Oracle-Free Repair Synthesis for Floating-Point Programs

DAMING ZOU, ETH Zurich, Switzerland
YUCHEN GU, Peking University, China
YUANFENG SHI, Peking University, China
MINGZHE WANG, Princeton University, USA
YINGFEI XIONG, Peking University, China
ZHENDONG SU, ETH Zurich, Switzerland

The floating-point representation provides widely-used data types (such as "`float`" and "`double`") for modern numerical software. Numerical errors are inherent due to floating-point's approximate nature, and pose an important, well-known challenge. It is nontrivial to fix/repair numerical code to reduce numerical errors — it requires either *numerical expertise* (for manual fixing) or high-precision *oracles* (for automatic repair); both are difficult requirements. To tackle this challenge, this paper introduces a *principled dynamic approach* that is *fully automated* and *oracle-free* for effectively repairing floating-point errors. The key of our approach is the novel notion of *micro-structure* that characterizes structural patterns of floating-point errors. We leverage micro-structures' statistical information on floating-point errors to effectively guide repair synthesis and validation. Compared with existing state-of-the-art repair approaches, our work is fully automatic and has the distinctive benefit of not relying on the difficult to obtain high-precision oracles. Evaluation results on 36 commonly-used numerical programs show that our approach is highly efficient and effective: (1) it is able to synthesize repairs instantaneously, and (2) versus the original programs, the repaired programs have orders of magnitude smaller floating-point errors, while having faster runtime performance.

CCS Concepts: • **Software and its engineering → Software testing and debugging**.

Additional Key Words and Phrases: floating-point error, micro-structure, program repair, dynamic analysis

ACM Reference Format:
Daming Zou, Yuchen Gu, Yuanfeng Shi, MingZhe Wang, Yingfei Xiong, and Zhendong Su. 2022. Oracle-Free Repair Synthesis for Floating-Point Programs. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 159 (October 2022), 29 pages. https://doi.org/10.1145/3563322

1 INTRODUCTION

Floating-point (FP) numbers are essential and widely-used in modern application domains, such as science, engineering, and finance [Sanchez-Stern et al. 2018]. Because they use finite precision to approximate real numbers, it is well-known that FP results can be inaccurate. Such inaccuracies have led to high-profile catastrophes, such as rocket launch failure [Lions et al. 1996], vehicle brake failure [Valdes-Dapena and Lah 2010], and the loss of human lives [Skeel 1992]. Modern systems, such as probabilistic programming systems [Dutta et al. 2018] and deep learning libraries [Pham

Authors' addresses: Daming Zou, Department of Computer Science, ETH Zurich, Switzerland, daming.zou@inf.ethz.ch; Yuchen Gu, School of Computer Science, Peking University, China, llgyc@pku.edu.cn; Yuanfeng Shi, School of Computer Science, Peking University, China, saigyouji17@pku.edu.cn; MingZhe Wang, Department of Computer Science, Princeton University, USA, mingzhew@cs.princeton.edu; Yingfei Xiong, School of Computer Science, Peking University, China, xiongyf@pku.edu.cn; Zhendong Su, Department of Computer Science, ETH Zurich, Switzerland, zhendong.su@inf.ethz.ch.

2475-1421/2022/10-ART159
https://doi.org/10.1145/3563322

et al. 2019], also suffer from FP inaccuracies. Thus, it is important to repair and reduce FP errors for improving the accuracy of numerical programs.

Repairing significant FP errors is nontrivial. Simply changing FP numbers with higher-precision types is not the silver bullet, since it may (1) *introduce extra errors* due to precision-specific operations [Wang et al. 2016] and precision-related code [Zou et al. 2020], and (2) *slow down program execution*, possibly by thousands of times [Benz et al. 2012; Peter Larsson 2013]. In practice, reducing FP errors relies heavily on human expertise. For widely-used numerical libraries such as the GNU Scientific Library (GSL), numerical experts manually craft sophisticated expressions and polynomials to reduce FP errors. For example, in GSL, even the simple function[1] for computing $\log(1+x)$ involves a polynomial with 21 coefficients and terms. Thus, it is difficult for non-experts to manually craft such polynomials to reduce FP errors.

Several techniques [Panchekha et al. 2015; Yi et al. 2019] have been proposed to automatically repair FP errors. However, they critically depend on *oracle* implementations for providing reference high-precision results. As aforementioned, higher-precision data types do not guarantee more accurate results due to precision-specific operations and precision-related code (see Section 3 for an example and discussion). Thus, for users without numerical expertise, it is difficult to construct oracles to apply such automated techniques.

Recent work has introduced *oracle-free* (*i.e.*, without relying on reference high-precision implementations) techniques [Guo and Rubio-González 2020; Zou et al. 2020] for detecting inputs that trigger significant FP errors. Such techniques are shown effective in detecting FP errors, thus enabling the construction of oracle-free repair techniques — oracle-free error detection provides the inputs for oracle-free repair, the problem that this paper targets.

Micro-structure and Aceso. Indeed, this paper introduces a novel oracle-free approach to repairing FP errors by synthesizing polynomial patches as numerical experts do, but fully automatically. This is a difficult challenge. Our *key observation* is that *FP errors near an input exhibit structural patterns with a suitably chosen scope*, which we refer to as a *micro-structure*. Intuitively, a micro-structure illustrates the distribution of FP errors near a specific input, such as an input that triggers a significant FP error. Analyzing the micro-structure provides valuable information on the FP error such as (1) the range and variance of the error, and (2) the expected value of the result, which we collectively refer to as the *statistical information* on the FP error. By leveraging this information, we can understand and describe the FP errors near any specific input without relying on an oracle.

For example, consider the following program `foo(x)`:

```
double foo(double x) {
    double bx = bessel_I1(x);
    return (exp(bx) - 1.0) / x;
}
```

which includes a library call to GSL's Bessel function. On the input x=1.2e-5, `foo(x)` returns 5.0000150001820265e-1. Without a provided oracle, it is difficult to analyze this result — we cannot measure the error, and cannot even know the existence of any error. However, the statistical information based on the micro-structure shows that, for this input x=1.2e-5, we can obtain the following information:

- Error variance: 2.86e-23
- Error range: [-9.31e-12, 9.37e-12]
- Expected result: 5.0000150001196952e-1

[1]We will elaborate this example in Section 2.4.

Note that the oracle result is 5.0000150001200006e-1 on this input, and the actual error is 6.20e-12. Thus, the above statistical information, which is computed in an oracle-free manner, reveals accurate, relevant information for understanding and analyzing FP errors (*cf.* Section 5.2).

Based on this insight on micro-structures of FP errors, we develop an effective oracle-free approach and its accompanying realization Aceso[2] to synthesize patches for repairing FP errors. Our approach has the following strengths:

- *Oracle-free*: The statistical information on FP errors allows our approach to be independent of oracles, thus making it generally applicable in practice.
- *Efficient*: Micro-structures and statistical information can be obtained directly from the original FP code, and do not require high-precision executions, thus leading to high efficiency.
- *Effective*: The statistical information reveals accurate knowledge on FP errors and effectively guides the synthesis of accurate polynomial patches.

At the high level, Aceso takes the erroneous interval $\mathcal{I}$ (which can be obtained from available error detection techniques [Guo and Rubio-González 2020; Zou et al. 2020]) for a program as input, samples a set of points within $\mathcal{I}$, collects statistical information on each of the sampled points, and synthesizes polynomial patches based on such information with a fitting model. It also validates the patches automatically, and only returns patches more accurate than the original program.

We evaluate Aceso on 32 commonly-used erroneous numerical programs, including 17 complex programs with library calls and 15 classical programs that have been studied in previous work [Panchekha et al. 2015; Solovyev et al. 2018; Wang et al. 2019], and 4 additional error-free programs (Table 2). Our results show that Aceso can synthesize patches that reduce FP errors by *orders of magnitude* (in terms of maximum absolute/relative error on the entire interval $\mathcal{I}$) for all the 32 erroneous programs. For the 4 error-free programs, Aceso correctly rejects the synthesized patches and reports "patch not available." Our results also show that the statistical information from micro-structures is key to Aceso's effectiveness — without such information, the synthesized patches become infeasible or erroneous. Aceso is also efficient and takes on average 0.73 seconds to repair each program. Furthermore, the synthesized patches are performant and take on average ~9 nanoseconds (*i.e.*, 10^{-9} seconds) for each execution, even faster than the original programs.

Targeted scenario. The scenario that our work targets is to repair FP errors in numerical code whose oracle is challenging to obtain. This is a common, practical scenario as a single library call in the numerical code can make constructing its oracle infeasible (*e.g.*, the aforementioned `foo(x)` with a call to `bessel_I1`). As an application developer, one most oftenly simply invokes a library call; it is difficult for the developer to construct a high-precision implementation of the call as the oracle required by oracle-dependent repair techniques. It is difficult because of the prevalent precision-related code [Zou et al. 2020] and precision-specific operations [Wang et al. 2016] in numerical libraries (see a detailed discussion in Section 3).

Contributions. In summary, we make the following main contributions:

- A *novel characterization of FP errors* via the concept of micro-structure and statistical information based on micro-structures to effectively capture how FP errors in a program distribute near a specific point;
- A *fitting model* for synthesizing numerically-accurate polynomial patches by leveraging the statistical information on FP errors;
- An *effective realization* of the approach Aceso — Aceso automatically synthesizes and validates polynomial patches based on the fitting model and statistical information, and does not rely on oracles, thus efficient and generally applicable; and

[2]The Greek goddess of the healing process and curing of sickness.

- An *empirical evaluation* of Aceso on 36 commonly-used programs to demonstrate that Aceso is effective and efficient in synthesizing accurate and performant patches.

2 PRELIMINARIES

This section presents the necessary background for our work.

2.1 Floating-Point Representation

Floating-point (FP) numbers approximate real numbers, and can represent a finite subset of the infinite set of real numbers. The IEEE 754 standard [Zuras et al. 2008] stipulates the representation of FP numbers, which consists of three parts: sign, exponent, and significand (also called mantissa). Table 1 shows the formats of FP numbers in different precision modes.

Table 1. IEEE 754 floating-point representations.

Precision	Sign	Exponent (m)	Significand (n)
Half (16 bits)	1	5	10
Single (32 bits)	1	8	23
Double (64 bits)	1	11	52

2.2 Error Measurement

Since FP numbers ($\mathbb{F}$) use finite bits to represent real numbers, it is natural that rounding errors exist. For an FP program $\mathcal{P}$: $\hat{y} = \hat{f}(\mathbf{x})$, $\hat{y} \in \mathbb{F}$, rounding errors can be introduced and accumulated for its FP operations, and may lead to inaccurate results. There are two standard mathematical measures for the error between the ideal result $f(\mathbf{x})$ and the FP result $\hat{f}(\mathbf{x})$: *absolute error* Err_{abs}, and *relative error* Err_{rel}:

$$Err_{abs}(f(\mathbf{x}), \hat{f}(\mathbf{x})) = \left| f(\mathbf{x}) - \hat{f}(\mathbf{x}) \right|$$
$$Err_{rel}(f(\mathbf{x}), \hat{f}(\mathbf{x})) = \left| (f(\mathbf{x}) - \hat{f}(\mathbf{x}))/f(\mathbf{x}) \right|$$

Besides absolute and relative errors, *units in the last place (ulp)* is a measure for the relative error [Zuras et al. 2008] and is specific for FP numbers. It is defined as [Kahan 2004; Muller 2005]:

$ulp(x)$ is the gap between the two FP numbers nearest x, even if x is one of them.

For double precision (64-bits) FP numbers, 1 *ulp* error corresponds to a relative error between 1.1×10^{-16} and 2.2×10^{-16}.

2.3 Error Detection

Given an FP program $\mathcal{P}$: $\hat{y} = \hat{f}(x)$ under analysis, the goal of FP error detection is to find test inputs that trigger significant FP errors. Most existing error detection approaches require the oracle $f(x)$ to be available for an arbitrary input x [Chiang et al. 2014; Yi et al. 2017; Zou et al. 2015], which is used to compute the FP error, Err_{rel} or Err_{abs}, during their respective search procedures.

Recent work [Guo and Rubio-González 2020; Zou et al. 2020] has introduced *oracle-free* approaches. For example, Atomu [Zou et al. 2020] is such an approach that is publicly available.[3] It significantly outperforms existing error detection approaches in terms of both effectiveness and efficiency, thus making detecting FP errors more practical. Our work aims for an *oracle-free* approach to *repairing* FP errors, thus directly benefiting from oracle-free error detection for providing the intervals to be patched.

[3] https://github.com/FP-Analysis/atomic-condition

2.4 Polynomials for Approximation

In numerical analysis, it is common to use polynomials to approximate mathematical functions [Hildebrand 1987], such as power series [Abramowitz et al. 1988] and Chebyshev series [Mason and Handscomb 2002]. One reason for using polynomial approximations is to reduce potential FP errors.[4] We use an example from the GNU Scientific Library (GSL) to show how numerical experts craft sophisticated polynomials to reduce FP errors.

The method "`log_1plusx`" from GSL calculates a simple function: $f(x) = \log(1+x)$ for $x > -1$. We observe that this function applies different algorithms for different values of x:

- For $|x| \in [0, 7.4 \times 10^{-4})$, it applies power series for evaluation;
- For $|x| \in [7.4 \times 10^{-4}, 0.5)$, it applies Chebyshev series for evaluation; and
- For $|x| \in [0.5, +\inf)$, it evaluates directly based on the expression $\log(1+x)$.

2.4.1 Power Series. Power series $P_n(x)$ is represented by the coefficients p_r in

$$P_n(x) = \sum_{r=0}^{n} p_r x^r \tag{1}$$

To approximate a mathematical function $f(x)$, the coefficients p_r are usually obtained manually, *e.g.*, from Taylor series, and hard-coded into the FP program.

In GSL's source code, it applies a power series with 10 terms to approximate its analytic expression $\log(1+x)$ for $|x| < 7.4^{-4}$, based on its Taylor series at $x = 0$:

$$P_{10}(x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots - \frac{x^{10}}{10} \tag{2}$$

2.4.2 Chebyshev Series. As discussed in Section 2.4.1, a power series is composed of linear combinations of x^r $(1, x, x^2, \textit{etc.})$. Correspondingly, a Chebyshev series is composed of linear combinations of *Chebyshev polynomials*, $T_s(x)$, with coefficients c_s:

$$P_n(x) = \sum_{s=0}^{n} c_s T_s(x), \text{where} \tag{3}$$

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \tag{4}$$

Chebyshev series is widely-used in numerical methods and aims to minimize approximation errors [Burden and Faires 2010].

In GSL's source code, when $|x| \in [7.4 \times 10^{-4}, 0.5)$, it applies Chebyshev series to produce more accurate results, rather than directly evaluating the expression $\log(1+x)$:

```
/* Chebyshev series for 7.4e-4 <= |x| < 0.5 */
const double lopx_data[21] = {
  2.16647910664395270521272590407,
 -0.28565398551049742084877469679,
  0.01517767255690553732382488171,
 -0.00200215904941415466274422081,
  ... };
// Domain adaptation
double t = 0.5*(8.0*x + 1.0)/(x+2.0);
return cheb_eval(lopx_data, t);
```

[4]Polynomial approximations have wide applications in numerical analysis, such as reducing FP errors, reducing runtime cost, and approximating mathematical functions with computable operations (*e.g.* + and ×), *etc.*

Since Chebyshev series requires the input domain to be $[-1, 1]$ [Gil et al. 2007], line 8 applies domain adaptation $t(x) = \frac{8x+1}{2x+4}$ to map $x \in (-0.5, 0.5)$ to $t(x) \in (-1, 1)$.[5] Line 9 evaluates the series with the hard-coded coefficients in `lopx_data`.

3 EXAMPLE

This section uses a concrete example to motivate and illustrate our approach, from obtaining statistical information on FP errors to synthesizing and validating the final patch.

Motivating example. Recall that this work targets the common scenario of repairing numerical code whose oracle is difficult to construct. As a concrete example, let us consider that an application developer wrote the following function $\hat{f}$: `foo(x)` which invokes GSL's Bessel function:

```
double foo(double x) {
    double bx = bessel_I1(x); // gsl_sf_bessel_I1(x)
    return (exp(bx) - 1.0) / x;
}
```

To check whether the code is numerically accurate, the developer applied an existing oracle-free FP error detection technique such as Atomu [Zou et al. 2020]. The tool reported an erroneous interval $\mathcal{I} = [-10^{-2}, 10^{-2}]$, *i.e.*, inputs in $\mathcal{I}$ trigger significant FP error in `foo(x)`.

To repair the FP error on this reported interval $\mathcal{I}$, an obvious approach is to construct a high-precision oracle and then apply oracle-based repair techniques [Panchekha et al. 2015; Yi et al. 2019]. However, constructing a high-precision oracle for `foo(x)` is challenging; it requires every involved operation to have high precision, including the library call, which further requires either

- *A high-precision implementation of* `bessel_I1`, which does not exist in the high-precision infrastructure, MPFR library [Fousse et al. 2007], and in practice, one cannot expect that a high-precision implementation exists for every library call; or
- *The assumption that replacing all operations inside the code of* `bessel_I1` *to high-precision types yields high-precision results*, which in general does not hold since the library code may involve precision-related code. In our example, `bessel_I1` contains a hard-coded Chebyshev series with fixed terms (*cf.* Section 2.4.2). Thus, an accurate oracle needs not only high-precision FP types, but also *additional terms* with carefully constructed coefficients, which is very challenging. Indeed, Section 6.2.2 and Figure 7 describe results to show that simply replacing FP types cannot produce accurate oracles.

In practice, application developers, who are typically not numerical experts like the GSL developers, simply invoke a library call; they are unable (and unwilling) to construct accurate oracles and repair FP errors with oracle-dependent techniques. Thus, oracle-free repair is needed.

Illustrative example for our approach Aceso. Our goal is, in an oracle-free manner, to *automatically synthesize a polynomial patch $p(x)$ with improved accuracy* on the erroneous interval $\mathcal{I}$, similar to what numerical experts do manually. Our procedure works as follows:

(1) Specify a polynomial template $p(x)$ with unknown coefficients c_i, *e.g.*, a power series template $p(x) = c_0 + c_1 x + c_2 x^2$.
(2) Sample a set of points $\{x_1, \ldots, x_n\}$ in the interval $\mathcal{I}$ and adapt $\mathcal{I}$ to $\mathcal{I}' = [-1, 1]$ (*cf.* Section 5.1 on interval sampling and domain adaptation).
(3) Observe the micro-structure and gather each sample point's statistical information. For example, on one of the sample points $x_j = 0.00383$, the statistical information shows
    - Estimated mean $\overline{f}(x_j) = 0.5004795765784401$

[5]Different domain adaptations lead to different coefficients. This domain adaptation, together with the corresponding coefficients, aims to increase the accuracy of the Chebyshev series approximation [Broucke 1973; Clenshaw 1962].

- Estimated error variance $Var(\varepsilon_j) = 2.78 \times 10^{-28}$

(*cf.* Sections 4.2 and 4.3 on observing micro-structure and gathering statistical information).

(4) Solve for the unknown coefficients c_i based on the statistical information on all sample points $x_j \in \{x_1, \ldots, x_n\}$: $\overline{f}(x_j)$ and $Var(\varepsilon_j)$ (*cf.* Section 5.2 on fitting with statistical information).

After the coefficients have been solved, the repair patch of this example is shown below:

```
double foo_patched(double x) {
  if (x >= -1e-2 && x <= 1e-2) {
    double c0 = 4.9999999998665351e-01;
    double c1 = 1.2500253909111360e-03;
    double c2 = 8.3334401054235840e-06;
    double u = 100 * x; // Domain Adaption
    return c0 + u*(c1 + u*c2);
  }
  else return foo(x);
}
```

Once a candidate patch is synthesized, our approach performs *patch validation* (*cf.* Section 5.3) to evaluate it on a set of random inputs with the corresponding statistical information and reject any inaccurate patches. Note that, for presentation clarity, this example patch is based on only four sampling points and with the highest degree of two. In practice, with suitable parameters, the patch's accuracy can be further improved.

4 MICRO-STRUCTURE AND LOCAL SAMPLING

The key behind our repair approach is a novel characterization of FP errors, which we call the *micro-structure* of FP errors. This section introduces micro-structure and discusses how to leverage it to gather statistical information on FP errors in an oracle-free manner.

We structure this section as follows. Section 4.1 introduces and illustrates the micro-structure of FP errors. Section 4.2 presents a method for selecting a suitable radius (or neighborhood) to observe the micro-structure. Then, Section 4.3 details how to gather statistical information on FP errors based on the selected neighborhood.

This section focuses on gathering error information near an arbitrary point, thus we refer to the workflow as *local sampling* (corresponding to *interval sampling* in Section 5). Figure 1 depicts the workflow, which consists of radius selection (Section 4.2) and information gathering (Section 4.3).

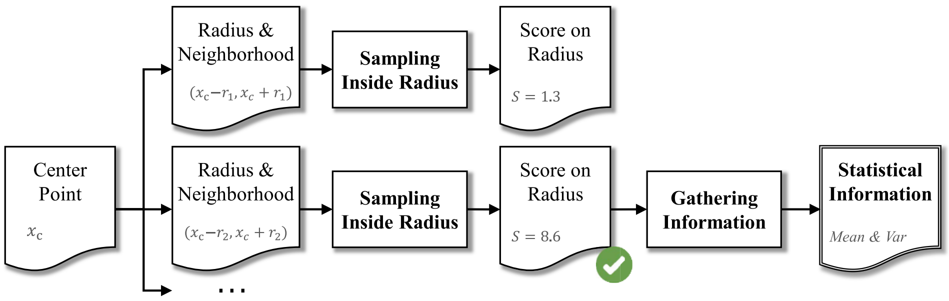


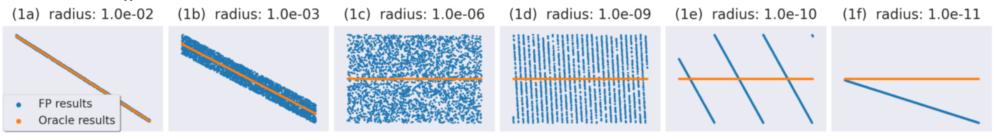
Fig. 1. Workflow of local sampling. The procedure to collect pointwise information on x_c.

4.1 Micro-Structure Overview

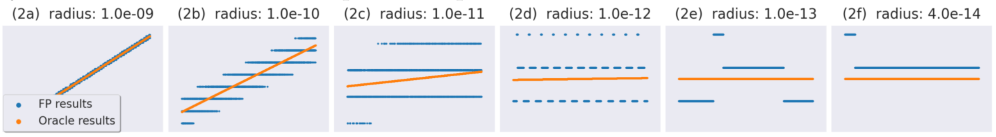
Before presenting the *micro-structure* of FP errors, we give the relevant notations and definitions:

- *Center point* (x_c) is an arbitrary point to observe the micro-structure in its neighborhood.

- $f(x) = \frac{1-\cos(x)}{x^2}$ with example center point $x_c = 1.2345 \times 10^{-3}$



- $f(x) = e^x - 2 + e^{-x}$ with example center point $x_c = 1.2345 \times 10^{-3}$



- $f(x) = \sqrt{\frac{e^{2x}-1}{e^x-1}}$ with example center point $x_c = 1.2345 \times 10^{-3}$

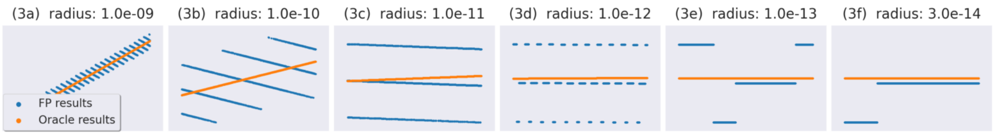


Fig. 2. Micro-structure of FP errors.[6] In each sub-figure, the x-axis denotes the *inputs* in the neighborhood (center point ± radius), the y-axis represents the corresponding FP results $\hat{f}(x)$ and the oracle results $f(x)$. The x-axis and y-axis are scaled to make the data points fill the entire sub-figure, thus the range for the axes are not the same across all sub-figures. The ideal neighborhoods (radii) to be selected in Section 4.2 are the ones in column (c) and (d), as the micro-structure in such neighborhoods are both *significant* and *complete*.

- *Radius* (r) determines the scope of the observation.
- *Neighborhood* $\phi(x_c, r) = (x_c - r, x_c + r)$, which is determined by the center point and radius.
- *Observation* of micro-structure is a set of sampling points within the neighborhood $\phi(x_c, r)$. Each sampling point is labeled with $\langle x_i, y_i \rangle$, where $x_i \in \phi$ is an input and $y_i = \hat{f}(x_i)$ is the corresponding FP result.

Figure 2 illustrates micro-structures on three functions. The sub-figures on each row correspond to observations with different neighborhoods. The center is set at the example value $x_c = 1.2345 \times 10^{-3}$ for all three functions. The blue points in Figure 2 correspond to the FP results. Although our approach is oracle-free, we also include the oracle results as orange points for better clarity and understanding.

We can summarize some general characteristics of micro-structures from Figure 2:

- FP errors are not completely acting like random variables with uniform distribution or normal distribution, as suggested by previous work [Chatelin and Brunet 1990; Higham and Mary 2019; Tienari 1970]. For example, although Figure 2 (1c) shows that FP errors appear random, there are striking patterns, *e.g.*, the patterns shown in Figure 2 (1d) and (1e). There are also structural patterns for the other two functions. We call these patterns the *micro-structures* of FP errors.
- The patterns of micro-structures vary; the example functions exhibit different micro-structures.

[6]Due to their figure resolutions, Figure 2 (2b), (2c), (3b), and (3c) may appear multi-valued. But the functions are all ordinary single-valued functions, and each input always maps to the same result, as in Figure 2 (2d) and (3d). The patterns are just too dense to be distinguished.

- Radii and neighborhoods for observing micro-structures vary. All the illustrated neighborhoods have the same center point, but micro-structures manifest under different radii, *e.g.*, sub-figures (1e), (2e), and (3e) in Figure 2.

More technically, a micro-structure is the *discontinuous*,[7] structural pattern of the FP program results *w.r.t.* FP arithmetic. It can be observed under certain scope/interval, where the ideal result $f(x)$ is continuous while the FP result $\hat{f}(x)$ is discontinuous, and the FP error dominates the result range. For example, in Figure 2 (3d) and (3e), $f(x)$ is virtually constant on this interval, while FP errors dominate the range of results, and make $\hat{f}(x)$ discontinuous.

We expect the existence of micro-structures to be a general property of FP errors. In fact, micro-structures result from discontinuous points in FP code, which are caused by changes in the rounding directions. In numerical programs, rounding happens in almost all FP operations and leads to discontinuities (or "jumps"); the combination, accumulation, and amplification of such discontinuities manifest as micro-structures. Therefore, it is a general property of FP errors that micro-structures exist.

By observing the micro-structure of FP errors, we can obtain information on the distribution of FP errors near the center point x_c (which we will elaborate in Section 4.3). We now propose two basic principles for observing micro-structures based on Figure 2:

- ***Significance***. To observe the micro-structure of FP errors, the radius should be sufficiently small to make FP errors *significant* within the neighborhood. For Figure 2 (1a), (2a), and (3a), the radii are relatively large, thus the ranges are dominated by the variation from the original functions, rather than FP errors.
- ***Completeness***. To observe the micro-structure of FP errors, the radius should be sufficiently large to contain the *complete* variations on FP errors. For Figure 2 (1f), (2f), and (3f), the radii are small, and the variations on FP errors in these neighborhoods are incomplete and biased.

Thus, an ideal neighborhood, *e.g.*, Figure 2 (d) in each row, should contain both significant and complete FP errors for the observation. We will discuss how to select such a radius and neighborhood in the next section.

4.2 Radius Selection for Error Observation

This section discusses how to select radius r_x for an arbitrary input x to determine the neighborhood under observation. The goal of radius selection, as discussed in Section 4.1, is to find a suitable neighborhood ϕ inside which the FP error is *significant* and *complete*.

At the high level, we propose a metric $S_\phi(n, \xi)$ to quantify the discontinuity for a neighborhood ϕ, so that we can generate different radii and select the radius/neighborhood based on $S_\phi(n, \xi)$.

Defining the metric $S_\phi(n, \xi)$**.** Since micro-structures may vary across different functions and even different center points, we cannot simply design a radius selection method based on a specific pattern of micro-structure. We leverage a key insight to guide radius selection: the micro-structure has the property of *discontinuity*. In Figure 2 (1e), (2e), and (3e), although the functions and micro-structures are different, we can clearly observe several discontinuities in these figures. In fact, discontinuity is a general property for FP errors since they are caused by changes in the rounding directions.

To quantitatively guide radius selection, we design a metric $S_\phi(n, \xi)$ with the following definition, which leverages the property of discontinuity:

[7] In this work, we use the words "discrete" and "discontinuous" with meanings different from their standard. FP numbers are always discrete since they have fixed numbers of bits, and there is a "unit in the last place" (ulp) between two adjacent FP numbers. Discontinuity in micro-structures is far larger than ulp for significant FP errors. Since the ulp can be easily obtained, we can simply distinguish the discrete and discontinuous points by setting a threshold based on ulp.

- $\phi(x_c, r) = [x_c - r, x_c + r]$ is a neighborhood.
- n is a tunable parameter for defining n ordered points $\{x_i \in \phi \mid x_c - r = x_1 < x_2 < \cdots < x_n = x_c + r\}$ with equal distances.
- $\Delta = \max\{f(x) \mid x \in \{x_1, \ldots, x_n\}\} - \min\{f(x) \mid x \in \{x_1, \ldots, x_n\}\}$ measures the range of the neighborhood ϕ, and is determined by ϕ and n.
- ξ is a tunable threshold.
- $S_\phi(n, \xi)$ measures the discontinuity within the neighborhood ϕ:

$$S_\phi(n, \xi) = \sum_{i=1}^{n-1} \frac{g(x_i)}{\Delta} \qquad g(x_i) = \begin{cases} \left|f(x_i) - f(x_{i+1})\right| & \text{if } \left|f(x_i) - f(x_{i+1})\right| > \xi \\ 0 & \text{if } \left|f(x_i) - f(x_{i+1})\right| \le \xi \end{cases} \tag{5}$$

As the number of points n increases, the size of the interval $[x_i, x_{i+1}]$ decreases and converges to 0. Based on the definition of continuity [Rudin et al. 1976], in theory, for any positive ξ, there exists a large enough n such that:

- All *continuous* intervals $[x_i, x_{i+1}]$ satisfy $f(x_{i+1}) - f(x_i) \le \xi$, thus $g(x_i) = 0$, and
- $g(x) > 0$ implies the existence of discontinuities in the interval (x_i, x_{i+1}).

Since the intervals $[x_1, x_2], \ldots, [x_{n-1}, x_n]$ cover the entire neighborhood ϕ, $S_\phi(n, \xi) = \sum g(x_i)/\Delta$ can quantitatively measure the discontinuity of the neighborhood ϕ.

Parameter settings in practice. In theory, the above definition on $S_\phi(n, \xi)$ guarantees soundness for measuring discontinuity. In practice, considering computational cost, we need a guideline to estimate and set parameters ξ and n (rather than an arbitrary ξ and a "large enough" n in theory):

- ξ can be set *w.r.t.* the range Δ on the entire neighborhood, such as $\xi = 0.1\Delta$.
- n can be set based on ξ, such as $n = 100\Delta/\xi = 1000$, which means, if a $g(x_i) > 0$, the domain $[x_i, x_{i+1}]$ only covers $1/1000$ of the neighborhood, while the range change is larger than $1/10$ of the entire range Δ. This is only possible for either discontinuity or extremely large second-order derivatives,[8] and the latter is rare in practice.

Note that we can set stricter (smaller) ξ and larger n for better estimating $S_\phi(n, \xi)$. However, $S_\phi(n, \xi)$ is used to guide radius selection, and is unnecessary to be extremely accurate.

How does $S_\phi(n, \xi)$ address significance and completeness? The metric $S_\phi(n, \xi) = \sum g(x_i)/\Delta$ addresses both the conditions for *significance* and *completeness*:

- The numerator, $\sum g(x_i)$, addresses the condition for completeness. For example, neighborhoods with incomplete micro-structures have fewer discontinuity points, *e.g.*, Figure 2 (1f), (2f), and (3f), thus the quantity $\sum g(x_i)$ is smaller than those for neighborhoods with complete and dense micro-structures.
- The denominator, Δ, addresses the condition for significance. The discontinuity measurement $\sum g(x_i)$ must be significant *w.r.t.* the total range change Δ. For example, Figure 2 (1a) and (2a) would have small $S_\phi(n, \xi)$ since the discontinuity is insignificant in these two neighborhoods.

As the $S_\phi(n, \xi)$ measures the portion of total discontinuity to the total value range in the neighborhood ϕ, it is already normalized and comparable across different neighborhoods. Thus, for different radii and corresponding neighborhoods, we select the radius with maximum $S_\phi(n, \xi)$ as a suitable choice for information gathering, which we discuss next.

[8] If the derivative is *constant* or nearly constant in the neighborhood, which holds in most cases since the neighborhoods are tiny, the magnitude of change in the domain (*e.g.* 1/1000) would be roughly the same as that of change in the range (~1/1000). Otherwise, the derivative changes significantly in this small fraction of the domain, which means an extremely large second-order derivative.

4.3 Statistical Information Gathering

Section 4.2 proposed how to select a suitable radius to make FP errors complete and significant within the corresponding neighborhood. We now discuss how to gather the statistical information on FP errors based on the selected neighborhood. More specifically, we estimate the *mean* of the FP results and the *variance* of the FP errors for describing the characteristics of FP errors near an arbitrary center point x_c.

Estimating the mean of FP results. According to Higham and Mary [2019], it is a common practice in probabilistic/statistical FP error analysis to model FP errors as random variables, and assume the mean of the errors to be zero. Such an assumption is related to the default rounding mode in the IEEE 754 standard, "round to nearest, ties to even". Although not guaranteed, the assumption holds in most practical scenarios [Higham and Mary 2019]. However, based on our observation in Section 4.1, if the neighborhood is too small to cover complete micro-structures, the FP errors will be biased. Thus, we follow and refine the above assumption: the FP errors in a neighborhood have mean zero when the neighborhood covers complete micro-structures.

Since the selected neighborhood contains complete micro-structures, we can apply *intensive* sampling inside the neighborhood. According to the law of large numbers (LLN) [Dekking et al. 2005], the mean of the results obtained from a large number of samples should be close to the expected value. Thus, for a center point x_c, the estimated result is represented as $\overline{f}(x_c)$, and the FP result on x_c is $\hat{f}(x_c)$:

$$\overline{f}(x_c) = Avg(\hat{f}(x_i)) \quad x_i \in selected\ \phi(x_c, r)$$

We note that:

- $\overline{f}(x_c)$ is considered more accurate than $\hat{f}(x_c)$ when the FP errors are significant and complete in the selected radius, so that the errors can be mitigated by the mean zero assumption and the law of large numbers.
- $\overline{f}(x_c)$ does not necessarily equal $p(x_c)$ for the synthesized patch $p(x)$. In the repair procedure (Section 5.2), $\overline{f}(x_c)$ provides guidance rather than mandatory constraints for repair synthesis. In fact, $p(x_c)$ is usually more accurate than $\overline{f}(x_c)$, since the fitting model for synthesizing $p(x)$ further mitigates the errors.

Estimating the variance of FP errors. The goal of this step is to estimate the variance of FP errors $Var(\varepsilon)$ within the selected neighborhood. Based on the definition of variance [Mendenhall et al. 2012] and the error's mean zero assumption, the variance can be estimated by the following equation, where $\widetilde{f}(x_i)$ is obtained via standard linear regression [Mendenhall et al. 2012] within the selected $\phi(x_c, r)$:

$$Var(\varepsilon) = \frac{\sum \varepsilon_i^2}{n-1} = \frac{\sum(\hat{f}(x_i) - f(x_i))^2}{n-1} = \frac{\sum(\hat{f}(x_i) - \widetilde{f}(x_i))^2}{n-1}$$

5 REPAIR SYNTHESIS

For a given FP program $\mathcal{P}$: $\hat{y} = \hat{f}(x)$, and an erroneous input interval $\mathcal{I}$, this section discusses how our approach synthesizes a polynomial patch $y = p(x)$, so that for $x \in \mathcal{I}$, the patch $p(x)$ is more accurate than the original program $\hat{f}(x)$. We named our approach as Aceso.

Figure 3 shows the overview of this section. Aceso works as follows:

- It starts from a given erroneous interval $\mathcal{I}$. It could be from oracle-free detection techniques or by manually given.
- It samples a set of input points (Chebyshev nodes) x_i in $\mathcal{I}$, *e.g.*, $x_1, x_2, \ldots, x_m$, which we will discuss in Section 5.1.

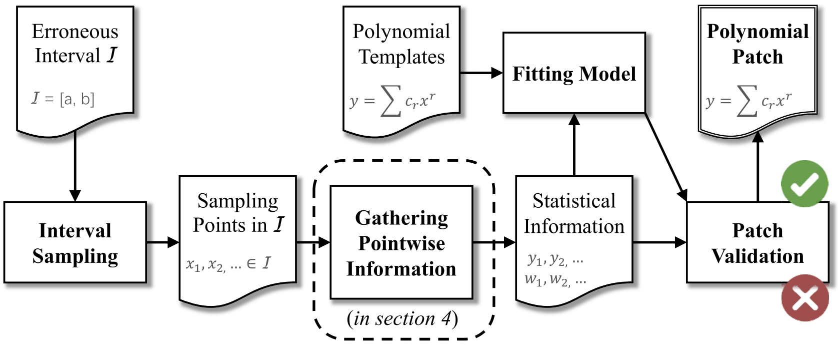


Fig. 3. Overview of ACESO's repair procedure. The repair procedure: (1) starts from an erroneous input interval I; (2) samples x_i within I; (3) collects the micro-structure and statistical information on **each** x_i (*cf.* Section 4); (4) synthesizes and validates the patch based on statistical information.

- It collects the micro-structure and statistical information on *each* point x_i, based on the insights in Section 4.
- It performs fitting with statistical information to generate a polynomial patch, which we will discuss in Section 5.2.
- It validates the patch (in oracle-free manner) to accept or deny it. Thus, it can report patch with strong confidence. We will discuss patch validation in Section 5.3.

5.1 Preliminaries before Fitting

For the erroneous interval I under repair, we need to select a set of sampling points $x_i \in I$ to describe its characteristics and guide the repair procedure. When using a polynomial to fit and approaxiamte functions, the selection of the sampling points would affect the accuracy of the polynomial [Burden and Faires 2010]. Thus, we utilize Chebyshev nodes as the sampling points within I, which is defined as

$$x_k = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2k-1}{2m}\pi\right), \quad k = 1, \ldots, m.$$

Existing literature [Burden and Faires 2010; Gil et al. 2007] shows that Chebyshev nodes provide the best[9] practical polynomial approximation.

To apply polynomial approximation, it is important to keep the variable domain of the polynomial series within $[-1, 1]$ [Newbery 1974; Oliver 1979], which helps bound FP errors. Thus, we need a general adaptation function $u(\cdot)$ to map the interval $[a, b]$ to $[-1, 1]$. We use the following equation to achieve this goal:

$$u(x) = \frac{2x - a - b}{b - a}, \text{ for } x \in [a, b] \tag{6}$$

Note that we also apply the Kahan-Babuska algorithm [Babuska 1968; Kahan 1965] to evaluate Equation (6) and to avoid potential FP errors.

5.2 Fitting with Statistical Information

This section presents our method for synthesizing polynomial approximations as repair patches. As discussed in Section 2.4, there are different forms of polynomials that can be used for the approximation, *e.g.*, power series and Chebyshev series. Therefore, in this section, we first propose

[9]The maximum possible error is minimized.

a general model suitable for various polynomial forms as repair templates. Then, we discuss how to incorporate the statistical information discussed in Section 4.3 into this model.

General fitting model. Given a polynomial form and the highest degree as the repair template, the template can be represented as

$$P_n(x) = \sum_{r=0}^{n} c_r \psi_r(u(x)), \quad \text{where} \tag{7}$$

- n is the highest degree.
- $u(x)$ is the domain adaption discussed in Section 5.1.
- The $\psi_r(u(x))$'s are terms of the polynomial template:
  - For power series, $\psi_r(u(x)) = u(x)^r$; and
  - For Chebyshev series, $\psi_r(u(x)) = \{1, u(x), 2u(x)^2 - 1, \dots\}$ are the Chebyshev polynomials, as defined in Equation (3).
- The c_r's are the unknown coefficients, for which this fitting model needs to solve.

For example, suppose the polynomial form is the power series with the highest degree $n = 2$, and the domain adaption function is $u(x) = x/4$. Then, the repair template will be $c_0 + c_1(x/4) + c_2(x/4)^2$, and the goal of the fitting model is to solve the coefficients c_0, c_1, and c_2.

A typical way to solve the c_r's is to apply ordinary least squares (OLS), which aims to *minimize* the cost function (also called squared residuals) based on a set of adapted sampling points u_i and the corresponding observations y_i [Datta 2010]:

$$\chi^2 = \sum_i \left(y_i - P_n(u(x_i))\right)^2 = \sum_i \left(y_i - \sum_r c_r \psi_r(u(x_i))\right)^2.$$

According to the Gauss-Markov theorem [Eaton 1983], in OLS, the errors on the observations having *equal variances* is a necessary condition for the solved c_r to be the *best linear unbiased estimator* (BLUE). However, FP errors vary at different points and have different variances. For FP programs, the error on different points can differ by orders of magnitude [Yi et al. 2019; Zou et al. 2015]. Thus, the OLS model is unsuited for this scenario. To this end, we adopt a generalization of OLS, *weighted least squares (WLS)*, whose fitting model can handle this situation. WLS has some additional scale factors, *i.e.*, the weights w_i, included in the fitting model [Datta 2010]. It minimizes the cost function with the weights w_i:

$$\chi^2 = \sum_i w_i \left(y_i - P_n(u(x_i))\right)^2 = \sum_i w_i \left(y_i - \sum_r c_r \psi_r(u(x_i))\right)^2. \tag{8}$$

The weights can be used to indicate the quality of each observation y_i. Specifically, if each weight w_i is equal to the reciprocal of the variance of the observation σ_i^2, *i.e.*, $w_i = 1/\sigma_i^2$, the solved c_i's are the best linear unbiased estimator (BLUE) [Datta 2010].

Following the typical routine [Ryan 2008], the coefficients $c = [c_0, c_2, \dots, c_n]^T$ can be solved with

$$c = (X^T W X)^{-1} X^T W y, \tag{9}$$

where $X_{ij} = \psi_j(u(x_i))$. For example, for power series,[10] we have

$$X = \begin{bmatrix} 1 & u(x_0) & u(x_0)^2 & \dots \\ 1 & u(x_1) & u(x_1)^2 & \dots \\ 1 & u(x_2) & u(x_2)^2 & \dots \\ \dots & \dots & \dots & \end{bmatrix} \quad W = diag(w_0, \dots, w_n) = \begin{bmatrix} w_0 & 0 & 0 & \dots \\ 0 & w_1 & 0 & \dots \\ 0 & 0 & w_2 & \dots \\ \dots & \dots & \dots & \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \dots \end{bmatrix}$$

[10] X, W, and y are similarly defined for Chebyshev series.

Statistical information for the fitting model. Based on Equation (9), to solve the coefficients c, we need (1) *terms matrix* X, (2) *weight vector* w, and (3) *observation vector* y. As for the X, it can be obtained from the adapted sampling points u_i and the polynomial template $\psi_r(u_i)$ directly. So, the important question is, how to obtain the weight w_i and the observation y_i?

The values of w and y are *critical* to the fitting quality, *i.e.*, the accuracy of the polynomial patch. Ideally, the y_i on each x_i should be the oracle result $f(x_i)$, and the w_i should be the the reciprocal of the variance on each observation y_i.

Without an oracle, neither the oracle result $f(x)$ nor the variance can be obtained. Thus, we apply the statistical information based on micro-structure for each sampling point $x_i \in \mathcal{I}$ to feed the fitting model:

$$w_i = \frac{1}{Var(\varepsilon_i)} \qquad y_i = \overline{f}(x_i) \tag{10}$$

where $\overline{f}(x_i)$ is the estimated mean of FP results, and $Var(\varepsilon_i)$ is the estimated variance of FP error (*cf.* Section 4.3).

In the fitting model, both $\overline{f}(x_i)$ and $Var(\varepsilon_i)$ from the micro-structure play critical roles. If the micro-structure is unavailable, the only available estimation on $f(x_i)$ would be the direct FP result $\hat{f}(x_i)$, which is far less accurate than $\overline{f}(x_i)$. Furthermore, there would be no available estimation on the observation variance, making the fitting result significantly affected by erroneous (outlier) points, and thus less accurate.

5.3 Patch Validation

Before reporting the synthesized polynomial patch $p(x)$ to the user, it is important to validate $p(x)$, and to only report *superior* patch (more accurate on the entire interval $\mathcal{I}$). This section proposes an oracle-free validation method based on micro-structures.

We notice that, in the erroenous interval $\mathcal{I}$, the FP errors may vary in magnitude. For example, some points may still be (relatively) accurate and only have a few hundreds of ulp error (approx. 10^{-13} relative error), while some points may gradually decay to 0.0, NaN, or Inf, thus having 10^{15} ulp error (approx. 1.0 relative error).

Hence, it is important to validate the patch $p(x)$ on the *entire* interval $\mathcal{I}$: (1) $p(x)$ should be more accurate on the decayed area (the FP result becomes NaN or has only few valid digits); and (2) $p(x)$ should also be more accurate on relatively stable areas, which is challenging for the patch — therefore, we need to validate it.

To validate $p(x)$ without an oracle, micro-structure again plays a critical role. We propose an oracle-free method to validate $p(x)$ based on local sampling (*cf.* Section 4):

(1) Generate a set of test inputs $t_1, t_2, \ldots, t_k$ within the erroneous interval $\mathcal{I}$.
(2) Select a suitable radius and neighborhood ϕ_i for each input t_i as discussed in Section 4.2.
(3) Measure the range $\langle a_i, b_i \rangle$ on the neighborhood ϕ_i by the sampling methods in Section 4.2, where $a_i = \min(\hat{f}(x_{ij}))$, $b_i = \max(\hat{f}(x_{ij}))$, and $x_{i0}, \ldots, x_{in}$ are sample points within ϕ_i.
(4) Validate whether the polynomial patch $p(x)$ on t_i satisfies $a_i \leq p(t_i) \leq b_i$.

If the validation fails on the last step, the patch is marked as inaccurate and rejected. Possible reasons for synthesizing an inaccurate patch are:

- The parameters, such as the degree of a polynomial template, the number of Chebyshev nodes, *etc.*, are inappropriate (too small). Thus, one may repeat the repair process with higher degrees or more nodes to synthesize more accurate polynomial patches.
- The ideal function $f(x)$ itself is discontinuous or ill-conditioned on the interval $\mathcal{I}$, thus beyond the capability of polynomial patches and our repair approach.

- The original FP program $\hat{f}(x)$ does not involve significant FP errors, thus the patch cannot be more accurate.

Recall the example that we illustrated in Section 3, where the parameters are set to be very small for simplicity. This synthesized patch will in fact fail the validation process.

However, It is still interesting to evaluate this patch $p(x)$ and the original program $\hat{f}(x)$ with the oracle to see what has occurred. We notice that $p(x)$ reduces the error on decayed area, *e.g.*, on $x = 10^{-10}$. It was rejected because it is less accurate on other inputs, *e.g.*, $x = 10^{-4}$ (the invalid digits are highlighted in bold):

- $p(10^{-4})$ = `0.50001250`**`107390649`** `// p(x) increases the error on relatively stable area.`
- $\hat{f}(10^{-4})$ = `0.50001250083`**`253623`**
- $p(10^{-10})$ = `0.4999999999`**`99915362`** `// p(x) reduces the error on decayed area.`
- $\hat{f}(10^{-10})$ = `0.5000000`**`4137018550`**

This example illustrates the criterion used by patch validation — it requires the patch to be more accurate on the *entire* interval $\mathcal{I}$. Even if the maximum error is reduced by the patch, the patch could still be inacurate and should not be reported. Thus, after passing the validation process, the patch can be reported with strong confidence.

6 EVALUATION

This section details the evaluation of our technique Aceso[11].

6.1 Evaluation Setup

Benchmarks. We collected 36 commonly-used univariate functions as well as their variants as our benchmarks, including

- *17 functions involve library calls from GSL and ALGLIB*[12], thus the oracles are difficult to construct (*cf.* Section 3), and can hardly be repaired by oracle-dependent approaches [Panchekha et al. 2015; Yi et al. 2019]. These functions are the variants of classical functions from the literature [Hamming 2012].
- *15 univariate functions from related FP analysis/repair work* [Panchekha et al. 2015; Solovyev et al. 2018; Wang et al. 2019].
- *4 additional error-free functions*, which we have manually checked to make sure that they are accurate. We use this control group to evaluate the validity of the validation stage (*cf.* Section 5.3), *i.e.*, whether the validation stage can correctly reject less accurate patches.

Details on these functions are shown in Table 2. The erroneous intervals are collected based on Atomu [Zou et al. 2020], which is an open-source oracle-free error detection tool. The erroneous interval is [2.7082818285, 2.7282818285] for function 13 and 14; [0.99, 1.01] for function s15; [1.5607963268, 1.5807963268] for function s4; and [-0.01, 0.01] for all the remaining functions. We also set the target interval as [-0.01, 0.01] for the error-free functions c1 to c4 to make Aceso work under similar criterion.

Error measurements. As discussed in Section 2.2, we use both absolute error (Err_{abs}) and relative error (Err_{rel}) to measure FP errors.

[11]The repository at https://bitbucket.org/FP-Aceso/aceso/ contains all our code and experimental data to facilitate the reproduction of our results.

[12]Both libraries are mature and actively maintained numerical projects, and with numerous manually crafted polynomials. Both libraries are implemented with `double`.

Table 2. Details of the evaluation subjects.

(a) This group (1-17) includes subjects whose oracles are hard to get due to the involved library calls.

No.	ID	Function	Involved Library Call *
1	exp_bI	$f(x) = \frac{e^{bI0(x)}-1}{x}$	`bessel_I0(x)`
2	bJ_sin	$f(x) = \frac{1-bJ0(x)}{\sin(x)}$	`bessel_J0(x)`
3	di_tan	$f(x) = \frac{1}{dilog(x)} - \frac{1}{\tan(x)}$	`dilog(x)`
4	log_erf	$f(x) = \frac{\log(1-erf(x))}{\log(1+x)}$	`erf(x)`
5	acos_fd	$f(x) = \frac{\arccos(x)^2-3fd1(x)}{x}$	`fermi_dirac_1(x)`
6	ei	$f(x) = \frac{\sin(ei(x))}{\cos(x)-e^x}$	`erf_inv(x)`
7	Q1_W	$f(x) = \frac{1+Q1(x)}{W(x)^2}$	`lambert_W0(x)`, `legendre_Q1(x)`
8	bj_tan	$f(x) = \frac{1-bj0(x)}{x\tan(x)}$	`bessel_j0(x)`
9	Si_tan	$f(x) = \frac{Si(x)-\tan(x)}{x^3}$	`Si(x)`
10	by_psi	$f(x) = y0(x)^2 - psi1(x)$	`bessel_y0(x)`, `psi_1(x)`
11	fdm_log	$f(x) = \frac{2fdm(x)-1}{\log(1+x)}$	`fermi_dirac_m1(x)`
12	eQ_sqrt	$f(x) = \frac{2eQ(x)-\sqrt{1+x}}{x}$	`erf_Q(x)`
13	W_var	$f(x) = \frac{W(x)-1}{W(x)^2-1}$	`lambert_W0(x)`
14	W_log	$f(x) = \frac{W(x)-1}{W(x)\log(x)-1}$	`lambert_W0(x)`
15	pow_df	$f(x) = (1+di(x))^{1/x}$	`dawsonintegral(x)` (ALGLIB)
16	chi_ci	$f(x) = \frac{chi(x)-ci(x)}{x^2}$	`hyperboliccosineintegral(x)`, `cosineintegral(x)` (ALGLIB)
17	fc_bj	$f(x) = 1/FC(x) + bj1(x) - \sin(x)/x^2$	`fresnelintegral(x)` (ALGLIB), `bessel_j1(x)`

* Basic functions from GNU C Library, *e.g.*, sin, exp, log, *etc.*, are omitted in the column *Involved Library Call*, as they are all supported by MPFR.

(b) This group (s1-s15) includes subjects from related FP analysis/repair work.

No.	ID	Function	No.	ID	Function
s1	cos_x2	$f(x) = \frac{1-\cos(x)}{x^2}$	s9	x_tan	$f(x) = \frac{1}{x} - \frac{1}{\tan(x)}$
s2	exp_x	$f(x) = \frac{(e^x-2)+e^{-x}}{x}$	s10	log_log	$f(x) = \frac{\log(1-x)}{\log(1+x)}$
s3	cos_sin	$f(x) = \frac{1-\cos(x)}{\sin(x)}$	s11	log_x	$f(x) = \log(\frac{1-x}{1+x})$
s4	sin_sin	$f(x) = \sin(x+\varepsilon) - \sin(x)$	s12	sqrt_exp	$f(x) = \sqrt{\frac{e^{2x}-1}{e^x-1}}$
s5	tan_tan	$f(x) = \tan(x+\varepsilon) - \tan(x)$	s13	sin_tan	$f(x) = \frac{x-\sin(x)}{x-\tan(x)}$
s6	cos_cos	$f(x) = \cos(x+\varepsilon) - \cos(x)$	s14	exp_x	$f(x) = \frac{e^x-1}{x}$
s7	exp_exp	$f(x) = e^x - e^{-x}$	s15	x_x2	$f(x) = \frac{x-1}{x^2-1}$
s8	exp_1	$f(x) = e^x - 1$			

** $\varepsilon = 10^{-6}$ for function s4, s5, and s6.

(c) This group (c1-c4) is the control group that consists of error-free functions.

No.	ID	Function	Involved Library Call
c1	control_1	$f(x) = e^{\sin(x)\cos(x)}$	-
c2	control_2	$f(x) = \sqrt{e^{x+1}}$	-
c3	control_3	$f(x) = \frac{\sin(2x)}{\sin(x)}$	-
c4	control_4	$f(x) = \sqrt{\sin(x)^2 + bJ0(x)^2}$	`bessel_J0(x)`

```
// Coefficients in GSL
// for computing LambertW
double coefs[12] = {
 -1.0000000000000000e+00,
  2.3316439815971242e+00,
 -1.8121878856393635e+00,
  1.9366311144923598e+00,
 -2.3535512018816145e+00,
  3.0668589010506319e+00,
 -4.1753356002581771e+00,
  5.8580237298747741e+00,
 -8.4010322175239774e+00,
  1.2250753501314460e+01,
 -1.8100697012472443e+01,
  2.7029044799010562e+01,
};
// Power series based on coefs.
double v = series_eval(x, coefs);
...
```

```
// Our crafted code for the oracle of LambertW.
mpreal p = sqrt(2.0*exp(1.0));
mpreal coefs[20] = {
  -1,
  p,                    // mpreal: 2.33164398159712420336353б...
  pow(p,2)*-1 / 3,      // mpreal:-1.81218788563936349024019l...
  pow(p,3)*11 / 72,
  pow(p,4)*-43 / 540,
  pow(p,5)*769 / 17280,
  pow(p,6)*-221 / 8505,
  pow(p,7)*680863 / 43545600,
  pow(p,8)*-1963 / 204120,
  pow(p,9)*mpreal("226287557") / mpreal("37623398400"),
  pow(p,10)*mpreal("-5776369") / mpreal("1515591000"),
  pow(p,11)*mpreal("169709463197") / mpreal("69528040243200"),
  pow(p,12)*mpreal("-1118511313") / mpreal("709296588000"),
  pow(p,13)*mpreal("667874164916771") / mpreal("650782456676352000"),
  pow(p,14)*mpreal("-500525573") / mpreal("744761417400"),
  ... }; // We omit further terms for simplicity.
```

Fig. 4. Comparison between the partial source code in GSL and our manually crafted oracle for the LambertW function. Left: coefficients in GSL; Right: coefficients in our manually crafted oracle. The coefficients on the right have higher precision and more terms for computing rigorous oracles. We have to carefully examine the numerical literature [Corless et al. 1996; Fukushima 2013] to craft the code.

Settings for Aceso. We set the repair template (as discussed in Section 5.2) to be Chebyshev series with the highest degree of six for all 36 evaluation subjects. The degree of the polynomial patch is a tunable parameter in Aceso.

For interval sampling (Section 5.1), we sample $m = 1{,}000$ Chebyshev nodes within the interval under repair $\mathcal{I}$. For local sampling on each node, we generate 100 different radii for radius selection (Section 4.2) and gathering statistical information (Section 4.3).

Ground truth. Aceso does not require oracles or high-precision results during its entire process (including patch validation). However, to evaluate the validity and accuracy of the repair results, we need to construct and provide the ground truth.

As our benchmarks involve various library calls from GSL and ALGLIB, we have to apply the high-precision implementations of such calls to construct rigorous oracles. However, many of the involved calls do not have high-precision implementations in the MPFR infrastructure.[13] Thus, we carefully crafted the oracles leveraging numerical expertise to ensure their correctness.

For example, the `LambertW` function is used in our benchmarks, and MPFR does not have its high-precision implementation. Thus, to construct its oracle, we inspected the source code of `LambertW` first. Figure 4 (left) shows a power series inside the `LambertW` function. As mentioned in Section 3, to obtain accurate oracles, we need to introduce *more terms* of this power series, rather than simply replacing the FP type. By referring the numerical literature [Corless et al. 1996; Fukushima 2013], we crafted the code in Figure 4 (right) as `LambertW`'s oracle.

We note that constructing oracles for code like the one in Figure 4 relies on strong numerical expertise and manual effort, a challenging demand for typical application developers who simply utilize numerical libraries like GSL. Therefore, our oracle-free approach Aceso provides a practical, effective solution to this paradoxical difficulty, *i.e.*, one needs accurate oracles to apply *oracle-dependent* repair techniques, but accurate oracles are difficult to construct.

6.2 Evaluation Results

This section presents our evaluation results to demonstrate Aceso's effectiveness and scalability. In particular, our evaluation addresses the following research questions (RQs):

[13] Actually, we cannot assume/take for granted that MPFR supports every library call in practice.

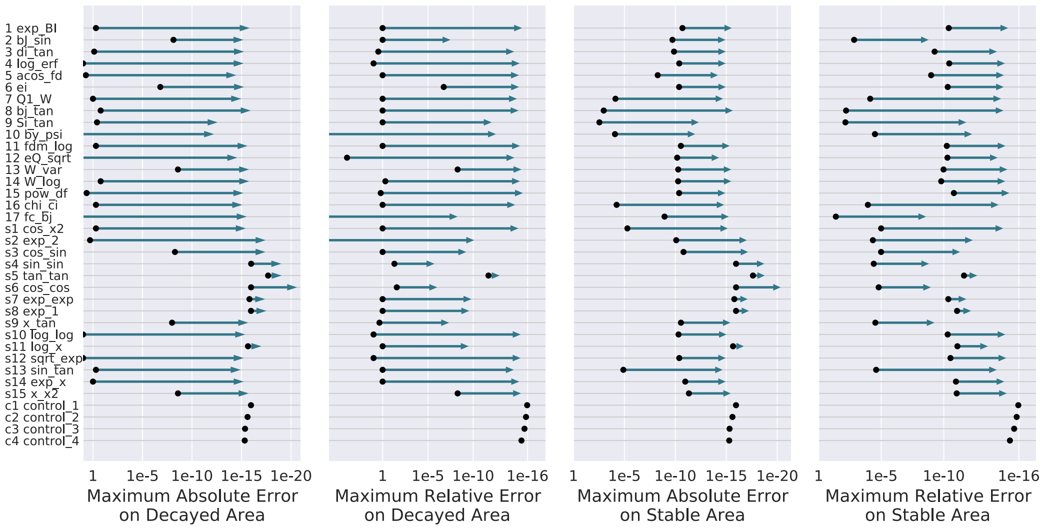


Fig. 5. Accuracy Improvements by Aceso. The accuracy improvements are reported *w.r.t.* both **absolute** and **relative** error, and on both **decayed** and **(relatively) stable** area of the erroneous input interval $\mathcal{I}$. The black dots represent the errors on original functions, and the arrowheads represent the errors on Aceso's results. Smaller error (right side) is better.

- **RQ1**: How effective is Aceso in repairing FP errors?
- **RQ2**: How significantly does micro-structure contribute to Aceso's effectiveness?
- **RQ3**: How scalable is Aceso?

6.2.1 RQ1: How effective is Aceso in repairing FP errors? Figure 5 shows the improvements in maximum error by Aceso *w.r.t.* both absolute and relative error. The maximum error for each function is collected by intensively sampling points in the target interval $\mathcal{I}$.

For the two "Decayed Area" subfigures, the sampling points are collected based on "*random in magnitude*", which can sample points to cover the most decayed and erroneous areas. For example, function s12 decays and returns NaN on inputs $|x| < 10^{-16}$; "*uniform random sampling*" in [-0.01, 0.01] can rarely generate such decayed points. We observe that the "Decayed Area" subfigures visualize the maximum error on $\mathcal{I}$, which is largely determined by the decayed points.

For the two "Stable Area" subfigures, the sampling points are collected based on "*random in value*", which focuses on those relatively accurate and stable points (in contrast to decayed points). Since we expect the patches reported by Aceso to be *superior* (more accurate on the entire interval, including both decayed and stable areas), we use these two subfigures to show its performance on the relatively stable areas.

As shown in Figure 5, on erroneous FP programs, *i.e.*, functions 1 to 17, and s1 to s15, the reported patches by Aceso are all superior to the original programs, *w.r.t.* both absolute and relative errors. Aceso can reduce FP errors by *orders of magnitude* on most functions. Error reduction occurs on both stable and decayed areas, demonstrating the high quality of the synthesized patches — the maximum absolute errors are smaller than 10^{-14} for all patches, while maximum relative errors are also significantly reduced.

We note that Aceso, being a dynamic approach, is agnostic to the program constructs used and can repair FP errors in quite general code. On functions 1 to 17, although the library calls and their underlying precision-related code (*cf.* Section 3) make it challenging to construct oracles, Aceso can still significantly reduce their FP errors. This is because: (1) Aceso is oracle-free and never tries to construct oracles, and (2) ACESO is a black-box analysis, thus it is agnostic to program constructs, *e.g.*, library calls, precision-related code and pointers, making it generally applicable.

Patches synthesized and validated by ACESO are *superior* to the original programs, with FP errors reduced by orders of magnitude.

On functions c1 to c4, ACESO also tries to synthesize patches on the target interval $\mathcal{I}$. Since all of these functions are accurate, we expect that it would be impossible for a polynomial to become more accurate. ACESO, without such information, is able to correctly reject the less accurate patches in its validation stage (*cf.* Section 5.3). This result shows the effectiveness of the validation stage, which is vital for ACESO. Without validation, the less accurate patches would be reported to the users. Since it is difficult for users to inspect the accuracy of reported patches, reporting such inferior patches would significantly diminish the practical value of a technique.

Patch validation is effective and vital for ACESO, which enables ACESO to report only superior patches with confidence.

6.2.2 *RQ2: How significantly does micro-structure contribute to ACESO's effectiveness?*

Micro-structure (and the corresponding statistical information) is critical for ACESO's effectiveness in two aspects:

- $\overline{f}(x_i)$ and $Var(\varepsilon_i)$ for synthesizing the patch (*cf.* Section 5.2), and
- $\langle a_i, b_i \rangle$ for validating the patch (*cf.* Section 5.3).

This research question is designed to evaluate whether and how significantly the micro-structure contributes to the effectiveness of ACESO. To this end, we repeat the repair procedure in Section 5 without the aforementioned information:

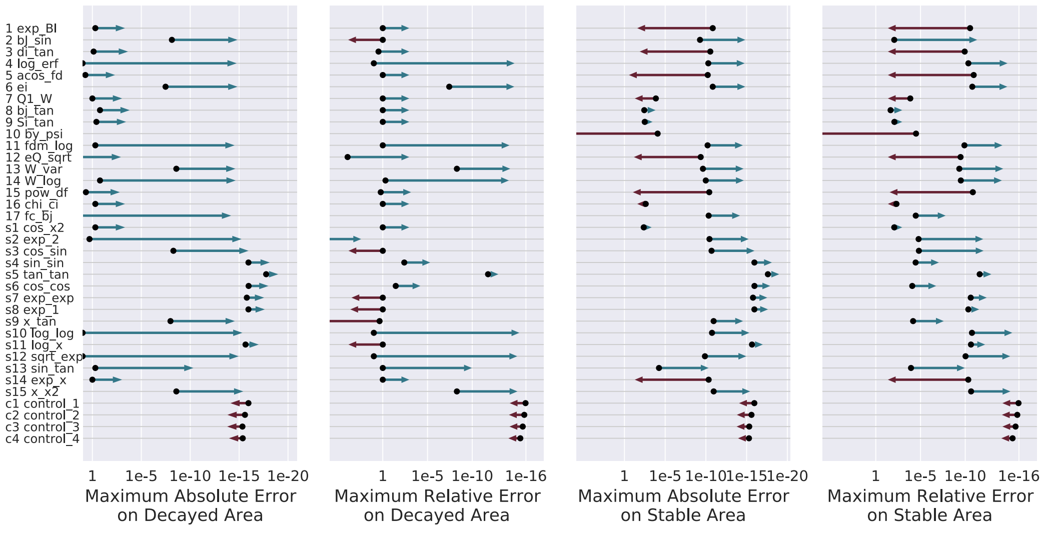


Fig. 6. Error when Fitting **without Micro-Structure**. (green: accuracy improvement; red: accuracy loss). Majority of the patches become *worse* than the original functions *w.r.t.* at least one criterion on **decayed** or **stable** area of the erroneous interval $\mathcal{I}$.

- Using the direct FP results $\hat{f}(x)$ to replace $\overline{f}(x)$;
- Removing the weights (*i.e.*, setting $w = 1$ for all points) as $Var(\varepsilon)$ is no longer available;
- Disabling the validation stage as $\langle a_i, b_i \rangle$ is no longer available, thus all patches are reported.

The results in Figure 6 show that: (1) all patches in Figure 6 are less accurate than the Aceso-synthesized patches (Figure 5); and (2) numerous patches become *worse* than the original programs *w.r.t.* at least one criterion (on decayed or stables area) — the errors increase by orders of magnitude in these patches. Thus, such patches are not *superior* to the original programs on the entire $\mathcal{I}$ and should not be reported.

Furthermore, without micro-structures, it becomes *impossible to validate and distinguish* the patches. We notice that some patches do have better performance than the original programs. However, one cannot distinguish such superior patches from other inferior patches, and has to report all (or none) of them to users. Thus, it is infeasible to simply repair FP errors by fitting with FP results without micro-structures.

Micro-structure is *critical* for Aceso's effectiveness, for both patch synthesis and validation.

Further ablation study 1. We have conducted a similar evaluation where one fits directly with the FP results $\hat{f}(x)$, with the weights $1/Var(\varepsilon)$ from statistical information, to check whether the correctly estimated weights alone could yield comparable results with Aceso. The detailed results, which we omit due to space constraints, are similar to those shown in Figure 6: (1) numerous patches are *worse* than the original programs; and (2) it cannot validate the patches, thus inferior patches would be reported to users. Therefore, our results show that micro-structure (and the gathered statistical information) is critical for Aceso's effectiveness.

Further ablation study 2. As discussed in Section 3 and in Figure 4's example, due to library calls and their precision-related code, it is difficult for typical application developers to construct accurate oracles to apply oracle-dependent repair techniques. On the other hand, it may still be feasible for them to ignore precision-related code and simply lift the FP type (*e.g.*, from `double` to MPFR) to obtain a *type-only oracle*.

We thus conduct a case study to check whether such *type-only oracles* could be effective for repairing FP errors. In particular, we use function 14 `W_log` as the subject since the details of the invoked library call `lambert_W0` are shown in Figure 4. To obtain `W_log`'s type-only oracle, we lift its FP type from `double` to MPFR (1,024 bits precision) in both the user's code (`W_log`) and the library code (inside `lambert_W0`).

The evaluation result in Figure 7 shows that the errors on the original `W_log` and its type-only oracle are of *the same order of magnitude* — the maximum absolute and relative errors are around 10^{-1}. This result confirms that the type-only oracle is as erroneous as the original program. In fact, if we had used the type-only oracle as the baseline, the maximum "error" on the original program would have only been around 10^{-13}, which is clearly incorrect and misleading.

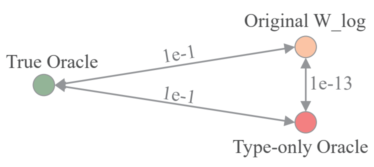


Fig. 7. Maximum errors/gaps between the original function, the accurate oracle, and the type-only oracle for `W_log`.

Thus, if a user applies an existing oracle-dependent repair technique with the type-only oracle, it may diminish the small gap between the original function and the type-only oracle. However, the most significant gap/error between the true oracle and the type-only oracle is unavoidable, and the repair result would still be erroneous. Therefore, simply lifting the FP type cannot produce accurate, effective oracles for repairing FP errors.

Table 3. Runtime on Aceso.

Runtime for Aceso to repair each function (in seconds)		
Min.	Max.	Avg.
0.46	2.55	0.73

Table 4. Runtime on patches.

	Runtime for original and patched functions (in nanoseconds, *i.e.*, 10^{-9} seconds)		
	Min.	Max.	Avg.
Origin	2.12	260.48	36.47
Patched	8.87	9.44	9.09

6.2.3 RQ3: How scalable is Aceso? This section discusses the cost of Aceso from two aspects: (1) cost for Aceso to repair functions, and (2) performance of the patches.

Since Aceso does not rely on oracles or high-precision calculation, it is expected to be efficient. Table 3 shows the cost for Aceso to repair each function, confirming Aceso's efficiency. To measure the cost of Aceso, we run it on each function 10 times and compute the average cost of it on each function. The results in Table 3 show that Aceso takes on average 0.73 seconds to synthesize and validate the patch for each function, and thus is highly efficient.

To measure the performance of the synthesized patches, we run the original functions and the patched functions on 1,000,000 random inputs (within the target interval $\mathcal{I}$) to collect the average execution times. Table 4 lists the execution times for the original functions and patched functions. The results show that the patched functions are highly efficient — on average execution time, the patched functions are even faster than the original functions.

Note that each of the synthesized patches is a polynomial with fixed terms, thus has *constant* time complexity. In other words, no matter how complex the original program is, the patch always has a similar execution time.

These results show that Aceso is a lightweight technique, in terms of usability (by being oracle-free), repair cost (by being efficient), and patch performance (by being fast and having constant complexity).

Aceso costs 0.73 seconds on average to synthesize a patch. The synthesized patches are highly efficient, and even faster than the original programs.

7 DISCUSSIONS

7.1 Error Analysis of Polynomial Patch

This section discusses the possible error between the generated patch $p(x)$ and the oracle $f(x)$. We divide the error into two parts: (1) *Internal error*: the FP error when evaluating $p(x)$, *i.e.*, $|\hat{p}(x)-p(x)|$; (2) *External error*: the approximation error between the polynomial and oracle, *i.e.*, $|p(x) - f(x)|$.

Internal error. For evaluating polynomials, our patch applies two methods to reduce error:

- Using domain adaptation to ensure that the adapted variable $u = u(x)$ satisfies $|u| \leq 1$ (*cf.* Section 5.1).
- Using the widely adapted evaluation scheme [Newbery 1974; Oliver 1979], *i.e.*, *Horner method* or *Clenshaw method*, to evaluate power series and Chebyshev series correspondingly.

The evaluation error on such schemes has been extensively discussed and proved [Müller 1983; Newbery 1974; Oliver 1979] to be insignificant as long as the adapted variable satisfies $|u| \leq 1$. Since Aceso ensures $|u| \leq 1$ by using domain adaptation, it can guarantee that the generated patch does not have significant FP errors (*i.e.*, internal errors).

External error. In Section 5, we use statistical information on Chebyshev nodes to synthesize polynomial patches on the erroneous interval $\mathcal{I}$. While on each Chebyshev node, the estimated mean is close to the oracle by the law of large numbers, it is unknown whether the polynomial patch $p(x)$ is accurate on an arbitrary point $x \in \mathcal{I}$.

Next, we establish a correlation between the error on Chebyshev nodes and on an arbitrary point within $\mathcal{I}$. More formally, with the following definitions and notations:

- $x_i = x_1, \dots, x_m$ are m Chebyshev nodes within the interval $\mathcal{I} = [a, b]$ (*cf.* Section 5.1),
- $p(x)$ is the synthesized polynomial,
- $f(x)$ is the oracle result, and assume $f(x)$ is a smooth function, and
- σ is the upper bound of errors on nodes x_i, then

  If $\forall i, |f(x_i) - p(x_i)| \leq \sigma$, then $\forall x \in \mathcal{I}, \exists \xi_x \in \mathcal{I}$ such that

$$\left| f(x) - p(x) \right| \leq \frac{(b-a)^m}{2^{2m-1} m!} f^{(m)}(\xi_x) + \sigma \left(\frac{2\ln(2m-1)+2}{\pi} + \pi \right) \tag{11}$$

The proof for Equation (11) is in the Appendix (in the seperate *supplementary material* file). In Equation (11)

- The first term converges extremely fast to zero. For example, in our setting, we have $m = 1{,}000$, and suppose $(b-a) = 10$, which is significantly larger than the interval size in practice, we have

$$\frac{(b-a)^m}{2^{2m-1} m!} \approx 10^{-2{,}170}, \quad \text{for } m = 1{,}000, (b-a) = 10$$

  which means that the higher-order derivative $f^{(m)}(\xi_x)$ needs to be at least $10^{2{,}100}$ to make the first term significant, which is almost impossible in practice.
- The second term determines the error bound, which increases slowly with larger m. For example,

$$\sigma \left(\frac{2\ln(2m-1)+2}{\pi} + \pi \right) = 8.62\sigma, \quad \text{for } m = 1{,}000$$

  which means that the maximum possible error for $p(x)$ on $\mathcal{I}$ will not exceed 9 times of the maximum error on the sampled Chebyshev nodes in this case.

Equation (11) proves that the error on the entire interval $\mathcal{I}$ is strictly bounded by the error (σ) on Chebyshev nodes. This result also implies that, for future work, improving the accuracy on Chebyshev nodes (reducing σ) is an effective way to improve the accuracy on the entire interval.

7.2 Assumptions, Generality, and Capability of Aceso

Assumptions. Our work has several assumptions, which we further elaborate. Note that these assumptions are required for the interval under repair ($\mathcal{I}$), not for the entire input domain:

- The program should be *continuous* in the interval $\mathcal{I}$. Since significant FP errors only occur on an extremely small portion of inputs [Bao and Zhang 2013], $\mathcal{I}$ can usually be small. Thus, this assumption easily holds in practice. If the program is discontinuous in $\mathcal{I}$, we can still split $\mathcal{I}$ into continuous sub-intervals and repair each separately.
- The program should not be *ill-conditioned* in the interval $\mathcal{I}$. Ill-conditioned problems, *e.g.*, functions with extremely large derivatives, are theoretically very difficult to cure their inaccuracies [Fu et al. 2015]. Thus, fixing such errors is beyond the scope of our work.

Generality. As discussed in Section 4.1, we expect the existence of micro-structures to be a general property of FP errors, since they are caused by changes in the rounding directions. Aceso, as a dynamic approach and viewing input FP programs as blackboxes (*i.e.*, only observing their results), is general and has some key advantages: (1) it is independent of oracles (*i.e.*, oracle-free), and

(2) it is agnostic to input program constructs (*e.g.*, library calls, loops, conditionals, castings, and pointers). It is also efficient and effective in synthesizing accurate and performant patches as shown in Section 6. Thus, we envision Aceso to be generally applicable in settings where oracle-free error detection like Atomu can be utilized.

It is worth mentioning that we collect univariate functions as our benckmark following the existing work [Yi et al. 2019; Zou et al. 2020]. We believe the micro-structure of FP errors will still exist in multivariate FP programs, since it is a general property caused by the change of rounding direction in FP operations. In future work, it would be interesting to explore: (1) how to generalize the neighborhood for observing micro-structure to higher dimensions, and (2) how to generalize the polynomial templates *e.g.*, Chebyshev series, for multiple variables.

Capability. The considered scenario for Aceso is quite challenging, yet common in practice. The only available information to Aceso is an erroneous numerical program without an oracle. Hence, in general, Aceso cannot provide provable accuracy guarantees on the patch. However, during the validation process, a patch would only pass if, on *all* the sampled inputs t_i, the patched value $p(t_i)$ falls into the range $[a_i, b_i]$ (*cf.* Section 5.3). Notice that a_i and b_i are actual values that the original program could produce in t_i's tiny neighbourhood.

Thus, although without provable guarantees, we have strong confidence that the validated patch is superior to the original program in the entire interval $\mathcal{I}$, which our empirical results have confirmed (*cf.* Section 6.2).

8 RELATED WORK

This section surveys several threads of closely-related work, which we discuss below.

Obtaining oracles of FP programs. High-precision shadow execution is a dynamic analysis that performs side-by-side execution with high-precision types for analyzing FP errors. In this category, FpDebug [Benz et al. 2012] and Herbgrind [Sanchez-Stern et al. 2018] are based on MPFR [Fousse et al. 2007] and binary analysis tool Valgrind [Nethercote and Seward 2007], while FPSanitizer [Chowdhary et al. 2020] and PFPSanitizer [Chowdhary and Nagarakatte 2021] reduce the runtime overhead significantly via compile-time instrumentation and parallel execution. Such techniques are effective for analyzing FP errors, however, they are not suited for obtaining oracles of FP programs. They assume that high-precision executions match the oracle results, which in turn assumes that the semantics of FP programs matches their underlying mathematical functions. However, this assumption does not hold on precision-specific operations [Wang et al. 2016] and precision-related code [Zou et al. 2020]. Thus, it remains a significant challenge to obtain oracles for FP programs. Even in recent work on detecting FP errors, the oracle results have to be obtained by manually modifying source code [Guo and Rubio-González 2020] or comparing different implementations [Vanover et al. 2020].

Detecting FP errors. Several approaches have been proposed to detect inputs that trigger significant FP errors. BGRT [Chiang et al. 2014] is based on heuristic binary search, LSGA [Zou et al. 2015] is based on a genetic algorithm, while EAGT [Yi et al. 2017] and DEMC [Yi et al. 2019] are based on differential evolution and Markov Chain Monte Carlo methods. All these approaches rely on the existence of oracles.

Recently, several *oracle-free* approaches have been proposed for detecting such error-triggering inputs. Atomu [Zou et al. 2020] searches for significant atomic conditions, which are highly connected with FP errors and can be obtained based on runtime information. FPGen [Guo and Rubio-González 2020] injects checks to detect large precision loss and cancellation, and applies symbolic execution to generate suspicious inputs.

In contrast, our work proposes an *oracle-free* approach, Aceso, to repairing FP errors, rather than detecting them. Aceso is a downstream task that benefits from oracle-free error detection techniques because it needs information on where significant errors exist. Without an oracle-free error detection technique, we would have to construct an oracle (expensive in both development cost and runtime cost [Zou et al. 2020]) for finding significant errors. Thus, we could also apply the oracle for repair. Oracle-free error detection enables our work and improves its practical utility.

Repairing FP errors. Several approaches aim to repair FP errors. Herbie [Panchekha et al. 2015] and Salsa [Damouche and Martel 2017] apply mathematical equivalent transformations to improve the accuracy of FP programs. Herbie is a dynamic tool, which needs oracles on sample points to judge and guide the transformations. Salsa is a *static* tool, which applies abstract interpretation to judge and guide transformations, and cannot deal with complex data structures, such as function pointers and library calls. AutoRNP [Yi et al. 2019] is a repair tool. It applies linear or quadratic approximation to repair the localized erroneous intervals, and relies heavily on the oracle to synthesize such approximations. Lim et al. [Lim et al. 2021; Lim and Nagarakatte 2021a,b] have proposed approaches to generate math library functions for different FP representations, such as Bfloat16 and 32-bit float, which rely on oracles during the synthesis procedure and guarantee errors to be less than 0.5 ulp (correctly rounded).

The goal of our approach is to *dynamically* repair FP errors *without relying on oracles*. Since obtaining an oracle is difficult and costly, we provide a lightweight approach to synthesizing polynomial approximations to reduce FP errors for wider applicability.

9 CONCLUSION

We have introduced a novel, effective, oracle-free approach to repairing significant FP errors in numerical programs. Our key insight is the concept of micro-structure that FP errors can be observed with structural patterns under a certain scope. Micro-structures allow one to gather rich statistical information about FP errors, which we have shown how to effectively leverage to guide patch synthesis and validation. We have designed and realized our general approach and evaluated it on 36 commonly-used numerical programs. Evaluation results have demonstrated that our approach is highly effective — it can synthesize patches quickly (*i.e.*, under one second), the synthesized patches drastically reduce FP errors (*i.e.*, by orders of magnitude), and the patches are efficient (*i.e.*, having better or similar runtime performance as the original programs).

ACKNOWLEDGMENTS

We thank the anonymous SPLASH/OOPSLA reviewers for their valuable feedback. This work is partially supported by the National Natural Science Foundation of China under Grant No. 61922003 and Grant No. 62161146003.

A APPENDIX: PROOF FOR ERROR ANALYSIS

Lemma A.1. $\forall i \in 1, 2, \cdots, m$,

$$\left| \prod_{j \neq i, j \in \{1,2,\cdots,m\}} \left(\cos\left(\frac{2i-1}{2m}\pi\right) - \cos\left(\frac{2j-1}{2m}\pi\right) \right) \right| = \left| \frac{m}{2^{m-1} \sin\left(\frac{2i-1}{2m}\pi\right)} \right| \tag{12}$$

PROOF. Using the sum-to-product formula, we have that (i is fixed, while $j \in 1, 2, \cdots, m$)

$$\begin{aligned} LHS &= 2^{m-1}\left|\prod_{i>j}\sin(\frac{i-j}{2m}\pi)\cdot\sin(\frac{i+j-1}{2m}\pi)\right|\cdot\left|\prod_{i<j}\sin(\frac{j-i}{2m}\pi)\cdot\sin(\frac{i+j-1}{2m}\pi)\right| \\ &= 2^{m-1}\left|\prod_{j=1}^{2i-2}\sin\frac{j}{2m}\pi\right|\cdot\left|\prod_{j=1}^{m-i}\sin\frac{j}{2m}\pi\right|\cdot\left|\prod_{j=2i}^{i+m-1}\sin\frac{j}{2m}\pi\right| \\ &= 2^{m-1}\left|\prod_{j=1}^{2i-2}\sin\frac{j}{2m}\pi\right|\cdot\left|\prod_{j=m+i}^{2m-1}\sin\frac{j}{2m}\pi\right|\cdot\left|\prod_{j=2i}^{i+m-1}\sin\frac{j}{2m}\pi\right| \\ &= 2^{m-1}\left|\frac{1}{\sin(\frac{2i-1}{2m}\pi)}\prod_{j=1}^{2m-1}\sin\frac{j}{2m}\pi\right| \end{aligned} \tag{13}$$

We know that

$$x^{m-1}+x^{m-2}+\cdots+x+1=\prod_{k=1}^{m-1}\left(x-e^{\frac{2k\pi}{m}i}\right) \tag{14}$$

Let $x = 1$:

$$m=\prod_{k=1}^{m-1}\left|1-\cos(\frac{2k\pi}{m})-i\sin(\frac{2k\pi}{m})\right|=\prod_{k=1}^{m-1}\left|2\sin(\frac{k\pi}{m})(\sin(\frac{k\pi}{m})-i\cos(\frac{k\pi}{m}))\right| \tag{15}$$

Then, we obtain:

$$\prod_{k=1}^{m-1}\sin(\frac{k\pi}{m})=\frac{m}{2^{m-1}} \tag{16}$$

$$\begin{aligned} \prod_{j=1}^{2m-1}\sin(\frac{j}{2m}\pi) &= \prod_{j=1}^{m-1}\sin(\frac{j}{2m}\pi)\sin(\frac{m-j}{2m}\pi) \\ &= \prod_{j=1}^{m-1}\sin(\frac{j}{2m}\pi)\cos(\frac{j}{2m}\pi) \\ &= \frac{1}{2^{m-1}}\prod_{j=1}^{m-1}sin(\frac{j}{m}\pi) \\ &= \frac{m}{2^{2m-2}} \end{aligned} \tag{17}$$

Finally with Equations (13) and (17), we have

$$LHS=\left|\frac{m}{2^{m-1}\sin\left(\frac{2i-1}{2m}\pi\right)}\right| \tag{18}$$

□

With the following definitions and notations:

- $x_i = x_1, \ldots, x_m$ are m Chebyshev nodes within the interval $\mathcal{I} = [a, b]$
$$x_i=\frac{1}{2}(a+b)+\frac{1}{2}(b-a)\cos\left(\frac{2i-1}{2m}\pi\right)$$
- $p(x)$ is the generated polynomial, with $p(x_i) = p_i$.
- $f(x)$ is the oracle result, and assume that $f(x)$ is a smooth function, with $f(x_i) = q_i$.
- $P_0(x)$ is the interpolation polynomial with q_i.

Theorem A.2. *If* $\forall i, |p_i - q_i| \leq \sigma$, *we have* $\forall x \in \mathcal{I}$, $\exists \xi_x \in \mathcal{I}$, *such that*

$$|f(x) - p(x)| \leq \frac{(b-a)^m}{2^{2m-1}m!} f^{(m)}(\xi_x) + \sigma\left(\frac{2\ln(2m-1)+2}{\pi} + \pi\right) \tag{19}$$

Proof. Based on the interpolation error theorem [Epperson 2013], the interpolation polynomial $P_0(x)$ satisfies

$$\exists \xi_x \in (a,b),\ f(x) - P_0(x) = \frac{1}{m!} f^{(m)}(\xi_x) \prod_{i=1}^{m} (x - x_i) \tag{20}$$

where we can substitute x with

$$x = \frac{a+b}{2} + \frac{b-a}{2}t, \quad \text{where } t \in [-1,1] \tag{21}$$

Then, Equation (20) equals to

$$\frac{((b-a)/2)^m}{m!} f^{(m)}(\xi_x) \prod_{i=1}^{m} (t - \cos(\frac{2i-1}{2m}\pi)) \tag{22}$$

Since $2^{m-1} \prod_{i=1}^{m} (t - cos(\frac{2i-1}{2m}\pi))$ is the m-order Chebyshev polynomial $\cos(m \arccos x)$, the error in Equations (20) and (22) satisfies

$$|f(x) - P_0(x)| \leq \frac{(b-a)^m}{2^{(2m-1)} m!} f^{(m)}(\xi_x) \tag{23}$$

We notice that

$$|f(x) - p(x)| \leq |f(x) - P_0(x)| + |P_0(x) - p(x)| \tag{24}$$

Thus, we simply need to show that

$$|P_0(x) - p(x)| \leq \sigma\left(\frac{2\ln(2m-1)+2}{\pi} + \pi\right) \tag{25}$$

Note that $Q(x)$ as $Q(x) = P_0(x) - p(x)$; thus, we have

$$Q(x) = \sum_{i=1}^{m} (q_i - p_i) \prod_{j \neq i} \frac{(x - x_j)}{(x_i - x_j)} \quad x \in [a,b] \tag{26}$$

We substitute x using Equation (21)

$$\begin{aligned} Q(t) &= \sum_{i=1}^{m} (q_i - p_i) \prod_{j \neq i} \frac{(t - \cos(\frac{2j-1}{2m}\pi))}{(\cos(\frac{2i-1}{2m}\pi) - \cos(\frac{2j-1}{2m}\pi))} \quad t \in [-1,1] \\ &\leq \sigma \sum_{i=1}^{m} \left| \prod_{j \neq i} \frac{(t - \cos(\frac{2j-1}{2m}\pi))}{(\cos(\frac{2i-1}{2m}\pi) - \cos(\frac{2j-1}{2m}\pi))} \right| \\ &\leq \sigma \sum_{i=1}^{m} \left| \frac{\frac{1}{2^{(m-1)}} \cos(m \arccos x)}{(t - \cos(\frac{2i-1}{2m}\pi)) \prod_{j \neq i} (\cos(\frac{2i-1}{2m}\pi) - \cos(\frac{2j-1}{2m}\pi))} \right| \end{aligned} \tag{27}$$

Using **Lemma A.1**, we can transform the *RHS* into

$$RHS = \sigma \sum_{k=1}^{m} \left| \frac{\sin(\frac{2k-1}{2m}\pi) \cos(m \arccos x)}{m(x - \cos\frac{2k-1}{2m}\pi)} \right| \quad x \in [-1,1] \tag{28}$$

Then, we only need to prove that

$$h(x) = \sum_{k=1}^{m} \left| \frac{\sin(\frac{2k-1}{2m}\pi) \cos(m \arccos x)}{m(x - \cos\frac{2k-1}{2m}\pi)} \right| \leq \frac{2\ln(2m-1)+2}{\pi} + \pi \tag{29}$$

Without loss of generality, we assume that $x = \cos\frac{t}{2m}\pi$ and $\frac{t}{2m}\pi \in [0, \pi/2]$. We consider the following two conditions separately:

- $\exists k \in \mathbb{Z}$, such that $t \in [2k-1, 2k]$
- $\exists k \in \mathbb{Z}$, such that $t \in [2k-2, 2k-2]$

In the first condition, $\exists k \in \mathbb{Z}$, such that $t \in [2k-1, 2k]$. We split $h(x)$ as $h(x) = S_1 + S_2$

$$\begin{aligned} S_1 &= \sum_{i \neq k} \left| \frac{\sin(\frac{2i-1}{2m}\pi)\cos(m \arccos x)}{m(x - \cos\frac{2i-1}{2m}\pi)} \right| \\ S_2 &= \left| \frac{\sin(\frac{2k-1}{2m}\pi)\cos(m \arccos x)}{m(x - \cos\frac{2k-1}{2m}\pi)} \right| \end{aligned} \tag{30}$$

$$\begin{aligned} S_1 &\leq \sum_{i \neq k} \left| \frac{\sin(\frac{2i-1}{2m}\pi)}{m(\cos\frac{t}{2m}\pi - \cos\frac{2i-1}{2m}\pi)} \right| \\ &\leq \frac{1}{2m} \sum_{i \neq k} \left(\left| \cot(\frac{t-2i+1}{4m}\pi) \right| + \left| \cot(\frac{t+2i-1}{4m}\pi) \right| \right) \end{aligned} \tag{31}$$

Since $\forall x \in [0, \pi/2)$, $\tan x \geq x$, we have $\cot x \leq 1/x, \forall x \in [0, \pi/2)$. Then

$$\begin{aligned} S_1 &\leq \frac{2}{\pi}(\sum_{i<k} \frac{1}{t-2i+1} + \sum_{i>k} \frac{1}{2i-1-t} + \sum_{i \leq m-k} \frac{1}{t+2i-1} + \sum_{i>m-k} \frac{1}{4m-t-2i+1}) \\ &\leq \frac{2}{\pi}(\sum_{i=1}^{k-1} \frac{1}{2i} + \sum_{i=1}^{m-k} \frac{1}{2i-1} + \sum_{i=k}^{m-1} \frac{1}{2i} + \sum_{i=m-k+1}^{m} \frac{1}{2i-1}) \\ &\leq \frac{2}{\pi} \sum_{i=1}^{2m-1} \frac{1}{i} \\ &\leq \frac{2}{\pi}(\ln(2m-1)+1) \end{aligned} \tag{32}$$

Next we prove $S_2 \leq \pi$

$$\begin{aligned} S_2 &= \frac{1}{m} \left| \frac{\sin(\frac{2k-1}{2m}\pi)\cos(\frac{t}{2}\pi)}{\cos\frac{t}{2m}\pi - \cos\frac{2k-1}{2m}\pi} \right| \\ &= \frac{1}{2m} \left| \frac{\sin(\frac{2k-1}{2m}\pi)\sin(\frac{t-2k+1}{2}\pi)}{\sin(\frac{t-2k+1}{4m}\pi)\sin(\frac{t+2k-1}{4m}\pi)} \right| \end{aligned} \tag{33}$$

Substitute $t-2k+1$ as s, $s \in [0,1]$, we have the fact that $\forall x \in [0, \pi/2)$, $2x/\pi \leq \sin x \leq x$, thus

$$\begin{aligned} S_2 &\leq \frac{\sin\frac{s}{2}\pi}{2m\sin\frac{s}{4m}\pi} \left| \frac{\sin\frac{2k-1}{2m}\pi}{\sin(\frac{2k-1}{2m}+\frac{s}{4m})\pi} \right| \\ &\leq \frac{\frac{s}{2}\pi}{2m\frac{s}{4m}\pi\frac{2}{\pi}} \left(\left| \cos\frac{s}{4m}\pi \right| + \left| \frac{\sin\frac{s}{4m}\pi}{\sin(\frac{2k-1}{2m}+\frac{s}{4m})\pi} \cos(\frac{2k-1}{2m}+\frac{s}{4m})\pi \right| \right) \end{aligned} \tag{34}$$

For $s \in [0,1]$, $\sin\frac{s\pi}{4m} \leq \sin(\frac{2k-1}{2m}+\frac{s}{4m})\pi$, so we have

$$S_2 \leq \pi \tag{35}$$

With Equations (32) and (35), we have

$$h(x) = S_1 + S_2 \leq \frac{2\ln(2m-1)+2}{\pi} + \pi \tag{36}$$

Therefore, the proof for the first condition is complete. For the second condition, $\exists k \in \mathbb{Z}$, such that $t \in [2k-2, 2k-1]$, the proof is similar. □

REFERENCES

Milton Abramowitz, Irene A Stegun, and Robert H Romer. 1988. Handbook of mathematical functions with formulas, graphs, and mathematical tables.

Ivo Babuska. 1968. Numerical stability in mathematical analysis.. In *IFIP Congress (1)*, Vol. 68. 11–23.

Tao Bao and Xiangyu Zhang. 2013. On-the-fly detection of instability problems in floating-point program execution. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 817–832.

Florian Benz, Andreas Hildebrandt, and Sebastian Hack. 2012. A dynamic program analysis to find floating-point accuracy problems. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. 453–462.

R Broucke. 1973. Algorithm: ten subroutines for the manipulation of Chebyshev series. *Commun. ACM* 16, 4 (1973), 254–256.

Richard L Burden and J Faires. 2010. Numerical analysis (9th). (2010).

Françoise Chatelin and Marie-Christine Brunet. 1990. A probabilistic round-off error propagation model. application to the eigenvalue problem. *Cox and Hammarling [CH90]* (1990), 139–160.

Wei-Fan Chiang, Ganesh Gopalakrishnan, Zvonimir Rakamaric, and Alexey Solovyev. 2014. Efficient search for inputs causing high floating-point errors. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. 43–52.

Sangeeta Chowdhary, Jay P Lim, and Santosh Nagarakatte. 2020. Debugging and detecting numerical errors in computation with posits. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 731–746.

Sangeeta Chowdhary and Santosh Nagarakatte. 2021. Parallel shadow execution to accelerate the debugging of numerical errors. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 615–626.

C.W. Clenshaw. 1962. Chebyshev series for mathematical functions. *National Physical Laboratory Mathematical Tables, Vol. 5.* (1962).

Robert M Corless, Gaston H Gonnet, David EG Hare, David J Jeffrey, and Donald E Knuth. 1996. On the LambertW function. *Advances in Computational mathematics* 5, 1 (1996), 329–359.

Nasrine Damouche and Matthieu Martel. 2017. Salsa: An Automatic Tool to Improve the Numerical Accuracy of Programs.. In *AFM@ NFM*. 63–76.

Biswa Nath Datta. 2010. *Numerical linear algebra and applications*. Vol. 116. SIAM.

Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. 2005. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media.

Saikat Dutta, Owolabi Legunsen, Zixin Huang, and Sasa Misailovic. 2018. Testing probabilistic programming systems. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 574–586.

Morris L Eaton. 1983. *The Gauss-Markov theorem in multivariate analysis*. Technical Report. University of Minnesota.

James F Epperson. 2013. *An introduction to numerical methods and analysis*. John Wiley & Sons.

Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)* 33, 2 (2007), 13–es.

Zhoulai Fu, Zhaojun Bai, and Zhendong Su. 2015. Automated backward error analysis for numerical code. In *ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 639–654.

Toshio Fukushima. 2013. Precise and fast computation of Lambert W-functions without transcendental function evaluations. *J. Comput. Appl. Math.* 244 (2013), 77–89.

Amparo Gil, Javier Segura, and Nico M Temme. 2007. *Numerical methods for special functions*. SIAM.

Hui Guo and Cindy Rubio-González. 2020. Efficient generation of error-inducing floating-point inputs via symbolic execution. In *ACM/IEEE International Conference on Software Engineering (ICSE)*. 1261–1272.

Richard Hamming. 2012. *Numerical methods for scientists and engineers*. Courier Corporation.

Nicholas J. Higham and Théo Mary. 2019. A New Approach to Probabilistic Rounding Error Analysis. *SIAM J. Sci. Comput.* 41, 5 (2019), A2815–A2835.

Francis Begnaud Hildebrand. 1987. *Introduction to numerical analysis*. Courier Corporation.

W Kahan. 1965. Further remarks on reducing truncation errors, Commun. *Assoc. Comput. Mach* 8 (1965), 40.

William Kahan. 2004. A logarithm too clever by half.

Jay P Lim, Mridul Aanjaneya, John Gustafson, and Santosh Nagarakatte. 2021. An approach to generate correctly rounded math libraries for new floating point variants. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–30.

Jay P Lim and Santosh Nagarakatte. 2021a. High performance correctly rounded math libraries for 32-bit floating point representations. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 359–374.

Jay P Lim and Santosh Nagarakatte. 2021b. RLIBM-ALL: A Novel Polynomial Approximation Method to Produce Correctly Rounded Results for Multiple Representations and Rounding Modes. *arXiv preprint arXiv:2108.06756* (2021).

Jacques-Louis Lions, Lennart Luebeck, Jean-Luc Fauquembergue, Gilles Kahn, Wolfgang Kubbat, Stefan Levedag, Leonardo Mazzini, Didier Merle, and Colin O'Halloran. 1996. Ariane 5 flight 501 failure report by the inquiry board.

John C Mason and David C Handscomb. 2002. *Chebyshev polynomials.* CRC press.

William Mendenhall, Robert J Beaver, and Barbara M Beaver. 2012. *Introduction to probability and statistics.* Cengage Learning.

Jean-Michel Muller. 2005. *On the definition of ulp (x).* Ph. D. Dissertation. INRIA, LIP.

Karl Hans Müller. 1983. Rounding error analysis of Horner's scheme. *Computing* 30, 4 (1983), 285–303.

Nicholas Nethercote and Julian Seward. 2007. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* 89–100.

ACR Newbery. 1974. Error analysis for polynomial evaluation. *Math. Comp.* 28, 127 (1974), 789–793.

Jack Oliver. 1979. Rounding error propagation in polynomial evaluation schemes. *J. Comput. Appl. Math.* 5, 2 (1979), 85–97.

Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. 2015. Automatically improving accuracy for floating point expressions. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* 1–11.

Peter Larsson. 2013. Exploring Quadruple Precision Floating Point Numbers in GCC and ICC. https://www.nsc.liu.se/~pla/blog/2013/10/17/quadruple-precision/. [Online; Accessed: 2021-01-01].

Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. 2019. CRADLE: cross-backend validation to detect and localize bugs in deep learning libraries. In *International Conference on Software Engineering (ICSE).* 1027–1038.

Walter Rudin et al. 1976. *Principles of mathematical analysis.* Vol. 3. McGraw-hill New York.

Thomas P Ryan. 2008. *Modern regression methods.* Vol. 655. John Wiley & Sons.

Alex Sanchez-Stern, Pavel Panchekha, Sorin Lerner, and Zachary Tatlock. 2018. Finding root causes of floating point error. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI).* 256–269.

Robert Skeel. 1992. Roundoff error and the Patriot missile. *SIAM News* 25, 4 (1992), 11.

Alexey Solovyev, Marek S Baranowski, Ian Briggs, Charles Jacobsen, Zvonimir Rakamarić, and Ganesh Gopalakrishnan. 2018. Rigorous estimation of floating-point round-off errors with symbolic taylor expansions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 41, 1 (2018), 1–39.

Martti Tienari. 1970. A statistical model of roundoff error for varying length floating-point arithmetic. *BIT Numerical Mathematics* 10, 3 (1970), 355–365.

Peter Valdes-Dapena and Kyung Lah. 2010. Toyota: Software to blame for Prius brake problems. http://edition.cnn.com/2010/WORLD/asiapcf/02/04/japan.prius.complaints/index.html. [Online; Accessed: 2021-01-01].

Jackson Vanover, Xuan Deng, and Cindy Rubio-González. 2020. Discovering discrepancies in numerical libraries. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA).* 488–501.

Ran Wang, Daming Zou, Xinrui He, Yingfei Xiong, Lu Zhang, and Gang Huang. 2016. Detecting and fixing precision-specific operations for measuring floating-point errors. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE).* 619–630.

Xie Wang, Huaijin Wang, Zhendong Su, Enyi Tang, Xin Chen, Weijun Shen, Zhenyu Chen, Linzhang Wang, Xianpei Zhang, and Xuandong Li. 2019. Global optimization of numerical programs via prioritized stochastic algebraic transformations. In *International Conference on Software Engineering (ICSE).* 1131–1141.

Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. 2017. Efficient global search for inputs triggering high floating-point inaccuracies. In *Asia-Pacific Software Engineering Conference (APSEC).* 11–20.

Xin Yi, Liqian Chen, Xiaoguang Mao, and Tao Ji. 2019. Efficient automated repair of high floating-point errors in numerical libraries. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29.

Daming Zou, Ran Wang, Yingfei Xiong, Lu Zhang, Zhendong Su, and Hong Mei. 2015. A genetic algorithm for detecting significant floating-point inaccuracies. In *International Conference on Software Engineering (ICSE)*, Vol. 1. 529–539.

Daming Zou, Muhan Zeng, Yingfei Xiong, Zhoulai Fu, Lu Zhang, and Zhendong Su. 2020. Detecting floating-point errors via atomic conditions. *Proceedings of the ACM on Programming Languages* 4, POPL (2020), 60:1–60:27.

Dan Zuras, Mike Cowlishaw, et al. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (Aug 2008), 1–70.