Computing Interval Range Approximations for Smooth Real Functions with Applications in Real-root Isolation

by

Ya Shi Zhang

A thesis submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Arts

Department of Computer Science

New York University

May, 2023

Professor Chee Yap

To my family, whose unwavering love and encouragement leave me forever grateful.

Acknowledgements

I would like to express my deepest appreciation to the following individuals and organizations for their invaluable support and assistance throughout the completion of this thesis:

First and foremost, I am grateful to my thesis advisor Chee K. Yap for his unwavering guidance, patience, and expertise. His insightful comments, constructive criticism, and encouragement have been critical to the development of this thesis.

I am thankful to Professor Kai Hormann of Università della Svizzera italiana for his invaluable contributions in the writing of the original preprint, assistance with computational experiments, and for being a kind mentor.

I am grateful to the faculty and staff of the Courant Institute of Mathematical Sciences, the Center for Data Science, and New York University for providing me with a stimulating and supportive academic environment. Their dedication to excellence in teaching and research has been an inspiration to me throughout my academic journey.

I am especially grateful to Dr. McDonald, my high school mathematics teacher, for inspiring all of my endeavors in mathematics and subsequently computer science. From my expository on Boolean rings and algebras in high school, I've come a long way in developing my mathematical maturity, though I still do not understand category theory.

I would like to express my appreciation to my family and friends for their constant love, encouragement, and support. Their unwavering belief in me has been a source of motivation and inspiration.

Preface

Many people learn about polynomials in middle school and high school. Quadratic polynomials are investigated in various depths, depending on mathematical maturity. When I was in middle school, I had not payed mathematics any attention. I vividly remember being asked to find the roots of the polynomial $x^2 - 3x + 2$ via factorization, where I could not answer.

It was not until almost a year afterwards when I found my passion in mathematics and began seriously studying it. At the time, I was particularly attracted to abstract algebra, notably groups, rings, and algebras. While writing my high school's expository essay, I encountered the non-solvability of the quintic polynomial (and more generally, the Abel-Ruffini theorem) and algebraic numbers. It wouldn't be until this year, when I learned Galois theory, that I would come to fully understand why.

In my third year at New York University, Chee introduced me to range functions and their applications in real-root isolation (which will be discussed later). I was immediately drawn in. As an undergraduate student, I am very delighted to have contributed new methods that aid in finding roots of polynomials, a topic that has piqued my interest from the beginning.

It is also slightly comical that I began my journey unable to factor a quadratic to find its roots, to now having played a role in developing a theoretical framework that approximates any polynomial (including those with degree greater than or equal to 5) to any arbitrary precision. I hope you enjoy reading this as much as I enjoyed working on this.

Note: this thesis will contain a lot novel material found in a recent preprint that Professor Kai Hormann, Professor Chee Yap, and I worked on. As such, this thesis can be viewed as a self-contained and motivating expository of the paper.

Abstract

We address the fundamental problem of computing range functions $\square f$ for a real function $f : \mathbb{R} \to \mathbb{R}$. In our previous work in ISSAC 2021, we introduced a family of **recursive interpolation range functions** based on the Cornelius–Lohner (CL) framework of decomposing f as $f = g + R$. The CL framework requires computing $g(I)$ "exactly" for an interval I. This requirement is impossible to fulfil in practice and limits the order of convergence to 6.

We generalize the CL framework by allowing $g(I)$ to be approximated by **strong range functions** $\square g(I; \varepsilon)$, where $\varepsilon > 0$ is a user-specified bound on the error. This new framework allows, for the first time, the design of interval forms for f with any desired order of convergence. To achieve our strong range functions, we generalize Neumaier's theory of constructing range functions from expressions over a Lipschitz class Ω of primitive functions. We show that the class Ω is very extensive and includes all common hypergeometric functions.

Traditional complexity analysis of range functions is based on individual evaluation on an interval. Such analysis cannot differentiate between our novel recursive range functions and classical Taylor-type range functions. Empirically, our recursive functions are superior in the "holistic" context of the root isolation algorithm Eval. We now formalize this holistic approach by defining the **amortized complexity** of range functions over a subdivision tree. Our theoretical model agrees remarkably well with the empirical results.

Among our previous novel range functions, we identified a Lagrange-type range function $\square_3^{L'} f$ as the overall winner. In this paper, we introduce a Hermite-type range function $\square_4^{H} f$ that is even better.

Contents

Dedication iv

Acknowledgments v

Preface vi

Abstract vii

List of Figures x

List of Tables xi

1 Introduction 1
- 1.1 Motivating Interval Arithmetic . . . 2
- 1.2 Preliminaries . . . 2
  - 1.2.1 Interval Arithmetic . . . 2
  - 1.2.2 Range Functions . . . 6
  - 1.2.3 Box Forms . . . 7

2 Review of Literature 9
- 2.1 Cornelius & Lohner Form . . . 10
  - 2.1.1 Higher Dimensional Extensions . . . 12
- 2.2 K. Hormann, L. Kania, and C. Yap 2021 . . . 13
  - 2.2.1 Classical Taylor Forms . . . 13
  - 2.2.2 Taylor Forms with Higher Orders of Convergence . . . 14
  - 2.2.3 Recursive Interpolation Forms . . . 15
  - 2.2.4 Recursive Lagrange Forms with Cubic Convergence . . . 16
  - 2.2.5 A Cheaper Variant of the Recursive Lagrange Form . . . 17
  - 2.2.6 Recursive Lagrange Forms with Quartic Convergence . . . 18

3 Results 20
- 3.1 Overview . . . 20
- 3.2 Generalizing the Cornelius & Lohner Framework . . . 20

3.3 Precision-Bounded Box Forms . . . 22
3.3.1 Lipschitz Expressions . . . 22
3.4 Quartic Range Functions using Hermite Interpolation . . . 25
3.5 Complexity Analysis of Practical Range Functions . . . 27
3.5.1 Amortized Complexity of the Cheap Cubic Lagrange Form . . . 28
3.5.2 Amortized Complexity of the Quartic Hermite Form . . . 29
3.5.3 Amortized Complexity for General Hermite Forms . . . 29

4 Experiments **31**
4.1 Real-Root Isolation and the Eval Algorithm . . . 31
4.2 Estimating the Range of the Derivative . . . 32
4.2.1 Generalized Taylor Forms . . . 32
4.2.2 Cubic Lagrange Forms . . . 33
4.2.3 Quartic Hermite Forms . . . 34
4.3 Methodology and Settings . . . 34
4.4 Results . . . 35
4.4.1 Non-Maximal Recursion Levels . . . 35
4.4.2 Running Times . . . 38

5 Conclusion **40**
5.1 Future Work . . . 40

A Appendix **41**

Bibliography **47**

List of Figures

1.1 Example of an Ω_0-expression 5
1.2 Natural Interval Extension Tree of Example 1.17 7

4.1 Speedup $\sigma(\ell)$ of $\mathrm{E}_{3,\ell}^{L'}$ (left) and $\mathrm{E}_{4,\ell}^{H'}$ (right) 37
4.2 Speedup of $E_4^{H'}$ over $E_3^{L'}$ with different polynomials and degrees 38

List of Tables

4.1 Size of the Eval Subdivision Tree 36
4.2 Average Running Time of Eval with 1024-bit Floating Point Arithmetic in Seconds. 39
4.3 Average Running Time of Eval with Multi-Precision Rational Arithmetic in Seconds. 39

1 | Introduction

Suppose we are given a function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}, \mathcal{Y}$ are sets, as well as a subset $X \subseteq \mathcal{X}$. We are interested in an algorithm that, given any such f and X, will output the image of X under f, i.e. we wish to compute $Y \coloneqq f(X) = \{y \in \mathcal{Y} \mid \exists x \in X,\, f(x) = y\}$. If X is a finite set, then this problem becomes trivial. If X is at least countably infinite, then this problem becomes intractable. In this thesis, we focus on a tractable and applicable relaxation of the problem.

In particular, let $\mathcal{X} = \mathbb{R}^n$ for some $n \geq 1$, $\mathcal{Y} = \mathbb{R}$ and suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is smooth, i.e. $f \in C^\infty(\mathbb{R}^n; \mathbb{R})$, where $C^\infty(\mathbb{R}^n; \mathbb{R})$ is the class of functions with domain $\mathbb{R}^n$ and codomain $\mathbb{R}$ and has continuous derivatives of any order. Though this is a stronger-than-needed assumption, it will simplify our analysis later on significantly. Furthermore, assume the subsets $X \subseteq \mathbb{R}^n$ belong to the set of boxes in $\mathbb{R}^n$ denoted as $\square\mathbb{R}^n \coloneqq \{[a_1, b_1] \times \ldots \times [a_n, b_n] \mid a_i \leq b_i \in \mathbb{R},\, 1 \leq i \leq n\}$. Moreover, for any connected subset $\mathcal{D} \subseteq \mathbb{R}^n$, we can denote the set of all intervals (or interval boxes) over $\mathcal{D}$ as $\square\mathcal{D}$. Even now, we can still come up with functions that, under arbitrary interval inputs, have images that are intractable for modern computers to compute exactly. For example, take $f(x) = e^x \sin x$ and $X = [\sqrt{2}, \pi\sqrt{2}]$.

Now, we change the objective of the problem from computing the exact range $Y = f(X)$ to finding a tight approximation $\tilde{Y} \supseteq Y$, preferably with an error bound that we can specify a priori. This notion of an error bound will be defined in 1.2.

Finally, we wish to compute the approximation such that it is 'exact' (or 'certified'), where the output of our algorithm must contain the ground truth up to rounding and computational errors. This is because modern computer architectures represent real numbers as either a floating point representation or a rational representation. In either case, it is infeasible to represent numbers such as $\sqrt{2}$ exactly. Consider $f(x) = \sin x$ and $X = [0, 1]$, while we can reason abstractly that $f(X) = [0, \sin 1]$, computing $\sin 1$ and storing it in a computer exactly is not feasible. Most computers would approximate the value using, for example, a Taylor approximation for $\sin x$. However, the truncated Taylor series for $\sin x$ may be an underestimate if we use an even number of terms, and the resulting output from the computer will not contain the true range.

Out of the many popular methods that remedies these issues that lie in the heart of numerical computing, a popular candidate is to extend our notion of arithmetic to intervals.

1.1 Motivating Interval Arithmetic

To sufficiently motivate our discussion for 1.2.1, let us consider a specific problem. We wish to compute a person's body mass index (BMI) under uncertainty about the person's body weight and determine whether their body mass index falls under the category of normal, underweight, or overweight, represented by $(18.5, 25)$, $(-\infty, 18.5]$, and $[25, \infty)$, respectively. Here, $f(m, h) = m/h^2$.

If a person who is 1.80m tall steps on the scale with known uncertainty ±0.5kg and the scale reads 80kg, it indicates the person has a true weight in the interval $[79.5, 80.5]$. It is, of course, implausible that this person weighs exactly 80kg. Hence, if we carry out the BMI calculation and get 24.69, we cannot be certain of this number. Alternatively, we can carry out the calculation using the uncertainty interval and get $[24.54, 24.85]$. Now, taking in all of the uncertainties into account, we can 'certify' this person is of normal weight.

This is a simple example, so we want to consider appropriate generalizations of this motivating idea. In the following section, we will build upon this idea and introduce primitive tools in the field of interval arithmetic and range functions. Afterwards in 2, we will explore frameworks and contemporary papers that we will either build upon or rely on for derivations of important properties.

1.2 Preliminaries

I will first introduce basic arithmetic operations on intervals and some of their idiosyncrasies. Afterwards, we will consider extending functions of the form $f : \mathbb{R}^n \to \mathbb{R}$ to $\square f : \square\mathbb{R}^n \to \square\mathbb{R}$. We will encounter some problems that will eventually need us to define the notion of range functions and box forms.

1.2.1 Interval Arithmetic

Suppose that we are given two intervals $[a, b], [\alpha, \beta] \in \square\mathbb{R}$ and a scalar $\lambda \in \mathbb{R}$, then we have the following definitions for interval arithmetic in $\square\mathbb{R}$, which can easily be extended to boxes in $\square\mathbb{R}^n$ via component-wise operations (i.e. viewing $\square\mathbb{R}^n$ as an n-dimensional vector over $\square\mathbb{R}$):

- Magnitude:

$$|[a, b]| = \max\{a, b\} \tag{1.1}$$

- Width:

$$w([a, b]) = b - a \tag{1.2}$$

- Radius:

$$r([a, b]) = (b - a)/2 \tag{1.3}$$

- Midpoint:

$$m([a,b]) = (a+b)/2 \tag{1.4}$$

- Hausdorff Distance:

$$d_H([a,b],[\alpha,\beta]) = \max\{|a-\alpha|, |b-\beta|\} \tag{1.5}$$

- Scalar Addition:

$$\lambda + [a,b] = [a+\lambda, b+\lambda] \tag{1.6}$$

- Addition:

$$[a,b] + [\alpha,\beta] = [a+\alpha, b+\beta] \tag{1.7}$$

- Subtraction:

$$[a,b] - [\alpha,\beta] = [a-\beta, b-\alpha] \tag{1.8}$$

- Scalar Multiplication:

$$\lambda[a,b] = \begin{cases} [\lambda a, \lambda b] & \text{if } \lambda \geq 0 \\ [\lambda b, \lambda a] & \text{if } \lambda < 0 \end{cases} \tag{1.9}$$

- Multiplication:

$$[a,b] \times [\alpha,\beta] = [\min S, \max S],\ S \coloneqq \{a\alpha, a\beta, b\alpha, b\beta\} \tag{1.10}$$

- (Multiplicative) Inversion:

$$\frac{1}{[a,b]} = \begin{cases} [\frac{1}{b}, \frac{1}{a}] & \text{if } 0 \notin [a,b] \\ (-\infty, \frac{1}{a}] \cup [\frac{1}{b}, +\infty) & \text{if } 0 \in [a,b] \\ [\frac{1}{b}, +\infty) & \text{if } a = 0 \\ (-\infty, \frac{1}{a}] & \text{if } b = 0 \end{cases} \tag{1.11}$$

Above, the Hausdorff distance will be our notion of an 'error bound' when discussing intervals, which also forms a complete metric topology over $\square\mathbb{R}^n$ [1].

A useful representation for intervals is that they can be decomposed into a centered interval (about 0) with an additive component.

[1] For a more detailed and mathematical development of the interval system with applications to other problems such as deriving global error bounds for differential equations, computer-assisted proofs, and more, please refer to [Moore et al. 2009], [Moore 1979], [Alefeld et al. 2012].

Lemma 1.1. *For any* $I = [a, b] \in \square\mathbb{R}$, $I = m(I) + \frac{1}{2}w(I)[-1, 1]$.

By inspection, we see that addition and multiplication are commutative. However, there are peculiarities with interval arithmetic that leads to information loss under multiplication and inversion that we want to mitigate. The most notable example is that interval multiplication is not distributive over addition.

Lemma 1.2 ('Sub-distributivity' of Interval Multiplication over Addition)**.** *For all* $a, b, c \in \square\mathbb{R}$,

$$a \cdot (b + c) \subseteq a \cdot b + a \cdot c$$

This is best illustrated by the following example.

Example 1.3. Suppose we wish to compute $[1, 2] \cdot ([1, 2] - [1, 2])$, then

$$[1, 2] \cdot ([1, 2] - [1, 2]) = [1, 2] \cdot [-1, 1] = [-2, 2], \text{ but}$$
$$[1, 2] \cdot [1, 2] - [1, 2] \cdot [1, 2] = [1, 4] - [1, 4] = [-3, 3]$$

This implies that we must be careful about our order of operations in order to minimize information lost. With these primitive tools, we are already able to compute exact ranges for affine functions, say $f(\boldsymbol{x}) = a \cdot \boldsymbol{x} + b$. In fact, we are able to find the ranges of any monotonic function by simply outputting the values of the function at the endpoints of the interval.

Example 1.4. We can naturally extend the exponential function to intervals.

$$\exp([a, b]) \coloneqq [\exp a, \exp b]$$

Furthermore, we can calculate the range for functions that can be decomposed into piece-wise monotonic functions by also taking the 'critical points' into consideration.

Example 1.5. Let $f(x) = \sin x$, and suppose we wish to compute $\sin[0, \pi]$. On this interval, $\sin x$ can be decomposed into two piece-wise monotonic functions, one defined on $[0, \pi/2]$ and the other defined on $(\pi/2, \pi]$. In addition to computing $\sin x$ at its endpoints, we must also consider the critical point $x = \pi/2$.

$$\sin[0, \pi] = [\min S, \max S],\ S = \{\sin 0, \sin \pi/2, \sin \pi\} = \{0, 1\}$$
$$\therefore \sin[0, \pi] = [0, 1]$$

Lemma 1.6. *Let* $f : \mathbb{R}^n \to \mathbb{R}$ *be a piece-wise monotonic function, then* $f(I)$ *can be theoretically computed exactly* $\forall I \in \square\mathbb{R}^n$.

Proof. Let $I = [a, b]$. We can generalize the procedure in Example 1.5: For each 'critical point' of the function in I (i.e. when the function changes direction), evaluate the function at that point, then evaluate $f(a)$ and $f(b)$. Then the minima and maxima of this set will be exactly $f(I)$. □

We can now combine this knowledge with our prior knowledge of interval arithmetic to compute approximate ranges of a wider class of functions. We must accept that our range will be a superset of the true range due to the sub-distributivity of interval multiplication. Let us first define the notion of 'expressions'.

Definition 1.7. A partial function $f : \mathcal{X} \to \mathcal{Y}$ is a function from some subset of $X \subseteq \mathcal{X}$ to $\mathcal{Y}$, i.e. some elements in the domain are undefined.

If $S = \mathcal{X}$, then f is a function. A common example of a partial function is division over the real numbers. $\div : \mathbb{R}^2 \to \mathbb{R}$, $\div(a, b) = a \div b$ is defined for all 2-tuples except for the subset $\{(a, 0) \mid a \in \mathbb{R}\}$.

Definition 1.8. An expression class Ω is a set of algebraic operations such that each $\omega \in \Omega$ is an n-ary operation defined by a partial function $f_\omega : \mathbb{Z}^n \to \mathbb{C}$.

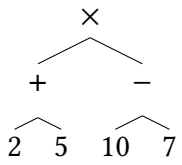
Some of the most common expression classes are:

- $\Omega_0 = \{\pm, \times\}$,
- $\Omega_1 = \{\pm, \times, \div\}$
- $\Omega_2 = \{\pm, \times, \div, \sqrt[n]{\cdot}\}$
- $\Omega_3 = \{\pm, \times, \div, \sqrt[n]{\cdot}, \text{Root}(\cdot)\}$

Where $\text{Root}(\cdot) : \mathbb{Z}^n \to \mathbb{C}$ takes in a sequence of integers that represents the coefficients of the polynomial in $\mathbb{Z}[x]$ and outputs the largest root of the polynomial [Li and Yap 2001]. In the case where we have conjugate complex roots of the same norm, we choose the root with the positive imaginary component.

Definition 1.9. An expression over Ω (also known as a Ω-expression) is a tree such that each non-leaf node is labeled with some n-ary operation $\omega \in \Omega$ and has n incoming child nodes. The leaf nodes are labeled with integers.

Figure 1.1: Example of an Ω_0-expression



Given any Ω_1-expression, we can extend the expression tree to take in intervals in the leaf nodes. This is called the *natural interval extension* of an expression.

Furthermore, we can also allow piece-wise monotonic functions to be included as nodes, i.e. these functions can act as n-ary operations. Using this and a composition of piece-wise monotonic functions, we can compute a larger class of functions.

1.2.2 Range Functions

We first introduce the two properties that any range function must have.

Definition 1.10. A function $\mathbf{f} : \square\mathbb{R}^n \to \square\mathbb{R}$ is an *interval extension* of $f : \mathbb{R}^n \to \mathbb{R}$ if for any $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and its corresponding degenerate interval $X \coloneqq ([x_1, x_1], \ldots, [x_n, x_n])$, we have $\mathbf{f}(X) = f(x)$.

Definition 1.11. A function $\mathbf{f} : \square\mathbb{R}^n \to \square\mathbb{R}$ is *inclusion isotonic* if $\forall I \subseteq J \in \square\mathbb{R}^n$, $\mathbf{f}(I) \subseteq \mathbf{f}(J)$.

Now, a range function for a function is simply a function over $\square\mathbb{R}^n \times \square\mathbb{R}$ that is both an interval extension and is inclusion isotonic.

Definition 1.12. Let $f : \mathbb{R}^n \to \mathbb{R}$ be any function, a *range function* for f is a function $\square f : \square\mathbb{R}^n \to \square\mathbb{R}$ such that for all $I \in \square\mathbb{R}$, $f(I) = \{f(x) \mid x \in I\} \subseteq \square f(I)$. If $\square f(I) = f(I)$ for all $I \in \square\mathbb{R}$, then $\square f$ is an *exact range function.*

Fixing the function f, we can also define a partial ordering on the class of range functions for f [Hormann et al. 2021].

Definition 1.13. Let $\square_1 f$, $\square_2 f$ be two ranges functions for f. Then $\square_1 f \preceq \square_2 f$ if $\forall I \in \square\mathbb{R}$, $\square_1 f(I) \subseteq \square_2 f(I)$. That is, $\square_1 f$ is *as tight as* $\square_2 f$.

Finally, we can define the natural interval extension of a function defined by an expression, which is a very important concept that we will be utilizing.

Definition 1.14. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that can be represented by an expression, then the natural interval extension $\mathbf{f} : \square\mathbb{R}^n \to \square\mathbb{R}$ is defined by that expression, by replacing inputs $\{x_1, \ldots, x_n\}$ and arithmetic operations by intervals $\{I_1, \ldots, I_n\}$ and interval operations, respectively.

An important subclass of functions that have *exact* natural interval extensions are those whose variables only appear once in the expression.

Theorem 1.15. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a function that can be represented by an expression, if the variables $\{x_1, \ldots, x_n\}$ only appear once in the expression, then its natural interval extension $\mathbf{f}$ is exact. i.e. $\mathbf{f}(I) = f(I)$, $\forall I \in \square\mathbb{R}^n$.*

Proof. Of course, we have $f(I) \subseteq \mathbf{f}(I)$ by the fact that natural interval extensions are range functions. For the other inclusion, we argue inductively over the tree expression, starting from the leaf nodes, using the exactness (or sub-distributivity) of interval arithmetic. □

We can extend this class if we recall our ability to find exact ranges for piece-wise monotonic functions.

Lemma 1.16. *Let $\overline{\Omega} \coloneqq \Omega_1 \cup \mathrm{PM}(\mathbb{R}^n; \mathbb{R})$, where $\mathrm{PM}(\mathbb{R}^n; \mathbb{R})$ is the set of all piece-wise monotonic functions $f : \mathbb{R}^n \to \mathbb{R}$. Then the set of all functions that can be presented by a subset of $\overline{\Omega}$-expressions on each input has a range function.*

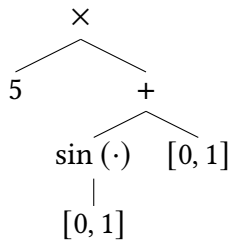
Proof. Let f be a function as described above and let $I \in \square\mathbb{R}$ be an input. View $f(I)$ as an expression tree. For each inner node, if the label is a piece-wise monotonic function, use Lemma 1.6, otherwise use interval arithmetic. Evaluating bottom-up, we arrive at an interval that surely contains $f(I)$. □

Example 1.17. Approximate the range of $f([0, \pi])$, where $f(x) = 5 \cdot (x + \sin x)$.
We know that $\sin [0, \pi] = [0, 1]$, and now we can extend our expression tree to approximate the range.

$$f([0, \pi]) \subseteq [0, 5 + 5\pi]$$

$f(x)$ is non-decreasing, so we can find that the precise range is $[0, 5\pi] \subset [0, 5 + 5\pi]$.

Figure 1.2: Natural Interval Extension Tree of Example 1.17



However, in practice, we do not know how a general function will decompose into monotonic pieces. This is problematic as our algorithm does not know a priori which function it needs to compute. Additionally, if interval division is needed at any point in the expression, then we may encounter intervals involving infinity, which destroys our approximation to the true range. Finally, computational errors arise in real-world computations, meaning that exact ranges of the above functions may not be exact when computing them using floating point or rational arithmetic.

With these limitations in mind, we want to forego the idea of computing intervals exactly, but still wish to compute approximate ranges for functions that are 'good enough' to use in applications where a guaranteed range is crucial.

1.2.3 Box Forms

Let us first develop definitions that express what we desire in range functions.

Definition 1.18. Let $f : \mathbb{R}^n \to \mathbb{R}$ be any real-valued function and $\square f : \square\mathbb{R}^n \to \square\mathbb{R}$ be a range function for f. Then $\square f$ has *order k convergence* with respect to an interval box $\mathcal{I} \subseteq \square\mathbb{R}^n$ if there exists $C \in \mathbb{R}$ such that for all $I \subseteq \mathcal{I}$, we have $d_H(\square f(I), f(I)) \leq Cw(I)^k$. If $k \geq 1$, then $\square f$ is *convergent* on $\mathcal{I}$.

From here on in, we assume that when we say $\square f$ is convergent, it means that the range function is convergent on the entire domain of f. Below is an important property of convergent range functions.

Lemma 1.19. *Let f be a real-valued function and let $\square f$ be a convergent range function for f. Then for any sequence of intervals $(I_i)_{i\geq 1}$ that converges monotonically to a fixed point $p \in I_1$, we have*

$$\lim_{i\to\infty} \square f(I_i) = f(p)$$

Proof. Since $\square f$ is convergent, we must have $d_H(\square f(I), f(I)) \leq Cw(I)$ for any I. Viewing p as an interval $I = [p, p]$, we have

$$d_H(\square f(I), f(I)) = d_H(\lim_{i\to\infty} \square f(I_i), f(p)) \leq Cw(I) = 0$$

Hence

$$|\lim_{i\to\infty} \square f(I_i) - f(p)| = 0 \implies \lim_{i\to\infty} \square f(I_i) = f(p)$$

. □

Definition 1.20. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real-valued function, a range function $\square f : \square\mathbb{R}^n \to \square\mathbb{R}$ is a *box form* of f if it is

(1) Conservative: $\square f(I) \supseteq f(I)$ for all I.

(2) Convergent: $\square f$ is convergent.

For any function that is k times differentiable, it can exhibit a *Taylor interval extension* that is also a box form for f via the use of Taylor's theorem followed by a natural extension of the Taylor form.

Lemma 1.21. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be k times differentiable in a box $I \in \square\mathbb{R}^n$, where $k \geq 1$,then f has a range function.*

Proof. Using Taylor's theorem, we have for every $x \in I$

$$f(x) = \sum_{i=0}^{k-1} \frac{1}{i!} D^i f(\overline{x}) \cdot (x - \overline{x})^i + \frac{1}{(k)!} D^k f(\xi)(x - \overline{x})^k$$

for any $\overline{x} \in I$, say $m(I)$, and some $\xi \in I$. We now replace the above Taylor expression with its natural interval extension, i.e. $x \leftarrow I$, $\xi \leftarrow I$. □

This idea of a Taylor interval extension will be very important for our later discussion and results. We now have developed sufficient material in order to review contemporary literature in this topic.

2 | Review of Literature

In this section, we aim to first explore important or contemporary literature that has contributed significantly to deriving frameworks and algorithms that provide accurate and fast-converging box forms for a wide class of functions. For simplicity, let us assume that we are working exclusively with functions of the form $f : \mathbb{R} \to \mathbb{R}$.

In Ramon E. Moore's foundational textbook [Moore et al. 2009], we see important first-order and second-order range functions for real-valued functions satisfying Lipschitz and/or differentiable assumptions.

Definition 2.1. A function $f : \mathbb{R} \to \mathbb{R}$ is said to be L-Lipschitz, where $L \geq 0$, if

$$\forall x, y \in \mathbb{R},\ |f(x) - f(y)| \leq L \cdot |x - y|$$

Definition 2.2. An interval extension $\mathbf{f} : \square\mathbb{R} \to \square\mathbb{R}$ for $f : \mathbb{R} \to \mathbb{R}$ is said to be *L-Lipschitz* over an interval $\mathcal{I} \in \square\mathbb{R}$ if

$$w(\mathbf{f}(I)) \leq L \cdot w(I),\ \forall I \subseteq \mathcal{I}$$

We can now begin with some theorems about box forms with linear or quadratic convergence as discussed in [Moore et al. 2009].

Theorem 2.3. *If $f : \mathbb{R} \to \mathbb{R}$ is L-Lipschitz, then its natural interval extension is a box form with convergence order ≥ 1.*

Proof. Refer to [Moore et al. 2009]. □

Theorem 2.4. *If $f : \mathbb{R} \to \mathbb{R}$ is differentiable such that its derivative f' is L-Lipschitz in the interval $\mathcal{I} \in \square\mathbb{R}$, then the mean value form*

$$\mathbf{f}(I; \overline{x}) := f(\overline{x}) + f'(I) \cdot (I - \overline{x})$$

for any $\overline{x} \in \mathcal{I}$ and $I \subseteq \mathcal{I}$ converges with order 2.

Proof. Refer to [Moore et al. 2009]. □

Up until [Cornelius and Lohner 1984], many articles were published that described range functions with quadratic convergence, most used some specific instance of *centered form expressions*

$$\mathbf{f}(I) = f(\overline{x}) + H(I, \overline{x}) \cdot (I - \overline{x}) \tag{2.1}$$

where H is L-Lipschitz in its first argument. Of course, the mean value form in Theorem 2.4 is a special case of this generalized center form expression.

One might wonder, we had discussed Taylor interval extensions in the previous chapter. Is it feasible to use higher order Taylor terms in a generalized 'centered form' to achieve not only higher than quadratic, but box forms of any order of convergence (up to differentiability)?

Although they appear to exhibit higher order convergence, Neumaier's paper [Neumaier 2003] provides a counterexample.

Example 2.5. Consider the function $f(x) = -x^2$ over the interval $[0.1h, 1.1h]$ for some $h > 0$. Then for a Taylor centered form of any order ≥ 1, the linear term will bound convergence and lead to a range overestimate of order $O(h^2)$.

2.1 Cornelius & Lohner Form

In [Cornelius and Lohner 1984], H. Cornelius and R. Lohner introduced a framework that would allow for higher than quadratic orders of convergence.

Suppose $f : \mathbb{R} \to \mathbb{R}$ can be represented in the form

$$f(x) = g(x) + r(x),\ g, r \text{ continuous.} \tag{2.2}$$

Also suppose that for any $\mathcal{I} \in \square\mathbb{R}$, there exists a range function $\square R : \square\mathcal{I} \to \square\mathbb{R}$ such that

$$r(I) \subseteq \square R(I),\ \forall I \in \square\mathcal{I}.$$

The function $g(x)$ can be interpreted as an approximation of $f(x)$ such that $g(x)$ is exactly computable by a Turing machine (or any modern computer). The function $r(x)$ is often regarded as the remainder term, and therefore the range function $\square R(I)$ is a range estimation of this remainder term.

Now, with the above assumptions, we can compose a range function for $f(x)$

$$\square f(I) = g(I) + \square R(I) \tag{2.3}$$

We can intuitively see how this range function of $f(\cdot)$ has an accuracy that can be bounded by the remainder term alone.

Theorem 2.6. *Let $f : \mathbb{R} \to \mathbb{R}$ be continuous and let $\square f : \square\mathbb{R} \to \square\mathbb{R}$ be a range function as described in 2.3. Then we have for all $I \in \square\mathbb{R}$,*

(a) $f(I) \subseteq \square f(I)$

(b) $d_H(f(I), \square f(I)) \leq w(\square R(I))$

Proof. (a) is easily shown using our assumptions on $g, r, \square R$.

To prove (b), we notice that since $f(\cdot)$ is continuous on $\mathbb{R}$, it is also continuous on the compact interval I. Hence, $f(\cdot)$ attains a minimum and maximum on this compact interval, denoted $\underline{x}, \overline{x} \in I$. This, of course, implies that $f(I) = [\underline{x}, \overline{x}]$.

Since $g(\cdot)$ is also continuous on I, it must also attain a minimum and maximum $\underline{y}, \overline{y} \in I$. Hence $g(I) = [\underline{x}, \overline{x}]$.

Now suppose $R(I) = [\underline{r}, \overline{r}]$, and we have

$$\begin{aligned} d_H(f(I), \square f(I)) &= d_H([f(\underline{x}), f(\overline{x})], [g(\underline{y}), g(\overline{y})] + [\underline{r}, \overline{r}]) \\ &= \max\{|f(\underline{x}) - g(\underline{y}) - \underline{r}|, |f(\overline{x}) - g(\overline{y}) - \overline{r}|\} \end{aligned}$$

In the first argument of $\max\{\cdot\}$, we have

$$\begin{aligned} |f(\underline{x}) - g(\underline{y}) - \underline{r}| &= f(\underline{x}) - g(\underline{y}) - \underline{r} \\ &\leq f(\underline{y}) - g(\underline{y}) - \underline{r} \\ &\leq (g(\underline{y}) + \overline{r}) - g(\underline{y}) - \underline{r} \\ &= \overline{r} - \underline{r} = w(\square R(I)) \end{aligned}$$

In the second argument, we have

$$\begin{aligned} |f(\overline{x}) - g(\overline{y}) - \overline{r}| &= g(\overline{y}) + \overline{r} - f(\overline{x}) \\ &\leq g(\overline{y}) + \overline{r} - f(\overline{y}) \\ &\leq g(\overline{y}) + \overline{r} - (g(\overline{y}) + \underline{r}) \\ &= \overline{r} - \underline{r} = w(\square R(I)) \end{aligned}$$

□

Since we require the range of $g(\cdot)$ to be exactly computable, we are restricted to functions that have expressions in $\overline{\Omega}$. A practical restriction, however, is that we also want the range of $g(\cdot)$ to be efficiently computable.

Another consideration is that the remainder form $\square R$ must also be efficiently computable, and since we can rewrite 2.3 as

$$r(x) = f(x) - g(x),$$

we can choose $g(x)$ such that $r(x)$ can be written in a 'nice' form relative to its range function. With exactness and efficiency requirements, we have incentive to choose g to be a Hermite interpolant of f (of which both Lagrange interpolation and Taylor expansions fall as a special case of Hermite interpolation). Of course, this would also require $f(x)$ being differentiable enough times such that $g(x)$ can be constructed and $r(x)$ is in a sufficiently 'nice' form.

That is, fixing $I \in \square\mathbb{R}$, suppose that $f(x)$ is k-times continuously differentiable. Then, for any $x_0, \ldots, x_\ell \in I$ distinct and $p_0, \ldots, p_\ell \in \mathbb{Z}^+$ such that $\sum_{i=0}^{\ell} p_i = k$, we define $g(x)$ to be the unique Hermite interpolant of $f(x)$ with degree $k-1$ where

$$g^{(j)}(x_i) = f^{(j)}(x_i),\ j = 0, \ldots, p_i - 1,\ i = 0, \ldots, \ell, \tag{2.4}$$

we can express the remainder function as

$$r(x) = \frac{1}{k!} f^{(k)}(\xi) \prod_{i=0}^{\ell} (x - x_i)^{p_i}, \tag{2.5}$$

for some $\xi \in I$. This form suggests the natural interval extension

$$\square R(I) \coloneqq \frac{1}{k!} f^{(k)}(I) \prod_{i=0}^{\ell} (I - x_i)^{p_i}. \tag{2.6}$$

We see that this remainder form has order k since $f(x)$ is decomposed into the Cornelius-Lohner form and we have

$$\begin{aligned} w(\square R(I)) &\leq 2|\square R(I)| \\ &\leq \left(2\frac{|\square f^{(k)}(I)|}{k!}\right) w(I)^k,\ |I - x_i| \leq w(I). \end{aligned}$$

Furthermore, we can construct a new Cornelius-Lohner form by defining

$$\hat{g}(x) \coloneqq g(x) + \frac{y}{k!} \prod_{i=0}^{\ell} (x - x_i)^{p_i},\ y \in f^{(k)}(I),$$

which induces a remainder form of

$$\square R_{\hat{g}}(I) = \frac{1}{k!}(\square f^{(k)}(I) - y) \prod_{i=0}^{\ell} (I - x_i)^{p_i}. \tag{2.7}$$

If, in addition, $f^{(k)}$ is L-Lipschitz, then we have an extra order of convergence as

$$\begin{aligned} |\square f^{(k)}(I) - y| &\leq w(\square f^{(k)}(I)) \\ &\leq C' w(I). \end{aligned}$$

The difference between $g(x)$ and $\hat{g}(x)$ is that $g^{(k)} \equiv 0$ while $\hat{g}^{(k)} \equiv y$. Although $\hat{g}$ has a higher degree compared to g, which may lead to computational inefficiencies, there is one trick that will be exploited in our discussion of [Hormann et al. 2021] that leads to an overall more efficient algorithm.

2.1.1 Extensions to $\mathbb{R}^n$

In [Cornelius and Lohner 1984], they also discuss extensions to when $f(x)$ is a real function with vector inputs, i.e. $f : \mathbb{R}^n \to \mathbb{R}$. The theorems do generalize to such functions, provided we use the multivariate versions of Taylor's theorem, Lipschitz continuity, and others.

However, taking practical computations into consideration, it is difficult to find a function g that is interpolating and is easy to compute the exact range over. The study of practical Hermite interpolation methods in dimensions ≥ 2 is still an active field of research today. While possible in principle, the practical computation is extremely inefficient.

2.2 K. Hormann, L. Kania, and C. Yap 2021

Whereas [Cornelius and Lohner 1984] focused more on finding appropriate exact functions $g(x)$, [Hormann et al. 2021] focuses more on the remainder term $r(x)$ and its box forms. In theory, the width of the box form $\square f$ is strictly controlled by the box form $\square R$. It also happens that our choice of remainder term relative to its *levels of recursion* in the various forms that we will discuss shortly are of significance for practical computing.

There are two main theoretical ideas that will be applied to the remainder term, along with a number of algorithmic realizations that outperformed state-of-the-art box forms in the context of real root isolation.

1. Expressing the remainder term in 2.5 in centered form, and
2. Approximating the $f^{(k)}(I)$ term in 2.6 by rewriting it in Cornelius-Lohner form and repeating recursively.

2.2.1 Classical Taylor Forms

The classical (and, at the time, state-of-the-art) approach is to set $\ell = 0$ and $p_0 = k = 2$ for the Hermite interpolant 2.4 and set the interpolating data $x_0 := m = m(I)$. If we end up using the ideas presented in [Cornelius and Lohner 1984], we end up with the *minimal Taylor form*

$$\tilde{\square}_2^T f(I) := g(I) + \square R_g(I), \tag{2.8}$$

where

$$g(I) = f(m) + f'(m)(I - m),\ \square R_g(I) = \frac{1}{2}\square f^{(2)}(I)(I - m)^2$$

and $\square f^{(2)}(I)$ is the natural interval extension of $f^{(2)}(x)$.

The above expression converges, under certain assumptions about $f(\cdot)$, with quadratic order. In [Hormann et al. 2021], however, the authors argue that the remainder can be further expanded, given that we have an n-times differentiable function $f(x)$, allowing us to further expand the remainder part as

$$\sum_{i=2}^{n-1} \frac{1}{i!} f^{(i)}(m)(x - m)^i + \frac{1}{n!} f^{(n)}(\xi)(x - m)^n$$

for some $\xi = \xi_x \in I$. For the last term, we can replace $f^{(n)}(\xi)$ with its natural interval extension $|f^{(n)}(I)|$ in order to properly enclose the range as we do not have information on what ξ will be.

This, in turn, gives us the classical *Taylor form*

$$\square_{2,n}^T f(I) := g(I) + \square R_g(I), \tag{2.9}$$

where

$$g(I) = f(m) + f'(m)(I - m),\ \square R_g(I) = \sum_{i=2}^{n-1} \frac{1}{i!} f^{(i)}(m)(I - m)^i + \frac{1}{n!} |\square f^{(n)}(I)|(I - m)^n.$$

One might ask, this Taylor form compared to the previous minimal Taylor form 2.8 both have quadratic convergence, in addition to having more terms in the remainder to compute. Notably, the coefficients in 2.9 requires $O(n \log n)$ arithmetic operations to compute while 2.8 only requires $O(n)$ arithmetic operations, so what do we gain?

It turns out that the higher order terms in the remainder of 2.8 converge with a higher 'order' (which we will revisit in 3 and aptly call it *recursion levels*), and will empirically result in a smaller subdivision tree and a faster runtime in the context of real root isolation.

2.2.2 Taylor Forms with Higher Orders of Convergence

As mentioned in [Cornelius and Lohner 1984], we need not be restrained by a linear Taylor approximation for the exact part $g(x)$. We can use higher order polynomials (whose range can be computed exactly) for the exact part in order to achieve higher-than-quadratic convergence. That is, to achieve an order k convergent Taylor form, we can replace the exact part by an $(k-1)$ Taylor polynomial for f about m:

$$g(x) = \sum_{i=0}^{k-1} \frac{1}{i!} f^{(i)}(m)(x-m)^i.$$

Again, if we simply follow the method outlined in [Cornelius and Lohner 1984], we would choose the remainder function to be

$$\frac{1}{k!} f^{(k)}(\xi)(x-m)^k,$$

which, via a natural interval extension, becomes

$$\frac{1}{k!} |\square f^{(k)}(I)| (I-m)^k.$$

This leads to a minimal higher order Taylor form

$$\tilde{\square}_k^T f(I) \coloneqq g(I) + \square R_g(I),$$

where

$$g(x) = \sum_{i=0}^{k-1} \frac{1}{i!} f^{(i)}(m)(x-m)^i, \ \square R_g(I) = \frac{1}{k!} |\square f^{(k)}(I)| (I-m)^k.$$

Of course, if we use the same methods as detailed in 2.2.1 and assume existence of sufficiently many derivatives of $f(x)$, we can continue the Taylor expansion of the remainder term up to $n-1$. This gives us the *generalized Taylor form of convergence order k and recursion level n*:

$$\square_{k,n}^T f(I) \coloneqq g(I) + \square R_g(I), \tag{2.10}$$

where

$$g(x) = \sum_{i=0}^{k-1} \frac{1}{i!} f^{(i)}(m)(x-m)^i$$

and

$$\square R_g(I) = \sum_{i=k}^{n-1} \frac{1}{i!} f^{(i)}(I-m)^i + \frac{1}{n!} |\square f^{(n)}(I)| (I-m)^n.$$

If $n = k$, we get the minimal recursion level higher order Taylor form, and if $n = \infty$, then we get the *maximal level recursion form.* Achieving maximal level requires more strict constraints on $f(x)$, i.e. it must be analytic and $r(I)$ must be sufficiently small to allow the power series to converge. [Hormann et al. 2021] only deals with the minimal and maximal level cases. In our novel approach in 3, however, we will also consider non-maximal levels of recursion in empirical computation.

Additionally, in practice, we would want our exact part to be of degree ≤ 4 due to both the unsolvability of quintic and higher order polynomials over the radicals as well as the computational drawbacks of precisely computing the range of $g(I)$. Indeed, while $k = 3$ ($g(x)$ is quadratic) only needs marginally more computational power to compute the exact range, the case when $k = 4$ ($g(x)$ is cubic) requires significantly more computational power which results in a trade-off between a theoretically higher order of convergence versus an empirical lower time of computation.

To compute the exact range of a quadratic polynomial over an interval $I = [a, b]$, we only need to consider three points: the endpoints of the interval a, b, and the critical point. If the critical point does not lie in the interval, then we only need to compute the endpoints. Otherwise, we compute the range by taking the extremum (or minimum) into account.

However, to compute the exact range of a cubic polynomial over the same interval, we would require a significantly more sophisticated approach that would also take much longer to run. Refer to Algorithm 1.

2.2.3 Recursive Interpolation Forms

Another way to formulate efficient range functions is to recursively use the Cornelius and Lohner form. Suppose that $h_0(x)$ is a Hermite interpolant of $f(x)$ with interpolation nodes $\{x_i\}_{0\leq i\leq \ell}$ and multiplicities $\{p_i\}_{0\leq i\leq \ell}$ such that the degree of $h_0(x)$ is $k-1$ (i.e. $\sum_{i=0}^{\ell} p_i = k$), matching the form in 2.4. We can then write the remainder in the form

$$R_{h_0}(x) = \frac{1}{k!}\omega(x) f^{(k)}(\xi),\ \omega(x) := \prod_{i=0}^{\ell} (x - x_i)^{p_i}$$

for some $\xi = \xi_x \in I$. We also have that

$$|R_{h_0}(I)| \leq \frac{1}{k!}\Omega |f^{(k)}(I)|.$$

Now, we can apply the same idea again, but to the remainder $f^{(k)}(\cdot)$. We can define another Hermite interpolant $h_1(x)$ for $f^{(k)}(x)$ with the same interpolating nodes and multiplicities to attain a remainder R_{h_1} that is one recursive lever deeper. We use the same bound above ($|R_{h_1}(I)| \leq \Omega |f^{2k}(I)|$) to get

$$|f^{(k)}(I)| \leq |h_1(I)| + \Omega |f^{(2k)}(I)|.$$

Algorithm 1 Computing $g(I) = [A, B]$ for the cubic polynomial $g(x) = \sum_{i=0}^{3} c_i(x-m)^i$.

$A := \min\{g(a), g(b)\}$
$B := \max\{g(a), g(b)\}$
$\Delta := c_2^2 - 3c_1c_3$
if $\Delta > 0$ **then**
    $L := \operatorname{sgn}(c_3)(c_1 + 3c_3r^2)$
    $R := 2|c_2|r$
    if $L > R$ **then**
        if $|c_2| < 3|c_3|r$ **then**
            $x^- := m - (c_2 - \sqrt{\Delta})/(3c_3)$
            $x^+ := m - (c_2 + \sqrt{\Delta})/(3c_3)$
            $A := \min\{A, g(x^-)\}$
            $B := \max\{B, g(x^+)\}$
    else if $L > -R$ **then**
        if $c_2 > 0$ **then**
            $x^- := m - (c_2 - \sqrt{\Delta})/(3c_3)$
            $A := g(x^-)$
        else if $c_2 < 0$ **then**
            $x^+ := m - (c_2 + \sqrt{\Delta})/(3c_3)$
            $B := g(x^+)$
return $[A, B]$

We can repeat this procedure for up to any n (assuming that $f(x)$ is nk-times differentiable) to obtain Hermite interpolants $h_i(x)$, $1 \le i \le n$. This gives us the bound

$$|R_{h_0}(I)| \le \sum_{j=1}^{n-1} |h_j(I)|\Omega^j + \Omega^n|\square f^{(nk)}(I)|.$$

Finally, combining this remainder bound with our exact Hermite interpolant $h_0(x)$ for $f(x)$, we get the *recursive remainder form of order k and recursion level n*:

$$\square_{k,n}^{R} f(I) := h_0(I) + (\sum_{j=1}^{n-1} |h_j(I)|\Omega^j + \Omega^n|\square f^{(nk)}(I)|). \tag{2.11}$$

With the same reasoning as in 2.9. Although the number of terms we must calculate for the remainder term increases, the higher order terms in the remainder gives us faster convergence in practice. In the same light, we can also define a minimal and maximal variant, equivalent to setting $n = 1$ and $n = \infty$, respectively.

2.2.4 Recursive Lagrange Forms with Cubic Convergence

An instantiation of 2.2.3 is to use Lagrange interpolant. We can set $p_i = 1\ \forall i$, choose the interpolation nodes to be $\{a, m, b\}$, where $I = [a, b]$ and $m = m(I)$, and let $\ell = 2$. This gives us a quadratic

Lagrange interpolant for $f(x)$ with cubic ($k = 3$) convergence.

At each recursion level $j = 0, 1, \ldots$, we have the following centered form Lagrange interpolant

$$h_j(x) \coloneqq d_{j,0} + d_{j,1}(x - m) + d_{j,2}(x - m)^2,$$

where,

$$\begin{aligned} d_{j,0} &= f^{(3j)}(m) \\ d_{j,1} &= \frac{f^{(3j)}(b) - f^{(3j)}(a)}{2r} \\ d_{j,2} &= \frac{f^{(3j)}(b) - 2f^{(3j)}(m) + f^{(3j)}(a)}{2r^2}, \; r \coloneqq r(I). \end{aligned}$$

Following, we can also compute the exact range of $\omega(x) = (x - a)(x - m)(x - b)$ over the interval I to be

$$\omega(I) = \frac{2\sqrt{3}}{9} r^3 [-1, 1] \implies \Omega = \frac{\sqrt{3}}{27} r^3.$$

With this instantiation, we get the *recursive cubic Lagrange form with recursion level n*:

$$\square^L_{3,n} f(I) \coloneqq h_0(I) + [-1, 1] T_{3,n}, \tag{2.12}$$

where

$$T_{3,n} \coloneqq \sum_{j=1}^{n-1} |h_j(I)| \Omega^j + \Omega^n |\square f^{(3n)(I)}| \in \mathcal{O}(r^3).$$

2.2.5 A Cheaper Variant of the Recursive Lagrange Form

In the previous section, our cubic Lagrange form achieves its cubic convergence solely through the exact term $h_0(\cdot)$, hence, we are motivated to balance the remainder term. The major trade-off for the remainder term $T_{3,n}$ is *speed of computation* versus *tightness of box form.*

In this section, we argue that we can forgo some tightness in the recursive Lagrange form in order to speed up computations. We observe that the terms in the remainder $\{|h_j(I)|\}_{j \geq 1}$ are quadratic interpolating polynomials that are computed exactly. Without disturbing the order of convergence, we can replace the exact evaluation of these terms with a centered form evaluation that is cheaper:

$$|h_j(I)| \leftarrow (d_{j,0} + r|d_{j,1}| + r^2|d_{j,2}|).$$

This gives us the *cheap recursive cubic Lagrange form with recursion level n*:

$$\square^{L'}_{3,n} f(I) \coloneqq h_0(I) + [-1, 1] T'_{3,n}, \tag{2.13}$$

where

$$T'_{3,n} \coloneqq \sum_{j=1}^{n-1} (d_{j,0} + r|d_{j,1}| + r^2|d_{j,2}|)\Omega^j + \Omega^n |\square f^{(3n)}(I)| \in \mathcal{O}(r^3)$$

The efficiency of this cheaper variant will be shown in empirical testing using real-root isolation as the sample problem.

2.2.6 Recursive Lagrange Forms with Quartic Convergence

Let us consider applying the trick introduced by Cornelius and Lohner in (2.7) to the generalized Taylor and Hermite forms above to achieve a 'free' higher order of convergence.

If we wish to apply this trick to the generalized Taylor forms in (2.9), we would modify the exact part of $\square_{k,n}^{T} f(I)$ so that

$$g(x) \leftarrow g(x) + \frac{f^{(k)}(m)}{k!}(x-m)^k.$$

However, we notice that this is simply the exact part for the generalized Taylor form of one higher order: $\square_{k+1,n}^{T} f(I)$.

While the trick does not work for (2.9), we can effectively apply this to our recursive Lagrange forms in 2.2.4 and 2.2.5 to achieve quartic convergence. Consider

$$\hat{h}_0(x) \coloneqq h_0(x) + \frac{f^{(3)}(m)}{6}\omega(x).$$

We can view this alternate form as a combination of a Lagrange interpolating polynomial that also satisfies $\hat{h}_0^{(3)}(m) = f^{(3)}(m)$. In its center form, the coefficients of the even-powered terms do not change, while the odd-powered terms are modified, i.e.

$$\hat{h}_0(x) \coloneqq d_{0,0} + \hat{d}_{0,1}(x-m) + d_{0,2}(x-m)^2 + \hat{d}_{0,3}(x-m)^3,$$

where

$$\hat{d}_{0,1} \coloneqq d_{0,1} - \frac{f^{(3)}(m)}{6}r^2,\ \hat{d}_{0,3} \coloneqq \frac{f^{(3)}(m)}{6}.$$

Following the idea of the trick, we can now define a new $h_1(x)$

$$\hat{h}_1(x) \coloneqq h_1(x) - f^{(3)}(m) = (d_{1,1} + d_{1,2}(x-m))(x-m).$$

We can see the remainder terms are still bounded:

$$\begin{aligned} |R_{\hat{h}_0}(I)| &\leq \Omega |f^{(3)}(I) - f^{(3)}(m)| \\ |R_{\hat{h}_1}(I)| &\leq \Omega |f^{(6)}(I)| \\ \implies |f^{(3)}(I) - f^{(3)}(m)| &\leq |\hat{h}_1(I)| + \Omega |f^{(6)}(I)|. \end{aligned}$$

If we continue recursively splitting the remainder terms into the Cornelius and Lohner form, we arrive at the *recursive quartic Lagrange form with recursion level n*:

$$\square_{4,n}^{L} f(I) \coloneqq \hat{h}_0(I) + [-1,1]T_{4,n}, \tag{2.14}$$

where

$$T_{4,n} \coloneqq |\hat{h}_1(I)|\Omega + \sum_{j=2}^{n-1} |h_j(I)|\Omega^j + \Omega^n |\square f^{(3n)}(I)| \in O(r^4).$$

Of course, we can also use the cheaper evaluations by replacing the exact evaluations of $|\hat{h}_1(I)|$ and $\{|h_j(I)|\}_{j\geq 2}$ with their centered form evaluations. This yields a faster but less tight box form:

$$\square_{4,n}^{L'} f(I) \coloneqq \hat{h}_0(I) + [-1,1]T'_{4,n}, \tag{2.15}$$

where

$$T'_{4,n} \coloneqq T'_{3,n} - |d_{1,0}|\Omega \in O(r^4).$$

Remark. *In many numerical computing applications, we encounter continuous real-valued functions that we can only query (i.e. we do not have access to the information on its expression, and only know values of $f(\cdot)$ at specific input values by querying). We can query the function that different points and use an infinite basis (e.g. polynomial basis $\{x^n\}_{n\geq 0}$, Fourier basis $\{e^{inx}\}_{n\in\mathbb{Z}}$) which can be chosen such that it is dense in the continuous real-valued functions using the Stone-Weierstrass theorem [Rudin 1976].*

This motivates us to begin considering f as a polynomial that was constructed as an approximation to some real-valued function belonging to the class of continuous (or continuous almost-everywhere) functions.

If f has degree d, then we know the maximal recursive form for, say, the cubic Lagrange polynomials will only depend on the terms $\{f^{(3j)}(a), f^{(3j)}(b), f^{(3j)}(b)\}_{0\leq j\leq \lfloor d/3\rfloor}$. The polynomials becomes identically zero after $d+1$ derivatives and so for all recursion levels $j \geq \lfloor d/3\rfloor$, the method becomes identical to the maximal recursive form.

We are now ready to move onto the novel methods that we have developed over the past year as a follow-up to [Hormann et al. 2021].

3 | Results

To motivate our generalizations and results, we make the argument that the Cornelius and Lohner form in (2.2) cannot be realized in practical settings. That is, the so-called 'exact range' of $g(I)$ cannot be precisely computed for any standard model of arithmetic in modern computer architectures, whether it be floating-point models, rational arithmetic models, or other families of models.

Regardless of the model of arithmetic, elements of $\mathbb{R}$ are represented by elements from the dyadic number system $\mathbb{Z}[\frac{1}{2}] = \{m2^n \mid m, n \in \mathbb{Z}\}$, which is a dense subset of $\mathbb{R}$ (this can easily be showed first by showing density of $\mathbb{Z}[\frac{1}{2}]$ in $\mathbb{Q}$, then showing density of $\mathbb{Q}$ in $\mathbb{R}$ and combining the results in an epsilon argument). Suppose we wish to represent the number $\sqrt{3}$ exactly in our computer, this is already impossible to do since $\sqrt{3} \notin \mathbb{Z}[\frac{1}{2}]$.

This problem can be circumvented using computer algebra systems and software such as Maple or Mathematica, giving us exact access to all algebraic numbers (which is still a strict subset of $\mathbb{R}$). However, the performance issues of, say, representing a real number that has a high degree (e.g. > 100) minimal polynomial over $\mathbb{Z}[\frac{1}{2}]$ makes these symbolic algebraic systems not applicable in practical computational scenarios.

3.1 Overview

As a result, we introduce a generalization of the Cornelius and Lohner form using new ideas for box forms that can provide a priori precision guarantees. We also introduce faster recursive range functions that exceed the performance of previous box forms discussed in 2.2 in the application of real-root isolation. We also develop an 'amortized' method of holistically evaluating the complexity of range functions and apply it to a general class of range functions. Finally, we explore the performance capabilities of non-maximal variants of the best performing range functions seen in 2.2.

3.2 Generalizing the Cornelius & Lohner Framework

Motivated by our discussion above, we replace the exact part $g(I)$ in the traditional Cornelius-Lohner form by a range function for g:

$$\square f(I) = \square g(I) + \square R_g(I) \tag{3.1}$$

Now, we generalize the bound of Theorem 2.6 so that we can bound the accuracy of the range function $\square f(\cdot)$ by its remainder alone.

Theorem 3.1. *Given $\square f(\cdot)$ as defined in (3.1), we have*

$$d_H(f(I), \square f(I)) \leq d_H(g(I), \square g(I)) + w(\square R_g(I))$$

Proof. First denote the endpoints of the intervals as follows:

$$f(I) = [f(\underline{x}), f(\overline{x})],\ g(I) = [g(\underline{y}), g(\overline{y})],\ \square R_g(I) = [a, b]$$

Which is allowed by our assumption that f, g are continuous on the compact interval I, giving rise to minima and maxima $\overline{x}, \underline{x}, \overline{y}, \underline{y}$ to f and g, respectively.

By the fact that $\square g$ is a range function for g, there must exist some interval $\varepsilon = [\underline{\varepsilon}, \overline{\varepsilon}]$, where $\underline{\varepsilon} \leq 0$ and $\overline{\varepsilon} \geq 0$, such that $\square g(I) = g(I) + \varepsilon$.

Therefore we have

$$\begin{aligned} d_H(g(I), \square g(I)) &= \max\{-\underline{\varepsilon}, \overline{\varepsilon}\} \\ f(I) \subseteq \square f(I) &= \square g(I) + \square R_g(I) \\ \implies d_H(f(I), \square f(I)) &= \max\{f(\underline{x}) - (g(\underline{y}) + \underline{\varepsilon} + a), (g(\overline{y}) + \overline{\varepsilon} + b) - f(\overline{x})\} \end{aligned}$$

For the first term $f(\underline{x}) - (g(\underline{y}) + \underline{\varepsilon} + a)$, we have

$$\begin{aligned} f(\underline{x}) - (g(\underline{y}) + \underline{\varepsilon} + a) &\leq f(\underline{y}) - (g(\underline{y}) + \underline{\varepsilon} + a) \\ &= (g(\underline{y}) + R_g(\underline{y})) - (g(\underline{y}) + \underline{\varepsilon} + a) \\ &= R_g(\underline{y}) - \underline{\varepsilon} - a \\ &\leq -\underline{\varepsilon} + (b - a). \end{aligned}$$

For the second term $(g(\overline{y}) + \overline{\varepsilon} + b) - f(\overline{x})$, we have

$$\begin{aligned} (g(\overline{y}) + \overline{\varepsilon} + b) - f(\overline{x}) &\leq (g(\overline{y}) + \overline{\varepsilon} + b) - f(\overline{y}) \\ &= (g(\overline{y}) + \overline{\varepsilon} + b) - (g(\overline{y}) + R_g(\overline{y})) \\ &= \overline{\varepsilon} + b - R_g(\overline{y}) \\ &\leq \overline{\varepsilon} + (b - a). \end{aligned}$$

Hence

$$\begin{aligned} d_H(f(I), \square f(I)) &= \max\{f(\underline{x}) - (g(\underline{y}) + \underline{\varepsilon} + a), (g(\overline{y}) + \overline{\varepsilon} + b) - f(\overline{x})\} \\ &\leq \max\{-\underline{\varepsilon} + (b - a), \overline{\varepsilon} + (b - a)\} \\ &= \max\{-\underline{\varepsilon}, \overline{\varepsilon}\} + w(\square R_g(I)) \\ &= d_H(g(I), \square g(I)) + w(\square R_g(I)). \end{aligned}$$

□

However, in order to apply Theorem 3.1, we must have a stronger notion of range functions that encapsulate this idea of specifying the error ($\varepsilon = [\underline{\varepsilon}, \overline{\varepsilon}]$ in the proof) a priori for us to instantiate reasonable range functions based on this theorem.

To do so, we introduce the notion of *precision-bounded range functions* or a *strong box form* for a function $g(x)$, denoted $\square g(I;\varepsilon)$, $\varepsilon > 0$. In addition to the usual properties that comes with being a range function, we also require, for all $\varepsilon > 0$, that

$$d_H(g(I), \square g(I;\varepsilon)) \leq \varepsilon.$$

Now, using precision-bounded range functions, we are able to use our generalized Cornelius and Lohner framework to construct useful box forms. We can define the framework for a box form (that is precision-bounded) for $f(x)$:

$$\square_{\text{pb}} f(I) \coloneqq \square g(I;\varepsilon) + \square R_g(I)$$

Of course, it is then obvious to see that box functions of the form $\square_{\text{pb}} f$ converge on the same order as $\square R_g$ by application of Theorem 3.1. Finally, if we set $g(x)$ to be some Hermite interpolant with degree $k-1$, then $\square R_g(I)$ will have convergence order k. In conjunction with a precision-bounded box form for $g(x)$, we can achieve box forms of arbitrary order of convergence.

3.3 Precision-Bounded Box Forms

In this section, we will explore the details in constructing these strong box forms.

3.3.1 Lipschitz Expressions

For $f : \mathbb{R}^n \to \mathbb{R}$, we can express $f(\cdot)$ for all elements in its proper domain (i.e. a subset of $\mathbb{R}^n$ such that the range of this subset under f is properly contained in $\mathbb{R}$) using some expression E_f. As discussed in Chapter 1, we can also input interval vectors into the expression E_f.

The expression E_f is *Lipschitz* in $\mathcal{B} \in \square\mathbb{R}^n$ if we have the following inductive property:

(a) The root of E_f is labeled by a variable or constant.

(b) Writing E_f as a function of sub-expressions, i.e. $E_f = \varphi(E_1, \ldots, E_m)$.

  (i) E_i is Lipschitz in $\mathcal{B}$ for all $1 \leq i \leq m$.

  (ii) $\varphi(E_1(\mathcal{B}), \ldots, E_m(\mathcal{B}))$ is defined, and there exists some $\varepsilon > 0$ such that φ is L-Lipschitz in $B_\varepsilon(E_1(\mathcal{B}), \ldots, E_m(\mathcal{B}))$, where $B_\varepsilon(\mathbf{x}) \coloneqq \{\mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{y}\|_{L^2} < \varepsilon\}$ is the open ball of radius ε around $(E_1(\mathcal{B}), \ldots, E_m(\mathcal{B}))$ (i.e. φ is L-Lipschitz in some open neighbourhood around $\mathcal{B}$).

If this is satisfied, then we can use the following result from [Neumaier 1990]:

Theorem 3.2. *If E is a Lipschitz expression over $\mathcal{B} \in \square\mathbb{R}^n$, then there exists a vector of $\ell = (\ell_1, \ldots, \ell_n) \in \mathbb{R}^n$ such that for all $B, B' \subseteq \mathcal{B}$,*

$$d_H(E(B), E(B')) \leq \langle \ell, d_H(B, B') \rangle,\ d_H(B, B') \coloneqq (d_H(I_1, I_1'), \ldots, d_H(I_n, I_n')),$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product over $\square\mathbb{R}^n$.

We can extend this result to box forms by replacing $E(B')$ with $\square E(B)$ to show that $\square E(B)$ encloses $E(B)$.

Now, we wish to develop an abstract representation, or model, of computing. We first assume that $f(\mathcal{B})$ and $\partial_i f(\mathcal{B})$ are computable in our model, which leads us to computing $\square f(\mathcal{B})$ and $\square \partial_i f(\mathcal{B})$, which is finally realized in Turing computability using the density of $\mathbb{Z}[\frac{1}{2}]$ in $\mathbb{R}$.

Let Ω be an expression class. We say that Ω is a *Lipschitz*$^+$ class if all Ω-expressions that induces a function f that is Lipschitz$^+$ (i.e. f has continuous almost-everywhere partial derivatives and both f and $\nabla f = (\partial_1 f, \ldots, \partial_n f)$ are locally L-Lipschitz).

Suppose we are given a Ω-expression $E(\cdot)$ for the function f, where Ω is Lipschitz$^+$, then we can define the gradient of the expressions $\nabla E = (\partial_1 E, \ldots, \partial_m E)$. Each $\partial_i E$ can be defined recursively as follows:

$$\partial_i E \coloneqq \begin{cases} 0 & \text{if } E \text{ a constant,} \\ \delta_{ij} & \text{if } E \text{ a placeholder variable,} \\ \sum_{j=1}^{m} (\partial_i f)(E_1, \ldots, E_m) \cdot \partial_i E_j & \text{if } E = f(E_1, \ldots, E_m), \end{cases}$$

where

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{otherwise,} \end{cases}$$

is Kronecker's delta function.

Now we can begin characterizing the function class for which there exists strong box forms for. First, given a function $f : \mathbb{R}^n \to \mathbb{R}$, a *strong approximation function* is a function $\widetilde{f} : \mathbb{R}^n \times \mathbb{R}_{>0} \to \mathbb{R}$ such that for all tuples $(x, \varepsilon) \in \mathbb{R}^n \times \mathbb{R}_{>0}$, we have

$$|\widetilde{f}(x; \varepsilon) - f(x)| \leq \varepsilon.$$

An additional concept that we will see being used for the proof of Theorem 3.5 below is notion of *ε-fineness*.

Definition 3.3. Given an interval box $\mathcal{B} \in \square\mathbb{R}^n$, a *subdivision* of $\mathcal{B}$ is a finite set of disjoint subsets $\mathcal{D} = \{B_1, \ldots, B_\ell\}$ such that

$$\bigcup_{i=1}^{\ell} B_i = \mathcal{B}.$$

Definition 3.4. A subdivision $\mathcal{D}$ of an interval box $\mathcal{B} \in \square\mathbb{R}^n$ is ε-fine with respect to a function $f : \mathbb{R}^n \to \mathbb{R}$ if

$$\Delta(f, B) \coloneqq \frac{1}{2} \sum_{i=1}^{n} w_i(B) \cdot |\partial_i f(B)| \leq \frac{\varepsilon}{4},\ \forall B \in \mathcal{D}.$$

Algorithm 2 Fine Subdivision Algorithm

Input: $(f, \mathcal{B}, \varepsilon)$
Output: An ε-fine subdivision $\mathcal{D}$ of $\mathcal{B}$.

Let $Q, \mathcal{D}$ be queues of boxes, initialized as $\mathcal{D} \leftarrow \emptyset$ and $Q \leftarrow \{\mathcal{B}\}$.
while $Q \neq \emptyset$ **do**
  $B \leftarrow Q$.pop()
  $(J_1 \dots J_n) \leftarrow \nabla f(B)$
  $\Delta(f, B) \leftarrow \sum_{i=1}^{n} |J_i| \cdot w_i(B)$
  if $\Delta(f, B) \leq \varepsilon/4$ **then**
    $\mathcal{D}$.push(B)
  else
    $i^* \leftarrow \arg\max_{i=1\dots n} |J_i| \cdot w_i(B)$
    Q.push(bisect(B, i^*)) ▷ Bisect dimension i^*
Output $\mathcal{D}$

For any f, $\mathcal{B}$, and ε, Algorithm 2 allows us to find an ε-fine subdivision of $\mathcal{B}$ with respect to f.

Now we can state a lemma that provides sufficient conditions for there to exist a strong box function.

Lemma 3.5. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be a Lipschitz$^+$ function in $\mathcal{B} \in \square\mathbb{R}^n$ and has a strong approximation function $\widetilde{f}$, then there exists a strong box function for f.*

Proof. Let us first compute an ε-fine subdivision $\mathcal{D}$ for $\mathcal{B}$ using Algorithm 2. Now for each $B \in \mathcal{D}$, by the mean value theorem, we have for all $b \in B$

$$|f(b) - f(m(B))| \leq \Delta(f, B).$$

Then we can define, for each $B \in \mathcal{D}$, the interval

$$J(B) \coloneqq [\widetilde{f}(m(B); \frac{\varepsilon}{4}) - \frac{\varepsilon}{2}, \widetilde{f}(m(B); \frac{\varepsilon}{4}) + \frac{\varepsilon}{2}].$$

Of course, by an epsilon argument, we have for all $B \in \mathcal{D}$

- $f(B) \subseteq J(B)$,
- $d_H(J, f(B)) < \varepsilon$.

Let $J(\mathcal{D}) \coloneqq \bigcup_{B \in \mathcal{D}} J(B)$, then by the same argument, we have

- $f(\mathcal{B}) \subseteq J(\mathcal{D})$,
- $d_H(f(\mathcal{B}), J(\mathcal{D})) < \varepsilon$.

□

This strong box function is also computable in our abstract model if we also have a Lipschitz$^+$ expression for it over some interval. This is our main theorem as it gives the necessary conditions for functions that allow us to abstractly compute range functions using Theorem 3.1.

Theorem 3.6. *Let Ω be a Lipschitz$^+$ class where each $f \in \Omega$ has a strong approximation function. If $E(\cdot)$ is a Lipschitz$^+$ expression for f over $\mathcal{B} \in \square\mathbb{R}^n$, then the strong box function characterized by $E(\mathcal{B}; \varepsilon)$ is abstractly computable.*

Proof. Using induction, we see the base case is trivial. Otherwise, the expression is re-writable as $E = f(E_1, \ldots, E_m)$, then we know $\widetilde{I_i} := E_i(B; \varepsilon_i)$ is abstractly computable for $i = 1, \ldots, m$. Setting $\widetilde{\mathcal{B}} := (\widetilde{I_1}, \ldots, \widetilde{I_m})$, we can use Lemma 3.5 and its proof to compute $f(\widetilde{\mathcal{B}}; \varepsilon)$. □

Remark. *One of the most extensive classes of functions that satisfy the requirements above is the class of Hypergeometric functions. i.e.*

$$ {}_2F_1(a, b, c; z) := \sum_{n=0}^{\infty} \frac{(a)_n (b)_n}{(c)_n} \frac{z^n}{n!}, $$

where

$$ (x)_n = \begin{cases} 1 & \text{if } n{=}0, \\ x \cdot (x+1) \cdot \ldots \cdot (x+n-1) & \text{if } n{>}0, \end{cases} $$

is the Pochhammer symbol. This is primarily due to the fact that Hypergeometric functions are closed under differentiation with respect to z, i.e. the derivative of a Hypergeometric function is another Hypergeometric function with different parameters.

For further discussion on the exact computation of Hypergeometric functions, please see [Du et al. 2002] and [Du and Yap 2006].

3.4 Quartic Range Functions using Hermite Interpolation

We now shift our focus to practical range functions that can be used to compute ranges of real-valued functions, and present a new range function based on Hermite interpolation that can achieve better performance (in the context of real-root isolation) than those in [Hormann et al. 2021].

Let h_0 be a cubic Hermite interpolant for f such that

$$ h_0(a) = f(a), \; h_0'(a) = f'(a), \; h_0(b) = f(b), \; h_0'(b) = f'(b), $$

where $[a, b] = I$. We can write out this Hermite interpolant explicitly as

$$ h_0(x) = c_{0,0} + c_{0,1}(x - m) + c_{0,2}(x - m)^2 + c_{0,3}(x - m)^3, \; m = m(I), $$

where

$$
\begin{aligned}
c_{0,0} &= \frac{f(b)+f(a)}{2} - \frac{f'(b)-f'(a)}{4}r,\\
c_{0,1} &= 3\frac{f(b)-f(a)}{4r} - \frac{f'(b)+f'(a)}{4},\\
c_{0,2} &= \frac{f'(b)-f'(a)}{4r},\\
c_{0,3} &= \frac{f'(b)+f'(a)}{4r^2} - \frac{f(b)-f(a)}{4r^3},\ r = r(I).
\end{aligned}
$$

Now, the remainder can be written as

$$
R_{h_0}(x) = \frac{\omega(x)}{4!}f^{(4)}(\xi),\ \omega(x) = (x-a)^2(x-b)^2,
$$

for some $\xi = \xi_x \in I$. This provides us an upper bound for $R_{h_0}(I)$:

$$
|R_{h_0}(I)| \leq \Omega |f^{(4)}(I)|,\ \Omega = \frac{|\omega(I)|}{4!} = \frac{r^4}{24}.
$$

Of course, we apply the same technique as in Section 2.2.4 in order to further bound $|f^{(4)}(I)|$. This leads to the set of cubic Hermite interpolants $\{h_j\}_{1\leq j\leq \ell}$, where ℓ is the recursion level:

$$
h_0(x) = c_{j,0} + c_{j,1}(x-m) + c_{j,2}(x-m)^2 + c_{j,3}(x-m)^3,
$$

with

$$
\begin{aligned}
c_{j,0} &= \frac{f^{(4j)}(b)+f^{(4j)}(a)}{2} - \frac{f^{(4j+1)}(b)-f^{(4j+1)}(a)}{4}r,\\
c_{j,1} &= 3\frac{f^{(4j)}(b)-f^{(4j)}(a)}{4r} - \frac{f^{(4j+1)}(b)+f^{(4j+1)}(a)}{4},\\
c_{j,2} &= \frac{f^{(4j+1)}(b)-f^{(4j+1)}(a)}{4r},\\
c_{j,3} &= \frac{f^{(4j+1)}(b)+f^{(4j+1)}(a)}{4r^2} - \frac{f^{(4j)}(b)-f^{(4j)}(a)}{4r^3}.
\end{aligned}
$$

If we set $R_{h_j} \coloneqq f^{(4j)} - h_j$, then we have

$$
\begin{aligned}
|f^{(4j)}(I)| &\leq |h_j(I)| + |R_{h_j}(I)|\\
&\leq |h_j(I)| + \Omega |f^{(4j+4)}(I)|\\
\implies |f^{(4)}(I)| &\leq |h_1(I)| + \Omega(|h_2(I)| + \Omega |f^{(12)}(I)|)\\
&\leq \dots\\
&\leq \sum_{j=1}^{\ell} |h_j(I)|\Omega^{j-1} + \Omega^{\ell}|\square f^{(4\ell+4)}(I)|,
\end{aligned}
$$

This gives us the recursive remainder bound

$$|R_{h_0}(I)| \le S_\ell,\ S_\ell := \sum_{j=1}^{\ell} |h_j(I)|\Omega^j + \Omega^{\ell+1}|\square f^{(4\ell+4)}(I)|.$$

This gives us the *recursive Hermite form of order* 4 *and recursion level* $\ell \ge 0$:

$$\square^H_{4,\ell} f(I) = h_0(I) + [-1,1]S_\ell, \tag{3.2}$$

which depends on the values

$$f^{(4j)}(a),\ f^{(4j+1)}(a),\ f^{(4j)}(b),\ f^{(4j+1)}(b),\ 0 \le j \le \ell.$$

Again, we can define the maximal recursion level Hermite form $\square^H_4 f(I) := \square^H_{4,\infty} f(I)$ if f is analytic and r sufficiently small. Further, if f is a polynomial of degree $d-1$, then

$$\square^H_4 f(I) = \square^H_{4,\ell} f(I),\ \ell \ge \lceil \frac{d}{4} \rceil - 1.$$

Of course, we can then introduce a cheap variant $\square^{H'}_4 f$ that forgoes some tightness of the range function for computing performance. This is the same argument as in previous sections. We end up having

$$\square^{H'}_{4,\ell} f(I) := h_0(I) + [-1,1]S'_\ell, \tag{3.3}$$

where

$$S'_\ell = \sum_{j=1}^{\ell} \big(|c_{j,0}| + r|c_{j,1}| + r^2|c_{j,2}| + r^3|c_{j,3}|\big)\Omega^j + \Omega^{\ell+1}|\square f^{(4\ell+4)}(I)|.$$

3.5 Complexity Analysis of Practical Range Functions

We provide a template for the amortized holistic complexity analysis for range functions $\square f(\mathcal{I})$, where f is a high degree polynomial, which is analysing its cost over a subdivision tree, not just on a single interval. The motivation for this is due to the fact that if we utilize the range functions in an algorithm that uses subdivision (like in the case of real-root isolation), then we may have already computed some of the data required for the construction of the range function in the subsequent subdivision intervals.

We begin by analysing the complexity of $\square^{L'}_3 f$ from (2.13) in [Hormann et al. 2021] which had previously performed the best in our experiments, then move on to our newly defined $\square^H_4 f$. Finally, with the motivation of the above two sections, we generalize our analysis to all Hermite interpolating schemes with equally spaced nodes with equal multiplicities.

Remark. *Note that the analysis of the cheap variant also applied for the non-cheap variant (and vice-versa) since the two variants do not change how the interval is subdivided and at which points the values need to be computed.*

Suppose we are given a polynomial function f whose degree $d \geq 2$ and an interval $\mathcal{I}$. We wish to construct a range function $\square f(\mathcal{I})$ based on the information of the values of $f, f', \ldots, f^{(d)}$ in $\mathcal{I}$.

Definition 3.7. A *subdivision tree* is a finite tree whose nodes are intervals such that

- The root of the tree is $\mathcal{I}$,
- For any non-leaf node representing the interval $[a, b]$, its left and right children will represent the intervals $[a, m]$ and $[m, b]$ respectively.

Now we are ready to begin providing theoretical accounts for the empirical speedups that we will observe in Chapter 4.

3.5.1 Amortized Complexity of $\square_3^{L'} f$

Further assume that $3 \mid d$. In the case $\square f(\mathcal{I}) = \square_3^{L'} f(\mathcal{I})$, we have

$$f^{(3j)}(a),\ f^{(3j)}(m),\ f^{(3j)}(b),\ 0 \geq j \geq d/3 - 1.$$

Then the cost for computing $\square_3^{L'} f(I)$ is d. Although we can show the cost to compute the maximal Taylor form of order 2, $\square_2^T f(I)$, is also d, when amortizing the complexity analysis over the entire subdivision tree, we will distinguish a clear advantage as our cheap cubic Lagrange form can re-use the information of the derivatives in its sub-intervals.

We wish to bound the *cost* $C_3^L(T)$, defined as the total number of values needed to compute $\square_3^{L'}$ for all $I \in T$. We can define the cost with a different parameters when we know that the tree T has n leaves.

Now we have the following recurrence relation:

$$C_3^L(n) = \begin{cases} d & \text{if } n = 1, \\ C_3^L(n_L) + C_3^L(n_R) - \frac{d}{3} & \text{if } n \geq 2, \end{cases}$$

where n_L, n_R are the number of leaves that the left and right subtrees have, respectively, and $n_L + n_R = n$. If we use induction on this recurrence relation, then we find that the amortized complexity is

$$C_L^3(n) = (2n + 1) \cdot \frac{d}{3}.$$

If we have a full binary tree, it would have $2n - 1$ nodes, hence the average cost per node will be

$$\frac{2n + 1}{2n - 1} \cdot \frac{d}{3} \sim \frac{d}{3},$$

this tells us that the asymptotic cost per node is $d/3$.

3.5.2 Amortized Complexity of $\square_{4,\ell}^{H} f$

Let ℓ be given. For simplicity and without loss of generality, assume that $d = 4(\ell+1)$. For $\square_4^H f(I)$ to be computed, we would require $4(\ell+1) = d$ evaluations. We then have the recurrence relation

$$C_4^H(n) = \begin{cases} d & \text{if } n = 1, \\ C_4^H(n_L) + C_4^H(n_R) - d/2 & \text{if } n \geq 2, \end{cases}$$

This is similar to the analysis of $\square_3^L f$, except that the midpoint of the interval is never evaluated and thus will never be shared to its children.

Of course, we can easily solve this recurrence and get that

$$C_4^H(n) = (n+1) \cdot \frac{d}{2},$$

and if we have a full binary tree with $2n-1$ nodes, our average cost per node will be

$$\frac{n+1}{2n-1} \cdot \frac{d}{2} \sim \frac{d}{4},$$

which means the asymptotic cost per node is $d/4$.

3.5.3 Amortized Complexity for General Hermite Forms

In the more general setting, suppose we have a Hermite interpolant $h_f(x)$ of f such that h_f interpolates f at the nodes $u = (u_0, \ldots, u_m)$ with multiplicities $h = \mu_0 = \mu_1 = \ldots = \mu_m$, where the nodes u are equally distributed in the input interval $\mathcal{I} = [a, b]$. That is,

$$u_i = a + \frac{i}{m}(b-a).$$

Then we know the degree of the Hermite interpolant is at most $d \coloneqq (m+1)h$, which is also the cost of realizing this Hermite interpolant over an interval. In order to amortize this cost over the entire subdivision tree T, let us first define $N_m(T)$ to be the number of distinct nodes for all intervals in T. We clarify distinctness as it is possible that two intervals I, J in T may share an interpolating node, in which case we are encouraged to reuse the data and not compute it twice.

Further, we can assume our tree has n leaves, re-define $T \leftarrow T_n$. Since T_n is a binary tree, the function $N_m(T_n)$ is independent of the shape of the tree in our analysis, and we can simplify notation to $N_m(n)$.

Therefore, the cost of evaluating the tree T_n is

$$C_d^{h_f}(n) \coloneqq h \cdot N_m(n).$$

Since a full binary tree with n leaves has $2n-1$ nodes, we can define the amortized cost per node for a general Hermite scheme to be

$$\overline{C}_d^{h_f} \coloneqq \lim_{n\to\infty} \frac{C_d^{h_f}(n)}{2n-1}.$$

Now, we can state and inductively prove our main theorem for amortized complexity analysis:

Theorem 3.8. *With $N_m(n)$, $C_d^{h_f}(n)$, and $\overline{C}_d^{h_f}$ as defined above, we have*

$$\begin{aligned} N_m(n) &= mn + 1, \\ C_d^{h_f}(n) &= h(mn + 1), \\ \overline{C}_d^{h_f} &= \frac{hm}{2} = \frac{1}{2}(d - h). \end{aligned}$$

Proof. Of course, as long as we show $N_m(n) = mn + 1$, the other two equations follow trivially. We claim that $N_m(n)$ satisfies the following recurrence relation:

$$N_m(n) = \begin{cases} m + 1 & \text{if } n = 1, \\ N_m(n_L) + N_m(n_R) - 1 & \text{if } n > 1. \end{cases}$$

This can be shown via induction, the base case is trivial. Consider the tree T_n with left sub-tree T_{n_L} and right sub-tree T_{n_R}, we see that the two intervals of the children have one common shared point, the midpoint. And since $n = n_L + n_R$, we are done. □

Remark. *While this is very useful to finding more optimal range functions, the theory is still primitive. We are also restricted to, given any d, using multiplicities h such that h divides d.*

4 Experiments

We have repeatedly mentioned in previous sections on the empirical efficiency of the novel range functions in the application of real-root isolation. This section will be to explain these concepts, methodologies, and experimental results.

We first begin by understanding the problem of real-root isolation, and how the Eval algorithm is able to solve this problem. We then turn to a theoretical limitation for using our previously constructed range functions for the Eval algorithm, as well as how previous results in numerical analysis literature circumvent this issue. Finally, we discuss the experimental methodology and settings for the experiment, and provide detailed tables and figures that capture the trends of the data.

We will see that the data does confirm our theoretical amortized complexity developed in Section 3.5.

4.1 Real-Root Isolation and the Eval algorithm

Consider the function $f : \mathbb{R} \to \mathbb{R}$, the zeros of the function on a subset of the domain $S \subseteq \mathbb{R}$ forms the set $Z_f(S) \coloneqq \{x \in S \mid f(x) = 0\}$. An *isolator* for f is an interval $I \in \square\mathbb{R}$ such that $|Z_f(I)| = 1$, i.e. I isolates the *unique* zero of f. We then formulate the root isolation problem for real-valued functions.

Problem 4.1 (Real-Root Isolation)**.** Given $f : \mathbb{R} \to \mathbb{R}$ and an interval $\mathcal{I} \in \square\mathbb{R}$, compute a set of intervals $Z = \{I_1, \ldots, I_n\} \subset \square\mathcal{I}$ such that $|Z_f(I_i)| = 1 \; \forall 1 \leq i \leq n$.

We pose the addition restriction that this computation must be *certified*, i.e. the output of any algorithm that solves this problem must be robust to all precision errors.

This makes the problem intractable in general. However, if we impose additional restrictions on f, namely that f is continuously differentiable and all roots of f in $\mathcal{I}$ have multiplicity 1, we are able to use the Eval algorithm to efficiently solve this problem with certification of correctness.

In terms of required numerical computation that Eval needs to do, we must have access to the range functions $\square f(\cdot)$ and $\square f'(\cdot)$. Eval uses two queue data structures to store intervals: Q for intervals yet to be processed, and Z for intervals that are isolators for f under $\mathcal{I}$. For a given interval in Q, it can

- Contain no roots, in which it is discarded.

- Contain a single root, in which it is added to Z.
- Contain more than one root, in which it is bisected about the midpoint and the two subintervals are added to the back of Q.

As seen in Algorithm 3, if the range functions $\square f$ and $\square f'$ are convergent on $\mathcal{I}$, then the algorithm will terminate and return the isolating set.

Algorithm 3 Real root isolation with range functions

Input: $f : \mathbb{R} \to \mathbb{R}$ and $\mathcal{I} \in \square\mathbb{R}$
Output: Z containing isolators for each $\zeta \in Z_f(\mathcal{I})$

```
1:  procedure Eval(f, I)
2:      initialize Q := {Ii} and Z := ∅
3:      while Q is non-empty do
4:          I := Q.pop(), where I = [a, b]
5:          if 0 ∈ □f(I) then                          ▷ I is implicitly discarded if 0 ∉ □f(I)
6:              if 0 ∈ □f'(I) then
7:                  Q.push([a, m], [m, b]), where m = m(I)
8:              else                                    ▷ f is strictly monotonic
9:                  if f(a)f(b) ≤ 0 then                ▷ 0 ∈ f(I)
10:                     Z.push(I)
11:     return Z
```

In Algorithm 3, the algorithm will push an interval into the solution $Z \Leftrightarrow f(a)f(b) \leq 0$. The intermediate value theorem guarantees correctness. Furthermore, the interval $[a, b]$ is guranteed to be strictly monotonic from the if condition that $0 \notin \square f'([a, b])$.

Additionally, if $f \in \mathbb{Q}[x]$ and our computation model uses rational arithmetic, the inequality in line 9 of Algorithm 3 is strict and we can further test $f((a+b)/2) = 0$ to see if the midpoint is a root since we will have exact rational representations in the computer.

4.2 Estimating the Range of the Derivative

Eval not only needs to construct a range function for f, but also for f'. Let us walk through the theoretical derivations for finding a range function for f' given that we have already computed a range function for f.

4.2.1 Generalized Taylor Forms

This issue resolves nicely if our choice of range function is a generalized Taylor form with recursion level $n-1$ for f' has form

$$\square^T_{k,n-1} f'(I) = g'_k(I) + [-1, 1] r^k S'_{k,n-1},$$

where

$$S'_{k,n-1} := \sum_{i=k+1}^{n-1} r^{i-k-1} \cdot \frac{|f^{(i)}(m)|}{(i-1)!} + r^{n-k-1} \cdot \frac{|\square f^{(n)}(I)|}{(n-1)!},$$

and $g'_k(I)$ is the derivative of the k-th order Taylor polynomial for f about m. Hence, we can easily see that the convergence order of the range function for both f and f' are the same (i.e. order k), and that the construction of these two range functions are also reliant on the same data.

4.2.2 Cubic Lagrange Forms

Let us first consider $\square^L_{3,n} f'(I)$. Here, we have more complications since the construction of this range function would require the evaluation of $f^{3j+1}(\{a, m, b\})$ for all j. If we compute this range function naïvely, this would double the computational cost and make the application of this range function in the context of real-root isolation useless.

However, we recall a result from [Shadrin 1995]:

Theorem 4.2 (Shadrin's Theorem)**.** *Let $f : \mathbb{R} \to \mathbb{R}$ be a real function and $h : \mathbb{R} \to \mathbb{R}$ be its Lagrange interpolating polynomial that interpolates f at $x_0, \ldots, x_\ell \in I$. Then*

$$\sup_{x \in I} |f^{(k)}(x) - h^{(k)}(x)| \leq |\omega^{(k)}(I)| \frac{f^{(\ell+1)}(I)}{(\ell+1)!}, \ 0 \leq k \leq \ell,$$

where

$$\omega(x) = \prod_{i=0}^{\ell} (x - x_i).$$

In the context of $\square^L_{3,n}$, we have

$$\sup_{x \in I} |f'(x) - h'_0(x)| \leq |\omega'(I)| \frac{|f^{(3)}(I)|}{6}.$$

Both of these terms are easily bounded:

$$\omega'(I) = [-1, 1] r^2, \quad \Omega |f^{(3)}(I)| \leq T_{3,n}.$$

Therefore, we can easily estimate $f'(I)$ using the following form,

$$\square^L_{2,n} f'(I) := h'_0(I) + \frac{3\sqrt{3}}{r} [-1, 1] T_{3,n},$$

where $T_{3,n}$ is as defined in (2.12), which depends on the same data as $\square^L_{3,n} f(I)$, albeit with only quadratic convergence. Note that this also applies to the cheap variant of the cubic Lagrange form (2.13).

However, this does not apply to $\square^L_{4,n}$ since $\hat{h}_0$ is no longer a Lagrange interpolant.

4.2.3 Quartic Hermite Forms

This case is also complicated as we do not want to double our computation time by separately constructing $\square^H_{4,\ell} f'(I)$. Let us first consider some Lagrange interpolant $L(x)$ for f, interpolating the four nodes $x_0, \ldots, x_3 \in I$. We can use Theorem 4.2 and get

$$\sup_{x \in I} |f'(x) - L'(x)| \leq \frac{|\omega'(I)|}{4!} |f^{(4)}(I)|, \quad \omega(x) = \prod_{i=0}^{3} (x - x_i).$$

We use an important remark in [Waldron 1997], that this supremum bound is continuous with respect to $\{x_i\}$. Hence, if we consider the limit as $x_0, x_1 \to a$ and $x_2, x_3 \to b$, our interpolant approaches (uniformly) h_0, which is our Hermite interpolant for $\square^H_{4,\ell}$:

$$\sup_{x \in I} |f'(x) - h_0'(x)| \leq \frac{\omega'(x)}{4!} |f^{(4)}(I)|.$$

Since we have $\omega'(I) = \frac{8}{9}\sqrt{3} r^3 [-1, 1]$, we can use the same recursive remainder bound that we have used prior to construct the remainder bound

$$|R'_{h_0}(I)| \leq \frac{8\sqrt{3}}{9} \frac{r^3}{4!} |f^{(4)}(I)| \leq \frac{8\sqrt{3}}{9r} S_\ell,$$

where S_ℓ is as defined in (3.2).

As a result, we have the following recursive Hermite form for f':

$$\square^H_{3,\ell} f'(I) \coloneqq h_0'(I) + \frac{8\sqrt{3}}{9r} S_\ell.$$

While these only have cubic convergence, they depend on the same data as $\square^H_{4,\ell} f(I)$, so we save computation time in practice. Note that this also work for the cheap variant by replacing S_ℓ with S'_ℓ.

4.3 Methodology and Settings

In practice, we implemented a version of Eval in C++ under the Core Library (see Core Library), which contains and implements various paradigms in exact computation. We also implemented all of the range functions mentioned in [Hormann et al. 2021] and in Chapter 3. In particular, the two box forms $\square^L_{3,\ell}$ and $\square^H_{4,\ell}$ and their associated cheap variants also had non-maximal recursion level forms implemented as well.

We ran the various introduced range functions with f belonging to various families of polynomials: dense with all roots real, corresponding to the Chebyshev polynomials T_n, Hermite polynomials H_n, and Wilkinson's polynomials W_n; dense with only two real roots, corresponding to the Mignotte cluster M_{2k+1}; and sparse without real roots, corresponding to S_n. Depending on the family of polynomials chosen, we provide different starting intervals I_0 to the Eval algorithm.

Holistically, we wish to measure the two components that best approximate the idea of 'performance' for the range functions: (1) size of the subdivision tree and (2) running time on (a) 1024-bit floating point arithmetic and (b) multi-precision rational arithmetic. Our experiments were ran on a Windows 10 laptop with a 1.8 GHz Intel Core i7-8550U processor with 16GB of RAM. For the multi-precision rational arithmetic case in particular, as the only non-rational number we have to explicitly compute is $\sqrt{3}$, it is replaced by the slightly larger number $17320508075688773 \times 10^{-16}$. Our implementation, including code and data, may be downloaded from the Core Library web page here.

In total, we tested eleven different range functions. Our first three range functions were the state-of-the-art performers in [Hormann et al. 2021]: $\square_2^T$, $\square_3^L$, $\square_3^{L'}$. We then tested three non-maximal variants of $\square_{3,\ell}^{L'}$, $\ell \in \{10, 15, 20\}$. The next two, $\square_4^H$ and $\square_4^{H'}$, are based on the quartic Hermite functions as well as the cheaper variant. Finally, the last three are the non-maximal cheap quartic Hermite range functions $\square_{4,\ell}^{H'}$, $\ell \in \{10, 15, 20\}$.

4.4 Results

Table 4.1 reports the sizes of the Eval subdivision trees, which serve as a measure of the tightness of the underlying range functions. In each row, the smallest tree size is underlined. As expected, the methods based on range functions with quartic convergence order outperform the others and in general the tree size decreases as the recursion level increases, except for sparse polynomials.

We further observe that the differences between the tree sizes for $\mathrm{E}_4^{L'}$ and $\mathrm{E}_4^{H'}$ are small, indicating that the tightness of a range function is determined mainly by the convergence order, but much less by the type of local interpolant (Lagrange or Hermite). However, as already pointed out in [Hormann et al. 2021], a smaller tree size does not necessarily correspond to a faster running time. In fact, $\mathrm{E}_3^{L'}$ was found to usually be almost as fast as $\mathrm{E}_4^{L'}$, even though the subdivision trees of $\mathrm{E}_3^{L'}$ are consistently bigger than those of $\mathrm{E}_4^{L'}$.

In Figure 4.2, we do a direct comparison of the Eval version based on our new range function $\mathrm{E}_4^{H'}$ with the previous leader $\mathrm{E}_3^{L'}$: for the test polynomials in our suite, the new function is faster for polynomials of degree greater than 25, with the speedup approaching and even exceeding the theoretical value of 1.33 of Section 4.2. In terms of tree size they are similar (differing by less than 5%, Table 4.1). Hence, $\mathrm{E}_4^{H'}$ emerges as the new winner among the practical range functions from our collection. even though it is possible to further decrease the runtime by taking advantage of range functions with non-maximal recursion levels.

4.4.1 Non-Maximal Recursion Levels

High order of convergence is important for applications such as numerical differential equations. But a sole focus on convergence order may be misleading as noted in [Hormann et al. 2021]: for any convergence order $k \geq 1$, a subsidiary measure may be critical in practice. For Taylor forms, this is the *refinement level* $n \geq k$ and for our recursive range functions, it is the *recursion level* $\ell \geq 0$. In [Hormann et al. 2021] we focused on maximal levels (for polynomials) after showing that the $\widetilde{\square}_2^T$ (the minimal level Taylor form of order 2) is practically worthless for the Eval algorithm.

Table 4.1: Size of the EVAL Subdivision Tree

f	$r(I_0)$	E_2^T	$E_3^{L'}$	$E_4^{L'}$	$E_{3,10}^{L'}$	$E_{3,15}^{L'}$	$E_{3,20}^{L'}$	E_4^H	$E_4^{H'}$	$E_{4,10}^{H'}$	$E_{4,15}^{H'}$	$E_{4,20}^{H'}$
T_{20}		319	243	231	243	243	243	239	239	239	239	239
T_{40}		663	479	463	479	479	479	471	479	479	479	479
T_{80}	10	1379	1007	955	1023	1007	1007	967	991	991	991	991
T_{160}		2147	1427	1347	1543	1451	1427	1351	1359	1439	1363	1359
T_{320}		-	2679	2575	3023	2699	2679	2591	2591	2803	2603	2591
H_{20}		283	215	207	215	215	215	199	207	207	207	207
H_{40}		539	423	415	423	423	423	415	419	419	419	419
H_{80}	40	891	679	655	711	679	679	659	683	695	683	683
H_{160}		1435	955	923	1083	959	955	923	927	1023	927	927
H_{320}		-	2459	2415	45287	10423	4419	2455	2499	15967	5195	3119
M_{21}		169	113	109	113	113	113	105	105	105	105	105
M_{41}	1	339	215	213	215	215	215	219	223	223	223	223
M_{81}		683	445	423	507	445	445	427	431	443	431	431
M_{161}		-	905	857	7245	1755	1047	861	861	2663	1079	905
W_{20}		485	353	331	353	353	353	331	335	335	335	335
W_{40}	1000	901	633	613	633	633	633	615	617	617	617	617
W_{80}		1583	1133	1083	2597	1133	1133	1097	1117	1485	1117	1117
W_{160}		-	2005	1935	293509	5073	2005	1959	1993	42413	5289	2817
S_{100}		973	633	609	611	621	625	613	613	595	609	613
S_{200}	10	1941	1281	1221	1211	1227	1237	1231	1231	1165	1187	1201
S_{400}		-	2555	2435	2379	2399	2413	2467	2467	2289	2319	2339

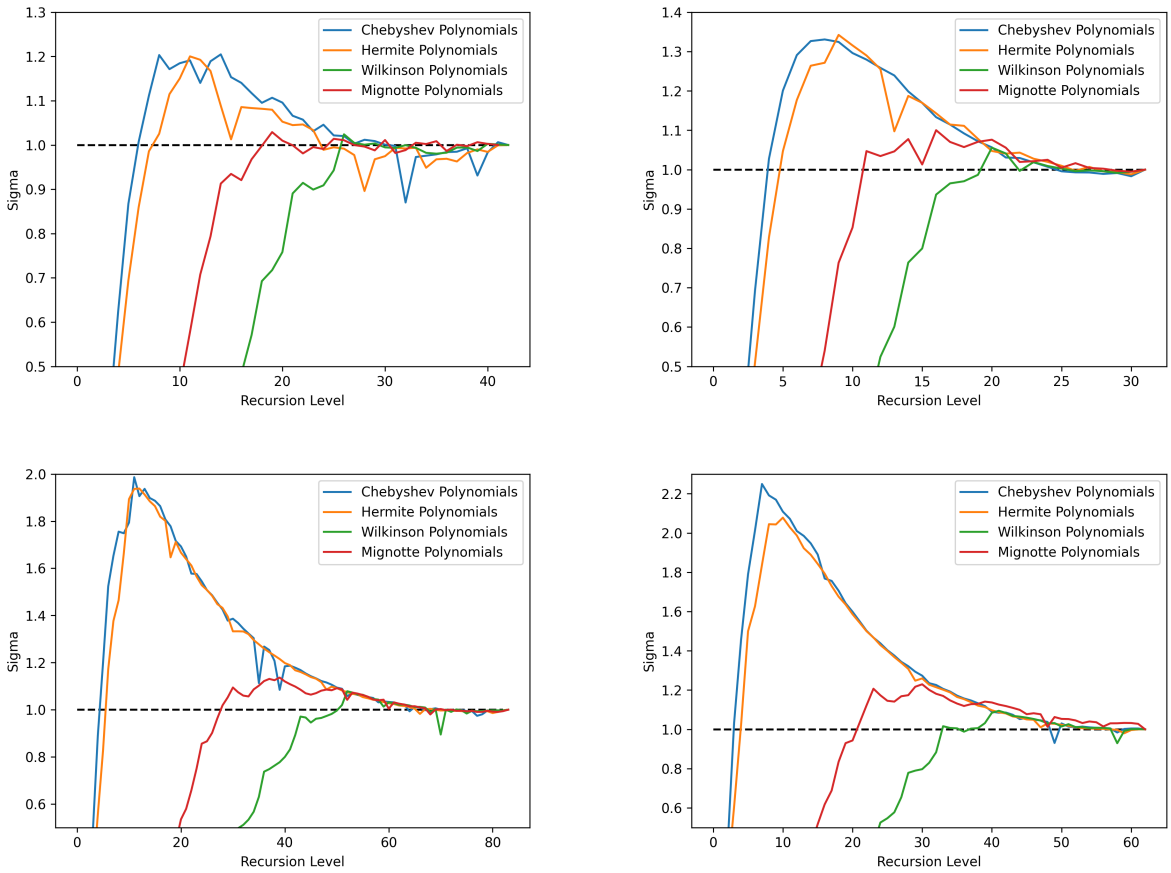


Figure 4.1: Speedup $\sigma(\ell)$ of $\mathrm{E}^{L'}_{3,\ell}$ (left) and $\mathrm{E}^{H'}_{4,\ell}$ (right) against their maximal level counterparts with respect to ℓ for polynomials of degree 125 (top) and 250 (bottom) from different families.

We now experimentally explore the use of non-maximal levels.

Figure 4.1 plots the (potential) **level speedup factor** $\sigma(\ell)$ against level $\ell \geq 0$. More precisely, consider the time for Eval to isolate the roots of a polynomial f in some interval I_0. Let $\square_{k,\ell} f$ be a family of range functions of order k, but varying levels $\ell \geq 0$. If $E_{k,\ell}$ (resp., E_k) is the running time of Eval using $\square_{k,\ell} f$ (resp., $\square_{k,\infty} f$), then $\sigma(\ell) := E_k / E_{k,\ell}$. Of course, it is only a true speedup if $\sigma(\ell) > 1$. These plots support our intuition in [Hormann et al. 2021] that minimal levels are rarely useful (except at low degrees). Most strikingly, the graph of $\sigma(\ell)$ shows a characteristic shape of rapidly increasing to a unique maxima and then slowly tapering to 1, especially for polynomials f with high degrees. This suggests that for each polynomial, there is an optimal level to achieve the greatest speedup. In our tests (see Figure 4.1), we saw that both the optimal level and the value of the corresponding greatest speedup factor depend on f. Moreover, we observed that the achievable speedup tends to be bigger for $\mathrm{E}^{H'}_4$ than for $\mathrm{E}^{L'}_3$ and that it increases with the degree of the polynomial f.

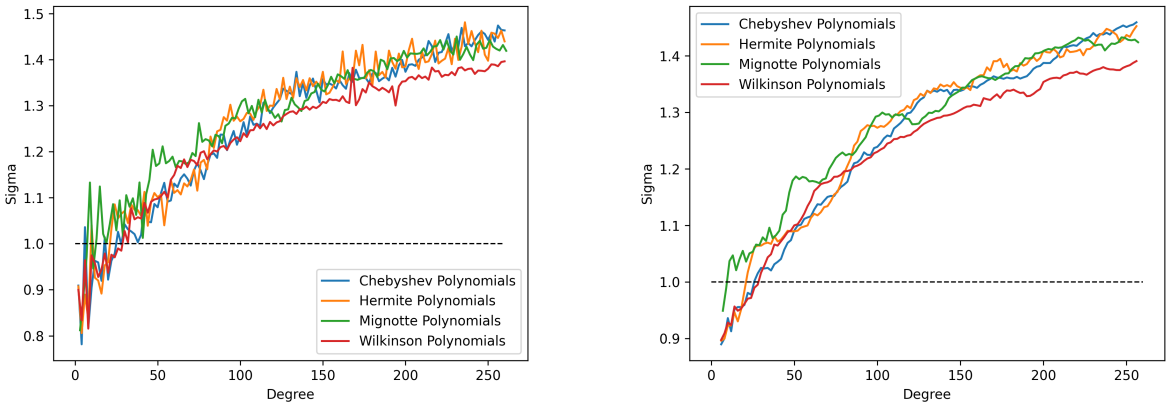


Figure 4.2: Speedup σ of $\mathrm{E}_4^{H'}$ with respect to $\mathrm{E}_3^{L'}$ for different families of polynomials and varying degree: raw (left) and smoothed with moving average over five points (right).

4.4.2 Running Times

In Table 4.2 and Table 4.3 we report the running times of our eleven different Eval versions for several families of polynomials. Times are given in seconds and averaged over at least four runs (and many more for small degree polynomials). Generally, $\mathrm{E}_{k,\ell}^{X}$ ($X = T, L, H$ for Taylor, Lagrange, Hermite forms) is the running time of Eval using the appropriate forms of order k and level ℓ (ℓ may be omitted when the level is maximal)

The last three columns in both tables report the speedup ratios $\sigma(\cdot)$ of $\mathrm{E}_4^{H'}$, $\mathrm{E}_{4,15}^{H'}$, and $\mathrm{E}_{3,15}^{L'}$ with respect to $\mathrm{E}_3^{L'}$, which was identified as the overall winner in [Hormann et al. 2021].

Table 4.2: Average Running Time of Eval with 1024-bit Floating Point Arithmetic in Seconds.

f	$r(I_0)$	E_2^T	$\mathrm{E}_3^{L'}$	$\mathrm{E}_4^{L'}$	$\mathrm{E}_{3,10}^{L'}$	$\mathrm{E}_{3,15}^{L'}$	$\mathrm{E}_{3,20}^{L'}$	E_4^H	$\mathrm{E}_4^{H'}$	$\mathrm{E}_{4,10}^{H'}$	$\mathrm{E}_{4,15}^{H'}$	$\mathrm{E}_{4,20}^{H'}$	$\sigma(\mathrm{E}_4^{H'})$	$\sigma(\mathrm{E}_{4,15}^{H'})$	$\sigma(\mathrm{E}_{3,15}^{L'})$
T_{20}		0.0288	0.0152	0.0153	0.0179	0.0212	0.0243	0.0201	0.0157	0.023	0.0274	0.0316	0.97	0.57	0.72
T_{40}		0.19	0.0669	0.0663	0.0723	0.068	0.0726	0.078	0.0637	0.0864	0.0944	0.102	1.05	0.71	0.98
T_{80}	10	1.35	0.379	0.363	0.366	0.386	0.397	0.398	0.327	0.465	0.494	0.49	1.16	0.77	0.98
T_{160}		8.23	1.82	1.71	1.23	1.35	1.45	1.61	1.38	1.56	1.78	2.04	1.31	1.02	1.35
T_{320}		-	12.7	12.1	5.11	5.44	6.19	10.4	9.53	6.68	7.84	9.29	1.33	1.62	2.34
H_{20}		0.0242	0.0127	0.013	0.0149	0.0177	0.0204	0.0159	0.0128	0.0191	0.0226	0.0256	0.99	0.56	0.72
H_{40}		0.15	0.0575	0.058	0.0632	0.0601	0.0652	0.0709	0.0547	0.0862	0.092	0.0923	1.05	0.63	0.96
H_{80}	40	0.881	0.259	0.255	0.26	0.263	0.266	0.273	0.225	0.324	0.349	0.346	1.15	0.74	0.98
H_{160}		5.47	1.22	1.16	0.854	0.872	0.953	1.1	0.972	1.1	1.23	1.38	1.26	1.00	1.4
H_{320}		-	11.6	11.4	77.4	21.2	10.3	9.88	9.21	38.4	15.7	11.3	1.26	0.74	0.55
M_{21}		0.0223	0.00767	0.00726	0.00826	0.0101	0.0123	0.00881	0.0072	0.0104	0.0125	0.0143	1.07	0.61	0.76
M_{41}	1	0.103	0.032	0.0319	0.0349	0.0325	0.035	0.0391	0.0309	0.0417	0.0444	0.0489	1.03	0.72	0.99
M_{81}		0.707	0.169	0.159	0.179	0.168	0.173	0.174	0.14	0.203	0.217	0.214	1.21	0.78	1.01
M_{161}		-	1.2	1.13	5.96	1.68	1.09	1.05	0.898	2.96	1.53	1.62	1.34	0.79	0.72
W_{20}		0.0492	0.0222	0.0201	0.0212	0.0211	0.0211	0.0261	0.0205	0.0256	0.026	0.0256	1.08	0.85	1.05
W_{40}	1000	0.282	0.0873	0.0874	0.096	0.0918	0.0995	0.114	0.0858	0.111	0.112	0.111	1.02	0.78	0.95
W_{80}		1.82	0.426	0.416	0.936	0.449	0.439	0.467	0.38	0.706	0.576	0.562	1.12	0.74	0.95
W_{160}		-	2.74	2.65	257	5.56	2.68	2.52	2.22	49.8	7.52	4.59	1.23	0.37	0.49
S_{100}		1.33	0.351	0.337	0.293	0.331	0.351	0.35	0.286	0.378	0.436	0.461	1.23	0.81	1.06
S_{200}	10	9.55	2.32	2.21	1.2	1.41	1.59	2.02	1.77	1.6	1.98	2.31	1.31	1.18	1.65
S_{400}		-	16.6	15.9	4.89	5.84	6.66	13.4	12.5	6.46	8.28	9.98	1.34	2.01	2.85

Table 4.3: Average Running Time of Eval with Multi-Precision Rational Arithmetic in Seconds.

f	$r(I_0)$	E_2^T	$\mathrm{E}_3^{L'}$	$\mathrm{E}_4^{L'}$	$\mathrm{E}_{3,10}^{L'}$	$\mathrm{E}_{3,15}^{L'}$	$\mathrm{E}_{3,20}^{L'}$	E_4^H	$\mathrm{E}_4^{H'}$	$\mathrm{E}_{4,10}^{H'}$	$\mathrm{E}_{4,15}^{H'}$	$\mathrm{E}_{4,20}^{H'}$	$\sigma(\mathrm{E}_4^{H'})$	$\sigma(\mathrm{E}_{4,15}^{H'})$	$\sigma(\mathrm{E}_{3,15}^{L'})$
T_{20}		0.0411	0.0223	0.0245	0.0269	0.0325	0.0378	0.0417	0.0233	0.0347	0.0429	0.0505	0.96	0.52	0.69
T_{40}		0.261	0.11	0.111	0.121	0.109	0.117	0.146	0.0959	0.126	0.141	0.156	1.15	0.78	1.01
T_{80}	10	1.76	0.631	0.611	0.62	0.644	0.658	0.824	0.524	0.769	0.805	0.781	1.2	0.78	0.98
T_{160}		11.3	3.14	2.87	2.23	2.36	2.62	3.82	2.41	2.7	2.96	3.36	1.3	1.06	1.33
T_{320}		-	31.8	30.8	13.7	14.1	15.9	36.2	21.8	16.6	18.5	21.8	1.46	1.72	2.25
H_{20}		0.03	0.0169	0.0182	0.0205	0.025	0.0296	0.0239	0.0176	0.0273	0.0338	0.0402	0.96	0.50	0.68
H_{40}		0.185	0.0858	0.0885	0.0956	0.0927	0.106	0.131	0.0844	0.109	0.123	0.136	1.02	0.70	0.93
H_{80}	40	1.1	0.399	0.391	0.41	0.412	0.423	0.541	0.329	0.495	0.523	0.504	1.21	0.76	0.97
H_{160}		7.51	1.99	1.89	1.5	1.51	1.65	2.55	1.47	1.81	1.87	2.13	1.35	1.06	1.32
H_{320}		-	29.5	28.9	303	67	27.7	39.1	20.9	123	40.8	26.2	1.41	0.72	0.44
M_{21}		0.0238	0.0115	0.0119	0.013	0.0154	0.0179	0.015	0.0106	0.0162	0.0198	0.0233	1.09	0.58	0.75
M_{41}	10	0.124	0.0466	0.0478	0.0529	0.0488	0.0537	0.07	0.0471	0.066	0.0746	0.0847	0.99	0.63	0.96
M_{81}		0.947	0.298	0.278	0.321	0.288	0.293	0.381	0.236	0.346	0.359	0.344	1.27	0.83	1.04
M_{161}		-	2.18	2.03	13.6	3.29	2.08	2.64	1.57	5.89	2.62	2.42	1.39	0.83	0.66
W_{20}		0.0652	0.0332	0.0346	0.0344	0.0343	0.0346	0.0491	0.0352	0.0445	0.0442	0.0452	0.94	0.75	0.97
W_{40}	1000	0.431	0.18	0.176	0.182	0.163	0.161	0.225	0.143	0.191	0.195	0.191	1.26	0.92	1.1
W_{80}		2.75	0.846	0.826	1.96	0.877	0.847	1.15	0.708	1.41	1.1	1.09	1.2	0.77	0.97
W_{160}		-	6.28	6.1	932	14.6	6.21	8.22	4.78	155	19	10.6	1.31	0.33	0.43
S_{100}		1.35	0.474	0.457	0.451	0.483	0.477	0.663	0.419	0.603	0.591	0.57	1.13	0.80	0.98
S_{200}	10	12	3.65	3.49	2.28	2.59	2.83	4.79	2.68	2.73	3.13	3.59	1.36	1.17	1.41
S_{400}		-	44.8	42.7	16.4	18.9	21.5	51.8	30	19.6	24.2	28.3	1.50	1.85	2.37

5 Conclusion

We have generalized the Cornelius and Lohner framework in order to achieve, for the first time, range functions that converge with arbitrarily high order. This is done by first developing sufficient theory, then transposing the theory into a realization of new recursive schemes, the construction of which are of importance theoretically, analytically, and practically. We have also chosen specific range functions such as $\square^{H'}_{4,\ell}$ that outperform other similar range functions, which we then show is significant for applications in real-root isolation.

Additionally, the amortized complexity model is also applicable in other problems that utilize subdivision algorithms, perhaps in higher dimensions (i.e compact interval boxes rather than intervals in the real line). The application of amortized complexity analysis to our current range functions yield theoretical speedup percentages that are well supported by our computational experiments.

Finally, the development of strong box functions for the generalized Cornelius and Lohner framework may also find importance in other applications.

5.1 Future Work

At the moment, there are a few possible extensions of the work presented, whether further in real-root isolation, range functions, or in entirely different domains. One possible example is to theoretically explain the 'uni-modal' phenomenon of $\sigma(\ell)$ captured by Figure 4.1 in order to develop techniques that allow for a priori estimation of the optimal choice of recursion level ℓ (with respect to minimum run time) for a given class of functions. We suspect that this inherently is dependent on the 'roughness' of the function, and leave the notion of 'roughness' to be fit with various metrics such as Hölder conditions, Lipschitz conditions, or others.

Moreover, we would also like to further understand the idiosyncrasies of the class of sparse polynomials S_n and why the size of the Eval subdivision tree increases with ℓ (in contrast to the other polynomial classes, which decrease as ℓ increases).

Another possible extension is to further develop the theory of strong box functions as it is quite primitive in its current form.

A | Appendix

Below is a program synopsis for the Eval program under the Core Library.

EVAL Program Synopsis

Introduction

This file provides a high level overview of the purpose and methods of classes used in the EVAL program, the relationship between these classes are also illustrated. For further details, please consult the README file or refer to the paper **K. Hormann, L. Kania, C. Yap. 2021. Novel Range Functions via Taylor Expansions and Recursive Lagrange Interpolation with Applications to Real Root Isolation. *ISSAC 2021*.**
Note: Some testing, debugging, timing, and other 'unessential' attributes and methods are ignored. All classes detailed below can be implemented using either rational or bigfloat arithmetic (Which can be initialized by base.h, baseF.h, and baseR.h).

Main Entry Point

After compilation, the EVAL program can be ran from the terminal by using preset commands from the Makefile (The Makefile commands provides all the customization needed). There are three EVAL executables: eval.exe (General Eval), evalF.exe (General Eval with Floating Point Arithmetic), evalR.exe (General Eval with Rational Arithmetic).

List of Files

- data
  - chebyshev.int chebyshev020.pol chebyshev040.pol chebyshev080.pol chebyshev160.pol chebyshev320.pol
  - hermite.int hermite020.pol hermite040.pol hermite080.pol hermite160.pol hermite320.pol
  - sparse.int sparse0100.pol sparse0200.pol sparse0400.pol sparse0800.pol
  - wilk.int wilk020.pol wilk040.pol wilk080.pol wilk160.pol wilk320.pol
- base.h baseF.h baseR.h
- eval.cpp eval.h
- evalL3.cpp evalL3.h
- evalL3cheap.cpp evalL3cheap.h
- evalL4.cpp evalL4.h
- evalL4cheap.cpp evalL4cheap.h
- evalLagrange.h
- evalT2.cpp evalT2.h
- evalT2min.cpp evalT2min.h
- evalT3.cpp evalT3.h
- evalT4.cpp evalT4.h
- evalTaylor.h
- exactRange.h
- interval.cpp interval.h
- mignotte.cpp
- polynomial.cpp polynomial.h
- eval.exe evalR.exe evalF.exe
- README Makefile

Supportive Classes

Class: Interval

Real a, b; ▷ Left/right endpoints
Integer $level$; ▷ Level of subtree (num. times the interval has been halved)
Integer $a_{data}, m_{data}, b_{data}$; ▷ Ptr to each point's list of derivatives: $[f^{(0)}(a/m/b), \ldots, f^{(n)}(a, m, b)]$

Interval(): $a, b, level \leftarrow 0$;
Interval(x): $a, b \leftarrow x$; $level \leftarrow 0$;
Interval($x, y, l = 0$): $a \leftarrow x$; $b \leftarrow y$; $level \leftarrow l$; ▷ If $x > y$ switch assignments

Methods for interval arithmetic, interval predicate, midpoint, width, and radius are implemented.

Class: Polynomial

Integer n; ▷ Degree of polynomial
Vector⟨Real⟩ c; ▷ Coefficients

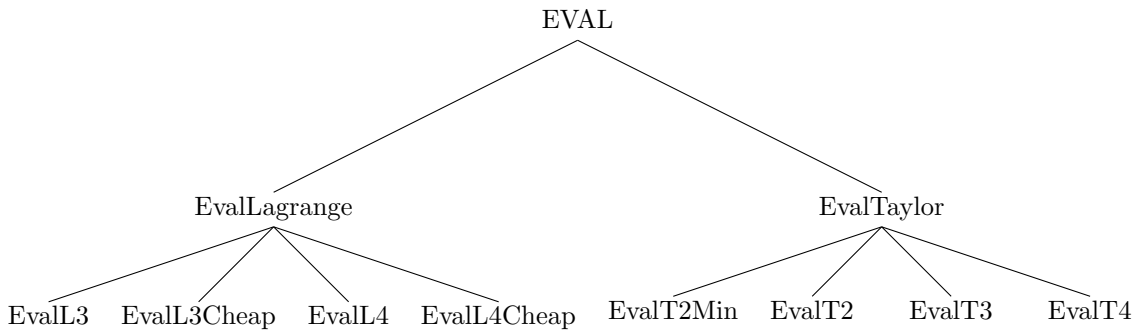
Polynomial(): $n \leftarrow -1$; $c \leftarrow 0$; ▷ Zero Polynomial
Polynomial(Integer nn): $n \leftarrow nn$; $c \leftarrow \langle 0, \ldots, 0 \rangle$ ▷ User fills in coefficients
Polynomial(Vector⟨Real⟩ v): $c \leftarrow v$; $n \leftarrow dim(v)$; ▷ Highest degree coefficient can't be 0

Real/Interval Eval(Real/Interval x): Return $f(x)$; ▷ Interval evaluated using Horner's method
Real EvalDiff(Real x, Integer $k = 1$): Return $f^{(k)}(x)$;
Real EvalTaylor(Real x, Integer $k = 1$): Return $\frac{f^{(k)}(x)}{k!}$;
Polynomial Shift(Real m, Integer $t = -1$):
    $g(x) \leftarrow f(x + m)$; Truncate g at degree t; ▷ If $t = -1$, do not truncate
    Return g
Polynomial Diff(Integer $k = 1$): Return $f^{(k)}$;

Class: ExactRange

Interval LinearRange(Real c0,c1,r):
    Return exact range of $c_0 + c_1(I - m)$; ▷ $I = [m - r, m + r]$, m is the midpoint
Interval QuadraticRange(Real c0,c1,c2,r):
    Return exact range of $c_0 + c_1(I - m) + c_2(I - m)^2$;
Interval CubicRange(Real c0,c1,c2,c3,r):
    Return exact range of $c_0 + c_1(I - m) + c_2(I - m)^2 + c_3(I - m)^3$;

Class Relation Tree



Core Classes

Class: Eval

Polynomial f; ▷ Input polynomial
Interval I_0; ▷ Interval to search
Integer m; ▷ Number of roots, if known
Vector⟨Real⟩ z; ▷ Roots of f, if known
Queue⟨Interval⟩ Q; ▷ Queue of intervals to be checked
Vector⟨Interval⟩ Z; ▷ Intervals that contain a single zero
Integer $NumInt$; ▷ Total num. of intervals (tree size)
String $name$; ▷ Name of range function method

Eval(Vector⟨Real⟩ v, Interval I):
    $f \leftarrow$polynomial(v);
    $f.n \leftarrow dim(v) - 1$;
    $m \leftarrow 0$;
    $I_0 \leftarrow I$;
Eval():
    Read c_{in} and parse into $f, f.n, m, I_0$;

Virtual Interval SplitLeft(Interval I); ▷ To be implemented in EvalLagrange & EvalTaylor
Virtual Interval SplitRight(Interval I);
Virtual Interval Get_f(Interval I); ▷ To be implemented in leaf classes from class relation tree
Virtual Interval $Get_{f'}$(Interval I);

void EVAL(): Runs the EVAL algorithm;

Class: EvalTaylor — Subclass of: Eval

Integer n; ▷ Degree of input polynomial f
Vector⟨Real⟩ c; ▷ Coefficients of Taylor expansion of f about m

EvalTaylor(Vector⟨Real⟩ v, Interval I): super(v, I); $n \leftarrow f.n$; Resize c into $n+1$ dimensional vector;
EvalTaylor(): super(); $n \leftarrow f.n$; Resize c into $n+1$ dimensional vector;

void ComputeTaylorCoefficients(Real m): Computes & stores Taylor coefficients of f about m in c;
Interval SplitLeft(Interval I): $[a, m].level \leftarrow I.level + 1$; Return $[a, m]$; ▷ m is midpoint of $I = [a, b]$
Interval SplitRight(Interval I): $[m, b].level \leftarrow I.level + 1$; Return $[m, b]$;

Class: EvalLagrange — Subclass of: Eval

Integer N; ▷ $\lfloor d/3 \rfloor$, where d is deree of input polynomial f
Vector⟨ Vector⟨Real⟩ ⟩ $data$; ▷ Data values at interval endpoints and midpoint

EvalLagrange(Vector⟨Real⟩ v, Interval I): super(v, I); $N \leftarrow (f.n)/3$;
EvalLagrange(): super(); $N \leftarrow (f.n)/3$;

void ComputeDerivatives(Vector⟨Real⟩ d, Real m):
    Compute and store $d \leftarrow [f(m), f^{(3)}(m), f^{(6)}(m), \ldots, f^{(3N)}(m)]$;
Interval SplitLeft(Interval I): $[a, m].level \leftarrow I.level + 1$; Return $[a, m]$; ▷ m is midpoint of $I = [a, b]$
Interval SplitRight(Interval I): $[m, b].level \leftarrow I.level + 1$; Return $[m, b]$;

Note: $Get_{f'}$ should be called after Get_f as it uses coefficients of the Hermite interpolant's coefficients already computed in Get_f.

Class: EvalL3 — Subclass of: EvalLagrange

Real $d_{0,0}, d_{0,1}, d_{0,2}$; ▷ Coefficients of $h_0(x)$
Real T; ▷ Remainder estimate $T_{3,n}$

EvalL3(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalL3';
EvalL3(): super(); $name \leftarrow$ 'EvalL3';

Interval Get_f(Interval I): Estimates range of f over I using Lagrange forms with cubic convergence;
Interval $Get_{f'}$(Interval I): Estimates range of f' over I using Lagrange forms with cubic convergence;

Class: EvalL3Cheap — Subclass of: EvalLagrange

Real $d_{0,0}, d_{0,1}, d_{0,2}$; ▷ Coefficients of $h_0(x)$
Real T'; ▷ Cheap remainder estimate $T'_{3,n}$

EvalL3Cheap(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalL3Cheap';
EvalL3Cheap(): super(); $name \leftarrow$ 'EvalL3Cheap';

Interval Get_f(Interval I): Estimates range of f over I using Lagrange forms with cubic convergence;
Interval $Get_{f'}$(Interval I): Estimates range of f' over I using Lagrange forms with cubic convergence;

Class: EvalL4 — Subclass of: EvalLagrange

Real $\hat{d}_{0,0}, \hat{d}_{0,1}, \hat{d}_{0,2}$; ▷ Coefficients of $\hat{h}_0(x)$
Real T; ▷ Remainder estimate $T_{4,n}$
Real $f^{(3)}(m)$;
Interval $\hat{h}_1(I)$;
Real Ω_3;

EvalL4(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalL4';
EvalL4(): super(); $name \leftarrow$ 'EvalL4';

Interval Get_f(Interval I): Estimates range of f over I using Lagrange forms with quartic convergence;
Interval $Get_{f'}$(Interval I): Estimates range of f' over I using Lagrange forms with quadratic convergence;

Class: EvalL4Cheap — Subclass of: EvalLagrange

Real $\hat{d}_{0,0}, \hat{d}_{0,1}, \hat{d}_{0,2}$; ▷ Coefficients of $\hat{h}_0(x)$
Real T'; ▷ Cheap remainder estimate $T'_{4,n}$
Real $f^{(3)}(m)$;
Interval $\hat{h}_1(I)$;
Real Ω_3;

EvalL4Cheap(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalL4Cheap';
EvalL4Cheap(): super(); $name \leftarrow$ 'EvalL4Cheap';

Interval Get_f(Interval I): Estimates range of f over I using Lagrange forms with quartic convergence;
Interval $Get_{f'}$(Interval I): Estimates range of f' over I using Lagrange forms with quadratic convergence;

Note: $Get_{f'}$ should be called after Get_f as it uses coefficients of the Taylor expansions already computed in Get_f.

Classes: EvalT2 — Subclass of: EvalTaylor

1: EvalT2(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalT2';
2: EvalT2(): super(); $name \leftarrow$ 'EvalT2';
3:
4: Interval Get_f(Interval I): Estimates range of f over I using quadratic Taylor form;
5: Interval $Get_{f'}$(Interval I): Estimates range of f' over I using quadratic Taylor form;

Classes: EvalT2Min — Subclass of: EvalTaylor

1: EvalT2Min(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalT2Min';
2: EvalT2Min(): super(); $name \leftarrow$ 'EvalT2Min';
3:
4: Interval Get_f(Interval I): Estimates range of f over I using minimal quadratic Taylor form;
5: Interval $Get_{f'}$(Interval I): Estimates range of f' over I using minimal quadratic Taylor form;

Classes: EvalT3 — Subclass of: EvalTaylor

1: EvalT3(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalT3';
2: EvalT3(): super(); $name \leftarrow$ 'EvalT3';
3:
4: Interval Get_f(Interval I): Estimates range of f over I using cubic Taylor form;
5: Interval $Get_{f'}$(Interval I): Estimates range of f' over I using cubic Taylor form;

Classes: EvalT4 — Subclass of: EvalTaylor

1: EvalT4(Vector⟨Real⟩ v, Interval I): super(v, I); $name \leftarrow$ 'EvalT4';
2: EvalT4(): super(); $name \leftarrow$ 'EvalT4';
3:
4: Interval Get_f(Interval I): Estimates range of f over I using quartic Taylor form;
5: Interval $Get_{f'}$(Interval I): Estimates range of f' over I using quartic Taylor form;

Bibliography

Alefeld, G., Herzberger, J., and Rockne, J. (2012). *Introduction to Interval Computation*. Elsevier Science, St. Louis.

Cornelius, H. and Lohner, R. (1984). Computing the range of values of real functions with accuracy higher than second order. *Computing*, 33(3–4):331–347.

Du, Z., Eleftheriou, M., Moreira, J., and Yap, C. (2002). Hypergeometric functions in exact geometric computation. *Electronic Notes in Theoretical Computer Science*, 66:53–64.

Du, Z. and Yap, C. (2006). *Uniform complexity of approximating hypergeometric functions with absolute error*, pages 246–249. Proceedings of the 7th Asian Symposium on Computer Math (ASCM 2005).

Hormann, K., Kania, L., and Yap, C. (2021). Novel range functions via taylor expansions and recursive lagrange interpolation with application to real root isolation. In *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*, ISSAC '21, page 193–200, New York, NY, USA. Association for Computing Machinery.

Li, C. and Yap, C. (2001). A new constructive root bound for algebraic expressions. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, page 496–505, USA. Society for Industrial and Applied Mathematics.

Moore, R. E. (1979). *Methods and Applications of Interval Analysis*. Society for Industrial and Applied Mathematics.

Moore, R. E., Kearfott, R. B., and Cloud, M. J. (2009). *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics.

Neumaier, A. (1990). *Interval methods for systems of equations*. Encyclopedia of mathematics and its applications. Cambridge University Press.

Neumaier, A. (2003). Taylor forms–use and limits. *Reliab. Comput.*, 9(1):43–79.

Rudin, W. (1976). *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill.

Shadrin, A. (1995). Error bounds for lagrange interpolation. *Journal of Approximation Theory*, 80(1):25–49.

Waldron, S. (1997). Lp-error bounds for hermite interpolation and the associated wirtinger inequalities. *Constructive Approximation*, 13:461–479.