Round to Odd & Correct Rounding

Seonghyeon Hong

Pohang Institute of Science and Technology

Based on:

[TC 08] “Emulation of a FMA and Correctly Rounded Sums: Proved Algorithms Using Rounding to Odd”

[POPL 22] “One polynomial approximation to produce correctly rounded results of an elementary function”

Presentation Overview

Motivation

Correct rounding principle: The result of a computation is the same as if it were first computed with infinite precision, then rounded to the precision of the destination format.

However, real hardware has finite precision, sometimes higher than the target precision.

Motivation

Two solutions:

1. Set a target floating-point precision
2. Calculate in the higher precision, then round to the target precision

Problems:

1. Setting a target precision is costly
2. Double rounding can introduce inaccuracy

When using the same rounding mode, double rounding can lead to less precise results.

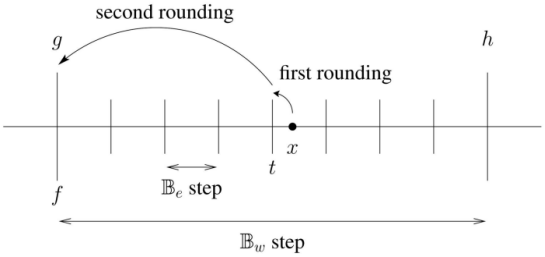

Figure: $\mathbb{B}_e$ is the extended precision, $\mathbb{B}_w$ is the working precision

Main Idea 1

Can we avoid this problem? $\rightarrow$ **Round-to-Odd**!

Rounding to odd first in the extended precision always produces correctly rounded results

Round-to-Odd

Definition (Round to Odd [TC08])

Denote by $\triangle$ rounding towards ∞ and by $\triangledown$ rounding towards $-\infty$

$$\square_{\text{odd}}(x) = \begin{cases} x & \text{if } x \in \mathbb{F} \\ \triangle(x) & \text{if its mantissa is odd} \\ \triangledown(x) & \text{otherwise} \end{cases} \tag{1}$$

Intuitively, round-to-odd keeps a value the same if it is representable, else it rounds to the neighboring FP number whos mantissa is odd

Theorem

Assuming $p \geq 2$, $k \geq 2$, and $E_e \geq 2 + E_w$ where $-E_e$ and $-E_w$ represents the minimum exponent of the extended and working precisions respectively,

$$\forall x \in \mathbb{R}, \quad \circ^p(\square_{odd}^{p+k}(x)) = \circ^p(x)$$

where $\circ(x)$ means round-to-nearest-even

In other words, double rounding where the first rounding mode is round-to-odd always produces the correct result.
The theorem can be extended to other rounding modes.

Round-to-Odd

Applications?

- FMA
- Iterative algorithm for sum of FP numbers
- **Polynomial approximation** [POPL22]

Motivation 2

Polynomial Approximation

Many math libraries don’t produce correctly rounded results for all possible inputs and multiple rounding modes

$\rightarrow$ Can we create a single polynomial approximation that produces correct results for all inputs & multiple rounding modes?

Yes!

Main Idea: Generate a polynomial approximation using **round-to-odd** mode with $n+2$ bits (i.e. $\mathbb{T}_{n+2}$)

Notation: We denote by $\mathbb{T}_n$ the FP representation with n bits, and $\mathbb{F}_{n,|E|}$ the set of values with total n bits and $|E|$ bits for the exponent

Let $v_{\mathbb{R}}$ be a real value. Denote

$$v_{sm} = \max\{v \in \mathbb{F}_{n,|E|} | v \leq v_{\mathbb{R}}\} \tag{2}$$

$$v_{lg} = \min\{v \in \mathbb{F}_{n,|E|} | v \geq v_{\mathbb{R}}\} \tag{3}$$

The rounding mode, represented by *rm*, specifies whether $v_{\mathbb{R}}$ is rounded to v_{sm} or v_{lg}. We denote the rounding function in the representation $\mathbb{T}_n$ as $RN_{\mathbb{T}_n,rm}(v_{\mathbb{R}})$

IEEE-574 specifies 5 rounding modes

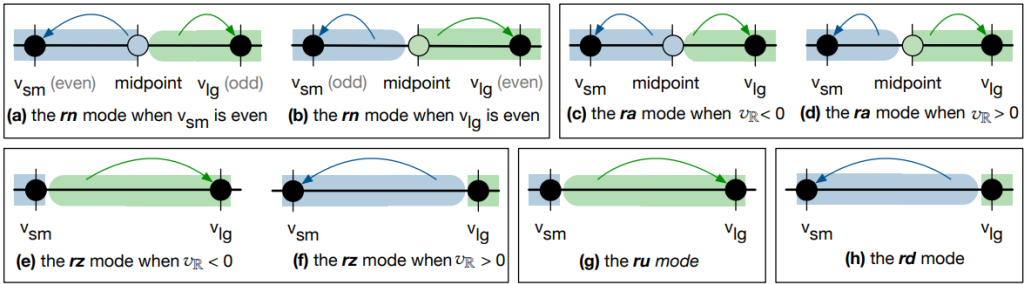

Figure: 5 different rounding modes

We identify four information ($s, v^-, rb, sticky$) from the real value $v_{\mathbb{R}}$ to correctly round according to rounding modes. We call these **rounding components**.

s represents the sign of $v_{\mathbb{R}}$, v^- represents the smaller of v_{sm} and v_{lg} in magnitude. rb and $sticky$ tell whether $v_{\mathbb{R}}$ is in the middle, or closer to v_{sm} or v_{lg}.

Represent $v_{\mathbb{R}}$ with $|E|$ exponent bits and an infinite number of mantissa bits:

$$B_{v_{\mathbb{R}}} = b_1 b_2 b_3 \cdots b_n b_{n+1} b_{n+2} \cdots$$

b_1 represents the sign bit, $b_2 \cdots b_{|E|+1}$ represents the exponent bits

We identify two positive values v^- and v^+ adjacent to $|v_{\mathbb{R}}|$, where $v^- \leq |v_{\mathbb{R}}| < v^+$

For v^-, truncate $B_{v_{\mathbb{R}}}$ into n bits:

$$B_{v^-} = 0 b_2 b_3 \cdots b_{n-1} b_n$$

v^+ is the succeeding value of $v^- \rightarrow v^+ = v^- + 1$
The following property is satisfied:

$$\begin{cases} -v^+ < v_{\mathbb{R}} \leq -v^- & \text{if } v_{\mathbb{R}} < 0 \quad (s = -1) \\ v^- \leq v_{\mathbb{R}} < v^+ & \text{if } v_{\mathbb{R}} \geq 0 \quad (s = 1) \end{cases} \tag{4}$$

Rounding bit: The $(n+1)^{th}$-bit of $B_{v_{\mathbb{R}}}$

$$\begin{cases} v^- \leq |v_{\mathbb{R}}| < \frac{v^- + v^+}{2} & rb = 0 \\ \frac{v^- + v^+}{2} \leq |v_{\mathbb{R}}| < v^+ & rb = 1 \end{cases} \tag{5}$$

rb alone doesn't tell us whether $|v_{\mathbb{R}}| = \frac{v^- + v^+}{2}$. When does this happen?
$\rightarrow |v_{\mathbb{R}}| = \frac{v^- + v^+}{2}$ when $b_{n+1} = 1$ and all bits starting from the $(n+2)^{th}$-bit is 0
$\rightarrow$ Determine if all bits starting from b_{n+2} is 0

Sticky bit: The bitwise OR of all bits starting from the $(n+2)^{th}$-bit

$$sticky = b_{n+2} | b_{n+3} | b_{n+4} | \cdots$$

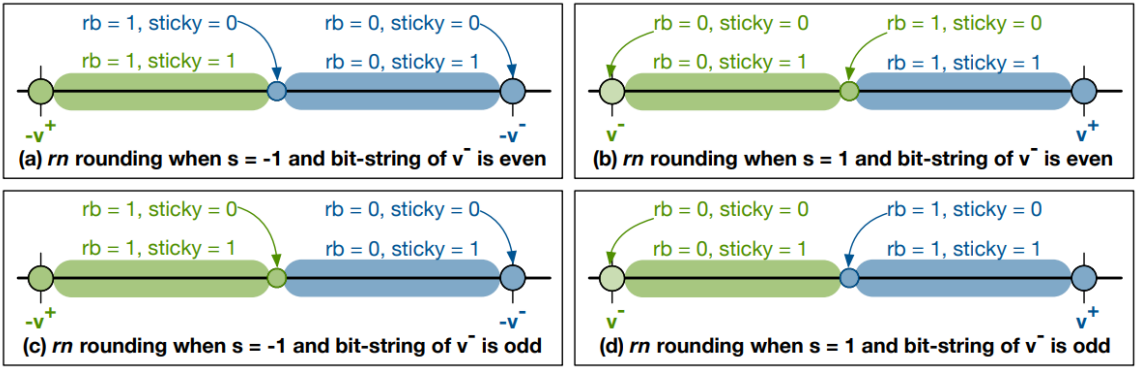

Figure: Rounding Components

The RLibm Approach

The RLibm approach for computing a function in a representation $\mathbb{T}$ consists of 4 steps:

1. Use an oracle to compute correctly roudned result y_{ro} of a function $f(x)$ for each input $x \in \mathbb{T}$
2. Identify an interval $[l, h]$ around y_{ro} s.t. any value in $[l, h]$ rounds to y_{ro}
   - Rounding intervals are in $\mathbb{H}$
3. Range reduction on x to x', the generated polynomial will approximate the result for x', then output compensation will produce the final result for x
   - Computation done in higher precision $\mathbb{H}$
   - Reverse output compensation to find interval $[l', h']$ for $P(x')$
4. Make a polynomial using an LP solver with constraints like $l' \leq P(x') \leq h'$

The RLibm Approach

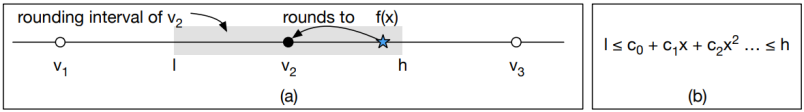

Figure: Rounding Interval

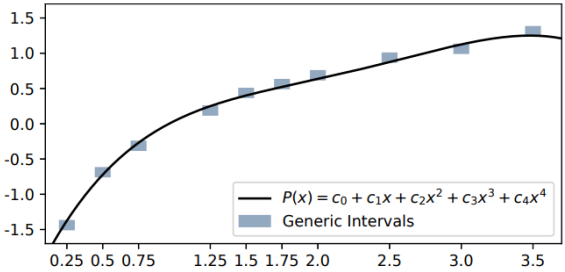

Figure: Polynomial Generation

Main Idea

Our goal: For a representation $\mathbb{T}_k$ where $|E| + 1 < k \leq n$ and a rounding mode $rm \in \{rn, ra, rz, ru, rd\}$, generate a polynomial approximation $A_{\mathbb{H}}(x)$ such that:

$$RN_{\mathbb{T}_k,rm}(A_{\mathbb{H}}(x)) = RN_{\mathbb{T}_k,rm}(f(x))$$

Main Idea: Create a polynomial approximation that produces correctly rounded results for $\mathbb{T}_{n+2}$ in *round-to-odd* mode using a similar method as RLibm.
We call the rounding intervals in this case as **odd intervals**.

Define round-to-odd using rounding components $(s, v^-, rb, sticky)$

Definition (Round-to-Odd [POPL22])

$$v_{ro} = RN_{\mathbb{T},ro}(v_{\mathbb{R}}) = \begin{cases} s \times v^- & \text{if } IsOdd(v^-) \vee (rb = 0 \wedge sticky = 0) \\ s \times v^+ & otherwise \end{cases} \quad (6)$$

where v^+ is the adjacent value to v^- in $\mathbb{T}$

This is the same definition as the previous paper, just stated differently

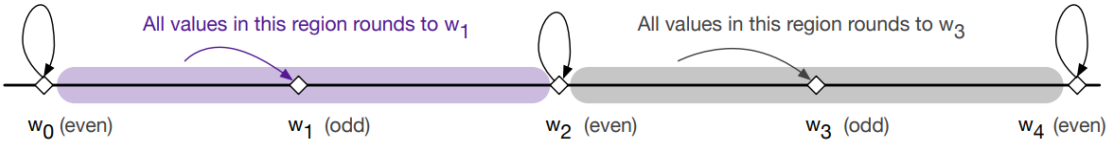

Figure: Round to Odd

We use the fact that double rounding with round-to-odd produces correct results. More specifically, the following theorem is true:

Theorem

Let $v_{\mathbb{R}} = f(x)$ be a real values result of an elementary function and $v_{ro} = RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}})$. Let v be a value in the odd interval of v_{ro}. Consider a rounding mode $rm \in \{rn, ra, rz, ru, rd\}$. Then,

$$RN_{\mathbb{T}_k,rm}(v) = RN_{\mathbb{T}_k,rm}(v_{\mathbb{R}})$$

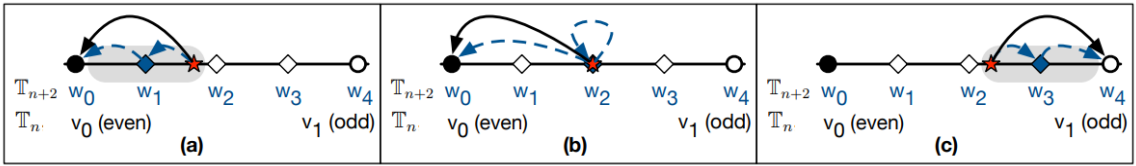

Figure: Intuition of Theorem

1 **Function** GenerateGenericPolynomial($f, \mathbb{T}_{n+2}, \mathbb{H}, X, d, RR_{\mathbb{H}}, OC_{\mathbb{H}}$):
2 $O \leftarrow$ CalcResultsInRO($f, \mathbb{T}_{n+2}, X$)
3 $(L, S) \leftarrow$ CalcOddIntervals($O, \mathbb{T}_{n+2}, \mathbb{H}$)
4 **if** $L = \emptyset$ **then return** *(false, ∅, DNE)*
5 $(status, P) \leftarrow$ RLibmPolyGen($L, \mathbb{H}, d, RR_{\mathbb{H}}, OC_{\mathbb{H}}$)
6 **return** $(status, S, P)$

Figure: Main Algorithm

```
1  Function CalcResultsInRO(f, T_{n+2}, X):
2      O ← ∅
3      foreach x ∈ X do
4          y = f(x)
5          (s, v⁻, rb, sticky) ← RComp(y, T_{n+2})
6          if IsOdd(v⁻) ∨ (rb = 0 ∧ sticky = 0)
             then
7              y_ro ← s × v⁻
8          end
9          else
10             v⁺ ← GetSuccVal(v⁻, T_{n+2})
11             y_ro ← s × v⁺
12         end
13         O ← O ∪ (x, y_ro)
14     end
15     return O
```

```
1  Function CalcOddIntervals(O, T_{n+2}, ℍ):
2      foreach (x, y_ro) ∈ O do
3          L ← ∅
4          S ← ∅
5          if IsEven(y_ro) then
6              S ← S ∪ (x, y_ro)
7          end
8          else
9              y⁻ ← GetPrecVal(y_ro, T_{n+2})
10             l ← GetSuccVal(y⁻, ℍ)
11             y⁺ ← GetSuccVal(y_ro, T_{n+2})
12             h ← GetPrecVal(y⁺, ℍ)
13             L ← L ∪ (x, [l, h])
14         end
15         return (L, S)
16     end
```

When computing odd intervals, some can be single points, which we call **singletons**. In round-to-odd, singletons only arise when the rounded value is even.

Singletons limit the amount of freedom when generating polynomials, so we handle them separately.

All inputs are rational values. Singletons arise when the output of $f(x)$ is representable in $\mathbb{T}$, hence rational. We use this to identify singletons beforehand using properties of the function.

Ex) By the Lindemann-Weierstrass theorem, e^x is transcendental if x is a non-zero rational. Therefore a singleton can only arise when $x = 0$.

Proof of Theorem

Lemma (1)

The round-to-odd result v_{ro} in $\mathbb{T}_{n+2}$ preserves the sign of $v_{\mathbb{R}}$

Proof.

The only value that rounds to 0 is 0, so all positive values round to positive values and vis versa. □

From this point onwards, we assume values to be positive.

Proof of Theorem

Lemma (2)

Let $v_{ro} = RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}})$. The first $(n+1)$-bits of v_{ro} and $v_{\mathbb{R}}$ are identical.

Proof.

v_{ro} is created using rounding components $(s_{v_{ro}}, v^{-}_{v_{ro}}, rb_{v_{ro}}, sticky_{v_{ro}})$. $v^{-}_{v_{ro}}$ is a truncated value or $v_{\mathbb{R}}$, so all the $(n+2)$-bits of $v^{-}_{v_{ro}}$ and $v_{\mathbb{R}}$ are identical.

After rounding v_{ro} is either $v^{-}_{v_{ro}}$ or the successor of $v^{-}_{v_{ro}}$. □

Proof of Theorem

Lemma (2)

Let $v_{ro} = RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}})$. The first $(n+1)$-bits of v_{ro} and $v_{\mathbb{R}}$ are identical.

Proof.

Case $v_{v_{ro}}^{-}$ is odd $\rightarrow v_{ro} = v_{v_{ro}}^{-}$:
All $(n+2)$-bits of v_{ro} and $v_{\mathbb{R}}$ are identical

Case $v_{v_{ro}}^{-}$ is even $\rightarrow$ Last bit of $v_{v_{ro}}^{-}$ is 0:
(1) $rb_{v_{ro}} = 0$ and $sticky_{v_{ro}} = 0$, then $v_{ro} = v_{v_{ro}}^{-}$, so the lemma holds
(2) $rb_{v_{ro}} \neq 0$ and $sticky_{v_{ro}} \neq 0$ then v_{ro} is equal to the successor of $v_{v_{ro}}^{-}$. The only bit that changes between $v_{v_{ro}}^{-}$ and the successor is the $(n+2)^{th}$-bit. So the first $(n+1)$-bits are the same. □

Proof of Theorem

Lemma (3)

The $(n+2)^{th}$-bit of v_{ro} is equal to the bitwise OR of all the bits of $v_{\mathbb{R}}$ starting from the $(n+2)^{th}$-bit

Proof.

Intuitively the lemma states that the last bit of v_{ro} is 0 if and only if all bits starting from the $(n+2)^{th}$-bit of $v_{\mathbb{R}}$ is 0.

Assume the rounding components are $(s_{v_{ro}}, v^{-}_{v_{ro}}, rb_{v_{ro}}, sticky_{v_{ro}})$. Again v_{ro} is either $v^{-}_{v_{ro}}$ or it's successor. □

Proof of Theorem

Lemma (3)

The $(n+2)^{th}$-bit of v_{ro} is equal to the bitwise OR of all the bits of $v_{\mathbb{R}}$ starting from the $(n+2)^{th}$-bit

Proof.

Case $v^-_{v_{ro}}$ is odd $\rightarrow$ $v_{ro} = v^-_{v_{ro}}$:
Then the $(n+2)^{th}$-bit of $v^-_{v_{ro}}$ is 1. Since $v^-_{v_{ro}}$ is a truncated $v_{\mathbb{R}}$ the $(n+2)^{th}$-bit of $v_{\mathbb{R}}$ is 1.

Case $v^-_{v_{ro}}$ is even $\rightarrow$ The $(n+2)^{th}$-bit of $v^-_{v_{ro}}$ and $v_{\mathbb{R}}$ is 0:
(1) $rb_{v_{ro}} = 0$ and $sticky_{v_{ro}} = 0$, then $v_{ro} = v^-_{v_{ro}}$.
By definition, $rb_{v_{ro}}$ is the value of the $(n+3)^{th}$-bit and $sticky_{v_{ro}}$ is the bitwise OR of all bits starting from the $(n+4)^{th}$-bit. So the bitwise OR of all bits starting from the $(n+2)^{th}$-bit is 0 □

Proof of Theorem

Lemma (3)

The $(n+2)^{th}$-bit of v_{ro} is equal to the bitwise OR of all the bits of $v_{\mathbb{R}}$ starting from the $(n+2)^{th}$-bit

Proof.

Case $v^-_{v_{ro}}$ is even $\rightarrow$ The $(n+2)^{th}$-bit of $v^-_{v_{ro}}$ and $v_{\mathbb{R}}$ is 0:
(2) $rb_{v_{ro}} \neq 0$ and $sticky_{v_{ro}} \neq 0$ then v_{ro} is equal to the successor of $v^-_{v_{ro}}$.
So the $(n+2)^{th}$-bit of v_{ro} is 1. By similar reasons as above, some bit after the $(n+2)^{th}$-bit in $v_{\mathbb{R}}$ is 1. So the lemma holds. □

Proof of Theorem

Lemma (4)

Let $(s_1, v_1^-, rb_1, sticky_1)$ *and* $(s_2, v_2^-, rb_2, sticky_2)$ *be the rounding components for two real values* v_1 *and* v_2 *in rounding them to a FP representation* $\mathbb{T}_n$*. If* $s_1 = s_2, v_1^- = v_2^-, rb_1 = rb_2$ *and* $sticky_1 = sticky_2$*, then* $RN_{\mathbb{T},rm}(v_1) = RN_{\mathbb{T},rm}(v_2)$ *for any rounding mode rm.*

Proof.

Follows directly from the definition of rounding components. □

Proof of Theorem

We prove the following theorem:

Theorem (2)

Given a real number $v_{\mathbb{R}}$, representations $\mathbb{T}_k$ and $\mathbb{T}_{n+2}$ with same number of exponent bits that satisfy $|E| + 1 < k \leq n$, and a rounding mode $rm \in \{rn, ra, rz, ru, rd\}$, then $RN_{\mathbb{T}_k,rm}(v_{\mathbb{R}}) = RN_{\mathbb{T}_k,rm}(RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}}))$.

Proof of Theorem

Proof.

Let $v_{ro} = RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}})$. By lemma 4, we just have to check if the rounding components for $v_{\mathbb{R}}$ and v_{ro} are the same.

Let $(s_1, v_1^-, rb_1, sticky_1)$ and $(s_2, v_2^-, rb_2, sticky_2)$ be the rounding components for $v_{\mathbb{R}}$ and v_{ro} respectively in $\mathbb{T}_k$. Let the representation of $v_{\mathbb{R}}$ in extended infinite precision representation $B_{v_{\mathbb{R}}}$ be

$$B_{v_{\mathbb{R}}} = b_1 b_2 \cdots b_{k-1} b_k b_{k+1} \cdots b_n b_{n+1} b_{n+2} b_{n+3} \cdots$$

By definitions of rounding components the following is true:

$$B_{v_1^-} = b_1 b_2 \cdots b_{k-1} b_k, \quad rb_1 = b_{k+1}, \quad sticky_1 = b_{k+2} | b_{k+3} | \cdots$$

□

Proof of Theorem

Proof.

By lemma 2 and lemma 3:

$$B_{v_{ro}} = b_1 b_2 \cdots b_{k-1} b_k \cdots b_n b_{n+1} t, \quad t = b_{n+2} | b_{n+3} | \cdots$$

Since $k \leq n$ there are at least one bit between b_k and t. The rounding components are:

$$B_{v_2^-} = b_1 b_2 \cdots b_{k-1} b_k, \quad rb_2 = b_{k+1}, \quad sticky_2 = b_{k+2} | b_{k+3} | \cdots | b_{n+1} | t$$

Comparing this with $v_{\mathbb{R}}$ we can see that all are the same, hence the rounded result is the same via lemma 4.

□

Proof of Theorem

Theorem 2 $\rightarrow$ Theorem 1.

Let v be a value in the odd interval of v_{ro}
$\rightarrow RN_{\mathbb{T}_{n+2},ro}(v) = v_{ro}$

$$RN_{\mathbb{T}_k,rm}(v) = RN_{\mathbb{T}_k,rm}(RN_{\mathbb{T}_{n+2},ro}(v)) = RN_{\mathbb{T}_k,rm}(v_{ro})$$

So,

$$\begin{aligned} RN_{\mathbb{T}_k,rm}(v_{\mathbb{R}}) &= RN_{\mathbb{T}_k,rm}(RN_{\mathbb{T}_{n+2},ro}(v_{\mathbb{R}})) && (7)\\ &= RN_{\mathbb{T}_k,rm}(v_{ro}) && (8)\\ &= RN_{\mathbb{T}_k,rm}(v) && (9) \end{aligned}$$

□

Experimental Evaluation

$f(x)$	Gen. Time (Min.)	# of Polynomials	Degree	# of Terms	FP34 *ro*
$\mathbf{ln(x)}$	325	2^{10}	3	3	✓
$\mathbf{log_2(x)}$	420	2^8	3	3	✓
$\mathbf{log_{10}(x)}$	546	2^8	3	3	✓
$\mathbf{e^x}$	241	2^7 2^7	4 4	5 5	✓
$\mathbf{2^x}$	151	2^7 2^7	3 3	4 4	✓

$f(x)$	Gen. Time (Min.)	# of Polynomials	Degree	# of Terms	FP34 *ro*
$\mathbf{10^x}$	402	2^8 2^8	3 3	4 4	✓
$\mathbf{sinh(x)}$	143	2^6	5	3	✓
$\mathbf{cosh(x)}$	135	2^5	4	3	✓
$\mathbf{sinpi(x)}$	308	2^2	5	3	✓
$\mathbf{cospi(x)}$	316	2^2	4	3	✓

Figure: Details about the generated polynomials

Experimental Evaluation

	Using RLIBM-ALL					Using glibc double libm					Using Intel double libm				
$f(x)$	*rn*	*rd*	*ru*	*rz*	*ra*	*rn*	*rd*	*ru*	*rz*	*ra*	*rn*	*rd*	*ru*	*rz*	*ra*
$\ln(x)$	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
$\log_2(x)$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
$\log_{10}(x)$	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
e^x	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✗	✓
2^x	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
10^x	✓	✓	✓	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓
$\sinh(x)$	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
$\cosh(x)$	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✗	✓
$\text{sinpi}(x)$	✓	✓	✓	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✓	✓	✓	✗
$\text{cospi}(x)$	✓	✓	✓	✓	✓	N/A	N/A	N/A	N/A	N/A	✓	✗	✓	✗	✗

	Using CRLIBM					Using RLIBM-32				
$f(x)$	*rn*	*rd*	*ru*	*rz*	*ra*	*rn*	*rd*	*ru*	*rz*	*ra*
$\ln(x)$	✗	✓	✓	✓	N/A	✓	✗	✗	✗	✓
$\log_2(x)$	✓	✓	✓	✓	N/A	✓	✗	✗	✗	✓
$\log_{10}(x)$	✗	✓	✓	✓	N/A	✓	✗	✗	✗	✓
e^x	✓	✓	✓	✓	N/A	✓	✗	✗	✗	✓
2^x	N/A	N/A	N/A	N/A	N/A	✓	✗	✗	✗	✗
10^x	N/A	N/A	N/A	N/A	N/A	✓	✗	✗	✗	✓
$\sinh(x)$	✗	✓	✓	✓	N/A	✓	✗	✗	✗	✓
$\cosh(x)$	✓	✓	✓	✓	N/A	✓	✗	✗	✗	✓
$\text{sinpi}(x)$	✓	✓	✓	✓	N/A	✓	✗	✗	✗	✓
$\text{cospi}(x)$	✓	✓	✓	✓	N/A	✓	✗	✗	✗	✓

Figure: Ability to generate correct results for a 32-bit float

Conclusion

This paper proposes a novel method to generate a single polynomial approximation that produces correctly rounded results for multiple representations and rounding modes.

Provides the first correctly rounded implementations of elementary functions for multiple representations.

The End