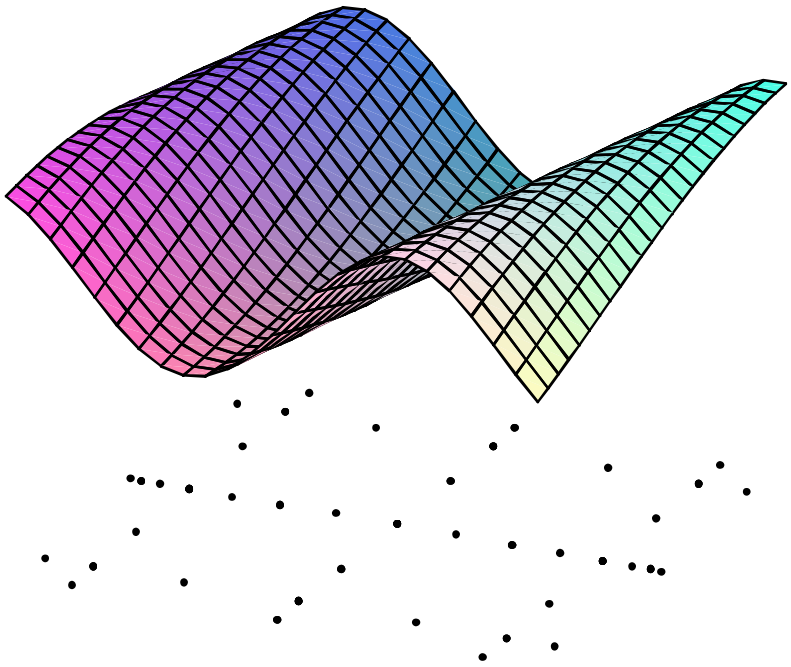
Error Estimation and Index Refinement for Dimension-Adaptive Sparse Grid Quadrature with Applications to the Computation of Path Integrals

Torsten Nahm



Diploma Thesis

University of Bonn

March 2005

Contents

1 Introduction

1.1 Multi-Dimensional Integrals of Real Functions

Integrals are a fundamental part of mathematics, with applications in a wide range of sciences. Among others, they feature prominently in physics, statistics, physical chemistry and financial mathematics. The most important case are integrals for functions with a domain that lies within $\mathbb{R}^d$ for some dimension d, and with values in $\mathbb{R}$. We shall cover only these functions in this thesis, but we note that the extension to $\mathbb{C}$-valued functions requires only slight modifications.

Path integrals are a particularly notable source of high-dimensional integration problems. They play an important role in financial mathematics and in statistical mechanics. Path integrals also arise as an alternative and complementary representation of certain partial differential equations. In particular, these path integrals allow for a simple representation of the Green function for the time evolution operator. We examine this relationship for the Schroedinger equation in quantum mechanics. Path integrals are infinite-dimensional by nature, and to approximate them numerically, we need to perform a temporal discretization. Since the error from discretization decreases with the number of time steps, we need high-dimensional integrals to obtain accurate results.

1.2 The Curse of Dimensionality

The following observation is of particular relevance to this thesis. If we want to integrate a function f in the interval $[0, 1]$, we might go ahead by taking its value at equidistant points, and calculating the mean of the function values on these points. This means evaluating f at N points. If we naively try to scale up this method to d dimensions, and similarly subdivide $[0, 1]^d$ equidistantly into d-dimensional hypercubes, we find we need to evaluate f at N^d points. That is, the amount of work needed to attain a given refinement level $\frac{1}{N}$ and a corresponding level of accuracy increases with the exponent d, and quickly grows beyond the capacity of today's computers for high dimensions d.

The "curse of dimensionality" is the colorful moniker used to describe this helplessness in the face of high dimensions. If we want to perform quadrature for high-dimensional functions, we need to find some way to avoid the curse of dimensionality.

The sampling methods Monte Carlo and Quasi Monte-Carlo accomplish this feat. The convergence of Monte Carlo quadrature is fully independent from the dimension of the problem, avoiding the curse of dimensionality. In many cases, Quasi Monte-

Carlo quadrature is a good alternative to Monte Carlo. Although unlike Monte Carlo it does suffer to some extent from the curse of dimensionality, it has a better convergence rate, which may outweigh this deficit in practice. However, the respective convergence rates of $\frac{1}{2}$ for Monte Carlo and at most 1 for Quasi-Monte Carlo make them ill-suited for problems that require high numerical accuracy.

Classical interpolation quadrature exploits the higher degrees of smoothness offered by many functions of interest and is able to attain much higher convergence rates. If it is scaled up to higher dimension using a tensor product approach, it does however suffer fully from the curse of dimensionality. The sparse grid method offers a way of retaining the advantages of interpolation quadrature while mitigating the effects of the curse of dimensionality. In recent years, adaptive sparse grid methods [17, 16] have been proposed in an effort to fully exploit the possibilities of the sparse grid method. Questions remain as to what strategies for adaptivity should be chosen. Also, the estimation of the error of quadrature is difficult for these methods, and requires further investigation.

1.3 Contributions

This thesis represents an application of the method of dimension-adaptive sparse grids [17] to the problem of quadrature. In particular, it extends the work on a corresponding algorithm proposed in [16]. The major contributions of the thesis are as follows.

- We suggest several different strategies for dimension-adaptive refinement based on a theoretical analysis of optimal convergence.
- We propose a hybrid strategy that combines the advantages of both conventional non-adaptive sparse grid quadrature [15] and of adaptive sparse grid quadrature, and obtain theoretical results for its convergence.
- We examine the problem of estimating the error of quadrature. We give theoretical results for the method proposed in [16], and suggest a new alternative method.
- The different strategies proposed in this thesis are implemented on the computer. The implementation is realized in an object-oriented, modular fashion that supports a mix-and-match approach for the various strategies. The implementation also includes several tools for visualizing the process of integration online. We analyze the run time complexity of the algorithm.
- Using the computer implementation and the Genz test suite, we compare the performance of the different refinement strategies, error estimators and quadrature rules with each other, and identify a set of parameters that performs well overall for the examined test cases.

- We compare the performance of the algorithm with established methods for multi-dimensional quadrature for the Genz test suite.
- We consider the problem of path integrals for two examples from quantum mechanics. We examine how the dimension-adaptive algorithm may be applied in this case, and compare its performance to that of the established methods.

Numerical mathematics is a field that some may describe as lacking mathematical stringency. This thesis makes an effort to present all results in a precise fashion, and to use modern mathematical structure to give concise proofs. We have also avoided the terms "trivial", "of course" and the big-O notation in proofs throughout this thesis, as we feel they may cover up problems [10].

1.4 Outline of the Thesis

The thesis starts off in chapter 2 with a review of several established approaches for multi-dimensional quadrature. The sampling methods Monte Carlo and Quasi-Monte Carlo are compared to the interpolation methods. We place particular emphasis on the performance of the different methods for high-dimensional problems. Chapter 3 reviews the sparse grid method for numerical quadrature, and examines its adaptive and non-adaptive versions. We also give some theoretical results for the convergence of the non-adaptive version, and show in particular that asymptotically, non-adaptive sparse grid methods fully break the curse of dimensionality. For actual computation, adaptive sparse grids may perform far better than non-adaptive methods. Their success depends on the quality of the adaptation strategy. In chapter 4, we consider the problem of adaptive refinement, and the related problem of error estimation. The computer implementation of the proposed algorithms, requisite data structures and questions of run time complexity are considered in chapter 5.

In chapter 6, we compare the different strategies proposed in earlier chapters with each other using the Genz test suite. Of the strategies and parameters compared, we identify the combination with the best overall characteristics. We then compare this algorithm with the established quadrature methods reviewed in chapter 2. In chapter 7, we examine the path integral representation of the harmonic oscillator and the anharmonic oscillator as examples of quantum mechanical problems. As in the previous chapter, we compare the performance of the adaptive sparse grid algorithm with the other quadrature methods. The thesis closes in chapter 8 with a discussion of the results obtained and of the questions and problems that remain.

1.5 Acknowledgements

I would like to thank Prof. Michael Griebel for sponsoring this thesis and Dr. Thomas Gerstner for his many helpful suggestions in developing the algorithm and in preparing this document. I would also like to thank Jörg Zimmermann for the many

insightful discussions on mathematics in general that sustained me throughout the work on this thesis.

2 Multi-Dimensional Quadrature Methods

This chapter gives an overview on the major established methods for the quadrature of multi-dimensional functions. Most of the results are standard material and may be found in any textbook on quadrature, although for reasons of consistency we have given in them in the form of trivial generalizations to arbitrary measures. We have paid special attention to show not only the capacities but also the limits of the different methods, in asymptotic theory as well as in pre-asymptotic practice. At the end of the chapter, we give a systematic comparison between the methods.

2.1 A Sketch of History

The modern theoretical foundation of integration theory dates back to Riemann in the 19th and Lebesgue in the early 20th century, and allows for the integration of large classes of functions. The integral values of functions of interest can often not be obtained analytically, i.e. by evaluating elementary functions. Instead, the integral values are approximated by numerical means (a process called numerical quadrature, or simply quadrature[1]).

The concept and first methods for numerical quadrature date back to the ancient Greeks. These approaches were geometrical in nature and are today described as the method of exhaustion (a prime example is Archimedes' approximation of π[1]). The method of exhaustion works by approximating a complex 2- or 3-dimensional shape by polygons and polyhedrons, respectively, which have a known area or volume. During the late 17th century, the development of calculus by Leibniz and Newton allowed for a wholly new approach to integrals. Their realization that differentiation theory also allowed for the solution of integrals by reversing the process of derivation opened the new field of analytical integration. It allowed for the symbolic integration of many simple functions by finding their "anti-derivative".

This discovery quickly fed into approximative integration theory, and by the end

[1] Several authors use the term "cubature" for multi-dimensional numerical integration, and reserves the term "quadrature" for univariate integrals. However, this hardly improves terminology, since the "cubature" as opposed to "quadrature" would at best suggest integrating over a 2-dimensional domain to obtain a 3-dimensional (cubic) volume. Furthermore, in the context of measure theory, integration is indeed a 2-dimensional process analogous to quadrature, since the value of the integral is obtained by multiplying the value of the function with the measure of the set (which may not even have a dimension) on which it has this value, thus (by abuse of abstraction) yielding a (2-dimensional) area. For these reasons, the term "quadrature" is used throughout the manuscript to describe the numerical approximation of integrals.

of the 17th century calculus had been used to generalize the Greek approach from geometric objects to general functions. For this, a given integrand is approximated by functions whose integrals are well known analytically (usually polynomials). This is known as quadrature by interpolation.

It was not until the 20th century that a wholly new class of quadrature methods, the sampling methods, was developed. Of these, Monte Carlo integration is the most prominent. Sampling methods are based on picking random or at least highly dispersive series of points, and taking the arithmetic mean of the function values obtained at these points as result for the quadrature. Their properties are quite different from interpolation methods, as we shall see later.

In recent decades, interest in numerical quadrature has surged, due mainly to the ubiquity of powerful computers that for the first time allow the computation of many quadrature values to high precision. Indeed, the problem of the quadrature of one-dimensional functions may be regarded as solved for the majority of functions of interest. For functions of higher dimensions, numerical quadrature still poses a huge challenge, and is an active area of current research. The reason that quadrature for higher dimensions is so difficult is outlined in the next section.

2.2 Definitions and notations

We will use the following notation:

Ω	denotes the domain of integration; often, we have $\Omega \subset \mathbb{R}^d$ for some d
μ	denotes the measure on Ω over which the function is to be integrated; often, μ will be the d-dimensional Lebesgue-measure λ^d
$\|\mu\|$	denotes the weight of the measure, i.e. $\|\mu\| := \mu(\Omega)$
f	denotes the integrand, with $f \in \mathcal{L}^1(\mu)$
$I(f)$	denotes the integral of f over the measure μ, that is $I(f) := \int_\Omega f d\mu$
$Q_N(f)$	denotes the result of numerical quadrature for the function f after evaluating f at N function points
$\epsilon(N)$	denotes the error of quadrature after evaluating N function points; it is defined as $\epsilon(N) = \|Q_N(f) - I(f)\|$
$\epsilon_{max}(N)$	denotes an upper bound for the error $\epsilon(N)$, as obtained from theoretical results
$\rho(N)$	denotes the convergence rate obtained from theory, defined as $\rho(N) := -\frac{N\frac{d}{dN}\epsilon_{max}(N)}{\epsilon_{max}(N)}$. This corresponds to the slope of $\epsilon_{max}(N)$ in a bilogarithmic plot. The minus sign in the definition make $\rho(N)$ satisfy the intuition that higher convergence rates are better. If $\lim_{N\to\infty} \rho(N)$ exists, we call it the asymptotic (logarithmic) convergence rate.

We will now formalize what we mean by a **quadrature method** (quadrature algorithm) for a given domain Ω and a measure μ. We will only consider the case of finite measures, i.e. we require $\|\mu\| < \infty$. We also require $\|\mu\| > 0$. This is needed for some technical considerations, and the case $\|\mu\| = 0$ is uninteresting anyhow.

Definition 2.2.1 *We say that Q_N is a **deterministic quadrature method** iff there exist functions*

$$p_{N,k} : \mathbb{R}^{k-1} \to \Omega \text{ for } k, N \in \mathbb{N},\ k \leq N$$

and

$$r_N : \mathbb{R}^N \to \mathbb{R} \text{ for } N \in \mathbb{N}$$

so that

$$Q_N(f) = r_N\left(f(x_{N,1}), \ldots, f(x_{N,N})\right)$$

with

$$x_{N,k} = p_{N,k}\left(f(x_{N,1}), \ldots, f(x_{N,k-1})\right)$$

$p_{N,k}$ is said to be the point generator, and r_N the result generator.

This definition formalizes the idea that for any given N only information gleaned from the function so far may be used to calculate the result and to decide which point to evaluate next (compare [34]). An example for this definition will be given in the next section.

Definition 2.2.2 *We say that Q_N is a **probabilistic quadrature method** iff there exist functions $p_{N,k}$ and r_N as above, but these functions and Q_N itself are understood to have an additional argument from a probability space $(S, \mathbb{S}, P)$. That is, we have $Q_N : S \times \mathcal{L}^1(\Omega) \to \mathbb{R}$, $p_{N,k} : S \times \mathbb{R}^{k-1} \to \Omega$, $r_N : S \times \mathbb{R}^N \to \mathbb{R}$. As is convention in probability theory, this argument is usually not explicitly listed.*

The two properties given in the next definition will often be of use.

Definition 2.2.3 *Q_N is called **positive semi-definite** iff we have $Q_N(f) \geq 0$ for all $f \geq 0$. Q_N is called **linear** iff Q_N is linear for all N (and for a probabilistic quadrature method for all probabilistic arguments $s \in S$), i.e. $Q_N(\lambda f + \mu g) = \lambda Q_N(f) + \mu Q_N(g)$ for all integrands f and g and all $\lambda, \mu \in \mathbb{R}$.*

Finally, we give several definitions relating quadrature to the integral.

Definition 2.2.4 *A quadrature method Q_N is called **consistent** iff for all constant functions $c \cdot 1_\Omega$, $c \in \mathbb{R}$ the equality*

$$Q_N(c \cdot 1_\Omega) = I(c \cdot 1_\Omega) = c\,\|\mu\|$$

holds. Iff for a set $\mathfrak{G} \subset \mathcal{L}^1(\mu)$ and a specific N we have

$$Q_N(g) = I(g)$$

for all $g \in \mathfrak{G}$, we say that Q_N is ***exact*** *for $\mathfrak{G}$. Finally, iff for a set $\mathfrak{G} \subset \mathcal{L}^1(\mu)$ we have*

$$\lim_{N\to\infty} Q_N(g) = I(g)$$

for all $g \in \mathfrak{G}$ we say that Q_N is ***limes-exact*** *for $\mathfrak{G}$.*

We see that the consistency of Q_N is simply a special case of exactness for $\mathfrak{G} = \{c \cdot 1_\Omega : c \in \mathbb{R}\}$. We will now examine the properties of the most common methods for multi-dimensional quadrature.

2.3 Sampling Methods

2.3.1 Monte Carlo quadrature

Not all quadrature algorithms suffer from the curse of dimensionality encountered by the primitive approach given in section 1.2. Monte Carlo quadrature especially is totally oblivious to dimension, and in fact any continuous structure at all. Monte-Carlo quadrature is a probabilistic quadrature method defined by

$$Q_N^{\mathbf{MC}}(f) := \frac{1}{N} \|\mu\| \sum_{i=1}^{N} f(X_i) \tag{2.3.1}$$

where the X_i are independent random variables with the distribution $\frac{\mu}{\|\mu\|}$.

Formalized according to definition 2.2.2, $Q_N^{\mathbf{MC}}$ is given by the point generator $p_{N,n} : (s, (y_i)_{i=1,\dots,n-1}) \mapsto s_n$ and the result generator $r_n : (s, (y_i)_{i=1,\dots,n}) \mapsto \frac{\|\mu\|}{N} \sum_{i=1}^{n} y_i$. The probability space $(S, \mathbb{S}, P)$ is given by $\left(\Omega^{\mathbb{N}}, \mathcal{A}^{\otimes\mathbb{N}}, \left(\frac{\mu}{\|\mu\|}\right)^{\otimes\mathbb{N}}\right)$, where $\mathcal{A}$ denotes the σ-algebra on which μ is defined. As can be seen from definition, Monte Carlo quadrature is consistent, positive semi-definite and linear.

Since the random variables $Y_i = f(X_i)$ are independent and have identical distribution, by Etemadi's theorem[11] we have $\lim_{N\to\infty} Q_N^{\mathbf{MC}}(f) = E(\|\mu\| \cdot f) = I(f)$ almost surely in P, that is, for any function $f \in \mathcal{L}^1(\mu)$ we know that $Q_N^{\mathbf{MC}}$ is limes-exact with probability 1. However, Etemadi's theorem does not give us any information about the rate of convergence.

To make a statement about the rate of convergence we need the additional condition that $f \in \mathcal{L}^2(\mu)$. An elementary result from probability theory is the following:

Proposition 2.3.1 *Let $f \in \mathcal{L}^2(\mu)$. For all $\eta > 0$, we have $P\left\{\epsilon(N) \geq \frac{1}{\sqrt{N}} c(\eta)\right\} \leq \eta$ with $c(\eta) = \frac{1}{\eta}\sqrt{\|\mu\|}\,\|f\|_{\mathcal{L}^2}$.*

Proof. Let $\tilde{f} = \|\mu\| \cdot f$. Take $\eta > 0$. Using Chebyshev's inequality for the exponent 2, we have

$$P\left\{\left|Q_N^{\mathbf{MC}}(f) - I(f)\right| \geq \frac{1}{\sqrt{N}} c(\eta)\right\} \quad \leq \frac{N}{c(\eta)^2} \quad E\left(\left(Q_N^{\mathbf{MC}}(f) - I(f)\right)^2\right)$$

By definition of $\tilde{f}$ and $Q_N^{\mathbf{MC}}$, it follows that

$$E\left(\left(Q_N^{\mathbf{MC}}(f) - I(f)\right)^2\right) = E\left(\left(\frac{1}{N}\sum_{i=1}^{N}\tilde{f}\left(X_i\right) - E\left(\tilde{f}\right)\right)^2\right) \tag{2.3.2}$$

Because $E\left(\frac{1}{N}\sum_{i=1}^{N}\tilde{f}\left(X_i\right) - E\left(\tilde{f}\right)\right) = 0$, we obtain

$$\begin{aligned} E\left(\left(\frac{1}{N}\sum_{i=1}^{N}\tilde{f}\left(X_i\right) - E\left(\tilde{f}\right)\right)^2\right) &= V\left(\frac{1}{N}\sum_{i=1}^{N}\tilde{f}\left(X_i\right) - E\left(\tilde{f}\right)\right) \\ &= \sum_{i=1}^{N} V\left(\frac{1}{N}\tilde{f}\left(X_i\right)\right) \end{aligned}$$

using the equality of Bienaymé for the last step. Finally,

$$\begin{aligned} \sum_{i=1}^{N} V\left(\frac{1}{N}\tilde{f}\left(X_i\right)\right) &= N \cdot V\left(\frac{1}{N}\tilde{f}\left(X_1\right)\right) \\ &\leq \frac{1}{N} E\left(\tilde{f}\left(X_1\right)^2\right) \\ &= \frac{1}{N}\int_{\Omega}\left(\|\mu\| \cdot f\right)^2 \frac{1}{\|\mu\|} d\mu \\ &= \frac{1}{N}\|\mu\|\,\|f\|_{\mathcal{L}^2}^2 \end{aligned}$$

In conclusion, we have $E\left(\left(Q_N^{\mathbf{MC}}(f) - I(f)\right)^2\right) \leq \frac{1}{N}\|\mu\|\,\|f\|_{\mathcal{L}^2}^2$. This combined with equation 2.3.2 yields the desired result. □

The proposition states that for any number η the error $\epsilon(N)$ is smaller than $\frac{1}{\sqrt{N}}c(\eta)$ with probability $1-\eta$, giving us a convergence rate of $\frac{1}{2}$ in a statistical sense. For example taking $\eta = \frac{1}{100}$, we know that $\epsilon(N) \leq \frac{1}{\sqrt{N}} \cdot 100\sqrt{\|\mu\|}\,\|f\|_{\mathcal{L}^2}$ with 99% probability. We see further see from the proposition that $c\left(\eta\right)$ and therefore the constant of convergence only depends on the $\mathcal{L}^2$ norm of f. This means that Monte Carlo quadrature performs equally well whether the function is smooth, only continuous, or in fact only measurable.

Due to the probabilistic nature of Monte Carlo quadrature, it is impossible to give a deterministic guarantee of convergence even for very good-natured functions.

Proposition 2.3.2 *Let $\Omega = [0,1]$, $\mu = \lambda^1\left([0,1]\right)$. Then there exists $f \in C^\infty(\Omega)$ so that for all N, $P\left\{\left|Q_N^{\mathbf{MC}}(f) - I(f)\right| \geq 1\right\} > 0$.*

Proof. Take $f\left(x\right) = 10x$. We have $I(f) = 5$. Let $A_N = \left\{X_i \leq \frac{1}{5}; i = 1, \ldots, N\right\}$. We have

$$Q_N^{\mathbf{MC}}(f) = \frac{1}{N}\sum_{i=1}^{N} f\left(X_i\right) \leq 10 \cdot \frac{1}{5} = 2$$

on A_N, so that

$$\left\{\left|Q_N^{\mathbf{MC}}(f) - I(f)\right| \geq 1\right\} \supset A_N$$

Since $P(A_N) = \frac{1}{4^N} > 0$, this proves the proposition. □

On the other hand, also due to its probabilistic structure, Monte Carlo quadrature has the unique property mentioned above that we have almost sure convergence for $N \to \infty$. All deterministic quadrature methods need stronger requirements to guarantee convergence. This is formulated in the following proposition.

Proposition 2.3.3 *If Q_N is a deterministic quadrature method for $\Omega = [0,1]$ and $\mu = \lambda^1([0,1])$. Then there exists a function $f \in \mathcal{L}^1(\mu)$ so that $Q_N(f)$ does not converge to $\int_\Omega f d\mu$.*

Proof. Take $g = 0$ constant. Let M_N be the set of points used for the evaluation of $Q_N(g)$. Let $M = \bigcup_{i=1}^{\infty} M_N$. Since the M_N are finite it follows immediately that M is countable. Now let $h = 1_{\Omega \backslash M}$ be the indicator function for $\Omega \backslash M$. Since M is countable, it is also Lebesgue-measurable and therefore h is a measurable function on Ω. Since Q_N is deterministic, and therefore depends only on the results of function evaluations, we have $Q_N(g) = Q_N(h)$ for all N, since g and h are equal on all possible points of evaluation M. Since $I(g) = 0$ and $I(h) = 1$, at least one of the statements $\lim_{N\to\infty} Q_N(g) = I(g)$ and $\lim_{N\to\infty} Q(h) = I(h)$ must be false. □

2.3.2 Quasi-Monte Carlo quadrature

Quasi-Monte Carlo quadrature presents a better alternative to Monte Carlo quadrature for many functions. Instead of choosing random points, the points are generated deterministically. We have

$$Q_N^{\mathbf{QMC}}(f) := \frac{\|\mu\|}{N} \sum_{i=1}^{N} f(x_i) \tag{2.3.3}$$

where the x_i are a deterministic sequence in Ω. Quasi-Monte Carlo quadrature is, just as Monte Carlo quadrature, consistent, positive semi-definite and linear.

For $\Omega = [0,1]^d$ and $\mu = \lambda^d(\Omega)$, the convergence rate for the Quasi-Monte Carlo method is given by the inequality of Koksma-Hlawka[18]:

$$\epsilon(N) \leq V(f) D_N^*(x_1, \ldots, x_N)$$

where $V(f)$is the bounded variation of f in the sense of Hardy and Krause[28], and $D_N^*(x_1, \ldots, x_N)$ is the star discrepancy of the family $(x_1, \ldots, x_N)$ (also [28]).

Commonly, so-called **low-discrepancy** sequences are used for Quasi-Monte Carlo quadrature. These sequences are defined by having a star discrepancy

$$D_N^*(x_1, \ldots, x_N) \leq C(d) \frac{(\log N)^d}{N}$$

for some constants $C(d)$. The Halton sequence and the Sobol sequence are examples of such sequences. For their construction, see [30].

For such a low discrepancy sequence and an integrand with bounded variation in the sense of Hardy and Krause, we therefore obtain an error bound of $\epsilon(N) \leq V(f)C(d)\frac{(\log N)^d}{N}$. Calculating the convergence rate according to the definition in section 2.2, we obtain $\rho^{\mathbf{QMC}}(N) = 1 - \frac{d}{\log N}$. The asymptotic convergence rate is $\lim_{N\to\infty} \rho^{\mathbf{QMC}}(N) = 1$.

2.4 Interpolation Methods

2.4.1 General interpolation

We now turn to a wholly different class of quadrature methods, the interpolation methods. Although (as noted in section 2.1) historically older, these methods are more involved and harder to implement than the Quasi-Monte Carlo and Monte Carlo sampling methods. The general idea is to find a quadrature method Q_N that is limes-exact for a certain class of functions $\mathfrak{G} \subset \mathcal{L}^1(\Omega)$ (often the polynomials), which is in some way dense within the set of functions of interest. To formalize this idea, the definition of a distance between a function and such a class is helpful.

Definition 2.4.1 *Let $\mathfrak{G} \subset \mathcal{L}^1(\Omega)$ be a set of functions. We define the **distance** of f to $\mathfrak{G}$ by* $\operatorname{dist}(f, \mathfrak{G}) := \inf_{g\in\mathfrak{G}} \|f - g\|_\infty$.

Note that the distance may be infinite. We now give a small lemma.

Lemma 2.4.2 *Let Q_N be linear and positive semi-definite. Then Q_N is monotonous, i.e. if $f \leq g$ then $Q_N(f) \leq Q_N(g)$. If additionally Q_N is consistent, then in particular $|Q_N(f)| \leq \|\mu\| \cdot \|f\|_\infty$.*

Proof. We have $Q_N(g-f) \geq 0$ because Q_N is positive semi-definite. By linearity $Q_N(g) \geq Q_N(f)$ follows immediately. If Q_N is consistent, we have $Q_N(1_\Omega) = \|\mu\|$. Because $-\|f\|_\infty \cdot 1_\Omega \leq f \leq \|f\|_\infty \cdot 1_\Omega$, it follows that $-\|\mu\| \cdot \|f\|_\infty \leq Q_N(f) \leq \|\mu\| \cdot \|f\|_\infty$. □

In particular, this leads to the following estimate.

Lemma 2.4.3 *For $h \in \mathcal{L}^1(\Omega)$, we have*

$$|Q_N(h) - I(h)| \leq 2\|\mu\| \cdot \|h\|_\infty$$

Proof. We have

$$|Q_N(h) - I(h)| \leq |Q_N(h)| + |I(h)|$$

By lemma 2.4.2

$$|Q_N(h)| \leq \|\mu\| \cdot \|h\|_\infty$$

and

$$|I(h)| \leq \|\mu\| \cdot \|h\|_\infty$$

by similar properties of the integral. □

These results allow us to immediately relate the distance of f to $\mathfrak{G}$ to the error of quadrature (see [4]):

Proposition 2.4.4 *Let Q_N be linear, consistent and positive semi-definite. Let Q_N be exact on $\mathfrak{G}$. Then*

$$\epsilon(N) \leq 2\,\|\mu\| \cdot \operatorname{dist}(f, \mathfrak{G})$$

Proof. For all $g \in \mathfrak{G}$, we have $Q_N(g) - I(g) = 0$ by definition. Because Q_N is linear, this implies

$$\begin{aligned} \epsilon(N) &= |Q_N(f) - I(f)| \\ &= |Q_N(f) - I(f) + Q_N(g) - I(g)| \\ &= |Q_N(f-g) - I(f-g)| \\ &\leq 2\,\|\mu\| \cdot \|f-g\|_\infty \end{aligned}$$

by the above lemma. Since this is true for any $g \in \mathfrak{G}$, we have

$$\epsilon(N) \leq 2\,\|\mu\| \cdot \inf_{g \in \mathfrak{G}} \|f-g\|_\infty$$

□

We can extend this result to quadrature methods that are only limes-exact on $\mathfrak{G}$.

Proposition 2.4.5 *Let Q_N be linear, consistent and positive semi-definite. Let Q_N be limes-exact on $\mathfrak{G}$. Then*

$$\limsup_{N\to\infty} \epsilon(N) \leq 2\,\|\mu\| \cdot \operatorname{dist}(f, \mathfrak{G})$$

Proof. Let $\epsilon > 0$. Let $g \in \mathfrak{G}$. Then

$$\begin{aligned} |Q_N(f) - I(f)| &= |Q_N(f) - Q_N(g) - I(f) + I(g) + Q_N(g) - I(g)| \\ &\leq |Q_N(f-g) - I(f-g)| + |Q_N(g) - I(g)| \\ &\leq 2\,\|\mu\| \cdot \|f-g\|_\infty + |Q_N(g) - I(g)| \end{aligned}$$

Since

$$\lim_{N\to\infty} |Q_N(g) - I(g)| = 0$$

this means

$$\limsup_{N\to\infty} |Q_N(f) - I(f)| \leq 2\,\|\mu\| \cdot \|f-g\|_\infty$$

Since $g \in \mathfrak{G}$ was arbitrary, this proves the proposition. □

Our main result on quadrature by interpolation follows as corollary.

Corollary 2.4.6 *Let Q_N be linear, consistent and positive semi-definite. Let Q_N be limes-exact for $\mathfrak{G}$. Then Q_N is limes-exact on $\overline{\mathfrak{G}}$, where $\overline{\mathfrak{G}}$ denotes the closure of $\mathfrak{G}$ with respect to the topology of uniform convergence.*

PROOF. We simply note that for $f \in \overline{\mathfrak{G}}$ we have $\mathrm{dist}(f, \mathfrak{G}) = 0$ by definition. □

2.4.2 One-dimensional quadrature on $\mathbb{R}$

In this section, we examine the important case $\Omega \subset \mathbb{R}$. In this case, because they are especially amenable to analytical integration, $\mathfrak{G}$is usually taken to be the set of polynomials. We denote the polynomials[2] of order m or less by $\mathcal{P}_m$, that is,

$$\mathcal{P}_m := \mathrm{span}_{\mathbb{R}} \left\{1, x, x^2, \ldots, x^m\right\}$$

and the set of all polynomials simply by $\mathcal{P}$. For the following considerations, we require that $\mathcal{P} \subset \mathcal{L}^1(\mu)$. This is the case for instance if Ω is bounded. From this follows that $\mathcal{P} \subset \mathcal{L}^2(\mu)$, since the squares of polynomials are themselves polynomials. This allows us to define the standard $\mathcal{L}^2(\mu)$ bilinear form $\langle f, g\rangle := \int_\Omega f \cdot g d\mu$.

We now proceed to find quadrature formulas that are limes-exact on the polynomials. We will examine formulas of the type

$$Q_N(f) = \sum_{i=1}^{N} w_{N,i} f(x_{N,i}) \tag{2.4.1}$$

with $w_{N,i} \in \mathbb{R}$ and $x_{N,i} \in \Omega$. Obviously, Q_N is linear. We note that Q_N is consistent iff

$$\sum_{i=1}^{N} w_{N,i} \quad = \quad \|\mu\| \tag{2.4.2}$$

and that the condition

$$w_{N,i} \geq 0 \tag{2.4.3}$$

for all N and $i = 1, \ldots, N$ is sufficient (but not always necessary) for Q_N to be positive semi-definite.

It is an elementary result that for any given set of abscissas $x_{N,1}, \ldots, x_{N,N}$, weights $w_{N,1}, \ldots, w_{N,N}$ (called Newton-Cotes weights) can be chosen so that Q_N is exact on $\mathcal{P}_{N-1}$. These may be obtained for example by integrating the Lagrange basis polynomials [9].

Remark 2.4.7 *Any quadrature formula Q_N that is exact on $\mathcal{P}_0$ is automatically consistent because $\mathcal{P}_0 = \{c \cdot 1_\Omega : c \in \mathbb{R}\}$ (compare definition 2.2.4). It is not, however,*

[2] By abuse of notation, we do not distinguish the polynomials per se as elements of $\mathbb{R}^{\otimes\mathbb{N}}$ from their incarnation as elements of $\mathrm{map}\,(\Omega, \mathbb{R})$.

necessarily positive semi-definite, so if we need this property, we have to establish it for the given measure and nodes.

For the Newton-Cotes weights, the abscissas $x_{N,i}$ could be chosen arbitrarily (as long as no two are the same). This gives us a quadrature formula that is exact on $\mathcal{P}_{N-1}$. For so-called Gauss formulas, special points (the nodes of orthogonal polynomials) are chosen as the abscissas, leading to formulas that are exact $\mathcal{P}_{2N-1}$ [9]. Gauss formulas do not always exist. However, it can be shown that if the bilinear $\langle ., . \rangle$ form is positive definite on $\mathcal{P}$ and if Ω is an interval, then for each N we have a positive semi-definite Gauss formula Q_N.

Remark 2.4.8 *Gauss formulas exist and are well-known for many important cases:*

$\Omega = [-1,1]\,,\ \mu = \lambda^1\left([-1,1]\right)$	*Gauss-Legendre rules*
$\Omega = [-1,1]\,,\ \mu = \frac{1}{\sqrt{1-x^2}}\lambda^1\left([-1,1]\right)$	*Gauss-Chebyshev rules*
$\Omega = [0,\infty[,\ \mu = e^{-x}\lambda^1\left([0,\infty[\right)$	*Gauss-Laguerre rules*
$\Omega =]-\infty,\infty[,\ \mu = \frac{1}{\sqrt{\pi}}e^{-x^2}\lambda^1\left(]-\infty,\infty[\right)$	*Gauss-Hermite rules*

All these formulas are linear, consistent and positive semi-definite.

If we have a quadrature method Q_N that is exact for increasing polynomial degrees with N, this in particular implies that Q_N is limes-exact for $\mathcal{P}$. This gives us an important result on convergence.

Proposition 2.4.9 *Let Q_N be linear, consistent and positive semi-definite. If Q_N is limes-exact for $\mathcal{P}$ and Ω is compact, then Q_N is limes-exact for $C^0(\Omega)$, where $C^0(\Omega)$ denotes the continuous functions on Ω.*

Proof. This follows immediately form the theorem of Stone-Weierstrass[13] and from proposition 2.4.6. □

This result answers the question of convergence only for compact Ω. However, many functions of interest are not defined on a compact domain, for example because they have a singularity. Empirically, we observe that the method of quadrature by polynomial interpolation given in this chapter still works quite well in many cases. Unfortunately, the theory of convergence for quadrature on non-compact domains is far less developed than that for compact domains and closed intervals in particular, so that we have only a few results for small and poorly characterized functions spaces. For Gauss-Hermite quadrature, for example, we obtain convergence if all derivatives have a common bound (compare [9]). Given the good convergence

attained on non-compact domains in practice, this gulf between practice and theory is quite unsatisfactory.

2.4.3 Convergence results for $\Omega = [0,1]$, $\mu = \lambda^1([0,1])$ and $C^\infty([0,1])$

We now give results for a case for which detailed analysis does exist, and which nonetheless encompasses many important problems. Specifically, we take $\Omega = [0,1]$ and $\mu = \lambda^1([0,1])$, and an integrand $f \in C^\infty([0,1])$. For this section, we assume that Q_N is exact on $\mathcal{P}_{N-1}$ (quadrature methods of this kind are sometimes called interpolatory).

For the base point $x_0 = \frac{1}{2}$, the Taylor interpolation polynomial is

$$p(x) = \sum_{k=0}^{M} \frac{\partial^k f\left(\frac{1}{2}\right)}{k!} \left(x - \frac{1}{2}\right)^k$$

This means that we have

$$f(x) = p(x) + \frac{\partial^{M+1} f(\tau)}{(M+1)!} \left(x - \frac{1}{2}\right)^{M+1}$$

for some $\tau \in [0,1]$. Because $p \in \mathcal{P}_M$, this implies

$$\operatorname{dist}(f, \mathcal{P}_M) \leq \|f - p\|_\infty \leq \frac{1}{2^{M+1}(M+1)!} \left\|\partial^{M+1} f\right\|_\infty \tag{2.4.4}$$

Since Q_N is exact on $\mathcal{P}_{N-1}$, we can now use proposition 2.4.4 to obtain the estimate

$$\epsilon(N) \leq \frac{2}{2^N N!} \left\|\partial^N f\right\|_\infty$$

Unfortunately, this generally allows no prediction about how fast $\epsilon(N)$ converges to 0 with increasing N, because $\left\|\partial^N f\right\|_\infty$ can increase rapidly with N. Indeed, if f is not analytical in $[0,1]$, we know the remainder term of the Taylor series does not converge to 0. To give estimates of convergence for general functions $f \in C^\infty([0,1])$, we need a different approach. Here the theory of approximation with algebraic polynomials comes into play [4]. Indeed, from approximation theory we have the estimate

$$\operatorname{dist}(f, \mathcal{P}_M) \leq \frac{\pi}{2^{r+1}(M-r+2)\cdots(M+1)} \|\partial^r f\|_\infty$$

for a fixed $r \geq 1$ and $M \geq r-1$ for any $f \in C^r([0,1])$ [32]. Since

$$\lim_{M\to\infty} \frac{(M+1)^r}{(M-r+2)\cdots(M+1)} = 1$$

this means we have constants $c_r < \infty$ with

$$\operatorname{dist}(f, \mathcal{P}_M) \leq \frac{c_r}{2} (M+1)^{-r} \|\partial^r f\|_\infty$$

Now we can again use proposition 2.4.4 to obtain

$$\epsilon(N) \leq 2 \|\mu\| \cdot \operatorname{dist}(f, \mathcal{P}_{N-1}) \leq c_r N^{-r} \left\|\partial^r f\right\|_\infty$$

for $M \geq r - 1$.

This gives us a convergence rate of $\rho(N) = r$ for $N \geq r$. For our function $f \in C^\infty([0,1])$, this means that we actually have an asymptotic convergence rate of infinity. While this sounds rather tantalizing, we must always remember that the calculations our computers are able to perform stop far short of infinity (we will return to this point in section 2.7). For finite N, we have the estimate

$$\epsilon(N) \leq \min_{r=1,\ldots,N} c_r M^{-r} \left\|\partial^r f\right\|_\infty$$

Each one of the terms $c_r M^{-r} \left\|\partial^r f\right\|_\infty$ will eventually overtake all previous terms $c_s M^{-s} \left\|\partial^s f\right\|_\infty$, $s < r$ and become the dominant term for the minimum. When this transition happens depends on the size of $c_r \left\|\partial^r f\right\|_\infty$. In practice therefore we do expect the observed convergence rate to actually speed up with increasing N, but how slowly or quickly this speed-up occurs is strongly dependent on the function f and its derivatives.

Remark 2.4.10 *The constants given in the estimates for* $\operatorname{dist}(f, \mathcal{P}_M)$ *above can be drastically improved with additional results (for example [3]). However, the qualitative nature of the results remains the same.*

Remark 2.4.11 *We have examined only functions of the class* $C^\infty([0,1])$. *As we can see from the results above, we in fact have* $\epsilon(N) \leq c_r M^{-r} \left\|\partial^r f\right\|_\infty$ *for any* $f \in C^r([0,1])$.

2.4.4 The tensor product

We now need to find a way to translate our one-dimensional interpolation quadrature formulas to the multi-dimensional case. The natural approach is to use some sort of tensor product of our one-dimensional operators. The history of the tensor product is complex, and it has been notoriously difficult to formalize. The formalization found in the context of algebra and category theory[20, 24] is mathematically exact, but is usually insufficient for analytical questions. This is due to the fact that for the algebraic tensor product, all tensors are *finite* sums of decomposable elements $a \otimes b$. However, to give an example, in general a continuous function $f \in C\left(\Omega^2\right)$ cannot be written as a sum $f(x,y) = \sum_{i=1}^n g_i(x) h_i(y)$ with $g_i, h_i \in C(\Omega)$. To overcome this limitation, some sort of completion process has to be performed, with careful thought to the proper norm for completion. For operators (like the Q_N), the situation becomes even more difficult, since we need to find conditions to make sure the operators can be properly extended onto the completed space. An overview of the required mathematics is given for example in [23].

Since none of these difficult concepts are actually needed for this thesis, we will sidestep the problem by defining the tensor product only for a very limited space of operators, which, however, is sufficient for our further considerations. Indeed, when we use the tensor product during this thesis, we always mean the tensor product of vector spaces from algebra.

Definition 2.4.12 *We define the* ***Dirac form*** δ_x *for* $x \in \Omega$ *by* $\delta_x(f) = f(x)$. *We define the space of* ***Dirac sums*** D_Ω *by* $D_\Omega := \operatorname{span}_{\mathbb{R}} \{\delta_x : x \in \Omega\}$.

This class D corresponds exactly to the class of quadrature formulas used in this section, as given in equation 2.4.1. We now take the d-fold algebraic tensor product of D_Ω with itself:

$$D_\Omega^{\otimes d} := \underbrace{D_\Omega \otimes \ldots \otimes D_\Omega}_{d \text{ times}}$$

We define a multi-linear mapping

$$E : (D_\Omega)^d \to D_{\Omega^d}$$

by defining it on the basis elements of D_Ω by

$$E(\delta_{x_1}, \ldots, \delta_{x_d}) = \delta_{(x_1,\ldots,x_d)}$$

By the universal property of the tensor product, this gives us a linear isomorphism of $D_\Omega^{\otimes d}$ into D_{Ω^d}, so that we can naturally take D_{Ω^d} as the tensor product $D_\Omega^{\otimes d}$.

2.4.5 Multi-Dimensional quadrature

We can now translate the results from section 2.4.2 to the multi-dimensional case by taking the Dirac formula $Q_N^{\otimes d}$, yielding a quadrature formula for $\mu^{\otimes d}$ on Ω^d. Written explicitly by expanding the terms and using the multi-linearity of $Q_N^{\otimes d}$, we have

$$Q_N^{\otimes d}(f) = \sum_{i_1=1}^{N} \ldots \sum_{i_d=1}^{N} w_{N,i_1} \cdots w_{N,i_d} f(x_{N,i_1}, \ldots, x_{N,i_d})$$

$Q_N^{\otimes d}$ is linear, and it inherits the characteristics of Q_N: It is consistent if Q_N is consistent, and positive semi-definite if Q_N is positive semi-definite.

Proposition 2.4.13 *If* Q_N *is consistent, then* $Q_N^{\otimes d}$ *is consistent.*

Proof. $Q_N^{\otimes d}(c \cdot 1_{\Omega^d}) = \sum_{i_1=1}^{N} \ldots \sum_{i_d=1}^{N} w_{N,i_1} \cdots w_{N,i_d} c = c \left(\sum_{i=1}^{N} w_{N,i} \right)^d = c \cdot \|\mu\|^d$. □

Proposition 2.4.14 *If* Q_N *is positive semi-definite, then* $Q_N^{\otimes d}$ *is positive semi-definite.*

PROOF. We will first examine the case of $d = 2$. Let $f \geq 0$ and $g(x) = Q_N(f(x,.))$. Because $f(x,.) \geq 0$ for all x and because Q_N is positive semi-definite, we obtain $g(x) \geq 0$. Because $Q_N \otimes Q_N(f) = Q_N(g)$, we obtain $Q_N \otimes Q_N(f) \geq 0$, which proves the proposition for $d = 2$. We can extend this proof to higher dimensions by iteration. □

In analogy to the last section, let $\mathcal{P}_M^d$ denote the d-dimensional polynomials of at most maximum degree M, i.e.

$$\mathcal{P}_M^d := \text{span}_{\mathbb{R}} \left\{1, x_1, \ldots x_d, \ldots, x_1^M, \ldots, x_d^M\right\}$$

and let $\mathcal{P}^d = \bigcup_M \mathcal{P}_M^d$ denote all d-dimensional polynomials.

We can now transfer the results about polynomial exactness from the last section to the multi-dimensional case. First, we prove a more general proposition.

Proposition 2.4.15 *If Q_N is exact on $\mathcal{G}$, then $Q_N^{\otimes d}$ is exact on $\mathcal{G}^{\otimes d}$.*

PROOF. Let $h(x_1, \ldots, x_d) = g_1(x_1) \cdots g_d(x_d)$, $g_i \in \mathcal{G}$ be a decomposable element of $\mathcal{G}$. We have

$$\begin{aligned}
Q_N^{\otimes d}(h) &= \sum_{i_1=1}^{N} \ldots \sum_{i_d=1}^{N} w_{N,i_1} \cdots w_{N,i_d} h(x_{N,i_1}, \ldots, x_{N,i_d}) \\
&= \sum_{i_1=1}^{N} \ldots \sum_{i_d=1}^{N} w_{N,i_1} \cdots w_{N,i_d} g_1(x_{N,i_1}) \cdots g_d(x_{N,i_d}) \\
&= \sum_{i_1=1}^{N} w_{N,i_1} g_1(x_{N,i_1}) \ldots \sum_{i_d=1}^{N} w_{N,i_d} g_d(x_{N,i_d}) \\
&= Q_N(g_1) \cdots Q_N(g_d) = \int_\Omega g_1 d\mu \cdots \int_\Omega g_d d\mu \\
&= \int_{\Omega^d} g_1(x_1) \cdots g_d(x_d)\, d\mu^{\otimes d}(x_1, \ldots, x_d) \\
&= \int_{\Omega^d} h d\mu^{\otimes d}
\end{aligned}$$

Since any element of $\mathcal{G}^{\otimes d}$ can be written as finite sum of decomposable elements, the proposition follows from the linearity of $Q_N^{\otimes d}$ and $\int_\Omega d\mu$. □

Corollary 2.4.16 *If Q_N is exact on $\mathcal{P}_M$, then $Q_N^{\otimes d}$ is exact on $\mathcal{P}_M^d$.*

PROOF. This follows immediately by noting that $\mathcal{P}_M^d = (\mathcal{P}_M)^{\otimes d}$. □

In analogy to the last section, we therefore know that if Q_N is exact for increasing polynomial degrees with larger N, then $Q_N^{\otimes d}$ is limes-exact on $\mathcal{P}^d$. Therefore, in analogy to proposition 2.4.9, we know that $Q_N^{\otimes d}$ is limes-exact on $C\left(\Omega^d\right)$ if Ω (and therefore Ω^d) is compact.

In this way, we have found a very straightforward approach to scaling up our quadrature formulas to higher dimensions. The bad news is that this approach suffers from the curse of dimensionality (see section 1.2). Calculating $Q_N^{\otimes d}(f)$ requires N^d function evaluations, making numerical evaluation impossible for larger N in high dimensions. We will return to this problem and an approach for its mitigation in the next chapter.

2.4.6 Convergence results for $\Omega = [0,1]^d$ and $\mu = \lambda^d\left([0,1]^d\right)$

In analogy to section 2.4.3, we now give estimates for convergence for functions on the multi-dimensional cube $[0,1]^d$ with $\mu = \lambda^d\left([0,1]^d\right)$. We take a function $f \in C^d(\Omega)$ and a quadrature formula Q_N that is exact on $\mathcal{P}_{N-1}$. We introduce the norm

$$\|f\|_{C^r} := \max_{i_1=0,\dots r} \dots \max_{i_d=0,\dots r} \max_{x\in\Omega} \left| \left(\frac{\partial}{\partial x_1}\right)^{i_1} \cdots \left(\frac{\partial}{\partial x_d}\right)^{i_d} f \right|$$

It can be shown that $\epsilon(N) \leq c_{d,r} N^{-\frac{r}{d}} \|f\|_{C^r}$ for some constants $c_{d,r}$ and for $N \geq r^d$, with the corresponding convergence rate of $\frac{r}{d}$[8]. Important for our considerations is the fact if we have a one-dimensional convergence rate of s, then the multi-dimensional convergence rate can, due to the curse of dimensionality, be only as good as $\frac{s}{d}$. Indeed, for a one-dimensional function $g \in \mathcal{L}^1\left(\lambda^1\left([0,1]\right)\right)$, we only need take $f \in \mathcal{L}^1\left(\lambda^d\left([0,1]^d\right)\right)$ with $f(x) = g(x_1)$. We then have

$$Q_N^{\otimes d}(f) = Q_N(g) \cdot Q_N(1_\Omega) \cdots Q_N(1_\Omega) = Q_N(g)$$

because Q_N is exact on 1_Ω. Therefore, we attain exactly the same convergence as we did in the one-dimensional case, however using N^d points for calculating $Q_N^{\otimes d}$ instead of only N points for Q_N. In the bilogarithmic plot of $\epsilon(N)$ over N, the abscissa is therefore stretched by the factor d, turning a convergence rate of s into only $\frac{s}{d}$. For this function, we are in effect wasting many evaluations N for the dimensions 2 to d where nothing is going on.

2.5 Nested quadrature

So far, we have not given consideration to the specific nature of the points where the function is evaluated. Of special interest is the question whether the points used to calculate $Q_N(f)$ are reused for higher values of N. The definitions 2.2.1 and 2.2.2 allow for both possibilities, because the point generator $p_{N,i}$ depends explicitly on N. This means that the points of evaluation may be different for every N. However, not all quadrature formulas make use of this freedom. For both Monte Carlo and Quasi-Monte Carlo, the points sequence is independent of N. This is in contrast to the quadrature formulas of section 2.4.2, where the nodes for the Gaussian formulas generally change completely with every N. We call formulas of the first kind **nested**, because new points are always added to (nested within) the old points.

This difference is important for **open-ended** (online) integration, where the value of N is not predetermined, but is continually increased until some sort of condition is met. In this case, nested quadrature formulas have the advantage that we only have one new function evaluation when going from Q_N to Q_{N+1}, whereas in the general case we would need to evaluate the function $N+1$ times. For example, we use open-ended integration if we want to obtain a quadrature value for a given error threshold. In this case, we increase N until the integration error (as estimated by some part of our algorithm) falls below the given threshold.

At first glance, it would seem that nested quadrature formulas have a large advantage in this case, because the number of function evaluations for $Q_1, \ldots, Q_N$ is N. In the case of a quadrature formula that is not nested, we have instead $\frac{1}{2}N(N+1)$, so that the evaluation complexity is of the order N^2instead of N. With a different strategy, we can however reduce this difference in degree to only a difference in constants. We achieve this by not evaluating Q_N for every N, but instead using the series $N_i = 2^i$, $i = 0, 1, \ldots$ If M is the minimum number of evaluations for which the condition is fulfilled, we have $N_i \geq M$ for some minimal r. For this r, we have $N_r \leq 2M$. The total number of evaluations is then

$$\sum_{i=0}^{r} N_i = \sum_{i=0}^{r} 2^i \leq 2^{r+1} = 2N_r \leq 4M$$

Thus for a non-nested formula, we need at most 4 times as many evaluations to reach a given condition. Thus, the difference between nested and non-nested quadrature boils down to a difference in constants. This especially entails that the convergence rate for open-ended integration is always the same as for predetermined N, irrespective of whether Q_N is nested or not nested.

Remark 2.5.1 *Calculation shows that the same qualitative result is obtained for* $N_i = \operatorname{ceil}\left(b^i\right)$ *for some* $b > 1$, *where* $\operatorname{ceil}(x) := \min\{n \in \mathbb{N} : n \geq x\}$ *is the smallest integer at least as large as* x, *albeit with a different constant. We calculate the asymptotic constant* $c(b)$ *using the notation from above*

$$\begin{aligned} c(b) &= \limsup_{M\to\infty} \frac{\sum_{i=0}^{r(M)} N_i}{M} \\ &= \limsup_{M\to\infty} \frac{1}{M} \frac{b^{r(M)+1} - 1}{b-1} \\ &= \limsup_{M\to\infty} \frac{1}{M} \frac{b \cdot b^{r(M)} - 1}{b-1} \\ &= \limsup_{M\to\infty} \frac{1}{M} \frac{b \cdot bM - 1}{b-1} \\ &= \frac{b^2}{b-1} \end{aligned}$$

$c(b)$ *has a minimum of* 4 *at* $b = 2$ *in the interval* $]1, \infty[$, *showing that our choice of*

base 2 *is optimal.*

2.6 Comparisons

In summary, we have the following results from the last section on the quadrature methods and their convergence.

Quadrature method	Function class	Error bound	Asymptotic convergence rate
Monte Carlo	$\mathcal{L}^1(\Omega)$	$\epsilon(N) \leq \frac{100}{\sqrt{N}} \sqrt{\|\mu\|} \, \|f\|_{\mathcal{L}^2}$ with at least probability 0.99	$\frac{1}{2}$ (in a statistical sense)
Quasi-Monte Carlo	Function on $[0,1]^d$ with bounded variation	$\epsilon(N) \leq V(f)C(d) \frac{(\log N)^d}{N}$	1
Polynomial interpolation	$C^r\left([0,1]^d\right)$	$\epsilon(N) \leq c_r N^{-r} \left\|\partial^r f\right\|_\infty$ for $N \geq r$	r
Tensor product polynomial interpolation	$C^r\left([0,1]^d\right)$	$\epsilon(N) \leq c_{d,r} N^{-\frac{r}{d}} \|f\|_{C^r}$ for $N \geq r^d$	$\frac{r}{d}$

We see that the smaller the function class for the quadrature (from top to bottom), the better the convergence rate is. This is not surprising: The higher the degree of smoothness, the less erratically the function can behave, and the quadrature methods can take advantage of this information.

Monte Carlo quadrature can be used for any function in $\mathcal{L}^1(\Omega)$, and does not make use of any differentiable or even continuous structure at all, working just as well for $\mathbb{R}$ as for $\mathbb{R}^{1000}$ or indeed any measure, whether it even has a dimension or not. On the other hand, it is also unable to take advantage of the smoothness offered by many of the problems of interest, always having the same statistical convergence rate of $\frac{1}{2}$. This means that for each further decimal digit of the integral we want to obtain, we need to evaluate 100 times as many points. For this reason, Monte Carlo quadrature is unsuitable for results that demand a high precision.

Quasi-Monte Carlo quadrature offers a good alternative to Monte Carlo for integration on $[0,1]^d$ for all but the most ill-natured functions, but is not able to make use of the additional information offered by differentiable functions. Also, the pre-asymptotic guaranteed convergence rate may be far lower than the asymptotic rate of 1, as we will see in the next section.

High degrees of smoothness are well exploited for 1-dimensional functions by interpolation quadrature. While tensor product interpolation quadrature also makes use of higher smoothness, it does so to a much lesser extent, having only a convergence rate of $\frac{r}{d}$. Even if, as is commonly the case, our function is of class $C^\infty(\Omega)$, simply taking a very high r for a fixed dimension d to obtain a large convergence rate $\frac{r}{d}$ is not an option, because this convergence rate only kicks in for $N \geq r^d$, again reflecting the curse of dimensionality.

In the next chapter, we will see that sparse grid methods allow us to avoid the curse of dimensionality for interpolation quadrature, significantly improving upon the convergence for multi-dimensional interpolation methods.

2.7 On the questionable significance of asymptotic behavior

We conclude this chapter by some remarks on the value of asymptotic analysis for practical computation, using the example of Quasi-Monte Carlo and Monte Carlo quadrature. As we have seen above, Quasi-Monte Carlo is half an order better than Monte Carlo asymptotically for functions with bounded variation. However, this is not true pre-asymptotically. Comparing the pre-asymptotic statistical rate of $\rho(N) = \frac{1}{2}$ for Monte Carlo and the pre-asymptotic rate of $\rho(N) = 1 - \frac{d}{\log N}$ for Quasi-Monte Carlo, we see that we need $N > e^{2d}$, or about $N > 4.8 \cdot 10^8$ for $d = 10$, for Quasi-Monte Carlo to have a better rate than Monte Carlo. For $d = 32$, we have $N > 6.3 \cdot 10^{27}$, placing the turnaround point firmly beyond the limits attainable by today's computers.

However, even these theoretical pre-asymptotic convergence rates may be quite misleading in practice, since they give only upper bounds which may be far too pessimistic. Taking the moderate dimension of $d = 10$, we have $\epsilon_{max}^{\mathbf{QMC}}(N) = c_1 N^{-1} (\log N)^{10}$ and $\epsilon_{max}^{\mathbf{MC}}(N) = c_2 N^{-\frac{1}{2}}$. For $\frac{c_1}{c_2} = 1$, we obtain the graph for $\frac{\epsilon_{max}^{\mathbf{QMC}}(N)}{\epsilon_{max}^{\mathbf{MC}}(N)}$ shown in figure 2.7.1 (p. 27). If $\frac{c_1}{c_2} < 1$ or $\frac{c_1}{c_2} > 1$, the graph is shifted up or down, respectively.

Let us examine how well this theoretical graph relates to practice for the simple test function $f(x_1, \ldots, x_{10}) = e^{-\sum_{i=1}^{10} x_i}$. The plot for the actual error quotient obtained for this function by numerical quadrature with a computer algorithm for one run each of Quasi-Monte Carlo and Monte Carlo can be seen in figure 2.7.2, p. 27. The same scale as for figure 2.7.1 was used. This makes for easy comparison of the two figures. Moreover, the white space to the right, for which computational results were not obtained due to run-time limitations, drastically illustrates the limits of numerical computability compared to theoretically required large Ns.

When comparing the two figures, we see immediately that they are qualitatively different. Especially, the actual convergence rate of Quasi-Monte Carlo seems to be better than that of Monte Carlo directly from the start, instead of worse for $N < 4.8 \cdot 10^8$ as suggested by theory. Indeed, the relative advantage seems to be

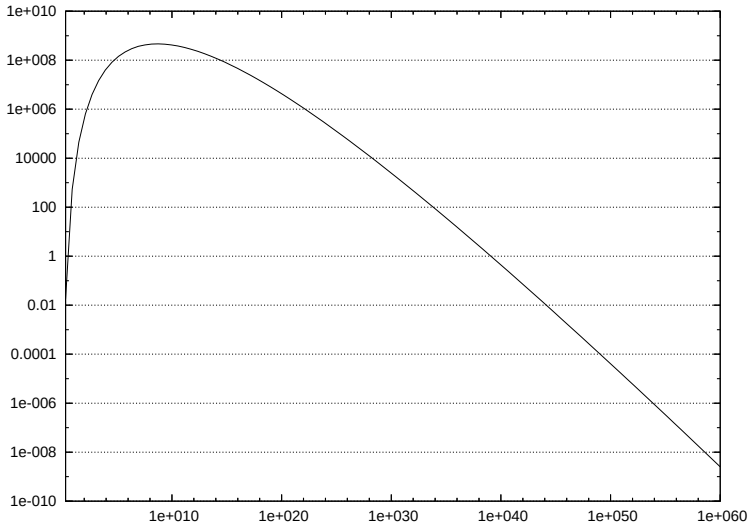


Figure 2.7.1: Bilogarithmic plot of the quotient $\frac{\epsilon_{max}^{\mathbf{QMC}}(N)}{\epsilon_{max}^{\mathbf{MC}}(N)}$, comparing the theoretical maximum errors for Quasi-Monte Carlo vs. Monte Carlo quadrature

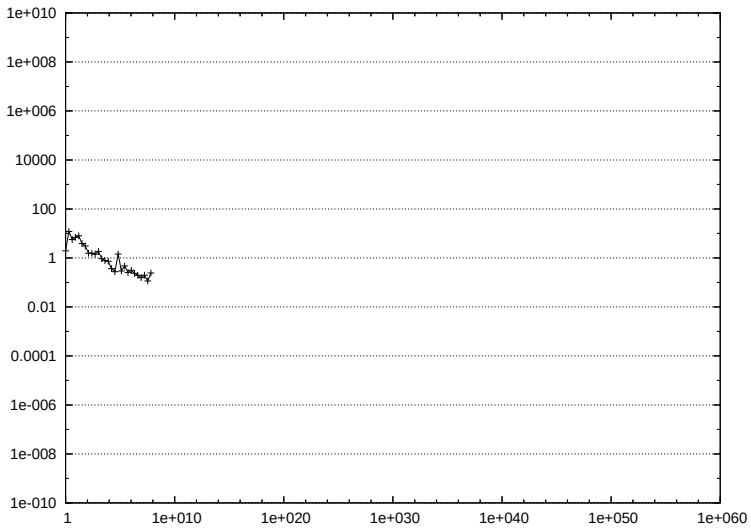


Figure 2.7.2: Actual quotient $\frac{\epsilon^{\mathbf{QMC}}(N)}{\epsilon^{\mathbf{MC}}(N)}$ for Quasi-Monte Carlo vs. Monte Carlo for numerical quadrature with a computer

close to $\frac{1}{2}$, which would only be expected asymptotically for very large N.

In summary, these comparisons show that the upper bounds offered by theory may be all but useless for practical considerations, and need always to be taken with at least several grains of salt.

3 Hierarchies and the Method of Sparse Grids

This chapter forms the conceptual core of this work. We have seen in the last chapter that we can obtain a multi-dimensional quadrature formula by combining one-dimensional formulas. While this method leads to correct results, it is also very inefficient, since it suffers from the curse of dimensionality. In 1963, Smolyak gave a method to avoid the curse of dimensionality [33]. This method has become known under several different names, including "sparse grid method", "Boolean method" and "discrete blending method". In all these case, the "blending" that is performed follows a fixed predetermined scheme. This chapter departs from traditional approaches by beginning with a generalized sparse grid method, which by nature turns out to be adaptive[17]. In this setting, the conventional predetermined sparse grid methods can be seen to arise as a special case. Specifically, we apply this adaptive method of sparse grids to the problem of quadrature. In this, we extend the dimension-adaptive sparse grid approach proposed in [16].

We give the standard results for convergence of non-adaptive sparse grid quadrature. We use a new approach for error estimation, which allows for a rather simple proof. This proof has the advantage that it is not limited to predetermined (simplicial) index sets, but can also be used for adaptive algorithms.

3.1 Definitions and notations

We will use the following notation:

$\mathbb{N}$	denotes the natural numbers including 0
α, β, γ, ...	denote multi-indices, i.e. elements of $\mathbb{N}^d$ for some dimension d
α_i	denotes the i-th component of the multi-index α, where $i = 1, \ldots, d$
$\lvert\alpha\rvert_1$	denotes the total length of the multi-index α, defined by $\lvert\alpha\rvert = \sum_{i=1}^{d} \alpha_i$
$\lvert\alpha\rvert_\infty$	denotes the maximum length of the multi-index α, defined by $\lvert\alpha\rvert = \max_{i=1,\ldots,d} \alpha_i$
$\mathbf{e}_i$	denotes the multi-index with a 1 at the i-th position and 0 otherwise, i.e. $(\mathbf{e}_i)_j = \delta_{ij}$

$\alpha \leq \beta$ — we declare a partial order on the multi-indices of a given dimension d by defining that $\alpha \leq \beta$ iff $\alpha_i \leq \beta_i$ for $i = 1, \ldots, d$, i.e. we use componentwise comparison. If $\alpha \leq \beta$ and $\alpha \neq \beta$, we say[1] that α is a predecessor of β, and that β is a successor of α.

Eval(A) — denotes the number of function evaluations needed to compute $A(f)$ for a Dirac sum A for any function f; Eval is a measure of computational complexity

A_α — for a family of Dirac sums $(A_i)_{i \in \mathbb{N}_0}$ on Ω we define the Dirac sum A_α on Ω^d by $A_\alpha = A_{\alpha_1} \otimes \ldots \otimes A_{\alpha_d}$ (see section 2.4.4 for the definition of Dirac sums and their tensor product)

3.2 The method of sparse grids

To apply the method of sparse grids to numerical quadrature, we define a refinement hierarchy of Dirac sums $(U_l)_{l \in \mathbb{N}}$ with $U_l = Q_{N(l)}$, where the strictly increasing mapping $N(l)$ correlates the level l of the hierarchy with the number of nodes of the quadrature formula Q_N. Often, $N(l)$ will increase exponentially with l. This immediately gives us a refinement hierarchy for the d-dimensional case. We need simply take $\left(U_l^{\otimes d}\right)_{l \in \mathbb{N}}$.

We define the family (Δ_l) of Dirac sums based on U_l by $\Delta_0 := U_0$, $\Delta_l := U_l - U_{l-1}$ for $l \geq 1$. We thus have $U_l = \sum_{i=0}^{l} \Delta_i$. We now rewrite $U_l^{\otimes d}$ in terms of Δ_l. By multi-linearity of the tensor product of Dirac sums, we have

$$\begin{aligned} U_l^{\otimes d} &= \left(\sum_{i=0}^{l} \Delta_i\right) \otimes \ldots \otimes \left(\sum_{i=0}^{l} \Delta_i\right) \\ &= \sum_{i_1=0}^{l} \ldots \sum_{i_d=0}^{l} \Delta_{i_1} \otimes \ldots \otimes \Delta_{i_d} \\ &= \sum_{|\alpha|_\infty \leq l} \Delta_\alpha \end{aligned}$$

with the definition of Δ_α given above. We thus have

$$U_l^{\otimes d}(f) = \sum_{|\alpha|_\infty \leq l} \Delta_\alpha(f)$$

If $U_l^{\otimes d}$ is limes-exact on a set of functions $\mathfrak{F}$, we have for $f \in \mathfrak{F}$

$$I(f) = \lim_{l \to \infty} U_l^{\otimes d}(f) \tag{3.2.1}$$

[1] While the notation $\alpha < \beta$ would of course be equivalent to $\alpha \leq \beta$ and $\alpha \neq \beta$ with the partial order given, we avoid this notation lest it be confused with the componentwise comparison $\alpha_i < \beta_i$ for $i = 1, \ldots, d$.

$$= \lim_{l\to\infty} \sum_{|\alpha|_\infty \le l} \Delta_\alpha(f)$$
$$= \sum_{l=0}^{\infty} \sum_{|\alpha|_\infty = l} \Delta_\alpha(f)$$

We see that approximating $I(f)$ by tensor product formulas of the type $U_l^{\otimes d}$ thus corresponds to adding the terms $\Delta_\alpha(f)$ in order of increasing size of $|\alpha|_\infty$.

The central insight on which the method of sparse grids is based comes from the fact this ordering is not necessarily the best for ensuring quick convergence to $I(f)$. This is due to two complementary considerations:

- The evaluation complexity $\mathrm{Eval}(\Delta_\alpha(f))$ is the product of the complexities of $\mathrm{Eval}(\Delta_{\alpha_i})$, $i = 1, \ldots, d$. It thus depends on all the components of α, not just on the maximum component. To give an example: If $d = 10$ and $\mathrm{Eval}(\Delta_i) = 2^i$, using $|\alpha|_\infty$ means giving the multi-indices $\alpha = (2, 0, 0, \ldots 0)$ and $\beta = (2, 2, \ldots, 2)$ the same precedence for evaluation, even though $\mathrm{Eval}(\Delta_\alpha) = 4$, whereas $\mathrm{Eval}(\Delta_\beta) = 4^{10} \approx 10^6$.
- The operator Δ_α represents a mixture of refinements in the different dimensions. Again, it depends on all components of α, not just on the maximum component.

To maximize the rate of convergence, we use the following general strategy:

Algorithm 3.2.1 *Let*

$$r_\alpha := \frac{|\Delta_\alpha(f)|}{\mathrm{Eval}(\Delta_\alpha(f))}$$

be the ratio of the contribution for α and the number of function evaluations required to calculate the contribution. Add the contributions $\Delta_\alpha(f)$ in order of decreasing r_α.

Note that this strategy involves a reordering of the indices. We will assume for the moment that this reordering is possible without changing the limit. A sufficient criterion for this is the absolute convergence of the series, which holds for many important cases, as we will show later.

This strategy is optimal in reducing the integration error $\epsilon(N)$ with respect to N if all contributions are positive. It may not be optimal for contributions of mixed signs, as these can cancel each other out to produce a smaller error term. We will make no attempt to predict the sign of contributions in this thesis, and therefore take the given algorithm as optimal under these circumstances.

Of course, we generally do not know the values of $\Delta_\alpha(f)$ without calculating them. The algorithm however would require us to know all the values in advance. Instead, we use heuristics to estimate the contributions from previous calculations, much as any adaptive algorithm does. We further assume that in general $|\Delta_\alpha(f)| \ge |\Delta_\beta(f)|$ for $\alpha \le \beta$, since for $\alpha \le \beta$ the Dirac sum Δ_β represents a higher level of refinement

than Δ_α for all dimensions. Since we generally have $\mathrm{Eval}(\Delta_\alpha) \leq \mathrm{Eval}(\Delta_\beta)$ in this case, we expect that mostly $r_\alpha \geq r_\beta$ for $\alpha \leq \beta$. Since we want to evaluate in order of decreasing r_α, this leads to the stipulation that we do not evaluate $\Delta_\beta(f)$ unless we have first evaluated $\Delta_\alpha(f)$ for all $\alpha \leq \beta$. This leads us to the following definition (compare [15]).

Definition 3.2.2 *We call a set of indices $A \subset \mathbb{N}^d$ **valid** iff for each index in A, all predecessors are also in A, i.e. iff $\forall \alpha \in A : \forall \beta \leq \alpha : \beta \in A$. We say that an index α is **valid** (with respect to A) iff $\alpha \notin A$ and $A \cup \{\alpha\}$ is valid.*

We consolidate our considerations in the following algorithm (compare [16]). The algorithm is the one central to this thesis, and the one on which the computer implementation is based. The result of quadrature is stored in the variable s.

Algorithm 3.2.3 *Start with $A := \emptyset$, $s := 0$.*
Repeat until a specified condition is reached:
From the set of indices valid with respect to A, pick an index α with the highest estimate for r_α
Set $A := A \cup \{\alpha\}$
Set $s := s + \Delta_\alpha(f)$
End Repeat

Remark 3.2.4 *Note that the one-dimensional refinement hierarchy $\Delta_l(g)$ represents a refinement of the integral for the complete function $g : \Omega \to \mathbb{R}$. Accordingly, if a multi-index α has a component $\alpha_i = l$, this means that the contribution $\Delta_\alpha(f)$ represents a refinement level of l for the whole dimension i. This is in contrast to the possibility of local refinement, where accuracy is increased selectively for parts of the domain of integration [2], and which we do not consider in this thesis. To make the distinction to locally adaptive quadrature clear, the term dimension-adaptive quadrature is used.*

Remark 3.2.5 *The construction given in this chapter is not limited to quadrature formulas. Indeed, we can use sparse grid methods for any kind of multi-dimensional objects that can be defined as a tensor product of one-dimensional objects, and for which there exists a one-dimensional hierarchy of refinement. Some examples are finite elements for partial differential equations or coefficient tensors for lossy data compression [17, 5].*

3.3 Estimates for $\Delta_\alpha(f)$ for $\Omega = [0,1]^d$ and $\mu = \lambda^d\left([0,1]^d\right)$

We take a positive semi-definite linear quadrature formula Q_N of Dirac sums so that Q_N is exact on $\mathcal{P}_{N-1}$ for each N (for construction of such formulas, see section 2.4.2). Note that because $1_\Omega \in \mathcal{P}_0$ and Q_N linear, Q_N is already consistent. We use the

hierarchy $U_l = Q_{N(l)}$ with $N(l) = 2^l$. For a one-dimensional function $g \in C^\infty([0,1])$ and $\mu = \lambda^1([0,1])$, we have

$$|\Delta_0(g)| = |Q_1(g)| \leq \|g\|_\infty \tag{3.3.1}$$

and for $l \geq 1$

$$|\Delta_l(f)| \leq |Q_{2^l}(f)| + |Q_{2^{l-1}}(f)| \leq 2\|g\|_\infty \tag{3.3.2}$$

Alternatively for $l \geq 1$, we have

$$|\Delta_l(f)| = |U_l(f-p) - U_{l-1}(f-p)|$$

for all $p \in \mathcal{P}_k$ with $k := 2^{l-1} - 1$, because U_l and U_{l-1} are exact for p. Recalling the definition of $\operatorname{dist}(f, \mathfrak{G})$ from section 2.4.1, this gives us the estimate

$$\begin{aligned} |\Delta_l(f)| &\leq \inf_{p\in\mathcal{P}_k} \left(|U_l(f-p)| + |U_{l-1}(f-p)|\right) \\ &\leq \inf_{p\in\mathcal{P}_k} \left(\|\mu\| \cdot \|f-p\| + \|\mu\| \cdot \|f-p\|\right) \\ &= 2 \cdot \operatorname{dist}(f, \mathcal{P}_k) \end{aligned}$$

In analogy to section 2.4.3, this leads to the estimates

$$|\Delta_l(f)| \leq c_r (k+1)^{-r} \|\partial^r f\|_\infty = c_r 2^{-r(l-1)} \|\partial^r f\|_\infty$$

for $2^{l-1} - 1 = k \geq r - 1$. With $p = 2^{-r}$, we therefore have

$$|\Delta_l(f)| \leq c_r 2^r p^l \|\partial^r f\|_\infty$$

for $2^l \geq r$. For those l for which $2^{l-1} < r$, we can use equations 3.3.1 and 3.3.2 to subsume the terms p^l under a constant. Taken together, this yields

$$|\Delta_l(g)| \leq a \cdot p^l \|g\|_{C^r}$$

for some constant a and the norm $\|g\|_{C^r} = \max_{i=0,\ldots,r} \left\|\partial^i g\right\|_\infty$.

We now translate this one-dimensional result for the convergence of $\Delta_i(g)$ to the multi-dimensional case. For this, we need the norm $\|.\|_{C^r}$ for multi-dimensional functions, which we already encountered in section 2.4.6.

We now give the main result of this section, following [29]. We use the notation $\partial_i = \frac{\partial}{\partial x_i}$ to denote the derivate with respect to the i-th component of a function.

Proposition 3.3.1 *For $f \in C^\infty\left([0,1]^d\right)$, we have $|\Delta_\alpha(f)| \leq a^d p^{|\alpha|_1} \|f\|_{C^r}$.*

Proof. We first examine the 2-dimensional case. Let $f \in C^\infty\left([0,1]^2\right)$. Let

$$g : x \mapsto \Delta_l(f(x,.))$$

Δ_l is a Dirac sum, and can therefore be written as

$$\Delta_l = \sum_{i=1}^{m} a_i \delta_{y_i}$$

for some a_i and y_i. We have for any $s \in \mathbb{N}_0$:

$$\partial^s g(x) \quad = \quad \partial_1^s \sum_{i=1}^{m} a_i f(x, y_i) = \sum_{i=1}^{m} a_i \partial_1^s f(x, y_i) = \Delta_l \left(\partial_1^s f(x, .)\right)$$

From the third term we can see that $\partial^s g$ exists and is continuous. Furthermore,

$$\begin{aligned} \|g\|_{C^r} &= \max_{s=0,\ldots,r} \max_{x \in \Omega} |\Delta_l \left(\partial_1^s f(x, .)\right)| \\ &\leq \max_{s=0,\ldots,r} \max_{x \in \Omega} a \cdot p^l \left\|\partial_1^s f(x, .)\right\|_{C^r} \\ &\leq a \cdot p^l \max_{x \in \Omega} \max_{s=0,\ldots,r} \left\|\partial_1^s f(x, .)\right\|_{C^r} \\ &= a \cdot p^l \|f\|_{C^r} \end{aligned}$$

Combining these two statements yields

$$|\Delta_k \otimes \Delta_l(f)| = |\Delta_k(g)| \leq a \cdot p^k \|g\|_{C^r} \leq a^2 \cdot p^{k+l} \|f\|_{C^r}$$

We obtain the proposition by iterating over d. □

3.4 General error bounds

We generalize the results from the last section with the following definition.

Definition 3.4.1 *We say that the contributions $\Delta_\alpha(f)$ are **exponentially convergent** to the base $p < 1$ with the constant c iff $|\Delta_\alpha(f)| \leq c \cdot p^{|\alpha|_1}$ for all α.*

Proposition 3.4.2 *Let $\Delta_\alpha(f)$ be exponentially convergent. Then the series $\sum_\alpha \Delta_\alpha(f)$ converges absolutely.*

Proof. We need to show that $\sum_\alpha |\Delta_\alpha(f)| < \infty$. We have

$$\begin{aligned} \sum_\alpha |\Delta_\alpha(f)| &= \sum_{k=0}^{\infty} \sum_{|\alpha|_1 = k} |\Delta_\alpha(f)| \\ &\leq \sum_{k=0}^{\infty} \sum_{|\alpha|_1 = k} c \cdot p^k \\ &= c \sum_{k=0}^{\infty} \#\left\{\alpha : |\alpha|_1 = k\right\} p^k \end{aligned}$$

Using the fact that $\#\{\alpha : |\alpha|_1 = k\} = \begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix}$, this implies[2]

$$\sum_{\alpha} |\Delta_\alpha(f)| \leq c \sum_{k=0}^{\infty} \begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix} p^k$$

We now apply the ratio test for this series:

$$\lim_{k\to\infty} \frac{\begin{pmatrix} k+1+d-1 \\ d-1 \end{pmatrix} p^{k+1}}{\begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix} p^k} = \lim_{i\to\infty} \frac{k+d}{k+1} \cdot p = p$$

Since we have $p < 1$ by assumption, this proves the proposition. □

For this section, we assume that $Q_N(f)$ converges to $I(f)$. With this assumption, proposition 3.4.2 allows us to write the error of quadrature as

$$\epsilon(N) = \sum_{\alpha \notin A} |\Delta_\alpha(f)|$$

where A is the set of indices whose contributions form the result $Q_N(f)$. Correspondingly, we have inequality

$$\epsilon(N) \leq \sum_{\alpha \notin A} |\Delta_\alpha(f)| \tag{3.4.1}$$

To find an estimate for this series, the following lemma is helpful.

Lemma 3.4.3 *Let $p < 1$, $l \in \mathbb{N}$. Then we have*

$$\sum_{k=l+1}^{\infty} \begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix} p^k = p^l \sum_{s=0}^{d-1} \begin{pmatrix} l+d \\ s \end{pmatrix} \left(\frac{p}{1-p}\right)^{d-s}$$

Proof. Let $F(p) = \sum_{k=l+1}^{\infty} p^{k+d-1}$. F is a power series in p with a 1 as radius of convergence. Because $p < 1$ by assumption, we can we can swap summation and differentiation:

$$\partial^{d-1} F(p) \quad = \quad \partial^{d-1} \sum_{k=l+1}^{\infty} p^{k+d-1}$$

[2] It is easy to remember the formula for the number of indices of a given length k by imagining that we have k entries to distribute among our d dimensions (for example $\bullet\bullet\bullet\bullet$ for $k = 4$). We now insert $d-1$ partitions between these entries (for example $\bullet|\bullet|\bullet\bullet \triangleq (1,1,2)$ or $\bullet\bullet\bullet||\bullet \triangleq (3,0,1)$ for $k = 4$ and $d = 3$). By basic combinatorics, the number of possibilities is then just $\begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix}$.

$$
\begin{aligned}
&= \sum_{k=l+1}^{\infty} \partial^{d-1} p^{k+d-1} \\
&= \sum_{k=l+1}^{\infty} (k+d-1)\cdots(k+1)\, p^k \\
&= (d-1)! \sum_{k=l+1}^{\infty} \binom{k+d-1}{d-1} p^k
\end{aligned}
$$

On the other hand, we can evaluate $F(p)$ as a geometric series, obtaining

$$
\begin{aligned}
\partial^{d-1} F(p) &= \partial^{d-1} \sum_{k=l+1}^{\infty} p^{k+d-1} \\
&= \partial^{d-1} \frac{p^{l+d}}{1-p} \\
&= \sum_{s=0}^{d-1} \binom{d-1}{s} \partial^s \left(p^{l+d}\right) \cdot \partial^{d-1-s} \left(\frac{1}{1-p}\right) \\
&= \sum_{s=0}^{d-1} \binom{d-1}{s} (l+d)\cdots(l+d-s+1)\, p^{l+d-s} \cdot \\
&\qquad 1\cdots(d-s-1)\, \frac{1}{(1-p)^{d-s}} \\
&= (d-1)! p^l \sum_{s=0}^{d-1} \binom{l+d-1}{s} \left(\frac{p}{1-p}\right)^{d-s}
\end{aligned}
$$

This leads to the equality

$$
(d-1)! \sum_{k=l+1}^{\infty} \binom{k+d-1}{d-1} p^k = (d-1)! p^l \sum_{s=0}^{d-1} \binom{l+d-1}{s} \left(\frac{p}{1-p}\right)^{d-s}
$$

which proves the proposition. □

We will use this inequality to obtain error bounds for sparse grid quadrature. First, we review Smolyak's original concept of predetermined **non-adaptive sparse grid quadrature**. In this case, we only allow index sets of the type $S^l := \{\alpha : |\alpha|_1 \leq l\}$, where S^l denotes the simplex of indices of depth l. This corresponds to evaluating indices in the order of increasing $|\alpha|_1$ in an online-line algorithm. Looked at in yet another way, it is equivalent to taking $c \cdot p^{|\alpha|_1}$ as an estimate for r_α in algorithm 3.2.3, which establishes the link between non-adaptive sparse grid quadrature and the concept of exponentially convergent contributions.

Because of the shape of the index set, we will also use the name of **simplicial sparse grid quadrature** for these this non-adaptive algorithm, and call indices ordered by increasing $|\alpha|_1$ **simplicial indices**.

We now give estimates for the simplicial method.

Proposition 3.4.4 *Let $\Delta_\alpha(f)$ be exponentially convergent to the base $p < 1$ with the constant c and let*

$$Q_N(f) = \sum_{\alpha \in S^l} \Delta_\alpha(f)$$

Then we have

$$\epsilon(N) \leq c \cdot p^l \sum_{s=0}^{d-1} \left(\begin{array}{c} l+d \\ s \end{array} \right) \left(\frac{p}{1-p} \right)^{d-s}$$

PROOF. Using equation 3.4.1, we have

$$\begin{aligned} \epsilon(N) &\leq \sum_{\alpha \notin A} |\Delta_\alpha(f)| \\ &\leq \sum_{\alpha \notin S^l} |\Delta_\alpha(f)| \\ &= \sum_{k=l+1}^{\infty} \sum_{|\alpha|_1 = k} |\Delta_\alpha(f)| \\ &\leq \sum_{k=l+1}^{\infty} \sum_{|\alpha|_1 = k} c \cdot p^k \\ &= c \sum_{k=l+1}^{\infty} \left(\begin{array}{c} k+d-1 \\ d-1 \end{array} \right) p^k \end{aligned}$$

Using lemma 3.4.3, we obtain the desired result. □

We now give an exceedingly simple corollary, which will nonetheless prove to be very important for general convergence theory of sparse grid methods, especially with regard to the hybrid algorithms that we will introduce in section 4.4.

Corollary 3.4.5 *Let $\Delta_\alpha(f)$ be exponentially convergent as above. Let$A \subset \mathbb{N}^d$ be the set of multi-indices evaluated for Q_N. Then if $A \supset S^l$, the estimate in the proposition for $\epsilon(N)$ also holds.*

PROOF. We note simply that

$$\epsilon(N) \leq \sum_{\alpha \notin A} |\Delta_\alpha(f)| \leq \sum_{\alpha \notin S^l} |\Delta_\alpha(f)|$$

□

For $l \geq d$, we can give the following simpler estimate

$$\begin{aligned} \epsilon(N) &\leq c \cdot \sum_{s=0}^{d-1} \left(\begin{array}{c} l+d \\ s \end{array} \right) \left(\frac{p}{1-p} \right)^{d-s} \\ &\leq c \cdot \sum_{s=0}^{d-1} \frac{(2l)^s}{s!} \left(\frac{p}{1-p} \right)^{d-s} \end{aligned}$$

$$\begin{aligned} &\leq c \cdot l^{d-1} \sum_{s=0}^{d-1} \frac{2^s}{s!} \left(\frac{p}{1-p}\right)^{d-s} \\ &= a \cdot p^l l^{d-1} \end{aligned}$$

with $a = c \cdot \sum_{s=0}^{d-1} \frac{2^s}{s!} \left(\frac{p}{1-p}\right)^{d-s}$. By adjusting the constant to cover the cases $l < d$, this corresponds to the estimate already given by Smolyak [33]

$$\epsilon(N) \leq a' \cdot p^l l^{d-1}$$

for some a' and $l \geq 1$, noting that p corresponds to $2^{-\alpha}$ in Smolyak's paper.

We now proceed to determine the relationship between N and l. For this, assume that $\text{Eval}(\Delta_i) \leq b \cdot q^i$ for some constant b and $q > 1$. Note that this is follows from the condition $\text{Eval}(U_i) \leq \frac{q}{q+1} b \cdot q^i$. This means that for a quadrature formula hierarchy $U_l = Q_{2^l}$, we have $b \leq \frac{3}{2}$ and $q = 2$. For the multi-dimensional case, we obtain

$$\text{Eval}(\Delta_\alpha) \leq b^d q^{|\alpha|_1}$$

The number of evaluations needed for S^l is then for $l \geq d$

$$\begin{aligned} N &= \sum_{\alpha \in S^l} \text{Eval}(\Delta_\alpha) \\ &= \sum_{k=0}^{l} \begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix} b^d q^l \\ &\leq b^d \sum_{k=0}^{l} (2l)^{d-1} q^l \\ &\leq b^d 2^{d-1} l^{d-1} \frac{q^{l+1}-1}{q-1} \end{aligned}$$

This gives us

$$N \leq \tilde{b} \cdot l^{d-1} q^l \tag{3.4.2}$$

for some constant $\tilde{b}$. On the other hand, we have

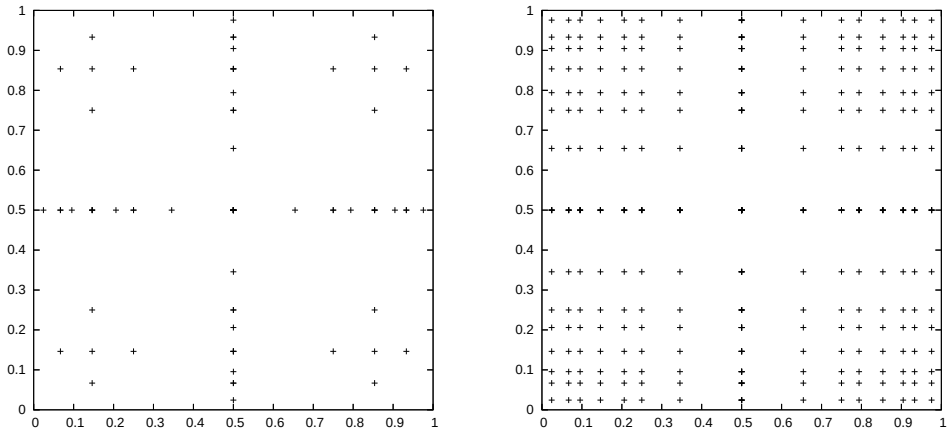
$$N = \sum_{\alpha \in S^l} \text{Eval}(\Delta_\alpha) = \sum_{k=0}^{l} \begin{pmatrix} k+d-1 \\ d-1 \end{pmatrix} b^d q^l \geq b^d q^l$$

and therefore

$$\log N \geq \hat{b} \cdot l \tag{3.4.3}$$

for some constant $\hat{b}$.

Now let Q_N be the simplicial sparse grid quadrature for the index set S^l. Since we have $p < 1$ and $q > 1$, there exists $s > 0$ with $p = q^{-s}$. Using equation 3.4.2, we



Sparse grid Full grid

Figure 3.4.1: A simplicial sparse grid vs. a full product grid

get the following estimate for the error for $l \geq 1$:

$$\begin{aligned} \epsilon(N) &\leq a \cdot p^l l^{d-1} \\ &= a \cdot \tilde{b}^s \cdot \left(\tilde{b} q^l l^{(d-1)}\right)^{-s} \cdot l^{(d-1)(s+1)} \\ &\leq a \cdot \tilde{b}^s N^{-s} \cdot l^{(d-1)(s+1)} \end{aligned}$$

Using equation 3.4.3, we obtain

$$\epsilon(N) \leq c \cdot N^{-s} \cdot (\log N)^{(d-1)(s+1)}$$

for some constant c and $N \geq 2$.

The quadrature formula used in the last section has $q = 2$ and $p = 2^{-r}$, so in this case we simply have $s = r$. The result in this case corresponds to that given in [29]. With this formula, we see that simplicial sparse grid quadrature has an asymptotic convergence rate of r, whereas the standard product quadrature has $\frac{r}{d}$. Recalling that standard product quadrature was equivalent to adding the contributions in order of increasing $|\alpha|_\infty$, we see that the simple change from $|\alpha|_\infty$ to $|\alpha|_1$ makes all the difference between the extremely bad convergence of $\frac{r}{d}$ due to the curse of dimensionality, and a asymptotic convergence rate r that is as good as for a one-dimensional function. Again, this is a very satisfying theoretical result. The logarithmic term in N and the constants involved are so large for high dimensions d however that we cannot hope to get near to asymptotic behavior in any actual calculations. We will see in the empirical data in chapter 6 however that adaptive sparse grid quadrature does far better than tensor product quadrature pre-asymptotically as well, as the general arguments given at the beginning of section 3.2 are generally valid.

Remark 3.4.6 *We close with a remark on the origin of the name "sparse grid" [35]. If we plot the points at which the function is evaluated for simplicial quadrature on*

S^l as opposed to tensor product quadrature of level l, the former grid appears thinned out, or "sparse". This can be seen in figure 3.4.1 on the preceding page, which shows the plots example for the nested Clenshaw-Curtis[7] rules with the hierarchy $U_l = Q_{2^l}$ for $d = 2$ and $l = 3$.

4 Index Refinement and Error Estimates

4.1 Introduction

In the last chapter, we gave algorithm 3.2.3 for dimension-adaptive sparse grid integration. While good as it stands, it leaves many details to be worked out. Most important of these, it begs the question of how the estimates for r_α should be obtained. We also need to address another problem, which arises for all numerical approximation methods. It is the fact that simply returning a number for the integral estimate is worthless unless some sort of error bound is also given. It makes quite a difference to receive a quadrature value of 2.14892 with error bounds of 1e-5, 1e-1 or 1e100. As we will see, the problems of estimating the r_α and of estimating the error are intertwined. For this reason, we treat them both in the same chapter.

4.2 Index refinement

We use the term **index refinement** to describe the process of successively picking indices for evaluation. As realized in algorithm 3.2.3, this corresponds to estimating the values r_α for the valid indices at each step, and picking the index α with the highest estimate. We recall that

$$r_\alpha = \frac{|\Delta_\alpha(f)|}{\text{Eval}\,(\Delta_\alpha)}$$

Since $\text{Eval}\,(\Delta_\alpha)$ is known, the problem boils down to estimating $|\Delta_\alpha(f)|$.

4.2.1 Estimation using the direct predecessors

One possibility for estimating $|\Delta_\alpha(f)|$ is by using the values $|\Delta_\beta(f)|$ that we have already obtained. We say that β is a direct predecessor of α if $\beta = \alpha - \mathbf{e}_i$ for some $i = 1, \ldots, d$. For a given α, let $c_1, \ldots, c_k$ be the values of $|\Delta_\beta(f)|$ for the direct predecessors β of α. This leads us directly to the following estimates for $|\Delta_\alpha(f)|$:

$\left(\prod_{i=1}^k c_i\right)^{\frac{1}{k}}$ (the geometric mean estimator)

$\min_{i=1,\ldots,k} c_i$ (the minimum estimator)

$\max_{i=1,\ldots,k} c_i$ (the maximum estimator)

Note that all these estimates are only defined if α has a least one direct predecessor, so that $k \geq 1$. This does not pose a problem. The only index that does not have a direct predecessor is the zero index $\alpha = (0, \ldots, 0)$. Since the zero index is the only index valid for the empty set, it is always the first index that is evaluated. For this reason, we can set the estimate r_α for the zero index to an arbitrary value, for example 0. Since this estimate is not based on any information from the function, it cannot be used for error estimation, so we start giving error estimates only when the first index (or better the first few indices) have actually been evaluated.

Our choice of the three estimates geometric mean, minimum and maximum raises the question of why the arithmetic average and the hyperbolic mean were not considered. The reason for this is practical in nature. For many functions, the values c_i will usually differ by orders of magnitude. In this case, the arithmetic average lies close to the maximum, and the hyperbolic average lies close to the minimum. We therefore restrict our considerations to the three most salient possibilities.

4.2.2 Estimation by evaluation

Another very direct way to estimate $|\Delta_\alpha(f)|$ is to compute it by evaluating $\Delta_\alpha(f)$ (note that the term estimate is used in a rather misleading sense here). Of course, since each evaluation $\Delta_\alpha(f)$ represents a further refinement of the quadrature value, and having calculated the $\Delta_\alpha(f)$ for all valid indices anyway, we would be ill-advised not to use these contributions for the quadrature result. This leads to the following, somewhat modified version of algorithm 3.2.3. This algorithm corresponds closely to the one given in [16]. The only difference is that in [16], instead of picking the index with the highest value for r_α, the index with the highest value of $|\Delta_\alpha(f)|$ is chosen; that is, the contribution is not weighted by the evaluation complexity.

Algorithm 4.2.1 *Start with $A := \emptyset$.*

Repeat until a specified condition is reached:

Let B be the set of valid indices with respect to A.

Set $s := \sum_{\alpha \in A \cup B} \Delta_\alpha(f)$.

From the set of indices valid with respect to A, pick the index α with the highest value for r_α.

Set $A := A \cup \{\alpha\}$.

End Repeat

We see that at any point of the algorithm, the result of quadrature s is the sum of all contributions from indices that are either in A or valid with respect to A. The contribution $\Delta_\beta(f)$ must be calculated as soon as β becomes valid. The assignment $s := \sum_{\alpha \in A \cup B} \Delta_\alpha(f)$ can be replaced by $s := s + \sum_{\alpha \in C} \Delta_\alpha(f)$, where C is the set of those indices in $A \cup B$ whose contributions have not yet been added to s. While this approach takes a little more work to implement, it should be used in practice, since it avoids adding all contributions from scratch for each iteration.

4.2.3 Trivial estimation

Finally we give a trivial case. We simply estimate $|\Delta_\alpha(f)|$ to be some constant, for example 1 (again, using the word estimation in a rather loose sense). In this case, we have $r_\alpha = \frac{1}{\text{Eval}(\Delta_\alpha)}$, meaning that the indices are traversed in the order of increasing evaluation complexity. In many cases, for example if $\text{Eval}(\Delta_i) = 2^i$, we have $\text{Eval}(\Delta_\alpha) > \text{Eval}(\Delta_\beta)$ for $|\alpha|_1 > |\beta|_1$. In these cases, the adaptive algorithm defaults to the classical simplicial method (with the addition that it is possible that the algorithm may finish when only part of the last layer $\{\alpha : |\alpha|_1 = l\}$ of the simplex S_l has been added, leaving us with an index shape in between $S_{l-1}$and S_l).

4.3 Error estimates

We now turn to the problem of error estimates. The most important property of an error estimate $\eta(N)$ is that it is **valid**, i.e. that we have $\epsilon(N) \leq \eta(N)$ for all N. Put differently, $\eta(N)$ should never underestimate the error $\epsilon(N)$, as this would lead to spurious results for all work depending on these quadrature results. On the other hand, we also want to avoid a large degree of overestimation. Note that with this definition, $\eta(N) = 10^6$ is a valid estimate if $\epsilon(N) = 2^{-N}$, but it is certainly not a very good one. Therefore, while we always require our estimates to be valid, we also wish for them to be **efficient** for them to be useful, i.e. for $\eta(N)$ generally to be only slightly larger or at least of the same order of magnitude as $\epsilon(N)$.

4.3.1 Estimates using the index structure

A first observation is that if at a given iteration of the algorithm we take the set A of evaluated indices, the current result of quadrature is $s = \sum_{\alpha \in A} \Delta_\alpha(f)$. The error is then constrained by

$$\begin{aligned}
\epsilon(N) &= |s - I(f)| \\
&= \left| \sum_{\alpha \in A} \Delta_\alpha(f) - \sum_{\alpha \in \mathbb{N}^d} \Delta_\alpha(f) \right| \\
&= \left| \sum_{\alpha \notin A} \Delta_\alpha(f) \right| \\
&\leq \sum_{\alpha \notin A} |\Delta_\alpha(f)|
\end{aligned}$$

We now assume that $\Delta_\alpha(f)$ is exponentially convergent to the base $p < 1$ (see definition 3.4.1). We further assume that this estimate is reflected in the actual contributions, that is, that we have

$$|\Delta_\beta(f)| \leq p^{(|\beta|_1 - |\alpha|_1)} \cdot |\Delta_\alpha(f)|$$

for all successors β of α. This allows us to give a bound for the sum of the contributions of all these successors:

$$\begin{aligned}
\sum_{\substack{\beta \geq \alpha,\\ \beta \neq \alpha}} |\Delta_\beta(f)| &\leq |\Delta_\alpha(f)| \cdot \sum_{\substack{\beta \geq \alpha,\\ \beta \neq \alpha}} p^{\left(|\beta|_1 - |\alpha|_1\right)} \\
&= |\Delta_\alpha(f)| \cdot \left(\left(\sum_{\gamma \in \mathbb{N}^d} p^{|\gamma|_1} \right) - 1 \right) \\
&= |\Delta_\alpha(f)| \cdot \left(\left(\sum_{i \in \mathbb{N}} p^i \right)^d - 1 \right) \qquad (4.3.1) \\
&= |\Delta_\alpha(f)| \cdot \left(\left(\frac{1}{1-p} \right)^d - 1 \right) \\
&= |\Delta_\alpha(f)| \cdot \left(\left(\frac{1}{1-p} \right)^d - 1 \right)
\end{aligned}$$

We now give a lemma and a proposition that turn this inequality for a single given multi-index into a global error estimator.

Lemma 4.3.1 *Let A be a valid set of indices. Let $\gamma \notin A$. Then there exists $\beta \leq \gamma$ so that β is valid with respect to A.*

Proof. We give a proof by induction.

For $d = 1$, we take $i = \max\{j : (k) \in A \text{ for all } k < j\}$. The maximum exists because 0 is always a member of the set and the set is bounded by γ_1. Let $\beta = (i)$. By definition of i, all predecessors of β lie in A, therefore β is valid. Since $\gamma \notin A$, this also means that γ cannot be a predecessor of β. Since the partial order of multi-indices is a total order for dimension 1, we conclude $\beta \leq \gamma$.

For $d > 1$, we assume the proposition has already been proved for the dimension $d-1$. For any d-dimensional set of indices D we denote the $(d-1)$-dimensional section through D at the height s in the last component by $D^{(s)}$, i.e.

$$D^{(s)} := \left\{ \alpha \in \mathbb{N}^{d-1} : (\alpha_1, \ldots, \alpha_{d-1}, s) \in D \right\}$$

Further, define $\check{\alpha} = (\alpha_1, \ldots, \alpha_{d-1})$ for any d-dimensional index α.

Now take $s = \gamma_d$. The set $A^{(s)}$ is valid, because for $\alpha' \in A^{(s)}$ and $\beta' \leq \alpha'$, we know that $(\beta_1, \ldots, \beta_{d-1}, s) \leq \alpha$ and therefore $\beta' \in A^{(s)}$ by definition. We have $\check{\gamma} \notin A^{(s)}$. This means that we can apply the inductive hypothesis to obtain a $\beta' \leq \check{\gamma}$ so that β' is valid with respect to $A^{(s)}$. In particular, this means that $\beta' \notin A^{(s)}$. Now let $r = \max\left\{j : \beta' \in A^{(k)} \text{ for all } k < j\right\}$. As above, the maximum exists, and we have $r \leq s = \gamma_d$.

Let $\beta := \left(\beta'_1, \ldots, \beta'_{d-1}, r\right)$. Because $\beta' \leq \check{\gamma}$ and $r \leq \gamma_d$, we have $\beta \leq \gamma$, which proves the first part of the lemma. Furthermore, we have $\beta' \notin A^{(r)}$, as otherwise r

would not be maximum. By definition of $A^{(r)}$, this implies $\beta \notin A$.

Now let α be any predecessor of β. If $\alpha_d = r = \beta_d$, then $\check{\alpha}$ is predecessor of $\check{\beta}$ and therefore $(\alpha_1, \ldots, \alpha_{d-1}, s)$ is predecessor of $(\beta_1, \ldots, \beta_{d-1}, s) = \beta'$. Because β' is valid with respect to $A^{(s)}$, we have $(\alpha_1, \ldots, \alpha_{d-1}, s) \in A$ and finally $\alpha \in A$ because $\alpha \leq (\alpha_1, \ldots, \alpha_{d-1}, s)$ and A is valid. If on the other hand we have $\alpha_d < r$, then because $\big(\beta'_1, \ldots, \beta'_{d-1}, \alpha_d\big) \in A$ and $\alpha \leq \big(\beta'_1, \ldots, \beta'_{d-1}, \alpha_d\big)$, we also have $\alpha \in A$. Since α was an arbitrary predecessor of β, and $\beta \notin A$, we know that β is valid with respect to A. This proves the second part of the lemma, and completes the proof. □

Proposition 4.3.2 *Let A be a valid set of indices and let B be the set of indices valid with respect to A. Assume that*

$$|\Delta_\gamma(f)| \leq |\Delta_\beta(f)|\, p^{(|\gamma|_1 - |\beta|_1)} \tag{4.3.2}$$

for $\gamma \geq \beta$ and $p < 1$. Then

$$\sum_{\gamma \notin A \cup B} |\Delta_\gamma(f)| \leq c \cdot \sum_{\beta \in B} |\Delta_\beta(f)|$$

with $c = \left(\frac{1}{1-p}\right)^d - 1$.

Proof. Using inequality 4.3.1, we have

$$\begin{aligned}
c \cdot \sum_{\beta \in B} |\Delta_\beta(f)| &= \sum_{\beta \in B} c \cdot |\Delta_\beta| \\
&\geq \sum_{\beta \in B} \sum_{\substack{\gamma \geq \beta, \\ \gamma \neq \beta}} |\Delta_\gamma(f)| \\
&= \sum_{\beta \in B} \sum_{\gamma \in D_\beta} |\Delta_\gamma(f)| \ \text{ with } D_\beta := \{\gamma : \gamma \geq \beta,\, \gamma \neq \beta\} \\
&\geq \sum_{\gamma \in D} |\Delta_\gamma(f)| \ \text{ with } D := \bigcup_{\beta \in B} D_\beta
\end{aligned} \tag{4.3.3}$$

Now let $\gamma \notin A \cup B$. By the preceding lemma, we know there exists a $\beta \leq \gamma$ so that $\beta \in B$. Because $\gamma \notin B$, we even have $\gamma \neq \beta$. This means that $\gamma \in D_\beta \subset D$. Since $\gamma \notin A \cup B$ was arbitrary, we have $D \supset \{\gamma : \gamma \notin A \cup B\}$. Combining it with the inequality above, we obtain

$$\sum_{\gamma \notin A \cup B} |\Delta_\gamma(f)| \leq \sum_{\gamma \in D} |\Delta_\gamma(f)| \leq c \cdot \sum_{\beta \in B} |\Delta_\beta(f)|$$

which proves the proposition. □

In this way, we have found a way to estimate $\epsilon(N)$ with the sum of contributions of indices that are currently valid with respect to A. Note that the sum $c \cdot \sum_{\gamma \notin A \cup B} |\Delta_\gamma(f)|$

for the quadrature error implies that we take $s = \sum_{\alpha \in A \cup B} \Delta_\alpha(f)$ and not only $s = \sum_{\alpha \in A} \Delta_\alpha(f)$ in the algorithm. This was exactly the case for our modified algorithm 4.2.1, so we can immediately use $c \cdot \sum_{\gamma \notin A \cup B} |\Delta_\gamma(f)|$ as an error estimate for it. However, we have the problem that p is generally not known.

The error estimate given in [16] corresponds to the estimate just given, with c heuristically set to 1. With the above constraints and theory, we can predict this estimator to only be valid if $c \leq 1$, that is, if $p \leq 1 - \frac{1}{\sqrt[d]{2}}$. Even if this is not the case, the estimator does seem to work reasonably well in practice. This is due to the fact we have given inequalities and not equalities. Especially in 4.3.3, many indices β that occur in several D_β are now considered only once in D. Also, we have used the absolute value of contributions throughout, and the cancellation of contributions of opposing signs may also attenuate the actual quadrature error.

There are other valid concerns, though. In particular, it is not clear how well the assumption 4.3.2 holds. Obviously, sometimes the contribution $\Delta_\gamma(f)$ will have a small absolute value simply by cancellation. We can hope that these sorts of effects become statistically small if we have a large number of indices. However, there may also be structural problems with 4.3.2, which may invalidate this approach.

We can modify this error estimate to work with our original algorithm 3.2.3. Instead of the actual contributions we have to take the *estimates* for the $|\Delta_\beta(f)|$ with β valid, since the actual values are unknown. In this algorithm, we have $s = \sum_{\alpha \in A} \Delta_\alpha(f)$, so we need to add the estimate for $\sum_{\beta \in B} |\Delta_\beta(f)|$ to the error estimate. In this way, we arrive at the estimate $(c+1) \cdot \sum_{\beta \in B} d_\beta$, where each d_β is the estimate for $|\Delta_\beta(f)|$. Again, we do not know the value of c, requiring us to take some heuristic value.

4.3.2 Black box estimates

The approach developed in the last section is in a way very natural, as it takes know local contributions and adds them to arrive at a global estimate, but we have seen that it poses many problems. A totally different approach ignores all this information, treating the sequence of quadrature results $Q_N(f)$ as a black box. This approach is suggested by the fact that for many functions in practice, we have a robust convergence with a stable convergence rate over large stretches of N, i.e. $\epsilon(N) \approx c \cdot N^{-r}$ for some $r > 1$ and some range $N \in [N_0, N_1]$. As we have seen in the last chapter, this sort of convergence is also expected from theoretical results.

Viewed in a bilogarithmic plot, we have $\log(\epsilon(N)) \approx \log c - r \log N$. Using a simply linear regression, we can estimate the constants c and r, yielding an error estimate of

$$\eta(N) = c_{\text{est}} \cdot N^{-r_{\text{est}}}$$

The problem is of course that we do not know $\epsilon(N)$ itself. All we have is the values $Q_N(f)$ as a function of N. Since $Q_N(f)$ is an estimate for $I(f)$, for a given N and for $M \leq N$ we can estimate the error $\epsilon(M) = |Q_M(f) - I(f)|$ by $\zeta_1(M, N) := |Q_N(f) - Q_M(f)|$. Note that that the estimate ζ_1 we have given for

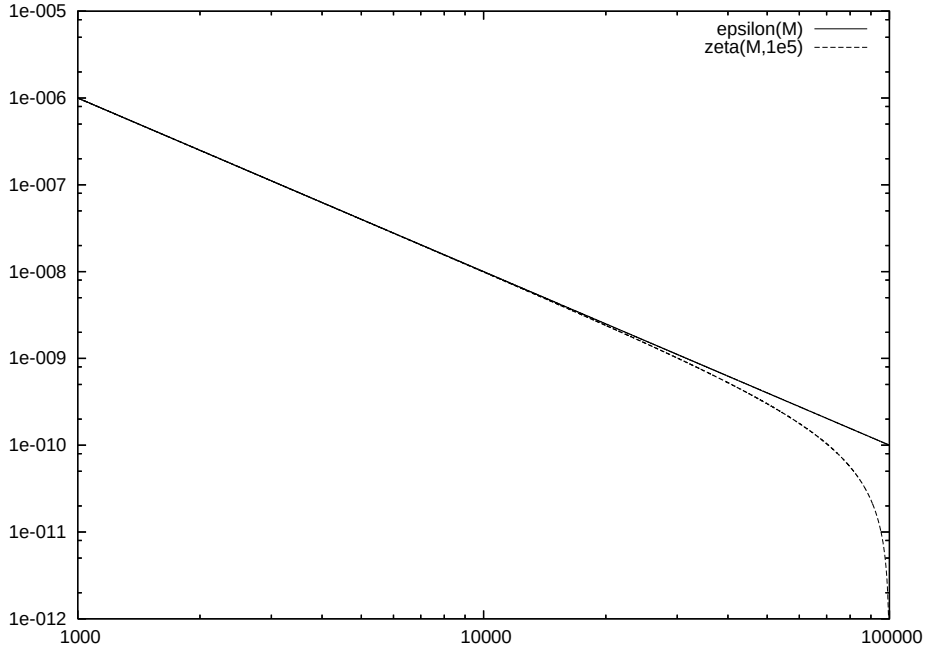


Figure 4.3.1: $\epsilon(M)$and $\zeta(M,N)$ for $\epsilon(N) = N^{-2}$ and $N = 10^5$

$\epsilon(M)$ is dependent on both M and N. Indeed, for a fixed N, the estimate $\zeta_1(M,N)$ will generally become more accurate with increasing N, because $Q_N(f) \to I(f)$ for $N \to \infty$.

Note however that the closer M gets to N, the more we will underestimate the error. To give an example, we take $\epsilon(N) = N^{-2}$. The resulting graphs for $\epsilon(M)$ and $\zeta_1(M,N)$ are shown in figure 4.3.1. We see that if M gets to close to N, we lose the linearity of the function. If we want to get a good fit for our regression, we must restrict ourselves to those values for M that are suitably smaller than N.

There is yet another concern. So far, we have assumed that $\epsilon(N)$ decreases with N. But this is usually not the case. Instead, $\epsilon(M)$ will often oscillate around the value 0, with the oscillations getting smaller with N. What we are interested in is not the error per se, but rather the expected degree of inexactness. Thus, our definition of ζ_1 does not really capture our intent. One way to deal with this problem is to simply stipulate that $\zeta(M,N)$ be decreasing in M. This leads us to the definition

$$\zeta_2(M,N) := \max_{M' \geq M} \zeta_1(M,N)$$

In this way, $\zeta_2(M,N)$ represents the maximum difference of $Q_{M'}(f)$ to $Q_N(f)$ for $M \leq M' \leq M$, capturing the intuition of the remaining inexactness or volatility of $Q_M(f)$ far better. We can see this difference in figure 4.3.2.

The problem of underestimating for ζ for M close to N still persists, however. We have undertaken several attempts to overcome this problem, but none have been very successful. An adaptive algorithm that attempts to lock in on the part of the curve

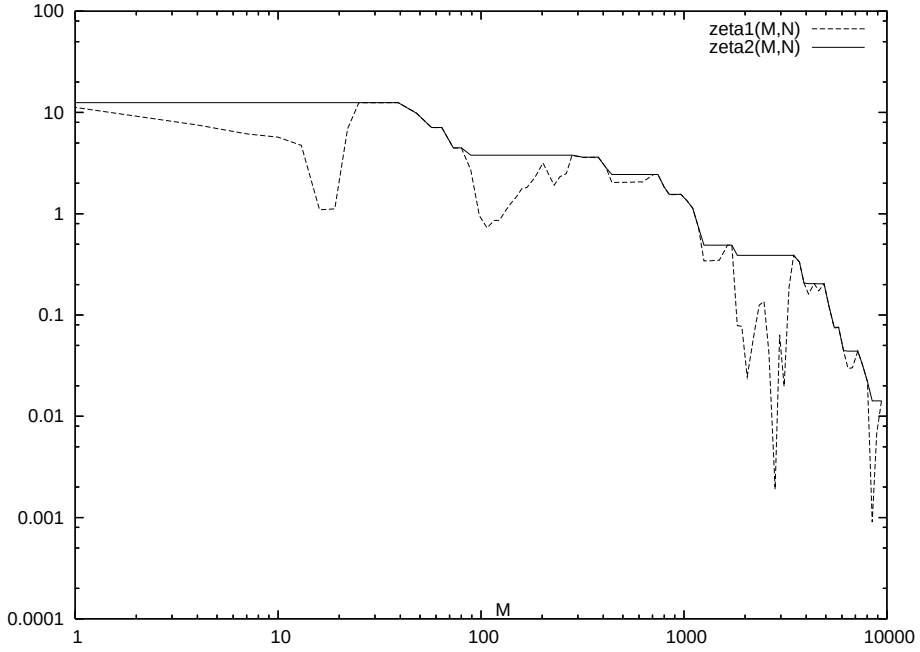


Figure 4.3.2: $\zeta_1(M,N)$ vs. $\zeta_2(M,N)$ for $N = 10^5$ and a Genz oscillatory function for $d = 8$ and $h = 18$ (see chapter 6)

where the $\zeta(M,N)$ deviates downward works in some cases, but goes wholly astray in others. A more complex approach, which explicitly modeled this deviation, had the same problem. In the end, we have settled to simply taking the interval $[M^{\frac{1}{2}}, M^{\frac{3}{4}}]$, that is, the third quarter of value of M in the bilogarithmic representation. This seems to work reasonably well in practice, but is unsatisfactory from a theoretical point of view.

Yet another problem is that the exponent r used above is not really a constant. Heuristically speaking, it may change over time. For example for functions of the class C^∞, r will increase with N.

In conclusion, the black box estimate given in this section is far more heuristic than the index-based estimate. Somewhat surprisingly, it often performs better in practice, as we will see in chapter 6. Still, unsolved problems remain with this approach, especially with the lower and upper cutoff for M used in regression.

4.4 Hybrid algorithms

It is very difficult to give error bounds for purely adaptive algorithms. The general problem is that the algorithm bases its decisions on where to perform further refinement on numerical values obtained at previous levels of refinement. If these values are very small, the algorithm will not continue along this path of refinement. In algorithm 3.2.3, for example, if an index α for some reason gets an r_α that is very small, its contribution will not be calculated for a long time (i.e. until all competing

valid indices β have even smaller values r_β). As long as α is not added to the index set, it also blocks further refinement past this index, as no β with $\beta \geq \alpha$, $\beta \neq \alpha$ can become valid as long as $\alpha \notin A$.

For example, we can construct non-constant functions f of class C^∞ that have $\Delta_{\mathbf{e}_1}(f) = 0$. Indeed, this is true for all $f \in C^\infty$ that have $f(x) = 0$ for all points x evaluated for $\Delta_{\mathbf{e}_1}$. As consequence, the algorithms given in section 4.2 will not evaluate $\Delta_{\mathbf{e}_2}(f)$, generally making convergence impossible. Non-adaptive sparse grid quadrature does not suffer from this problem, and we have given convergence results to this effect in the last chapter.

What we would wish for, then, is a sparse grid quadrature algorithm that is at least as good as an adaptive version while retaining the guaranteed convergence of the simplicial (static) version. This is easier than it sounds. We need simply let the adaptive and the non-adaptive version run in parallel, i.e. alternating between the two with every evaluation and then selecting the better of the two results at each step. This would only mean a moderate increase in evaluation complexity for each algorithm by the constant factor 2. We would, however, need to figure out how to choose which of the two results is better at any given time.

But this is not necessary, because we can do even better. Instead of letting the two approaches, adaptive and static, run independently, we merge them, letting them inform each other. For this, we give a modified hybrid version of our main algorithm 3.2.3 .

Algorithm 4.4.1 *Start with* $A := \emptyset$, $s := 0$, $n_{adapt} := 0$, $n_{static} := 0$.

Repeat until a specified condition is reached:

From the set of indices valid with respect to A*, pick an index* α *with the highest estimate for* r_α

From the set of indices valid with respect to A*, pick an index* β *with the smallest size of* $|\beta|_1$

If $n_{adapt} + \mathrm{Eval}(\Delta_\alpha) \leq n_{static}$ *set* $\gamma := \alpha$, $n_{adapt} := n_{adapt} + \mathrm{Eval}(\Delta_\alpha)$

otherwise set $\gamma := \beta$, $n_{static} := n_{static} + \mathrm{Eval}(\Delta_\beta)$

Set $A := A \cup \{\gamma\}$

Set $s := s + \Delta_\gamma(f)$

End Repeat

In this way we alternate between adaptive and static indices, giving each an equal share of the function evaluations. Of course, we cannot always exactly have $n_{adapt} = n_{static}$. Indeed, once we pick a simplicial index α for evaluation, n_{static} increases by $\mathrm{Eval}(\Delta_\alpha)$, which n_{adapt} remains constant, and vice versa for an adaptive index. The algorithm above resolves this problem in a very one-sided manner, ensuring that at any time we have $n_{adapt} \leq n_{static}$, that is, it is biased towards preferring simplicial indices. This bias was chosen because we have known convergence results for the simplicial algorithm, but none for the adaptive algorithm, and this bias allows us to have the same guaranteed convergence rate for the hybrid algorithm by assuring that at all times at least half the function evaluations have been used for simplicial

indices. Indeed, if for some number of function evaluations M the non-adaptive algorithm has completed the simplex S^l, then we have $A \supset S^l$ for the hybrid version at $N = 2M$. Using proposition 3.4.5, we see that $\epsilon_{max}^{hybrid}(N) \leq \epsilon_{max}^{static}\left(\frac{N}{2}\right)$. In terms of convergence rate, this translates to $\rho^{hybrid}(N) = \rho^{static}\left(\frac{N}{2}\right)$, in particular giving us the same asymptotic convergence rate. On the other hand, the algorithm still uses half its evaluations for adaptive indices, so we can expect it to perform as well as the fully adaptive algorithm (again, with the caveat of needing an N that is twice as large).

With this hybrid algorithm, adaptive and static methods work on the same index set, cooperating and informing each other, instead of only running in parallel. Indeed, some indices will be selected by both the static and the adaptive method. By operating on a shared set of evaluated indices, these indices are evaluated only once, in effect halving the evaluation complexity for these indices for each component. Even more important is another result of this reciprocal cooperativity. Returning to the example above, where we had $\Delta_{\mathbf{e}_1}(f) = 0$, we see that now the index $\mathbf{e}_1$ would soon be evaluated as a static index. In this way, all indices $\beta \geq \mathbf{e}_1$, $\beta \neq \mathbf{e}_1$ are not only evaluated by the static method, but also opened up for the adaptive algorithm, which so to speak is lifted over the road-block of $\mathbf{e}_1$ by the static method. We can therefore expect hybrid methods to attain better results than the optimum of the solely static or the solely adaptive methods together.

Of course, the number of function evaluations allotted to the non-adaptive method need not be $\frac{1}{2}$, and we can easily modify the algorithm to support any ratio. For any ratio $r > 0$ of simplicial function evaluations out of all evaluations, we can give the same guarantee on convergence and asymptotic convergence rate as above, noting that in this case the convergence is slowed by the factor $\frac{1}{r}$. For the special case of $r = 1$, we get the non-adaptive version of the algorithm. On the other hand, for the ratio $r = 0$, we get the fully adaptive version, and do not have a guarantee of convergence as above.

5 The Implementation

5.1 Introduction

We have implemented the dimension-adaptive algorithm of sparse grid integration for the computer. Although the algorithm in its stated form seems simple enough, many difficult choices have to be made during implementation pertaining to problems such as estimation of the index contribution, error estimation and data structures. In many cases, it was not clear a priori what the best choice would be. For this reason, an object-oriented programming language was chosen, which allows for the easy and robust design of the program in a modular fashion. For each of these modules we can then program different implementations, which can be mixed and matched seamlessly to find the optimal combination.

We have chosen Java as a programming language for several reasons. The most important is its stringent object-oriented design, which allows for a clearer structure than C++. It provides a high level of safety mechanisms (for example array bound enforcement) and reduces program complexity (for example through automatic garbage collection). It is also available for a wide variety of platforms, where it can be run immediately without the need for error-prone recompilation. On the other hand, Java is not well respected in the high performance community because it is perceived as significantly slower than C/C++. While this was true for the very first Java versions, the use of just in time compiler technology has significantly closed the gap. Indeed, current versions of Java outperform C in several benchmarks[21]. In any case, the current implementation is intended to explore general possibilities of implementation, and to compare the different strategies and choices for the different modules, and not to give maximum performance. Java was chosen as the best compromise for a programming language that is established, offers relatively high performance, and allows for easy and robust development.

In the following sections, we describe the various modules that make up the implementation and compare the different options for their realization. We will only explain the main ideas behind the code, limiting discussion of the details of implementation to the essentials. For details, we refer to the authors website[25], where all code and extensive documentation are available.

5.2 The core algorithm

In the algorithm 3.2.3, we specified no method by which to arrive at an estimate for the r_α. We have developed several mechanism for estimation, which are realized as

different classes. All of these classes implement the main interface

```
public interface Integrator<Evaluator> {
    IntegrationResult integrate(Evaluator integrand,
        StopCondition condition,
        List<Visualizer> visualizers)
      throws IntegrationFailedException;
    ...
}
```

We will describe the two major implementations of the this interface. The class `EstimateIntegrator` implements algorithm 4.4.1, supporting simplicial quotas between 0 (fully adaptive) and 1 (non-adaptive). It can use the minimum, maximum and geometric estimate introduced in section 4.2.1. The class `EvaluateIntegrator` implements a hybrid version of algorithm 4.2.1, and also supports simplicial quotas between 0 and 1.

The method `integrate` forms the main entry point for quadrature. The integrator classes all work the same way: They perform an open-ended quadrature until the condition specified by the given `StopCondition` class is reached. At this point, the method returns, giving an `IntegrationResult` return value. This class gives the result of quadrature, an error estimate and the number of times the function has been evaluated during quadrature. It may also give supplemental information about the quadrature process, for example issuing a warning that the function behaved erratically and that the results should therefore be treated with caution.

The third argument gives a list of classes that give visual feedback about the quadrature process. We will cover this part of the implementation in detail in section 5.5.

The most tricky part of the implementation concerns the first argument. What must be understood is that the algorithm as implemented never itself sees the function to be integrated. Indeed, such a function must not even exist. Instead of a function, the first argument specifies a more abstract `Evaluator`:

```
public interface Evaluator  {
    int dimension();
    double deltaEvaluate(Index index)
      throws IntegrationFailedException;
    boolean canEvaluate(Index index);
    int pointsForIndex(Index index);
}
```

We first cover the standard case, where the `Evaluator` directly acts on a function. In this case, the first method returns the function dimension. The second method takes a multi-index α (implemented as `Index`) and returns the value $\Delta_\alpha(f)$ for that index. The third method declares whether the `Evaluator` is able to evaluate the value for

the given index. Evaluation may not be possible, for example, if the one-dimensional quadrature formula used by the `Evaluator` only supports a limited number of nodes. If evaluation is possible, the last method returns the number of function evaluations need to perform evaluation and compute the value $\Delta_\alpha(f)$ for the given index.

This, as stated, is the standard case. As we can see, the `Evaluator` makes no explicit mention of a function anywhere. It is a black box that only returns a dimension, some value for each index, and information on whether an index can be evaluated and how costly this is. No restrictions are made on where this information comes from. We have implemented non-standard evaluators for infinite-dimensional integrals and for virtual indices, where several indices are subsumed to one, reducing the size of the index set. As of the writing of this thesis, both of these components are still in early development, and are not covered further.

Another feature of algorithm 3.2.3 that has to be cast into more concrete terms is the stopping condition. We can easily come up with several such conditions that may be of use:

- Stop when the estimate for the error $\epsilon(N)$ falls below a given threshold
- Stop when the estimate for the relative error $\frac{\epsilon(N)}{|Q_N(f)|}$ falls below a given threshold
- Stop when the number of function evaluations N reaches a given threshold

All of these are available in the implementation. They are realized as classes implementing the interface `StopCondition`:

```
public interface StopCondition {
    boolean stop(IntegrationResult result);
    ...
}
```

The main method is `stop(IntegrationResult)`. It takes an `IntegrationResult` supplied by the main algorithm, which describes the current state of integration. Based on this information, the method returns `true` if the condition is satisfied and `false` otherwise. The class `MultipleStopCondition` allows the combination of several stopping conditions, returning the signal to stop as soon as one of the constituent conditions is satisfied.

Special care needs to be taken with the stopping conditions based on the error estimate. This is because the error estimate returned as part of the `IntegrationResult` may underlie fluctuations due to the sampling process. If we poll the error estimate continuously to see if it has fallen below a given threshold, these fluctuations bias us towards stopping too early. The reason for this one-sided bias is that stopping is a one-sided operation. If we stop, we don't continue to see if maybe the error estimate goes up again. If we don't stop, however, we simply continue, sampling the error estimate until it does fall below the threshold. We have tried to mitigate this decision

Name	Distribution	Hierarchy	Exact on	Nested
Trapezoidal	$\lambda^1([0,1])$	$N(0) = 1$ $N(l+1) = 2^{l-1} + 1$	n/a	yes
Gauss-Legendre	$\lambda^1([0,1])$	$N(l) = 2^{l+1} - 1$	$\mathcal{P}_{2N(l)-1}$	no
Clenshaw-Curtis	$\lambda^1([0,1])$	$N(l) = 2^{l+1} - 1$	$\mathcal{P}_{N(l)-1}$	yes
Patterson	$\lambda^1([0,1])$	$N(l) = 2^{l+1} - 1$	$\mathcal{P}_{\frac{3}{2}N(l)+\frac{1}{2}}$	yes
Gauss-Hermite	$N(0,1)$	$N(l) = 2^{l+1} - 1$	$\mathcal{P}_{2N(l)-1}$	no

Table 5.1: Properties of the different quadrature rules

bias at least somewhat by only sampling the error estimate at increasing intervals. In the implementation, we have chosen the following strategy: if we sample the error rate at $N = M$, we wait until $N > \frac{3}{2}M$ until sampling again.

5.3 The quadrature formulas

The implementation supports a wide range of quadrature rules. For $\Omega = [0,1]$ and $\mu = \lambda^1([0,1])$, we have trapezoidal rules, Gauss-Legendre rules, Clenshaw-Curtis[7] rules and Patterson rules (a.k.a. Kronrod-Legendre rules)[31]. The Patterson rules are designed to attain a maximal level of polynomial exactness while retaining nodes from previous levels (i.e. being nested). Some of the rules given here originally refer to an interval different from $[0,1]$. In this case, they have been rescaled accordingly. The case of $\Omega = \mathbb{R}$ and $\mu = N(0,1)$, where $N(0,1)$ denotes the Gaussian normal distribution, is supported by a rescaled version of the Gauss-Hermite rules. The properties of the quadrature rules as implemented are summarized in table 5.1. For each of these quadrature rules, we have a hierarchy of quadrature formulas U_l (compare section 3.2). The hierarchy column of the table gives the relationship between the level l and the number of nodes $N(l)$ for that level.

The quadrature formulas U_l are Dirac sums, and are modeled by the

```
public class QuadratureFormula {
    public int getSize();
    public double getNode(int index);
    public double getWeight(int index);
}
```

These method `getSize()` returns the number of nodes of the Dirac sum, and thus corresponds to $N(l)$ (again using the notation from section 3.2). The nodes and weights themselves are returned by the eponymous methods.

In this way, each quadrature rule can be implemented as a class that returns a `QuadratureFormula` for each level l. This idea is captured in the

```
public interface Generator {
    QuadratureFormula getByLevel(int level);
```

```
    int maxLevel();
    ....
}
```

Most of the generators produce the requested quadrature formula on the fly. This causes a brief delay the first time a quadrature formula is requested for a given level. Once it has been produced, however, the `QuadratureFormula` is stored in a cache and is available without any delay on subsequent request. In contrast to this, the Patterson quadrature rule works with pre-specified tables of values. In both cases, only levels up to the maximum supplied by the method `maxLevel()` are supported. This is either because the algorithm for quadrature formula generation is numerically stable only up to a certain level and number of nodes, or, in the case of the Patterson rules, because tabulated values are only available up to a maximum level.

The class `DeltaGenerator` is used to produce the Δ_l that are needed for sparse grid quadrature. It wraps around a `Generator` for a specific quadrature rule, producing the Δ_l from the U_l produced by this `Generator`. The `DeltaGenerator` class automatically recognizes when some nodes used in U_l are present again in U_{l+1}, as in the Patterson formula. In these cases, it fuses the weights from U_l and U_{l+1} to produce only one node for the combined weight, so that the total number of nodes and therefore $\mathrm{Eval}(\Delta_{l+1})$ is as small as possible.

5.4 The applet

The implementation contains a Java applet that serves as an interactive front end for the quadrature process. It allows the user to select from various function classes and to set the stopping conditions. It supports dimension-adaptive sparse grid quadrature, non-adaptive sparse grid quadrature, Monte Carlo quadrature and Quasi-Monte Carlo quadrature. The quadrature rules can be specified for both the adaptive and the non-adaptive sparse grid quadrature method. For adaptive sparse grid quadrature, the user can also select various refinement and error estimation strategies.

In figure 5.5.1, we see some of these settings in the window at the top center. Specifically, the adaptive sparse grid ("ASG") quadrature is used with the Patterson rules for the Genz oscillatory function (see chapter 6) with dimension 8 and difficulty 18.

5.5 Visualization

The implementation supports a wide range of modules for visualizing the quadrature process online. This has proved extremely helpful in the development of the algorithms, as the graphic display of relevant data makes it far easier to analyze and grasp intuitively why the algorithm is or is not working, and are more revealing than only benchmark data on convergence. In the same vein, changes to the algorithm can immediately be assessed, and parameters can be tuned interactively. A

screenshot of the implementation with several visualization modules active is shown in figure 5.5.1 on the next page.

The figure shows the results of a quadrature for a Genz oscillatory function with dimension 8 and difficulty 18. We used the fully adaptive approach with contribution estimation by geometric average, and performed quadrature up to $N = 10^4$ evaluations. We will give a short description of the components shown. For a detailed description of all the features, please refer to the online documentation and code at [25].

5.5.1 The Grid window

The grid window shows 2-dimensional slices through the index set. The top two sliders allow the user to select the dimensions i and j of interest. The window then displays all indices α with $\alpha_k = 0$ for $k \neq i, j$. The sizes of the contributions $\Delta_\alpha(f)$ are visualized in the left pane by color scale. A black dot signifies a positive contribution, otherwise we have $\Delta_\alpha(f) < 0$. The slider on the right adjusts the color scale. The slider on the bottom is used to retroactively analyze the quadrature process. It allows the user to select any M with $0 \leq M \leq N$, and show the state of quadrature at this point in the process. On the right pane, we see the contributions known to the algorithm at this point as full squares. The valid indices and the current estimates for the size of their contributions are shown as the smaller squares. A mouse-over function displays the numerical values of $\Delta_\alpha(f)$, its estimate and $\mathrm{Eval}\,(\Delta_\alpha(f))$ for the indices.

5.5.2 The Extent window

This visualization component display the maximum $|\alpha|_1$ encountered during quadrature as a red horizontal bar. The black vertical bars display the maximum α_i encountered for each dimension $i = 1, \ldots, d$. This allows the user to quickly appreciate how important each dimension was in the quadrature process.

5.5.3 The Result window

The result window displays the quadrature error $\epsilon(N)$ and several related statistics over the course of quadrature. Of course, the actual error $\epsilon(N)$ can only be shown if the correct value for the integral $I(f)$ is known. The main panel shows a bilogarithmic plot displaying the various statistics against the function calls N on the abscissa. The red and black curve shows the absolute value of the error $|\epsilon(N)|$; a black segment indicates that $\epsilon(N)$ is positive for this segment, and a red segment that it is negative. The pink curve displays the error estimate given by the quadrature algorithm with respect to N.

The error line starts at the first vertical grid line, which corresponds to $N = 1$. The next gridlines are for $N = 10$, $N = 100$, etc. The pink error curve only starts at $N \approx 300$, because for $N < 300$, the algorithm deemed the amount of information

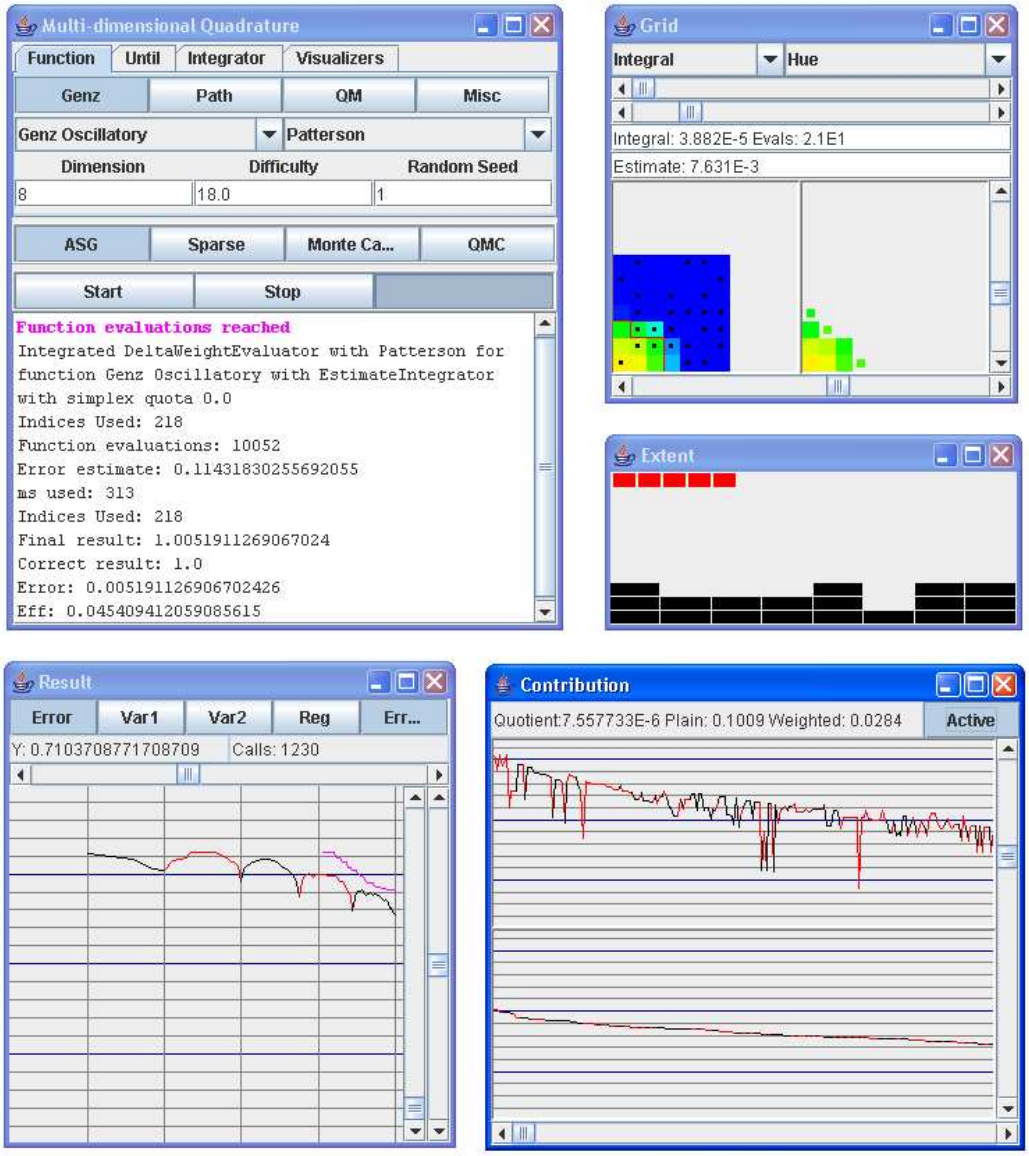


Figure 5.5.1: The application

accumulated too small to give a meaningful estimate. The different sliders allow the user to adjust the section of the plot that is shown. This window also features a mouse-over function that numerically displays the grid coordinates in the plot.

5.5.4 The Contribution window

The contribution window shows the actual values of r_α for evaluated and non-evaluated indices. The top pane shows r_α for those indices $\alpha_1, \alpha_2, \ldots, \alpha_k$ that were evaluated during quadrature. The sizesof the r_α is plotted logarithmically against the non-logarithmic abscissa i=1,...,k. Again, black codes for positive and red for negative contributions. The lower pane shows the values for r_β for those indices β that were not evaluated by the quadrature algorithm. These are sorted in order of decreasing r_β. The values r_β are computed in the background after the quadrature is finished in an open-ended process.

The top pane of the window can be used to assess if the algorithm actually did manage to trawl through the indices in order of decreasing r_α. Any indices β with large r_β that the algorithm missed will be visible in the lower pane. In an ideal world, the top panel would show a graph of monotonously decreasing r_α, and all r_β in the lower pane would be smaller than the rightmost (lowest) r_α of the top pane, meaning that the algorithm used the optimal set of indices.

5.6 Data structures and complexity

Throughout this thesis, we have used the evaluation complexity N as basis for describing the efficiency of our algorithms. In this section, we justify this choice for our main algorithm 3.2.3. This is not trivial. We need complicated data structures for managing the set of multi-indices, and we need to pay careful attention that these structures are efficient. Otherwise, we might spend more time managing ourselves than doing actual work[1].

5.6.1 Multi-indices

Multi-indices are modeled by the interface `Index`. They conform to the immutable pattern, that is, the multi-index α they represent cannot be modified once they have been instantiated. This allows the indices to be freely used and exchanged in the program without extra logic to ensure that they are not modified outside of their current scope.

The multi-index itself is coded in a sparse fashion, so that instead of all α_i, the index stores tuples (i, α_i) for only those i where $\alpha_i > 0$. Let $C(\alpha)$ denote the number of non-zero components for the index α, i.e. $C(\alpha) := \#\{i : \alpha_i > 0\}$. In this way, the amount of memory required to store an index is independent of the dimension of the problem and is instead proportional to $C(\alpha)$. Because $C(\alpha) \leq d$, the dimension d

[1] Although tempted, the author refrains from any obvious jokes about bureaucracy

forms an upper bound for $C(\alpha)$. Often, especially for high d, $C(\alpha)$ will be far smaller than d.

The tuples (i, α_i) are stored in order of increasing i. This allows us to perform a binary search on the tuples when a particular component is accessed, with logarithmic time in $C(\alpha)$.

5.6.2 The index set

The set of indices is stored by means of a hash table, which allows fast random access. This is important because the algorithm often needs to access the direct predecessors of a given index β, for example to calculate its expected contribution as per section 4.2.1, or to determine if all direct predecessors of β are already in the set, and therefore β is valid. The lookup time of a hash table is of the order $C(\beta)$.

5.6.3 Queues

An important part of the algorithm is finding an index α with a maximum value of r_α. To accomplish this efficiently, the valid indices are stored in a priority queue. A priority queue of the size M takes time on the order of $\log_2(M+1)$ for inserting elements and for removing a maximum element, which is far more efficient than the time of the order M that would be used for trawling through all elements to find an element with a maximum value.

5.6.4 The complete algorithm

We now give a step by step walk-through of the algorithm 3.2.3, listing all operations that have time dependent on d and N. For this section, we assume that $\mathrm{Eval}(\Delta_i) \geq 2^i$. This is the case for all quadrature rules in the implementation.

The following operations are performed for each index in the index set:

1. We evaluate $\Delta_\alpha(f)$ for the index. Because $\mathrm{Eval}(\Delta_i) \geq 2^i$, this corresponds to at least $2^{|\alpha|_1}$ evaluations of the d-dimensional function, giving a total time order of at least $d \cdot 2^{|\alpha|_1}$.

2. We add α to the index set. The time required is of the order $C(\alpha)$.

3. We figure out which new indices have become valid through the last step. For this, we check each of the forward neighbors $\alpha + \mathbf{e}_i$, $i = 1, \ldots, d$ if it has become valid. The index $\alpha + \mathbf{e}_i$ has become valid if all its backward neighbors are in the index set. Since an index has $C(\alpha + \mathbf{e}_i) \leq C(\alpha) + 1$ backward neighbors, and checking whether an index $\beta = \alpha + \mathbf{e}_i - \mathbf{e}_j$ is in the set requires $C(\beta)$ time with $C(\beta) \leq C(\alpha)$, the total time is of the order $d \cdot (C(\alpha) + 1)^2$.

Adding these terms, the time for the management operations is of the order

$$C(\alpha) + d \cdot (C(\alpha) + 1)^2$$

On the other hand, the time for function evaluations is at least

$$d \cdot 2^{|\alpha|_1}$$

Because $C(\alpha) \leq |\alpha|_1$, this shows that management time is at most of the same order as the time for function evaluations.

We need also examine the operations related to the index queue. Here, the situation is more complicated. We first examine the simplicial case. For this, let β_i, $i \in \mathbb{N}$ be a list of all indices in order of increasing length $|\beta_i|_1$. We give the following proposition.

Proposition 5.6.1 *Let $S \in \mathbb{N}$, and let $A = \{\beta_i : i \leq S\}$. Let $N(S)$ be the corresponding number of function evaluations. Then we have*

$$\sum_{k=1}^{S} \log_2(k+1) \leq \max(3,d) \cdot N(S)$$

Proof. Let $a = \max(3,d)$. We use the fact that

$$a \cdot N(S) = a \cdot \sum_{k=1}^{S} \mathrm{Eval}(\Delta_{\beta_k}) \geq \sum_{k=1}^{S} a \cdot 2^{|\beta_k|_1}$$

and perform a componentwise comparison of the two sums

$$\sum_{k=1}^{S} \log_2(k+1) \quad \text{and} \quad \sum_{k=1}^{S} a \cdot 2^{|\beta_k|_1}$$

For a given k, let

$$l = \max\{r : \#S_r \leq k\}$$

be the level of the largest simplex with less or equal than k elements. Because the indices β_i are arranged in simplicial order, this means that $|\beta_k| \geq l$ and therefore

$$2^l \leq 2^{|\beta_k|_1} \tag{5.6.1}$$

We have

$$k < \#S_{l+1} = \begin{pmatrix} d+l+1 \\ l+1 \end{pmatrix} \leq (d+l+1)^{l+1}$$

and therefore

$$\log_2(k+1) \leq (l+1)\log_2(d+l+1) \tag{5.6.2}$$

We now want to prove that for $l \geq 1$

$$(l+1)\log_2(d+l+1) \leq a \cdot 2^l \tag{5.6.3}$$

By definition of a, this holds for $l = 1$. It also holds for $l > 1$ because

$$\begin{aligned} & \frac{\partial}{\partial l}\Big(a \cdot 2^l - (l+1)\log_2(d+l+1)\Big) \\ = \;& a \cdot (\log 2) \cdot 2^l - \log_2(d+l+1) + \frac{l+1}{d+l+1} \\ \geq \;& \frac{2}{3} d \cdot 2^l - \log_2(d+l+1) + 1 \\ > \;& 0 \end{aligned}$$

Combining equation 5.6.3 with equations 5.6.1 and 5.6.2 yields

$$\log_2(k+1) \leq a \cdot 2^{|\beta_k|_1}$$

which completes the proof. □

Corollary 5.6.2 *Let A be an arbitrary index set, and $S = \#A$. Let N the number of function evaluations. Then*

$$\sum_{i=1}^{S} \log_2(k+1) \leq \max(3, d) \cdot N$$

Proof. We need simply arrange the elements $\alpha_1, \ldots, \alpha_s$ in order of increasing length. Because the $\beta_1, \ldots, \beta_s$ have minimal length, we have $2^{|\alpha_i|_1} \geq 2^{|\beta_i|_1}$ for $i = 1, \ldots, S$ and therefore $N \geq N(S)$. □

For algorithm 4.2.1, the indices in the queue are always a subset of the index set A. At the time that we evaluate index α_k, there can therefore be at most k indices in the queue. A queue operation then needs time of the order $\log_2(k+1)$. For each index, we have one insertion and at most one removal. Therefore the total time for queue operations is

$$2 \cdot \sum_{k=1}^{S} \log_2(k+1)$$

which by the corollary is of the same order as the time $d \cdot N$ used for evaluating the function.

The situation is less favorable for algorithm 3.2.3. Here, for each index in the index set, up to d forward neighbors may be placed in the queue along with an estimate for their contribution without any time being spent for evaluation of the contribution of these indices. Indeed, we can construct index sets for which the amount of time for queue operations is larger by the order of d than the amount of time spent on function evaluations. In this respect, algorithm 4.2.1 is preferable to algorithm 3.2.3 for large dimensions d, as the latter may then spend the larger amount of time with management instead of actual evaluation work.

6 The Genz Test Suite

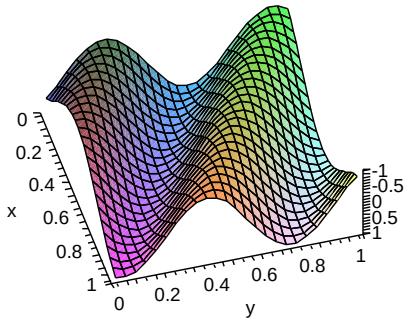
6.1 Introduction

The theoretical results available for the convergence of a specific quadrature method only give some indication to its utility. For one, theoretical results are usually only available for specific cases. In practice, we often observe that a quadrature method performs quite well for many functions for which no theoretical error bounds are known. Even where theoretical results are available, they generally give only upper bounds for a whole class of functions, and are valid for the worst case performance within this class, even though performance may be much better for the majority of functions. For this reason, a computer-based benchmark that gives real-world performance constitutes a valuable complement to the convergence results from theory.

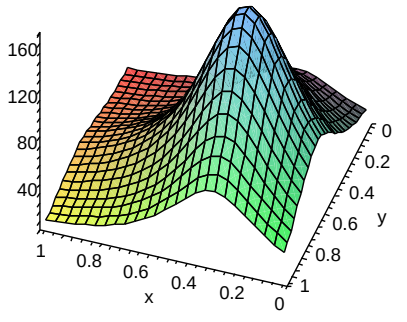
6.2 The Genz test functions

Genz [14] proposed a test suite for benchmarking multi-dimensional quadrature methods in 1984. The suite is composed of six different functions classes, each representing an aspect or problem typical for multi-dimensional integration. All function classes have been designed to be used on the multi-dimensional unit cube $[0,1]^d$ with the measure $\lambda^d\left([0,1]^d\right)$:

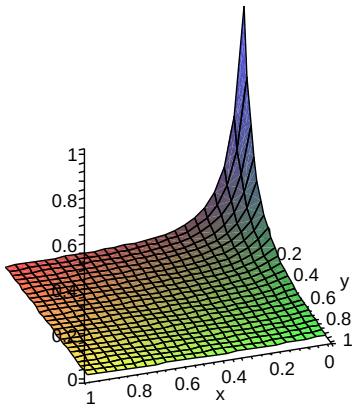
$$
\begin{aligned}
f_1(x) &:= \cos\left(2\pi u_1 + \sum_{i=1}^{d} a_i x_i\right) \\
f_2(x) &:= \prod_{i=1}^{d} \left(a_i^{-2} + (x_i - u_i)^2\right)^{-1} \\
f_3(x) &:= \left(1 + \sum_{i=1}^{d} a_i x_i\right)^{-(d+1)} \\
f_4(x) &:= \exp\left(-\sum_{i=1}^{d} a_i^2 (x_i - u_i)^2\right) \\
f_5(x) &:= \exp\left(-\sum_{i=1}^{d} a_i |x_i - u_i|\right) \\
f_6(x) &:= \begin{cases} 0 & \text{for } x_1 > u_1 \text{ or } x_2 > u_2 \\ \exp\left(\sum_{i=1}^{d} a_i x_i\right) & \textit{otherwise} \end{cases}
\end{aligned}
$$



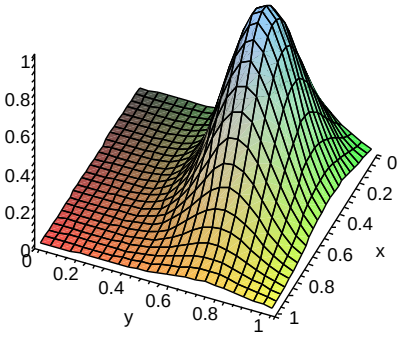
Oscillatory



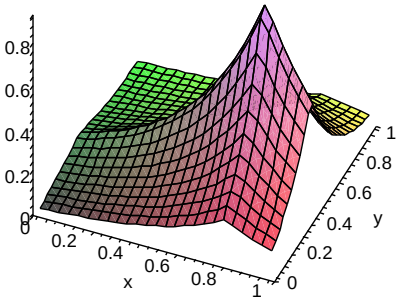
Product Peak



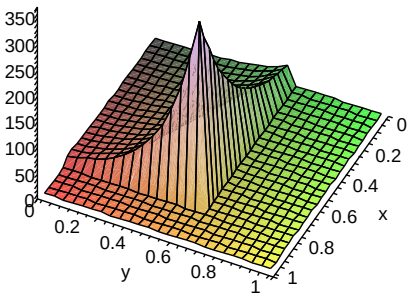
Corner Peak



Gaussian



Continuous



Discontinuous

Figure 6.2.1: The Genz test functions

These functions are given the names "oscillatory", "product peak", "corner peak", "Gaussian", "continuous" and "discontinuous", respectively, reflecting their salient characteristics. Illustrations of these functions for $d = 2$ are given in figure 6.2.1 on the facing page.

Each function is dependent on a set of parameters a_i and u_i. The parameters $a_1, \ldots, a_d$ can take values in $\mathbb{R}_{>0}$ and reflect the difficulty of quadrature, for example determining how quickly the function oscillates or how sharp the peak is. The parameters $u_1, \ldots, u_d$ take values in $[0, 1]$ and are more or less independent of the difficulty, instead shifting the function in space. For example, they determine the position of the wave crests for f_1 or of the discontinuity for f_6. In this way, each function class has many different representatives. We can therefore run our benchmark with a large number of functions for each class, and give statistical comparisons for the different function classes.

The Genz functions were designed in such a way that their integrals can be easily determined analytically. We have

$$
\begin{aligned}
I(f_1) &= 2^d \cos\left(\frac{1}{2}\left(4\pi u_1 + \sum_{i=1}^{d} a_i\right)\prod_{i=1}^{d}\sin\left(\frac{a_i}{2}\right)\right)\left(\prod_{i=1}^{d} a_i\right)^{-1} \\
I(f_2) &= \prod_{i=1}^{d} a_i \left(\arctan\left(a_i\left(1-u_i\right)\right) + \arctan\left(a_i u_i\right)\right) \\
I(f_3) &= \left(d!\prod_{i=1}^{d} a_i\right)^{-1} \sum_{\alpha\in\{0,1\}^d} \frac{(-1)^{|\alpha|_1}}{1+\sum_{i=1}^{d}\alpha_i a_i} \\
I(f_4) &= \prod_{i=1}^{d} \frac{\sqrt{\pi}}{2a_i}\left(\operatorname{erf}\left(a_i u_i\right) - \operatorname{erf}\left(a_i\left(u_i-1\right)\right)\right) \\
I(f_5) &= \prod_{i=1}^{d} a_i^{-1}\left(2 - e^{-a_i u_i} - e^{-a_i(1-u_i)}\right) \\
I(f_6) &= \prod_{i=1}^{\min(2,d)} \frac{e^{a_i u_i}-1}{a_i}\prod_{i=3}^{d}\frac{e^{a_i}-1}{a_i}
\end{aligned}
$$

where $\operatorname{erf}(x) := \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$ is the error function.

When generating test functions, we choose the a_i and u_i pseudo-randomly with uniform distribution on $[0, 1]$. The a_i are then rescaled to $a_i' := \frac{h_k \cdot a_i}{\|a\|_1}$ for a given number h_k for each function class. This h_k then reflects the difficulty level of the function f_k that is generated. The difficulty levels h_k, $k = 1, \ldots, 6$ must be determined in advance for each dimension d examined. They should be chosen in such a way that the different functions classes have comparable difficulty in some sense.

6.3 Finding good parameters for the algorithm

In the last chapter we have seen that the implementation of the dimension-adaptive sparse grid quadrature algorithm is composed of many different modules, each of which has different realizations. In this section, we will use the Genz test suite as a benchmark to select which combination of realizations for these modules works best. Because of the large set of parameters, we do not attempt an optimization on the full parameter space. Rather, we only modify one parameter at a time, holding the other parameters fixed, and hoping that the optimum found then is also the optimum in other cases. Indeed, the modules are to a large part independent of one another, and the results rather clear, so that this assumption gains credibility.

We have performed the benchmarks in dimension $d = 8$, using the difficulties

$$\begin{aligned} h_1 &= 9 \\ h_2 &= 19 \\ h_3 &= 2.1 \\ h_4 &= 12 \\ h_5 &= 15 \\ h_6 &= 2.9 \end{aligned}$$

We used the Monte Carlo method as a baseline for determining these values. Specifically, we chose h_i so that for $N = 10^4$ function evaluations, the relative error of quadrature for the Monte Carlo method was about 10^{-2}. We have calculated the results of quadrature $Q_N(f)$ for $N = 100, 200, 400, \ldots, 102400$. In some cases, the actual values are slightly higher due to the fact that calculating the contribution for an index is an atomic operation for the algorithm. For each benchmark, we evaluated 100 randomly generated functions from each class.

6.3.1 The choice of index refinement strategy

We compared the following index refinement strategies:

Algorithm	Descriptor
Algorithm 4.2.1	Evaluate
Algorithm 3.2.3 with minimum estimator	Estimate Min
Algorithm 3.2.3 with maximum estimator	Estimate Max
Algorithm 3.2.3 with geometric estimator	Estimate Geom

In each case, we use a hybrid version of the algorithm with a simplicial quota of 0.5, and the Gauss-Legendre quadrature rules. The results are shown in figure 6.3.1 on the next page. Here and in all subsequent figures of this type, we plot the average number of correct digits (that is $-\log_{10} \frac{\epsilon(N)}{|I(f)|}$) against $\log_{10} N$. We see the differences between the various refinement strategies are not very large, showing that the general

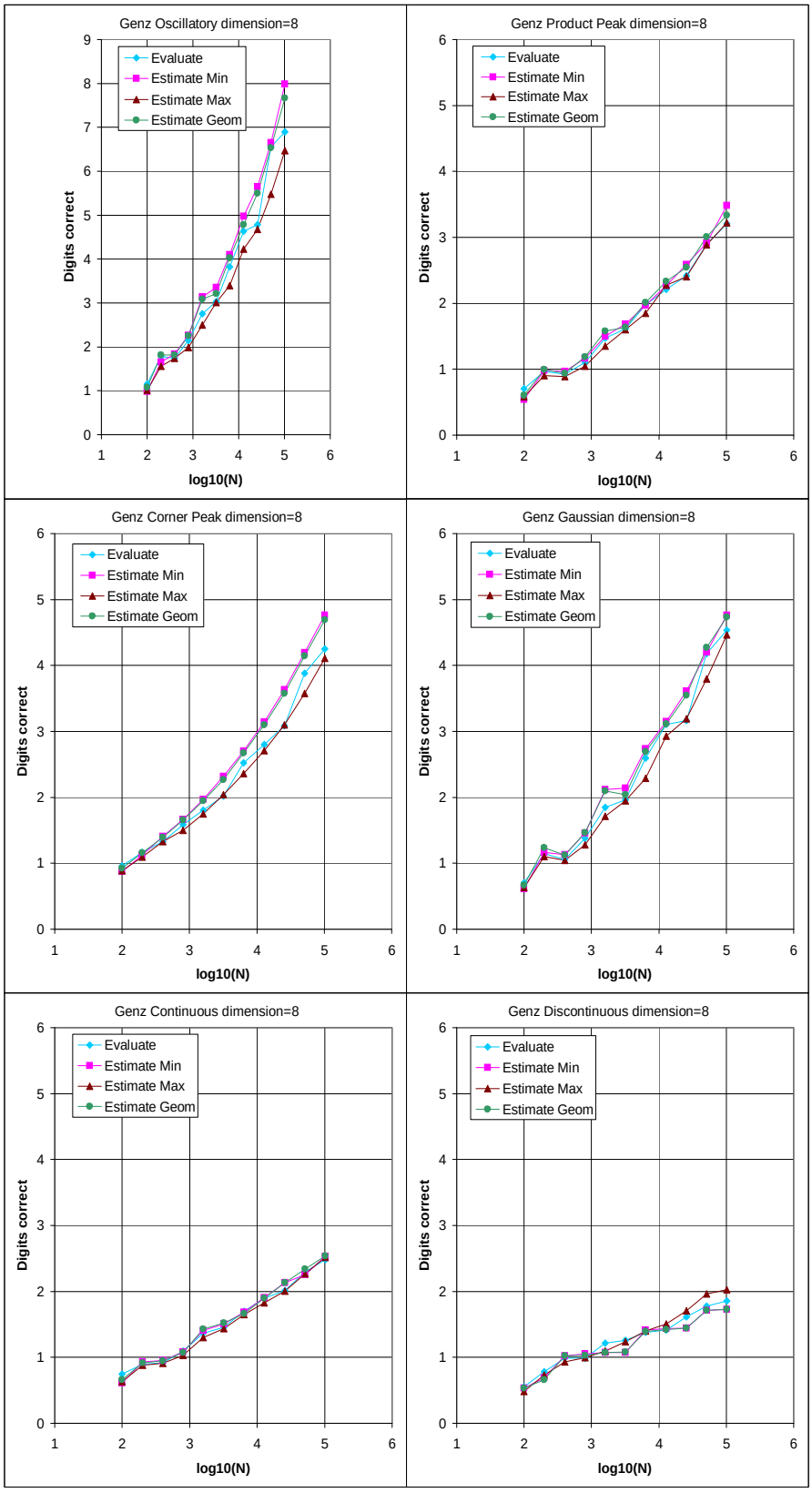


Figure 6.3.1: Comparison of different refinement strategies with fixed quadrature rule (Gauss-Legendre) and simplicial quota (0.5)

strategy of the algorithm is only influenced little by the specific form of the estimate. Of the all strategies, "Estimate Min" seems to perform best overall by a little bit, so we have chosen it for further testing.

6.3.2 The choice of quadrature rules

Taking the "Estimate Min" algorithm with a simplicial ratio of 0.5, we now compare different quadrature rules:

Quadrature Rule	Comment
Patterson	Nested rule with optimal degree of polynomial exactness (of the order $\frac{3}{2}N$)
Clenshaw-Curtis	Nested rule with suboptimal polynomial exactness (of the order N)
Gauss-Legendre	Non-nested rule with optimal degree of polynomial exactness (of the order $2N$)
Trapezoidal	Nested rule for piecewise linear interpolation

The results are shown in 6.3.2 on the facing page. We see that that the Patterson rules perform best, closely followed by Gauss-Legendre. It seems that the higher polynomial exactness afforded by Gauss-Legendre loses out to the fact that it requires more evaluations because it is not nested. The Clenshaw-Curtis rules are structurally inferior to the Patterson rules, and perform noticeably worse. The trapezoidal rules, which do not use differentiable structure much, unsurprisingly come in last.

There is however a surprise: Looking at the results for the Continuous class, we see that the polynomial interpolation rules perform quite a bit better than the trapezoidal rule. This is unexpected, because in the Continuous case, no derivative exists, so there is no theoretical reason to expect polynomial interpolation to perform better than trapezoidal quadrature. Indeed, looking at the results of the Discontinuous class, this is exactly what we see. There is no smoothness to be taken advantage of, and all quadrature formulas perform equally.

6.3.3 The choice of the simplicial ratio

So far, we have established the "Estimate Min" algorithm with a simplicial ration of 0.5 and the Patterson rules to be the best choice for our particular Genz benchmark. We now vary the simplicial ratio, to see in what way the adaptive algorithm performs better than simplicial quadrature, and whether the combination between the two does indeed exhibit synergistic effects, as speculated in section 4.4. We test the following simplex ratios:

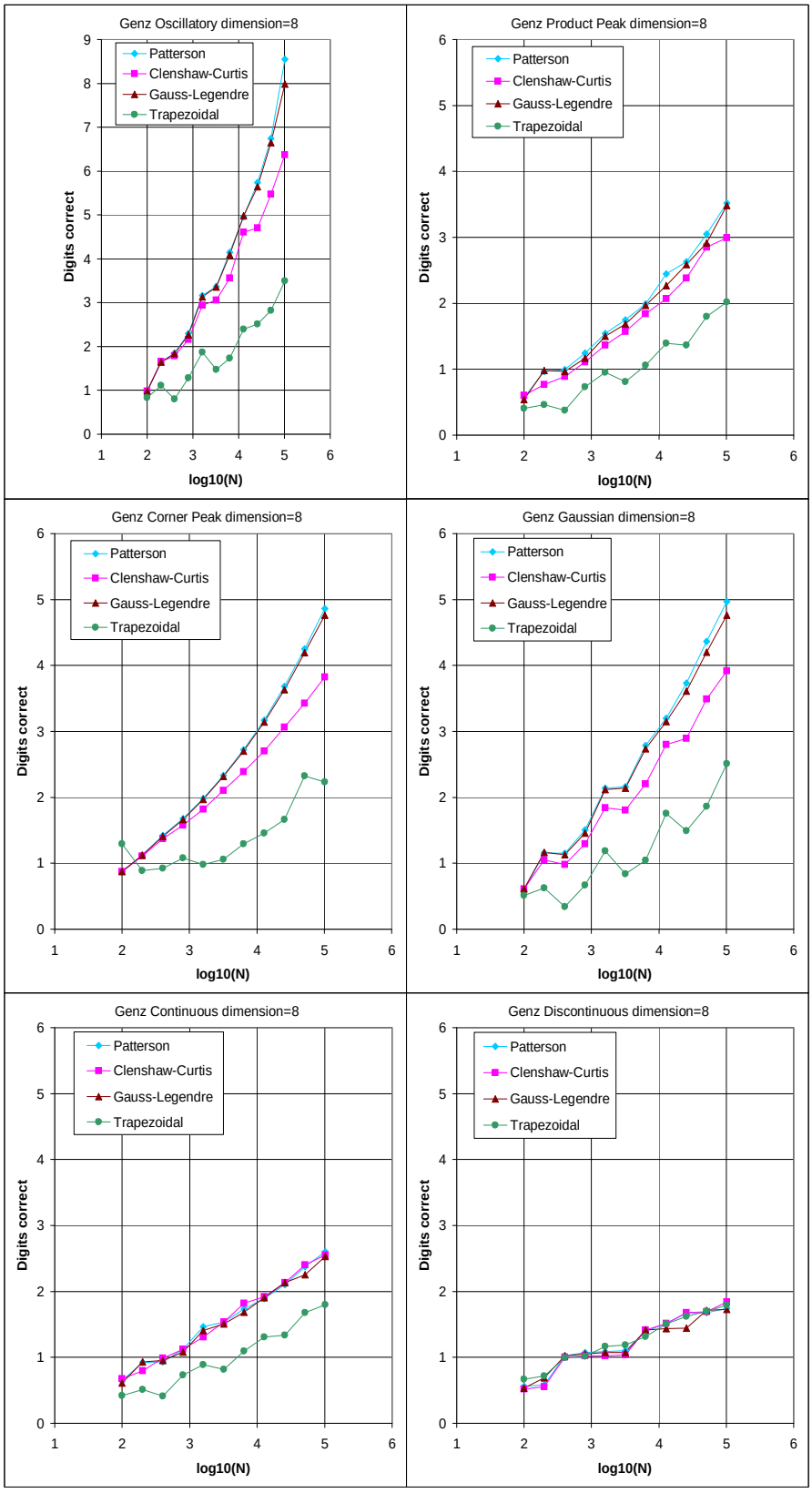


Figure 6.3.2: Comparison of different quadrature rules with fixed refinement strategy (Estimate Min) and simplicial quota (0.5)

Simplicial ratio	Comment
0.0	Fully adaptive
0.1	Adaptive with small non-adaptive component
0.5	Half-half hybrid
1.0	Classical non-adaptive quadrature

The results are shown in 6.3.3 on the next page. We see that for the smooth Genz functions, full adaptivity works best, and additional simplicial components simply slow down the algorithm in finding the best indices. The situation is quite different for the Discontinuous class. Here, the adaptive algorithm fails completely to identify the best indices. However, neither does the fully non-adaptive algorithm perform best. Here we indeed have the case that the two components inform each other, and the half-half hybrid comes out on top. It seems that the less smooth a problem is, the more irregular is the index structure. In these cases, the adaptive method goes awry, and should be supported by a strong non-adaptive component.

In conclusion, we choose the ratio of 0.5 for our algorithm. As seen in figure 6.3.3 on the facing page, the convergence is only slightly worse, but we gain a great deal of robustness.

6.4 Comparisons with the standard methods

6.4.1 d=8

We are now ready to compare the adaptive sparse grid method with the established multi-dimensional quadrature methods described in chapter 2. Specifically, we compare the following methods:

Method	Descriptor
Adaptive sparse grid, minimum estimator, 0.5 simplicial ratio, Patterson rule	Adaptive
Non-adaptive sparse grid with Patterson rule	Simplicial
Tensor-product quadrature with Gauss-Legendre rule	Product
Quasi-Monte Carlo quadrature with Halton sequence	QMC
Monte Carlo quadrature	MC

The "Product" quadrature was introduced in chapter 2 to be of the form $Q_M^{\otimes d}$. This would allow only the numbers M^d, $M = 1, 2, \ldots$ for the number of function evaluations. For $d = 8$, this corresponds to the integers $1, 256, 6561, 65536, \ldots$ which are spaced apart too far to fit in with our sequence $100, 200, 400, \ldots$ of evaluations. We have mitigated the problem by mixing formulas of the type Q_M and Q_{M-1} to give a product quadrature formulas of the type $Q_M^{\otimes k} \otimes Q_{M-1}^{\otimes (d-k)}$ for $k = 1, \ldots, d$, which allows a better approximation of the given values.

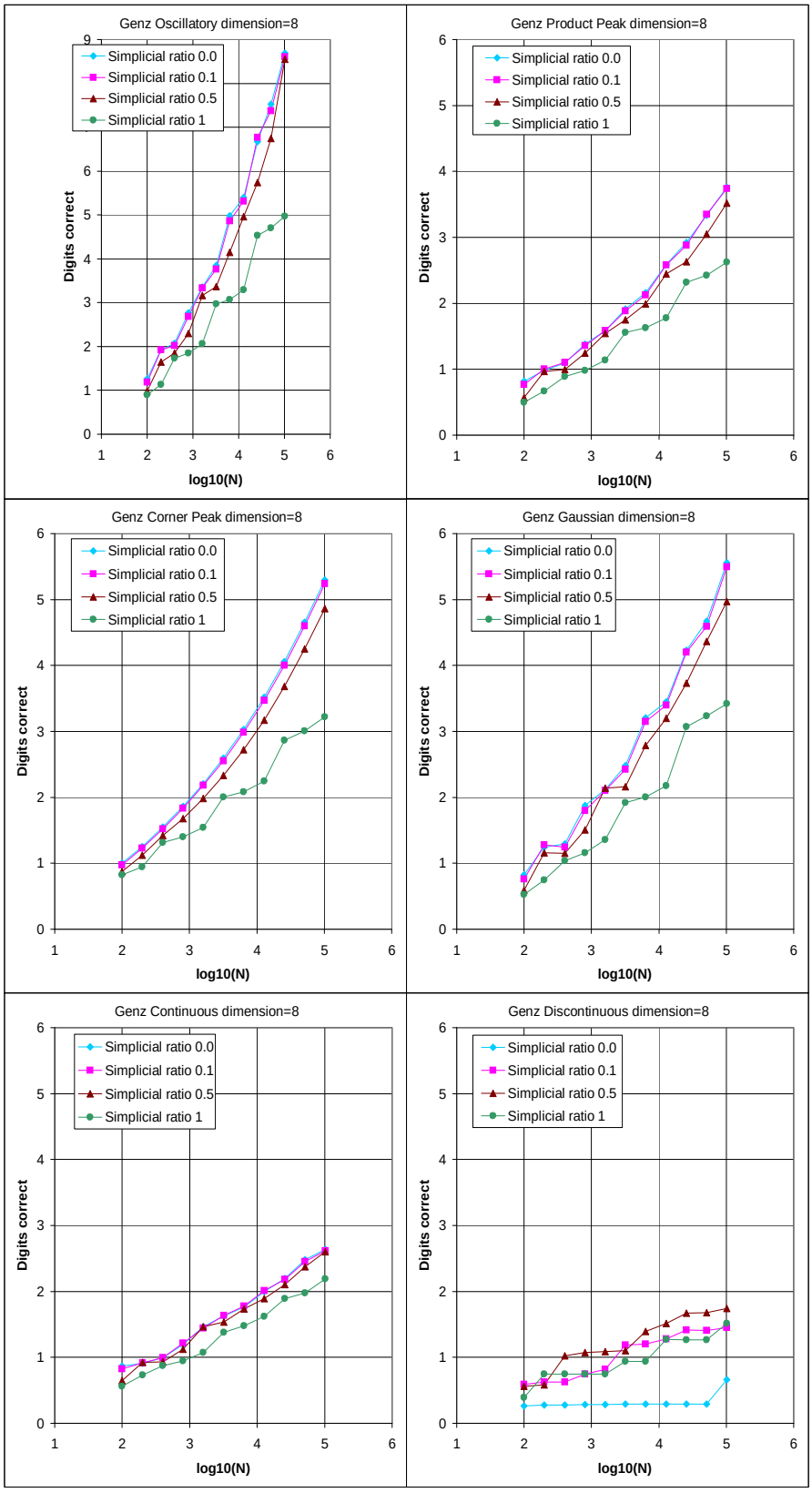


Figure 6.3.3: Comparison of different simplicial ratios with fixed refinement strategy (Estimate Min) and quadrature rule (Patterson)

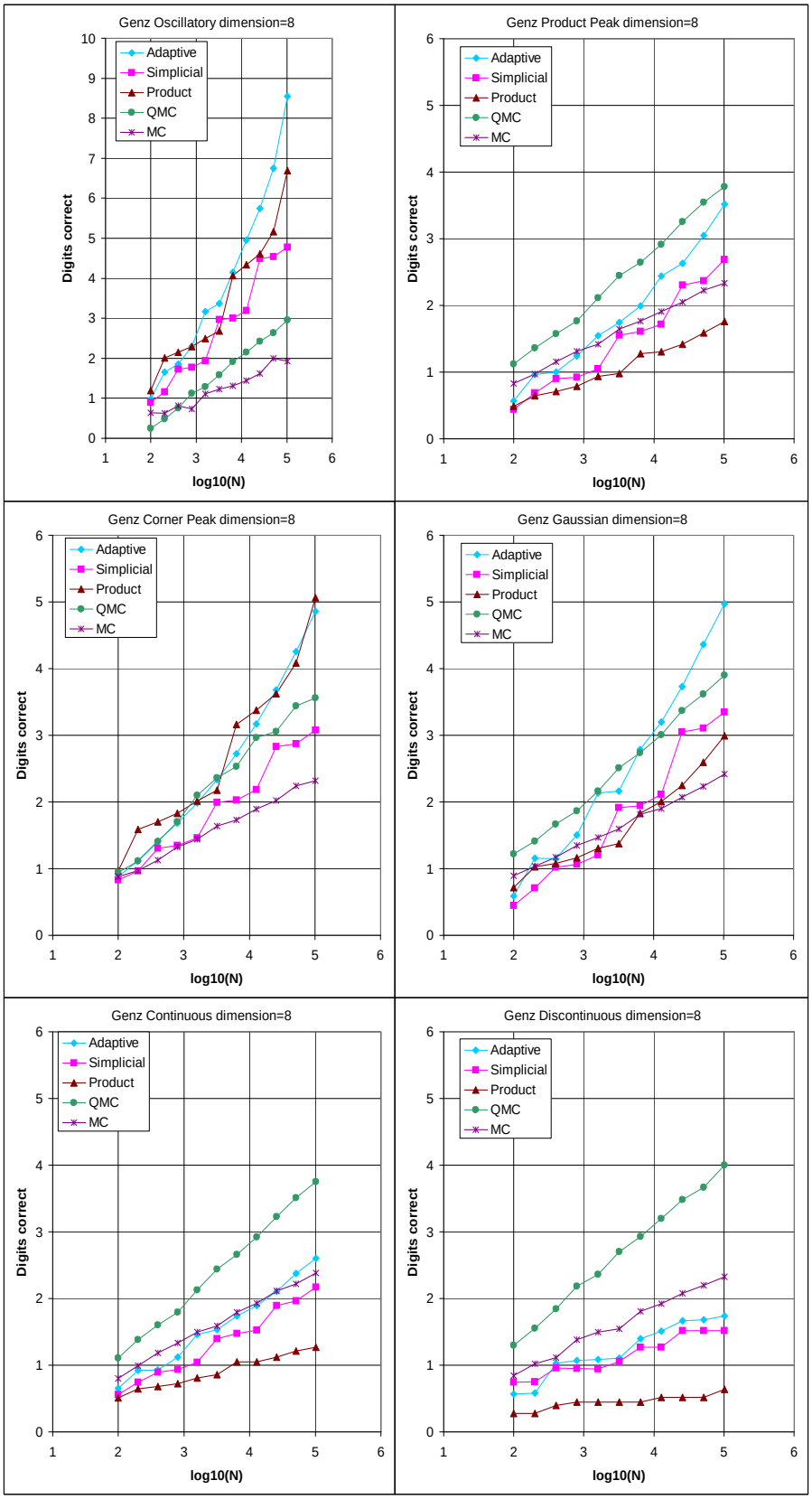


Figure 6.4.1: Comparison of the different quadrature methods for dimension 8

The results of the comparison are shown in 6.4.1 on the preceding page. We see that the "Adaptive" method does well for the smooth Genz functions, although it loses out to "QMC" for the Corner Peak. It should be noted that, in contrast to expectations based on logarithmic convergence rates (see 2.6) , the "Product" quadrature does not do too badly, although the "Adaptive" quadrature has overall better performance. For the Continuous and Discontinuous classes, we see that "QMC" is the method of choice. For these classes, interpolation methods cannot make use of any smoothness, and therefore show slow convergence.

6.4.2 d=4

To see to what extent the relative advantages of the algorithms depend on the dimension of the function, we have also performed benchmarks for the cases $d = 4$ and $d = 16$ (next section). The calibration of the difficulties for the Genz classes was performed as in section 6.3, yielding:

$$\begin{aligned} h_1 &= 8 \\ h_2 &= 18 \\ h_3 &= 3 \\ h_4 &= 10 \\ h_5 &= 17 \\ h_6 &= 3 \end{aligned}$$

The results are shown in figure 6.4.2 on the following page. We see that for dimension 4, standard tensor product quadrature performs well for the smooth Genz classes, overall being on par with the adaptive sparse grid approach. Simplicial sparse grids perform only a little worse than the adaptive version. For the Continuous and Discontinuous classes, again the sampling methods dominate. Note that due to the limitations of floating point arithmetic, an accuracy of about 14 digits is the highest attained.

6.4.3 d=16

In this case, the difficulties obtained from calibration were:

$$\begin{aligned} h_1 &= 8 \\ h_2 &= 20 \\ h_3 &= 1.5 \\ h_4 &= 16 \\ h_5 &= 26 \end{aligned}$$

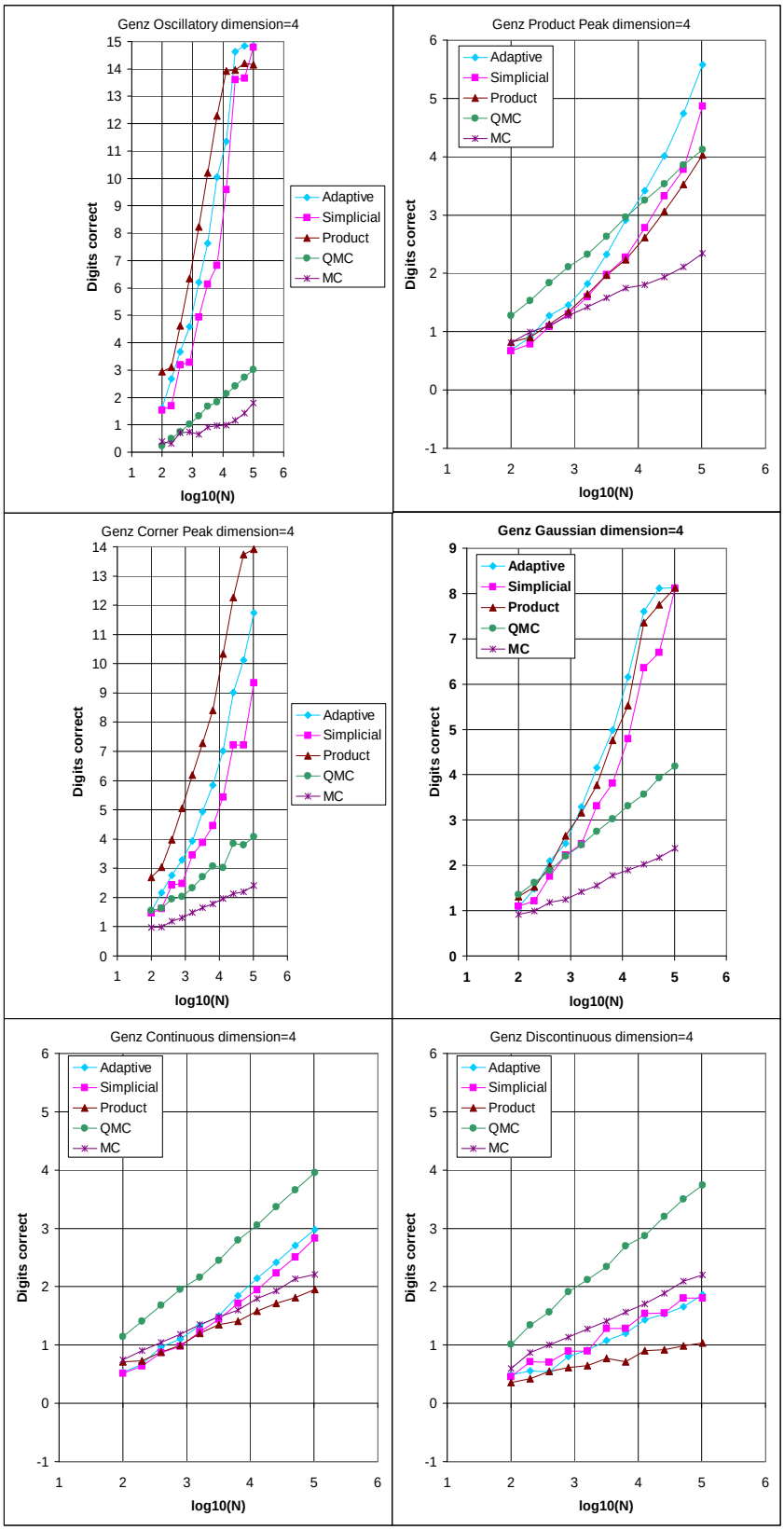


Figure 6.4.2: Comparison of the different quadrature methods for dimension 4

$$h_6 = 5$$

In order to reflect the higher difficulty of quadrature in high dimensions, we increased the maximum number of function evaluations, so that now $N = 100, 200, \ldots, 100 \cdot 2^{16} \approx 6.6 \cdot 10^6$.

The results are shown in figure 6.4.3 on the next page. We see that "QMC" performs best for the Product Peak and Gaussian classes. For the non-smooth classes, the sampling methods and "QMC" in particular win hands down. We see that for high dimension, the "Adaptive" method loses accuracy in relation to the sampling methods. For the Corner Peak and Oscillatory classes, it is still overall the best method. It performs very well for the Oscillatory class, which also judging by the previous results seems especially amenable to interpolation quadrature.

6.5 Dimension-adaptive vs. simplicial methods

From the results of the benchmark in figure 6.4.1 on page 72 we see that the dimension-adaptive sparse grid method always performs better than the simplicial non-adaptive method in terms of accuracy. In this section, we relate this observation to the structure of the Genz functions and examine the implication for the index set and index contributions. For each Genz class, we show a salient index grid as produced by the visualization component of the algorithm (see section 5.5).

These grids are shown in figure 6.5.1 on page 77. We have chosen dimension 2 to create the results. The structure of the 2-dimensional sections through grids for the higher dimensional representatives of each Genz class look similar, because a 2-dimensional sections in effect represents the grid of a 2-dimensional function due to the tensor product nature of the sparse grid. In each case, we delineate the indices selected by the adaptive method by a red line, and those chosen by the simplicial method by a dashed line. Quadrature was performed for $N = 300$ evaluations with the Patterson rule. To emphasize the difference between the methods, we have not used the hybrid method here, and compared the fully adaptive "Estimate Min" method to the simplicial sparse grid. We used the difficulties 8, 4, 4, 4, 4, 4.

As we can see in the graphs, there are two factors that contribute to the better convergence of the adaptive method. The first is that the dimensions itself may be of varying importance. In the case of the Genz functions, the difficulty of the dimension i increases with the size of the parameter a_i (compare section 6.2). Since these parameters are chosen randomly, some will be larger than others. For the Oscillatory and Product Peak functions, for example, the dimension in vertical direction is more important, whereas for the Corner Peak function, the horizontal dimension has higher contributions. The simplicial method is not able to take advantage of this fact, and treats all dimensions equally. The second factor concerns the shape of the index set. For the Product Peak class, the optimal index set is more square (convex) than the simplicial triangle, whereas for the Continuous class, it is more hollowed out (concave).

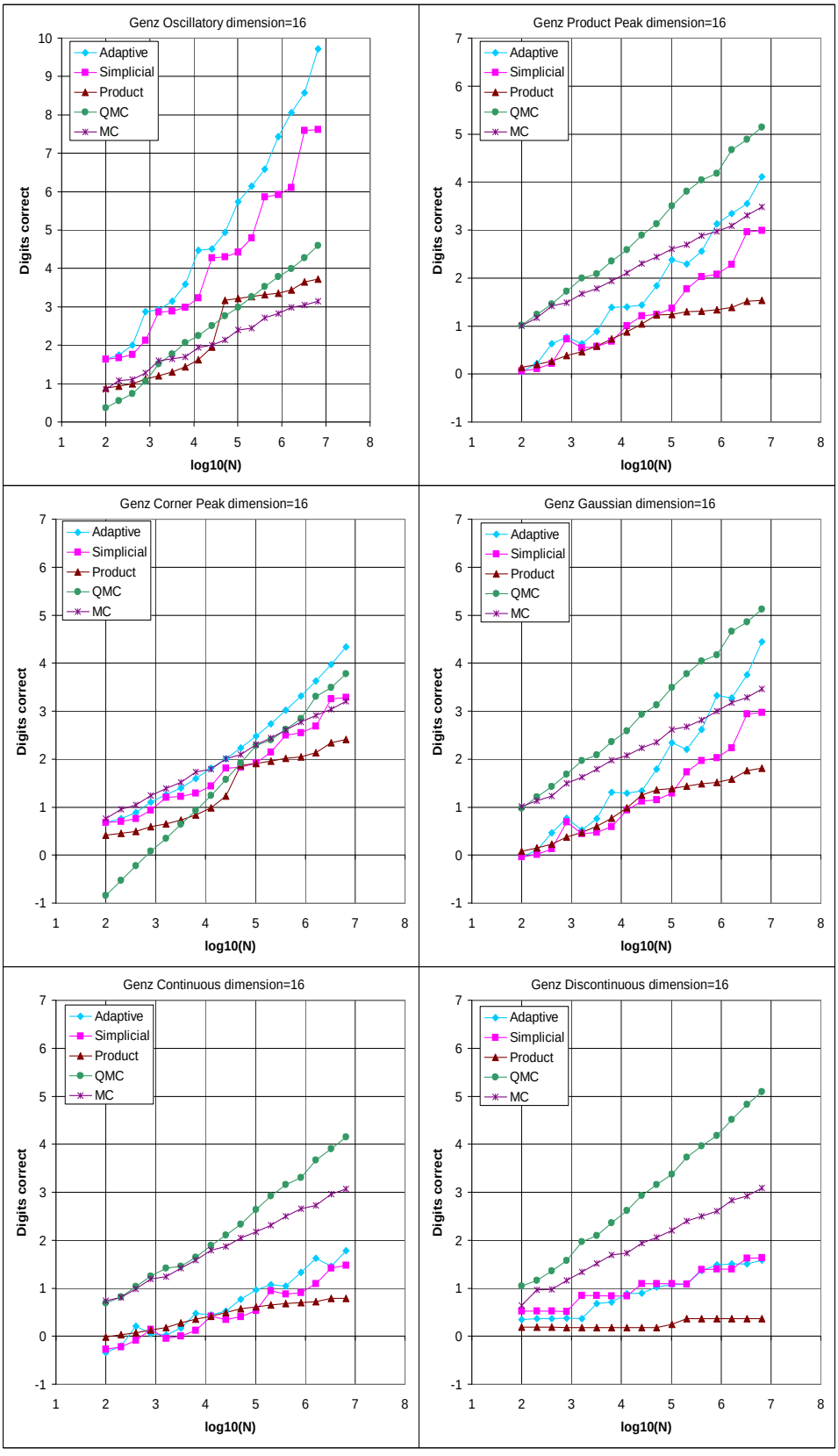


Figure 6.4.3: Comparison of the different quadrature methods for dimension 16

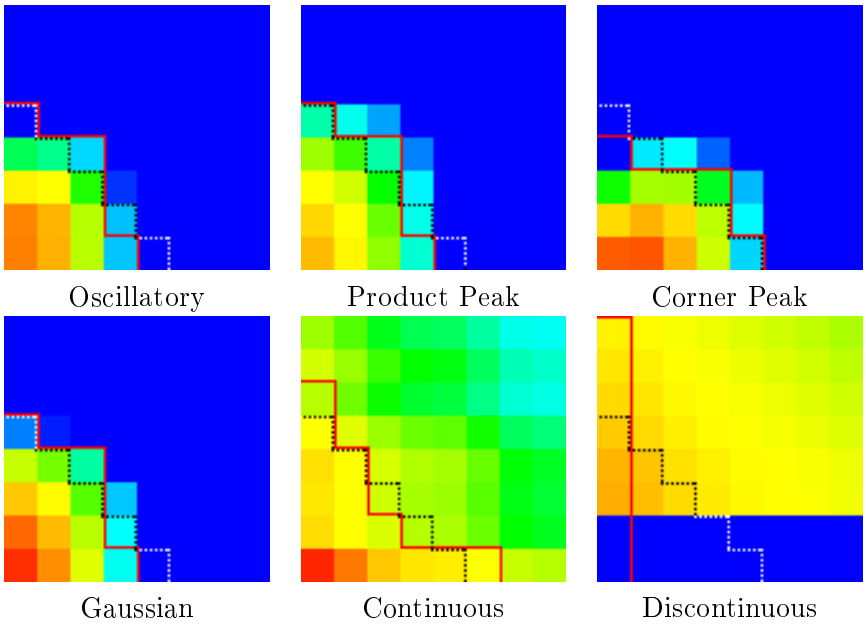


Figure 6.5.1: Typical index contributions for the Genz classes, calculated for $d = 2$. The red line and dashed black line delimit the contributions selected by the adaptive and the simplicial method for $N = 300$ points, respectively

For the most case, the differences between the adaptive and the simplicial index set appear small. However, the contributions of directly neighboring indices often differ by an order of magnitude, so missing only a few important indices can result in a large difference in accuracy, as evidenced by the graphs in the last section.

The picture for the Discontinuous function demonstrates nicely why the fully adaptive algorithm fails in this case. In the beginning, it encounters only contributions of 0. When it finally hits upon a non-zero contribution, it expands along this path, and never goes back to expand from the other indices with 0 contribution.

6.6 Error estimates

We conclude this chapter by examining the quality of the error estimate. In section 4.3, we had given two different estimation strategies, the contribution-based estimate and the black box estimate. We evaluated both methods for $d = 8$ and the "Estimate Min" hybrid algorithm with 0.5 simplicial ratio and the Patterson rule.

The results for the black box estimate are given in figure 6.6.1 on the following page. This error estimate only becomes available for $N \geq 10^3$ (compare section 5.5), so we start the abscissa there. The left graph shows the error reliability, that is, the proportion of test functions for which the actual error was less or equal to the given estimate, and the estimate was therefore valid. A value of 1 is optimal and means that for all 100 test functions of this class, the given estimate was valid. The right graph shows the error efficiency, that is, the ratio of the actual error to the error

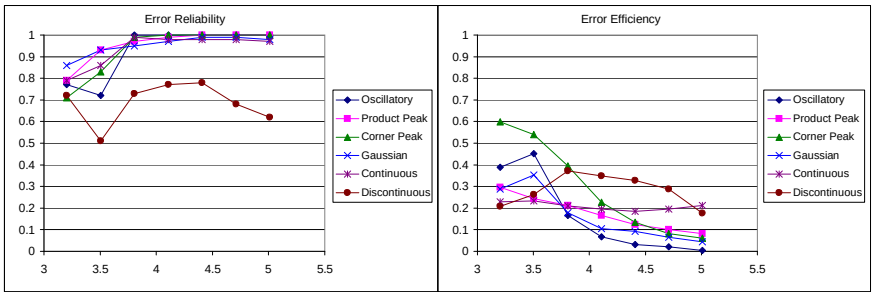


Figure 6.6.1: Black-box error estimate for the "Estimate Min" algorithm

estimate. For this ratio, we consider only those estimates that are valid. The reason for this is that otherwise, invalid estimates would spuriously lead to a seemingly better efficiency.

A value of r for the error efficiency means that the error was overestimated by the factor $\frac{1}{r}$. Note that while higher values for the error efficiency are better, we cannot hope for an optimum of 1, as this would mean that we would know the *exact* distance to the correct result. In fact, if we assume that the error is distributed randomly according to a Gaussian distribution with mean 0, and we require an error reliability of at least 99%, the optimal efficiency we can hope for is about 0.3. Indeed, taking a so that

$$N(0,1)([-a,a]) = 0.99$$

for the normal distribution N(0,1), we have

$$\frac{1}{0.99}\int_{-a}^{a}\frac{|x|}{a}dN(0,1)(x) = 0.3015\ldots$$

as the average efficiency for the valid error estimates.

We see that with increasing N, the black box estimator becomes more reliable, but also less efficient. Whereas there is always a trade off between reliability and efficiency, we would like this trade off to be independent of N, which the algorithm does not satisfy well.

This effect is even more pronounced for the contribution-based estimate, as seen in figure 6.6.2 on the next page. Here, the error estimate is very reliable, but the efficiency is extremely low. In the case of the "Estimate Geom" and "Estimate Max" algorithms, this effect is even stronger, because the higher individual estimates for the index contributions lead to a higher total error estimate (results not shown).

In both cases, the error estimator does not work well for the Discontinuous class, which is not surprising considering that both error estimators were based on assumptions of a regular contribution structure and convergence.

Although none of the error estimators is fully satisfactory, the black box estimator seems the better choice overall because of the extremely low efficiency of the

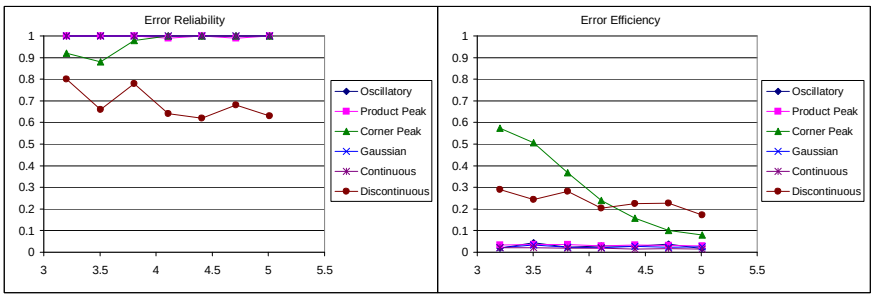


Figure 6.6.2: Contribution-based error estimate for the "Estimate Min" algorithm

contribution-based estimator.

6.7 Conclusion

Based on the Genz test suite for $d = 8$, we identify the following combination of parameters as the overall best choice:

- The hybrid algorithm 4.4.1 with the minimum estimator and a simplicial ratio of 0.5
- The Patterson quadrature rule
- The black box error estimator

For the Genz test suite, this algorithm performs very well for the smooth functions. In many cases, especially for the Oscillatory class, it gives results that are orders of magnitude better than those available with the sampling methods. These benefits are especially prominent for low and medium dimensions, and begin to diminish with higher dimensions. On the other hand, because Monte Carlo and Quasi-Monte Carlo have maximum convergence rates of $\frac{1}{2}$ and 1 respectively, they will usually not suffice if we need high accuracy for a given problem. In this case, it seems advisable to at least try out the adaptive algorithm.

The adaptive hybrid algorithm always performs better than the non-adaptive simplicial quadrature, and should therefore replace it for all applications where the additional memory demands of the index management allow it.

7 Path integrals for quantum mechanics

7.1 Introduction

The central tenet of quantum mechanics is the Schroedinger equation[27], a partial differential equation with values in $\mathbb{C}$ whose solutions represent possible wave functions. In this chapter, we examine the simple case of a single particle with mass m in one spatial dimension for a given time-independent potential $V(x)$. In this case, the Schroedinger equation for the wave function $\phi(t,x)$ of the particle is given by

$$i\hbar\frac{\partial}{\partial t}\phi(x,t) = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}\phi(x,t) + V(x)\phi(x,t)$$

For numerical computation, the time t is substituted by the negative imaginary time τ, i.e. $t = -i\tau$. Also, we choose units so that $\hbar = 1$, and set $m = 1$. In this way, we obtain the diffusion-type equation

$$-\frac{\partial}{\partial \tau}\psi(x,\tau) = -\frac{1}{2}\frac{\partial^2}{\partial x^2}\psi(x,\tau) + V(x)\psi(x,\tau) \tag{7.1.1}$$

The original solution ϕ is obtained by analytical continuation of ψ into the complex plane (corresponding to the original real time values).

Since this thesis deals with integration and not partial differential equations, we use the Feynman-Kac formula [12, 22]. Given a specified initial condition (i.e. $\psi(x,0) = u(x)$), the formula gives a solution of equation 7.1.1 by

$$\psi(x,\tau) = \int u(W_\tau)\cdot\exp\left(-\int_0^\tau V(W_s)ds\right)dP_x(W)$$

where W on the probability measure P_x is the Brownian motion starting in x at time 0 [19]. Of particular interest are the corresponding Green functions. Mathematically, they can be represented by taking Brownian paths with fixed start *and* end points. We use the notation $P_{x,\tau,y}$ to denote the measure for Brownian paths that start at x at time 0 and end in y at time τ. We then have

$$K(x,y,\tau) := \int \exp\left(-\int_0^\tau V(W_s)ds\right)dP_{x,\tau,y}(W) \tag{7.1.2}$$

In this notation, $y \to K(x,y,t)$ is the Green function for a given time t and starting point x. In physics, K is sometimes called the transfer matrix or transition matrix.

The Brownian paths W in the equations above are infinite-dimensional objects. To compute the elements $K(x, y, \tau)$ numerically, we use a discretized approximation of equation 7.1.2. Let M be the number of time steps, and $\tau_k = \frac{k}{M} \cdot \tau$ the discrete times. For W we define

$$\pi_M(W) = (W_{\tau_0}, \ldots, W_{\tau_M})$$

to be the projection of the process W to its values at the times $\tau_0, \ldots, \tau_M$. Approximating the integral

$$\int_0^\tau V(W_s)ds$$

by the trapezoidal sum

$$\sum_{i=0}^{M} w_{M,i} V\left(W_{\tau_i}\right)$$

with

$$w_{M,i} = \begin{cases} \frac{1}{2}\frac{\tau}{M} & \text{for } i = 0, M \\ \frac{\tau}{M} & \text{otherwise} \end{cases}$$

and using the transformation formula, we obtain

$$K(x, y, \tau) \approx \int u(\xi_M) \cdot \exp\left(-\sum_{k=0}^{M} w_{M,k} V(\xi_k)\right) d(\pi_M)_* P_{x,\tau,y}(\xi) \tag{7.1.3}$$

Remark 7.1.1 *It is of course also possible to use a different quadrature formula, for example the Gauss-Legendre rule, to approximate the integral $\int_0^\tau V(W_s)ds$. However, because the underlying Brownian motion is not differentiable, the same is generally true for the integrand $V(W_s)$, so we do not expect there to be any additional smoothness to be exploited. We obtained preliminary results using interpolatory formulas, which indeed did not show to any improvements in convergence.*

7.2 The discretized measure

The measure

$$\nu := (\pi_M)_* P_{x,\tau,y}$$

is a M-dimensional Gaussian measure given by

$$\mathrm{E}_\nu(\xi_k) = \mathrm{E}_{P_{x,\tau,y}}(W_{\tau_l}) = x + \frac{k}{M}(y - x)$$

$$\mathrm{Cov}_\nu(\xi_k, \xi_m) = \mathrm{Cov}_{P_{x,\tau,y}}(W_{\tau_k}, W_{\tau_m}) = \frac{\tau}{M}\left(\min(k, m) - \frac{km}{M}\right)$$

Gaussian measures of this kind can easily be constructed using the Brownian bridge method [6]. The idea of the Brownian Bridge is to begin with fixed start and end points, and then successively intercalate the remaining points. For any two points ξ_r and ξ_t at times $r < t$, and for a time s with $r < s < t$, the intercalated point ξ_s is

constructed by

$$\xi_s = \frac{(t-s)\xi_r + (s-r)\,\xi_t}{t-r} + \sqrt{\frac{(t-s)(s-r)}{t-r}} \cdot Z_s$$

for a random variable Z_s obeying the normal distribution. In this way, for any M we have a linear mapping

$$B_M : \mathbb{R}^{M-1} \to \mathbb{R}^{M+1}$$

that describes a construction of the measure ν from independent Gaussian random variables. We need $M-1$ of these variables because the start $\xi_0 = x$ and the end $\xi_M = y$ of the Brownian bridge are fixed, leaving $\xi_1, \ldots, \xi_{M-1}$ to be determined. Altogether, we have

$$(B_M)_* N(0,1)^{\otimes(M-1)} = \nu$$

where $N(0,1)$ is the Gaussian normal distribution. The function B_M can be implemented on a computer so that the time used is of the order M. The Brownian bridge construction thus offers a quick method to implement the desired measure ν. It is easiest to implement if M is a power of 2, i.e. $M = 2^k$ for $k \in \mathbb{N}$, because in this way we can simply choose $s = \frac{r+t}{2}$ for the intercalation times. We will limit ourselves to this case. Using the results above for equation 7.1.3 gives us the formula

$$K(x,y,\tau) \approx \int u(\xi_M) \cdot \exp\left(-\frac{\tau}{M}\sum_{k=1}^{M} V(\xi_k)\right) d(B_M)_* N(0,1)^{\otimes(M-1)}(\xi)$$

We have now reduced the problem of calculating $K(x,y,\tau)$ to an integral over the measure $N(0,1)^{\otimes(M-1)}$. In the notation of section 2.4, we have $\mu = N(0,1)$ and $d = M-1$. This immediately suggests using the Gauss-Hermite rule for quadrature. Alternatively, we can use the cumulative distribution function

$$F_{N(0,1)}(x) := N(0,1)(]-\infty, x])$$

to generate the measure $N(0,1)$ from $\lambda^1([0,1])$, using the fact that

$$(F_{N(0,1)})^* \lambda^1([0,1]) = N(0,1)$$

In summary, we can calculate the integral 7.1.3 as a $(M-1)$-dimensional integral, using either the measure $N(0,1)$ or the measure $\lambda^1([0,1])$ by applying the appropriate transformation.

7.3 The harmonic oscillator

The harmonic oscillator is a classical problem in quantum mechanics, and we have chosen it as an example to explore adaptive sparse grid integration for path integrals.

The harmonic oscillator is given by the potential

$$V(x) = \frac{\omega^2}{2}x^2$$

where ω is the oscillation frequency.

Of particular interest is the fact that analytical solutions are known for the Schroedinger equation itself as well as for the discretized problem, allowing comparisons against the true value and the exact time-discrete value. The analytical result for the Green function is

$$K(x,y,\tau) = \sqrt{\frac{\omega}{2\pi\sinh(\omega\tau)}} \cdot \exp\left(-\frac{\omega}{2\sinh(\omega\tau)}\left((x^2+y^2)\cosh(\omega\tau) - 2xy\right)\right)$$

We obtained the result of the discretized integral 7.1.3 by writing the measure ν as a weight on $\lambda^{M-1}(\mathbb{R}^{M-1})$. This leads to the Gauss-type integral

$$\left(\frac{mM}{2\pi\tau}\right)^{\frac{M}{2}} \int_{-\infty}^{\infty}\dots\int_{-\infty}^{\infty} \exp(-\frac{\omega^2}{2}\sum_{k=0}^{M} w_{M,k}\xi_k^2)\cdot\exp(\frac{m}{2}\frac{M}{\tau}\sum_{k=1}^{M}(\xi_k-\xi_{k-1})^2)d\xi_1\dots\xi_{M-1} \tag{7.3.1}$$

with $\xi_0 = x$ and $\xi_M = y$. By induction over M, we obtain a recursion formula that can be used to quickly calculate the correct value (see the code for details [25]). Alternatively, we can write the integral 7.3.1 in the form

$$\int\dots\int e^{\xi^T A\xi} d\xi_1\dots\xi_{M-1}$$

with a symmetrical matrix $A \in M(n+1, n+1)$, and solve the integral by diagonalization of the matrix.

We have evaluated the terms $K(x,y,\tau)$ for the times $\tau = 0.1$, $\tau = 1$ and $\tau = 10$. We used $M = 128$ time steps, because a plot of the correct discrete vs. continuous results suggested that for this number of time steps, we have a good trade off between the size of the discretization error and number of dimensions. Of course, for real problems we do not have analytical results to work with, and a suitable number of time steps must be determined from experience and by trial and error.

For each choice of τ, the average number of correct digits for 25 runs was computed. For each run, the parameters x and y were chosen with uniform probability from the interval $[-1, 1]$. The results of quadrature were compared to the value of the discrete integral 7.1.3 and to the correct value of the stochastic integral 7.1.2. The results are shown in figure 7.3.1 on the facing page.

We see that the discrete interval is solved best by the adaptive algorithm for $\tau = 0.1$ and $\tau = 1$, but better by the sampling methods for $\tau = 10$. For $\tau = 0.1$ and $\tau = 1$, the Gauss-Hermite rule performs noticeably better than the Patterson rule. We conjecture that the transformation from distribution $\lambda^1([0,1])$ to $N(0,1)$ via the function $F_{N(0,1)}$ makes the function less smooth and therefore slows down convergence. The Gauss-Hermite rules are naturally adapted to the Gaussian distribution,

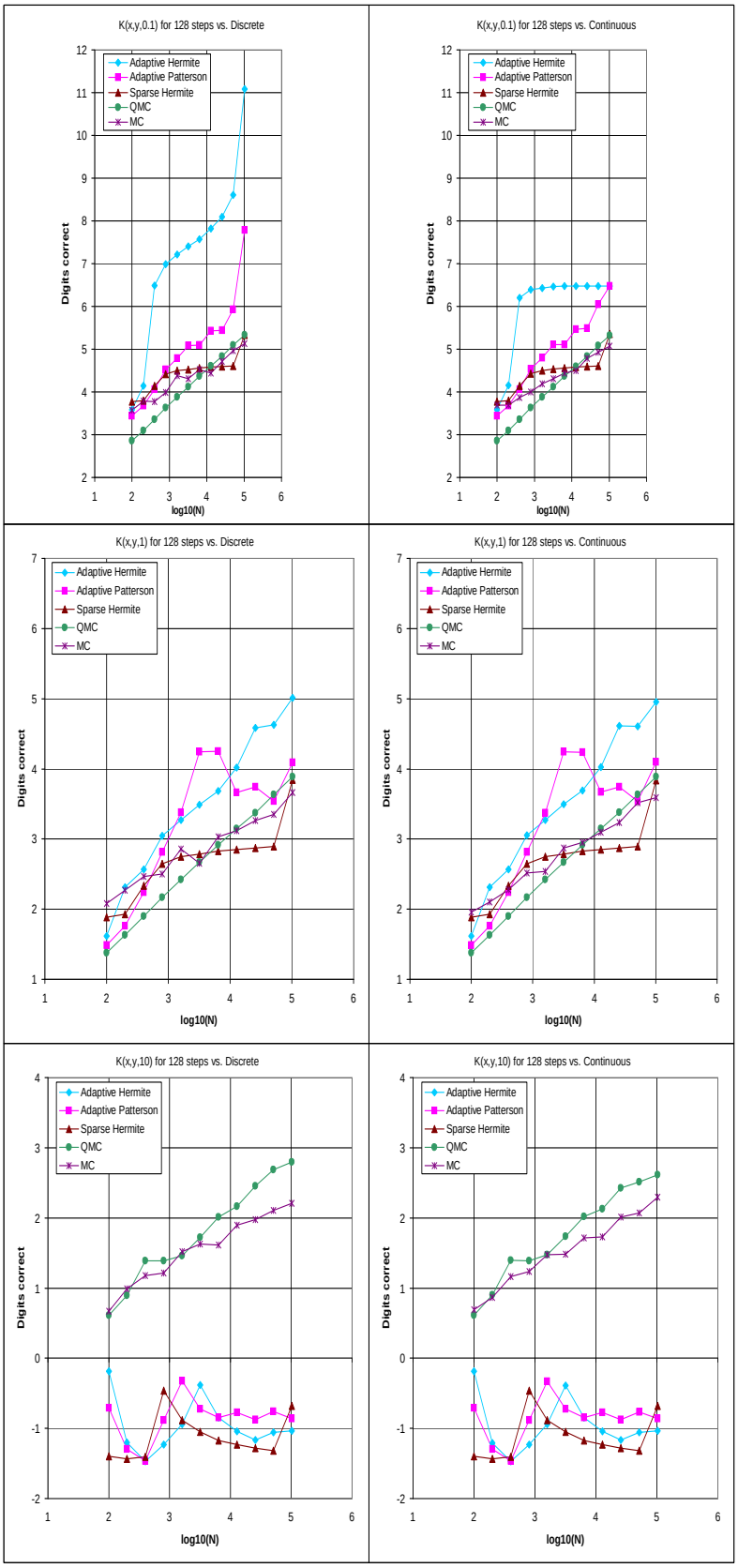


Figure 7.3.1: Quadrature results for the Green function K(x,y,t) vs. the correct discrete (left) and continuous (right) results for the harmonic oscillator

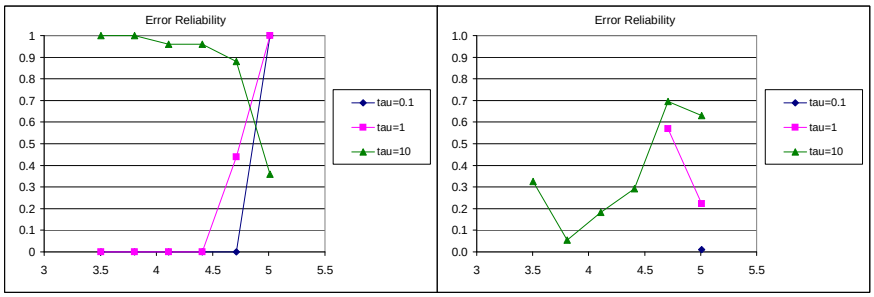


Figure 7.3.2: Error reliability and efficiency for the Adaptive Hermite algorithm for the harmonic oscillator vs. the correct discrete value

and better suited to the problem. The good performance of the Patterson rules for $\tau = 1$ between $N = 10^3$ and $N = 10^4$ is due to a pass through zero at this point, that occurs independently of the start and end points x and y (results not shown). We did not investigate this further.

In the right graphs, which plot the error against the continuous result, we see that for $\tau = 0.1$, some of the additional accuracy of the adaptive algorithms is spurious, and represents refinement past the error of discretization. Even with this taken into account, the adaptive methods give far better results than the sampling methods. The error of discretization does not come into play for the times $\tau = 1$ and $\tau = 10$.

The reliability and efficiency of the error estimator for quadrature against the correct time-discrete value are shown in 7.3.2. We see that the estimator fails completely in this case. In many instances, the reliability was 0, so the efficiency rating is not available. As we can see from the left graph for $\tau = 0.1$ in figure 7.3.1 on the preceding page, convergence seems very erratic for this path integral problem. This probably throws the black box estimator off course, since it depends on a steady convergence rate. We show the estimator only for the runs performed against the discrete value, as the algorithm has no mechanism for judging the result against the continuous value.

7.4 The anharmonic oscillator

As a second example, we consider the anharmonic oscillator, given by

$$V(x) = \frac{1}{2}\omega^2 x^2 + \lambda(x^2 - f^2)^2$$

We chose $\omega = 0$, $\lambda = 1$ and $f = \frac{1}{2}$ to obtain a two well potential with wells at $-\frac{1}{2}$ and $\frac{1}{2}$ (see figure 7.4.1 on the next page). Because of this, it is considered to be more difficult to solve computationally than the harmonic oscillator. Again we calculate the Green function $K(x, y, \tau)$ for x and y chosen uniformly from $[-1, 1]$

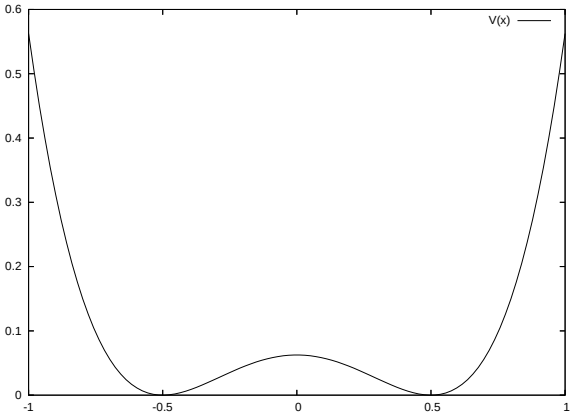


Figure 7.4.1: The anharmonic potential $V(x) = (x^2 - \frac{1}{4})^2$

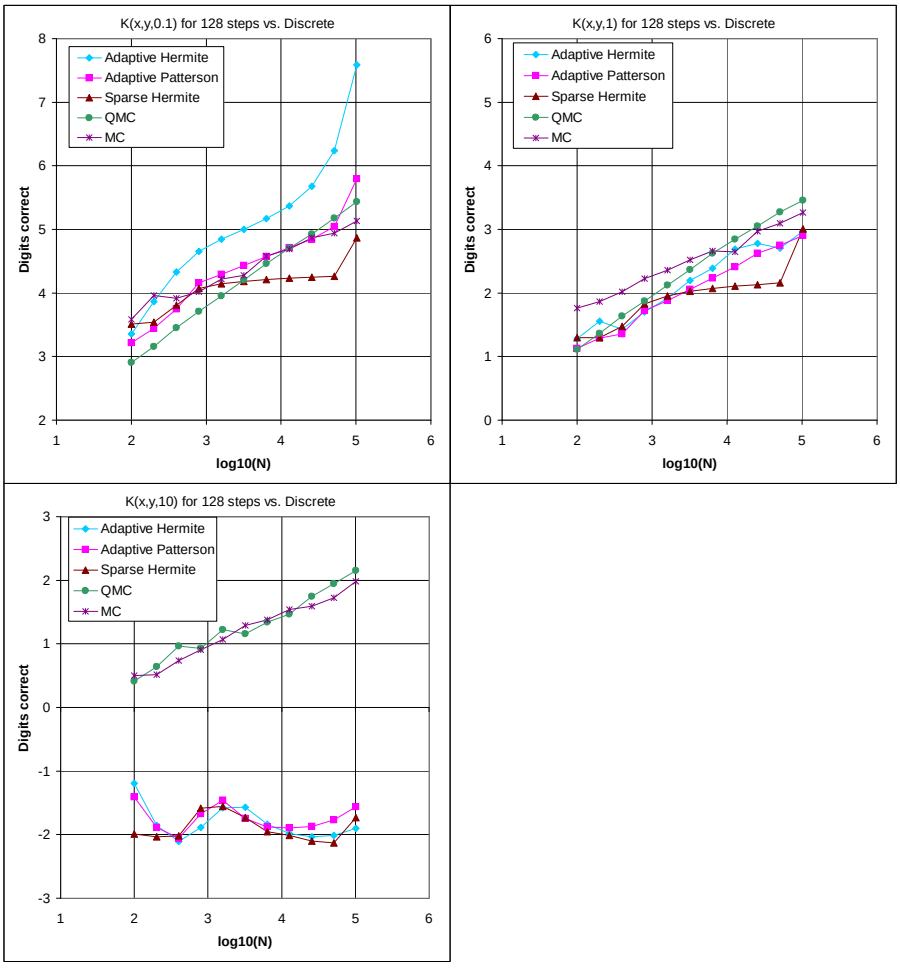


Figure 7.4.2: Quadrature results for the Green function K(x,y,t) for the anharmonic oscillator

for the times $\tau = 0.1$, $\tau = 1$ and $\tau = 10$. Since no analytical solutions for the anharmonic oscillator are known, we do not have any analytical reference values. Instead, we determined preliminarily which algorithm works best for each time τ, and approximated the true result by performing quadrature with 20 times as many evaluations as the maximum used in the benchmark.

The results for the anharmonic oscillator are qualitatively similar to those for harmonic oscillator. However, while the sampling methods attain about the same accuracy as for the harmonic oscillator, the adaptive methods perform by more than an order of magnitude.

The results for error estimation are similar to those for the harmonic oscillator, and are not shown.

7.5 Conclusion

The adaptive sparse grid algorithm works well for calculating the green function $K(x, y, \tau)$ for the time $\tau = 0.1$, is a little better (harmonic oscillator) or a little worse (anharmonic oscillator) for $\tau = 1$ and fails for $\tau = 10$ as compared to the sampling methods. The Gauss-Hermite rule performs better than the Patterson rule, because it is better suited to the Gaussian noise underlying the generation of the path. The error estimator goes fully astray, and cannot be used for these path integrals.

Compared to the simplicial sparse grid, the adaptive method benefits from the fact that the Brownian bridge method introduces a hierarchy between the different dimensions. Due to the method of construction by intercalation, the Gaussian random variables used first influence the shape of the path to a greater degree. The adaptive method can then refine specifically for these dimensions, which leads to an overall better accuracy (compare [16, 6]).

While the results are encouraging for short times, many problems in quantum mechanics require the computation for long time intervals. Also, instead of a pure Monte Carlo method, modified versions with importance sampling, like the Metropolis algorithm [26] are employed, that improve performance considerably. Locally adaptive refinement might yield similar gains for the adaptive sparse grid methods, but was not considered in this thesis. Further investigation is needed to determine if and how adaptive sparse grid quadrature can provide efficient solutions for quantum mechanical questions and other problems from physics.

8 Conclusion

Dimension-adaptive sparse grid quadrature offers an interesting approach to computing integrals of moderate and high dimensions. In contrast to the sampling methods, which are the main quadrature methods used for high dimensional integrals today, the sparse grid method is interpolatory and can therefore exploit high degrees of smoothness of the integrands. Dimension-adaptive sparse grids improve the method of sparse grids by taking into account the different importance of dimensions. This difference may be a coincidental property of the function, as for the Genz test functions. It may also arise from purposeful design, as for the Brownian Bridge [6]. For the latter case, adaptive sparse grids constitute a good method for exploiting this hierarchy, from which simplicial sparse grids cannot profit. This leads to an overall better convergence [16].

In this thesis, we have introduced a new black box method of error estimation and established theoretical results for contribution-based error estimation. Neither of these methods works very well, however.

We have also examined several different methods of index refinement. All are based on some sort of estimate of the size of additional index contributions. These methods work well in finding an index set suited to the function. We have seen that the different methods of estimation lead to very similar convergence, showing that index selection by estimation is a robust process that does not depend greatly on the details of the estimation strategy itself.

We have compared the performance of different quadrature rules for the sparse grid method. The Patterson rule represents a good choice for many integrands. However, for functions based on Gaussian noise, the Gauss-Hermite formulas seem to be particularly well suited and perform better than the Patterson rule.

We have implemented the different strategies proposed in this thesis on the computer in the object-oriented language Java. The implementation delivers on the goal of a modular design, and allows for the effortless mixing and matching of different strategies. As such, it presents a good base for further exploration of adaptive sparse grid methods for quadrature.

Although theoretical results show that asymptotically, the sparse grid method breaks the curse of dimensionality, the pre-asymptotic practice is more ambivalent, as evidenced by the Genz benchmarks and the quantum mechanical path integrals. Some types of integrands, notably the Genz Oscillatory function and the path integrals for short times, yield a high convergence, and the adaptive sparse grid algorithm then performs several orders of magnitude better than both simplicial sparse grids and the sampling methods. For other integrands, and notably those with a low

degree of smoothness, adaptive sparse grid quadrature does not perform well. As expected from theory, Monte Carlo performs truly independently of dimension in practice, whereas convergence for the sparse grid methods is noticeably slowed for higher dimensions in the settings of the Genz benchmark.

The new hybrid method introduced in this thesis appears to be a suitable combination of adaptive and simplicial sparse grids. It has the same convergence rate in theory and the same robust behavior in practice as the non-adaptive sparse grid method, but also incorporates the improvements of the adaptive method. Indeed, for difficult functions, the hybrid method even shows synergistic effects between the two approaches. With regard to these results, and to the fact that index selection by estimation is a robust, the hybrid approach offers a good standard with no immediate need for improvement.

Further research is required on classifying the types of problems for which adaptive sparse grids work well, and on how to employ the adaptive method to obtain highly accurate results. In particular, for the path integral problem from quantum mechanics, we see that adaptive sparse grids may be a good choice for short times. Unfortunately, many problems in physics require the integration over large times, for which the adaptive method is inferior to the established Monte Carlo and Quasi-Monte Carlo approaches. Further exploration will be needed to determine whether there is a niche in physics where adaptive sparse grids present an improvement over traditional sampling methods.

Path integrals occur naturally in financial mathematics in the form of stock prices and for other values. Since the Brownian Bridge construction allows for a hierarchy between the dimensions from which the dimension-adaptive algorithm can benefit, it will be worthwhile to examine the utility of the adaptive method for this domain. First results [16] look encouraging.

No satisfactory solution has been found for the estimation of the quadrature error. This is unfortunate and requires further effort, because a quadrature value by itself without a qualified estimate of its accuracy holds only little value.

Bibliography

[1] Archimedes, "On Measuring the Circle" ("Archimedes' Collected Works", transl. by TL Heath, 1897, Cambridge University Press)

[2] T Bonk. A new algorithm for multi-dimensional adaptive numerical quadrature. Adaptive Methods (eds. W Hackbusch, G Wittum) 54-68. Vieweg, 1994

[3] H Brass, K-J Förster. On the estimation of linear functionals. Analysis 7:237-258, 1987

[4] H Brass. Error bounds based on approximation theory. Numerical Integration (eds. TO Esplid and A Genz) 147-163. Kluwer, 1992

[5] H-J Bungartz, M Griebel. Sparse grids. Acta Numerica 13:147-269, 2004

[6] R Caflisch et al. Valuation of mortgage-backed securities using Brownian bridges to reduce effective dimension. Journal of Computational Finance 1:27-46, 1997

[7] CW Clenshaw, AR Curtis. A method for numerical integration on an automatic computer. Numerische Mathematik 2:197-205, 1960

[8] P Davis, P Rabinowitz. Methods of numerical integration. Academic Press, 1975

[9] P Deuflhard, A Hohmann. Numerische Mathematik I, 2. ed., p. 308. deGruyter, 1993

[10] G Dodig-Crnkovic. Theory of Science. p. 43-44. Online at www.mrtc.mdh.se/publications/0333.pdf

[11] N Etemadi. An elementary proof of the strong law of large numbers. Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete, 55:119-122, 1981

[12] RP Feynman. Space-time approach to nonrealtivistic qunatum mechanics. Review of Modern Physics 20:367-387, 1948

[13] L Führer. Allgemeine Topologie. Vieweg, 1977

[14] AC Genz. Testing Multiple Integration Software. Tools, Methods and Languages for Scientific and Engineering Computation, eds. B Ford et al. 208-217. North Holland, 1984

[15] T Gerstner, M Griebel. Numerical Integration using Sparse Grids. Numerical Algorithms 18:209-232, 1998

[16] T Gerstner and M Griebel. Dimension-Adaptive Tensor-Product Quadrature. Computing, 71(1):65-87, 2003

[17] M Hegland. Adaptive Sparse Grids. ANZIAM Journal 44(E):C335-C353, 2003

[18] E Hlawka. Funktionen von beschränkter Variation in der Theorie der Gleichverteilung. Annali di Matematica Pura ed Applicata 54:325-333, 1961

[19] I Karatzas, SE Shreve. Brownian Motion and Stochastic Calculus, 2nd edition. Springer, 1991

[20] S Lang. Algebra, Third Edition. Addison-Wesley, 1993

[21] JP Lewis, U Neumann. Performance of Java versus C++. http://www.idiom.com/~zilla/Computer/javaCbenchmark.html, 2004

[22] M Kac. On distributions of certain Wiener functionals. Transactions of the American Mathematical Society 65:1-13, 1949

[23] WA Light, EW Cheney. Approximation Theory in Tensor Product Spaces. Springer, 1985

[24] S Mac Lane. Categories for the Working Mathematician, Second Edition. Springer, 1998

[25] http://www.torstennahm.de/java/sparse.html

[26] N Metropolis et al. Equations of State Calculations by Fast Computing Machines. Journal of Chemical Physics 21:1087-1091, 1953.

[27] J von Neumann. Mathematical Foundations of Quantum Mechanics. Princeton University Press, 1955

[28] H Niederreiter. Random Number Generation and Quasi-Monte Carlo Methods, 1992, Society for Industrial and Applied Mathematics

[29] E Novak, K Ritter. High dimensional integration of smooth functions over cubes. Numerische Mathematik 75:79-97, 1996

[30] WH Press et al. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 2 edition, 1992

[31] TNL Patterson. The optimum addition of points to quadrature formulae. Mathematics of Computation 22:847-856, 1968

[32] HF Sinwel. Uniform approximation of differentiable functions by algebraic polynomials. Journal of Approximation Theory 32:1-8, 1981

[33] SA Smolyak. Quadrature and Interpolation Formulas For Tensor Products of Certain Classes of Functions. Doklady Akademii Nauk 148:1042-1045, 1963 (Russian). Soviet Mathematics Doklady 4:240-243, 1963 (English Translation)

[34] JF Traub. Information-Based Complexity. Academic Press, 1988

[35] C Zenger. Sparse grids. Parallel Algorithms for Partial Differential Equations (ed. W Hackbusch) p. 241-251. Vieweg, 1991