys the symmetry

$$\tilde{x}_k = \tilde{x}^*_{m-k}$$

where * denotes the complex conjugation and $m-k$ is taken modulo m. Moreover, we always have $\tilde{x}_0 \in \mathbb{R}$ and if m is even $\tilde{x}_{m/2} \in \mathbb{R}$, as well. Alternatively, we also say that $\tilde{\mathbf{x}}$ is *half-complex*.

The DFT is computed for another real valued vector $\mathbf{y} = \mathbf{y}_{0:m-1}$. Of course, the resulting vector $\tilde{\mathbf{y}} = \mathrm{FFT}(\mathbf{y})$ obeys the same properties as $\tilde{\mathbf{x}}$. Now for $\tilde{\mathbf{z}}$ being the vector we obtain by component-wise forming the complex product $\tilde{z}_k = \tilde{x}_k \cdot \tilde{y}_k$, the algorithm computes the *inverse* DFT of $\tilde{\mathbf{z}}$ by means of a inverse fast Fourier transform (IFFT),

$$\mathbf{z} = \mathrm{IFFT}(\tilde{\mathbf{z}}) \in \mathbb{R}^m.$$

If we implemented what we described above just as we put it there it would require to temporarily store the vectors $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{y}}$ by means of complex valued datatypes. However, we can halve the required memory if we use an implementation of FFT, say FFT^r, which is specialized for real valued input and respectively a special version of IFFT, IFFT^{hc}, accepting only half-complex input. The GNU Scientific Library comes along with such implementations. They use a special storing scheme which stores a half-complex vector $\tilde{\mathbf{a}}$ in a corresponding real valued vector $\mathbf{a}^{hc}$ of same length, thus halving the consumed memory.

We describe the storing scheme. For this purpose, define a map $\tau_m : \{0,\dots,m-1\} \to \{0,\dots,m-1\}$ by

$$\tau_m : k \mapsto \begin{cases} 0 & \text{if } k = 0 \\ 2k-1 & \text{if } 0 < k < \frac{m+1}{2} \\ \frac{3(m+1)}{2} - 2k & \text{if } \frac{m+1}{2} \le k < m \end{cases}$$

for m odd and

$$\tau_m : k \mapsto \begin{cases} 0 & \text{if } k = 0 \\ 2k-1 & \text{if } 0 < k \le \frac{m}{2} \\ \frac{3}{2}m - 2(k-1) & \text{if } \frac{m}{2} < k < m \end{cases}$$

if m is even. Then $\tilde{\mathbf{a}}$ and $\mathbf{a}^{hc}$ are related as follows:

- For $k = 0$ we have $\mathrm{Re}\,\tilde{a}_k = a^{hc}_k$ and $\mathrm{Im}\,\tilde{a}_k = 0$.
- For m odd and $0 < k < m$ it holds true that

$$\mathrm{Re}\,\tilde{a}_k = a^{hc}_{\tau_m(k)}$$

$$\mathrm{Im}\,\tilde{a}_k = \begin{cases} a^{hc}_{\tau_m(k)+1} & \text{if } 0 < k < \frac{m+1}{2} \\ -a^{hc}_{\tau_m(k)+1} & \text{if } \frac{m+1}{2} \le k < m \end{cases}.$$

- For m even we have the same relation except for $\mathrm{Im}\,\tilde{a}_{m/2} = 0$.

Now let $\tilde{\mathbf{x}}$, $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$ as given above. We get the vectors $\mathbf{x}^{hc}$ and $\mathbf{h}^{hc}$ by applying FFT^r to $\mathbf{x}$ and $\mathbf{y}$, respectively. Hence the crucial point here is how to obtain the vector $\mathbf{z}^{hc}$ from $\mathbf{x}^{hc}$ and $\mathbf{y}^{hc}$. Based on the storing scheme we derive that $\mathbf{z}^{hc}$ is calculated as follows:

- for $k = 0$ we calculate $z_k^{hc} = x_k^{hc} \cdot y_k^{hc}$
- for odd k with $0 < k < m-1$ we calculate
$$z_k^{hc} = x_k^{hc} \cdot y_k^{hc} - x_{k+1}^{hc} y_{k+1}^{hc}.$$
- for even k with $0 < k < m-1$ we calculate
$$z_k^{hc} = x_{k-1}^{hc} \cdot y_k^{hc} + x_k^{hc} \cdot y_{k-1}^{hc}.$$
- if m is even we additionally calculate $z_{m-1}^{hc} = x_{m-1}^{hc} \cdot y_{m-1}^{hc}$.

Shortcomings of the GSL FFT Routines

For the facts we refer to in this paragraph, see [G97]. The cost in $O(n \log n)$ which is claimed for the fast CBC algorithm basically founds on the fast Fourier transformation to compute the DFT of a vector of size n in $O(n \log n)$ arithmetical operations instead of $O(n^2)$.

The FFT routines of the GSL follow up on this promise merely with restrictions as we are going to sketch briefly in the following. Assume we are given $n \in \mathbb{N}$. Then the number of arithmetical operations needed for computing the DFT of a vector $\mathbf{x}$ of size n by means of the GSL FFT routine is generally bounded by

$$Cn \sum_{k=1}^{s} f_k$$

where $C > 0$ is some constant and $n = f_1 f_2 ... f_s$ is a factorization of n. But then it depends highly on the given factorization whether the cost are close to $Cn \log n$ or rather close to Cn^2. For instance, if $n = 2^m$ for $m \in \mathbb{N}$ then we actually see a bound of the form $C' n \log n$ for $C' = C / \log 2$.

From a practical point of view it is even more important to note that the size of the constant C heavily depends on the given factorization. Only for factors $f = 2, 3, 4, 5, 6$ the GSL implementation uses heavily optimized subroutines, otherwise a rather slow subroutine for general factors f is used.

However, see [Ny07, p. 76] for references that it is possible to implement FFTs with $O(n \log n)$ cost for general n.

4.4 Multilevel Parameter Strategies

The library comes along with three distinct strategies ϑ_1, ϑ_2 and ϑ_3 controlling the choice of the multilevel parameter values. As we standardized the communication between the strategies and other components of the library, all strategy implementations use the user-specified error bound ε to determine the parameter values.

The first two strategies are very simple and purely for experimental purposes. The third strategy ϑ_3 is adaptive in the sense that it chooses the dimension and numbers of integration points for the multilevel algorithms based on a heuristic estimate of the the rooted mean squared integration error. This estimate aims at meeting the user-specified error bound ε.

To be compatible with our rank-1 lattice point generator implementation, all three strategies choose only prime numbers for the numbers of integration points.

S1. No-multilevel

Given the input ε the no-multilevel strategy chooses one dimension $d = \varepsilon^{-1/2}$ and one number of n integration points with n the next prime number greater than $\varepsilon^{-1/2}$. Of course, combined with our pseudorandom integration point generator the no-multilevel strategy turns our multilevel algorithm into the classical Monte-Carlo method. Hence it provides a baseline for other strategies.

S2. Strategy Based on Theoretical Results

Niu et al.[NHMR10, Proof of Theorem 3] achieve their results by a geometric growth in the dimensions and geometric decay in the number of integration points. We form a strategy ϑ_2 from this as follows: If we are given an error bound ε we take the number of levels L as

$$L = \left\lceil \log(\varepsilon^{-1}) \right\rceil . \tag{4.13}$$

Then for each level $l = 1, ..., L$ we specify the dimension to be

$$d_l = 2^l.$$

The number of integration points n_l is specified to be the next prime number greater than or equal to 2^{L-i+1}.

S3. Adaptive Strategy Using Bias and Variance Estimates

Both the strategies we considered above lack in the possibility to check for the integration error of the multilevel algorithm on the run. The following adaptive strategy tries to overcome this deficiency and was proposed in Giles et al.[GW09]. Let L denote the number of different levels the strategy triggers to be computed. Then let $\mathbf{d} = (d_1, ..., d_L)$ and $\mathbf{n} = (n_1, ..., n_L)$ denote the corresponding dimensions and numbers of integration points, respectively. Further put $M = 32$ as suggested by Giles et al. Now, for ϑ_3 working as intended there are two requirements on the used method of point generation P. First, the component P must be such that the level quadrature rules $Q_l = Q(l, f, P, \vartheta_3)$, $l = 1, ..., L$, become independent random variables, having the property that

$$\begin{aligned} \mathrm{E}[Q_l] &= I(\Psi_{1:d_1,\mathbf{a}} f) && \text{for } l = 1, \\ \mathrm{E}[Q_l] &= I(\Psi_{1:d_l,\mathbf{a}} f - \Psi_{1:d_{l-1},\mathbf{a}} f) && \text{for } l > 1. \end{aligned}$$

Second, P has to be such that for $i = 1, ..., M$ the repeated level computations $Q_l^{(i)}$ yield independent realizations of Q_l.

Denote by e the mean squared integration error (2.17) for the multilevel algorithm $Q^{ml} = Q^{ml}(f, P, \vartheta, \varepsilon)$ and some fixed integrand f, that is

$$e^2 = e^2(Q^{ml}, f).$$

Further write

$$b^2 = b^2(Q^{ml})$$

for the bias of Q^{ml}. As we generate the integration points independently for each level, we obtain

$$\mathrm{Var}(Q^{ml}) = \sum_{l=1}^{L} \mathrm{Var}(Q_l).$$

The idea is now to pick estimators $\widehat{b}$ and $\widehat{V}_l$ for b and $\mathrm{Var}(Q_l)$ respectively, and choose the number of levels L, the dimensions $\mathbf{d}$ and the numbers of integration points $\mathbf{n}$ such that

$$\widehat{b}^2 + \sum_{l=1}^{L} \widehat{V}_l \leq \varepsilon^2.$$

For $\mathrm{Var}(Q_l)$ we take the usual estimator

$$\widehat{V}_l = \frac{1}{M-1} \sum_{i=1}^{M} (Q_l^{(i)} - Q_l^{\mathrm{res}})^2 \tag{4.14}$$

with $Q_l^{\mathrm{res}} = 1/M \sum_{i=1}^{M} Q_l^{(i)}$. For the bias b the authors propose the estimator

$$\widehat{b} = \max\left\{ \frac{1}{2} |Q_{l-1}^{\mathrm{res}}|, |Q_l^{\mathrm{res}}| \right\}. \tag{4.15}$$

Finally, assuming that b^2 and $\mathrm{Var}(Q_l)$ contribute equally to e^2 we get a strategy ϑ_3 acting as indicated in Algorithm 4.4.

Remark 4.7. We have no certainty on the quality of $\widehat{b}$. Further the assumption that b^2 and $\mathrm{Var}(Q_l)$ contribute equally to e^2 is heuristic, as well as the greedy approach in line 5 of Algorithm 4.4. Hence strategy ϑ_3 yields a heuristic for the integration error. But as numerical experiments show, we can hope that it leads to $e \leq \varepsilon$ most of the time, see Section 5.4.

4.5 Estimating Orders of Convergence

Let f be a subroutine representing an integrand, P a method of integration point generation, and ϑ a strategy for multilevel parameter choice. In this section we describe an experimental design which allows to estimate exponents of convergence for the integration error of the multilevel algorithm $Q^{ml}(f, P, \vartheta, \cdot)$ by means of a linear regression.

For this purpose we require to be given a target value for $I(f)$. Further, we require to be given a regression model which specifies an assumed functional relation between error and cost up to constants and exponents.

Algorithm 4.4 Adaptive Strategy Using Bias and Variance Estimates

1: ϑ_3 sets $L \leftarrow 1, d_1 \leftarrow 2, n_1 \leftarrow 2$ and then puts $l \leftarrow 1$ into level queue.
2: Q^{ml} pops level l from queue and computes the M realizations $Q_l^{(i)}$. Then Q_l^{res} is taken as level result.
3: ϑ_3 updates $\widehat{V}_l$ for $l = 1, ..., L$.
4: **if** $\sum_{l=1}^{L} \widehat{V}_l \geq \varepsilon^2/2$ **then**
5: $\quad\vartheta_3$ doubles n_l at level l with maximal $\widehat{V}_l/(n_l d_l)$ and puts this l into level queue.
6: $\quad$Goto line 2.
7: **else**
8: $\quad$**if** $\widehat{b} \geq \varepsilon/\sqrt{2}$ **then**
9: $\quad\quad\vartheta_3$ sets $L \leftarrow L + 1$ and puts $l \leftarrow L$ into level queue.
10: $\quad\quad$Goto line 2.
11: $\quad$**end if**
12: **end if**

Data Acquisition

We execute the following procedure:

Input: user-specified error bounds $\varepsilon_1 > ... > \varepsilon_M$, a sample size N, f, P, ϑ
for $j = 1, ..., M$ **do**
for $i = 1, ..., N$ **do**
$Q_{i,j}^{ml} \leftarrow Q^{ml}(f, P, \vartheta, \varepsilon_j)$.
end for
end for

Thereby we assume that for each $j = 1, ..., M$ we can treat $(Q_{i,j}^{ml})_{i=1,...,N}$ as a realization of N i.i.d. copies of some random variable which describes the outcomes of $Q^{ml}(f, P, \vartheta, \varepsilon_j)$.

Remark 4.8. The above proceeding is justified, e.g. if we use $P = P_{\text{rand}}$ or $P = P_{\text{lattice}}$ with independent shift sampling for each level repetition.

For each outcome $Q_{i,j}^{ml}$ we record the cost $C_{i,j}^{ml}$ the multilevel algorithm run produced. Thereby we do not record actual runtimes or count arithmetic operations but follow an adapted version of the variable subspace sampling cost model we introduced in Section 2.2.

The multilevel algorithm performs evaluations of f in the form of $f(\mathbf{x}_{1:d_2})$ or $f(\mathbf{x}_{1:d_2}) - f(\mathbf{x}_{1:d_1})$ where d_1, d_2 denote dimensions with $d_1 < d_2$. In both cases we count the evaluation cost with d_2. This is reasonable for the integrands we encounter in this thesis since calculating $f(\mathbf{x}_{1:d_2}) - f(\mathbf{x}_{1:d_1})$ roughly produces only the cost of calculating $f(\mathbf{x}_{1:d_2})$ while getting $f(\mathbf{x}_{1:d_1})$ almost for free as a byproduct.

Let $L(i, \vartheta, \varepsilon_j)$ denote the number of levels the strategy ϑ determined in run i for the user-specified error bound ε_j. Analogously, we denote by $\mathbf{n}(i, \vartheta, \varepsilon_j)$, $\mathbf{d}(i, \vartheta, \varepsilon_j)$ and $\mathbf{r}(i, \vartheta, \varepsilon_j)$ the numbers of integration points, the dimensions and the number of repeated level computations, respectively. Following the above argumentation on the evaluation cost we record the cost $C_{i,j}^{ml}$ such that

$$C_{i,j}^{ml} = \sum_{l=1}^{L(i,\vartheta,\varepsilon_j)} r_l(i, \vartheta, \varepsilon_j) \cdot n_l(i, \vartheta, \varepsilon_j) \cdot d_l(i, \vartheta, \varepsilon_j).$$

Data Interpretation

To prepare the observed data we require the user to specify a referential value R of $I(f)$ and then calculate the observed integration errors

$$E_{i,j}^{ml} = |Q_{i,j}^{ml} - R|$$

for $i = 1, ..., N$ and $j = 1, ..., M$. Then for each user-specified error bound ε_j we compute the sample mean square error

$$\widehat{E}_j^2 = \frac{1}{N} \sum_{i=1}^{N} (E_{i,j}^{ml})^2$$

which is an unbiased estimator of $e^2(Q^{ml}(f, P, \vartheta, \varepsilon_j), f)$. Furthermore we compute the sample mean square cost

$$\widehat{C}_j^2 = \frac{1}{N} \sum_{i=1}^{N} (C_{i,j}^{ml})^2.$$

We estimate confidence intervals both for $e^2(Q^{ml}(f, P, \vartheta, \varepsilon_j), f)$ as well as for the cost produced for given ε_j. Therefor we split our sample into K batches of size $N_b = N/K$. Then for $A = E$ or $A = C$ we calculate for each batch k the batch average

$$\widehat{A}_j^2(k) = \frac{1}{N_b} \sum_{i=1+(k-1)N_b}^{kN_b} (A_{i,j}^{ml})^2$$

which is approximately Gaussian for N_b large enough. The mean over the batch averages obviously is $\widehat{A}_j^2$ and the standard estimator for the variance of the batch averages is given by

$$\hat{\sigma}_j^2 = \frac{1}{K-1} \sum_{k=1}^{K} (\widehat{A}_j^2(k) - \widehat{A}_j^2)^2.$$

Using the Student t-distribution with $K - 1$ degrees of freedom and a significance level α we obtain a confidence interval

$$(\hat{A}_j^2 - \delta\hat{A}_j^2, \hat{A}_j^2 + \delta\hat{A}_j^2)$$

with $\delta\widehat{A}_j^2 = t_{1-\alpha,K-1}(\sqrt{\widehat{\sigma}_j^2/K})$. In this case the actual mean square integration error or the actual mean cost lie in the corresponding confidence interval with probability $1 - \alpha$.

Experience shows that a batch size $N_b \geq 15$ is sufficient to get reliable confidence intervals, see [KP92, p. 312].

Let us now finally turn to measuring orders of convergence. If theoretical results are known for integration problems the relation between error e and cost Γ is usually of the form

$$e = O(\Gamma^{-p})$$

or

$$e = O(\Gamma^{-p}(\log \Gamma)^q).$$

This motivates to do a linear regression in the logarithmized error and cost data

$$\begin{aligned}\mathbf{X} &:= (\log(\widehat{C}_j))_{j=1,\dots,M}\\ \mathbf{Y} &:= (\log(\widehat{E}_j))_{j=1,\dots,M}.\end{aligned}$$

Following the cost-error relations sketched above possible models might then be

$$\begin{aligned}&\mathbf{Y} = \beta_0 + \beta_1 \mathbf{X}\\ \text{or}\quad &\mathbf{Y} = \beta_0 + \beta_1 \log(\mathbf{X}) + \beta_2 \mathbf{X}\end{aligned}$$

with $\exp(\beta_0)$ estimating the constant hidden behind the big-O notation and $-\beta_1$ and β_2 estimating p and q, respectively.

5 Numerical Experiments for the Asian Call Option

In Chapter 3 we presented anchored weighted Sobolev spaces for which theoretical results on the order of convergence of deterministic multilevel rank-1 lattice rules and rank-1 lattice rules are available, both using deterministically shifted lattices.

But important integrands fitting in our setting R1 (see Section 2.1) are neither contained in an anchored weighted Sobolev space nor are there yet any theoretical results for suitable function spaces. In particular, this comprises the Asian Call Option in the Black Scholes model where we recall that it is given by the payoff

$$\varphi_{AC}(S) = e^{-rT} \max\left(\frac{1}{T}\int_0^T S_t \,\mathrm{d}t - K, 0\right)$$

with S the geometric Brownian motion on the time interval $[0, T]$. Hence numerical experiments are currently the only way to explore the potential of rank-1 lattices for the calculation of the option prize.

We are particularly interested in how well the combination of rank-1 lattice rules and the multilevel technique performs. For multilevel shifted rank-1 lattice rules recent numerical experiments by Giles and Waterhouse [GW09] yield promising results. For the Asian call option the experiments reveal the multilevel algorithm using shifted rank-1 lattice rules to have level variances declining significantly faster than with the building blocks being classical Monte Carlo rules.

Motivated by the former we execute experiments to measure orders of convergence for shifted rank-1 lattice rules both with and without the multilevel technique, comparing these results to classical Monte Carlo integration. Then in a second line of experiments we examine the impact of deterministic shifts on observed integration errors.

5.1 Preliminaries

For the reader's convenience, we restate from Section 4.2 the implementations of the Asian call option related to SDE setting R1:

F1: $f_1(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(L^{(l)}(\mathbf{x}_{1:d_l}))$,

F2: $f_2(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(\hat{W}^{(l)}(\mathbf{x}_{1:d_l}))$,

F3: $f_3(\mathbf{x}_{1:d_l}) = \varphi_{AC} \circ \Gamma(K^{(l)}(\mathbf{x}_{1:d_l}))$.

For classical Monte Carlo integration it makes no difference whether we use the Lévy-Ciesielski partial sum $L^{(l)}$ or $\hat{W}^{(l)}$ as both share the same distribution. However, for

lattice based methods there a numerous numerical experiments where the former performed much better than the latter, see [G08] for further references. Hence we focus on implementation F1 and F3 in the following.

Recall that internally F3 has to apply a quadrature rule to approximate the integral over the given approximate path of geometric Brownian motion. With our current implementation this actually leads to cost increasing nonlinearly in the evaluation dimension d_l. However, we take the liberty to assume the following fictitious modification:

F3a: We assume the internal quadrature rule to be given by a subroutine which always approximates the pathwise integral in a number of operations increasing linear in d_l. This version F3a then has cost in $O(d_l)$.

Our motivation for F3a is as follows: the fictitious subroutine can not calculate the pathwise integral better than F1 determines the corresponding integral anyway. Hence the latter cost assumption allows to point out more clearly how well F3 utilizes the rank-1 lattice integration points compared to F1.

As parameters for the Asian call option we choose

$$T = 1,\ s_0 = 2,\ r = 0.05,\ \sigma = 0.5,\ K = 2.$$

For this choice of parameter values [V02] states

$$E[\varphi(S)] \approx 0.246416$$

which we take as referential value.

All experiments described below follow the basic experimental setting given in Section 4.5. For confidence interval estimation we use a significance level of $\alpha = 0.01$. Regarding the choice of a regression model we find inspiration by Remark 2.34. Though referring to SDE setting R2 we nevertheless expect to observe experimentally roughly the same orders of convergence for classical Monte Carlo integration and implementations referring to SDE setting R1. Moreover, we hope to observe structurally the same cost-error evolution with lattice based methods. In particular, we suppose that the $\log N$ term does not only show up in the upper bound (2.24) but is rather a crucial characteristic of the cost-error evolution when the multilevel technique is involved.

Hence we are motivated to assume a cost-error relation

$$e(N) = c \cdot (\log N)^{\beta_1} \cdot N^{\beta_2}$$

with $c = \exp(\beta_0)$ and consequently apply the regression model

$$\mathbf{Y} = \beta_0 + \beta_1 \log(\mathbf{X}) + \beta_2 \mathbf{X}$$

with $\mathbf{X}$ and $\mathbf{Y}$ according to Section 4.5. Note that we slightly abuse the notation as β_0, β_1 and β_2 in the regression model are merely estimators of the corresponding quantities in the assumed cost-error relation. We apply the regression model regardless of whether or not the experiment involves the multilevel technique. In the latter case we expect $\beta_1 = 0$ for ideal data.

For lassical Monte Carlo integration we use the pseudorandom integration point generator P_{rand} and for lattice based methods the rank-1 lattice integration point generator P_{lattice}. Both are described in detail in Section 4.3. To obtain integration points suitable for our purposes we transform the generated integration points in both cases by applying component-wise the inverse of the cumulative Normal distribution function using the implementations provided by [GSL11].

The basic configuration of P_{lattice} is as follows: As shift-invariant kernel we choose the Korobov kernel

$$\omega(x) = B_2(x) = 2\pi^2(x^2 - x + \frac{1}{6}).$$

For the weight sequences $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ we set

$$\begin{aligned}\beta_j &= 1,\\ \gamma_j &= C_\gamma \cdot j^{-4}.\end{aligned}$$

with $C_\gamma = 1$.

Remark 5.1. Note that we have no a priori information at hand how to choose the kernel and the weights here. For the kernel we take B_2 as it is the one we encounter with the theoretical results given in Chapter 3. For the weights we played around a bit. At least, to choose $\boldsymbol{\gamma}$ declining in j with polynomial order reflects the theoretical insight we gained in Section 3.2.

5.2 Results for Multilevel Shifted Rank-1 Lattice Rules

The purpose of this experiment is to learn how well multilevel shifted rank-1 lattice rules perform, compared with quadrature rules where the multilevel technique is omitted or classical Monte Carlo sampling is done for the integration points.

Additionally to the basic setting we configure P_{lattice} such that for each level computation a new shift is randomly generated according to the uniform distribution. Furthermore, we recall from Section 4.4 the strategy ϑ_1 which uses always only on level (thus turning the multilevel technique off). We also recall strategy ϑ_2 where the dimensions grow geometrically with the level and the numbers of integration points decay geometrically with the level. Then we run our multilevel algorithm with the following four combinations:

1. **no-ml rand:** (classical Monte Carlo rule) $\vartheta = \vartheta_1$, $P = P_{\text{rand}}$
2. **no-ml lattice:** (shifted rank-1 lattice rule) $\vartheta = \vartheta_1$, $P = P_{\text{lattice}}$
3. **ml rand:** (multilevel Monte Carlo rule) $\vartheta = \vartheta_2$, $P = P_{\text{rand}}$
4. **ml lattice:** (multilevel shifted rank-1 lattice rule) $\vartheta = \vartheta_2$, $P = P_{\text{lattice}}$

For combinations involving ϑ_1 we spend 10 different user-specified error bounds per combination to get results for different cost values, combinations involving ϑ_2 get spend 15 user-specified error bounds each. We repeat this procedure $N = 1000$ times for each combination.

Using implementation F1 the results are illustrated in the Figures 5.1. We see that multilevel rank-1 lattice rules clearly outperform the other methods, reaching an exponent $\beta_2 = -0.71$ which dominates the cost-error evolution on an asymptotic scale. Moreover, we note that for classical Monte Carlo rules and multilevel Monte Carlo rules the experimental results are consistent with the theoretic results given by (2.23) and (2.24), respectively. In particular, the upper bounds seem to be matching bounds for the corresponding quadrature rules. Observing exponents β_1 different from 0 for classical Monte Carlo and shifted rank-1 lattice rules is due to a slight overfitting of the data by the regression model. Particularly the data points at low cost values do not perfectly lie on a line in the loglog plot.

Remark 5.2. To transform the plotted cost given in Figure 5.1 roughly into real runtime cost one has to multiply by factor of 643 nsec $= 643 \cdot 10^{-9}$ sec.

We run the experiment for implementation F3, as well. For F3a the results suggest that the principal component construction used with F3 utilizes rank-1 lattices even better than the Brownian bridge construction used with F1, see Figure 5.2. However, if we fully incorporate the cost of the quadrature rules which F3 uses internally, then we get the results as given in Table 5.1. As the cost grow quadratically with the evaluation dimension, now the β_2 exponents are halved.

method	c	β_1	β_2
no-ml rand	0.36	0.04	-0.12
no-ml lattice	0.28	0.20	-0.22
ml rand	0.28	0.63	-0.25
ml lattice	0.27	0.94	-0.4

Table 5.1: Comparing orders of convergence for F3.

5.3 Effect of Constant Factors in the Weights

When we prepared the experiments described in the previous section we tried only different γ weights in terms of order of decay. Later we had the idea to test the effect of constant factors in the γ weights on the performance of multilevel shifted rank-1 lattice rules. Results based on F1 are given in Table 5.2.

C_γ	c	β_1	β_2
0.001	0.46	0.84	-0.68
0.05	0.46	0.91	-0.71
0.2	0.45	0.98	-0.73
1	0.44	0.94	-0.71
2	0.44	0.84	-0.64
20	0.44	0.41	-0.54

Table 5.2: Effect of constant C_γ on the orders of convergence, using implementation F1.

We see rather no improvement. But increasing C_γ seems to counteract the multilevel technique. With β_1 becoming smaller, the cost-error curve tends to a form which is similar

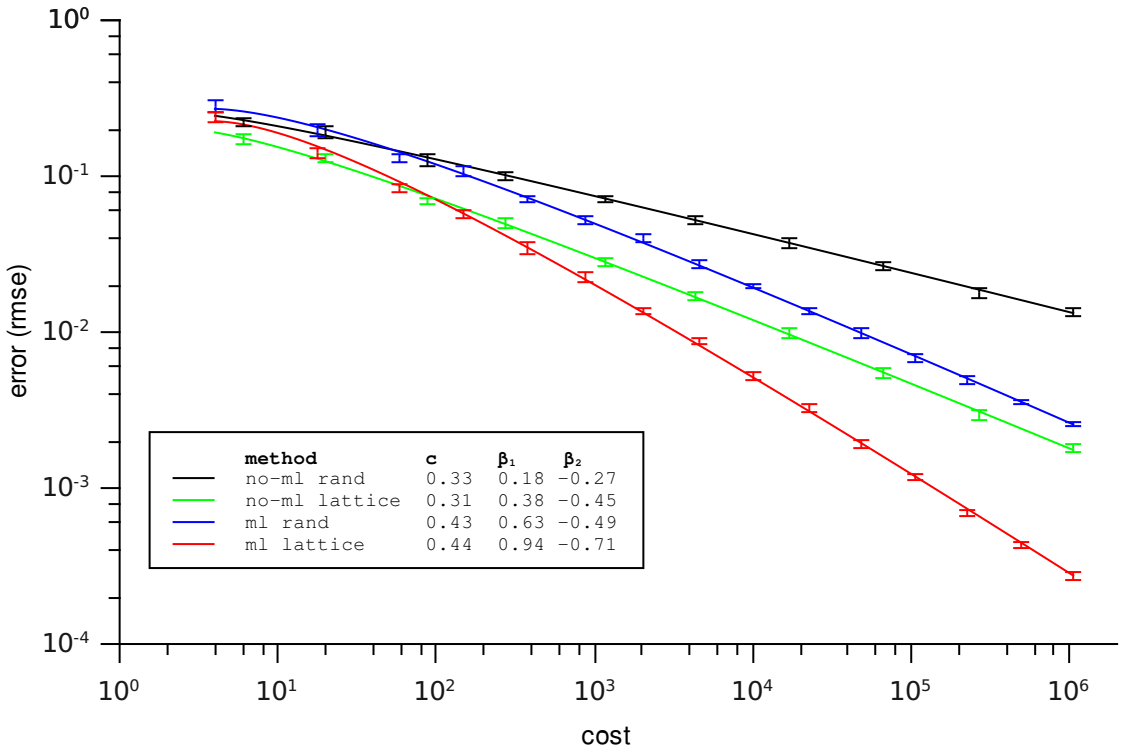

Figure 5.1: Comparing orders of convergence for F1.

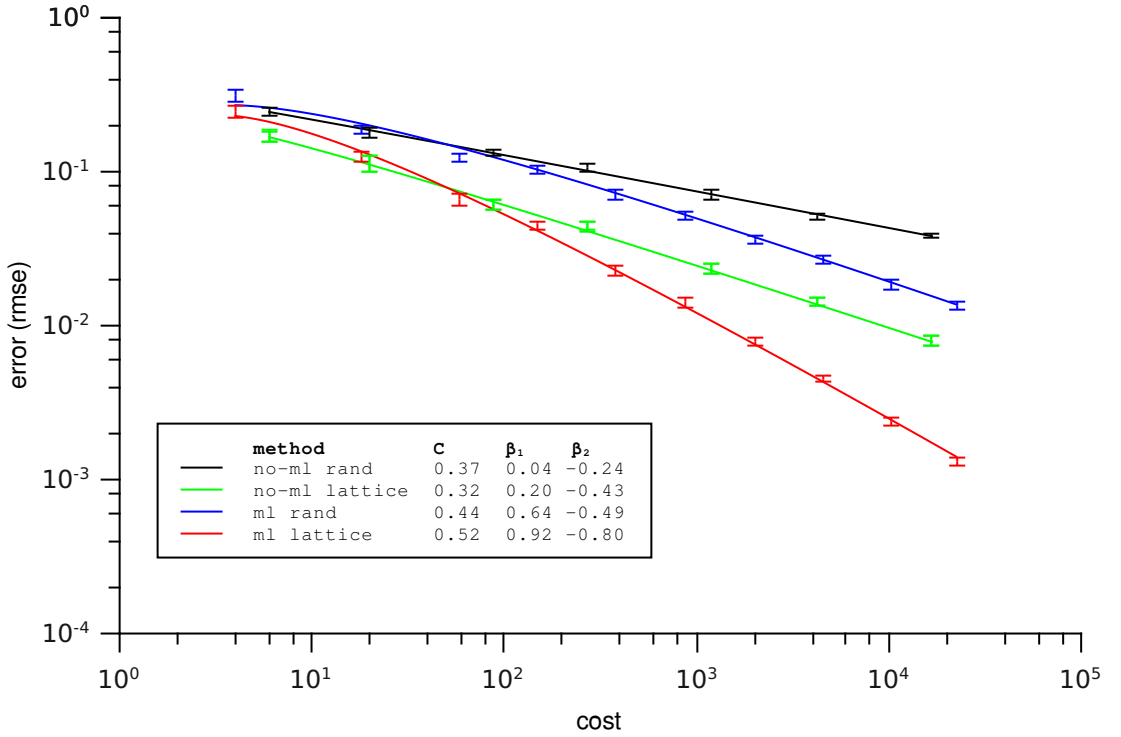

Figure 5.2: Comparing orders of convergence for F3a.

as in the case of shifted rank-1 lattice rules. The β_2 exponent seems to tend towards the value we observed for shifted rank-1 lattice rules in the previous experiment.

5.4 Results for Adaptive Multilevel Parameter Strategy

So far we determined the multilevel parameters for the multilevel shifted rank-1 lattice rule statically, in the sense that we only use a priori evidence from theory how to choose the parameters. However, using strategy ϑ_2 we do not have any indication whether the choice is optimal for a given integrand. Moreover, we have no estimate at runtime which error the multilevel algorithm currently produces.

Mainly the latter makes this strategy useless in practice. Hence we pass on to testing the adaptive strategy ϑ_3 we introduced in paragraph S3 of Section 4.4. Recall that using this strategy the evaluation dimension still increases geometrically with the level but the number of levels and the number of integration points in each level is determined adaptively on the basis of a heuristic rooted mean square error estimate. Further note that this strategy uses a number $M = 32$ of level repetitions.

The experiment's result is given in Figure 5.3. The plotted data points are the rooted mean square errors given at rooted mean square cost observed for user-specified error bounds

$$\varepsilon = \frac{1}{100}, \frac{1}{400}, \frac{1}{900}, \frac{1}{1600}, \frac{1}{2500}, \frac{1}{3600}, \frac{1}{4900}, \frac{1}{6400}, \frac{1}{8100}, \frac{1}{10000}.$$

The mean values are taken over a sample size of $N = 600$.

A first interesting observation is that the current multilevel parameter strategy seems to produce "flat regions" where increasing the cost does not diminish the error. From Table 5.3 we learn further that for the user-specified error bound $\varepsilon = 1/6400$ the corresponding rooted mean square error is even larger than ε.

The exponent $\beta_2 = -1.03$ estimated by the regression indicates that we improved the multilevel shifted rank-1 lattice rule on an asymptotic scale. Note that the estimate stays roughly the same even when we eliminate the data points being far away from the regression curve. But we are not sure how trustworthy this estimate is. In particular, due to the results we are less committed to the regression model than in the previous experiments.

5.5 Shift Sensitivity of Rank-1 Lattice Rules

With multilevel shifted rank-1 lattice rules the shift is merely used as an element to introduce randomization. In this way we conceal that we do not now how to choose a good shift deterministically. Nevertheless we now try to explore how small we would be able to make the integration error if we knew a good shift. It is reasonable to start experimenting with rank-1 lattice rules, omitting the multilevel technique, as we can not expect the same shift to have equal effects in each level in the first place.

We run the following experiment. Using implementation F1 and strategy ϑ_1 we configure P_{lattice} such that the shift is now manually set, resulting in a rank-1 lattice rule with deterministic shift. Then we limit ourself to examine shifts on the diagonal. In preparation of the experiment we found that the error reacts on changes in the shift up to the

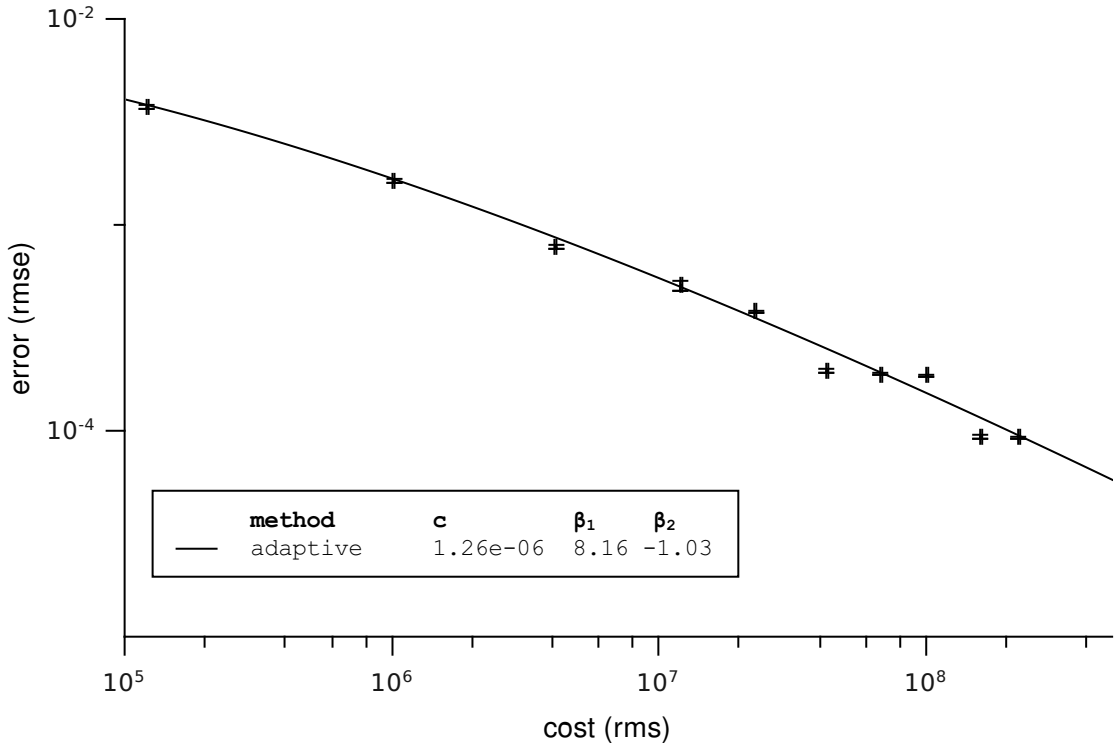

Figure 5.3: Adaptive Strategy S3 applied to F1.

e (rmse)	ε	$e - \varepsilon$
3.87e-03	1/100	-6.13e-03
1.67e-03	1/400	-8.31e-04
7.90e-04	1/900	-3.21e-04
4.96e-04	1/1600	-1.29e-04
3.88e-04	1/2500	-1.25e-05
1.93e-04	1/3600	-8.51e-05
1.89e-04	1/4900	-1.49e-05
1.85e-04	1/6400	2.84e-05
9.27e-05	1/8100	-3.07e-05
9.34e-05	1/10000	-6.61e-06

Table 5.3: Difference between rms error e and user-specified error bound ε.

fourth digit after the decimal point. Hence we start with $\Delta = 0.0001$ and successively increase the shift by 0.0001 until we reach $\Delta = 0.9999$, resulting in a number of $10,000$ runs of the multilevel algorithm each time changing the shift.

Results for dimensions $d = 16, 64$ and numbers of integrations points $n = 17, 37, 67$ are illustrated in Figure 5.4. On the x-axis we plot the values of Δ, on the y-axis we plot the absolute integration error. We see that the shift heavily affects the integration error. There a shifts where the computed integral value almost matches the reference value (which is specified up to the sixth digit after the decimal point). The number of those shifts approximately coincides with n. However, what we find irritating in the data is that we observe greater minimal, maximal and average errors when we increase the dimension d but keep the number n of integration points fixed, see Table 5.4. This is rather counterintuitive since larger values of d correspond to finer path approximations.

d	n	min e	max e	avg e
16	17	4,70e-06	0,9051	0,0841
16	37	3,08e-06	0,4259	0,0435
16	67	6,98e-07	0.2230	0.0228
64	17	1,60e-05	1,1634	0,1111
64	37	1.79e-05	0.5626	0.0690
64	67	9,27e-06	0,2999	0,0346

Table 5.4: Minimal, maximal and average error for tested shifts.

We conclude this section by noting that we applied the same experimental setting as above to multilevel rank-1 lattice rules with deterministic shift. We did not examine the results for all shifts, but for those we did we roughly found the situation exemplarily illustrated by Figures 5.5 and 5.6. Note that in both plots the ith data point corresponds to a multilevel algorithm run using i levels. The algorithm runs which use two levels ($L = 2$) most of the time reach the best integration error in the range of tested numbers of levels L. Then integration errors worsen, going up and down for some values of L and eventually decrease steadily with L.

If we do a regression as in the previous sections, but now only for the range of level numbers where we see a stable error decline, we observe values of β_2 only up to 0.5, which is less than what we observed in the randomized case in Section 5.2. We conclude that if it was possible to search for good shifts this would have to be done on a per-level basis.

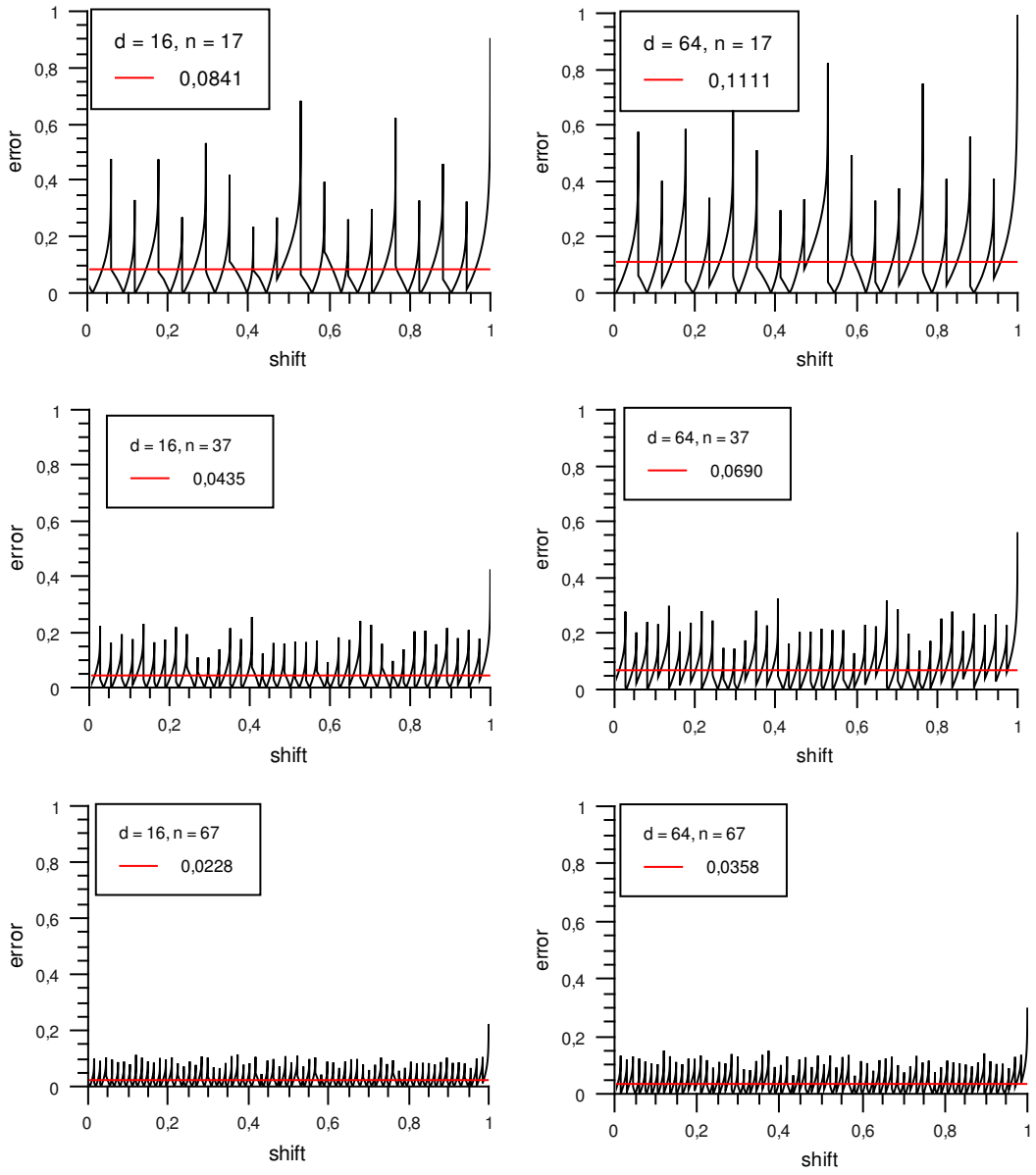

Figure 5.4: Shift sensitivity of rank-1 lattice rule using F1. The red bar shows the average error over all tested shifts.

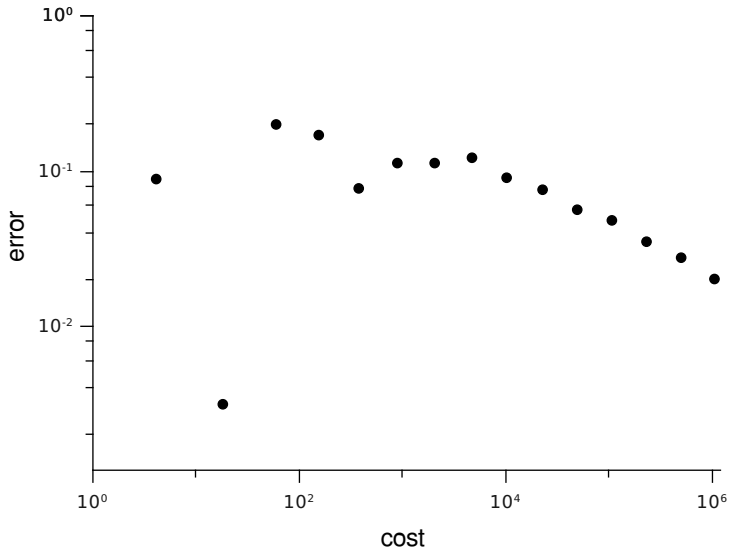

Figure 5.5: Multilevel rank-1 lattice rule using F1 and shift $\Delta = 0.8125$.

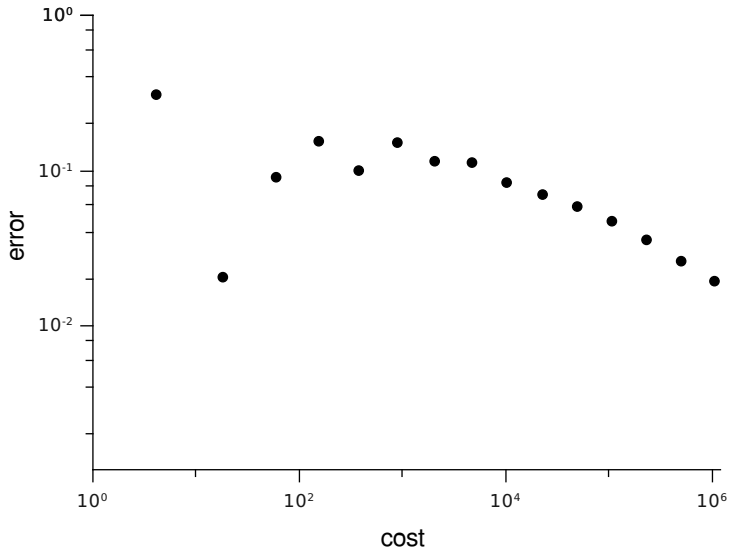

Figure 5.6: Multilevel rank-1 lattice rule using F1 and shift $\Delta = 0.4$.

6 Prospects

We give a few examples of open problems and possible future work related to this thesis.

How to Find a Good Shift

In our experiments we saw that the shift heavily affects the quality of deterministic rank-1 lattice rules and multilevel rank-1 lattice rules. However, neither are there theoretical results nor are there heuristic known yet how to find a good shift. We would find it fascinating to find at least a heuristic for this purpose. But even if we had one we would probably still have to rely on randomization in order to get error estimates.

Proper Choice of Weights and a Kernel

We also saw in our experiments that we have an influence on the quality of multilevel shifted rank-1 lattice rules by the choice of the weights on which the lattice generator is based. We did not alter the kernel, but this would certainly have an effect, too. However, we are in the situation that a given integrand usually will not give any indication how to choose the weights the best.

Experiments for Other Integrands

Our numerical experiment were limited to the case of the Asian call option in the Black-Scholes model. From a practical point of view it would be interesting to test multilevel shifted rank-1 lattice rules as well for more complicated, but more realistic models for pricing of options. A particular example would be the Heston model. There the price process of the underlying asset is no longer modelled by a geometric Brownian motion but by a process which solves the SDE

$$\mathrm{d}X_t = rX_t\mathrm{d}t + \sqrt{v_t}X_t\mathrm{d}W_t^X$$

where v_t, the instantaneous variance, is given by

$$\mathrm{d}v_t = \kappa(\theta - v_t)\mathrm{d}t + \xi\sqrt{v_t}\mathrm{d}W_t^v.$$

In the above equations W^X and W^v are Wiener processes with correlation ρ, and r, κ, θ and ξ are situation-specific parameters.

Further Development of the Library

Our C++ library in its present form offers the opportunity for improvement at a number of points:

- The random number generation is currently inapplicable with parallelized computation. This forces us to use one sequence of random numbers for all threads and we are giving away a lot of performance. However, it is already well studied how to configure the Mersenne Twister to generate uncorrelated pseudo random numbers for computation executed in parallel, see [MN00]. Hence there is only some extra implementational work to do.

- The strategy itself how we split parts of the multilevel algorithm into multiple programme threads is rather rudimentary in its current version. But here it demands probably a bit more hard work to design things more neatly, if possible at all.
- The library could be enhanced by further methods for integration point generation, e.g. methods based on scrambling nets.
- It would be reasonable to replace the GSL routines we used for fast Fourier transform by the implementations [FJ05] which seem to be state-of-the-art among the freely available implementations.

Bibliography

[A50] N. Aronszajn (1950): *Theory of Reproducing Kernels*, Transactions of the American Mathematical Society, Vol. 68(3), 337-404.

[CDMR08] J. Creutzig, S. Dereich, T. Müller-Gronbach, K. Ritter (2008): *Infinite-dimensional Quadrature and Approximation of Distributions*, Found. Comp. Math, 391-429.

[FJ05] M. Frigo, S. G. Johnson (2005): *The Design and Implementation of FFTW3*, Proceedings of the IEEE 93 (2), 216-231.

[G08] M. B. Giles (2008): *Multilevel Monte Carlo Path Simulation*, Operations Research, 56(3), 2008.

[GW09] M. B. Giles, B. J. Waterhouse (2009): *Multilevel Quasi-Monte Carlo Path Simulation*, Radon Series Comp. Appl. Math 8, 1-18.

[G97] B. Gough (May 1997): *GSL FFT Algorithms*, GNU Scientific Library Documentation, `http://www.gnu.org/software/gsl/manual/`.

[GSL11] M. Galassi, J. Theiler et al. (2011): *GNU Scientific Library*, `http://www.gnu.org/software/gsl/`.

[H98] S. Heinrich (1998): *Monte Carlo Complexity o Global Solutions of Integral Equations*, Journal of Complexity (14), 151-175.

[H01] S. Heinrich (2001): *Multilevel Monte Carlo Methods*, Lecture Notes in Computer Science, Vol. 2179, Springer, 58-67.

[HS99] S. Heinrich, E. Sindambiwe (1999): *Monte Carlo Complexity of Parametric Integration*, Journal of Complexity (15), 317-341.

[HSW04] F. J. Hickernell, I. H. Sloan, G. W. Wasilkowski (2004): *The Strong Tractability of Multivariate Integration Using Lattice Rules*, In [Ni04, pp. 259 - 273].

[HMNR10] F. J. Hickernell, T. Müller-Gronbach, B. Niu, K. Ritter (2010): *Multi-level Monte Carlo Algorithms for Infinite-dimensional Integration on* $\mathbb{R}^{\mathbb{N}}$, Journal of Complexity (26), 229-254.

[K06] A. Klenke (2006): *Wahrscheinlichkeitstheorie*, Springer (Berlin).

[KP92] P. Kloeden, E. Platen (1992): *Numerical Solutions of Stochastic Differential Equations*, Springer.

[KS98] I. Karatzas, S. Shreve (1998): *Brownian Motion and Stochastic Calculus*, Graduate Texts in Mathematics, Springer, Second Edition.

[MM10] M. McCutchen (2010): *C++ Big Integer Library*, version 2010-04-30 17:37:03 -0400,
https://mattmccutchen.net/bigint/.

[Ni04] H. Niederreiter (ed.) (2004): *Monte Carlo and quasi-Monte Carlo methods 2002*, Springer, Berlin.

[NHMR10] B. Niu, F. J. Hickernell, T. Müller-Gronbach, K. Ritter (2010): *Deterministic Multi-level Algorithms for Infinite-dimensional Integration on* $\mathbb{R}^{\mathbb{N}}$, Journal of Complexity (27), 331-351

[MNR11] T. Müller-Gronbach, E. Novak, K. Ritter (2011): *Monte-Carlo Algorithmen*, Springer-Lehrbuch Masterclass, Springer.

[MN00] M. Matsumoto, T. Nishimura (2000): *Dynamic Creation of Pseudorandom Number Generators*, in Monte Carlo and Quasi-Monte Carlo Methods 1998, Springer, 56-69.

[NW10] E. Novak, H. Woźniakowski (2010); *Tractability of Multivariate Problems, Volume II: Standard Information for Functionals*, European Mathematical Society, Chapter 16.

[NC06a] D. Nuyens, R. Cools (2006): *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.

[NC06b] D. Nuyens, R. Cools (2006): *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, Springer-Verlag, 371-385.

[Ny07] D. Nuyens (2007): *Fast Construction of Good Lattice Rules*, PhD Thesis, ISBN 978-90-5682-804-2,
http://hdl.handle.net/1979/860.

[R00] K. Ritter (2000): *Average-Case Analysis of Numerical Problems*, Lecture Notes in Mathematics, Springer.

[SJ94] I.H. Sloan, S. Joe (1994): *Lattice Methods for Multiple Integration*, Oxford University Press.

[SR02] I. H. Sloan, A. V. Reztsov (2002): *Component-by-component Construction of Good Lattice Rules*, Math. Comp. 71(237), 263-273.

[V02] J. Vecer (2002): *Unified Pricing of Asian Options*, Risk, 113-116.