Monte Carlo Integration: A Comparison to Numerical Quadrature

Author

Mirjam Karlsson-Müller 951012-4200

Advisor

Philipp Birken

LUND UNIVERSITY

Department of Mathematics

Contents

Acknowledgements

First and foremost, I'd like to thank my advisor Philipp Birken for providing feedback, suggestions and knowledge every step of the way, especially for his thorough feedback on the first full version. I'd also like to thank Marco, Edmund and Kanishka, giving me much to correct on my drafts. And my grandfather, who proofread the final version and as an experienced mathematician shared his wisdom to my results. And last but not least, my husband, my mother and Ansgar for making sure I use english and swedish the way its intended.

Monte Carlo - Det är inte bara för Hasardspel

En Populärvetenskaplig Sammanfattning

Det var faktiskt för ungefär 80 år sedan, när matematiker som jobbade med att utveckla kärnvapen myntade begreppet för först gången. I grund och botten, använder Monte Carlo metoder slumpmässiga värden för att lösa problem. I det här projektet är det specifikt integrationsproblem.

Under ens matematikstudier, kommer vem som helst med ett litet intresse för matematik att stöta på integraler redan i gymnasiet. På universitetsnivå handlar en stor del av första året om att lära sig hur man beräknar integraler för hand. Men i verkligheten är beräkningar för hand inte alltid att föredra eller ens möjliga. När det här är fallet, tar vi datorn till hjälp för att beräkna ungefärliga lösningar till integralen. Olika områden inom matematik, i det här fallet numerisk analyis och statistik, erbjuder olika metoder. När man är specialiserad inom ett område når ens kunskaper ofta inte mer än grundnivån i andra områden. Föreställ dig att du är en turist i ett annat land. Som turist besöker du de mest berömda platserna och tar del av de mest populära upplevelserna, men de flesta av oss kommer aldrig dyka djupare in i kulturen. Det är ungefär så en student från numerisk analys känner när den läser en kurs inom statistik.

Det här projet kommer att försöka sig på just en sådan djupdykning. Med en grund i numerisk analys så är standard verktyget för att uppskatta integraler numerisk kvadratur. Det vilar dock en förbannelse över metoden: The curse of dimensionality. När man integrerar i höger dimensioner, så blir resultatet för felaktiga för att motivera en högre beräkningskostnad. Det verkar uppenbart att fråga sig själv om det finns ett bättre alternativ inom statistiken? Det är här Monte Carlo integration kommer in i bilden. Det här projektet kommer att försöka göra en direkt jämnförelse mellan Monte Carlo integration och numerisk kvadratur med målet att undersöka om Monte Carlo kan vara en metod som är mer lämpad för integration i två eller tre dimensioner.

Även om det är utom räckvid för detta projekt att få ett slutgiltig svar är det i alla fall en början. I enlighet med teorin som presenteras i projektet, är Monte Carlo integration en konkurrent till numerisk kvadratur i högre dimensioner. Men det visar sig att två eller tre dimensioner kan vara otillräcklig, eftersom båda metoderna producerar resultat med ungefär samma noggrannhet i testerna.

Abstract

Integrals are present everywhere in science, and their computation an emphasis in education. When methods of exact computation fail, a great variety of methods of approximation can step in. This project is interested in the Monte Carlo integration methods, an approach, where the integral is approximated based on the Law of Large Numbers. These methods are compared to the methods of numerical quadrature and tested on implementations, with the goal of seeing whether Monte Carlo integration could be a competitor for numerical quadrature in three and four dimensions. The comparison is made in terms of convergence, by looking at the n-th minimal error of an asymptotically optimal algorithm of each method. This shows that numerical quadrature methods have a smaller n-th minimal error for specific sets of functions in one dimension, but for sets of multivariable functions, whose smoothness are small compared to their dimension, Monte Carlo integration is a better pick.

1 Introduction

Integrals can be found in any field of science, in fact there are too many applications to name them all, but to name a few: In physics work is described by an integration of the force over the distance and electric flux is computed by the integral of an electric field over the surface. The Maxwell equations are multidimensional integrals used in electromagnetism to calculate total magnetic and electric fields. They are used to calculate the areas under and between curves and therefore used in, for example, biology for basic NMR spectroscopy, calculating the area under the peaks. With all these applications, it is no surprise that integrals take an important role in mathematics, and early on math students learn about them in calculus classes. While they are taught many tricks and strategies, most integrals can not be solved exactly, instead their solution has to be approximated. A simple example is

$$\int_0^2 e^{x^2} dx,$$

but also the equations describing Gaussian distributions and others. In these cases we rely on numerical methods, which approximate the result for us. Other situations where we use these methods are for example when confronted with a costly integral or when the value of the integrand is only known at a few points. There are different methods available to this end but this project is especially interested in two types: Methods based on Monte Carlo integration and how they compare to methods based on numerical quadrature in terms of rate of convergence.

The name Monte Carlo first came up related to mathematics in the 1940s, when scientists started studying games of chance and their behavior and outcomes and applied them to different fields. One of the earliest examples of Monte Carlo methods being used under this name, would be by the scientists working on the development of thermonuclear weapons during that same time period.[4]. The methods gained popularity after Fermi, von Neumann and Ulam discovered the possibility of applying Monte Carlo Methods to deterministic problems. According to Hammersley and Handcomb [3], this lead to an intense study of Monte Carlo Methods in the 1950s, were it was attempted to apply Monte Carlo methods to any problem, often more interested if it was possible, instead of whether it was plausible. Consequently, the methods decreased in popularity for a while afterwards, but recovered with the availability of modern digital computers.

While Monte Carlo integration comes from statistics and is based on probability theory, numerical quadrature has its origins in numerical analysis and is based on equations for area computation. Both methods have their strength and weaknesses and are applied to different problems. Numerical quadrature is efficient and cheap in small dimensions given a single problem, but struggles in higher dimension due to the minimal cost of computation growing exponentially in the dimension of the problem [6]. This phenomenon is also called the *Curse of Dimensionality*. It can be counteracted to some extent with a high degree of smoothness, as we will see in the end of section 5, but not compensated entirely. The Monte Carlo methods do not suffer from the curse and therefore offer themselves as a replacement for higher dimensions.

As Robert and Casella said[8, p. 64]: "Lastly, numerical integration tools cannot easily face the highly (or even moderately) multidimensional integrals that are the rule in statistical problems. Devising specific integration tools for those problems would be too costly, especially because we can take advantage of the probabilistic nature of those integrals." The Monte Carlo integration's weakness however, lies in their slow convergence, making them a worse choice to use in low dimensions [5].

Are Monte Carlo methods a competition for numerical quadrature when integrating two or three variable functions? This project aims to find an answer to this central question. It requires us to first take a look at each method, to understand how they work, orienting ourselves on chapters 2 and 5 of [5], but also chapter 10 of [2] and [1]. Then we make a theoretical comparison examining the rate of convergence of algorithms based on each method. To this end, the n-th minimal error will be introduced: It describes the smallest maximum error for a class of algorithms with cost $n \in \mathbb{N}$. The comparison of the two methods is made by finding an asymptotically equivalent expression for the n-th minimal error corresponding to an optimal algorithm of each method, which will then also be introduced. This is done for specific sets of functions, in one and in higher dimensions. This comparison follows chapter 7 of [5].

Based on the theoretical comparison, we will look at the implementation of the optimal algorithm for each approach using Python. The algorithms are tested on example functions chosen from the sets of functions used in the theoretical comparison, using one variable integrands in one dimensions, and two and three variable integrands in higher dimensions. This allows us to compare the theoretical results to the error plots generated by our tests. Based on Robert and Casella [7] we expect the numerical methods to be more efficient in the one dimensional set of functions. In two or three variable integrands, Monte Carlo methods could be a valid alternative for numerical quadrature.

2 Comparing Algorithms

For the comparison of algorithms to each approach, numerical quadrature and Monte Carlo methods, we first require some basic concepts. These are introduced here based on [5]. There are two types of algorithms which are of interest to us.

Randomised Algorithms include generating random numbers, which means that even if their input stays the same, their output may vary every time we run it. An example of this are the algorithms based on Monte Carlo integration. Since their outcome depends on random numbers, so does the size of their error.

Deterministic Algorithms do not use any random numbers, which means that the same input will always produce the same output. An example of this are algorithms based on numerical quadrature.

Let M be an algorithm that defines a transformation from a function $f \in F$ to $\mathbb{R}$:

$$M : F \to \mathbb{R}$$

where the output $M(f)$ approximates the integral of $f \in F$

$$S(f) = \int_G f(x)dx.$$

The error of M for a specific input $f \in F$ is then

$$\Delta(M,f) = |S(f) - M(f)|.$$

Mind that for randomized algorithms, this is not a deterministic value, but depending on the random values generated within the algorithm. When comparing two algorithms, one option is to look at their maximum error

$$\Delta(M,F) = \sup_{f \in F} \Delta(M,f).$$

Similarly, we can compare their maximum cost

$$\text{cost}(M,F) = \sup_{f \in F} \text{cost}(M,f).$$

To quantify cost, elementary functions and arithmetic operations are counted as one operation, whereas function evaluations heavily depend on the function f. Therefore, we give them the cost variable $c \geq d$, where d is the number of variables. In this project, we restrict ourselves to algorithms with maximum cost n, where n is an integer. The main criteria by which we compare algorithms in this project is the *n-th minimal error*. To introduce it, we require two other definitions first.

Definition 2.1: *Let $\mathfrak{M}^{det}(F)$ be the set of all deterministic algorithms, taking a function $f \in F$ as input. Set $n \in \mathbb{N}$. The set of deterministic algorithms with maximal cost n is then given by*

$$\mathfrak{M}_n^{\text{det}}(F) = \{M \in \mathfrak{M}^{\text{det}}(F) | \text{cost}(M,F) \leq n\}. \tag{1}$$

A similar definition can be made for randomized algorithms.

Definition 2.2: *Let $\mathfrak{M}^{ran}(F)$ be the set of all deterministic algorithms, taking a function $f \in F$ as input. Set $n \in \mathbb{N}$. The set of deterministic algorithms with maximal cost n is then given by*

$$\mathfrak{M}_n^{\text{ran}}(F) = \{M \in \mathfrak{M}^{\text{ran}}(F) | \text{cost}(M,F) \leq n\}. \tag{2}$$

Based on these two definitions, the n-th minimal error can now be introduced.

Definition 2.3: *The n-th minimal error of a set of algorithms $\mathfrak{M}_n^{det}(F)$ or $\mathfrak{M}_n^{ran}(F)$ with maximum cost n is given by*

$$e_n^{\mathrm{det}}(F) = \inf\{\Delta(M,F) \mid M \in \mathfrak{M}^{\mathrm{det}}(F)\}. \tag{3}$$

and

$$e_n^{\mathrm{ran}}(F) = \inf\{\Delta(M,F) \mid M \in \mathfrak{M}_n^{\mathrm{ran}}(F)\} \tag{4}$$

respectively.

We can use this error as a criteria for optimality. We say that a deterministic algorithm $M \in \mathfrak{M}_n^{\mathrm{det}}(F)$ is optimal in F, if

$$\Delta(M,F) = e_n^{\mathrm{det}}(F).$$

Analogously, we call a randomized algorithm $M \in \mathfrak{M}_n^{\mathrm{ran}}(F)$ optimal in F if

$$\Delta(M,F) = e_n^{\mathrm{ran}}(F).$$

2.1 Asymptotically Optimal Algorithms

Often cost and errors cannot be determined exactly, so Müller-Gronbach, Novak and Ritter [5] suggest to investigate their asymptotic behavior instead. For this we require a criteria for asymptotic equivalence.

Definition 2.4: *Two series of real numbers a_n, b_n on $[0,\infty[\cup\{\infty\}$ are considered weakly asymptotically equivalent if from an index n_0 all values are finite and*

$$c_1 \cdot a_n \leq b_n \leq c_2 \cdot a_n$$

for $n \geq n_0$ and with constants $0 < c_1 \leq c_2$. We write

$$a_n \asymp b_n.$$

Given this criterion for weak asymptotic equivalence, we can extend the definition to an optimal algorithm.

Definition 2.5: *We call a series of deterministic algorithms $M_k \in \mathfrak{M}_n^{det}(F)$ asymptotically optimal on F if*

$$\Delta(M_k,F) \asymp e_n^{det}(F).$$

Equally, a series of randomized algorithms $M_k \in \mathfrak{M}_n^{ran}(F)$ is called asymptotically optimal on F if

$$\Delta(M_k,F) \asymp e_n^{ran}(F).$$

However, not all constructed algorithms M_k can be constructed in a way that their cost seamlessly increases through the natural numbers. Hence a broader definition of the n-th minimal error is needed.

Definition 2.6: *We call a series of algorithms $M_k \in \mathfrak{M}^{det}(F)$ asymptotically optimal if for its corresponding minimal error it holds that*

$$e_n^{det}(F) \asymp inf\{\Delta(M_k,F) \mid cost(M_k,F) \leq n\}.$$

Analogously, a series of algorithms $M_k \in \mathfrak{M}^{ran}(F)$ is asymptotically optimal if for its corresponding n-th minimal error it holds that

$$e_n^{\mathrm{ran}}(F) \asymp \inf\{\Delta(M_k,F) \mid \mathrm{cost}(M_k,F) \leq n\}.$$

These last two definitions are at the center of the comparison we will make in section 5. First however, we will move on to looking at numerical quadrature and Monte Carlo integration, introducing the two approaches but also finding expressions for the n-th minimal error.

3 Numerical Quadrature

The basic idea of *Numerical Quadrature* is to integrate a simpler version of f instead of f itself.

$$S(f) = \int_G f \approx \int_G f_{\mathscr{X}} dx.$$

This simpler version $f_{\mathscr{X}}$ is given by polynomial interpolation, where $\mathscr{X}$ is the set of nodes the interpolation is based on. Given nodes $x_1,...,x_n \in G$ and weights $a_1,...,a_n \in \mathbb{R}$ the numerical quadrature $Q_n(f)$ approximates $S(f)$.

$$Q_n(f) = \sum_{i=1}^{n} a_i \cdot f(x_i). \tag{5}$$

Based on $Q_n(f)$ a deterministic algorithm can be constructed with maximum cost

$$\text{cost}(Q_n, F) = n \cdot (c+2) - 1 \asymp n \cdot c, \tag{6}$$

where c is the cost of a function evaluation. Note also that here the n stands for the number of nodes, not for the maximum cost. To look at numerical quadrature in several dimensions, we also need to understand interpolation in several dimensions, which we do next.

3.1 Multivariate Interpolation with Lagrange Basis

This project will focus on the generalization of *Lagrange Interpolation* to multivariate problems, as it is later used for testing. We know Lagrange interpolation of a one variable function $f(x)$ as follows. Given a function f and nodes $x_1,...,x_n$ we can approximate f with a polynomial p of degree $n-1$:

$$f_{\mathscr{X}} = \sum_{i=1}^{n} f(x_i) L_i(x),$$

where L_i given by

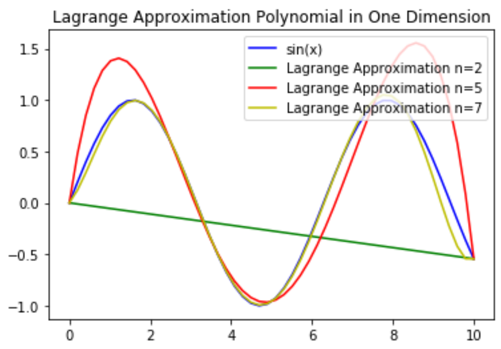
$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \qquad 1 \leq i \leq n.$$

This setting can be generalized for multivariate functions. Say we have a function of two variables $f(x,y)$, then $L_i(X)$ would be using a vector $X = (x,y)$. Using $x_1,...,x_n \in \mathscr{X}$ as nodes, we can write

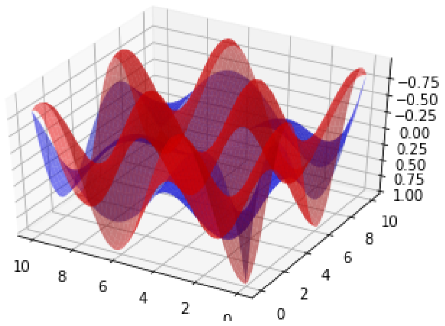
$$L_i(x,y) = L_i(x) \cdot L_i(y) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \cdot \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{y - x_j}{x_i - x_j}.$$

The same concept also applies for functions with more variables, and will be used in section 6 for the implementation of numerical quadrature for two and three variable functions.

In the one dimensional space, the Lagrange basis polynomials have degree $n-1$, as they are a product of $n-1$ affine factors. Cheney and Light [2] state that the same applies in higher dimensional cases, as a product of k affine functions on $\mathbb{R}$ is still a polynomial of degree k on $\mathbb{R}^s$, where an affine function is simply a member of $\Pi_1(\mathbb{R}^s)$.



(a) Lagrange Interpolation of $\sin(x)$ with different numbers of nodes.



(b) red=$\sin(x)\sin(y)$
blue=Lagrange Interpolation with $n=5$

Figure 1: Lagrange Interpolation.

An increase in interpolation nodes usually increases accuracy, see figure (a). However, when dealing with equidistant and/or more interpolation nodes than the degree of the function we are interpolating, an inaccuracy in form of oscillation towards the bounds of the interpolation interval can occur. We will go more into detail in the next section. Figure (b) shows the approximation in three dimensions for a specific n.

3.2 Composite Numerical Quadrature

From one dimensional interpolation we are familiar with a concept called *Runge's phenomenon* which stands for oscillations towards the boundaries of the interpolation interval when using equidistant points or too many points. In one dimensional numerical quadrature this problem is solved by splitting the integration interval into several integration intervals, interpolating and integrating on each separately and then adding the results. This greatly reduces the error towards the boundaries. To demonstrate: The standard *Mid Point Rule* for an integration interval $G=[0,1]$ is given by

$$K(f)=f\left(\frac{1}{2}\right)$$

The *Composite Mid Point Rule* however, takes the middle point of each of the n subintervals, hence changing the equation to

$$K_n(f)=\frac{1}{n}\sum_{i=1}^{n} f\left(\frac{2i-1}{2n}\right). \tag{7}$$

This approach translates to a multidimensional approach. By splitting the hypercube $G=[0,1]$ into k smaller hypercubes with side $1/k$ we can create a composite numerical quadrature equation. Let $T(G_i)$ be the smaller hyper cubes such that $G^1\cup\ldots\cup G^k=G$ which have been created by a linear transformation

$$T^i(x)=\frac{1}{k}\cdot(i+x) \qquad \text{where} \qquad i\in I=\{0,\ldots,k-1\}, k\in\mathbb{N}, x\in\mathbb{R}.$$

The *composite interpolation polynomial* is then given by

$$f_{\mathscr{X},k} = \sum_{i \in I} 1_G^i \cdot f_{T^i(\mathscr{X})},$$

where $\mathscr{X} \in G$ is the set of initial interpolation nodes. The composite numerical quadrature equation is hence given by

$$Q_{\mathscr{X},k}(f) = S(f_{\mathscr{X},k}) = \sum_{i \in I} \int_{T^i(G)} f_{T^i(\mathscr{X})}(x)dx. \tag{8}$$

Both, the composite mid point rule and the composite numerical quadrature, will be used in section 5, where they will represent the algorithms based on numerical quadrature and be compared to algorithms based on the Monte Carlo methods.

3.3 The n-th minimal error of deterministic algorithms

In section 5, the n-th minimal error of deterministic algorithms (3) will be compared to the n-th minimal error of randomized algorithms. For this we will look at asymptotically equivalent expressions of the corresponding errors. By the definition of asymptotic equivalence, we know that this requires an upper and a lower bound. Therefore, we will now find general expressions for the upper and lower bound of e^{det}. This section is based on chapters 7.1.2 and 7.1.5 by Müller-Gronbach, Novak and Ritter's [5]. The first step to derive a lower bound of e^{det} is a definition.

Definition 3.1: *The set of generalized deterministic algorithms $\widetilde{\mathfrak{M}}^{det}(F)$ contains all deterministic algorithms $M : F \to \mathbb{R}$, which can be written as a succession of transformations $\phi_k : \mathbb{R}^k \to \mathbb{R}$:*

$$M(f) = \phi_{v(f)}\big(N_{v(f)}(f)\big), \tag{9}$$

where $v(f)$ is the total number of function evaluations during the algorithm M, based on an input f, and $N_k(f)$ is the data obtained after doing $v(f) = k$ function evaluations. Note that this definition of $v(f)$ plays with the idea that it is possible for the number of total function evaluations to be only determined as the algorithm progresses, making the decision to terminate the algorithm depending on the information $N(f)$ obtained up to this point. The maximum amount of function evaluations $v(f)$ for $f \in F$ is given by

$$v(M,F) = \sup_{f \in F} v(M,f).$$

The definition of $\widetilde{\mathfrak{M}}^{\text{det}}$ includes all deterministic algorithms, ignoring whether $M \in \widetilde{\mathfrak{M}}^{\text{det}}$ can actually be computed by a deterministic algorithm. Consequently, the definition is broader than that of $\mathfrak{M}^{\text{det}}$. Therefore,

$$\mathfrak{M}^{\text{det}}(F) \subset \widetilde{\mathfrak{M}}^{\text{det}}(F).$$

Similarly, as in section 2, we can define $\widetilde{\mathfrak{M}}_n^{\text{det}}$ as the set of generalized deterministic algorithms with maximum cost n.

Definition 3.2: *The n-th minimal error of a set of algorithms $\widetilde{\mathfrak{M}}_n^{det}(F)$ with maximum cost n is given by*

$$\tilde{e}_n^{\text{det}}(F) = \inf\{\Delta(M,F) \mid M \in \widetilde{\mathfrak{M}}_n^{\text{det}}(F)\}. \tag{10}$$

The maximum costs of $M \in \widetilde{\mathfrak{M}}_n^{\det}$ can be estimated by

$$\text{cost}(M,F) \geq c \cdot v(M,F),$$

where c is again the cost of a function evaluation and costs of elementary functions and arithmetic operations are ignored, as they do not vary. Therefore, we know that the maximum cost of $M \in \widetilde{\mathfrak{M}}^{\det}$ is bound by

$$c \cdot v(M,F) \leq \text{cost}(M,F) \leq n,$$

from which we can conclude that M has a maximum of $[n/c]$ function evaluations at sequentially picked nodes. Following from this and (3.3), we conclude that for $n \geq c$

$$\mathfrak{M}_n^{\det}(F) \subset \widetilde{\mathfrak{M}}_{n/c}^{\det}(F).$$

Since all mappings $M \in \mathfrak{M}_n^{\det}(F)$ are also in $\widetilde{\mathfrak{M}}_n^{\det}(F)$, the n-th minimal error of $\mathfrak{M}_n^{\det}(F)$ is also a possible candidate for the n-th minimal error of $\widetilde{\mathfrak{M}}_n^{\det}(F)$, however, since there are even more mappings in this set, there can be an even smaller error. It follows that

$$e_n^{\det}(F) \geq \tilde{e}_{[n/c]}^{\det}(F).. \tag{11}$$

Therefore, the lower bound of $e^{\det}$ follows from the lower bound of $\tilde{e}^{\det}$.We finish up the derivation of the lower bound with a theorem, which describes the lower boundary of $\tilde{e}^{\det}$ in the multidimensional case.

Theorem 3.3 [5, p. 260] *Let $m > n$. Assume there exist functions $g_1,\ldots,g_m : G \to \mathbb{R}$ and a constant $\epsilon > 0$, so that the following three statements hold.*

(i) *the sets* $\{x \in G | g_i(x) \neq 0\}$ *are pairwise disjoint*

(ii) $\{\sum_{i=1}^{n} \delta_i \cdot g_i | \delta_1,\ldots,\delta_m \in \{\pm 1\}\} \subset F$

(iii) $S(g_1) \geq \epsilon$ *for all* $i = 1,\ldots,m$

Then it holds that

$$\tilde{e}_n^{\det}(F) \geq (m-n) \cdot \epsilon.$$

We will apply this to a set F in the comparison part of the project, where we also look at the problem specific lower bound for the one dimensional case.

For the upper bound of $e_n^{\det}(F)$, $\tilde{e}_n^{\det}(F)$ is used as well as the fact that under general geometric assumptions, numerical quadrature is asymptotically optimal, see [5]. This leads to the next theorem.

Theorem 3.4[5, p. 271] *Let S be linear and F symmetric and convex. Let $M \in \widetilde{\mathfrak{M}}_n^{det}(F)$ be a generalized algorithm with node x_1 and functions $\psi_2,\ldots,\psi_n$. Then there exists a quadrature equation Q_n with node x_1 and*

$$x_i = \psi_i(0), \qquad i = 2,\ldots,n$$

so that

$$\Delta(Q_n,F) \leq \Delta(M,F)$$

holds.

From this it follows that

$$\tilde{e}_n^{\det}(F) = \inf\{\Delta(Q_n,F) | Q_n \text{ with } n \text{ nodes}\}.$$

We know the maximum cost of Q_n on F from (6).

$$\text{cost}(Q_n, F) = n \cdot (c+2) - 1 \geq n.$$

It follows that

$$e_n^{\text{det}}(F) \leq \tilde{e}^{\text{det}}_{[n+1]/[c+2]}(F),$$

for $n \geq c+1$. Thus we have obtained an upper and lower boundary for e^{det}:

$$\tilde{e}^{\text{det}}_{[n+1]/[c+2]}(F) \geq e_n^{\text{det}}(F) \geq \tilde{e}^{\text{det}}_{[n/c]}(F) \geq (m - [n/c]) \cdot \epsilon. \tag{12}$$

4 Monte Carlo Integration

Monte Carlo Integration is a statistical approach to the approximation of an integral. Based on it, one can construct a randomized algorithm. To understand Monte Carlo integration, we need a bit of statistic terminology first. Monte Carlo integration is based on the *Law of Large Numbers*. There are several weaker and stronger versions of it, based on different types of convergence. Recall that the *Expectation* of a random variable X is given by

$$E(X) = \int_\Omega X(\omega) dP(\omega),$$

where Ω is the outcome space, $\omega \in \Omega$ and $P(\omega)$ is the probability of outcome ω.

Theorem 4.1: ***Law of Large numbers (strong version)[1, p.204]:*** *Assume that* $\{Y_n\}_{n\geq 1}$ *is a sequence of independent random variables with finite variance* $\text{Var}(Y_i) = \sigma^2 < \infty$ *and expectation* $E(Y_i) = \mu$. *Then the arithmetic mean*

$$X_n = \frac{1}{n} \sum_{i=1}^{n} Y_i$$

converges in quadratic mean to μ:

$$X_n \xrightarrow{L^2} \mu.$$

Hence the expectation which is defined by an integral, can be approximated by an arithmetic mean. This is the basic idea behind Monte Carlo integration.

Alternatively, using the same terminology as for the numerical approach, Monte Carlo integration approximates an integral

$$S(f) = \int_G f(x) dx$$

by

$$D_n(f) = \frac{1}{n} \sum_{i=1}^{n} f(x_i),$$

where x_i are n random samples in G.This holds accordingly if x_i are the samples of a equally distributed random vector X with length d. To optimize the methods many different alterations can be made, hence this version is also called the *Classic Monte Carlo Integration* as it is unaltered.

Based on Monte Carlo integration, one can construct a randomized algorithm $M \in \mathfrak{M}^{\text{ran}}$. Unlike with numerical quadrature, the output of M for a constant input f will not always be the same, but will depend on the random values generated. This means the output $M(f)$ is itself a random variable. Therefore, the error

$$|S(f) - M(f)|$$

is also a random variable. Its error is defined to be the standard error of the random variable,

$$\Delta(M,f) = (E(S(f) - M(f))^2)^{1/2}.$$

The cost of a randomized algorithm possibly also depends on the random numbers generated, therefore, it is a random variable itself. However, in the further comparisons, the integration area is limited to $G = [0,1]^d$, for which the maximum cost is

$$\mathrm{cost}(D_n, F) = n \cdot (c + d + 1),$$

where c is the cost of a function evaluation, n the amount of samples generated and d the dimension.[1]

4.1 The n-th Minimal Error for Randomized Algorithms

Similarly to section 3.3, we will in this section derive a lower bound for the n-th minimal error of randomized algorithms, see (4). We will however, not derive a general upper bound for it, since the upper bound is problem specific for randomized algorithms. This section follows Müller-Gronbach, Novak and Ritter [5], chapter 7.2.2. To start, we require a definition.

Definition 4.2: *Let Ω be an outcome space and $\omega \in \Omega$ one possible outcome in the outcome space. The set of generalized random algorithms $\widetilde{\mathfrak{M}}^{ran}(F)$ contains all maps $M : F \times \Omega \to \mathbb{R}$ which fulfill*

(*i*) *For any $\omega \in \Omega$ we can rewrite M such that $M(\cdot, \omega) \in \widetilde{\mathfrak{M}}^{det}(F)$.*

(*ii*) *For all $f \in F$, $M(f, \cdot) : \Omega \to \mathbb{R}$ and $v(f, \cdot) : \Omega \to \mathbb{N}$ are random variables.*

Note that as in (9), $v(f, \cdot)$ stands for the total number of function evaluations during an algorithm M. To understand the first statement, imagine that the randomly generated numbers ω of the randomized algorithm are determined beforehand and during the algorithm will only be called. By fixing this $\omega \in \Omega$ beforehand, we remove the random component of the mapping $M : F \times \Omega \to \mathbb{R}$, enabling us to write

$$M(\cdot, \omega) \in \widetilde{\mathfrak{M}}^{\mathrm{det}}(F).$$

Therefore, M fulfills (i) if this process of fixing $\omega \in \Omega$ is possible for all $\omega \in \Omega$. The second statement plays to the random part of M, saying that since we generate random numbers in the algorithm, the outcome $M(f, \cdot)$ of a fixed input f will itself be a random variable. We remember (9), where we treated $v(f)$ as a function of the input f, with the idea that it is possible that the amount of function evaluations is only determined as the algorithm is running. When to terminate, does then only depend on the information $N_k(f)$ acquired after $v(f) = k$ function evaluations. In the case of a randomized algorithm, $N_k(f, \omega)$ does not only depend on the input f, but also on the random variables generated during the algorithm. We can conclude that therefore the amount of total function evaluations also depends on the random values. If we thus fix the input f, the output $v(f, \cdot)$ only depends on the random values, therefore fulfilling $v(f, \cdot) : \Omega \to \mathbb{N}$.

Analogously to (9), we are not interested if a map $M \in \widetilde{\mathfrak{M}}^{\mathrm{ran}}(F)$ can actually be computed with a randomized algorithm, therefore it holds that

$$\mathfrak{M}^{\mathrm{ran}}(F) \subset \widetilde{\mathfrak{M}}^{\mathrm{ran}}(F).$$

[1]See Chapter 2, Example 2.5 in Müller-Gronbach, Novak and Ritter p. 19 [5].

In correspondence to section 3.3 we ignore the cost for arithmetic and comparison operations, evaluations of elementary functions and generating of random values, thereby reducing the cost of an algorithm $M \in \widetilde{\mathfrak{M}}^{\text{ran}}(F)$ to

$$\text{cost}(M,F) \geq c \cdot v(M,F),$$

where

$$v(M,F) = \sup_{f \in F} E(v(M,f,\cdot)).$$

Definition 4.3: *Reducing the cost of a mapping $M \in \widetilde{\mathfrak{M}}^{ran}$ to the amount of function evaluations and their cost, the generalized random algorithms with maximum cost n are given by*

$$\widetilde{\mathfrak{M}}_n^{ran}(F) = \{M \in \widetilde{\mathfrak{M}}^{ran}(F) | v(M,F) \leq n\}$$

and their corresponding n-th minimal error by

$$\tilde{e}_n^{ran}(F) = \inf\{\Delta(M,F) | M \in \widetilde{\mathfrak{M}}_n^{ran}(F). \tag{13}$$

With the same reasoning that we used to arrive at 11, we conclude that

$$e_n^{\text{ran}}(F) \geq \tilde{e}_{[n/c]}^{\text{ran}}(F). \tag{14}$$

Therefore, it is possible to bound $e_n^{\text{ran}}(F)$ with a lower bound derived for $\tilde{e}_{[n/c]}^{\text{ran}}(F)$. To derive a lower bound for it however, we require some additional definitions.

Definition 4.4: *Let $f_1,...f_m \in F$ be functions and $\alpha_1,...,\alpha_m > 0$ be weights such that $\sum_{k=1}^m \alpha_k = 1$. Furthermore, let $\mu(A)$ be a discrete probability measure on F, defined by*

$$\mu(A) = \sum_{k=1}^m \alpha_k \cdot 1_A(f_k), \qquad A \in F. \tag{15}$$

Then the average error of $M \in \widetilde{\mathfrak{M}}^{det}$ with respect to μ is given by

$$\Delta(M,\mu) = \left(\int_F (S(f) - M(f))^2 d\mu(f)\right)^{1/2} = \left(\sum_{k=1}^m \alpha_k \cdot (S(f_k) - M(f_k))^2\right)^{1/2}.$$

Meaning that the supremum of the maximum error $\Delta(M,F)$ gets replaced with an average error. A new set of generalized deterministic algorithms is constructed with this.

$$\widetilde{\mathfrak{M}}_n^{\text{det}}(\mu) = \{M \in \widetilde{\mathfrak{M}}^{\text{det}}(F) | v(M,\mu) \leq n\}$$

with

$$v(M,\mu) = \int_F v(M,f) d\mu(f)$$

describes all mappings $M \in \widetilde{\mathfrak{M}}^{\text{det}}(F)$, which use an average of n function evaluations. Analogously to (10) and (13), $\tilde{e}_n^{\text{det}}(\mu)$ can be described by

$$\tilde{e}_n^{\text{det}}(\mu) = \inf\{\Delta(M,\mu) | M \in \widetilde{\mathfrak{M}}_n^{\text{det}}(\mu)\}.$$

This n-th minimal error can be used to define a lower bound for $\tilde{e}_n^{\text{ran}}(F)$, see the next theorem.

Theorem 4.5[5, p.280] *For every discrete probability measure μ such that (15), then it holds that*

$$\tilde{e}_n^{\text{ran}}(F) \geq \frac{1}{\sqrt{2}} \tilde{e}_{2n}^{\text{det}}(\mu).$$

Analogously to Theorem 3.3, there exists a theorem describing the lower bound of $\tilde{e}_n^{\text{det}}(\mu)$.

Theorem 4.6[5, p.281] *Let $m > 2n$. If there exists functions $g_2,...,g_m : G \to \mathbb{R}$ and a constant $\epsilon > 0$ such that the following are fulfilled.*

(i)	*the sets $\{x \in G \mid g_i(x) \neq 0\}$ are pairwise disjoint.*
(ii)	$\tilde{F} = \{\sum_{i=0}^{m} \delta_i \cdot g_i \mid \delta_1,...,\delta_m \in \{\pm 1\}\} \subset F$
(iii)	$S(g_i) \geq \epsilon$ *for all* $i = 1,...,m$

Then it holds that

$$\tilde{e}^{\text{det}}(\mu) \geq (m/2 - n)^{1/2} \cdot \epsilon.$$

Taking into consideration (14), Theorem 4.5 and Theorem 4.6, we have derived an expression for the lower bound of $e_n^{\text{ran}}(F)$:

$$e_n^{\text{ran}}(F) \geq \tilde{e}_{[n/c]}^{\text{ran}}(F) \geq \frac{1}{\sqrt{2}} \tilde{e}_{[2n/c]}^{\text{det}}(\mu) \geq (m/2 - \frac{2n}{c})^{1/2} \cdot \epsilon \tag{16}$$

with $m > 2n$. This will be applied in the next section of the project. We will then also investigate problem specific upper boundaries for $e_n^{\text{ran}}(F)$.

5 Comparison

In this part, a direct comparison between Monte Carlo integration and numerical quadrature will be made for two specific sets of functions F, one containing one variable functions and the other containing multivariable functions. On one side we have the team of algorithms based on numerical quadrature, sending forth their best fighter, meaning asymptotically optimal algorithm, for each setting. It will go up against the best fighter of the team of algorithms based on Monte Carlo integration on the other side. We then compare the n-th minimal error of the two best fighters for each specific set of functions and see who comes out on top. This ensures equal a priori assumptions for $f \in F$. Since the result of a deterministic algorithm could always be randomly matched by the result of a randomized algorithm, we can say that

$$\widetilde{\mathfrak{M}}_n^{\text{det}}(F) \subset \widetilde{\mathfrak{M}}_n^{\text{ran}}(F),$$

from which it follows that

$$e_n^{\text{det}}(F) \geq e_n^{\text{ran}}(F).$$

The randomized algorithms are then a better choice to the deterministic on a F, if $e_n^{\text{ran}}(F) < e_n^{\text{det}}(F)$. The theorems and proofs of this section of the project are taken from/based on sections 7.1.2, 7.1.4, 7.2.2 and 7.2.3 of [5].

5.1 Comparison on F^1

To make this comparison we first need to define F^1.

Definition 5.1: *Let $f \in C^1([0,1])$ and $||\cdot||$ be a semi norm defined by*

$$||f|| = ||f'||_\infty.$$

F^1 is then given by

$$F^1 = \{f \in C^1([0,1]) \mid ||f|| \leq 1\}.$$

For this set of functions, we expect the numerical approach to be more successful, as previously mentioned in the introduction to this project. We now compare the n-th minimal error of each approach's best fighter.

5.1.1 Best Fighter Numerical Quadrature: Composite Mid Point Rule

The best figher of numerical quadrature on F^1 is the composite mid point rule, but equivalently also the trapezoidal rule or Gaussian quadrature as they are asymptotically optimal as well. However, in this project we will restrict ourselves to the mid point rule. The lower bound of this method follows from the next theorem.

Theorem 5.2 [5, p.257] *Let K_n be the composite mid point rule given by (7). For the set of functions F^1 it holds that*

$$\tilde{e}_n^{\text{det}}(F^1) = \Delta(K_n, F^1) = \frac{1}{4n}$$

From which a Corollary follows:

Corollary 5.3[5, p.259] *The mid point rule defines an asymptotically optimal series of algorithms on the set F^1. It holds that*

$$e_n^{\text{det}}(F^1) \asymp \frac{c}{n}.$$

To be able to proof this corollary, we require the following lemma.

Lemma 5.4 [5, p.259] *Let a_n, b_n be two montonely decreasing series of positive real numbers, which have subsequences with the property*

$$a_{n_k} \leq c_1 \cdot b_{n_k}$$

with $c_1 > 0$. Furthermore either

$$a_{n_k+1} \geq c_2 \cdot a_{n_k} \qquad \text{or} \qquad b_{n_k+1} \geq c_2 \cdot b_{n_k}$$

holds, with $0 < c_2 \leq 1$. Then it follows that

$$a_n \leq c_1/c_2 \cdot b_n \qquad \text{for } n \geq n_1.$$

Proof Corollary 5.3: If we combine Theorem 5.2 with (11), then we get the following lower bound for $e_n^{\text{det}}(F)$:

$$e_n^{\text{det}}(F) \geq \tilde{e}_{[n/c]}^{\text{det}} = \frac{c}{4n}.$$

For the upper bound we apply Lemma 5.4 with $b_n = \frac{c}{n}$ and

$$a_n = \inf\{\Delta(K_k, F^1) \mid \text{cost}(K_k, F^1) \leq n\} = e_n^{\text{det}}(F),$$

where K_k is the middle point rule with k nodes. The cost of the mid point rule K_k is given by

$$\text{cost}(K_k, F^1) = k \cdot (c+1) = n_k.$$

Due to this and theorem 5.2, it follows that

$$a_{n_k} = \Delta(K_k, F^1) = \frac{1}{4k}.$$

Replacing $k = (c+1)/n$ and $c/n_k = b_{n_k}$, we conclude

$$\begin{aligned}\frac{1}{4k} &= \frac{c+1}{4n_k}\\ &= \frac{c+1}{4c} b_{n_k}\\ &= b_{n_k}\left(\frac{1}{4} + \frac{1}{4c}\right) \leq \frac{1}{2} b_{n_k}.\end{aligned}$$

It follows by lemma 5.4 that $a_n \leq b_n$ for $n \geq c+1$. In other words

$$a_n = e_n^{\text{det}}(F) \leq \frac{c}{n},$$

which combined with the lower bound is enough to prove the corollary. □.

5.1.2 Best Fighter Monte Carlo Integration: Random Riemann Sums

Random Riemann Sums are a version of Monte Carlo integration which is defined by

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} f(X_i)$$

with independently equally distributed random variables X_i on $B_i = [(i-1)/n, i/n]$. The corresponding theorem to theorem 5.1 is as follows.

Theorem 5.5 [5, p.283] *For the set of functions F^1 and the method R_n of random Riemann sums*

$$\tilde{e}_n^{\text{ran}}(F^1) \leq \Delta(R_n, F^1) = \frac{1}{2\sqrt{3} \cdot n^{3/2}}$$

and

$$\tilde{e}_n^{\text{ran}}(F^1) \geq \frac{1}{144\sqrt{2} \cdot n^{3/2}}$$

hold. From this theorem also follows a corollary.

Corollary 5.6[5, p.284] *The method of random Riemann sums R_n defines an asymptotically optimal sequence of algorithms on F^1 and*

$$e_n^{\text{ran}}(F^1) \asymp \left(\frac{c}{n}\right)^{3/2}$$

holds.

Proof: From Theorem 5.5 we know that

$$\frac{1}{144\sqrt{2}} \cdot n^{-3/2} \leq \tilde{e}_n^{\text{ran}}(F^1) \leq \frac{1}{2\sqrt{3}} \cdot n^{-3/2}.$$

The next step is applying definition 2.4

$$e_n^{\text{ran}}(F) \asymp n^{-3/2},$$

which implies

$$e_n^{\text{ran}}(F) \geq \tilde{e}_{[n/c]}^{\text{ran}}(F) = \left(\frac{c}{n}\right)^{3/2}.$$

□

5.1.3 Conclusion

From corollary 5.3 we know

$$e_n^{\det}(F) \asymp \frac{c}{n}$$

and from corollary 5.6 we know that

$$e_n^{\text{ran}}(F) \asymp \left(\frac{c}{n}\right)^{3/2}.$$

As we have assumed $n \geq c$ to arrive at (11) and (14), therefore it holds that

$$\frac{c}{n} \geq \left(\frac{c}{n}\right)^{3/2}.$$

When looking at the behavior of these two errors as n goes towards infinity, it is clear that the randomized algorithm has a higher speed of convergence. This goes against our expectations, however, we will look at this behavior in practice in section 6.

5.2 Comparison on F_d^r

As in the previous section, we first define the set of functions we are comparing for.

Definition 5.7: *Let $G = [0,1]^d$ be the integration area, $f \in C^r(G)$ and $||\cdot||$ be a semi norm defined by*

$$||f|| = \max|\alpha| = r||f^{(\alpha)}||_\infty$$

F_d^r is then given by

$$F_d^r = \{f^r(G) \mid ||f|| \leq 1\}.$$

Given the results of the previous section and the already known problems of numerical quadrature due to the curse of dimensionality, Monte Carlo's best fighter is expected to come out on top here.

5.2.1 Best Fighter Numerical Quadrature: Composite Numerical Quadrature

We have seen composite numerical quadrature in section 3.2 and especially in (8), so we will move straight to the theorem describing its n-th minimal error.

Theorem 5.8 [5, p.269] *On the set of functions* F_d^r

$$\tilde{e}_n^{\det}(F_d^r) \asymp n^{-r/d}$$

holds. Furthermore the composite quadrature equations $Q_{\mathscr{X},k}$ *fulfill*

$$\Delta(Q_{\mathscr{X},k}, F_d^r) \asymp k^{-r} \asymp n_k^{-r/d},$$

where n_k *refers to the amount of nodes of* $Q_{\mathscr{X},k}$.
A corrollary follows from this theorem.

Corollary 5.9 *The sequence of composite numerical quadrature equations* $Q_{\mathscr{X},k}$ *is asymptotically optimal on* F_d^r *and*

$$e_n^{\det}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d}$$

holds.

Proof: The lower bound is quickly proved with

$$e_n^{\det}(F_d^r) \geq \tilde{e}_{[n/c]}^{\det}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d}.$$

For the upper bound and asymptotic optimality we use the same strategy from the proof of corollary 5.3.
Let

$$\begin{aligned} a_n &= \{\Delta(Q_{\mathscr{X},k}, F_d^r | \operatorname{cost}(Q_{\mathscr{X},k}, F_d^r) \leq n\}, \\ b_n &= \left(\frac{c}{n}\right)^{r/d}, \\ n_k &= \operatorname{cost}(Q_{\mathscr{X},k}, F_d^r). \\ a_{n_k} &= \Delta(Q_{\mathscr{X},k}, F_d^r) \end{aligned}$$

According to theorem 5.8,

$$a_{n_k} = \tilde{e}_{n_k}^{\det}(F_d^r) \asymp n_k^{-r/d} = b_{n_k} \cdot c^{-r/d} \leq b_n.$$

By lemma 5.4 it follows that $a_n \leq b_n$, or in other words

$$a_n = e_n^{\det}(F_d^r) \leq \left(\frac{c}{n}\right)^{r/d},$$

which combined with the lower boundary is enough to prove the corollary. □

5.2.2 Best Fighter Monte Carlo Integration: Monte Carlo with Control Variates

As briefly mentioned in the introduction to Monte Carlo integration, there is a grand variety of possibilities to optimize the classic method. Here we are interested in the *Control Variates*, which is one of the methods that utilize variance reduction.
The idea is actually quite similar to numerical quadrature: Instead of approximating the expectation for

$$Y = f(X),$$

we instead pick a function that is closely related to $f(X)$, but whose expectation is known or easily computable and let it define a second random variable

$$Z = \tilde{f}(X).$$

We calculate the expectation of

$$\tilde{Y}_b = bE(\tilde{f}(X)) + (f - b\tilde{f})(X),$$

already rewriting the expression by approximating the expectation with the mean we get

$$E(\tilde{Y}_b) = \tilde{D}_{n,b} = bE(\tilde{f}(X)) + \frac{1}{n}\sum_{i=1}^{n}(f - b\tilde{f})(X_i),$$

where $b \in \mathbb{R}$. It is therefore crucial how Z and b are chosen. Müller-Gronbach, Novak and Ritter [5] suggest using composite interpolation so that

$$Z = \tilde{f}(X) = f_{\mathscr{X},k}.$$

Hence the Monte Carlo method using composite interpolation is given by

$$M_{\mathscr{X},k}(f) = S(f_{\mathscr{X},k}) + \frac{1}{k^d} \cdot \sum_{i=0}^{k^d}(f - f_{\mathscr{X},k})(X_i),$$

where X_i are equally distributed on G.

Theorem 5.10 [5, p.285] *For the set of functions F_d^r it holds that*

$$\tilde{e}_n^{\text{ran}}(F_d^r) \asymp n^{-(r/d+1/2)}.$$

Furthermore the Monte Carlo methods $M_{\mathscr{X},k}$ fulfill

$$\Delta(M_{\mathscr{X},k}, F_d^r) \asymp k^{-(r+d/2)} \asymp m_k^{-(r/d+1/2)},$$

where m_k are the amount of function evaluations of $M_{\mathscr{X},k}$.

Corollary 5.11 *The sequence of Monte Carlo methods $M_{\mathscr{X},k}$ is asymptotically optimal and*

$$e_n^{\text{ran}}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d+1/2}.$$

Proof: The lower bound is proven with

$$e_n^{\text{ran}}(F_d^r) \geq \tilde{e}_{[n/c]}^{\text{ran}}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d+1/2}.$$

For the upper bound and asymptotic optimality we use the same strategy from the proof of corollary 5.2. Let

$$\begin{aligned}
a_n &= \{\Delta(M_{\mathscr{X},k}, F_d^r | \operatorname{cost}(M_{\mathscr{X},k}, F_d^r) \leq n\},\\
b_n &= \left(\frac{c}{n}\right)^{r/d+1/2},\\
n_k &= \operatorname{cost}(M_{\mathscr{X},k}, F_d^r).\\
a_{n_k} &= \Delta(M_{\mathscr{X},k}, F_d^r)
\end{aligned}$$

According to theorem 5.10

$$a_{n_k} = \tilde{e}_{n_k}^{\text{ran}}(F_d^r) \asymp n_k^{-(r/d+1/2)} = b_{n_k} \cdot c^{-(r/d+1/2)} \leq b_n \cdot c^{-1/2}.$$

By lemma 5.4 it follows that $a_n \leq b_n$, or in other words

$$a_n = e_n^{\text{ran}}(F_d^r) \leq \left(\frac{c}{n}\right)^{(r/d+1/2)},$$

which combined with the lower boundary is enough to prove the corollary. □

5.2.3 Conclusion

From corollary 5.8 we know that the n-th minimal error of a deterministic algorithm based on composite numerical quadrature the n-th minimal error is given by

$$e_n^{\mathrm{det}}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d}.$$

Corresponding, corollary 5.10 showed that the n-th minimal error of a randomized algorithm based on control variates is given by

$$e_n^{\mathrm{ran}}(F_d^r) \asymp \left(\frac{c}{n}\right)^{r/d+1/2}$$

Assuming d, r are given and c is the same for both, we are only interested in how the error changes with respect to n. The n-th minimal error of composite numerical quadrature is proportional to

$$n^{-r/d}$$

and the n-th minimal error of control variates is proportional to

$$n^{-r/d-1/2}$$

Consequently, it is smaller for the Monte Carlo algorithm, which shows that on F_d^r the randomized algorithm based on control variates, a Monte Carlo method, is a better choice than the deterministic algorithm based on composite numerical quadrature. This is especially the case, when the smoothness r is small in relation to the dimension d due to the curse of dimensionality.

6 Testing

In this section, we want to test the theoretic results of the previous section but also the unaltered approach for each method. For this the best fighters implementation get tested on f's from F^1 and F_d^r and then the errors plotted and obtained. As the theoretic results did, this section also stays in an integration area of $G = [0,1]^d$. Recall that the error of randomized algorithms is itself a random variable, and hence changes every time the algorithm is run. To get an estimate for a general error, one takes the expectation of said random variable error, which can then be approximated by the mean. Therefore, the classic Monte Carlo and the random Riemann implementation was run $j = 1000$ times, and the control variates implementation was run $j = 10$ times, and the results averaged and then compared. Furthermore, the true value of the integral $S(f)$ was computed with *scipy.integrate.quad*, and for the error calculated with

$$|S(f) - M(f)|,$$

where $M(f)$ is the output of the algorithm. Consistent with the last section, we start with testing on $f \in F^1$. The code to all the implementations in this section can be found in section 8. All the plots are on a logarithmic scale.

6.1 Testing Best Fighters for $f \in F^1$:

For functions in F^1 the following were chosen:

$$f_1(x) = \sin(x)$$
$$f_2(x) = e^{\frac{1}{2}x}$$

6.1.1 Implementations of Composite Mid Point Rule and Random Riemann Sums

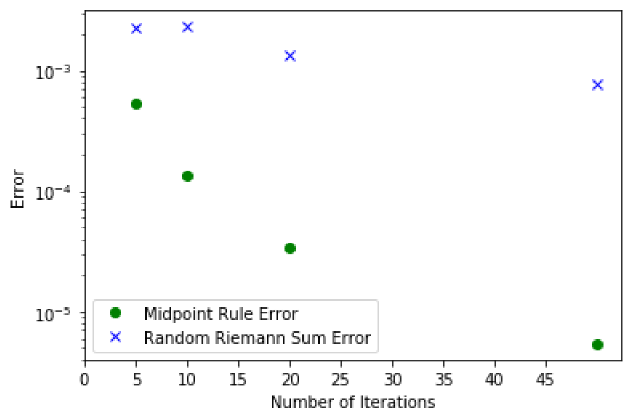
The composite mid point rule was implemented in a way, that it would first calculate the midpoints m_i for each of the n subintervals of the integration interval $[0,1]$ and then sum them up as follows

$$K_n = \frac{1}{n}\sum_{i=1}^{n} f(m_i).$$

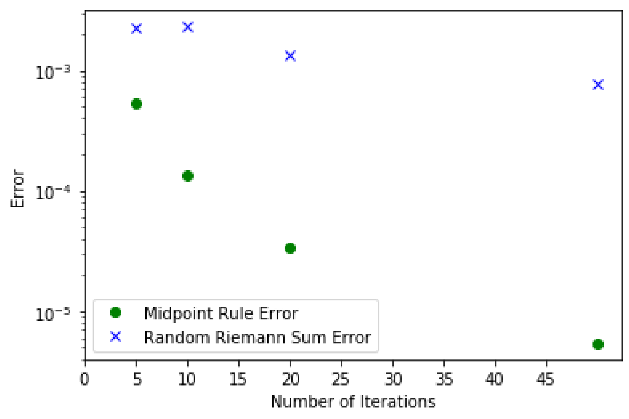
The random Riemann sums implementation works very similar, but instead of using the midpoints, it uses n randomly generated points x_i in the integration interval $[0,1]$.

$$R_n = \frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

6.1.2 Results for Mid Point Rule and Random Riemann Sums



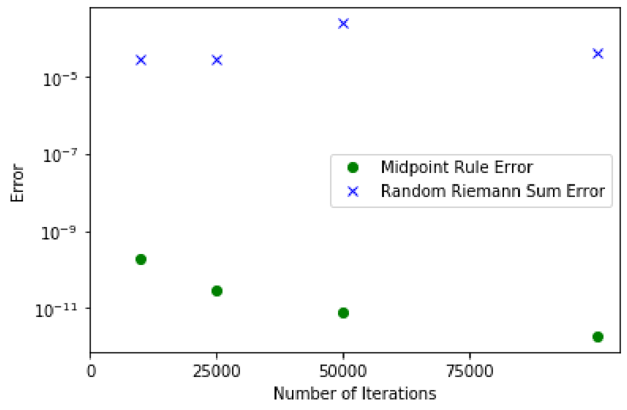
(a) $f_1(x) = \sin(x)$



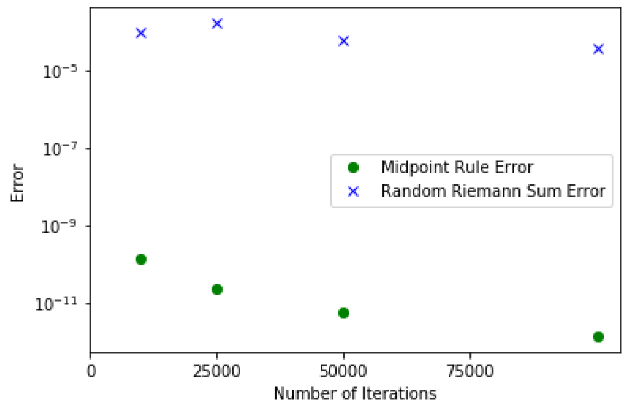
(b) $f_2(x) = e^{\frac{1}{2}x}$

Figure 2: Error per Number of Iterations

These plots show what we initially expected: For integration on a one dimensional set, it is best to stick to algorithms based on numerical quadrature, such as the mid point rule. But in the theoretical comparison we saw that the rate of convergence is higher for Random Riemann Sums, so why is this not visible in the plots? There can be several reasons for that. First, the results of section 5 are based on asymptotic equivalence. Second, even if the rate of convergence is higher, if the error is much larger to begin with, it will take many computations for the error curve of Random Riemann sums to cross the error curve of the composite mid point rule. The plots below are showing the error for a much higher amount of iterations, but even going to up to 100'000 points, the error of the Random Riemann sum does not fall beneath the error of the mid point equation. Hence, it is most likely due to the asymptotic nature of the results, that the application does not agree with the theory. It is also to note that even if the error of Random Riemann sum would have crossed the error of the mid point rule with such a high number of points, the cost at this point by far exceed the costs of the algorithm based on the mid point rule, because the output of the random Riemann sum implementation varies substantially. Therefore, it is only useable after running it many times and taking an average. When doing that for a large number of points, the costs quickly increase. Note that in the plots below the average error of the Random Riemann Sum is not taken over 1000 runs, but over 100.



(a) $f_1(x) = \sin(x)$



(b) $f_2(x) = e^{\frac{1}{2}x}$

Figure 3: Error per Number of Iterations

It takes many iterations more for the random Riemann sum method to reach an accuracy of around 10^{-3}, as it does for the mid point rule.

6.2 Testing the Best Fighters and the Not Optimized Versions on F_d^r

The test functions in F_d^r are

$$\begin{aligned} g_1(x,y) &= \cos(x)\cos(y) \\ g_2(x,y) &= e^{-1/2x}\cdot e^{-1/2y} \\ g_3(x,y,z) &= \sin(x)\sin(y)\sin(z) \\ g_4(x,y,z) &= e^{-1/3x}\cdot e^{-1/3y}\cdot e^{-1/3z} \end{aligned}$$

The first two implementations are based on classic Monte Carlo and basic numerical quadrature with a Lagrange basis, without any optimizations to decrease the error.

6.2.1 Numerical Quadrature Implementation

As seen in 3.2, the basic numerical quadrature is of the form

$$Q_n(F) = \sum_{i=1}^{n} a_i \cdot f(x_i).$$

To test the weights a_i from Lagrange interpolation were used.

$$a_i = \int_G L_i dx.$$

It was tested in two and three dimensions over $[0,1]^d$. For a two variable integrand, numerical quadrature with a Lagrange basis looks as follows

$$Q_n(f) = \sum_{i=1}^{n}\left(f(x_i,y_i)\cdot\int_0^1 L_i(x)dx\cdot\int_0^1 L_i(y)dy\right),$$

which was used as a basis for the implementation. Similarly, for a two variable integrand, it is given by

$$Q_n(f) = \sum_{i=1}^{n} \left(f(x_i, y_i, z_i) \cdot \int_0^1 L_i(x)dx \cdot \int_0^1 L_i(y)dy \cdot \int_0^1 L_i(z)dz \right).$$

6.2.2 Classic Monte Carlo Implementation

As seen in part 4, the idea behind Monte Carlo integration is to the arithmetic average of n random samples in the integration area.

$$D_n(f) = \frac{1}{n} \sum_{i=1}^{n} f(x_i).$$

Since the tests were done with two and three variable integrands, it was implemented to take $n \cdot d$ samples in $[0,1]$ which where then assembled into n random vectors with length d.These were then summed up and divided by n.

Two dimensions:

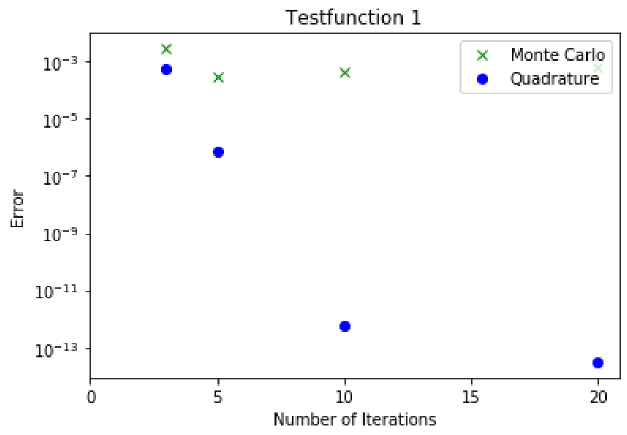
$$D_3(f) = \frac{1}{n} \sum_{i=1}^{n} f(x,y) \qquad \text{with} \qquad (x,y) \in [0,1]^2.$$

Three dimensions:

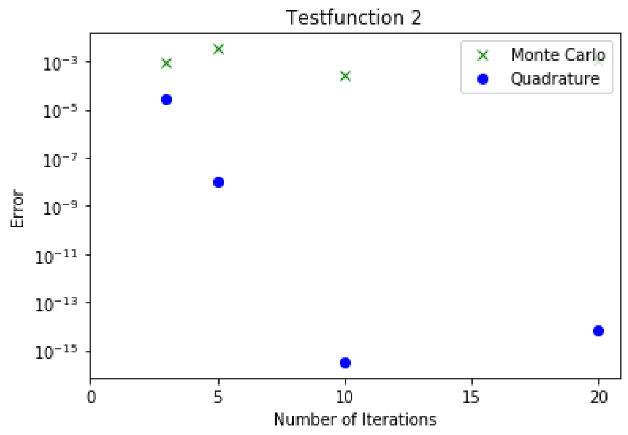
$$D_4(f) = \frac{1}{n} \sum_{i=1}^{n} f(x,y,z) \qquad \text{with} \qquad (x,y,z) \in [0,1]^3.$$

6.2.3 Results of Numerical Quadrature and Classic Monte Carlo

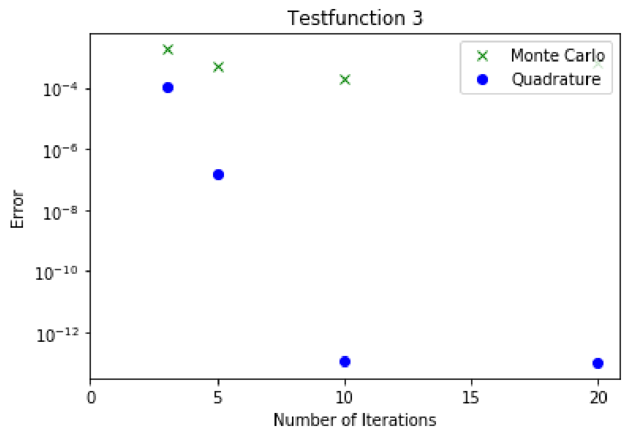
The plots show the error over amount of iterations.



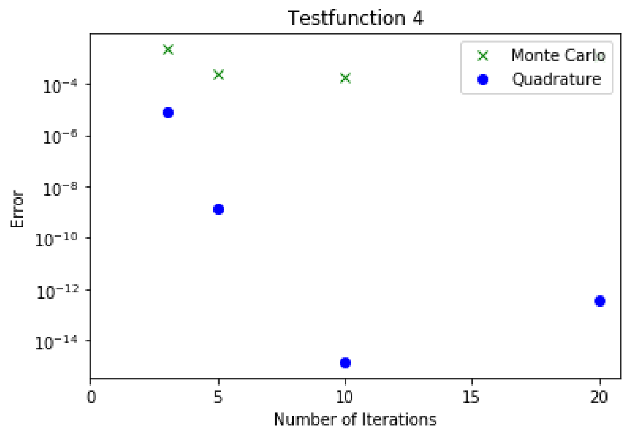
(a) $g_1(x,y) = \cos(x)\cos(y)$



(b) $g_2(x,y) = e^{-1/2x} \cdot e^{-1/2y}$



(c) $g_3(x,y,z) = \sin(x)\sin(y)\sin(z)$



(d) $g_4(x,y,z) = e^{-1/3x} \cdot e^{-1/3y} \cdot e^{-1/3z}$

Figure 4: Error per Number of Iterations

While numerical quadrature has an error which scales depending on the dimension, two or three variable functions don't appear to make the error worse than what it is for Monte Carlo. However, neither of these methods is asymptotically optimal. Therefore, it seems plausible that the theory prediction does not apply to them.

6.2.4 Composite Numerical Quadrature Implementation

Recall that composite numerical quadrature is given by

$$Q_{\mathscr{X},k}(f) = S(f_{\mathscr{X},k}) = \sum_i^k \int_{T^i(G)} f_{T^i(\mathscr{X})}(x)dx,$$

where $f_{T^i(\mathscr{X})}(x)$ is the Lagrange polynomial evaluated at x, with nodes $x \in \mathscr{X}$ in the area $T^i(G)$. The implementation of composite numerical quadrature splits the integration area $[0,1]^d$ in k smaller hyper cube shaped areas, represented in the previous equation with $T^i(G)$, and then uses the implementation of numerical quadrature in 6.2.1 to get an approximation of the integral over each of these areas. These are then summed up in the final step.

6.2.5 Control Variates Implementation

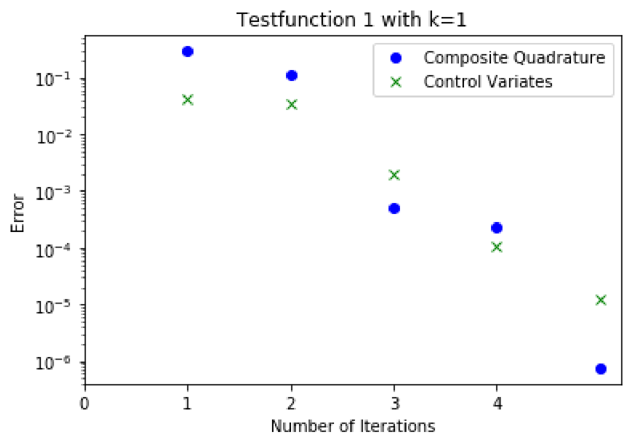
In section 5.2.2 we described control variates, utilizing composite interpolation with

$$M_{\mathscr{X},k}(f) = S(f_{\mathscr{X},k}) + \frac{1}{k^d} \cdot \sum_{i=0}^{k^d} (f - f_{\mathscr{X},k})(X_i),$$

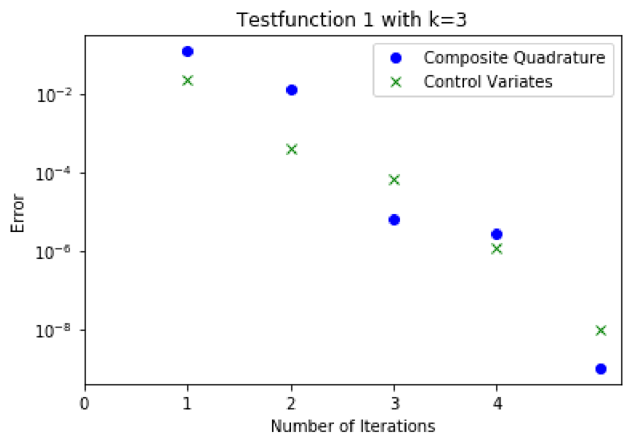
where $f_{\mathscr{X},k}$ is the composite Lagrange polynomial on the area $G = [0,1]^d$ with k subareas and X_i are equally distributed on G.
For the control variates implementation, an additional function was needed, which computes the composite Lagrange polynomial for a subarea of G and evaluates it at a point x. The control variates generates the k^d random values and makes sure they get summed up according to which subarea of G they are in.

6.2.6 Results of Composite Numerical Quadrature and Control Variates

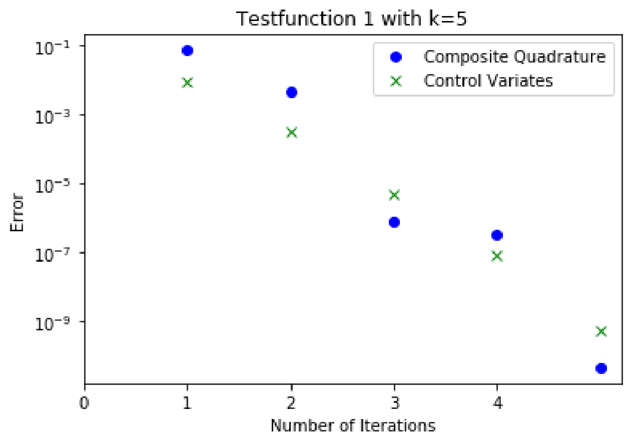
The plots show the error per iteration for each of the test functions separately. This time the integration area is split into k smaller areas, creating a second variable k. To show better how the accuracy develops for each function with varying k, we look at the plots of one function at a time.



(a) $k = 1$



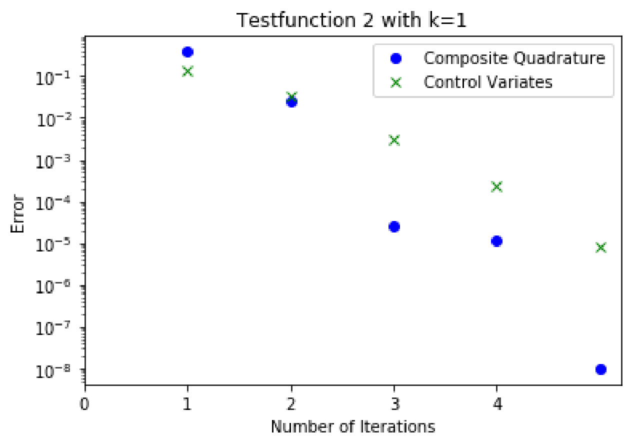
(b) $k = 3$



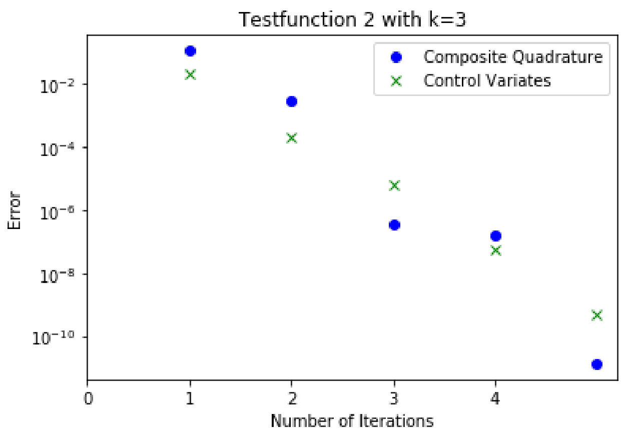
(c) $k = 5$

Figure 5: Plots for Test Function $g_1(x, y) = \cos(x)\cos(y)$: Error per Number of Iterations.

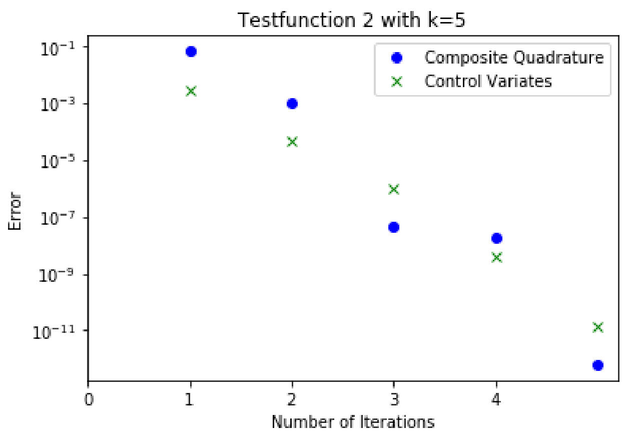
The error converges with very few iterations for both control variates and composite quadrature. This is actually a pattern we see for the following test functions as well.



(a) $k = 1$

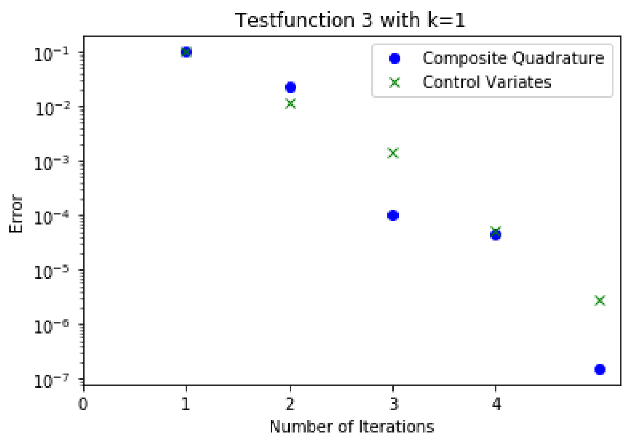


(b) $k = 3$

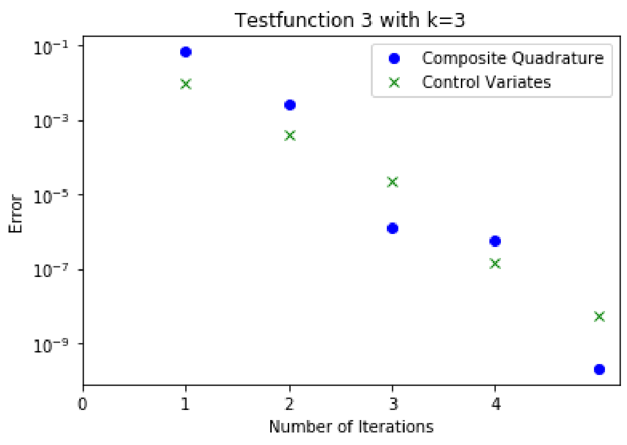


(c) $k = 5$

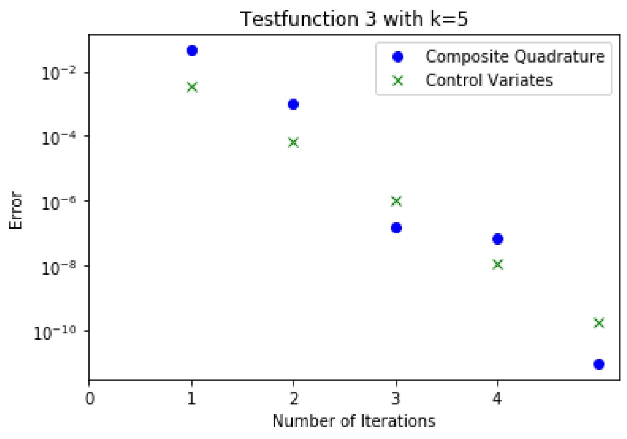
Figure 6: Plots for Test Function $g_2(x, y) = e^{-1/2x} \cdot e^{-1/2y}$: Error per Number of Iterations.



(a) $k = 1$

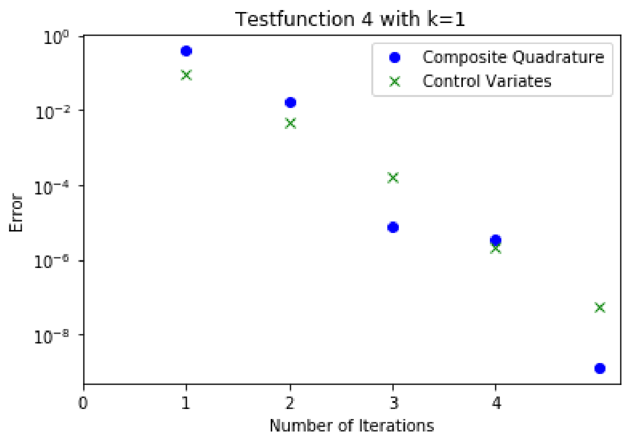


(b) $k = 3$

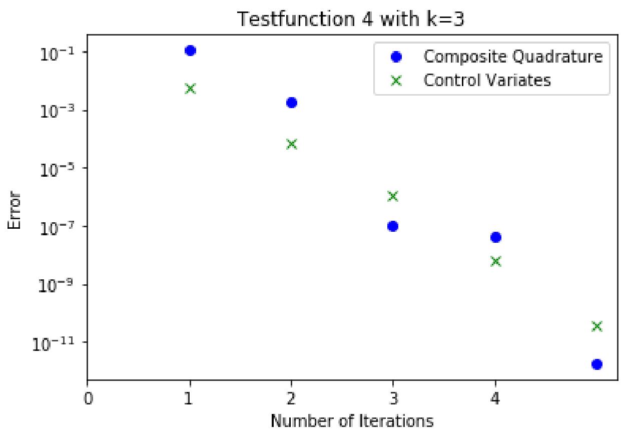


(c) $k = 5$

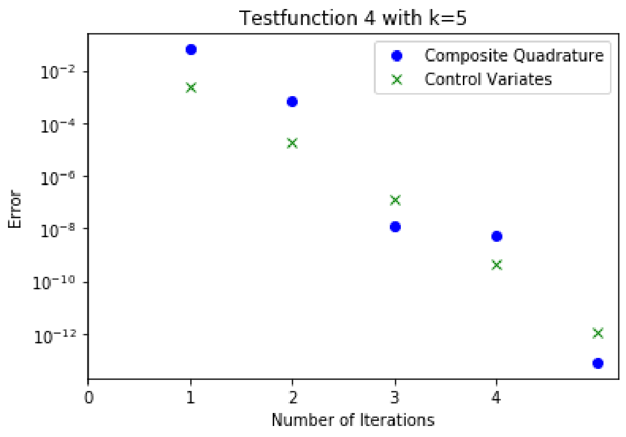
Figure 7: Plots for Test Function $g_3(x,y,z) = \sin(x)\sin(y)\sin(z)$: Error per Number of Iterations.



(a) $k = 1$



(b) $k = 3$



(c) $k = 5$

Figure 8: Plots for Test Function $g_4(x, y, z) = e^{-1/3x} \cdot e^{-1/3y} \cdot e^{-1/3z}$: Error per Number of Iterations.

As for the plots corresponding to the first test function, the plots for the remaining test functions also show fast convergence and small error already for a small amount of iterations. It can also be noted that the difference in error between composite numerical quadrature and control variates varies, both methods obtaining smaller and bigger errors than the other. Therefore, it is not possible to say that this exactly confirms the theoretical statement. However, the theoretical statement is about the asymptotic n-th minimal error, which means there is room for variation in the definition. It also has to be said that since the control variates method's output is a random variable, the error of it is as well. We use the expectation of the error and approximate it with the mean over several runs, to get the value displayed in the plots. Consequently, the values bigger than the deterministic approach could be because the random values generated were in this case ending up above the expected value.

7 Conclusion and Outlook

In section 5 we have shown how the n-th minimal error of the deterministic approach and the randomized approach compare to each other. We showed that for both $f \in F_d^r$, where r is the smoothness and d the dimension, the n-th minimal error of Monte Carlo integration based methods converges faster towards zero as n goes towards infinity. When putting these results to the test, the experiments for one dimensional functions opposed this result. Achieving the same accuracy with Random Riemann sums as with the mid point rule takes far too many iterations. The most possible cause of this is the asymptotic nature of the result for the n-th minimal error. Based on the tests made for one variable functions, it is clear that the mid point rule is a better suited option. This also confirms our expectations from the introduction based on [5] and [8]. For two and three variable test functions, these results were not as conclusive as it varies which method produces a smaller error. This can be due to several factors: On one hand, we stated in section 5 that the randomized algorithms are especially favoured, when the smoothness r of a function is very low and the dimension d is very high. The test functions used in section 6.2 all have high smoothness r, decreasing the error caused by the curse of dimensionality. On the other hand, two or three might not be a high enough dimension to cause a significant difference in error between the two methods.

While this project gave a basic idea of how the methods, given a specific situation, of the two approaches perform, this project is far from making a conclusive comparison. There are many other optimization strategies for Monte Carlo integration, which could be investigated as well. The same holds respectively for numerical quadrature. The asymptotically optimal algorithm this project touched upon, control variates, depends on the result of composite numerical quadrature. Proving that combining the methods, gives composite numerical quadrature a worthy competitor. Furthermore, this project solely focused on rate of convergence to make a comparison. Maybe focusing purely on a worst case analysis with maximum cost and maximum error, would give a different result.

All these things considered, this barely scratches the surface and there is much left to look at to make a conclusive comparison between the two approaches. Or maybe it is as Robert and Casella[8, p. 22-23] say: "However, given the dependence on specific problem characteristics, it is fruitless to advocate the superiority of one method over the other, say of the simulation-based approach over numerical methods. Rather, it seems more reasonable to justify the use of simulation-based methods by the statistician in terms of *expertise*. The intuition acquired by a statistician in his or her everyday processing of random models can be directly exploited in the implementation of simulation techniques (...), while purely numerical techniques rely on less familiar branches of mathematics." In which case a comparison would better be conducted by a statistician with more "expertise". It is to note that they conclude with suggesting that a combination of the two perspectives often produces a "desireable approach", which based on the small error of the control variates in section 6, we can agree with.

8 Code Appendix

For all implementations the following imports were made

```python
from scipy.integrate import quad
import numpy as np
import random
```

8.1 One Dimensional Implementations

These are corresponding to the descriptions in 6.1.1.

8.1.1 Mid Point Equation

```python
def  Midpoint(f,n):
    """
    Parameters
    ----------
    f : function
        to be integrated
    n : integer
        #subintervals

    Returns
    -------
    float
        Approximative Solution for the integral of f.

    """
    deltax=1/n
    l=[i/n for i in range(0,n+1)]
    m=[(l[j]+l[j-1])/2 for j in range(1,len(l))]
    s=sum(f(i) for i in m)
    return s/n
```

8.1.2 Random Riemann Sum

```python
def Riemann(f,n):
    """
    Parameters
    ----------
    f : function
        to be integrated
    n : integer
        #subintervals

    Returns
    -------
    R : float
        Approximative Solution for the integral of f.

    """
    X=random.sample(list(np.linspace(0,1,1000000)), n)
    R=1/n*sum(f(x) for x in X)
    return R
```

8.2 Three and Four Dimensional Implementations

These correspond to the implementations described in 6.2.

8.2.1 Numerical Quadrature

For the implementation of numerical quadrature, first a function is needed that computes a Lagrange basis polynomial L_i at a given point.

```python
def L(x,n,i,A,B):
    """
    Generates Lagrange Basis Polynomial L_i(x) on interval [A,B]
    with n nodes.

    Parameters
    ----------
    x : float
        point of evaluation
    n : integer
        #interpolation nodes
    i : integer
        index of basis polynomial
    A : float
        lower boundary of interpolation interval
    B : float
        upper boundary of interpolation interval

    Returns
    -------
    lx : float
        Evaluation of the Lagrange Basis Polynomial L_i at point x.

    """
    X=np.linspace(A,B,n)
    k=0
    lx=1
    while k<n:
        if k == i:
            lx=lx
        else:
            l=(x-X[k])/(X[i]-X[k])
            lx=lx*l
        k+=1
    return lx
```

This will then be used in the numerical quadrature implementation.

```
def Quadrature(f,n,d,a,b):
    """
    Approximates the interval of f based on Numerical quadrature on an
    integration cube [ax,bx],[ay,by],*[az,bz].
    Parameters
    ----------
    f : function
        Function with 2 or 3 variables
    n : integer
        Amount of Interpolationpoints
    d : integer
        dimension i.e. 2 variables, d=3.
    a : list
        lower boundaries of interpolation [ax, ay, *az]
    b : list
        upper boundaries of interpolation [bx, by, *bz]

    Returns
    -------
    float
        Integral approximation of f on hypercube
    """
    X=np.linspace(a[0],b[0],n)
    Y=np.linspace(a[1],b[1],n)
    Q=0
    if d==3:
        i=0
        while i<n:
            j=0
            while j<n:
                Q+=quad(L,a[0],b[0],args=(n,i,a[0],b[0]))[0]*quad(L,a[1],b[1],args=(n,j
                j+=1
            i+=1
    elif d==4:
        Z=np.linspace(a[2],b[2],n)
        i=0
        while i<n:
            j=0
            while j<n:
                h=0
                while h<n:
                    Q+=quad(L,a[0],b[0],args=(n,i,a[0],b[0]))[0]*quad(L,a[1],b[1],args=
                    h+=1
                j+=1
            i+=1
    else:
        return "Please enter dimension 3 or 4."
    return Q
```

8.2.2 Classic Monte Carlo

```python
def MonteCarlo(f,n,d):
    "How do i generate a sample with a known average??"
    X=random.sample(list(np.linspace(0,1,10000)), n*(d-1))
    i=0
    S=0
    if d==3:
        while i<n*(d-1)-1:
            S+=f(X[i],X[i+1])
            i+=2
    elif d==4:
        while i<n*(d-1)-2:
            S+=f(X[i],X[i+1],X[i+2])
            i+=3
    else:
        return "Please enter dimension 3 or 4"
    return S/n
```

8.2.3 Composite Numerical Quadrature

```python
def CompositeQuadrature(f,n,d,k):
    """

    Parameters
    ----------
    f : function
        to be approximated (2 or 3 variables)
    n : integer
        #interpolation nodes
    d : integer
        dimension, 3 or 4
    k : integer
        #subintervals of [0,1]

    Returns
    -------
    float
        Approximated solution of integral of f over [0,1].

    """
    Q=0
    if d==3:
        for j in range(0,k):
            ax,bx=j/k,(j+1)/k
            for m in range(0,k):
                ay,by=m/k,(m+1)/k
                a,b=[ax,ay],[bx,by]
                Q+=Quadrature(f,n,d,a,b)
        return Q

    elif d==4:
        for j in range(0,k):
            ax,bx=j/k,(j+1)/k
            for m in range(0,k):
                ay,by=m/k,(m+1)/k
                for h in range(0,k):
                    az,bz=h/k,(h+1)/k
                    a,b=[ax,ay,az],[bx,by,bz]
                    Q+=Quadrature(f,n,d,a,b)
        return Q
    else:
      return "Please enter dimension 3 or 4."
```

8.2.4 Control Variates

This needs a function, which evaluates a composite Lagrange approximation polynomial at a given point, which in itself needed a classic Lagrange polynomial function.

```
def LagrangeInterpolation(f,n,d,x,a,b):
    """
    Creates the Lagrange interpolation of a function f with n nodes
    on a hypercube [ax,bx],[ay,by],*[az,bz].

    Parameters
    ----------
    f : function
        to be interpolated
    n : integer
        #interpolation nodes
    d : integer
        dimension
    x : list
        coordinates of evaluation
    a : list
        lower boundaries of interpolation [ax, ay, *az]
    b : list
        upper boundaries of interpolation [bx, by, *bz]

    Returns
    -------
    float
        Evaluation of the Lagrange Interpolation Polynomial
        at point x.

    """
    K=np.linspace(a[0],b[0],n)
    M=np.linspace(a[1],b[1],n)
    P=0
    if d==3:
        i=0
        while i<n:
            j=0
            while j<n:
                P+=f(K[i],M[j])*L(x[0],n,i,a[0],b[0])*L(x[1],n,j,a[1],b[1])
                j+=1
            i+=1
    elif d==4:
        H=np.linspace(a[2],b[2],n)
        i=0
        while i<n:
            j=0
            while j<n:
                h=0
                while h<n:
                    P+=(f(K[i],M[j],H[h])*L(x[0],n,i,a[0],b[0])*L(x[1],n,j,a[1],b[1])*L
                    h+=1
                j+=1
            i+=1
    else:
        return "Please enter dimension 3 or 4."
    return P


def CompositeLagrangeInterpolation(f,n,d,k,x):
    """
    Parameters
    ----------
    f : function
        to be approximated (2 or 3 variables)
    n : integer
        #interpolation nodes
    d : integer
        dimension, 3 or 4
    k : integer
        #subintervals of [0,1]
    x : list
        coordinates of function evaluation

    Returns
    -------
    float
        Evaluation of the Composite Lagrange Interpolation Polynomial at
        point x.

    """
    if d==3:
        for j in range(0,k):
            ai,bi=j/k,(j+1)/k
            if ai<=x[0]<=bi:
                ax,bx=ai,bi
            if ai<=x[1]<=bi:
                ay,by=ai,bi
        return LagrangeInterpolation(f,n,d,x,[ax,ay],[bx,by])
    elif d==4:
        for j in range(0,k):
            ai,bi=j/k,(j+1)/k
            if ai<=x[0]<=bi:
                ax,bx=ai,bi
            if ai<=x[1]<=bi:
                ay,by=ai,bi
            if ai<=x[2]<=bi:
                az,bz=ai,bi
        return LagrangeInterpolation(f,n,d,x,[ax,ay,az],[bx,by,bz])
    else:
        return 'Please enter dimension 3 or 4'
```

This is then used in the "Control Variates" function.

```
def ControlVariates(f,k,n,d):
    """
    Approximates the interval of f based on Control Variates on an
    integration cube [ax,bx],[ay,by],*[az,bz]

    Parameters
    ----------
    f : function
        to be approximated (2 or 3 variables)
    n : integer
        #interpolation nodes
    d : integer
        dimension, 3 or 4
    k : integer
        #subintervals of [0,1]

    Returns
    -------
    CV : float
        approximate integral solution

    """
    X=random.sample(list(np.linspace(0,1,10000)), k**(d-1))
    Y=random.sample(list(np.linspace(0,1,10000)), k**(d-1))
    Z=random.sample(list(np.linspace(0,1,10000)), k**(d-1))
    Q=CompositeQuadrature(f,n,d,k)
    S=0
    if d==3:
        i=0
        while i<k**(d-1):
            S+=f(X[i],Y[i])-CompositeLagrangeInterpolation(f,n,d,k,[X[i],Y[i]])
            i+=1
    elif d==4:
        i=0
        while i<k**(d-1):
            S+=f(X[i],Y[i],Z[i])-CompositeLagrangeInterpolation(f,n,d,k,[X[i],Y[i],Z[i]
            i+=1
    CV=Q+(1/(k**(d-1)))*S
    return CV
```

9 References

[1] D. Anevski. *A Concise Introduction to Mathematical Statistics*, chapter 10. Studentlitteratur, 2017.

[2] W. Cheney and W. Light. *A Course in Approximation Theory*, chapter 10. Brooks/Cole, 2000.

[3] J.M. Hammersley and D.C. Handscomb. *Monte Carlo Methods*, chapter 1. Methuen Co LTD, 1975.

[4] M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods*, chapter 1. Wiley-VCH Verlag GmbH Co., 2008.

[5] T. Müller-Gronbach, E. Novak, and K. Ritter. *Monte Carlo Algorithmen*, chapter 2,5,7. Springer Verlag, 2012.

[6] E. Novak and K. Ritter. The curse of dimension and a universal method for numerical integration. In J.W. Schmidt G. Nürnberger and G. Walz, editors, *Multivariate Approximation and Splines*, pages 177–188. Birkhäuser, Basel, 1997.

[7] C.P. Robert and G. Casella. *Monte Carlo Statistical Methods*, chapter 1. Springer, 2004.

[8] C.P. Robert and G. Casella. *Introducing Monte Carlo Methods with R*, chapter 3. Springer, 2010.