Range Functions of Any Convergence Order and their Amortized Complexity Analysis

Kai Hormann[1], Chee Yap[2], and Ya Shi Zhang[2]

[1] Università della Svizzera italiana
[2] Courant Institute, NYU

Abstract. We address the fundamental problem of computing range functions $\square f$ for a real function $f\colon \mathbb{R} \to \mathbb{R}$. 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. There are 2 problems: this approach limits the order of convergence to 6, and exact computation is impossible to achieve in practice. 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. We further explore speeding up applications by choosing non-maximal recursion levels.

1 Introduction

Given a real function $f : \mathbb{R} \to \mathbb{R}$, the problem of tightly enclosing its range $f(I) = \{f(x) : x \in I\}$ on any interval I is a central problem of interval and certified computations [11,13]. The interval form of f may be[3] denoted $\square f : \square\mathbb{R} \to \square\mathbb{R}$ where $\square\mathbb{R}$ is the set of compact intervals and $\square f(I)$ contains the range $f(I)$. Cornelius & Lohner [3] provided a general framework for constructing such $\square f$. First, choose a suitable $g : \mathbb{R} \to \mathbb{R}$ such that for any interval $I \in \square\mathbb{R}$, we can

[3] Definitions of our terminology are collected in Section 1.3.

compute $g(I)$ exactly. Then $f(I) = g(I) + R_g(I)$ where $R_g(x) := f(x) - g(x)$. The standard measure for the accuracy of approximate functions like $\square f$ is their order of convergence. Suppose R_g has an interval form $\square R_g$ with convergence order $n \geq 1$. Then

$$\square_g f(I) := g(I) + \square R_g(I) \tag{1}$$

is an interval form for f with order of convergence n. This is an immediate consequence of the following theorem.

Theorem A [3, Theorem 4].

$$d_H(f(I), \square_g f(I)) \leq w(\square R_g(I)).$$

where d_H is the Hausdorff distance on intervals and $w(I)$ is the width of interval I.

Prior to [3], interval forms with convergence order larger than 2 were unknown. Cornelius & Lohner showed that there exists g such that $\square R_g$ has convergence order up to 6.

Example 1. Let $g(x)$ be the Taylor expansion of $f(x)$ at $x = m$ up to order $n \geq 1$ and $R_g(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x-m)^{n+1}$ for some ξ_x between x and m. Then the following is a range function for $R_g(I)$:

$$\square R_g(I) := \frac{|\square f^{(n+1)}(I)|}{(n+1)!}\left(\frac{w(I)}{2}\right)^{n+1}, \tag{2}$$

where $I = [a, b]$, $m = (a+b)/2$, and $w(I) = b - a$. Assuming that $I \subseteq I_0$ for some bounded I_0, we have $\frac{|\square f^{(n+1)}(I)|}{(n+1)!} = O(1)$. Then (2) implies that $\square R_g(I)$ has convergence order $n+1$. If $n \leq 3$, then $g(I)$ can be computed "exactly" as described in [9, Appendix]. This proves the existence of range functions of order 4.

1.1 Why we must extend the CL framework

Unfortunately, there is an issue with the CL framework: *we cannot compute the "exact range $g(I)$" in any standard implementation models.* Standard implementation models include (i) the IEEE arithmetic used in the majority of implementations, (ii) the Standard Model of Numerical Analysis [8,17], or (iii) bigNumber packages such as GMP [7], MPFR [6], and MPFI [14]. In practice, "real numbers" are represented by dyadic numbers, i.e., rational numbers of the form $m2^n$ where $m, n \in \mathbb{Z}$. So, rational numbers like $1/3$ cannot be exactly represented. Even if we allow arbitrary rational numbers, irrational numbers like $\sqrt{2}$ are not exact. See, for example, [20] for an extended discussion of exact computation. In computer algebra systems, the largest set of real numbers which can be computed exactly are the algebraic numbers, but we do not include them under "standard implementation models" because of inherent performance issues.

In [9], we used the term "exact computation of $g(I)$" in a sense which is commonly understood by interval and numerical analysts, including Cornelius & Lohner. But first let us address the non-interval case: the "exact computation of $g(x)$". The common understanding amounts to:

$$\begin{array}{l} g(x) \textit{ can be computed exactly if } g(x) \textit{ has a closed-form} \\ \textit{expression } E(x) \textit{ over a set } \Omega \textit{ of primitive operations.} \end{array} \tag{3}$$

There is no universal consensus on the set Ω but typically all real constants, four rational operations ($\pm, \times, \div$), and $\sqrt{\cdot}$ are included. E.g., Neumaier [11, p. 6] allows these additional operations in Ω:

$$|\cdot|, sqr, \exp, \ln, \sin, \cos, \arctan,$$

where[4] sqr denotes squaring. Next, how does the understanding (3) extend to the exact computation of $g(I)$? Cornelius & Lohner stated a sufficient condition that is well-known in interval analysis [3, Theorem 1]:

$$\begin{array}{l} g(I) \textit{ can be computed exactly if there is an expression } E(x) \\ \textit{for } g(x) \textit{ in which the variable } x \textit{ occurs at most once.} \end{array} \tag{4}$$

It is implicitly assumed in (4) that, given an expression $E(x)$ for $g(x)$, we can compute $g(I)$ by evaluating the interval expression $E(I)$, assuming all the primitive operators in $E(x)$ have exact interval forms. But this theorem has very limited application, and cannot even compute the exact range of a quadratic polynomial $g(x) = ax^2 + bx + c$ with $ab \neq 0$.

Example 2. To overcome the limitations of (4) in the case of a quadratic polynomial $g(x) = ax^2 + bx + c$, we can proceed as follows: first compute $x^* = -b/2a$, the root of $g'(x) = 2ax + b$. If $I = [\underline{x}, \overline{x}]$), then

$$g(I) = [\min(S), \max(S)],$$

where

$$S := \begin{cases} \{g(\underline{x}), g(\overline{x})\} & \text{if } x^* \notin I, \\ \{g(x^*), g(\underline{x}), g(\overline{x})\} & \text{else.} \end{cases}$$

We call this the "endpoints algorithm", since we directly compute the endpoints of $g(I)$. The details when g is a cubic polynomial are derived and implemented in our previous paper [9, Appendix]. How far can we extend this idea? Under the common understanding (3), we need two other ingredients:

(E1) The function $g(x)$ must be exactly computable.
(E2) The roots of $g'(x)$ must be exactly computable.

Note that (E1) is relatively easy to fulfill. For instance, $g(x)$ can be any polynomial. However, (E2) limits g to polynomials of degree at most 5, since the roots of g' are guaranteed to have closed form expressions when g' has degree at most 4. Cornelius & Lohner appear to have this endpoint algorithm in mind when they stated in [3, p. 340, Remark 2] that their framework may reach up to order 6 convergence, namely one more than the degree of g.

[4] The appearance of sqr may be curious, but that is because he will later define interval forms of the operations in Ω.

1.2 Overview

In Section 2, we present our generalized CL framework for achieving range functions with any order of convergence. In Section 3, we provide details for a new family of *recursive range operators*[5] $\{\square^H_{4,\ell} : \ell = 0, 1, \ldots\}$ with quartic convergence order and recursion level $\ell \geq 0$, based on Hermite interpolation. In Section 4, we present our "holistic" framework for evaluating the complexity of range functions. The idea is to amortize the cost over an entire computation tree. Experimental results are in Section 5. They show that in the context of the EVAL algorithm, $\square^H_4$ is superior to our previous favorite $\square^{L'}_3$. The theoretical model of Section 4 is also confirmed by these experiments. Another set of experiments explore the possible speed improvements by non-maximal convergence levels. We conclude in Section 6.

1.3 Terminology and notation

This section reviews and fixes some terminology. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a real function for some $n \geq 0$. The *arity* of f is n. We identify 0-arity functions with real constants. In this paper, we do not assume that real functions are total functions. If f is undefined at $\boldsymbol{x} \in \mathbb{R}^n$, we write $f(\boldsymbol{x}) \uparrow$; otherwise $f(\boldsymbol{x}) \downarrow$. If any component of $\boldsymbol{x}$ is undefined, we also have $f(\boldsymbol{x}) \uparrow$. Define the *proper domain* of f as $\mathtt{dom}(f) := \{\boldsymbol{x} \in \mathbb{R}^m : f(\boldsymbol{x}) \downarrow\}$. If $S \subseteq \mathbb{R}^m$, then $f(S) \uparrow$ if f is undefined at some $\boldsymbol{x} \in S$; else $f(S) := \{f(\boldsymbol{x}) : \boldsymbol{x} \in S\}$. Define the *magnitude* of $S \subseteq \mathbb{R}$ as $|S| := \max\{|x| : x \in S\}$. Note that we use bold font like $\boldsymbol{x}$ to indicate vector variables.

The set of compact boxes in $\mathbb{R}^n$ is denoted $\square\mathbb{R}^n$; if $n = 1$, we simply write $\square\mathbb{R}$. The Hausdorff distance on boxes $B, B' \in \square\mathbb{R}^n$ is denoted $d_H(B, B')$. For $n = 1$, it is often denoted $q(I, J)$ in the interval literature. A *box form* of f is any function $F : \square\mathbb{R}^n \to \square\mathbb{R}$. satisfying two properties: (1) *conservative*: $f(B) \subseteq F(B)$ for all $B \in \square\mathbb{R}^n$. (2) *convergent*: for any sequence $(B_i)_{i=0}^{\infty}$ of boxes converging to a point, $\lim_{i\to\infty} F(B_i) = f(\lim_{i\to\infty} B_i)$. In general, we indicate box forms by a prefix meta-symbol "$\square$". Thus, instead of F, we write "$\square f$" for any box form of f. We annotate $\square$ with subscripts and/or superscripts to indicate specific box forms. E.g., $\square_i f$ or $\square^L f$ or $\square^L_i f$ are all box forms for f. In this paper, we mostly focus on $n = 1$. A *subdivision tree* is a finite tree T whose nodes are intervals satisfying this property: if interval $[a, b]$ is a non-leaf node of T, then it has two children represented by the intervals $[a, m]$ and $[m, b]$. If I_0 is the root of T, we call the set $\mathcal{D} = \mathcal{D}(T)$ of leaves of T a *subdivision* of I_0.

For Turing computability of our box functions, we replace real intervals $\square\mathbb{R}$ by *dyadic intervals* $\square\mathbb{D}$, i.e., intervals whose endpoints are dyadic numbers which are elements of $\mathbb{D} = \{m2^n : m, n \in \mathbb{Z}\}$. A real function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be *computable* if its restriction to dyadic inputs will produce dyadic outputs,

[5] Each $\square^H_{4,\ell}$ is an operator that transforms any sufficiently smooth function $f : \mathbb{R} \to \mathbb{R}$ into the range function $\square^H_{4,\ell} f$ for f.

and this restriction is Turing computable. This notion extends to box functions, $\square f : \square\mathbb{R}^n \to \square\mathbb{R}$.

Let $\boldsymbol{u} = (u_0, \ldots, u_m)$ denote a sequence of $m+1$ distinct points, where the u_i's are called *nodes*. Let $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_m)$, where each $\mu_i \geq 1$ is called a *multiplicity*. The *Hermite interpolant* of f at $\boldsymbol{u}, \boldsymbol{\mu}$ is a polynomial $h_f(x) = h_f(x; \boldsymbol{u}, \boldsymbol{\mu})$ such that ${h_f}^{(j)}(u_i) = f^{(j)}(u_i)$ for all $i = 0, \ldots, m$ and $j = 0, \ldots, \mu_i - 1$. The interpolant $h_f(x)$ is unique and has degree less than $d = \sum_{i=0}^{m} \mu_i$. If $m = 0$, then $h_f(x)$ is the Taylor interpolant; if $\mu_i = 1$ for all i, then $h_f(x)$ is the Lagrange interpolant.

2 Generalized CL Framework

In this section, we develop an approach to computing range functions of arbitrary convergence order. To avoid the exact range computation, we replace $g(I)$ in (1) by a range function $\square g(I)$ for g:

$$\square f(I) := \square g(I) + \square R_g(I) \tag{5}$$

We now generalize Theorem A as follows.

Theorem B. *With $\square f(I)$ defined as in (5), we have*

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

Proof. Consider the endpoints of the intervals $f(I), g(I)$ and $\square R_g(I)$ as given by

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

for some $\underline{x}, \overline{x}, \underline{y}, \overline{y} \in I$ and a, b. We can also write

$$\square g(I) = [g(\underline{y}), g(\overline{y})] + [\underline{\varepsilon}, \overline{\varepsilon}]$$

for some $\underline{\varepsilon} \leq 0 \leq \overline{\varepsilon}$. Thus we have

$$\begin{aligned} d_H(g(I), \square g(I)) &= \max\{-\underline{\varepsilon}, \overline{\varepsilon}\}, \\ \square f(I) &= [g(\underline{y}), g(\overline{y})] + [\underline{\varepsilon}, \overline{\varepsilon}] + [a, b]. \end{aligned} \tag{6}$$

We write the inclusion $f(I) \subseteq \square f(I)$ in terms of endpoints:

$$[f(\underline{x}), f(\overline{x})] \subseteq [g(\underline{y}), g(\overline{y})] + [\underline{\varepsilon}, \overline{\varepsilon}] + [a, b].$$

Hence

$$d_H(f(I), \square f(I)) = \max\big\{f(\underline{x}) - \big(g(\underline{y}) + \underline{\varepsilon} + a\big), \big(g(\overline{y}) + \overline{\varepsilon} + b\big) - f(\overline{x})\big\}$$

Since $w(\square R_g(I)) = b - a$ and in view of (6), our theorem follows from

$$f(\underline{x}) - \big(g(\underline{y}) + \underline{\varepsilon} + a\big) \leq -\underline{\varepsilon} + (b - a), \tag{7}$$

$$\big(g(\overline{y}) + \overline{\varepsilon} + b\big) - f(\overline{x}) \leq \overline{\varepsilon} + (b - a). \tag{8}$$

To show (7), we have, since $f(\underline{x}) \le f(\underline{y})$,

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

The proof for (8) is similar.

2.1 Achieving any order of convergence

To apply Theorem B, we introduce *precision-bounded range functions* for $g(x)$ denoted $\square g(I;\varepsilon)$, where $\varepsilon > 0$ is an extra "precision" parameter. The output interval is an *outer ε-approximation* in the sense that $g(I) \subseteq \square g(I;\varepsilon)$ and

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

We also call $\square g(I;\varepsilon)$ a *strong box function*, since it implies box forms in the original sense: E.g., a box form of g may be constructed as follows:

$$\square g(I) := \square g(I, w(I)). \tag{9}$$

The box form in (9) has the pleasing property that $w(I)$ is an implicit precision parameter.

Returning to the CL Framework, suppose $f = g + R_g$ where g has a strong range function $\square g(I;\varepsilon)$. We now define the following box form of f:

$$\square_{\mathrm{pb}} f(I) := \square g(I;\varepsilon) + \square R_g(I), \tag{10}$$

where $\varepsilon = |\square R_g(I)|$. The subscript in $\square_{\mathrm{pb}}$ refers to "precision-bound". To compute $\square_{\mathrm{pb}} f(I)$, we first compute $J_R \leftarrow \square R_g(I)$, then compute $J_g \leftarrow \square g(I, |J_R|)$, and finally return $J_g + J_R$.

Corollary 1. *The box form $\square_{pb} f(I)$ of (10) has the same convergence order as $\square R_g(I)$.*

For any $n \ge 1$, if $g(x)$ is a Hermite interpolant of f of degree n, then $\square R_g(I)$ has convergence order $n+1$ (cf. Example 1.1). We have thus achieved arbitrary convergence order.

Remark 1. Theorem B is also needed to justify the usual implementations of "exact $g(I)$" under the hypothesis (3) of the CL framework. Given an expression $E(x)$ for $g(x)$, it suffices to evaluate it with error at most $|\square R_g(I)|$. This can be automatically accomplished in the Core Library using the technique of "precision-driven evaluation" [21, §2].

2.2 Strong box functions

Corollary 1 shows that the "exact computation of $g(I)$" hypothesis of the CL framework can be replaced by strong box functions of g. We now address the construction of such functions. We proceed in three stages:

A. Lipschitz Expressions. Our starting point is the theory of evaluations of expressions over a class Ω of Lipschitz functions, following [11]. Let Ω denote a set of continuous real functions that includes $\mathbb{R}$ as constant functions as well as the rational operations. Elements of Ω are called *primitive functions*. Let $\mathrm{Expr}(\Omega)$ denote the set of expressions over $\Omega \cup X$ where $X = \{X_1, X_2, \ldots\}$ is a countable set of variables. An expression $E \in \mathrm{Expr}(\Omega)$ is an ordered DAG (directed acyclic graph) whose nodes with outdegree $m \geq 0$ are labeled by m-ary functions of Ω, with variables in X viewed as 0-ary. For simplicity, assume E has a unique root (in-degree 0). Any node of E induces a subexpression. If E involves only the variables in $\boldsymbol{X} = (X_1, \ldots, X_n)$, we may write $E(\boldsymbol{X})$ for E. We can evaluate E at $\boldsymbol{a} \in \mathbb{R}^n$ by substituting $\boldsymbol{X} \leftarrow \boldsymbol{a}$ and evaluating the functions at each node in a bottom-up fashion. The value at the root is $E(\boldsymbol{a})$ and may be undefined. If $f : \mathbb{R}^n \to \mathbb{R}$ is a function, we call E an *expression for* f if the symmetric difference $\mathtt{dom}(E) \Delta \mathtt{dom}(f)$ is a finite set. E.g., if $f(x) = \sum_{i=0}^{n-1} x^i$, then $E(X_1) = \frac{X_1^n - 1}{X_1 - 1}$ is an expression for f since $f(a) = E(a)$ for $a \neq 1$, but $f(1) = n$ and $E(1) \uparrow$. Similarly, we can define the *interval value* $E(B)$ at the box $B = (I_1, \ldots, I_n) \in \square\mathbb{R}^n$. If each f in E is replaced by a box form $\square f$, we obtain a *box expression* $\square E(\boldsymbol{X})$.

Following [11, pp. 33, 74], we say that $E(\boldsymbol{X})$ is *Lipschitz* at $B \in \square\mathbb{R}^n$ if the following properties hold inductively:

- (Base case) The root of E is labeled by a variable X_i or a constant function. This always holds.
- (Induction) Let $E = f(E_1, \ldots, E_m)$, where each E_j is a subexpression of E. Inductively, each E_i is Lipschitz at B. Moreover, $f(E_1(B), \ldots, E_m(B))$ is defined and f is Lipschitz[6] in a neighborhood U of $(E_1(B), \ldots, E_m(B)) \subseteq \square\mathbb{R}^m$.

Theorem C [11, p. 34]. *If $E(\boldsymbol{x})$ is a Lipschitz expression on $B_0 \in \square\mathbb{R}^n$, then there is a vector $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_n)$ of positive constants such that for all $B, B' \subseteq B_0$,*

$$d_H(E(B), E(B')) \leq \boldsymbol{\ell} * d_H(B, B'),$$

where $d_H(B, B') = (d_H(I_1, I'_1), \ldots, d_H(I_n, I'_n))$ and $*$ is the dot product.

Theorem C can be extended to the box form $\square E(\boldsymbol{X})$, and thus $\square E(B)$ is an enclosure of $E(B)$. To achieve strong box functions, we will next strengthen Theorem C to compute explicit Lipschitz constants.

[6] The concept of a function f (not expression) being Lipschitz on a set U is standard: it means that there exists a vector $\boldsymbol{\ell} = (\ell_1, \ldots, \ell_m)$ of positive constants, such that for all $\boldsymbol{x}, \boldsymbol{y} \in U \subseteq \mathbb{R}^m$, $|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \boldsymbol{\ell} * |\boldsymbol{x} - \boldsymbol{y}|$ where $*$ is the dot product and $|\boldsymbol{x} - \boldsymbol{y}| = (|x_1 - y_1|, \ldots, |x_m - y_m|)$. Call $\boldsymbol{\ell}$ a *Lipschitz constant vector* for U.

B. Lipschitz$^+$ Expressions. For systematic development, it is best to begin with an *abstract model* of computation that assumes $f(B)$ and $\partial_i f(B)$ are computable. Eventually, we replace these by $\square f(B)$ and $\square\partial_i f(B)$, and finally we make them Turing computable by using dyadic approximations to reals. This follows the "AIE methodology" of [19]. Because of our limited space and scope, we focus on the abstract model.

Call Ω a *Lipschitz$^+$ class* if each $f \in \Omega$ is a Lipschitz$^+$ function in this sense that f has continuous partial derivatives at its proper domain $\texttt{dom}(f)$ and both f and its gradient $\nabla f = (\partial_1 f, \ldots, \partial_m f)$ are locally Lipschitz. Given an expression $E(\boldsymbol{X})$ over Ω, we can define $\nabla E := (\partial_1 E, \ldots, \partial_n E)$, where each $\partial_i E(\boldsymbol{X})$ is an expression, defined inductively as

$$\partial_i E(\boldsymbol{X}) = \begin{cases} 0 & \text{if } E = \text{const}, \\ \delta(i = j) & \text{if } E = X_j, \\ \sum_{j=1}^{m} (\partial_j f)(E_1, \ldots, E_m) \cdot \partial_i E_j & \text{if } E = f(E_1, \ldots, E_m). \end{cases}$$

Here, $\delta(i = j) \in \{0, 1\}$ is Kronecker's delta function that is 1 if and only if $i = j$.

C. Strong Box Evaluation Let $f : \mathbb{R}^n \to \mathbb{R}$ be a Lipschitz$^+$ function. Suppose it has a *strong approximation function* $\widetilde{f}$, i.e.,

$$\widetilde{f} : \mathbb{R}^n \times \mathbb{R}_{>0} \to \mathbb{R}, \tag{11}$$

such that $|\widetilde{f}(\boldsymbol{a}; \varepsilon) - f(\boldsymbol{a})| \leq \varepsilon$. We show that f has a strong box function. Define $\Delta(f, B) := \frac{1}{2} \sum_{i=1}^{n} |\partial_i f(B)| \cdot w_i(B)$. Then, for all $\boldsymbol{a} \in B$, we have

$$|f(\boldsymbol{a}) - f(\boldsymbol{m}(B))| \leq \Delta(f, B). \tag{12}$$

by the Mean Value Theorem.

Lemma 1. *Suppose $\Delta(f, B) \leq \varepsilon/4$. If*

$$J := [\widetilde{f}(\boldsymbol{m}(B); \varepsilon/4) \pm \tfrac{1}{2}\varepsilon], \tag{13}$$

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

Motivated by Lemma 1, we say that a subdivision $\mathcal{D}$ of B_0 is *ε-fine* if $\Delta(f, B) \leq \varepsilon/4$ for each $B \in \mathcal{D}$. Given an ε-fine subdivision $\mathcal{D}$ of B_0, let $J(\mathcal{D}) := \bigcup_{B \in \mathcal{D}} J(B)$, where $J(B)$ is defined in (13).

Corollary 2. *If $\mathcal{D}$ is an ε-fine subdivision of B_0, then $f(B_0) \subseteq J(\mathcal{D})$ and $d_H(f(B_0), J(\mathcal{D})) < \varepsilon$.*

Algorithm 1 shows how to compute an ε-fine subdivision of any given B_0.

Note that the value of $\Delta(f, B)$ is reduced by a factor less than or equal to $(1 - \frac{1}{2n})$ with each bisection, and therefore the subdivision depth is at most $\ln(\varepsilon/\Delta(f, B_0))/\ln(1 - \frac{1}{2n})$. This bound is probably overly pessimistic (e.g., $|J_i| = |\partial_i f(B)|$ is also shrinking with depth). We plan to do an amortized bound of this algorithm. In any case, we are now able to state the key result.

Algorithm 1 Fine Subdivision Algorithm

```
Input: (f, B_0, ε)
Output: An ε-fine subdivision D of B_0.
Let Q, D be queues of boxes, initialized as D ← ∅ and Q ← {B_0}.
while Q ≠ ∅ do
    B ← Q.pop()
    (J_1, ..., J_n) ← ∇f(B)
    Δ(f, B) ← Σ_{i=1}^{n} |J_i| · w_i(B)
    if Δ(f, B) ≤ ε/4 then
        D.push(B)
    else
        i* ← argmax_{i=1,...,n} |J_i| · w_i(B)
        Q.push(bisect(B, i*))                    ▷ Bisect dimension i*
Output D
```

Theorem D. *Let Ω be a Lipschitz$^+$ class, where each $f \in \Omega$ has a strong approximation function as in (11). If $E(\boldsymbol{X})$ is an expression that is Lipschitz$^+$ over $B \in \square\mathbb{R}^n$, then the strong interval function $E(B;\varepsilon)$ is abstractly computable.*

Proof (Proof sketch). Use induction on the structure of $E(\boldsymbol{X})$. The base case is trivial. If $E(\boldsymbol{X}) = f(E_1, \ldots, E_m)$, then, by induction, $\widetilde{I}_i = E_i(B;\varepsilon_i)$ is abstractly computable ($i = 1, \ldots, m$). Lemma 1 can be generalized to allow the evaluation of $f(\widetilde{B};\varepsilon)$, where $\widetilde{B} = (\widetilde{I}_1, \ldots, \widetilde{I}_m)$.

Which functions satisfy the requirements of Theorem D? One of the most extensive classes with strong approximation algorithms is the class of hypergeometric functions; Johansson [10] describes a state-of-the-art library for such functions. In [4,5], we focused on the real hypergeometric functions and provide a uniform algorithm with complexity analysis for rational input parameters. Note that Theorem D also needs strong box functions which were not treated in [5,10]; such extensions could be achieved because hypergeometric functions are closed under differentiation. A complete account of the preceding theory must replace the abstract computational model by box functions $\square f$, finally giving dyadic approximations $\widetilde{\square} f$ following the AIE methodology in [19].

3 A Practical Range Function of Order 4

In this section, we consider a new recursive range function based on Hermite interpolation which will surpass the performance of $\square_3^{L'} f$ [9]. Let h_0 be the Hermite interpolant of f based on the values and first derivatives at the endpoints of the interval $I = [a, b]$, that is, h_0 is the unique cubic polynomial with

$$h_0(a) = f(a), \quad h_0'(a) = f'(a), \quad h_0(b) = f(b), \quad h_0'(b) = f'(b).$$

With $m = (a + b)/2$ denoting the midpoint of I, it is not hard to show that h_0 can be expressed in centered form as

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

with coefficients

$$c_{0,0} = \frac{f(a)+f(b)}{2} - \frac{f'(b)-f'(a)}{4} r, \qquad c_{0,1} = 3\frac{f(b)-f(a)}{4r} - \frac{f'(a)+f'(b)}{4},$$
$$c_{0,2} = \frac{f'(b)-f'(a)}{4r}, \qquad c_{0,3} = \frac{f'(a)+f'(b)}{4r^2} - \frac{f(b)-f(a)}{4r^3},$$

where $r = (b-a)/2$ is the radius of I. Since the remainder $R_{h_0} = f - h_0$ can be written as

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

for some $\xi_x \in I$, we can upper bound the magnitude of $R_{h_0}(I)$ as

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

To further upper bound $|f^{(4)}(I)|$, following [9, Sec. 3], we consider the cubic Hermite interpolants h_j of $f^{(4j)}$ for $j = 1, 2, \ldots, \ell$:

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

with coefficients

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

Denoting the remainder by $R_{h_j} = f^{(4j)} - h_j$ and using the same arguments as above, we have

$$|f^{(4j)}(I)| \leq |h_j(I)| + |R_{h_j}(I)| \leq |h_j(I)| + \Omega |f^{(4j+4)}(I)|. \tag{14}$$

By repeated application of (14), we get

$$\begin{aligned}
|f^{(4)}| &\leq |h_1(I)| + \Omega |f^{(8)}(I)| \\
&\leq |h_1(I)| + \Omega\big(|h_2(I)| + \Omega |f^{(12)}(I)|\big) \leq \cdots \\
&\leq \sum_{j=1}^{\ell} |h_j(I)| \Omega^{j-1} + \Omega^{\ell} |f^{(4\ell+4)}(I)|,
\end{aligned} \tag{15}$$

resulting in the recursive remainder bound

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

Overall, we get the *recursive Hermite form* of order 4 and *recursion level* $\ell \geq 0$,

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

which depends on the $4\ell + 4$ values

$$f^{(4j)}(a), \quad f^{(4j+1)}(a), \quad f^{(4j)}(b), \quad f^{(4j+1)}(b), \quad (j = 0, \ldots, \ell). \tag{16}$$

If f is analytic and r is sufficiently small, or if f is a polynomial, then S_∞ is a convergent series and we define $\square^H_4 f(I) := \square^H_{4,\infty} f(I)$ as the *maximal* recursive Hermite form. Clearly, if f is a polynomial of degree at most $d-1$, then $\square^H_4 f = \square^H_{4,\ell} f$ for $\ell = \lceil d/4 \rceil - 1$.

To avoid the rather expensive evaluation of the exact ranges $h_j(I)$, $j = 1, \ldots, \ell$, we can use the classical Taylor form for approximating them, resulting in the cheaper but slightly less tight range function

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

where

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

In case we also have to estimate the range of f', we can compute the $2\ell + 2$ additional values

$$f^{(4j+2)}(a), \quad f^{(4j+2)}(b), \qquad j = 0, \ldots, \ell \tag{17}$$

and apply $\square^H_{4,\ell}$ to f'. But we prefer to avoid (17) by re-using the data used for computing $\square^H_{4,\ell} f(I)$ in the following way. A result by Shadrin [15] asserts that the error between the first derivative of f and the first derivative of the Lagrange polynomial $L(x)$ that interpolates f at the 4 nodes $x_0, \ldots, x_3 \in I$ satisfies

$$|f'(x) - L'(x)| \leq \frac{|\omega'_L(I)|}{4!}|f^{(4)}(I)|, \qquad x \in I,$$

for $\omega_L(x) = \prod_{i=0}^{3}(x - x_i)$. As noted by Waldron [18, Addendum], this bound is continuous in the x_i, and so we can consider the limit as x_0 and x_1 approach a and x_2 and x_3 approach b to get the corresponding bound for the error between f' and the first derivative of the Hermite interpolant h_0,

$$|f'(x) - h'_0(x)| \leq \frac{|\omega'(I)|}{4!}|f^{(4)}(I)|, \qquad x \in I.$$

Since a straightforward calculation gives $\omega'(I) = \frac{8}{9}\sqrt{3}r^3[-1,1]$, we conclude by (15) that

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

resulting in the recursive Hermite forms

$$\square^{H}_{3,\ell}f'(I) = h'_0(I) + \frac{8\sqrt{3}}{9r}[-1,1]S_\ell \qquad \text{and} \qquad \square^{H'}_{3,\ell}f'(I) = h'_0(I) + \frac{8\sqrt{3}}{9r}[-1,1]S'_\ell,$$

which have only cubic convergence, but depend on the same data as $\square^{H}_{4,\ell}f(I)$ and $\square^{H'}_{4,\ell}f(I)$.

4 Holistic Complexity Analysis of Range Functions

By the "holistic complexity analysis" of $\square f(I)$, we mean to analyze its cost over a subdivision tree, not just its cost at a single isolated interval. The cost for a node of the subdivision tree might be shared with its ancestors, descendants, or siblings, leading to cheaper cost per node. Although we have the EVAL algorithm [9, Section 1.2] in mind, there are many applications where the algorithms produce similar subdivision trees, even in higher dimensions.

4.1 Amortized complexity of $\square^{L'}_3 f$

We first focus on the range function denoted $\square^{L'}_3 f$ in [9]. This was our "function of choice" among the 8 range functions studied in [9, Table 1]. Empirically, we saw that $\square^{L'}_3$ has at least a factor of 3 speedup over $\square^{T}_2$. Note that $\square^{T}_2$ was the state-of-the-art range function before our recursive forms; see the last column of the Tables 3 and 4 in [9]. We now show theoretically that the speedup is also 3 if we only consider evaluation complexity. The data actually suggest an asymptotic speedup of at least 3.5 – this may be explained by the fact that $\square^{L'}_3$ has order 3 convergence compared to order 2 for $\square^{T}_2$. We now seek a theoretical account of the observed speedup[7].

In the following, let $d \geq 2$. Given any f and interval $[a, b]$, our general goal is to construct a range function $\square f([a, b])$ based on d derivatives of f at points in $[a, b]$. In the case of $\square^{L'}_3 f([a, b])$, we need these evaluations of f and its higher derivatives:

$$f^{(3j)}(a), \quad f^{(3j)}(m), \quad f^{(3j)}(b), \qquad j = 0, \ldots, \lceil d/3 \rceil - 1,$$

where $m = (a + b)/2$. That is a total of $3\lceil d/3 \rceil$ derivative values. For simplicity, assume d is divisible by 3. Then the *cost* for computing $\square^{L'}_3 f([a, b])$ is $3\lceil d/3 \rceil = d$. Note that the cost to compute $\square^{T}_2 f(I)$, the maximal Taylor form of order 2, is also d. So there is no difference between these two costs over isolated intervals. But in a "holistic context", we see a distinct advantage of $\square^{L'}_3$ over $\square^{T}_2$: the

[7] Note that in our EVAL application, we must simultaneously evaluate $\square^{L'}_3 f(I)$ as well as its derivative $\square^{L'}_2 f'(I)$. But it turns out that we can bound the range of f' for no additional evaluation cost.

evaluation of $\square_3^{L'} f(I)$ can reuse the derivative values already computed at the parent or sibling of I; no similar reuse is available to $\square_2^T$.

Given a subdivision tree T, our goal is to bound the *cost* $C_3^L(T)$ of $\square_3^{L'} f$ on T, i.e., the total number of derivative values needed to compute $\square_3^{L'} f(I)$ for all $I \in T$. We will write $C_3^L(n)$ instead of $C_3^L(T)$ when T has n leaves. This is because it is n rather than the actual[8] shape of T that is determinative for the complexity. We have the following recurrence:

$$C_3^L(n) = \begin{cases} d & \text{if } n = 1, \\ C_3^L(n_L) + C_3^L(n_R) - \frac{d}{3} & \text{if } n \geq 2, \end{cases} \tag{18}$$

where the left and right subtrees of the root have n_L and n_R leaves, respectively. Thus $n = n_L + n_R$. Let the intervals I, I_L, I_R denote the root and its left and right children. The formula for $n \geq 2$ in (18) comes from summing three costs: (1) the cost d at the root I; (2) the cost $C_3^L(n_L)$ but subtracting $2d/3$ for derivatives shared with I; (3) the cost $C_3^L(n_R) - 2d/3$ attributed to the right subtree.

Theorem 1. *(Amortized Complexity of $\square_3^{L'}$)*

$$C_3^L(n) = (2n+1) \cdot \tfrac{d}{3}. \tag{19}$$

Thus the cost per node is $\sim d/3$ asymptotically.

Proof. The solution (19) is easily shown by induction using the recurrence (18). To obtain the cost per node, we recall that a full binary tree with n leaves has $2n-1$ nodes. So the average cost per node is $\frac{2n+1}{2n-1} \cdot \frac{d}{3} \sim d/3$.

This factor of 3 improvement over $\square_2^T$ is close to our empirical data in [9].

4.2 Amortized complexity of $\square_4^H f$

We do a similar holistic complexity analysis for the recursive range function $\square_{4,\ell}^H f(I)$ from Section 3 for any given f and $\ell \geq 0$. According to (16), our recursive scheme requires the evaluation of $4(\ell+1)$ derivatives of f at the two ends of I. Let $d = 4(\ell+1)$, so that computing $\square_{4,\ell}^H f(I)$ costs d derivative evaluations. For holistic analysis, let $C_4^H(n)$ denote the cost of computing $\square_{4,\ell}^H f(I)$ on a subdivision tree with n leaves. We then have the recurrence

$$C_4^H(n) = \begin{cases} d & \text{if } n = 1, \\ C_4^H(n_L) + C_4^H(n_R) - \frac{d}{2} & \text{if } n \geq 2, \end{cases} \tag{20}$$

where $n_L + n_R = n$. The justification of (20) is similar to (18), with the slight difference that the midpoint of an interval J is not evaluated and hence not shared with the children of J.

[8] If d is not divisible by 3, we can ensure a total cost of d evaluations per interval of the tree but the tree shape will dictate how to distribute these evaluations on the $m+1$ nodes.

Theorem 2. *(Amortized Complexity of $\square_4^H$)*

$$C_4^H(n) = (n+1) \cdot \tfrac{d}{2}. \tag{21}$$

Thus the cost per node is $\sim d/4$ asymptotically.

Proof. The solution (21) follows from (20) by induction on n. Since a full binary tree with n leaves has $2n-1$ nodes, the average cost per node is $\frac{n+1}{2n-1}\frac{d}{2} \sim d/4$.

Thus we expect a 4-fold speedup of $\square_4^H$ when compared to the state-of-art $\square_2^T$. Or a 4/3-fold or 33% speedup when compared to $\square_3^{L'}$. This agrees with our empirical data below.

4.3 Amortized complexity for Hermite schemes

We now generalize the analysis above. Recall from Section 1.3 that $h_f(x) = h_f(x; \boldsymbol{u}, \boldsymbol{\mu})$ is the Hermite interpolant of f with node sequence $\boldsymbol{u} = (u_0, \ldots, u_m)$ and multiplicity $\boldsymbol{\mu} = (\mu_0, \ldots, \mu_m)$. We fix the function $f : \mathbb{R} \to \mathbb{R}$. Assume $m \geq 1$ and the nodes are equally spaced over the interval $I = [u_0, u_m]$, and all μ_i are equal to $h \geq 1$. Then we can simply write $h(x; I)$ for the interpolant on interval I. Note that $h(x; I)$ has degree less than $d := (m+1)h$.

Our cost model for computing $\square f(I)$ is the number of evaluations of derivatives of f at the nodes of I. Based on our recursive scheme, this cost is exactly $d = (m+1)h$ since I has $m+1$ nodes. To amortize this cost over the entire subdivision tree T, define $N_m(T)$ to be the number of distinct nodes among all the intervals of T. In other words, if intervals I and J share a node u, then we do not double count u. This can happen only if I and J have an ancestor-descendant relationship or are siblings. Let T_n denote a tree with n leaves. It turns out that $N_m(T_n)$ is a function of n, independent of the shape of T_n. So we simply write $N_m(n)$ for $N_m(T_n)$. Therefore[9] the *cost of evaluating the tree T_n* is

$$C_d^h(n) := h \cdot N_m(n), \qquad \text{where } d = (m+1)h.$$

Since T_n has $2n-1$ intervals, we define the *amortized cost* of a recursive Hermite range function as

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

Theorem 3.

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

[9] The notation "$C_d^h(n)$" does not fully reproduce the previous notations of $C_3^L(n)$ and $C_4^H(n)$ (which were chosen to be consistent with $\square_3^{L'}$ and $\square_4^H$). Also, d is implicit in the previous notations.

Proof. We claim that $N_m(n)$ satisfies the recurrence

$$N_m(n) = \begin{cases} m+1 & \text{if } n = 1, \\ N_m(n_L) + N_m(n_R) - 1 & \text{if } 1 < n = n_L + n_R. \end{cases} \tag{22}$$

The base case is clear, so consider the inductive case: the left and right subtrees of T_n are T_{n_L} and T_{n_R}, where $n = n_L + n_R$. Then nodes at the root of T_n are already in the nodes at the roots of T_{n_L} and T_{n_R}. Moreover, the roots of T_{n_L} and T_{n_R} share exactly one node. This justifies (22). The solution $N_m(n) = mn+1$ is immediate. The amortized cost is $\lim_{n\to\infty} C_d^h(n)/(2n-1)$ since the tree T_n has $2n-1$ intervals.

Remark 2. Observe that the amortized complexity $\overline{C}_d^h = \frac{d-h}{2}$ is strictly less than d, the non-amortized cost. For any given d, we want h as large as possible, but h is constrained to divide d. Hence for $d = 4$, we choose $h = 2$. We can also generalize to allow multiplicities $\boldsymbol{\mu}$ to vary over nodes: e.g., for $d = 5$, $\boldsymbol{\mu} = (2, 1, 2)$.

Remark 3. The analysis of $C_3^L(n)$ and $C_4^H(n)$ appears to depend on whether m is odd or even. Surprisingly, we avoided such considerations in the above proof.

5 Experimental Results

To provide a holistic application for evaluating range functions, we use Eval, a simple root isolation algorithm. Despite its simplicity, Eval produces near-optimal subdivision trees [1,16] when we use $\square_2^T f$ for real functions with simple roots; see [9, Sections 1.2, 1.3] for its description and history. We now implemented a version of Eval in *C++* for range functions that may use any recursion level (unlike [9] which focused on maximal levels). We measured the size of the Eval subdivision tree as well as the average running time of Eval with floating point and rational arithmetic on various classes of polynomials. These polynomials have varying root structures: dense with all roots real (Chebyshev T_n, Hermite H_n, and Wilkinson's W_n), dense with only 2 real roots (Mignotte cluster M_{2k+1}), and sparse without real roots (S_n). Depending on the family of polynomials, we provide different centered intervals $I_0 = [-r(I_0), r(I_0)]$ for Eval to search in, but always such that *all* real roots are contained in I_0. Our experimental platform is a Windows 10 laptop with a 1.8 GHz Intel Core i7-8550U processor and 16 GB of RAM. We use two kinds of computer arithmetic in our testing: 1024-bit floating point arithmetic and multi-precision rational arithmetic. In rational arithmetic, $\sqrt{3}$ is replaced by the slightly larger $17320508075688773 \times 10^{-16}$. Our implementation, including data and Makefile experiments, may be downloaded from the Core Library webpage [2].

We tested eleven versions of Eval that differ by the range functions used for approximating the ranges of f and f'; see Tables 1–3. The first three, E_2^T, $\mathrm{E}_3^{L'}$, $\mathrm{E}_4^{L'}$, are the state-of-the-art performers from [9], followed by three non-maximal variants of $\mathrm{E}_3^{L'}$, namely $\mathrm{E}_{3,\ell}^{L'}$ for $\ell \in \{10, 15, 20\}$. The next two, E_4^H and $\mathrm{E}_4^{H'}$,

Table 1. Size of the Eval subdivision tree. Here, Eval is searching for roots in $I_0 = [-r(I_0), r(I_0)]$.

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	$\underline{231}$	243	243	243	239	239	239	239	239
T_{40}		663	479	$\underline{463}$	479	479	479	471	479	479	479	479
T_{80}	10	1379	1007	$\underline{955}$	1023	1007	1007	967	991	991	991	991
T_{160}		2147	1427	$\underline{1347}$	1543	1451	1427	1351	1359	1439	1363	1359
T_{320}		-	2679	$\underline{2575}$	3023	2699	2679	2591	2591	2803	2603	2591
H_{20}		283	215	207	215	215	215	$\underline{199}$	207	207	207	207
H_{40}		539	423	$\underline{415}$	423	423	423	415	419	419	419	419
H_{80}	40	891	679	$\underline{655}$	711	679	679	659	683	695	683	683
H_{160}		1435	955	$\underline{923}$	1083	959	955	923	927	1023	927	927
H_{320}		-	2459	$\underline{2415}$	45287	10423	4419	2455	2499	15967	5195	3119
M_{21}		169	113	109	113	113	113	$\underline{105}$	105	105	105	105
M_{41}	1	339	215	$\underline{213}$	215	215	215	219	223	223	223	223
M_{81}		683	445	$\underline{423}$	507	445	445	427	431	443	431	431
M_{161}		-	905	$\underline{857}$	7245	1755	1047	861	861	2663	1079	905
W_{20}		485	353	$\underline{331}$	353	353	353	331	335	335	335	335
W_{40}	1000	901	633	$\underline{613}$	633	633	633	615	617	617	617	617
W_{80}		1583	1133	$\underline{1083}$	2597	1133	1133	1097	1117	1485	1117	1117
W_{160}		-	2005	$\underline{1935}$	293509	5073	2005	1959	1993	42413	5289	2817
S_{100}		973	633	609	611	621	625	613	613	$\underline{595}$	609	613
S_{200}	10	1941	1281	1221	1211	1227	1237	1231	1231	$\underline{1165}$	1187	1201
S_{400}		-	2555	2435	2379	2399	2413	2467	2467	$\underline{2289}$	2319	2339

are based on the maximal recursive Hermite forms $\square_4^H f$ and $\square_3^H f'$ and their cheaper variants $\square_4^{H'} f$ and $\square_3^{H'} f'$, respectively, and the last three derive from the non-maximal variants of the latter, again for recursion levels $\ell \in \{10, 15, 20\}$.

Table 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. It requires future research to investigate the latter. We further observe that the differences between the tree sizes for $E_4^{L'}$ and $E_4^{H'}$ 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 [9], a smaller tree size does not necessarily correspond to a faster running time. In fact, $E_3^{L'}$ was found to usually be almost as fast as $E_4^{L'}$, even though the subdivision trees of $E_3^{L'}$ are consistently bigger than those of $E_4^{L'}$.

In Tables 2 and 3 we report the running times for our eleven Eval versions and the different families of polynomials. Times are given in seconds and averaged over at least four runs (and many more for small degree polynomials). The last three columns in both tables report the speedup ratios $\sigma(\cdot)$ of $E_4^{H'}$, $E_{4,15}^{H'}$, and $E_{3,15}^{L'}$ with respect to $E_3^{L'}$, which was identified as the overall winner in [9].

In Figure 1, we provide a direct comparison of the Eval version based on our new range function $E_4^{H'}$ with the previous leader $E_3^{L'}$: 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 1). Hence, $E_4^{H'}$ emerges as the new winner among the practical range functions from our collection.

Table 2. Average running time of Eval with 1024-bit floating point arithmetic in seconds.

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	$E_4^{H'}$	$E_{4,10}^{H'}$	$E_{4,15}^{H'}$	$E_{4,20}^{H'}$	$\sigma(E_4^{H'})$	$\sigma(E_{4,15}^{H'})$	$\sigma(E_{3,15}^{L'})$
T_{20}		0.0288	$\underline{0.0152}$	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	$\underline{0.0637}$	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	$\underline{0.327}$	0.465	0.494	0.49	1.16	0.77	0.98
T_{160}		8.23	1.82	1.71	$\underline{1.23}$	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	$\underline{5.11}$	5.44	6.19	10.4	9.53	6.68	7.84	9.29	1.33	1.62	2.34
H_{20}		0.0242	$\underline{0.0127}$	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	$\underline{0.0547}$	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	$\underline{0.225}$	0.324	0.349	0.346	1.15	0.74	0.98
H_{160}		5.47	1.22	1.16	$\underline{0.854}$	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	$\underline{9.21}$	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	$\underline{0.0072}$	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	$\underline{0.0309}$	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	$\underline{0.14}$	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	$\underline{0.898}$	2.96	1.53	1.62	1.34	0.79	0.72
W_{20}		0.0492	0.0222	$\underline{0.0201}$	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	$\underline{0.0858}$	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	$\underline{0.38}$	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	$\underline{2.22}$	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	$\underline{0.286}$	0.378	0.436	0.461	1.23	0.81	1.06
S_{200}	10	9.55	2.32	2.21	$\underline{1.2}$	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	$\underline{4.89}$	5.84	6.66	13.4	12.5	6.46	8.28	9.98	1.34	2.01	2.85

Table 3. Average running time of Eval with multi-precision rational arithmetic in seconds.

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	$E_4^{H'}$	$E_{4,10}^{H'}$	$E_{4,15}^{H'}$	$E_{4,20}^{H'}$	$\sigma(E_4^{H'})$	$\sigma(E_{4,15}^{H'})$	$\sigma(E_{3,15}^{L'})$
T_{20}		0.0411	$\underline{0.0223}$	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	$\underline{0.0959}$	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	$\underline{0.524}$	0.769	0.805	0.781	1.2	0.78	0.98
T_{160}		11.3	3.14	2.87	$\underline{2.23}$	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	$\underline{13.7}$	14.1	15.9	36.2	21.8	16.6	18.5	21.8	1.46	1.72	2.25
H_{20}		0.03	$\underline{0.0169}$	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	$\underline{0.0844}$	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	$\underline{0.329}$	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	$\underline{1.47}$	1.81	1.87	2.13	1.35	1.06	1.32
H_{320}		-	29.5	28.9	303	67	27.7	39.1	$\underline{20.9}$	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	$\underline{0.0106}$	0.0162	0.0198	0.0233	1.09	0.58	0.75
M_{41}	10	0.124	$\underline{0.0466}$	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	$\underline{0.236}$	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	$\underline{1.57}$	5.89	2.62	2.42	1.39	0.83	0.66
W_{20}		0.0652	$\underline{0.0332}$	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	$\underline{0.143}$	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	$\underline{0.708}$	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	$\underline{4.78}$	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	$\underline{0.419}$	0.603	0.591	0.57	1.13	0.80	0.98
S_{200}	10	12	3.65	3.49	$\underline{2.28}$	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	$\underline{16.4}$	18.9	21.5	51.8	30	19.6	24.2	28.3	1.50	1.85	2.37

5.1 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 [9]: 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* $\ell \geq 0$. Note that Ratschek [12] has a notion called "order $n \geq 1$" for box forms on rational functions that superficially resembles our level concept. When restricted to polynomials, it

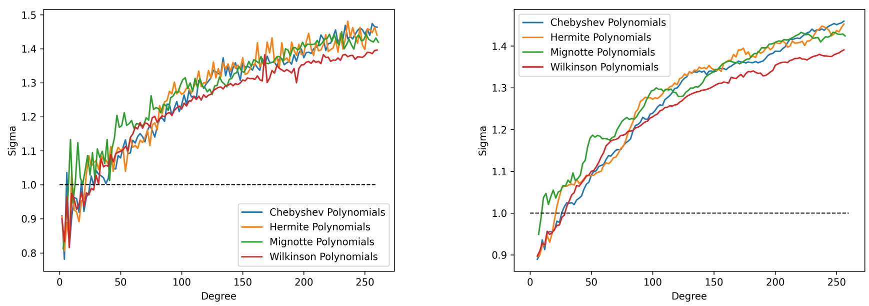

Fig. 1. Speedup σ of $\mathrm{E}_4^{H'}$ with respect to $\mathrm{E}_3^{L'}$ for different families of polynomials and varying degree: raw (left) and smoothed with moving average over five points (right).

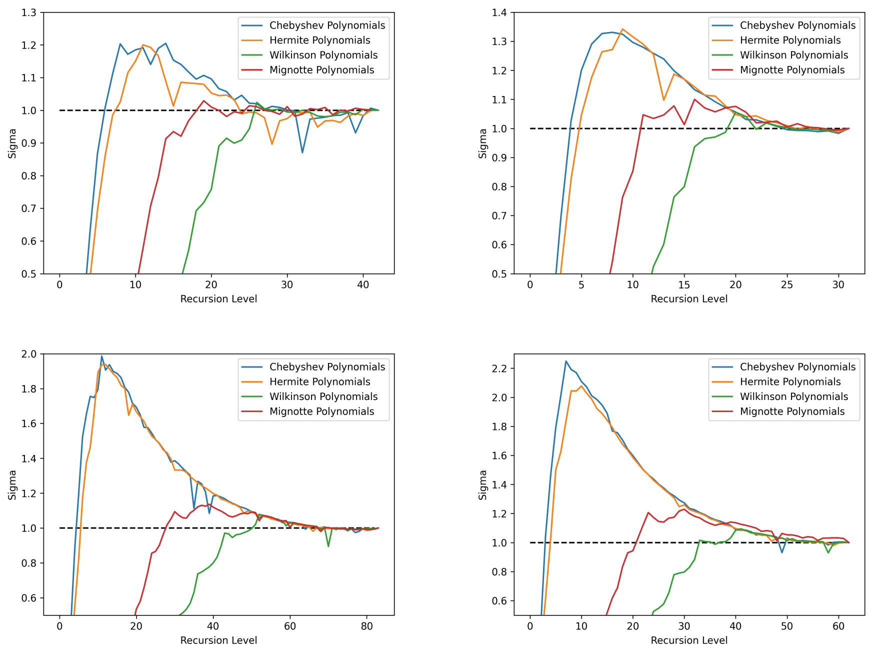

Fig. 2. Speedup $\sigma(\ell)$ of $\mathrm{E}_{3,\ell}^{L'}$ (left) and $\mathrm{E}_{4,\ell}^{H'}$ (right) against their maximal level counterparts with respect to ℓ for polynomials of degree 125 (top) and 250 (bottom) from different families.

diverges from our notion. In other words, we propose to use[10] the pair (k, ℓ)

[10] This is a notational shift from our previous paper. We previously indexed the recursion level by $n \geq 1$. Thus, level ℓ in this paper corresponds to $n - 1$ in the old notation.

of convergence measures in evaluating our range functions. In [9] 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. We now experimentally explore the use of non-maximal levels.

Figure 2 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 [9] 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 2), 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 $\mathrm{E}_4^{H'}$ than for $\mathrm{E}_3^{L'}$ and that it increases with the degree of the polynomial f.

6 Conclusions and Future Work

We generalized the CL framework in order to achieve, for the first time, range functions of arbitrarily high order of convergence. Our recursive scheme for such constructions is not only of theoretical interest, but are practical as shown by our implementations. Devising specific "best of a given order" functions like $\square_{4,\ell}^H f(I)$ is also useful for applications.

The amortized complexity model of this paper can be used to analyze many subdivision algorithms in higher dimensions. Moreover, new forms of range primitives may suggest themselves when viewed from the amortization perspective.

We pose as a theoretical challenge to explain the observed phenomenon of the "unimodal" behavior of the $\sigma(\ell)$ plots of Figure 2 and to seek techniques for estimating the optimal recursion level that achieves the minimum time. Moreover, we would like to better understand why the size of the Eval subdivision tree increases with ℓ in the case of sparse polynomials (see Table 1), while it decreases for all other polynomials from our test suite.

Finally, we emphasize that strong box functions have many applications. Another future work therefore is to develop the theory of strong box functions, turning the abstract model of Section 2.2 into an effective (Turing) model in the sense of [19].

References

1. Burr, M., Krahmer, F.: SqFreeEVAL: An (almost) optimal real-root isolation algorithm. J. Symbolic Computation **47**(2), 153–166 (2012)

2. Core Library homepage (since 1999): Software download, source, documentation and links: https://cs.nyu.edu/exact/core_pages/svn-core.html
3. Cornelius, H., Lohner, R.: Computing the range of values of real functions with accuracy higher than second order. Computing **33**(3), 331–347 (Sep 1984)
4. Du, Z., Eleftheriou, M., Moreira, J., Yap, C.: Hypergeometric functions in exact geometric computation. In: V. Brattka, M. Schoeder, K. Weihrauch (eds.) Proc. 5th Workshop on Computability and Complexity in Analysis, pp. 55–66, 2002
5. Du, Z., Yap, C.: Uniform complexity of approximating hypergeometric functions with absolute error. In: S. Pae, H. Park, (eds.) Proc. 7th Asian Symp. on Computer Math, pp. 246–249, 2006
6. Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., Zimmermann, P.: MPFR: A multiple-precision binary floating-point library with correct rounding. ACM Transactions on Mathematical Software **33**(2), Article 13, 15 pages (Jun 2007). The MPFR library is available at `https://www.mpfr.org`
7. Granlund, T., the GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library. `http://gmplib.org/`, 6.2.1 edn. (Nov 2020)
8. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. Society for Industrial and Applied Mathematics, Philadelphia, second edn. (2002)
9. Hormann, K., Kania, L., Yap, C.: Novel range functions via taylor expansions and recursive lagrange interpolation with application to real root isolation. In: Int'l Symp. Symbolic and Alge. Comp. (46th ISSAC), pp. 193–200, 2021
10. Johansson, F.: Computing hypergeometric functions rigorously. ACM Trans. on Math. Software **45**(3), 1–26 (2019)
11. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
12. Ratschek, H.: Centered forms. SIAM J. Num. Anal. **17**(5), 656–662 (1980)
13. Ratschek, H., Rokne, J.: Computer Methods for the Range of Functions. Horwood Publishing Limited, Chichester, West Sussex, UK (1984)
14. Revol, N., Rouillier, F.: Motivations for an arbitrary precision iinterval arithmetic and the MPFI library. Reliable Computing **11**(4), 275–290 (Aug 2005). The MPFI library is available at `https://gitlab.inria.fr/mpfi/mpfi`
15. Shadrin, A.: Error bounds for Lagrange interpolation. J. Approximation Theory **80**(1), 25–49 (Jan 1995)
16. Sharma, V., Yap, C.: Near optimal tree size bounds on a simple real root isolation algorithm. In: 37th Int'l Symp. Symbolic and Alge. Comp.(ISSAC'12). pp. 319–326, 2012
17. Trefethen, L.N., Bau, D.: Numerical Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia (1997)
18. Waldron, S.F.: L_p-error bounds for Hermite interpolation and the associated Wirtinger inequalities. J. Constructive Approximation **13**(4), 461–479 (1997)
19. Xu, J., Yap, C.: Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis. In: Int'l Symp. Symbolic and Alge. Comp. (44th ISSAC), pp. 355–362, 2019
20. Yap, C.K.: On guaranteed accuracy computation. In: Chen, F., Wang, D. (eds.) Geometric Computation, chap. 12, pp. 322–373. World Scientific Publishing Co., Singapore (2004)
21. Yu, J., Yap, C., Du, Z., Pion, S., Bronnimann, H.: Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In: 3rd Proc. Int'l Congress on Mathematical Software (ICMS), pp. 121–141. Springer (2010). LNCS No. 6327.