SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Formalization of Selected Integral Approximation Algorithms

Christian Philip Zimmerer

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Formalization of Selected Integral Approximation Algorithms

Formalisierung ausgewählter Algorithmen zur Integralapproximation

Author:	Christian Philip Zimmerer
Supervisor:	Tobias Nipkow
Advisor:	Katharina Kreuzer
Submission Date:	16.08.2023

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 16.08.2023 Christian Philip Zimmerer

Acknowledgments

Throughout this thesis, the guidance, feedback and insights provided by my advisor Katharina Kreuzer have been particularly valuable. Her support was a great contribution to the writing process and definitely helped to spark my motivation for further work in the field of automated theorem proving. Additionally, I greatly appreciate the input of my fellow researchers regarding their related work. Especially Prof. Bas Spitters has provided lightning-quick responses to all of my questions despite being on vacation. Furthermore, I would like to thank all my friends who have proofread the paper, including Julia Spindler, Lukas Niekerke, Viktor Boskovski, Tobias Kamm, Nicole Kettler, Valentin Metz and Henrik Tscherny. Also, the proving process would have taken me a lot longer without the hints from Alexandra Graß regarding Isabelle. Finally, a special thanks goes out to my father Peter Zimmerer, who has provided me with delicious snacks and smoothies and ensured that I could dedicate my entire brain capacity to this thesis.

Abstract

Numerical integration algorithms are vital tools with many applications because they can quickly provide relatively accurate approximations to integrals with challenging analytical solutions. This thesis formally proves the error bounds of the composite midpoint rule and the composite Cavalieri-Simpson rule for numerical integration in Isabelle/HOL. Furthermore, it presents an executable implementation of both rules complemented by a numerical analysis, laying the foundation for extending the proof procedures described by Hölzl [10] to inequations containing integrals.

Kurzfassung

Numerische Integrationsalgorithmen sind essentielle Werkzeuge mit vielen Anwendungen, da sie Integrale mit schwierigen analytischen Lösungen in kurzer Zeit relativ genau berechnen können. In dieser Arbeit werden die Fehlerabschätzungen der zusammengesetzten Mittelpunktregel und der zusammengesetzten Cavalieri-Simpson-Regel zur numerischen Integration in Isabelle/HOL formal bewiesen. Außerdem wird eine ausführbare Implementierung der beiden Regeln präsentiert und mit einer numerischen Analyse ergänzt, um den Grundstein für die Erweiterung der durch Hölzl beschriebenen Beweistaktik [10] auf Ungleichungen mit Integralen zu legen.

Contents

1 Introduction

Numerical integration is a practical and indispensable tool in various fields, facilitating the approximation of definite integrals and enabling the solution of complex mathematical problems. While analytical methods offer elegance and exactness in numerous cases, some integrals prove to be challenging or even impossible to solve analytically.

As an example, consider the cumulative density function of the standard normal distribution:

Definition 1.1 (Standard Normal Distribution, [2, p. 112f])

$$\Phi(x) = \int_{-\infty}^{x} \varphi(x)dx \text{ where } \varphi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{-x^2}{2}\right)$$

Attempting to find an analytical solution to this integral presents considerable hurdles because the antiderivative cannot be represented using only elementary functions. However, in most practical use cases, the precision of analytical methods is not required and an approximation of the result suffices. Therefore, by leveraging numerical techniques, integrals with challenging analytical solutions can be handled effectively.

Proofs are the most crucial building block of the mathematical world. The process of building them relies on the foundational framework provided by previously established proofs. However, just one human error in any part of the structure can be catastrophic as it requires all subsequent work to be repaired. As a result, Interactive Theorem Provers (ITPs), such as Isabelle [17] or Coq [1], have been developed. They provide a formal environment for expressing logical statements and their proofs so that logical fallacies can be detected algorithmically, ensuring the validity of mathematical results.

To efficiently prove inequalities over real numbers in Isabelle, Hölzl developed a framework for translating inequalities into interval arithmetic calculations on arbitrary-precision floating point numbers and evaluating them using the code generator [10]. In his work, he utilized the reification framework created by Chaieb [3] to implement a proof procedure by mapping arithmetic expressions in Isabelle to the newly introduced `floatarith` datatype and then evaluating them with an approximation function up to

a specified floating point precision. The numerical error is included in the calculation by returning an upper and lower bound for the expression instead of a single value.

Isabelle already contains a large collection of theorems regarding integration theory. However, no public formalizations of numerical quadrature exist in Isabelle at the time of writing this thesis. As a consequence, proving inequalities between integrals currently almost always involves their analytical resolution with all possible intricacies. Adding numerical integration techniques to the Isabelle framework could drastically accelerate proof development in some cases. Similar implementations have already been designed for the Coq theorem prover as both a standalone theory by Mahboubi et al. [14] and as a stepstone for the implementation of a solver for ordinary differential equations by Makarov and Spitters [15].

This thesis aims to lay the foundation for extending the numerical proof tactic to support inequalities on integrals by implementing two algorithms for integral approximation using Hölzl's framework. Visualized in Figure 1.1, the first implemented quadrature rule is the composite midpoint rule [20, p. 50], which is the intuitive rule that comes to mind from the definition of the Riemann integral [8, p. 214]. A more precise approximation may be obtained using the implementation of the composite Cavalieri-Simpson rule [20, p. 53].

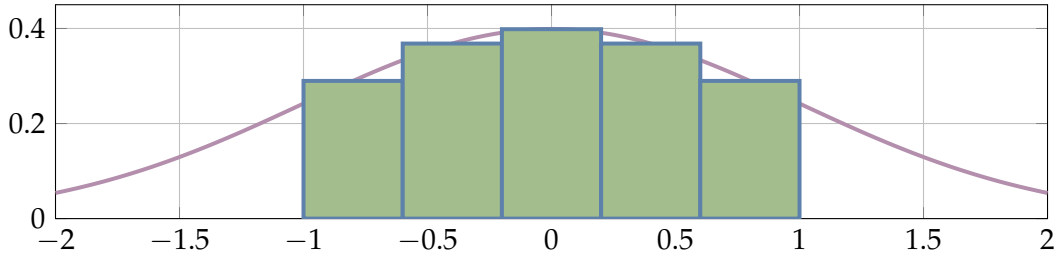


Figure 1.1: Approximation of $\int_{-1}^{1} \varphi(x)dx$ using the Composite Midpoint Rule

To determine the usefulness of both approximations, an extensive error analysis is carried out. In Chapter 2, the implemented formulas are derived from their mathematical origins, leading to two extensive proofs of both analytical error bounds. To support their validity, a formal verification in Isabelle/HOL is provided. Chapter 3 provides a comprehensive overview on the technical details, choices and problems of the formalization process. Afterwards, the implementation-specific numerical properties are examined in Chapter 4, including theoretical considerations about condition and stability as well as an evaluation of the practical algorithmic results. Finally, the outcomes are summarized and an outlook on future work is given in Chapter 5.

The entire Isabelle source code is available on GitHub under [23].

1.1 Notation

Notation	Semantics
(a, b)	Tuple
$(a; b)$	Open interval
$[a; b]$	$(a; b) \cup \{a, b\}$
$[a; b)$	$(a; b) \cup \{a\}$
$\mathbb{N}$ or $\mathbb{N}_0$	Set of nonnegative integers
$\mathbb{N}_+$	Set of positive integers
$[n]$	$\{0, 1, 2, ..., n\}$
$\mathbb{Z}$	Set of integers
$\mathbb{R}$	Set of real numbers
$\mathbb{P}^r$	Set of polynomials of degree r
$\mathbb{F}$	Set of infinite-precision floating point numbers, Definition 4.5
$\mathbb{F}_p$	Set of p-precise floating point numbers, Definition 4.7
$\mathbb{A}^n$	Set of n-tuples over $\mathbb{A}$
$\lim_{x^+ \to y} f(x)$	right limit of $f(x)$ approaching y
$\lim_{x^- \to y} f(x)$	left limit of $f(x)$ approaching y
$I(f, a, b)$	Integral of f on $[a; b]$
$\left[F(x)\right]_a^b$	$F(b) - F(a)$
$f'(x)$	Derivative of f
$f^{(n)}(x)$	n-th derivative of f
$f \in C_A^n$	f is n times continuously differentiable on A
$\varphi(x)$	Probability Density Function of the standard normal distribution
$I_n(f, a, b)$	Integral approximation of f on $[a; b]$
$E_{I_n}(a, b)$	Quadrature error
$L_n(x)$	Lagrange polynomial
$N_n(f, a, b)$	Newton-Cotes rule
$N_n(f, a, b)_c$	Closed Newton-Cotes rule
$N_n(f, a, b)_o$	Open Newton-Cotes rule
$M(f, a, b)$	Midpoint rule
$S(f, a, b)$	Cavalieri-Simpson rule
$I_{n,N}(f, a, b)$	Composite integral approximation of f on $[a; b]$ with N subintervals
$M_N(f, a, b)$	Composite midpoint rule with N subintervals
$S_N(f, a, b)$	Composite Cavalieri-Simpson rule with N subintervals

2 Numerical Integration

In this section, the core principles of numerical integration are introduced. The two formalized quadrature rules are presented as members of the family of Newton-Cotes formulas, including detailed proofs of their error bounds. Further mathematical details can be obtained from [20].

2.1 Integral Approximation

There are various approaches to numerically obtaining the quadrature of a function. The most common approach for one-dimensional integration problems is to not integrate the original function, but to interpolate it first and then integrate the result.

Definition 2.1 (Integral Approximation by Integrand Approximation, [20, p. 47])
Let $f : \mathbb{R} \to \mathbb{R}$ be the integrand and let $I(f,a,b) = \int_a^b f(x)dx$ the desired integral. By computing an approximation $f_n(x)$ of $f(x)$ that depends on $n \in \mathbb{N}^+$ and subsequently computing its integral, an approximation of the original integral is defined as

$$I_n(f,a,b) = I(f_n,a,b) = \int_a^b f_n(x)dx$$

The absolute quadrature error is defined as $|E_{I_n}(a,b)| = |I(f,a,b) - I_n(f,a,b)|$.

Remark 2.2 ([20, p. 48])
For a practically computable approximation, $f_n(x)$ should be trivially integrable. This is especially the case if it is a polynomial.

2.2 Newton-Cotes Formulas

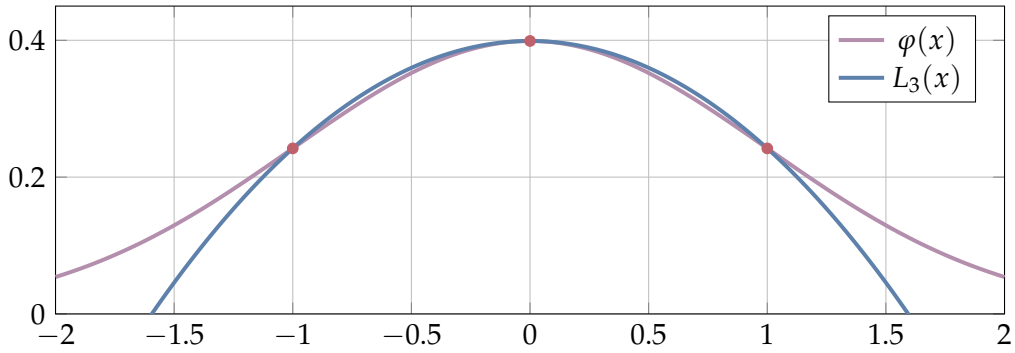
To approximate the integrand, it is helpful to consider a limited set of function values from points inside the interval and then generate an approximation based on these

values. One of the easiest and most popular methods is using the Lagrange polynomials.

Definition 2.3 (Lagrange Polynomial [20, equation 8.3f])
Given $n+1$ points (x_i) where $i \in [n]$, the unique Lagrange polynomial on these points is defined as

$$L_n(x) = \sum_{i=0}^{n} f(x_i) l_i(x) \text{ where } l_i(x) = \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \right).$$

Example 2.4
The Lagrange Polynomial of $\varphi(x)$ on $(-1, 0, 1)$ is graphed below, including the interpolation points.



Remark 2.5
Integration yields

$$\int_a^b L_n(x)dx = \int_a^b \sum_{i=0}^{n} f(x_i) l_i(x) dx = \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x) dx$$

Since the distinct values l_i only depend on the values x_i, but not on f, their definite integral

$$\omega_i = \int_a^b l_i(x) dx$$

from a to b can be computed just after selecting the points, which yields the following

family of interpolation formulas:

$$I_n(f) = \sum_{i=0}^{n} f(x_i)\omega_i, n \in \mathbb{N}$$

The number and location of interpolation points can be arbitrarily chosen. Intuitively, it seems reasonable to pick them inside $[a, b]$ in order to obtain a good approximation function. Selecting equidistant points is the most obvious procedure and leads to the following family of quadrature rules.

Definition 2.6 (Newton-Cotes Formula, [20, p. 54f])

Let x_0 be a chosen starting point, and let $x_n = x_0 + nh$ where $h > 0$ corresponds to the distance between two neighbours x_n and x_{n+1}. The integral

$$N_n(f, a, b) = \int_a^b L_n(x)dx$$

of the Lagrange polynomial on these points is then called a Newton-Cotes formula of degree n.

The parameters (h, x_0) are chosen from either

$$\left(\frac{b-a}{n}, a\right) \text{ or } \left(\frac{b-a}{n+2}, a+h\right).$$

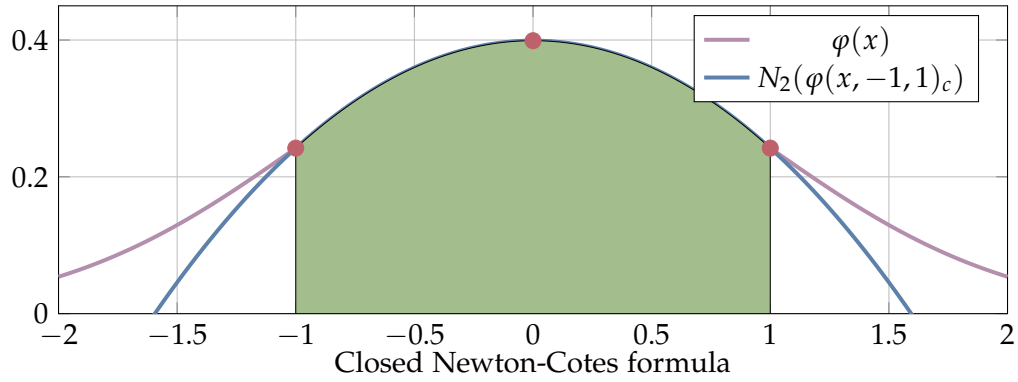
The formula is called closed in the first case and open in the second case, denoted as $N_n(f, a, b)_o$ and $N_n(f, a, b)_c$ respectively.

Example 2.7

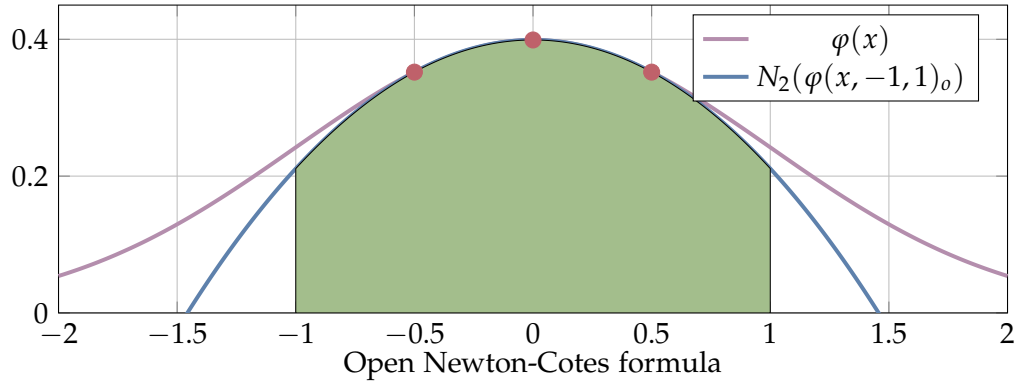
The difference between open and closed Newton-Cotes formulas is visualised in the following graphs. Both diagrams show the approximation of

$$\int_a^b \varphi(x)dx$$

by the respective Newton-Cotes formula of degree 2.



Closed Newton-Cotes formula



Open Newton-Cotes formula

Remark 2.8 ([20, p. 55])

Although the interpolation points are defined using the integration interval, it can be shown that the Newton-Cotes formulas can also be expressed as:

$$N_n(f,a,b) = h\sum_{i=0}^{n} f(x_i)\int_m^l \prod_{k=0,k\neq i}^{n} \frac{t-k}{i-k}dt$$

where (m,l) is $(0,n)$ in closed and $(-1,n+1)$ in open formulas. It is noticeable that the integrals only depend on n, allowing them to be precomputed and thus making the implementation of the Newton-Cotes formulas very straightforward.

Usually, rules with relatively small n are used because with large n, the Lagrange interpolation polynomial may start oscillating drastically, resulting in a huge loss of precision. This is known as Runge's phenomenon [20, p. 5f]. As the following definition shows, this problem transfers to the Newton-Cotes formulas as well.

Definition 2.9 (Order of an Integral Approximation, [20, p. 56])
The order of a quadrature formula $I_n(f,a,b)$ is defined as

$$\max\left\{p \in \mathbb{Z} \middle| |I(f,a,b) - I_n(f,a,b)| = \mathcal{O}\left((b-a)^p\right)\right\}$$

In other words, the order specifies the asymptotic growth of the error bound with regard to the interval width.

Remark 2.10 ([20, p. 56f])
The order of a Newton-Cotes formula of degree n is $n+3$ for even and $n+2$ for odd n. It is easy to see that as n grows, the asymtotic error bound increases exponentially.

Two commonly used rules are now analyzed in further detail.

Definition 2.11 (Midpoint Rule, [20, p. 49])
At $n = 0$, the open Newton-Cotes formula simplifies to the midpoint rule:

$$M(f,a,b) = (b-a)f\left(\frac{a+b}{2}\right)$$

Derivation:

$$I_0(f,a,b) = hf(x_i)\int_{-1}^{1} 1dt = \frac{b-a}{2}f(x_i)\cdot 2 = (b-a)f\left(\frac{a+b}{2}\right) = M(f,a,b)$$

As the name suggests, this formula uses the midpoint of $[a;b]$ as its only sampling point. Although it is simple to compute, its major downside is its suboptimal approximation quality compared to other procedures. An error estimate will be proven later to underline this fact, but for now, the degree of exactness is introduced to provide a superficial justification.

Definition 2.12 (Degree of Exactness, [20, p. 49])
Let $\mathbb{P}^r$ be the set of polynomials of degree r. The degree of exactness of an integral

approximation is defined as

$$\max \left\{ r \in \mathbb{N}_0 \middle| I_n(f,a,b) = I(f,a,b) \ \forall f \in \mathbb{P}^r \right\}$$

In other words, r is the highest degree of polynomials whose definite integral can be exactly calculated by the approximation.

Remark 2.13 ([20, p. 56f])

Newton-Cotes rules with even n are exact up to degree $n+1$, while closed Newton-Cotes rules with odd n are exact up to degree n. Subsequently, the midpoint rule is exact up to degree 1.

To achieve a higher exactness, three Lagrange interpolation points can be used to generate a polynomial of degree 2, resulting in the following approximation:

Definition 2.14 (Cavalieri-Simpson Rule, [20, p. 53])

At $n = 2$, the closed Newton-Cotes formula simplifies to the Cavalieri-Simpson rule:

$$S(f,a,b) = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

Derivation:

$$\begin{aligned}
&I_2(f,a,b)\\
=&h\left(f(a)\int_0^2 (t-1)\frac{(t-2)}{2}dt + f(a+h)\int_0^2 -t(t-2)dt + f(a+2h)\int_0^2 \frac{t(t-1)}{2}dt\right)\\
=&h\left(\frac{f(a)}{2}\left[\frac{1}{3}t^3 - \frac{3}{2}t^2 + 2t\right]_0^2 - f(a+h)\left[\frac{1}{3}t^3 - \frac{3}{2}t^2 + 2t\right]_0^2 + \frac{f(b)}{2}\left[\frac{1}{3}t^3 - \frac{1}{2}t^2\right]_0^2\right)\\
=&\frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) = S(f,a,b)
\end{aligned}$$

Remark 2.15

Contrary to the midpoint rule, the Cavalieri-Simpson rule requires three function

evaluations instead of one, which appears to be a tradeoff between speed and accuracy at first glance. However, this conclusion proves to be a fallacy as the superior precision of the Cavalieri-Simpson rule compensates all speed losses when used in a composite rule. More information will be provided in Example 2.30 and in Section 4.4.

2.2.1 Error Bounds

While exactness and order are useful metrics to determine the precision of a numerical integration rule, they do not paint the full picture. A precise error bound for a given integration interval is required in most cases to determine the usefulness of an approximation. In this subsection, only the analytical error of both rules is analyzed.

Theorem 2.16 (Midpoint Rule Error Bound, [20, p. 49])
Assuming that $f \in C^2_{[a;b]}$, the maximum quadrature error of the midpoint rule is equal to:

$$|E_M(a,b)| \leq \frac{k(b-a)^3}{24} \text{ where } k \geq \max_{[a,b]} |f''(x)|$$

Remark 2.17
In the usual literature, k may be expressed implicitly as $f''(\xi)$. This notation is based on the Lagrange remainder form of Taylor's theorem [8, p. 284]. However, as this causes vital information about ξ to be omitted, the explicit form is used here.

Remark 2.18
It follows from Remark 2.10 that the order of the midpoint rule is 3. This can also be deducted directly from the error bound, which contains $(b-a)^3$ as a factor.

The general proof idea is inspired by Estepho [4].

Proof. The quadrature error of the midpoint rule is given by:

$$|I(f,a,b) - M(f,a,b)| = \left| \int_a^b f(x)dx - (b-a) f\left(\frac{a+b}{2}\right) \right|$$

To derive the error, the integral must be transformed so that the midpoint rule is cancelled out. It only appears logical to split the integral into two subintervals so that the midpoint (abbreviated by m) is a bound:

$$\int_a^b f(x)dx = \int_a^m f(x)dx + \int_m^b f(x)dx$$

The first subinterval can be rearranged by repeated integration by parts to yield the intermediate result (for which $f \in C^2$ is required):

$$\begin{aligned}\int_a^m f(x)dx &= \int_a^m 1 \cdot f(x)dx = \Big[(x+c_1)f(x)\Big]_a^m - \int_a^m (x+c_1)f'(x)dx \\ &= \Big[(x+c_1)f(x)\Big]_a^m - \Big[\left(\frac{(x+c_1)^2}{2} + c_2\right) f'(x)\Big]_a^m + \int_a^m \left(\frac{(x+c_1)^2}{2} + c_2\right) f''(x)dx\end{aligned}$$

The second subinterval is rearranged in the same way:

$$\int_m^b f(x)dx = \Big[(x+c_3)f(x)\Big]_m^b - \Big[\left(\frac{(x+c_3)^2}{2} + c_4\right) f'(x)\Big]_m^b + \int_m^b \left(\frac{(x+c_3)^2}{2} + c_4\right) f''(x)dx$$

The only term that contains f of both intermediate results is the first one, so the constants c_1 and c_3 need to be chosen in a way that they cancel out the midpoint rule:

$$\begin{aligned}&(m+c_1)f(m) - (a+c_1)f(a) + (b+c_3)f(b) - (m+c_3)f(m) \\ =&(c_1-c_3)f(m) - (a+c_1)f(a) + (b+c_3)f(b)\end{aligned}$$

Because the terms containing $f(a)$ and $f(b)$ will not cancel out with the midpoint rule, they need to be zero. Setting $c_1 = -a$, $c_2 = -b$ yields the desired term $(b-a)f(m)$.

The error should not contain f', so the respective second terms of the intermediate result need to cancel out.

$$\begin{aligned}&\Big[\left(\frac{(x-a)^2}{2} + c_2\right) f'(x)\Big]_a^m + \Big[\left(\frac{(x-b)^2}{+}c_4\right) f'(x)\Big]_m^b \\ =&\left(\frac{(m-a)^2}{2} + c_2\right) f'(m) - c_2 f'(a) + c_4 f'(b) - \left(\frac{(m-b)^2}{2} + c_4\right) f'(m) \\ =&\left(\frac{(m-a)^2 - (m-b)^2}{2} + c_2 - c_4\right) f'(m) - c_2 f'(a) + c_4 f'(b)\end{aligned}$$

Because $m - a = b - m$, the equation resolves to

$$(c_2 - c_4)f'(m) - c_2 f'(a) + c_4 f'(b) \stackrel{!}{=} 0$$

which is only valid for arbitrary f' if $c_2 = c_4 = 0$. By these results, it holds that

$$\begin{aligned}
|E_M(a,b)| &= \left| \int_a^m \frac{(x-a)^2}{2} f''(x)dx + \int_m^b \frac{(x-b)^2}{2} f''(x)dx \right| \\
&\leq \left| \int_a^m \frac{(x-a)^2}{2} f''(x)dx \right| + \left| \int_m^b \frac{(x-b)^2}{2} f''(x)dx \right| \\
&\leq \int_a^m \left| \frac{(x-a)^2}{2} f''(x) \right| dx + \int_m^b \left| \frac{(x-b)^2}{2} f''(x) \right| dx \\
&= \int_a^m \frac{(x-a)^2}{2} \left| f''(x) \right| dx + \int_m^b \frac{(x-b)^2}{2} \left| f''(x) \right| dx \\
&= \int_a^m \frac{(x-a)^2}{2} \left| f''(x) \right| dx + \int_m^b \frac{(x-b)^2}{2} \left| f''(x) \right| dx \\
\text{(with } f''(x) \leq k) &\leq \frac{k}{2} \left(\int_a^m (x-a)^2 dx + \int_m^b (x-b)^2 dx \right) \\
&= \frac{k}{2} \left(\left[\frac{(x-a)^3}{3} \right]_a^m + \left[\frac{(x-b)^3}{3} \right]_m^b \right) \\
&= \frac{k}{2} \left(\frac{(m-a)^3}{3} - \frac{(m-b)^3}{3} \right) = \frac{k}{6} \left((m-a)^3 - (m-b)^3 \right) \\
&= \frac{k}{6} \left(\left(\frac{b-a}{2} \right)^3 - \left(\frac{a-b}{2} \right)^3 \right) = \frac{k}{3} \left(\frac{b-a}{2} \right)^3 = \frac{k(b-a)^3}{24}
\end{aligned}$$

□

Theorem 2.19 (Cavalieri-Simpson Rule Error Bound)
Assuming that $f \in C^4_{[a;b]}$, the maximum quadrature error of the Cavalieri-Simpson

rule is equal to:

$$|E_S(a,b)| \leq \frac{k(b-a)^5}{90 \cdot 2^5} \text{ where } k \geq \max_{[a,b]} |f^{(4)}(x)|$$

Remark 2.20

In the literature, the error may sometimes be expressed in terms of $\frac{b-a}{2}$ instead of $b-a$. Hence, the denominator is notated in a way that the equivalence is clearly visible.

Remark 2.21

It follows from Remark 2.10 that the order of the Cavalieri-Simpson rule is 5. Just like for the midpoint rule, this can also be deducted from the appearance of the factor $(b-a)^5$ in the error bound.

The general proof idea is inspired by Estepho [5].

Proof. The integral is equal to the Cavalieri-Simpson rule plus the quadrature error:

$$\int_a^b f(x)dx = S_f + E_S = \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right) + E_s(a,b)$$

From the results of the previous proof, it is inferred that E_S may be expressed as a multiple of

$$\int_a^m p_1(x)f^{(4)}(x)dx + \int_m^b p_2(x)f^{(4)}(x)dx$$

where p is a polynomial of degree 4, so that repeated integration by parts yields a term containing $\int f(x)dx$. For the sake of notational brevity, it is assumed that the polynomial is monoquartic, i.e. $p_n(x) = 1x^4 + \mathcal{O}(x^3)$ (and thus $p_n^{(4)} = 24$). The reader

may verify that the following equality holds:

$$\begin{aligned}
&\int_a^m p_1(x)f^{(4)}(x)dx + \int_m^b p_2(x)f^{(4)}(x)dx \\
=&\left[f^{(3)}(x)p_1(x)\right]_a^m + \left[f^{(2)}(x)p_1^{(1)}(x)\right]_a^m + \left[f^{(1)}(x)p_1^{(2)}(x)\right]_a^m + \left[f(x)p_1^{(3)}(x)\right]_a^m \\
+&\left[f^{(3)}(x)p_2(x)\right]_m^b + \left[f^{(2)}(x)p_2^{(1)}(x)\right]_m^b + \left[f^{(1)}(x)p_2^{(2)}(x)\right]_m^b + \left[f(x)p_2^{(3)}(x)\right]_m^b \\
+&24\int_a^m f(x)dx + 24\int_m^b f(x)dx
\end{aligned}$$

As the Cavalieri-Simpson rule only contains terms dependent on $f(x)$, it follows that all terms that do not depend on it must cancel out. This follows if the following equations hold:

$$\begin{aligned}
p_1(a) = p_1^{(1)}(a) = p_1^{(2)}(a) = 0 \\
p_2(b) = p_2^{(1)}(b) = p_2^{(2)}(b) = 0 \\
p_1^{(k)}(m) = p_2^{(k)}(m); k \in \{0,1,2\}
\end{aligned}$$

The first two conditions are easily resolved by setting $p_1 = (x-a)^3(x+d_1)$ and $p_2 = (x-b)^3(x+d_2)$. The last condition yields a system of linear equations. It is safe to assume $(b-a) \neq 0$ (Otherwise, $E_S = S(f) = \int_a^b f(x)dx = 0$ $\square$). The first condition can be solved without any preliminary work:

$$\begin{aligned}
p_1(m) = \left(\frac{a+b}{2} - a\right)^3\left(\frac{a+b}{2} + d_1\right) &= \left(\frac{a+b}{2} - b\right)^3\left(\frac{a+b}{2} + d_2\right) = p_2(m) \\
\left(\frac{b-a}{2}\right)^3\left(\frac{a+b}{2} + d_1\right) &= -\left(\frac{b-a}{2}\right)^3\left(\frac{a+b}{2} + d_2\right) \\
\frac{a+b}{2} + d_1 &= -\left(\frac{a+b}{2} + d_2\right) \\
-(a+b) &= d_1 + d_2
\end{aligned}$$

To solve the second condition, p needs to be derivated. The trivial calculations are omitted in this proof.

$$\begin{aligned}
p_1^{(1)}(x) = (x-a)^2(4x + 3d_1 - a) \\
p_2^{(1)}(x) = (x-b)^2(4x + 3d_2 - b)
\end{aligned}$$

The second condition can now be checked:

$$p_1^{(1)}(m) = \left(\frac{b-a}{2}\right)^2 \left(4\frac{a+b}{2} + 3d_1 - a\right) = \left(\frac{a-b}{2}\right)^2 \left(4\frac{a+b}{2} + 3d_2 - b\right) = p_2^{(1)}(m)$$
$$2(a+b) + 3d_1 - a = 2(a+b) + 3d_2 - b$$
$$b - a = 3(d_2 - d_1)$$

Since there are two equations now, d_1 and d_2 can be solved to yield:

$$(d_1, d_2) = \left(\frac{-(a+2b)}{3}, \frac{-(2a+b)}{3}\right)$$

Another derivation is required:

$$p_1^{(2)}(x) = 4(x-a)(3x-2a-b)$$
$$p_2^{(2)}(x) = 4(x-b)(3x-a-2b)$$

Finally, the last condition can be verified:

$$p_1^{(2)}(m) = 4(m-a)(3m-2a-b) = 4(m-b)(3m-a-2b) = p_2^{(2)}(m)$$
$$3m - 2a - b = a + 2b - 3m$$
$$6m = 3(a+b)$$

Since all necessary conditions are satisfied, the original error can now be reexpressed:

$$\int_a^m p_1(x) f^{(4)}(x)dx + \int_m^b p_2(x) f^{(4)}(x)dx$$
$$= \left[f(x)p_1^{(3)}(x)\right]_a^m + \left[f(x)p_2^{(3)}(x)\right]_m^b + 24\int_a^m f(x)dx + 24\int_m^b f(x)dx$$
$$= \left[f(x)p_1^{(3)}(x)\right]_a^m + \left[f(x)p_2^{(3)}(x)\right]_m^b + 24\int_a^b f(x)dx$$

With $(p_1^{(3)}(x), p_2^{(3)}(x)) = (4(6x-5a-b), 4(6x-a-5b))$, the terms depending on f

can be resolved:

$$
\begin{aligned}
&\left[f(x)p_1^{(3)}(x)\right]_a^m + \left[f(x)p_2^{(3)}(x)\right]_m^b \\
=&f(m)\cdot 4(6m-5a-b) - f(a)\cdot 4(6a-5a-b) + f(b)\cdot 4(6b-a-5b) - f(m)\cdot 4(6m-a-5b) \\
=&-4(b-a)\left(f(a)+f(b)\right) + 4f(m)(4a-4b) \\
=&-4(b-a)\left(f(a)+f(b)+4f(m)\right) \\
=&-24\left(\frac{b-a}{6}\right)\left(f(a)+4f(m)+f(b)\right) = -24S(f)
\end{aligned}
$$

Substituting this into the error term yields:

$$
\int_a^m p_1(x)f^{(4)}(x)dx + \int_m^b p_2(x)f^{(4)}(x)dx = -24S(f) + 24\int_a^b f(x)dx = 24E(S)
$$

The absolute error is resolved in a similar manner as in the midpoint rule proof:

$$
\begin{aligned}
24|E_S| &= \left|\int_a^m p_1(x)f^{(4)}(x)dx + \int_m^b p_2(x)f^{(4)}(x)dx\right| \\
&= \left|\int_a^m p_1(x)f^{(4)}(x)dx\right| + \left|\int_m^b p_2(x)f^{(4)}(x)dx\right| \\
&\leq \int_a^m \left|p_1(x)f^{(4)}(x)\right|dx + \int_m^b \left|p_2(x)f^{(4)}(x)\right|dx \\
&\leq \int_a^m |p_1(x)|\left|f^{(4)}(x)\right|dx + \int_m^b |p_2(x)|\left|f^{(4)}(x)\right|dx \\
\text{(with } f^{(4)}(x)\leq k) &\leq k\left(\int_a^m |p_1(x)|\,dx + \int_m^b |p_2(x)|\,dx\right) \\
&= k\left(\int_a^m \left|(x-a)^3\left(x-\frac{a-2b}{3}\right)\right|dx + \int_m^b \left|(x-b)^3\left(x-\frac{2a-b}{3}\right)\right|dx\right)
\end{aligned}
$$

To eliminate the absolute values, the terms inside them need to be non-negative. This property is examined for the first term in the bounds $[a, m]$. Since $(x-a)^3 \geq 0$, it

suffices to analyse the right subterm.

$$\frac{a+b}{2} - \frac{a-2b}{3} = \frac{1}{6}(3a - 3b - 2a - 4b) = \frac{-(b-a)}{6} \leq 0$$

Because the right subterm is monotonically increasing, it follows that the whole left term is non-positive. The same analysis is done for the left term in $[m, b]$. This time, $(x-b)^3$ is non-positive in the interval. For the right subterm, it holds that:

$$b - \frac{2a-b}{3} = \frac{3b-2a-b}{3} = \frac{2(b-a)}{3} \geq 0$$

The two terms are now rewritten to be non-negative so that their absolute values are equal to their actual values:

$$\begin{aligned}
24|E_S| \leq & k\left(\int_a^m \left|(x-a)^3\left(\frac{a+2b}{3} - x\right)\right| dx + \int_m^b \left|(b-x)^3\left(x - \frac{2a+b}{3}\right)\right| dx\right) \\
= & k\left(\int_a^m (x-a)^3\left(\frac{a+2b}{3} - x\right) dx + \int_m^b (b-x)^3\left(x - \frac{2a+b}{3}\right) dx\right) \\
= & k\left(\int_a^m (x-a)^3\left(\frac{2b-2a}{3} - (x-a)\right) dx + \int_m^b (b-x)^3\left(\frac{2b-2a}{3} - (b-x)\right) dx\right) \\
= & k\left(\frac{2(b-a)}{3}\int_a^m (x-a)^3 dx - \int_a^m (x-a)^4 dx + \frac{2(b-a)}{3}\int_m^b (b-x)^3 dx - \int_m^b (b-x)^4 dx\right) \\
= & k\left(\frac{(b-a)}{6}(m-a)^4 - \frac{1}{5}(m-a)^5 + \frac{(b-a)}{6}(b-m)^4 - \frac{1}{5}(b-m)^5\right) \\
= & k\left(\frac{(b-a)}{6}\left(\frac{b-a}{2}\right)^4 - \frac{1}{5}\left(\frac{b-a}{2}\right)^5 + \frac{(b-a)}{6}\left(\frac{b-a}{2}\right)^4 - \frac{1}{5}\left(\frac{b-a}{2}\right)^5\right) \\
= & k\left(\frac{(b-a)^5}{48} - \frac{2}{5}\left(\frac{b-a}{2}\right)^5\right) = \frac{k(b-a)^5}{120}
\end{aligned}$$

To obtain the final error form, only a few arithmetic operations remain.

$$|E_S(a,b)| \leq \frac{k(b-a)^5}{120 \cdot 24} = \frac{k(b-a)^5}{15 \cdot 2^3 \cdot 6 \cdot 2^2} = \frac{k(b-a)^5}{90 \cdot 2^5}$$

□

Remark 2.22

Because of the stricter requirements on continuous differentiability, there are cases where the midpoint rule error can be bounded, but the Cavalieri-Simpson rule error cannot. Consider the function

$$f(x) = \begin{cases} x^3 \sin\left(\frac{1}{x}\right) & \text{if } x \neq 0, \\ 0 & \text{if } x = 0. \end{cases}$$

It can be shown that $\lim_{x^+ \to 0} f^{(2)}(x) = \lim_{x^- \to 0} f^{(2)}(x)$, but not $\lim_{x^+ \to 0} f^{(4)}(x) = \lim_{x^- \to 0} f^{(4)}(x)$. Consequently, the midpoint rule error is bounded if the function is approximated over any interval containing 0, but the Cavalieri-Simpson rule error is not.

2.2.2 Composite Rules

On large intervals, the presented rules leave a lot to be desired in terms of accuracy. Therefore, a mitigation technique needs to be applied to achieve a reasonable precision. One of the simplest approaches is to split the interval into N equidistant subintervals, apply a simple quadrature rule piecewise and sum the result.

Definition 2.23 (Composite Rules)

Let $I_n(f, a, b)$ be an integral approximation and let $N \in \mathbb{N}^+$. With $H = (b-a)/N$, the corresponding composite rule is defined as

$$I_{n,N}(f, a, b) = \sum_{i=0}^{N-1} I_n\left(f, a + iH, a + (i+1)H\right)$$

The composite midpoint rule is thus equal to

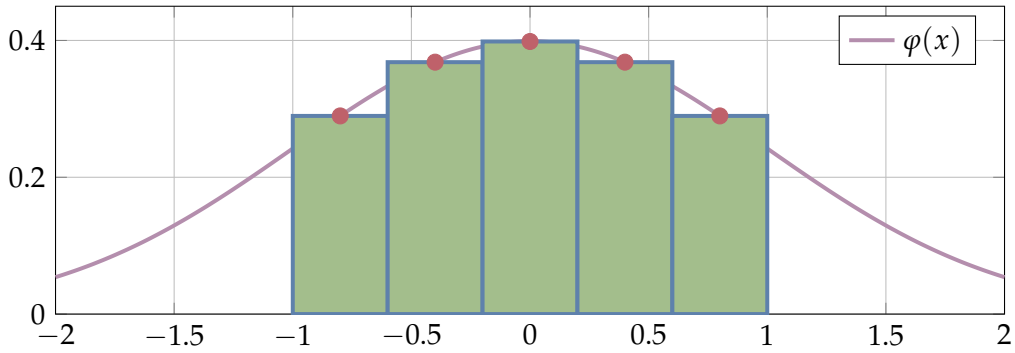
$$M_N(f, a, b) = H \sum_{i=0}^{N-1} f\left(a + (2i+1)\frac{H}{2}\right)$$

and the composite Cavalieri-Simpson rule is equal to

$$S_N(f, a, b) = \frac{H}{6}\left(f(a) + f(b) + 2\sum_{i=1}^{N-1} f(a + iH) + 4\sum_{i=0}^{N-1} f\left(a + (2i+1)\frac{H}{2}\right)\right)$$

Example 2.24

The composite midpoint rule of $\int_{-1}^{1} \varphi(x)dx$ with $N = 5$ is visualized below.



Remark 2.25

Interestingly, each two adjacent intervals of the composite Cavalieri-Simpson rule share an evaluation point. As a consequence, the computational effort does not scale by a factor of $3N$ as expected, but by a factor of $2N + 1$. The midpoint rule, however, does not benefit from composition in terms of resource efficiency.

It can now be shown that, as expected, the error of a composite Newton-Cotes rule shrinks dependent on both the number of intervals and the order of the original approximation.

Theorem 2.26 (Composite Rule Error Bound)

Let $I_n(f, a, b)$ be an integral approximation with an error bound of the form:

$$|E_{I_n}(a, b)| \leq \frac{k(b-a)^m}{l}$$

Then the error of its respective composite rule is bounded by

$$|E_{I_{n,N}}(a, b)| \leq \frac{k(b-a)^m}{l \cdot N^{m-1}}$$

Proof. The absolute error of the entire interval is at most equal to the sum of the

subinterval errors.

$$\begin{aligned}
|E_{I_{n,N}}(a,b)| &\leq \sum_{i=0}^{N-1} E_{I_n}\left(\left(a+i*\frac{b-a}{N}\right),\left(a+(i+1)*\frac{b-a}{N}\right)\right) \\
&\leq \frac{k}{l}\sum_{i=0}^{N-1} x\left(\left(a+(i+1)*\frac{b-a}{N}\right)-\left(a+i*\frac{b-a}{N}\right)\right)^m \\
&= \frac{k}{l}\sum_{i=0}^{N-1}\left(\frac{b-a}{N}\right)^m = \frac{Nk}{l}\left(\frac{b-a}{N}\right)^m = \frac{Nk}{l}\frac{(b-a)^m}{N^m} \\
&= \frac{k(b-a)^m}{l\cdot N^{m-1}}
\end{aligned}$$

□

Remark 2.27

Newton-Cotes rules with higher order gain a higher precision advantage from composition. Consequently, the composite Cavalieri-Simpson rule has a distinct precision advantage over the composite midpoint rule.

The composite error bounds of the presented rules can now be formulated.

Corollary 2.28 (Composite Midpoint Rule Error Bound, [20, p. 50])
Assuming that $f \in C^2_{[a;b]}$, the absolute quadrature error of the composite midpoint rule is at most equal to:

$$|E_{M_N}(a,b)| \leq \frac{k(b-a)^3}{24\cdot N^2}$$

Corollary 2.29 (Composite Cavalieri-Simpson Rule Error Bound, [20, p. 53])
Assuming that $f \in C^4_{[a;b]}$, the absolute quadrature error of the composite Cavalieri-Simpson rule is at most equal to:

$$|E_{S_N}(a,b)| \leq \frac{k(b-a)^5}{90\cdot 2^5\cdot N^4}$$

Example 2.30 (Approximation of the Normal Distribution)
Consider the probability density function of the normal distribution with standard

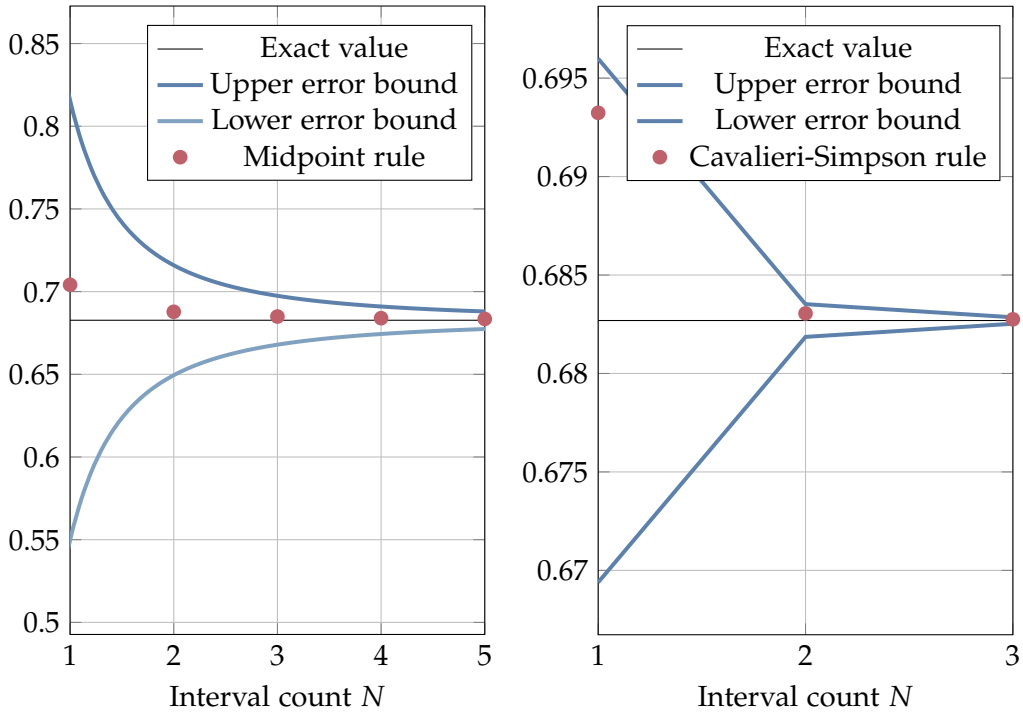
deviation 1 and expectation 0:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

The probability that a normal distributed random variable lies in the bounds $[-1, 1]$ is exactly

$$\int_{-1}^{1} \varphi(x)dx$$

The approximation of this integral using both composite rules with different N is now plotted together with its error.



It is immediately obvious that the Cavalieri-Simpson rule offers an enormous precision advantage for this specific integral. However, the precision difference always depends on the specific function.

3 Formalization of the Proof in Isabelle

In this chapter, the relation between the formalized theorems and the previously discussed proofs is explained by examining the difference between the definitions of the rules and their error bounds. Furthermore, motivations for formalization choices are presented and a quick summary of inconveniences during the proving process is given.

Familiarity with Isabelle is required in the remainder of this thesis. A short introduction to the practical proving process with Isabelle was presented by Nipkow [16].

3.1 Defining the Procedures

Before the proofs themselves can be translated, the desired goals and definitions need to be formally stated. The approximation rules are expressed according to the definitions in chapter 2.

```
definition midpoint_rule :: "(real ⇒ real) ⇒ real ⇒ real ⇒ real" where
  "midpoint_rule f a b = (b - a) * f ((a + b)/2)"

definition simpson_rule :: "(real ⇒ real) ⇒ real ⇒ real ⇒ real" where
  "simpson_rule f a b = (b - a) / 6 * (f a + 4 * f ((b + a)/2) + f b)"

definition midpoint_rule_comp ::
  "(real ⇒ real) ⇒ real ⇒ real ⇒ nat ⇒ real"
  where "midpoint_rule_comp f a b n = (let H = (b - a)/n
    in H * (∑ k ← [0..<n]. f (a + (2 * k + 1) * h / 2)))"

definition simpson_rule_comp ::
  "(real ⇒ real) ⇒ real ⇒ real ⇒ nat ⇒ real"
  where "simpson_rule_comp f a b n = (let H = (b - a)/n
    in H/6 * (f a + f b
              + 2 * (∑ k ← [1..<n]. f (a + k * H))
              + 4 * (∑ k ← [0..<n]. f (a + H/2 + k * H))))"
```

Expressing the necessary assumptions for the error bounds introduces the first technical limitation of Isabelle. The original definition states that the integrand f needs to be continuously differentiable two or four times. In the source code however, a slight deviation is found. Consider this incomplete definition of the midpoint error bound theorem:

```
theorem midpoint_approx_error:
assumes deriv1: "⋀x. (f  has_real_derivative f'  x) (at x within {a..b})"
 and    deriv2: "⋀x. (f' has_real_derivative f'' x) (at x within {a..b})"
 and  cont_f'': "continuous_on {a..b} f''"
 and f''_bound: "⋀x. x ∈ {a..b} ⟹ k ≥ |f'' x|"
 ...
shows "| ??? - midpoint_rule f a b| ≤ k * ((b-a)^3)/24"
```

Because the concept of differentiability classes does not exist in Isabelle, the assumption is instead split into consequent single derivatives. Unfortunately, this infers that the derivatives need to be specified explicitly when the theorem is used. As long as there is no implicit way to refer to the nth derivative of a function, this problem cannot be avoided.

As the reader might have noticed, the above definition is still incomplete. It is obvious that the placeholder term "???" must refer to the actual integral of f inside its bounds, which can be expressed in a variety of different ways, with each of them having their advantages and drawbacks.

3.2 Choosing the Appropriate Integral Definition

Isabelle provides two major approaches to the integration of real functions. On one hand, there is the popular Lebesgue integral [9], which is built upon measure theory. Superficially, it relies on mapping each element in the value space of a function to its subset in the domain, then obtaining its "length" using a supplied measure function and multiplying it with the corresponding value. In contrast to the Riemann integral, it can only integrate over the whole domain of a function. To narrow down the integration region, the original function can be multiplied with the indicator function of the new domain, effectively setting the new function value outside of the desired set to zero. The Isabelle library represents this in the following way (although with less syntactic sugar):

```
definition "∫x ∈ A. f x ∂M = ∫x.(indicator A x * f x) ∂M"
```

Here, `M` is an arbitrary measure space and `A` is the integration domain. For the purpose of this formalization, `M` will always be the Lebesgue-Borel measure `lborel`, which represents a formal definition of the intuitive concept of integration. Since intervals are common integration domains, Isabelle provides an additional specialization for interval integrals as well.

```
definition interval_lebesgue_integral where
  "∫x=a..b|f x. M = (if a ≤ b then ∫x ∈ einterval a b. f x ∂M
                              else ∫x ∈ einterval b a. f x ∂M)"
```

In order to allow the expression of improper integrals, this definition uses the datatype `ereal`, which extends the `real` datatype with additional values for positive and negative infinity.

On the other hand, the lesser-known Henstock-Kurzweil (HK) integral, a generalization of the Riemann integral, provides a different approach to integration over sets. Its mathematical idea is covered in [7]. In the Isabelle library, it is expressed through the `integral` function.

```
definition integral :: "'a set ⇒ ('a ⇒ 'b) ⇒ 'b"
```

The HK-integrable functions are a superset of the Lebesgue-integrable functions. However, this fact does not influence the suitability for this application because all continuous functions are Lebesgue-integrable and the proof only contains integrals on continuous functions. Subsequently, the integral definitions are convertible into each other easily at any point in the proof.

At first glance, the interval Lebesgue integral appears to be the most appealing solution since the approximations are only defined on intervals. However, the support of indefinite integrals (which is unnecessary in this case) turns out to be a major annoyance in the proving process. Not only does it induce notational clutter with extra type conversions between `real` and `ereal`, but it also creates superfluous case distinctions on both bounds to ensure that neither of them contains an infinite value. This leaves the choice between the set Lebesgue integral and the HK integral. Since none of them offer significant usability advantages over one another, the Lebesgue integral was ultimately chosen because it was more familiar to the author. The definition can now be finished.

```
theorem midpoint_approx_error:
assumes deriv1: "⋀x. (f  has_real_derivative f'  x) (at x within {a..b})"
 and    deriv2: "⋀x. (f' has_real_derivative f'' x) (at x within {a..b})"
 and  cont_f'': "continuous_on {a..b} f''"
 and f''_bound: "⋀x. x ∈ {a..b} ⟹ k ≥ |f'' x|"
```

```
 and    a_le_b: "a ≤ b"
shows "|∫x ∈ {a..b}. f x ∂lborel - midpoint_rule f a b|
       ≤ k * ((b-a)^3)/24"
```

An additional assumption `a_le_b` is required by the interval notation of sets because the set `{b..a}` is empty (or contains one element if `a=b`).

The Cavalieri-Simpson error bound is stated in a similar way.

```
theorem simpson_approx_error:
assumes deriv1: "⋀x. (f has_real_derivative f1 x) (at x within {a..b})"
 and    deriv2: "⋀x. (f1 has_real_derivative f2 x) (at x within {a..b})"
 and    deriv3: "⋀x. (f2 has_real_derivative f3 x) (at x within {a..b})"
 and    deriv4: "⋀x. (f3 has_real_derivative f4 x) (at x within {a..b})"
 and    cont_f4: "continuous_on {a..b} f4"
 and   f4_bound: "⋀x. x ∈ {a..b} ⟹ k ≥ |f4 x|"
 and    a_le_b: "a ≤ b"
shows "|∫x ∈ {a..b}. f x ∂lborel - simpson_rule f a b|
       ≤ k * ((b-a)^5)/(90*2^5)"
```

The composite rules merely extend the assumptions of the single-interval rules by the obvious requirement that `N` needs to be positive.

```
theorem midpoint_comp_approx_error:
assumes ... and "N > 0"
shows "|∫x ∈ {a..b}. f x ∂lborel - midpoint_rule_comp f a b N|
       ≤ k * ((b-a)^3)/(24*N^2)"

theorem simpson_comp_approx_error:
assumes ... and "N > 0"
shows "|∫x ∈ {a..b}. f x ∂lborel - simpson_rule_comp f a b N|
       ≤ k * ((b-a)^5)/(90*2^5*N^4)"
```

3.3 Differences between the Paper Proof and the Formal Proof

In general, the formal proofs of the simple rules follow the principle described in chapter 2. The original integral is first split at the midpoint. After both sides have been repeatedly integrated by parts and all terms have been cancelled out, the absolute upper bounds of both integrals are estimated using the assumed derivative bound. All remaining polynomial integrals are consequently resolved to yield the final approximation. Fortunately, it was possible to skip the resolution of the linear equation systems because

the integration constants were already known from the paper proof. Subsequently, the required polynomials could just be filled in. To reduce code duplication, the repeated integration by parts as well as the polynomial derivatives have been extracted into their own lemma.

In contrast, the formal proofs of the composite rules do not follow the paper at all and an induction over the subinterval count was chosen instead. Working with finite sums in Isabelle can be quite challenging, especially when the individual terms contain a large amount of symbols. Additionally, while theoretically possible, splitting the integral into an arbitrary amount of subintervals proved to be quite cumbersome. Using natural induction turned out to be significantly easier, mainly because the base case was already proven and the induction step only involved instantiating the hypothesis and the basic theorem followed by simple arithmetic operations, but also because the integral is only required to be split into just two parts.

3.4 Notable Difficulties during the Formalization

It should be no surprise that the translation process was not as smooth as it may have been suggested up until now. Since the author's prior Isabelle experience was limited, a lot of trial and error was involved in finding the right formulations, lemmas and tactics. The `find_theorems` command proved to be extremely helpful, although the wildly inconsistent naming schemes between different integral definitions definitely did not. For instance, the fundamental theorem of calculus is defined as `fundamental_theorem_of_calculus` for the HK integral, whereas the set Lebesgue integral offers both `integral_FTC_Icc` and `integral_FTC_Icc_real`, which each require different formulations of continuity. This led to frequent conversions between HK and Lebesgue integral in order to use a required lemma that could not be found quickly for the current definition. Once the proof had been finished though, all conversions to the HK integral were able to be eliminated.

Furthermore, automatically obtaining derivatives of any function, including simple polynomials, was impossible, even when utilizing the purposefully designed `derivative_intros`. In the beginning, this problem was resolved by creating a step-by-step proof to slowly build up the derivative. After a while, the author discovered that the derivative rules could be chained using forward reasoning. As a result, derivative goals were solved in a single goal using comically large instantiations of a single rule, the largest of which had a stunning length of 209 characters.

Having access to the `ISAR` language, especially to the chain reasoning with `also` and `finally`, was a true blessing. However, Isabelle ocasionally failed to continue the chain and got stuck in an endless loop while using these keywords. The equation chain was

then split up into multiple smaller chains that were unfolded at the final goal again.

As the approximation parameters contain both `real` and `nat` values, type conversions between them were unavoidable. Surprisingly, Isabelle did not handle these operations well, especially when natural powers of real numbers were involved. Nevertheless, the `argo` and `algebra` tactics were able to prove a large number of arithmetic goals, albeit in many tiny steps. However, the most significant problems in this regard occured when using sums of lists that map natural numbers onto real numbers. Solving equalities of the form

```
( ∑ i ← [0..<n::nat]. f (real i)) = ( ∑ i ← map real [0..<n::nat]. f i)
```

was completely impossible on multiple occasions, even if both sides of the equation were otherwise entirely equal.

4 The Executable Implementation

This chapter presents two functions that compute an approximation of an integrand given in the `floatarith` datatype using the composite midpoint rule and the composite Cavalieri-Simpson rule and provides a rundown of the theoretical and practical details.

4.1 The Approximation Library

The numerical algorithms are expressed using arithmetic expressions of the `floatarith` type, which allows variables, constants and a considerable number of arithmetic operations to be expressed. A full list of available definitions can be found in its definition [10, p. 3]:

```
datatype floatarith = Add floatarith floatarith | Minus floatarith
  | Mult floatarith floatarith | Inverse floatarith | Cos floatarith
  | Arctan floatarith | Abs floatarith | Max floatarith floatarith
  | Min floatarith floatarith | Sqrt floatarith | Exp floatarith
  | Powr floatarith floatarith | Ln floatarith | Power floatarith nat
  | Floor floatarith | Var nat | Num float | Pi
```

These expressions can then be evaluated using the `approx` function, which additionally accepts a list of bound variables and a natural number that controls the precision of the floating point calculations. The technical details can be found in the original paper [11].

```
fun approx :: "nat ⇒ floatarith ⇒ (float interval) option list
                  ⇒ (float interval) option" where
  "approx prec ... vs = ..." (*omitted for brevity*)
```

As the reader might have noticed, the `floatarith` type lacks subtraction and division operators. According to the original paper, these should be expressed by their inverse operations. As an example, the division of `x` by `y` is written as `Mult x (Inverse y)` [11, ch. 4.5.1].

4.2 Implementing the Quadrature Methods

With these definitions, the quadrature methods can now be expressed. The arguments of the approximation function should contain the precision value for the `approx` function, the interval count, the integrand including its integration variable and the integral bounds. Some type abbreviations are introduced to reduce notational clutter and increase readability. Furthermore, the abbreviation `sum_expr` is introduced as $\sum_{i=low}^{high}$ `Var i`.

```
type_synonym prec = nat
type_synonym ivl_count = nat
type_synonym integr_var = nat
type_synonym val = "(float interval) option"
type_synonym int_approx = "prec ⇒ ivl_count
       ⇒ floatarith ⇒ integr_var ⇒ val ⇒ val ⇒ val list ⇒ val"

abbreviation "sum_expr low high ≡ fold (Add ∘ Var) [low..high] 0"

fun int_approx_midpoint :: int_approx where
  "int_approx_midpoint prec N f x a b vals = ???"

fun int_approx_simpson :: int_approx where
  "int_approx_midpoint prec N f x a b vals = ???"
```

Notice that `sum_expr` creates a right-associative sum instead of a left-associative sum. The significance of the summation order will be discussed in Section 4.3.2.

Recall the definition of the composite midpoint rule:

$$M_N(f,a,b) = H \sum_{i=0}^{N-1} f\left(a + (2i+1)\frac{H}{2}\right) \text{ where } H = \frac{b-a}{N}$$

Computing this expression with a single call to `approx` proves to be difficult because `f` itself is an arbitrary expression which may contain variables and should only be evaluated with the right parameters. Thus, the computation process is split into multiple steps. Each argument to `f` in the sum is computed separately, then the function itself is evaluated with the result. Finally, the function values are summed up and multiplied by `H`. To reduce the number of unnecessary computations, the value of `H` is computed in advance as it occurs in each call in the first step as well as in the last call.

Obviously, all arithmetic expressions need to be stated using the `floatarith` datatype. However, as the concrete syntax is quite unreadable, expressions of this type will be

displayed in a readable format instead. To reduce notational clutter, the `Num` constructor and all type conversions are omitted. Addition, multiplication and negation are displayed by their standard operators instead of their respective constructors.

```
int_approx_midpoint prec N f x a b vals = (let apprx = approx prec;
    H = apprx (((Var 2) + (-Var 1)) * (Var 0)^(-1)) [N, a, b];
    x_expr = (Var 1) + (2 * (Var 0) + 1) * (Var 2) * 2^(-1);
    xs = map (λi. apprx x_expr [i, a, H]) [0..<N];
    fs = map (λv. apprx f (vals[x := v])) xs;
  in apprx (Var 0 * sum_expr) (H # fs))
```

The Cavalieri-Simpson rule is implemented in a similar way. Recall its definition:

$$S_N(f,a,b) = \frac{H}{6}\left(f(a) + f(b) + 2\sum_{i=1}^{N-1} f(a+iH) + 4\sum_{i=0}^{N-1} f\left(a + (2i+1)\frac{H}{2}\right)\right)$$

Similar to before, the value `H/2` is precomputed because it occurs in most terms. Consequently, the function arguments on the subinterval bounds and on the subinterval midpoints are obtained and their function values are computed. The last expression finally evaluates the approximated integral.

```
int_approx_simpson prec N f x a b vals = (let apprx = approx prec;
    H2 = apprx ((Var 2 + (-Var 1)) * (Var 0 * 2)^(-1)) [N, a, b];
    x_evn_expr = Var 2 + (2 * Var 0) * Var 1;
    x_odd_expr = x_evn_expr + Var 1;
    xs_evn = map (λi. apprx x_evn_expr [i, H2, a]) [0..N];
    xs_odd = map (λi. apprx x_odd_expr [i, H2, a]) [0..<N];
    fs = map (λv. apprx f (vals[x := v])) (xs_evn @ xs_odd);
    final_expr = Var 0 * 3^(-1) * (Var 1 + Var (N+1) + 2 * sum_expr 2 N
                                        + 4 * sum_expr (N+2) (2*N+1))
  in apprx final_expr (H2 # fs))
```

Both functions are executable using the ML code generator. For example, to approximate the integral $\int_0^1 e^x dx$ with precision 5 on 4 subintervals, the following code can be used:

```
abbreviation "ivl_coerce ≡ Some ∘ interval_of"
value "int_approx_midpoint 5 4 (Exp (Var 0)) 0
          (ivl_coerce 0) (ivl_coerce 1) [(ivl_coerce 0)]"
  (* Result: [1.625; 1.78125] *)
```

```
value "int_approx_simpson 5 4 (Exp (Var 0)) 0
          (ivl_coerce 0) (ivl_coerce 1) [(ivl_coerce 0)]"
  (* Result: [1.59375; 1.8125] *)
```

Notice that the Cavalieri-Simpson rule computes a wider interval than the midpoint rule. This can be explained using the following definition.

Definition 4.1 (Analytical and Numerical Error)
The overall error of an implementation of an approximation consists of the error occuring from the approximation formula itself (analytical error) and the rounding errors occuring during computation (numerical error).

It should be noted that the interval width is only determined by the numerical error. The analytical error of both rules must be computed separately.

4.3 Numerical Considerations

Although a formal proof of the equivalence between the mathematical expression and its `floatarith` representation was not conducted due to a skill issue on the author's side, it appears very plausible that both implemented functions compute an interval in which the exact result of each respective integration rule lies. However, the numerical properties of an algorithm need to be analyzed as well in order to determine its usefulness. In this subsection, some numerical core concepts are briefly introduced and subsequently applied to the implementations.

4.3.1 Condition

In the field of numerics, the condition of a problem refers to its sensitivity to small perturbations in the input data. It is unrelated to the specific algorithm used in practice. Detailed information on this concept can be found in [19, p. 36f]. In this thesis, the problem is finding the value of the expression

$$I(f,a,b) = \int_a^b f(x)dx \text{ where } f \in C^1_{[a;b]} \subseteq C^n_{[a;b]}\,.$$

The interval bounds a and b are the only numerical parameters in this expression. To determine the sensitivity of a problem, its condition number is often used.

Definition 4.2 (Condition Number, [19, p. 36f])
Let d be the exact input, let $d + \delta_d$ be the perturbated input of an algorithm and let D be a region around d. Further, let $(x, x + \delta_x)$ be its respective outputs. The relative condition number of the algorithm is defined as

$$K(d) = \sup_{\delta_d \in D} \frac{\|x + \delta_x\| / \|x\|}{\|d + \delta_d\| / \|d\|}$$

A problem is *well-conditioned* if its relative condition number is low and *ill-conditioned* if it is high. The judgement on the terms "low" and "high" depends on the problem.

Remark 4.3
Technically, the definition of $K(d)$ only considers a single input. However, the integration problem has two inputs a and b. For simplicity reasons, its condition number is defined as

$$K_I(a, b) = \sup_{\delta_a, \delta_b \in D} \frac{|I(f, a + \delta_a, b + \delta_b)| / |I(f, a, b)|}{|h + \delta_b + \delta_a| / |h|} \text{ where } h = (b - a)$$

in this thesis. This is viable as the d in the definition models the ”size“ of the input, which refers to the integration interval in this case.

To compute $K_I(a, b)$, the perturbed integral needs to be resolved.

$$\begin{aligned} I(f, a + \delta_a, b + \delta_b) &= \left| \int_{a+\delta_a}^{b+\delta_b} f(x)dx \right| = \left| \int_{a+\delta_a}^{a} f(x)dx + \int_{a}^{b} f(x)dx + \int_{b}^{b+\delta_b} f(x)dx \right| \\ &\leq \left| \int_{a}^{b} f(x)dx \right| + \left| \int_{a}^{a+\delta_a} f(x)dx \right| + \left| \int_{b}^{b+\delta_b} f(x)dx \right| \\ &= |I(f, a, b)| + |F(a + \delta_a) - F(a)| + |F(b + \delta_b) - F(b)| \end{aligned}$$

Since F is obviously differentiable, it is also Lipschitz-continuous [8, §11.3a].

Definition 4.4 (Lipschitz-continuous Function, [8, §11.3a])
A function $f : D \to \mathbb{R}, D \subseteq \mathbb{R}$ is Lipschitz-continuous with Lipschitz constant $L \in \mathbb{R}_+$ if $\forall x, x' \in D$, it holds that

$$|f(x) - f(x')| \leq L|x - x'|$$

Assuming that L_F is the Lipschitz constant of F restricted to $D_a \cup D_b$, the expression can be rearranged.

$$\begin{aligned}
&|I(f,a,b)| + |F(a+\delta_a) - F(a)| + |F(b+\delta_b) - F(b)| \\
\leq &|I(f,a,b)| + L_F|a+\delta_a - a| + L_F|b+\delta_b - b| \\
= &|I(f,a,b)| + L_F(\underbrace{|\delta_a| + |\delta_b|}_{=\delta})
\end{aligned}$$

It follows that

$$\begin{aligned}
K_I(a,b) &= \sup_{\delta_a,\delta_b \in (D_a \times D_b)} \frac{|I(f,a+\delta_a,b+\delta_b)|/|I(f,a,b)|}{|h+\delta_b-\delta_a|/|h|} \\
&\leq \sup_{\delta_a,\delta_b \in (D_a \times D_b)} \left(1 + \frac{\delta L_F}{|I(f,a,b)|}\right) \Big/ \left|1 + \frac{\delta_b - \delta_a}{h}\right|
\end{aligned}$$

A few properties of K_I can be observed. First of all, the derivation is only legal if $f \in C_{[a+\delta_a;b+\delta_b]}$. Although this is usually the case, it must be remembered that a violation of this property may cause the integral to be very ill-conditioned.

Considering the denominator, it is clear that the condition worsens when the error shrinks the interval. From the numerator, it can be deduced that the condition also worsens if the error is large (which is obvious) and if F has a high Lipschitz constant around the boundary points. This means that functions that are very large at the integration bounds compared to the values inside the interval may result in an ill-conditioned integral. However, in most practical use cases, the condition number is very low, meaning that the problem is generally well-conditioned.

4.3.2 Stability and Rounding Errors

In contrast to the condition, the concept of stability refers to the sensitivity of a specific algorithm to small perturbations in the input data. If the rounding errors of the float arithmetic accumulate disproportionally in relation to the precision of the input data and to the expected precision, the algorithm is considered instable [21, ch. 1.5.2].

A full stability analysis vastly exceeds the scope of this thesis. However, both implementations suffer from one core problem. This section explores the root of this issue and suggests a mitigation technique.

Consider the approximation of $I(f,1,10)$ and $I(-f,10,1)$ where $f(x) = 100(x^{10})^{-1}$ using the implemented algorithm of the composite midpoint rule with 20 subintervals and precision 5. It can be proven (as an exercise) that the composite Cavalieri-Simpson

rule yields the same exact result on paper for both functions, raising the expectation that the computed rules will yield similar results.

```
abbreviation "f ≡ (Mult (Num 100) (Inverse (Power (Var 0) 10)))"
abbreviation "apprx fn a b = int_approx_midpoint 5 20 fn 0
          (ivl_coerce a) (ivl_coerce b) [ivl_coerce 0]"
value "apprx f 1 10"
  (* Result: [4.375, 11.0] *)
value "apprx (Minus f) 10 1"
  (* Result: [0.75, 52] *)
```

Surprisingly, the size of both intervals is vastly different. To explore this issue, the intermediate steps of the computation are unfolded. Considering the distinct function values before summation may explain the results.

```
apprx f 1 10 = H * (((... + [0.03, 0.08]) + [0.40, 0.70]) + [9.75, 14.5])
apprx f 10 1 = H * ((([9.75, 14.5] + [0.40, 0.70]) + [0.03, 0.08]) + ...)
```

It seems that the summation order may play a role in the explanation. To obtain a deeper understanding of the problem, a few definitions are needed.

Definition 4.5 (Floating Point Numbers, [18, ch. 7.3.2])

A floating point number $x \in \mathbb{F}$ with mantissa m and exponent e, represented by the `float` datatype in Isabelle, is defined as

$$x = m \cdot 2^e \text{ where } m, e \in \mathbb{Z}$$

The floating point numbers $\mathbb{F}$ are closed under all elementary arithmetic operations [12].

Definition 4.6 (Rounding)

The functions

$$\lfloor x \rfloor_A = \left(\max_{a \in A} a \leq x \right) \text{ and } \lceil x \rceil_A = \left(\min_{a \in A} a \geq x \right)$$

round a number x up or down to the next value in the set A.

Definition 4.7 (p-precise Floating Point Numbers)
With $p \in \mathbb{N}$, the set of p-precise floating numbers is defined as

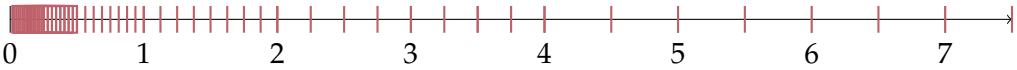
$$\mathbb{F}_p = \left\{ \lfloor x \cdot 2^p \rfloor_{\mathbb{Z}} \cdot 2^{-p} \middle| x \in \mathbb{R} \right\} \subset \mathbb{F}$$

Their rounding functions are concretely defined as

$$\lfloor x \rfloor_{\mathbb{F}_p} = \lfloor x \cdot 2^p \rfloor_{\mathbb{Z}} \cdot 2^{-p} \text{ and } \lceil x \rceil_{\mathbb{F}_p} = \lceil x \cdot 2^p \rceil_{\mathbb{Z}} \cdot 2^{-p}$$

An interval $[x; y] \in \mathcal{F}_p^2$ is p-precise iff its bounds are p-precise.

Remark 4.8 (Intuition of $\mathbb{F}_p$. Image Source: [13])
Intuitively, the precision controls the "fineness" of a floating point set, i.e. how many values lie between 2^z and 2^{z+1} for any $z \in \mathbb{Z}$. The cardinality of the set $[2^z; 2^{z+1}) \subset \mathbb{F}_p$ is 2^p. As an example, the set $\mathbb{F}_3$ is visualized in the interval $[0; 2^3)$.



The set is very dense around 0 and becomes more and more sparse towards infinity. There are $2^3 = 8$ steps in each interval $[2^z; 2^{z+1})$.

Theorem 4.9 (Floating Point Approximation Interval, [10, ch. 3.3])
A floating-point value $x = m \cdot 2^e \in \mathbb{F}$ can be approximated with precision p by any interval

$$[\underline{x}; \overline{x}] \in \mathbb{F}_p^2$$

where $\overline{x} - x \leq 2^{e-p}$ and $x - \underline{x} \leq 2^{e-p}$. The unique interval generated by the p-precise rounding functions on x fulfills these constraints.

Definition 4.10 (Floating Point Interval Addition, [10])
The addition of two floating-point intervals $x = [\underline{x}; \overline{x}], y = [\underline{y}; \overline{y}] \in \mathbb{F}_p^2$ with precision p is defined as

$$x + y = \left[\left\lfloor \underline{x} + \underline{y} \right\rfloor_{\mathbb{F}_p}; \left\lceil \overline{x} + \overline{y} \right\rceil_{\mathbb{F}_p} \right]$$

Conjecture 4.11 (Absorption)

Let $x, y \in \mathbb{F}_p^+$ be two p-precise positive floating numbers with $x > y$. If e is the exponent of x and y is relatively small, i.e. if y approaches or is smaller than 2^{e-p}, the relative rounding errors

$$\frac{\lceil x+y \rceil - x}{y} \text{ and } \frac{\lfloor x+y \rfloor - x}{y}$$

should increase. If $y \leq 2^{e-p}$, it should also hold that

$$\lceil x+y \rceil - x = 2^{e-p} \text{ and } \lfloor x+y \rfloor - x = 0$$

from the definitions, i.e. the rounding result changes by at least one "step" when rounding up or stays unchanged when rounding down.

Finally, recall the approximations from the beginning of the subsection:

```
apprx f 1 10 = H * (((... + [0.03, 0.08]) + [0.40, 0.70]) + [9.75, 14.5])
  (* result: [4.375; 11.0] *)
apprx f 10 1 = H * ((([9.75, 14.5] + [0.40, 0.70]) + [0.03, 0.08]) + ...)
  (* result: [0.75; 52] *)
```

The Absorption Conjecture would explain the observations about the intervals. In the first function, the summands are approximately equal and a reasonable precision is reached. However, in the second function, the summands differ magnificently and the the relative rounding error explodes for each individual sum, resulting in a horrendously imprecise result. In this example, the error can be mitigated easily, but for arbitrary functions, the problem is unavoidable using the current libraries.

To mitigate the numerical errors in large sums, the `floatarith` type could be extended to include a `Sum [floatarith]` constructor. The `approx` function could then approximate all expressions at once and perform a suitable reordering procedure, simultaneousy opening the door for optimizations through parallelization.

4.4 Evaluation of the Extracted Precision Data

While theoretical reasoning may help to provide important general statements about the implementations, some important conclusions can often only be drawn from real data. This is especially the case for the relation between the approximation parameters `prec`, `N` and the resulting approximation quality. As the supplied function can be arbitrary

within the constraints, the amount of theoretical reasoning that can be conducted is rather limited. While a deeper analysis with a specifically chosen function can technically be conducted, it is often faster to extract some data and draw conclusions based on observations.

For this section, precision data from three specific integration problems has been extracted from Isabelle which will now be analyzed.

4.4.1 Data Analysis for $I(f,0,1)$ where $f(x) = \cos(x^{-1} + 2^{-10})$

The function is internally represented by the `floatarith` expression

```
abbreviation "f ≡ Cos (((Var 0) + 2^(-10))^(-1))"
```

and visualized by the following graph.

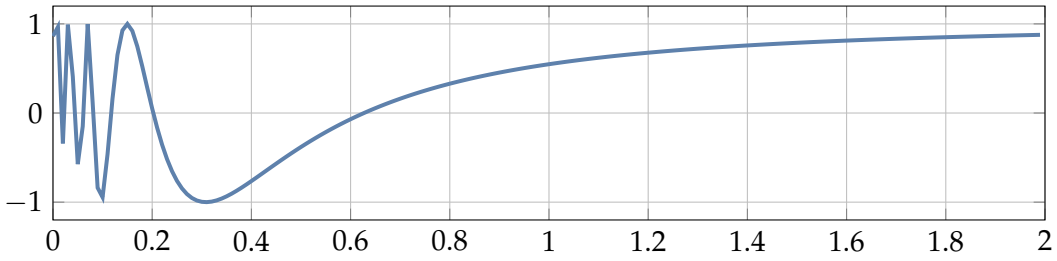


Figure 4.1: Inaccurate plot of $f(x) = \cos((x)^{-1} + 2^{-10})$

Strong oscillation is observed in $[-1;1]$ for smaller values, making the function a prime example for a very ill-conditioned problem. In fact, the numerical properties of the plotting tool prevent it from being accurately displayed. Consequently, the approximation results printed in the following graph should not come as a surprise.

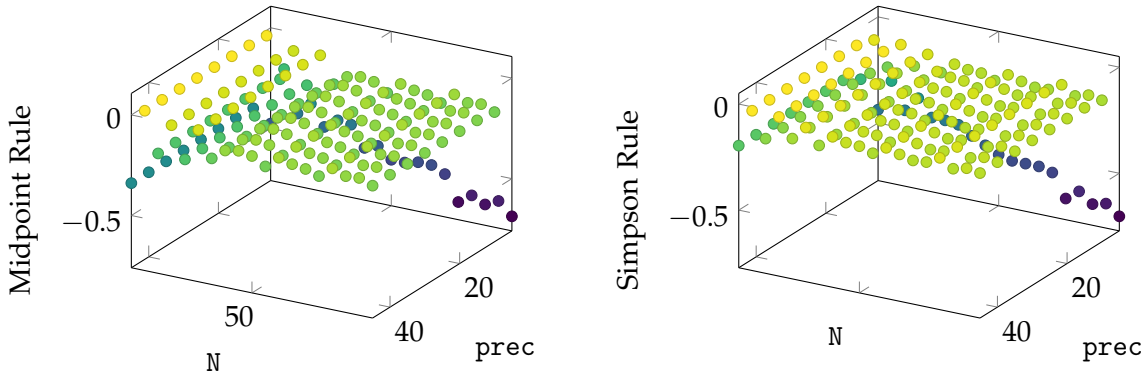


Figure 4.2: Linear plot of the lower approximation bound of $I(f,0,1)$ with both functions. Both parameters range from 5 to 50 in steps of 5.

The results vary greatly in size and the trend only becomes visible at a very high precision and interval count, confirming the theoretical reasoning about condition from Section 4.3.1.

4.4.2 Data Analysis for $I(\varphi, -1, 1)$

The function is internally represented by the `floatarith` expression

```
abbreviation "φ ≡ (Sqrt (2 * Pi))^(-1) * Exp (-(Var 0)^2 * 2^(-1))"
```

To obtain a general picture, both parameters are first plotted against their lower approximation bound.

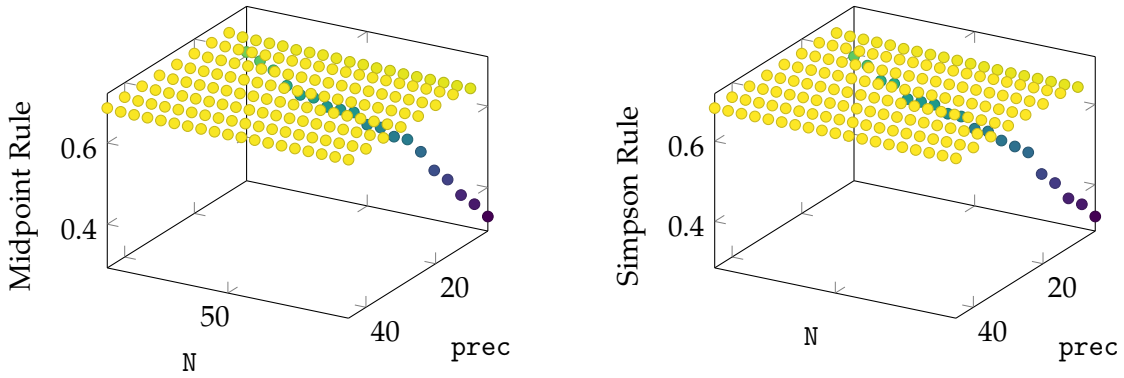


Figure 4.3: Linear plot of the lower approximation bound of $I(f, 0, 1)$ with both functions. Both parameters range from 5 to 50 in steps of 5.

It seems that condition problems do not play a role here and the function is suitable to be approximated. To determine the best approximation strategy, both parameters are plotted against the numerical error of the calculation.

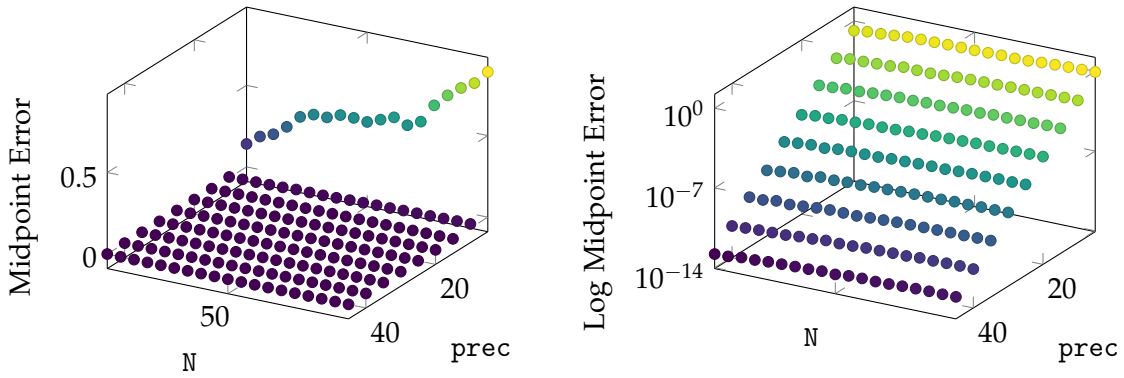


Figure 4.4: Linear and logarithmic plot of the numerical error resulting from the approximation of $I(\varphi, -1, 1)$ with the Midpoint rule. Both parameters range from 5 to 50 in steps of 5.

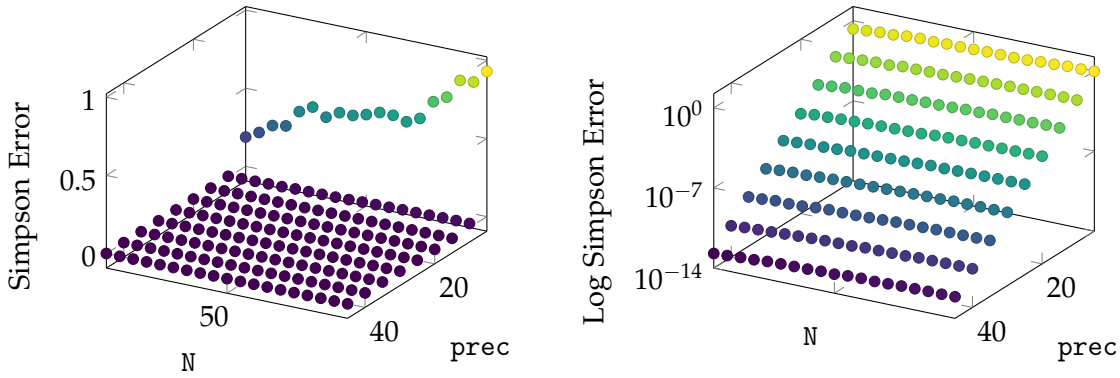


Figure 4.5: Linear and logarithmic plot of the numerical error resulting from the approximation of $I(\varphi, -1, 1)$ with the Cavalieri-Simpson rule. Both parameters range from 5 to 50 in steps of 5.

As expected, it appears that the numerical error decreases exponentially in relation to the chosen floating point precision. From the data lines with precision 5, it appears that a linear increase in interval count also leads to a linear increase in the numerical error. To support these hypotheses, the data is plotted again for two distinct precision values.

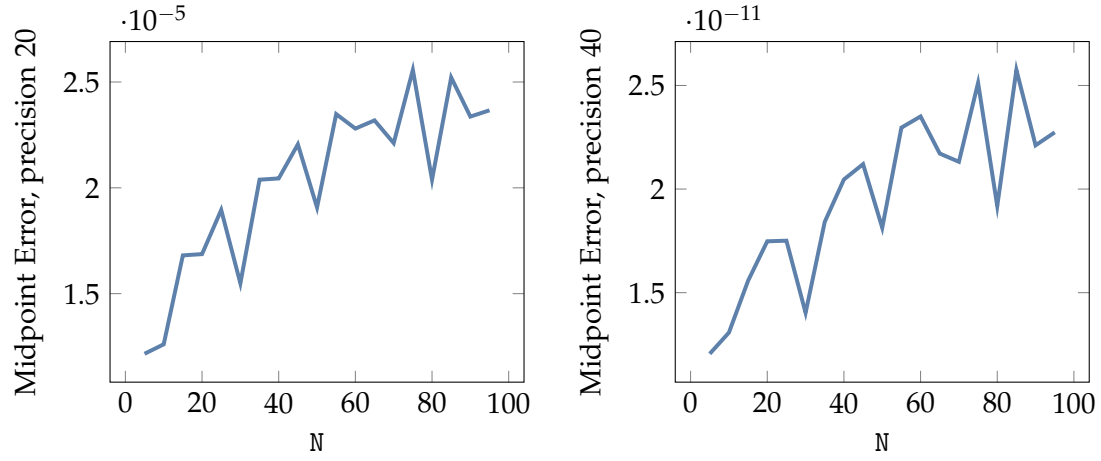


Figure 4.6: Linear plot of the numerical error resulting from the midpoint rule approximation of $I(\varphi, -1, 1)$ for `prec` $\in \{20, 40\}$ and `N` ranging from 5 to 95 in increments of 5.

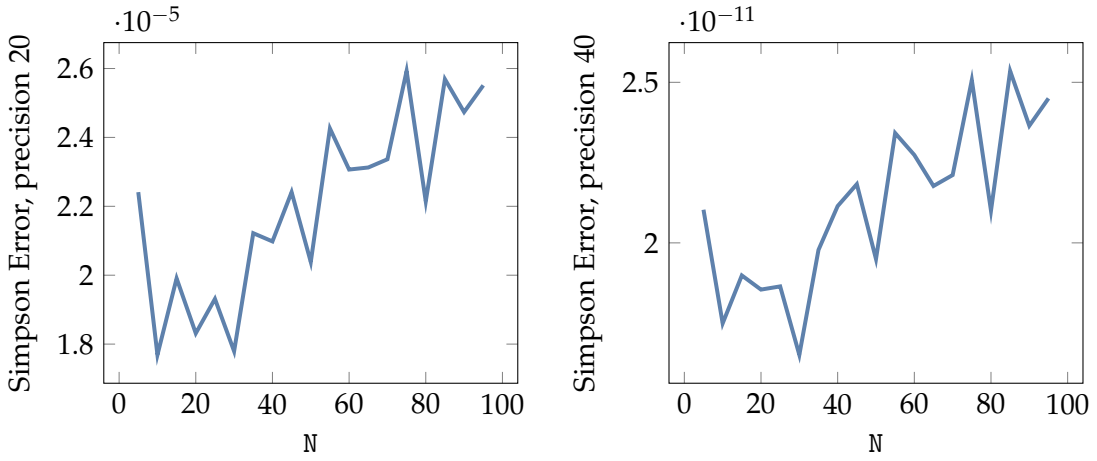


Figure 4.7: Linear plot of the numerical error resulting from the Cavalieri-Simpson approximation of $I(\varphi, -1, 1)$ for `prec` $\in \{20, 40\}$ and `N` ranging from 5 to 95 in increments of 5.

The additional diagrams confirm that the numerical error probably correlates linearly to the interval count. Furthermore, the error generally oscillates between even and odd values of n, which is investigated using the following plot.

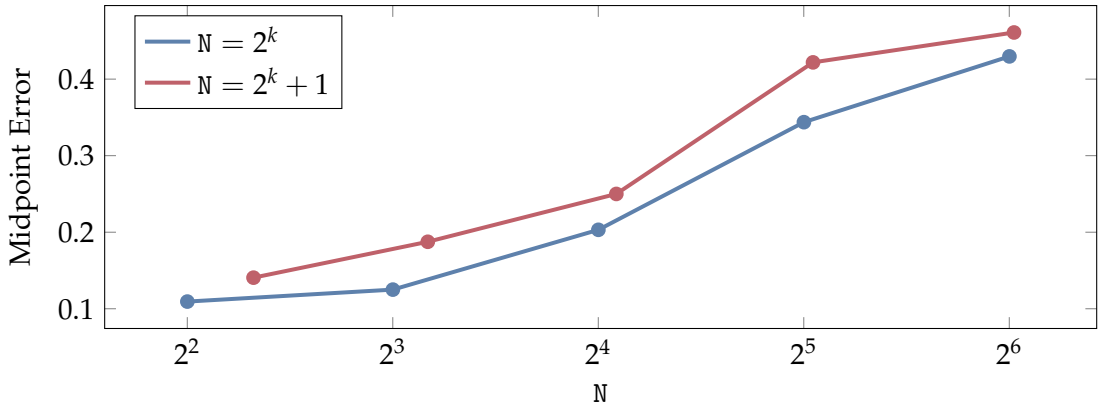


Figure 4.8: Linear plot of the numerical error resulting from the approximation of $I(\varphi, -1, 1)$ using the Midpoint rule with precision 5.

It can be seen that the error is significantly lower when the `N` is a power of two. This is a result of the difference between the division errors from the calculation of the interval width. While division by a power of two does not cause a loss of precision due to the nature of the floating point format, any other division introduces more rounding

errors.

From these findings, it is concluded that an small interval count that is a power of two minimizes the numerical error. However, it must be remembered that choosing a small `N` results in a bigger analytical error as seen in Theorem 2.26ff. To find an optimal procedure for obtaining a result of a desired precision quickly, another function is analyzed in the next subsection.

4.4.3 Data Analysis for $I(\exp, 0, 1)$

The function is internally represented by the `floatarith` expression `Exp (Var 0)`. Because all of its derivatives are equal, they are all bounded by $\rceil$ in $[0;1]$, allowing for an easy computation of the analytical error bounds. This step is not automatized yet, so for this function, a separate implementation was designed that includes the analytical error into the resulting interval, allowing the data to be evaluated in a holistic manner.

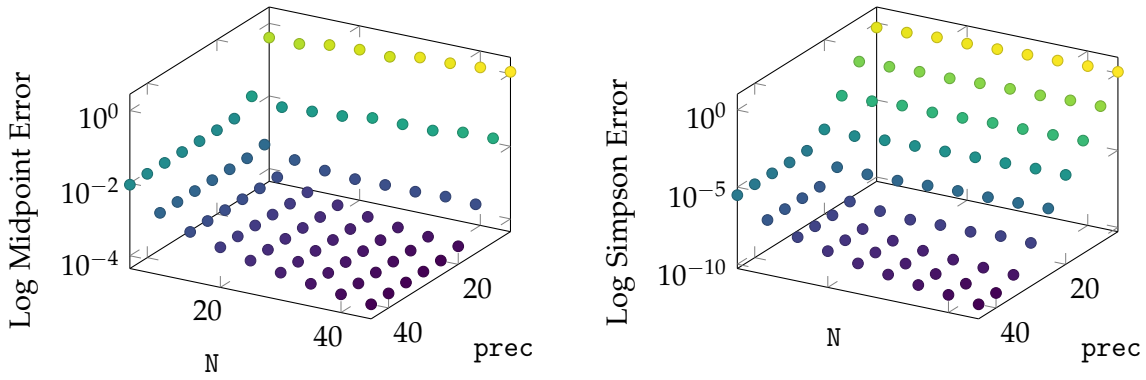


Figure 4.9: Logarithmic combined error resulting from the approximation of $I(\exp, 0, 1)$ with the Midpoint rule. Both parameters range from 5 to 45 in steps of 5.

It is clearly visible that for a given `N`, an increase in precision exponentially decreases the error up to the point where the analytical error bound is reached. Hence, it is concluded that the fastest procedure for finding the right parameters is probably to first choose the interval count to bound the analytical error to a reasonable tolerance and subsequently increase the precision until the desired accuracy is reached.

Additionally, the Cavalieri-Simpson rule seems to achieve a much higher precision than the midpoint rule for a given interval count in this particular case. This aligns with the expectations from Theorem 2.26ff and as a result, the former algorithm should be preferred in most cases.

4.5 Performance Considerations

When analyzing an algorithm, performance is usually one of the main concerns because time is often a valuable resource. In this particular case however, the speed of the specific implementations does not play a significant role for various reasons.

One one hand, both functions are designed to mainly be used for proving a goal by approximation, which requires only a single evaluation at a time, so unless the function is enormously inefficient, its computation time is negligible compared to the duration of the proving process.

On the other hand, it is extremely hard to measure the performance of an algorithm inside of Isabelle. As Hölzl noticed, the runtime of any call to `approx` can be split into the compilation time of the computation functions and the duration of the computation itself [11, p. 37]. In testing, both the midpoint algorithm and the Cavalieri-Simpson algorithm achieved extremely precise results after about one second for precision values up to 200 and interval count up to 1000 on the test system[1].

For comparison, Planck length, i.e. the shortest physically measurable distance, is about $1.6 \cdot 10^{-35}m \approx 1.1 \cdot 2^{-116}m$ [22]. No practical use case will ever require a precision even remotely close to these parameters. Considering that an evaluation of the term `approx 1 (Num 0) []` took one second as well, it was deduced that the compilation time is probably the deciding factor in the overall runtime. This hypothesis could not be confirmed as Isabelle lacks a profiler.

From these considerations, it was decided that an extensive performance analysis would not be particularly meaningful or interesting to most audiences.

[1] Intel Core i5-8600K (5.1GHz), 32GB DDR4-3200, Arch Linux, Kernel 6.4.9-zen1-1-zen, Isabelle2022-vsce-emacs (commit 9c93b59822) [6] with JEdit

5 Conclusion

The algorithmic implementation of both the midpoint rule and Cavalieri-Simpson rule and the formalization of their analytical error bounds was successfully completed and described in great detail in Chapter 2 and 3. During the examination of the numerical properties in Chapter 4, it was found that both algorithms are able to calculate precise approximations for well-conditioned problems and a quick strategy was found to choose the approximation parameters according to the specific precision requirements of any given problem. However, reaching the original goal of extending Hölzl's proof tactic [10] to support inequations on integrals still requires a significant amount of work.

5.1 Future Work

For some functions, neither of the presented algorithms may yield an acceptably precise approximation and another algorithm may be suited better. Expanding the available repertoire with new implementations may help to cover even more integration problems sufficiently.

To automatically prove inequalities between integrals, the reification process and the underlying `floatarith` datatype would obviously need to be modified to support at least one of the definitions mentioned in Section 3.2. This task should not be too complicated.

Furthermore, the proof procedure would need to obtain an approximation containing both the numerical and the analytical error of the computation. While the former is already included, calculating the latter involves finding bounds for the derivatives of the integrand, which has not been automated yet at the time of writing this thesis. Presumably, the implementation effort would be around equal to the scope of this thesis.

A formal proof of the equivalences between the mathematical representation of both rules and their algorithmic counterparts would have exceeded the scope of this thesis greatly and thus remains to be conducted. Due to the technical details of the approximation library, this step probably presents the greatest challenge for the future.

List of Figures

Bibliography

[1] I. national de recherche en informatique et en automatique. *The Coq Proof Assistant*. 1989. URL: https://coq.inria.fr.

[2] C. Brell, J. Brell, S. Kirsch, C. Brell, J. Brell, and S. Kirsch. "Wahrscheinlichkeitstheorie." In: *Statistik von Null auf Hundert: Mit Kochrezepten schnell zum Statistik-Grundwissen* (2017). DOI: 10.1007/978-3-662-53632-2_9.

[3] A. Chaieb. "Automated methods for formal proofs in simple arithmetics and algebra." PhD thesis. Technische Universität München, 2008. URL: https://mediatum.ub.tum.de/doc/649541/document.pdf.

[4] M. Estepho. *Approximate Integration: Midpoint Rule Error Bound: Proof. Math Easy Solutions*. Last accessed on 2023-08-07. 2015. URL: https://peakd.com/mathematics/@mes/approximate-integration-midpoint-rule-error-bound-proof.

[5] M. Estepho. *Simpson's Rule Approximation: Error Bound Proof. Math Easy Solutions*. Last accessed on 2023-08-07. 2016. URL: https://peakd.com/mathematics/@mes/simpson-s-rule-approimation-error-bound-proof.

[6] M. Fleury. *isabelle-emacs*. URL: https://github.com/m-fleury/isabelle-emacs.

[7] A. Fonda and Heine. *The Kurzweil-Henstock Integral for Undergraduates*. Springer, 2018. DOI: 10.1007/978-3-319-95321-2.

[8] O. Forster. *Analysis 1*. Springer, 2016. DOI: 10.1007/978-3-658-11545-6.

[9] O. Forster. *Analysis 3*. Springer, 2017. DOI: 10.1007/978-3-658-16746-2.

[10] J. Hölzl. "Proving inequalities over reals with computation in Isabelle/HOL." In: *Proceedings of the ACM SIGSAM 2009 International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS'09)*. 2009, pp. 38–45. URL: https://home.in.tum.de/~hoelzl/documents/hoelzl09realinequalities.pdf.

[11] J. Hölzl. "Proving Real-Valued Inequalities by Computation in Isabelle/HOL." Diploma thesis. Institut für Informatik, Technische Universität München, Apr. 2009. URL: https://home.in.tum.de/~hoelzl/documents/hoelzl09diplomathesis.pdf.

[12] J. Hölzl and F. Immler. *HOL/Library/Float.thy*. 2012. URL: https://isabelle.in.tum.de/repos/isabelle/file/4a2c9d32e7aa/src/HOL/Library/Float.thy.

[13] T. Kamm. *Darstellungsbereich von Gleitkommazahlen*. Licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License (CC BY-NC-SA 4.0). 2022. URL: https://openasp.aengelke.net/videos/floating-point/slides.pdf.

[14] A. Mahboubi, G. Melquiond, and T. Sibut-Pinote. "Formally Verified Approximations of Definite Integrals." In: *Journal of Automated Reasoning* 62.2 (Feb. 2019), pp. 281–300. ISSN: 1573-0670. DOI: 10.1007/s10817-018-9463-7.

[15] E. Makarov and B. Spitters. "The Picard Algorithm for Ordinary Differential Equations in Coq." In: *Proceedings of the 4th International Conference on Interactive Theorem Proving*. ITP'13. Rennes, France: Springer-Verlag, 2013, pp. 463–468. ISBN: 9783642396335. DOI: 10.1007/978-3-642-39634-2_34.

[16] T. Nipkow. *Programming and proving in Isabelle/HOL*. Last accessed on 2023-08-07. URL: https://isabelle.in.tum.de/dist/Isabelle2022/doc/prog-prove.pdf.

[17] T. Nipkow, M. Wenzel, and L. C. Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002. DOI: 10.1007/3-540-45949-9.

[18] S. Obua and T. Nipkow. "Flyspeck II: the basic linear programs." In: *Annals of Mathematics and Artificial Intelligence* (2009). DOI: 10.1007/s10472-009-9168-z. URL: https://doi.org/10.1007/s10472-009-9168-z.

[19] A. Quarteroni, R. Sacco, and F. Saleri. *Numerische Mathematik 1*. Springer Berlin, 2002. ISBN: 9783540678786.

[20] A. Quarteroni, R. Sacco, and F. Saleri. *Numerische Mathematik 2*. Springer Berlin, 2002. DOI: 10.1007/978-3-642-56191-7.

[21] H.-R. Schwarz and N. Köckler. *Numerische Mathematik*. Springer-Verlag, 2013.

[22] U. The NIST Reference on Constants and Uncertainty. *2018 CODATA Value: Planck length*. URL: https://physics.nist.gov/cgi-bin/cuu/Value?plkl.

[23] C. Zimmerer. *Isabelle-Numerical-Integration*. 2023. URL: https://github.com/halbGefressen/Isabelle-Numerical-Integration.