Exploring Taylor series approximation of certain functions

By Henrik Vestermark (hve@hvks.com)

Abstract

This paper analyzes optimization techniques for evaluating Taylor series approximations of transcendental functions, mainly trigonometric functions, as well as exponential, logarithm, and hyperbolic functions. We examine four fundamental techniques: period mapping, argument reduction, efficient series evaluation, and argument scale-back. Through analysis, we demonstrate that combining these techniques can significantly improve computational efficiency and numerical accuracy. Our results show that aggressive argument reduction to approximately 0.001 provides an optimal balance between accuracy and performance, achieving 14-15 digits of precision while requiring fewer Taylor series terms. We provide visualizations of error distribution across different evaluation strategies and analyze the specific sources of accuracy loss in each computation step. The findings are implemented in a web-based tool that allows interactive exploration of these concepts. Lastly, we present a novelty within argument reduction and expansion for the hyperbolic functions arcsinh(x) and arccosh(x).

Introduction

Taylor series approximations are fundamental tools in numerical computing, particularly for evaluating transcendental functions. However, direct implementation of these series often leads to significant computational inefficiencies and accuracy losses, especially for larger input values. These challenges arise from various sources, including catastrophic cancellation in alternating series, loss of significance in floating-point operations, and the increasing number of terms required for convergence as input magnitudes grow.

This paper systematically optimizes Taylor series evaluations through techniques that address these challenges. We focus primarily on trigonometric functions, using sin(x) as our primary example, though the principles apply more broadly to other transcendental functions.

The optimization techniques we discuss build upon each other in a logical progression. We begin with period mapping, which exploits the periodic nature of trigonometric functions to reduce input arguments to a manageable range. This is followed by argument reduction using function-specific identities, which reduces the inputs' magnitude. We then examine efficient series evaluation techniques that automatically determine the optimal number of terms needed. Finally, we address the careful process of scaling back reduced arguments to obtain final results.

Our analysis shows that these techniques are effective, as demonstrated through error distribution graphs and performance metrics. We pay particular attention to the trade-offs between computational cost and accuracy, identifying optimal thresholds for argument reduction and analyzing where accuracy losses occur in the evaluation process.

The findings from this research have practical applications in numerical computing, particularly in environments where both efficiency and accuracy are crucial. I have

implemented these techniques in a web-based tool that allows practitioners to explore and visualize these concepts interactively. See https://www.hvks.com/Numerical/webTaylor.html

Contents

Taylor series Overview

We assume the reader knows the Taylor series for the trigonometric, exponential, logarithmic, and hyperbolic functions. We will address the following Taylor series.

1. Trigonometric functions
   Sin(x), Cos(x), Tan(x), Arcsin(x), Accos(x), Arctan(x)
2. Hyperbolic functions:
   Sinh(x), Cosh(x), Tanh(x), Arcsinh(x), Arccosh(x), Arctanh(x)
3. Exponential and logarithm function:
   e^x , Ln(x)

Let us break down each Taylor series with a more detailed explanation:

When looking at the Taylor series expansions for sine and cosine, it helps to visualize how each new term refines the result and why smaller input values lead to quicker convergence. For sin(x), the expansion has the familiar form.

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots for\ any\ real\ value\ x$$

Only odd powers of x appear, and each term alternates in sign. This series converges quickly for |x|<1is because raising a number smaller than one to higher and higher powers makes those powers rapidly shrink, while factorial denominators (3!, 5!, 7!, …) grow very fast. As a result, each successive term becomes so tiny that adding more of the terms hardly changes the sum beyond a certain point, so the approximation stabilizes in fewer steps. However, if |x| is large, the partial sums can briefly grow before the factorial in the denominator catches up, leading to more terms needed and potentially more numerical round-off errors. To avoid summing a large number of terms for an enormous |x|, one often applies an argument-reduction step (see later)—such as taking x modulo 2π—so the actual series is evaluated on a smaller (primary period), more manageable interval where the function's behavior is similar. Still, the series converges faster and more reliably.

A similar pattern holds for cos(x), which takes the form.

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} \ldots for\ any\ real\ value\ x$$

The cosine series starts with 1, reflecting cos(0) = 1. Only even powers appear due to cosine's even symmetry. The alternating signs create bounded oscillation. Like sine, factorial denominators ensure convergence for all x. Each term represents a correction to the approximation's curvature.

The series differs from sine mainly because it starts with one and includes only even powers of x. Like the sine series, there is an alternating sign pattern, and factorial denominators ensure that each new correction shrinks dramatically if |x|<1. Consequently, the cosine series also converges quickly for small inputs but can encounter similar difficulties with large inputs. Alternating sign expansions are prone to what is known as catastrophic cancellation,

where subtracting two nearly equal numbers can lose significant digits of precision. Keeping the argument within a small range limits how large partial sums can become in the first place, thereby reducing these subtraction-driven rounding errors. That is why libraries often prefer to minimize the argument so that |x| stays under a threshold, usually less than 1, and then apply the appropriate series expansion. This strategy preserves both the speed of convergence and the numerical accuracy.

When dealing with the Taylor series expansion for tan(x), there is an important distinction compared to sin(x) and cos(x). The simplest expression of the tangent series often involves Bernoulli numbers, as in

$$\tan(x) = x + \frac{x^3}{3} + \frac{2x^5}{15} + \frac{17x^5}{315} + \frac{62x^9}{2835} + \cdots \frac{2^{2n}(2^{2n}-1)B_{2n}x^{2n-1}}{(2n)!} + \cdots$$

Where the coefficient of each higher-order term involves the Bernoulli numbers B_{2n}, because you can't know in advance how many Bernoulli numbers are required (and since generating them for very high orders can be nontrivial), the direct Taylor series for tan(x) becomes more cumbersome to implement than sin(x) or cos(x), whose expansions rely "only" on factorial growth rather than additional constants like Bernoulli numbers.

In practical numerical code, rather than computing these coefficients on the fly, many libraries prefer to calculate tan(x) in ways that avoid those series coefficients. One common trick is to take advantage of the identity:

$$tan(x) = \frac{\sin(x)}{\sqrt{1-si\ ^2(x)}}$$

This sidesteps the need to repeatedly compute Bernoulli numbers because sin(x) can be expanded in simpler, factorial-based series and combined to get tan(x). Suppose you already have suitable methods for sin(x) and cos(x). In that case, you can piggyback off them for the tangent without re-implementing a standalone tan(x) polynomial that references potentially large sets of Bernoulli numbers.

As with other trigonometric expansions, ensuring |x| is not too large makes the series converge faster. For tan(x), you might still apply argument reduction to keep x within a smaller interval, such as $[-\pi/2, \pi/2]$. The Taylor series for sine and cosine (and thus tangent through the ratio) converges efficiently. If |x| is large, reducing it with modulo π is common—though one must watch out for neighborhood singularities at $\pm\pi/2+k\pi$, where the tangent has vertical asymptotes. Once x is in a smaller range and away from those asymptotes, the direct Bernoulli-based series or the ratio of sin(x) to cos(x) will converge smoothly and avoid the larger partial sums that invite numerical instability.

Ultimately, the Bernoulli numbers make the raw Taylor expansion for tan(x) look more intimidating. However, with careful argument reduction or delegating to other trigonometric series, you can still achieve fast and accurate approximations for a wide range of x.

The hyperbolic sine function can be expanded in a Taylor series as:

$$\sinh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} \ldots$$

This means it includes only odd powers of x but does not alternate in sign. In other words, each new term—$x, x^3/3!, x^5/5!, \ldots$ x,—is always added, never subtracted. That is because the hyperbolic sine function, unlike the ordinary sine, involves exponentials that grow positively for large x, resulting in a strictly increasing series of positive contributions.

When |x| is small (less than 1, for instance), the series converges quickly because of each successive term $x^{2n+1}/(2n+1)!$ Gets drastically smaller thanks to the factorial growth in the denominator. For large |x|, the terms themselves can get big initially, but eventually, the factorials in the denominator still dominate, making the infinite sum convergent for any real x. Nonetheless, summing many large terms in finite precision arithmetic can risk numerical inaccuracy (though not as severely as if the signs were alternating and canceling each other).

The hyperbolic cosine function expands into a series with only even powers of x:

$$\cosh(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} \ldots for\ any\ real\ value\ x$$

Here, all terms are positive, reflecting cosh(0) = 1, and that cosh(x) grows exponentially as x becomes large. As with sinh(x), if |x| is small, each new power x^{2n} is relatively tiny, and dividing by (2n)! It only makes the terms shrink faster. This leads to rapid convergence for modest |x|. If x is large, the same principle applies: the series still converges for all real x because factorial growth eventually overtakes any power growth. In practical implementations, libraries often rely on exponent-based definitions of sinh and cosh (e.g., $\cosh(x) = \frac{1}{2}(e^x + e^{-x})$ or use argument reductions, but the pure Taylor expansion is always valid in theory.

The hyperbolic tangent function, tanh(x), can be written in a direct power series, but this form typically involves Bernoulli numbers:

$$\tanh(x) = x - \frac{x^3}{3} + \frac{2x^5}{15} - \frac{17x^7}{315} + \cdots \frac{(-1)^{n-1} \cdot 2^{2n}(2^{2n} - 1)B_{2n}x^{2n-1}}{(2n)!} + \cdots$$

Because Bernoulli numbers B_{2n} are required for each term's coefficient, implementing this direct approach can become cumbersome—you need to generate or look up increasingly large sets of Bernoulli numbers. Instead, many numerical algorithms exploit more straightforward expansions for sinh(x) and cosh(x) and then compute

$$tanh(x) = \frac{sinh(x)}{cosh(x)}$$

By dividing the two, you avoid the Bernoulli coefficient altogether. From a convergence standpoint, sinh(x) and cosh(x) each converge nicely thanks to factorial denominators, and dividing one by the other is straightforward as long as cosh(x) is not extremely large or near zero (which it never is, except for potential floating-point overflow issues at huge |x|).

For large |x|, tanh(x) approaches 1 (or -1) quite rapidly; in practice, summing many terms of the series to confirm that fact is rarely necessary. For small |x|, the expansions of sinh and cosh again converge quickly, making the ratio-based approach an efficient and accurate method to compute tanh(x) for most typical numerical contexts.

The inverse sine function arcsin(x) can be expanded in a series around x=0, producing a sum involving only the odd x powers. A typical form is:

$$Arcsin(x) = x + \frac{x^3}{2\cdot 3} + \frac{3x^5}{2\cdot 4\cdot 5} + \frac{3\cdot 5x^7}{2\cdot 4\cdot 6\cdot 7} + \frac{3\cdot 5\cdot 7x^9}{2\cdot 4\cdot 6\cdot 8\cdot 9} \ldots$$

In principle, this series converges for |x|≤1, but if x is near ±1, you may need many terms because the partial sums change slowly. The coefficients look complicated, but a known recurrence moves from one term to the next by multiplying by a ratio of polynomials in x.

If we denote the nth Taylor term, r, we can go from one Taylor term to the next using the following recurrence:

$$r_1 = x$$

$$r_n = r_{n-1}\frac{(2n-3)^2 \cdot x^2}{(2n-1)(2n-2)} \; for\; n = 2,3,\ldots,m$$

When |x| is less than 1 by a comfortable margin, this polynomial approach converges quickly and directly yields arcsin(x). For |x| close to 1, a transformation or additional identity might help, but in straightforward scenarios, the Taylor expansion often runs faster than a generic method like Newton's iteration.

Since arccos(x)+arcsin(x)=π/2, many implementations rely on the more straightforward arcsin expansion. They compute

$$Arccos(x) = \frac{\pi}{2} - Arcsin(x)$$

That way, once you have a function for arcsin, you get arccos "for free." There isn't a separate, significant optimization for the arccos series itself, so argument reductions and expansions typically focus on arcsin. When |x| is large (bigger than 1), you're outside the domain for real arcsin or arccos, so real-valued expansions don't apply.

A more familiar inverse function is arctan(x), which has the classic alternating power series:

$$Arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \cdots, where\ |x| \le 1$$

valid for |x|≤1. To handle |x|>1 or to speed things up when |x| is moderately large, many libraries exploit the identity

$$Arctan(x) = 2 \cdot \arctan\left(\frac{x}{1+\sqrt{1+x^2}}\right)$$

Often repeating it until the transformed input is small enough for the Taylor series to converge fast. This approach reduces argument by shrinking x into a more comfortable zone (like |x|<1), so you need fewer terms. The sign-alternation in the arctan expansion (plus or minus) helps keep partial sums within a bounded range. Still, it can also mean a certain susceptibility to floating-point cancellation if terms get close in size. Even so, for typical |x|<1 uses, it converges quickly.

For the inverse hyperbolic sine (arcsinh), you can use two slightly different expansions depending on whether |x|< 1 or |x| is large. When |x|<1, the series:

$$\text{Arcsinh}(x) = x - \frac{1}{2}\frac{x^3}{3} + \frac{1\cdot3}{2\cdot4}\frac{x^5}{5} - \frac{1\cdot3\cdot5}{2\cdot4\cdot6}\frac{x^7}{7} + \frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8}\frac{x^9}{9} - \cdots$$

If we denote the nth Taylor term, r, we can go from one Taylor term to the next using the following recurrence:

$$r_0 = x$$

$$r_{n+1} = -r_n \frac{(2n+1)^2 \cdot x^2}{(2n+2)(2n+3)} \; for\; n = 0,1,2,3,\dots,m$$

Converges similarly to an arcsine-like pattern. Each term involves progressively higher odd powers of x divided by factorial-like products. For |x|≥1, a different expansion is used, often starting with ±ln(2x) (where the sign is chosen depending on whether x≥1 or x≤−1) and then subtracting a power series in 1/x.

$$\text{Arcsinh}(x) = \pm\ln(2x) + \frac{1}{2}\frac{1}{2x^2} - \frac{1\cdot3}{2\cdot4}\frac{1}{4x^4} + \frac{1\cdot3\cdot5}{2\cdot4\cdot6}\frac{1}{6x^6} - \frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8}\frac{1}{8x^8} - \cdots$$

+ if x ≥1 and – if x ≤ -1. This second approach converges fine for large |x| but is quite slow near x=1. Libraries typically check if |x|<1 and choose the first series, or if |x|≥1 and choose the second, bridging the middle zone carefully. Although neither series converges as swiftly as a simple factorial-based approach for small |x|, argument reduction (like dividing x if it's too big) and combining logs can help keep partial sums from ballooning.

If we denote the nth Taylor term, r, we can go from one Taylor term to the next using the following recurrence:

$$r_1 = \frac{1}{4x^2}$$

$$r_{n+1} = -r_n \frac{(2n+1)n}{2(n+1)^2x^2} \; for\; n = 1,2,3,\dots,m$$

Since arccosh(x) is defined for |x|≥1, it's common to see expansions beginning with ln(2x) and then subtracting further terms:

$$\text{Arccosh}(x) = \pm\ln(2x) - \left(\frac{1}{2}\frac{1}{2x^2} + \frac{1\cdot3}{2\cdot4}\frac{1}{4x^4} + \frac{1\cdot3\cdot5}{2\cdot4\cdot6}\frac{1}{6x^6} + \frac{1\cdot3\cdot5\cdot7}{2\cdot4\cdot6\cdot8}\frac{1}{8x^8} + \cdots\right)$$

This parallels the arcsinh expansion for large |x|, but here, x≥1 always, so it doesn't "switch expansions" the same way arcsinh(x) might at a threshold. Though straightforward, it converges more slowly as x moves closer to 1. If x is slightly above 1, partial sums can change gradually, requiring more terms. For truly large x, subtracting big terms from ln(2x) can also introduce floating-point issues, so some numeric libraries reduce the argument or use a direct $arccosh(x) = ln\left(x + \sqrt{x^2 - 1}\right)$ Approach. The series-based method is conceptually simple and works in principle for any x≥1.

Finally, arctanh(x) mirrors arctan(x) in structure but lacks alternating signs:

$$\text{Arctanh(x)} = \text{x} + \frac{\text{x}^3}{3} + \frac{\text{x}^5}{5} + \frac{\text{x}^7}{7} + \frac{\text{x}^9}{9} - \cdots, \text{where } |\text{x}| < 1$$

valid for |x|<1. Because these terms are all positive for x>0, the partial sums can get large if x is near 1, and the series converges more slowly. Unlike arctanh(x), it doesn't have a handy argument reduction formula to shrink |x|. If x is close to 1 or -1, you might need many terms to get a stable result in floating-point arithmetic. Some implementations instead rely on the relationship:

$$arctanh(x) = \frac{1}{2} ln\left(\frac{1 + x}{1 - x}\right)$$

For part of its domain, which often proves faster than summing many terms of a slower-converging power series.

The exponential function has the classic Taylor series.

$$\exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots$$

This expansion converges for every real x because factorial denominators (1!,2!,3!,…) grow extremely quickly, ensuring that higher-order terms become very small after some point. However, if x is very large and positive, the terms can be huge before the factorial overtakes the power of x. Summing many big intermediate terms might be slower or risk floating-point round-off issues. Conversely, if x is large and negative, you might prefer to compute exp(-x) as 1/exp(x). A common trick for negative x is to evaluate exp(|x|) and then invert it, which can sometimes be more efficient and numerically stable.

For moderate values of x, the series converges quickly because $x^n/n!$ Diminishes rapidly—particularly once n exceeds |x| by a few steps. But for truly large |x|, direct summation of the series may still require many terms.

The natural logarithm is somewhat trickier to handle with a straightforward Taylor approach because the basic expansion around x=1:

$$\ln(x) = (x - 1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \cdots$$

Converges well only if 0<x≤2. If x falls outside that range, you must apply an argument reduction strategy—such as repeatedly halving or doubling x—to bring it within (0.5,2) or

(1,2]. Another option is to rely on a different formula that typically converges faster, for instance:

$$\ln(x) = 2 \cdot \operatorname{artanh}\left(\frac{x-1}{x+1}\right) = 2\left(\frac{x-1}{x+1} + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \cdots\right)$$

Here, you effectively expand the powers of $\frac{x-1}{x+1}$, which often is small when x is near 1, making the series converge more quickly. If x is not near 1, you can still apply a transformation (like $x \to \frac{x}{2^n}$ until the fraction $\frac{x-1}{x+1}$ is modest in size, then multiply out the result. Such argument reduction typically speeds up the calculation and helps avoid slow convergence or numerical errors in floating-point arithmetic.

Optimizing Taylor Series Evaluations: A Systematic Approach

Before optimizing, we need to get an upper bound of the errors for the Taylor series. We will initially use the Taylor series for the sin(x), but the same error analysis can be applied to any function's Taylor series.

The Taylor series for sin(x) around x = 0 is:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} \ldots for\ any\ real\ value\ x$$

When truncating to n terms, we must consider the remainder term $R_n(x)$ to understand the error bound. We will use the first 10 Taylor terms of the sin(x) function as an example.

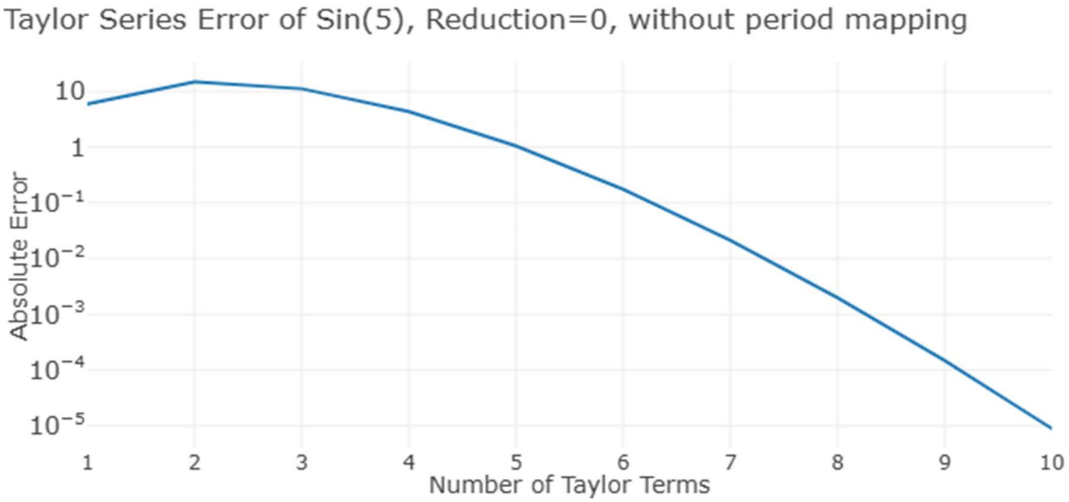
With 10 terms, our last included term is $x^{19}/19!$, and the first omitted term is $x^{21}/21!$

For $|x| > 0$, the absolute error is bounded for the first 10 terms of the sin(x) Taylor series:

$$|R_{10}(x)| \leq |x|^{21}/(21!)$$

This gives us our a priori error bound for any given x. e.g., For different values of x:

- x = 1: Error $\leq 1^{21}/21! \approx 2 \cdot 10^{-20}$
- x = 5: Error $\leq 5^{21}/21! \approx 9 \cdot 10^{-6}$
- x = 10: Error $\leq 10^{21}/21! \approx 2 \cdot 10^{1}$
- x = 20: Error $\leq 20^{21}/21! \approx 4 \cdot 10^{7}$

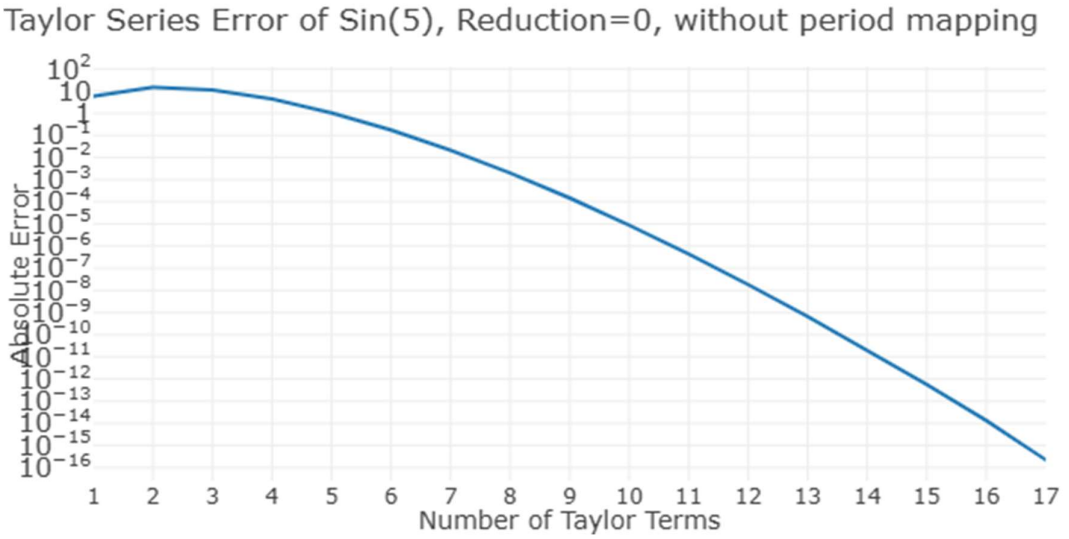


The picture above shows the first 10 Taylor terms for sin(5).
The error grows approximately as $|x|^{21}$ for large |x|. This rapid growth explains why the approximation becomes unusable for large arguments without periodic reduction. Notice that the errors grow first until the factorial in the denominator of a Taylor term overcomes the power of x in the numerator.

The ratio of consecutive terms in the Taylor series is:

$$|a_{k+1}/a_k| = |x^2/(k+1)(k+2)|$$

This ratio needs to be less than 1 for convergence, showing why larger |x| values require more terms for accurate approximation. From our example above, we needed 17 Taylor terms to reduce the error to ~ 10^{-16}



This leads us to observe that fewer Taylor terms are needed for smaller |x| values.

We have already alluded that argument reduction is a standard technique for speeding up the series evaluation. However, there are other techniques for optimizing Taylor series

evaluations of transcendental functions. Each technique builds upon the previous one to achieve computational efficiency and numerical accuracy. Most techniques are general and can usually be applied across all Taylor series.

In general, you can use period mapping and argument reductions.

Period mapping

Period mapping exploits the function's symmetry and periodic behavior. E.g., sin(x) has a period of 2π, and the primary period between 0 and 2π. While at the same time, for $0 \le x \le \pi$, there is a symmetry around $\pi/2$. Lastly, there is a sign of reflection since sin(-a)=-sin(a). All the period mapping ensures that sin(x) for any value of x can be brought into the interval between 0 and $\pi/2$.

The period mapping technique is only applied to functions with a period behavior, such as the trigonometric functions sin(x), cos(x), and tan(x). The first optimization involves mapping the input argument to a primary period interval, typically $[0, 2\pi]$ for trigonometric functions. This technique exploits functions' periodic nature to reduce the magnitude of the input argument.

For trigonometric functions, period mapping uses modulo operations to bring any trigonometric function into the primary period of $[-2\pi..2\pi]$.

Thereafter, you can explore symmetry mapping, which sometimes maps the function into a specific quadrant by changing the function sign.

For the sin(x), the argument mapping involves:

- Modulo operation with the function's period (2π)
- Sign preservation and quadrant mapping
- Use of symmetry properties

Example for sin(x):

```
x = x mod 2π     // Reduce to [-2π, 2π]
if x > π
   x = x-2π      // Map to [-π, 0]
if x < -π
   x = x+2π      // Map to [0, π]
if x < 0
   x =-x         // Record negative sign. Now [0..π]
if x > π/2
   x = π - x     // Map to [0, π/2]
```

Example for tan(x):

```
x = x mod π     // Reduce to [-π, π]
if x> π/2
   x = x-π      // Map to [0, -π/2]
```

```
if x< -π/2
   x = x+π       // Map to [0, π/2]
if x<0
   x=-x          // Record sign. Now [0..π/2]
```

There are also other symmetries you can explore e.g., sin(-x)=-sin(x), tan(-x)=-tan(x), cos(-x)=cos(x), exp(-x)=1/exp(x) etc.

This initial reduction is crucial as it prevents overflow in subsequent calculations, reduces the magnitude of intermediate terms, and improves overall numerical stability.

Argument Reduction

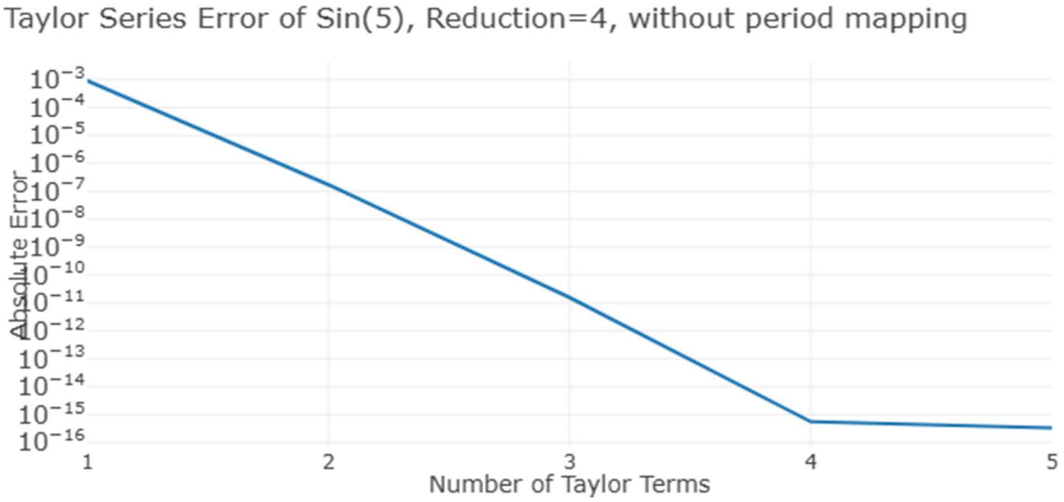
After period mapping, further reduction is possible using function-specific identities. This step is crucial for improving the Taylor series' convergence rate.

Sin(x): We know that the period mapping has brought in the value to the range of [0..π/2], and we notice that the Taylor series for sin(x) consists of the odd power of x. To help the convergence of the Taylor series for sin(x), we want the x argument less than one, and a good choice is < 0.1 or even smaller. This means that for every Taylor series, the numerator will drop with at least 1/100 per Taylor term. The sin(x) has several formulas for the multi-angle formula. However, only the odd multiple-angle formula can be done knowing only the value of sin(x). A good choice is, therefore, the Triple-Angle Formula:

$$\sin(3x) = 3\sin(x) - 4\sin^3(x)$$

Within the current range of [0..π/2], we only need to use this reduction a maximum of three times to get our x value below 0.1.

From the example before sin(5), we required 17 Taylor terms without period mapping to get the error down to ~ 10-6. Now, with argument reduction to minimize x to less than 0.1, you only need 5 Taylor terms after four argument reductions, bringing the argument x down from 5 to ~0.06. See the graph below.



Unsurprisingly, you can use period mapping and argument reduction to lower the number of Taylor terms needed and improve performance. See a later chapter explaining where accuracy is lost in the Taylor series with period mapping and argument reduction. However, as a general rule, always use both. The period mapping is nearly free to apply, with the positive effect of reducing the number of Taylor terms. In contrast, argument reduction can also significantly reduce the number of Taylor terms. Still, some moderate accuracy could be lost in the reduction and scale-back of the Taylor series value.

Cos(x): Most of the same argument for the Taylor series for cos(x) can be said. However, the choice of multi-angle reduction is wider since they all involve only the value of cos(x). A good choice is the triple-angle formula:

$$\cos(3x) = 4\cos^3(x) - 3\cos(x)$$

However, others can do like the double angle formula: $\cos(2x) = 2\cos2(x) - 1$.

Tan(x): As for the tan(x) we use the same technique as for sin(x) and cos(x) using the triple-angle formula for tan(x):

$$\tan(3x) = \frac{3\tan(x) - \tan^3(x)}{1 - 3\tan^2(x)}$$

Others can do as well, but the triple angle formula is a good blend of simplicity and efficiency.

Sinh(x): The hyperbolic function for sinh(x) is very similar to the Taylor series for sin(x). The only difference is the lack of alternating signs, which makes the Taylor series more stable without the risk of catastrophic cancellation failures, as we struggled with in the Taylor series for sin(x). Again, we use the triple angle formula for sinh(x):

$$\sinh(3x) = 3\sinh(x) + 4\sinh^3(x)$$

We repeat using the reduction formula until the argument is |x|<0.1

Cosh(x): The cosh(x) is similar to the cos(x) Taylor series but without the alternating sign. As for the sinh(x), the triple angle is adequate for our needs:

$$\cosh(3x) = 4\cosh^3(x) - 3\cosh(x)$$

We repeat using the reduction formula until the argument is |x|<0.1

Tanh(x): Similar techniques using the triple angle formula for tanh(x) are also used here.

$$tanh(3x) = \frac{3tanh(x) + tanh^3(x)}{1 + 3tanh^2(x)}$$

We repeat using the reduction formula until the argument is |x|<0.1

Arcsin(x): For arcsin(x), we use the using the reduction formula:

$$Arcsin(x) = 2 \cdot ArcSin(\frac{x}{\sqrt{2}\sqrt{1+\sqrt{1-x^2}}})$$

Then, we can restore the original value to compute by multiplying the reduction result with 2^k, where k is the number of reductions performed before the Taylor series computation.

Arccos(x): For Arccos(x), we used the identity:

$$Arccos(x) = \frac{\pi}{2} - Arcsin(x)$$

We rely on the arcsin(x) Taylor series.

Arctan(x): To reduce argument x to a smaller value, we use the identity:

$$Arctan(x) = 2 \cdot \arctan\left(\frac{x}{1+\sqrt{1+x^2}}\right)$$

Then, we can restore the original value to compute by multiplying the reduction result with 2^k, where k is the number of reductions performed before the Taylor series computation.

Arcsinh(x): One of the trickier ones is for arcsinh(x). Often, you don't see it implemented using argument reduction. And particularly for values near 1, the Taylor series converges slowly. However, you can benefit from using the addition formula for arcsinh(x):

$$\text{arcsinh(a)} + \text{arcsinh(b)} = \text{arcsinh}(a\sqrt{1+b^2} + b\sqrt{1+a^2})$$

Look at the formula for arcsinh(x) in the previous section.

$$\text{Arcsinh(x)} = x - \frac{1}{2}\frac{x^3}{3} + \frac{1\cdot 3}{2\cdot 4}\frac{x^5}{5} - \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{x^7}{7} + \frac{1\cdot 3\cdot 5\cdot 7}{2\cdot 4\cdot 6\cdot 8}\frac{x^9}{9} - \cdots, |x| < 1$$

And

$$\text{Arcsinh(x)} = \pm\ln(2x) + \frac{1}{2}\frac{1}{2x^2} - \frac{1\cdot 3}{2\cdot 4}\frac{1}{4x^4} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\frac{1}{6x^6} - \frac{1\cdot 3\cdot 5\cdot 7}{2\cdot 4\cdot 6\cdot 8}\frac{1}{8x^8} - \cdots, |x| \geq 1$$

There is one for |x|<1 and one for |x|≥1.

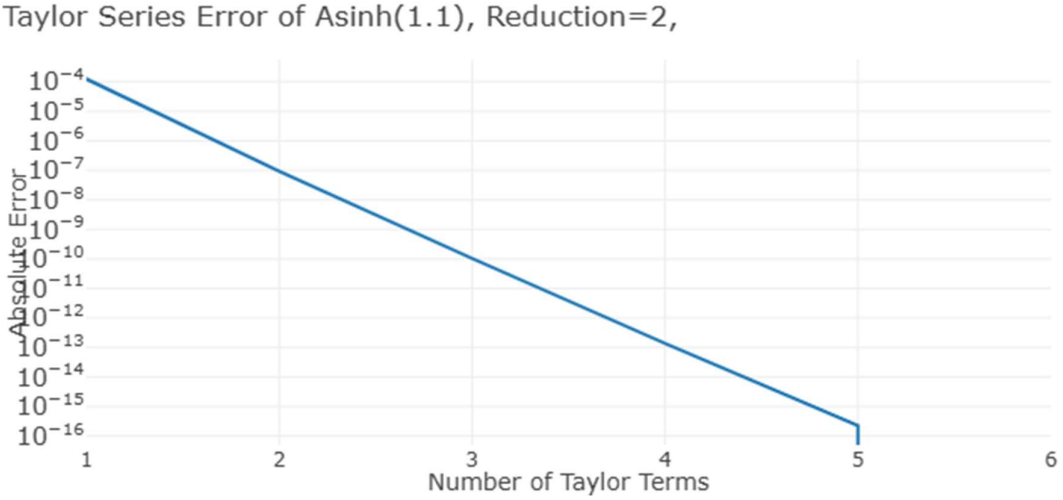
For the formula |x|≥1, it is beneficial to use argument *expansion* since the Taylor series will need fewer terms to complete. To simplify the argument reduction, we set a=b on the above addition formula and get:

$$2\text{arcsinh(a)} = \text{arcsinh}(2a\sqrt{1+a^2}) \Rightarrow$$

$$arcsinh(a) = \frac{1}{2} arcsinh(2a\sqrt{1+a^2})$$

We can replace a with $2a\sqrt{1+a^2}$ Before computing the Taylor series, the scale-back value can be obtained by multiplying the result by ½. If needed, we can repeat this argument expansion several times.

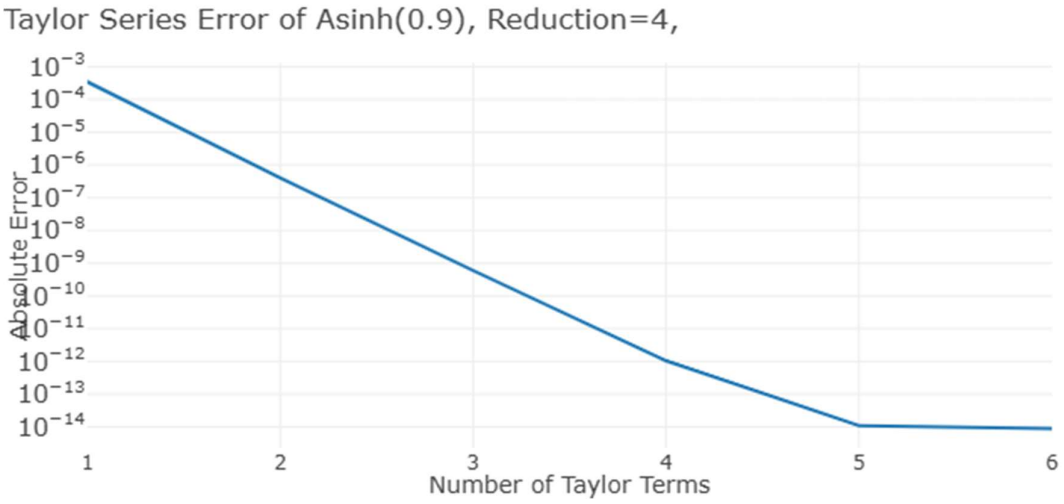
e.g., arcsinh(1.1) applying 2 argument expansion. The x=1.1 is expanded twice to ~22.3 before using the Taylor series and reaches a solution after 5 Taylor terms with an error < 10^{-16}



For |x|<1, we use the same formula the other way. We set x = $2a\sqrt{1+a^2}$ And now want to solve this for a. We get:

$$a = \sqrt{\frac{\sqrt{1+x^2}-1}{2}}$$

The reduction process continues until the argument falls below a threshold where Taylor series evaluation becomes efficient (typically |x| < 0.1). For arcsinh(0.9) using four argument reductions where the argument is reduced to x~0.05 before the Taylor series computation, it reaches the solution after 6 Taylor terms with an error ~$9 \cdot 10^{-15}$. If no argument reduction were used, 137 Taylor terms would be required to reach the same solution. Using argument reductions can save a lot when evaluating the Talor series.



Arccosh(x): A similar addition formula as for the arcsinh(x) exist for arccosh(x):

$$arccosh(a) + arccosh(b) = arccosh(ab + \sqrt{a^2 - 1}\sqrt{b^2 - 1}), where\ a, b \geq 1$$

To simplify, we set a=b and get:

$$2arccosh(a) = arccosh(a^2 + \sqrt{a^2 - 1}\sqrt{a^2 - 1}) = arccosh(2a^2 - 1)\ \Rightarrow$$

$$arccosh(a) = \frac{1}{2} arccosh(2a^2 - 1)$$

We can then replace a with $2a^2 - 1$ Increasing the argument results in faster convergence using fewer Taylor terms. Before computing the Taylor series, compute the original value by multiplying the result with ½. You can repeat the argument expansion several times.

ArcTanh(x): Similar to the arctan(x) the arctanh(x) has this equivalent identity:

$$Arctanh(x) = 2 \cdot \text{arctanh}\left(\frac{x}{1+\sqrt{1-x^2}}\right)$$

That can be used for argument reduction.

Exp(x): For moderate values of x, the Taylor series for e^x converges quickly because $x^n/n!$ Diminishes rapidly—particularly once n exceeds |x| by a few steps. But for truly large |x|, direct summation of the series may still require many terms. Numerical libraries often use other methods—like repeated squaring for large positive x or argument reduction—to avoid summing excessive terms. Nevertheless, the factorial-based series remains the most straightforward conceptual definition of exp(x) and works fine for small-to-medium x ranges.

We prefer to have our |x| < 1 to ensure the Taylor series converges more quickly. We can accomplish that using an argument reduction technique to work with a smaller number to converge faster to e^x using fewer *terms* of the Taylor series.

We can use the identity:$e^x = (e^{\frac{x}{2}})^2$ To reduce the argument with a factor of two, we can square the result after the Taylor iterations to find the correct value, for e^x.
Or, more generally, we can reduce the argument x for some k where:

$$e^x = (e^{\frac{x}{2^k}})^{2^k}$$

Iterate through the Taylor terms of the reduced argument. $\frac{x}{2^k}$ Then, square the result k times after the Taylor iterations. This makes sense since, for each Taylor term, you need to divide it with the factorial, which is much more time-consuming than squaring the result k times after the Taylor iterations.

However, instead of just squaring, we could use the Brent enhancement to preserve a higher accuracy in the process. We use the identity as suggested by Brent [9]:

$$e^x - 1 = \left(e^{\frac{x}{2}} - 1\right)\left(e^{\frac{x}{2}} + 1\right) \Rightarrow\ e^x - 1 = 2\left(e^{\frac{x}{2}} - 1\right) + \left(e^{\frac{x}{2}} - 1\right)^2$$

Ln(x): We prefer to have our x in a small neighborhood around one to ensure the Taylor series converges faster. We can do both argument mapping and argument reductions. In Argument mapping, we utilize that we can reduce x using:

$$ln(2x) = ln(x) + ln(2)$$

This can be continued until the x value is mapped into the range of [0.5..2]. e.g., ln(3)=ln(1.5)+ln(2). On the other hand, if x is very low, we can expand it using:

$$ln(\frac{x}{2}) = ln(x) - ln(2)$$

e.g., ln(0.25)=ln(0.5)-ln(2).

Argument reduction helps us work with numbers closer to one, which results in faster converging of ln(x) (using fewer *terms* of the Taylor series).

We can use the identity:

$$\ln(x) = \ln\left(\left(\sqrt{x}\right)^2\right) = 2 \cdot \ln\left(\sqrt{x}\right)$$

To reduce the argument by repeating, take the square root of x until it gets closer to 1. If we take k square roots, reducing $x \Rightarrow x^{\frac{1}{2^k}}$ and get closer to one. After the Taylor iterations, we can multiply the result with 2k to find the correct value of ln(x). If mapping was performed, adjust the result by adding m·Ln(2), where m is the number of times the mapping was repeated.

Efficient Taylor Series Evaluation

To evaluate the Taylor series efficiently, you need to apply

1. Domain check to ensure the argument is a valid domain.
2. Periodic Mapping and symmetry mapping of the argument.
3. Argument reduction using a well-known reduction formula.
4. Taylor series computation.
5. Scale back to compensate for any argument reduction performed.
6. Sign handling dealing with odd functions.

Points 2 and 3 were already detailed in the previous section, and they are followed by the evaluation of the Taylor series with automatic termination based on convergence criteria.

1. To generate the next Taylor term with as little effort as possible, the next Taylor term is calculated based on the result of the previous term, reducing the computational overhead per term. E.g., for the sin(x)
   `term[n] = term[n-1] * x² / ((2n)*(2n+1))  // for sin(x)`
2. Convergence Testing is performed at the relative error of a Taylor term and the accumulation of the sum of the Taylor term. Stopping when one of the two criteria has been fulfilled:
   `if |term[n]| < ε * |sum|:  return sum or you can use if |sum+term[n]| == |sum|`
3. We use the Epsilon machine for error control to ensure the highest technical accuracy.

Scale-Back Process

Points 5 and 6 are the final steps, which involve reversing the argument reduction steps to obtain the result for the original input.

```
Example for sin(x) with triple-angle reduction:
// If we reduced x → x/3 twice
y = sin(x/9)          // Taylor series result
y = y*(3 - 4y²)       // First scale-back
y = y*(3 - 4y²)       // Second scale-back
```

Error Propagation:

Each scale-back step can amplify errors, requiring careful consideration to compute correctly.

Performance Analysis

The effectiveness of these techniques can be demonstrated through error distribution analysis:

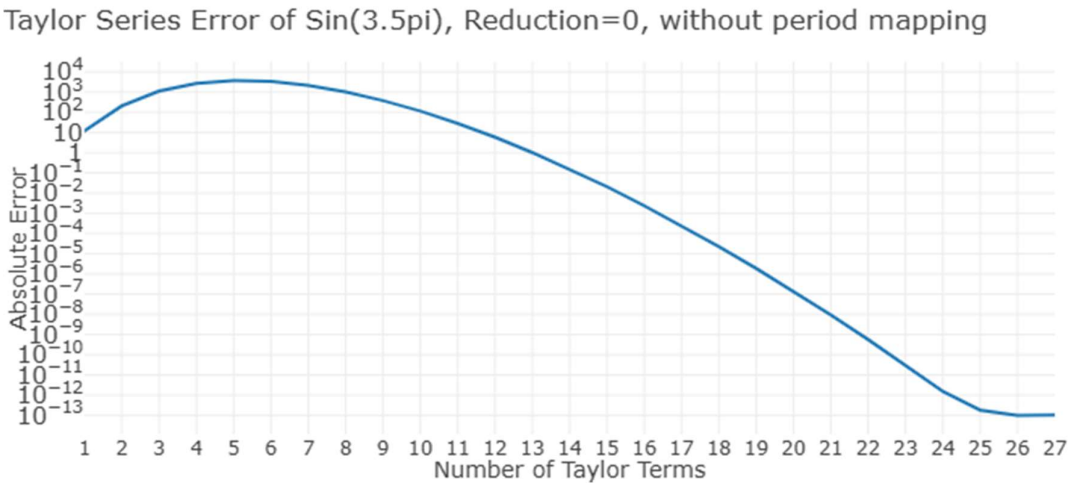
1. Without Reduction:
   - Error grows rapidly with |x|
   - Requires more terms for convergence
   - Subject to catastrophic cancellation
2. With Period Reduction:
   - Bounds maximum argument magnitude

- Reduces term count requirements
- Improves numerical stability

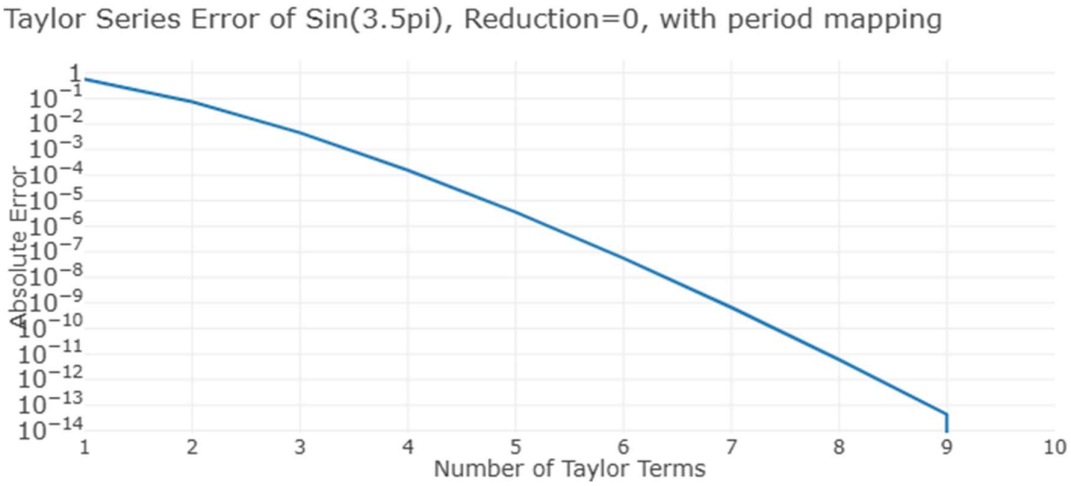
3. With Full Reduction:
   - Achieves optimal convergence
   - Minimizes computational cost
   - Maintains accuracy across the input range

Examples: let's analyze the sin(x) function at points x=3.5π. We let the Taylor terms stop automatically for the lowest error and perform no period mapping or argument reductions.

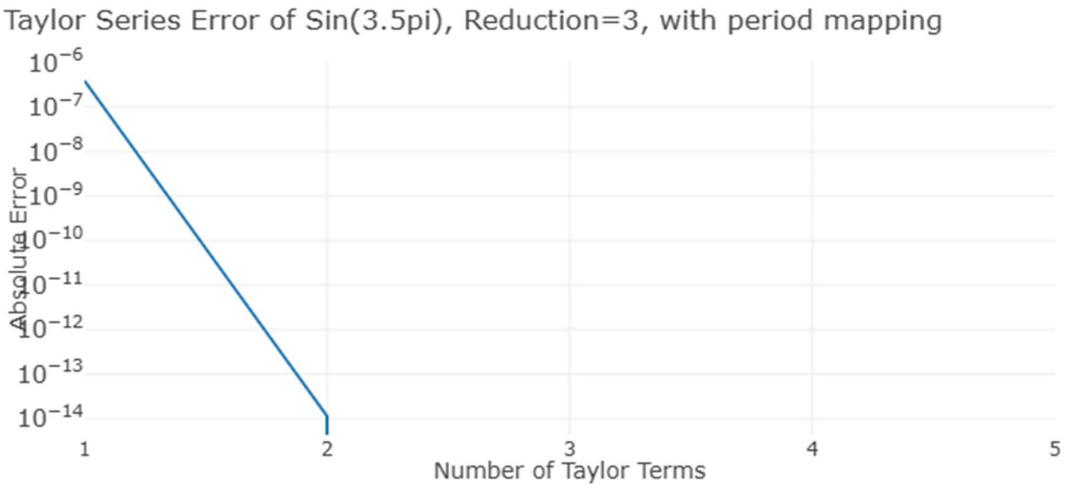
For x=3.5π, after 27 Taylor terms, we stopped evaluating further terms, and the accuracy was found to be ~ 1.0e-13. See the picture below. As expected, the error grows initially until the factorial in the denominator overpowers the power of x in the numerator.



If we add period mapping and bring the 3.5π down to 0.5π before we start the Taylor series and after only taking 10 Taylor terms, the error is 0.

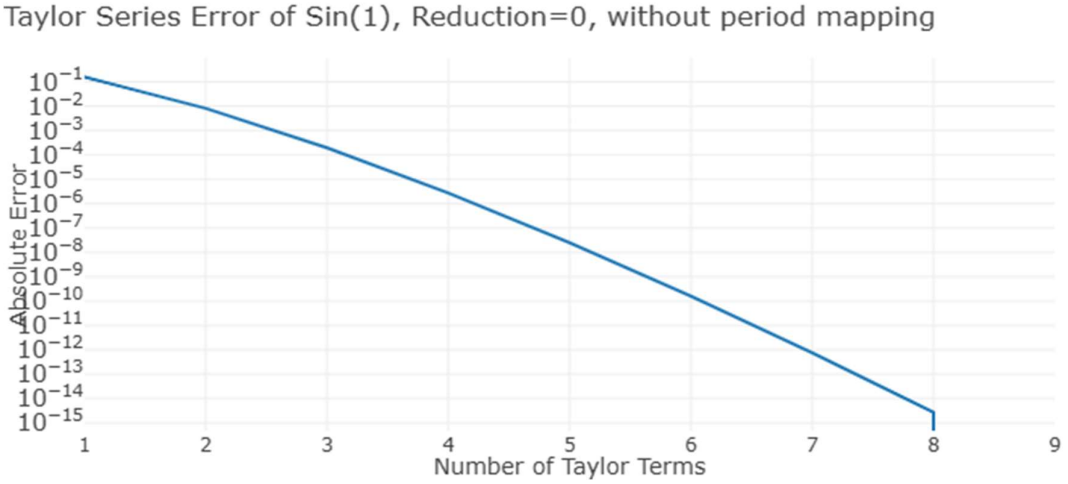


If we finally select auto reduction, we get after 3 Taylor terms and three argument reductions and an error of 0.



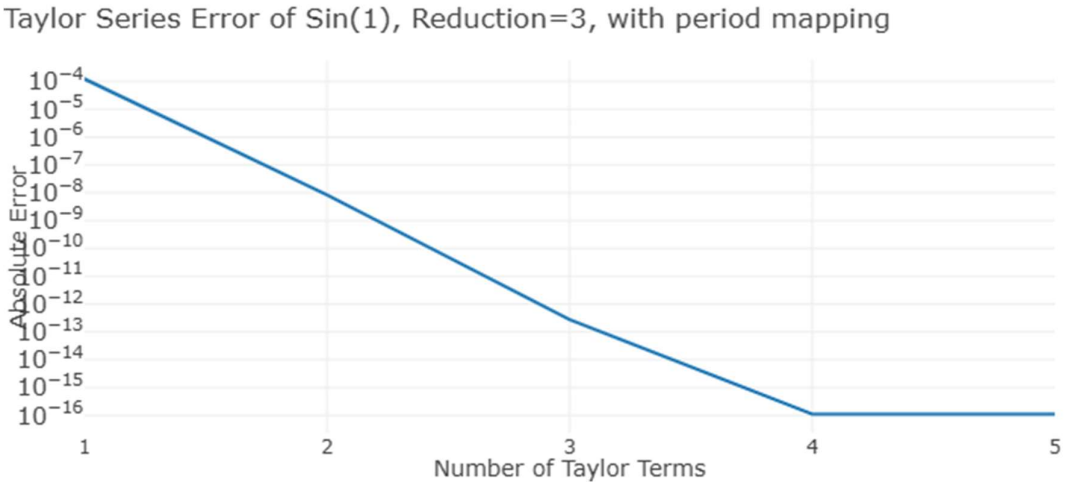
To some degree, you can substitute period mapping with a few extra reductions (if the x argument is not too large). For example, sin(3.5π) using only auto reduction and auto Taylor terms will require two extra reductions (5 instead of 3) to get similar results. However, if we sin(200π), you would need eight reductions instead of 3 with period mapping. However, period mapping carries considerably less accuracy loss than argument reduction. The best strategy is always to perform period mapping first and only do what is needed in argument reductions.

For x=1, after 8 Taylor terms, we can get the result without error. See the plot below. Again, applying argument reduction lowers the number of Taylor terms.



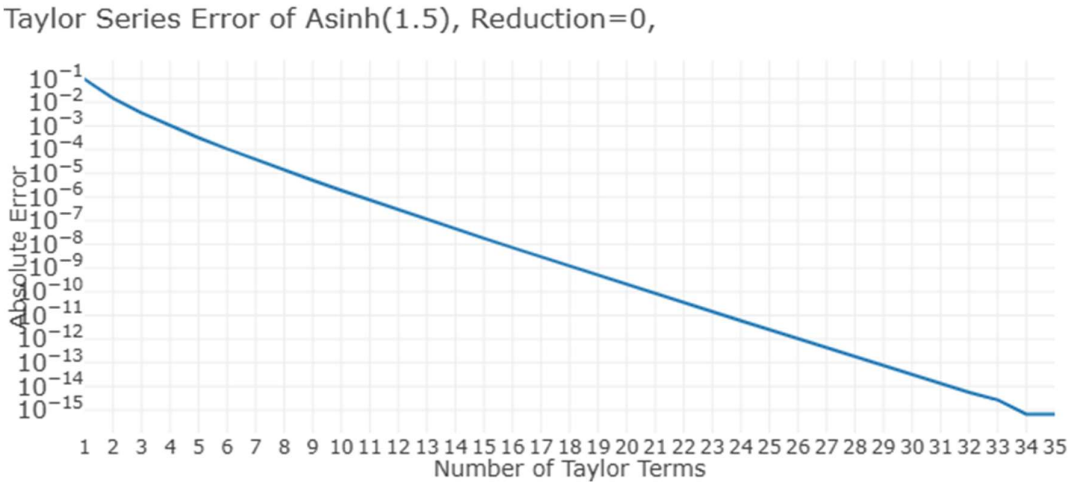
It doesn't matter whether or not period mapping is performed here because x=1 is within the target period mapping of [0..0.5π].

If we now apply the auto reduction. We get after 4 Taylor terms and applying 3 reductions an error of ~1.1e-16.

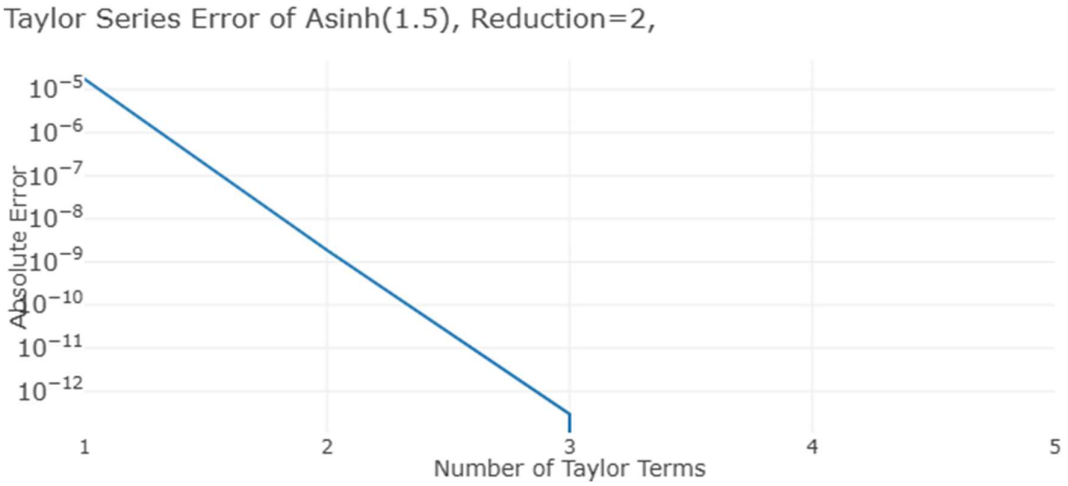


We can conclude that period mapping helps maintain accuracy for large x values greater than π/2, and argument reduction significantly reduces the number of Taylor terms needed to evaluate.

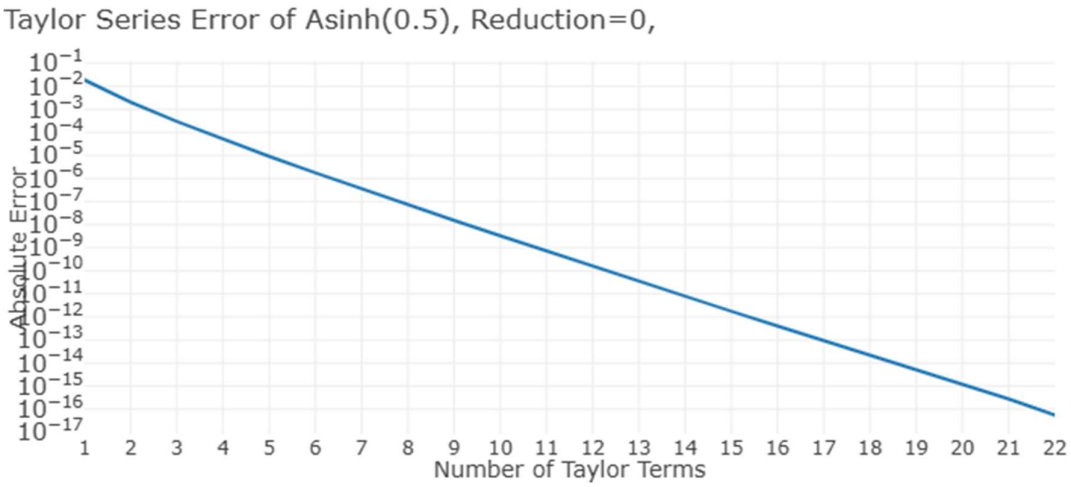
Another example is arcsinh(1.5). Remember there were two Taylor series. One for $|x|<1$ and oner for $|x| \geq 1$. Both series perform poorly close to one. See below for Arcsinh (1.5), where we needed 35 Taylor terms to get a result.



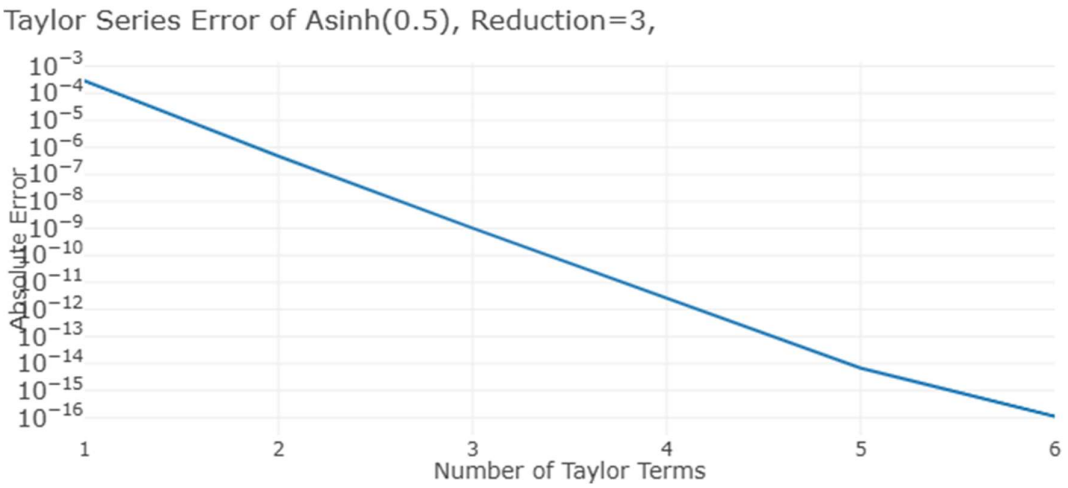
If we add the argument reduction auto (an argument expansion for $|x| \geq 1$), we only need 3 Taylor terms after two rounds of argument expansion.



The same benefit is seen with an argument of less than one. First, without argument reduction, we need 22 Taylor terms to complete.



And with auto argument reduction, we only need 6 Taylor terms.



Where Accuracy is lost in the Taylor series

Let me break this down to analyze where accuracy is most impacted, explore additional optimizations, and use the Sin(x) as an example, which can be translated to all the other functions.

Analysis of Accuracy Loss in Each Step

a) Mapping to [0..π/2]:

- Relatively low accuracy loss
- The main operations are:
  - Getting the sign (minimal loss)
  - Modulo operation with 2π (some loss due to floating-point representation of π)
  - Reduction to [0..π/2] using symmetry properties (minimal loss)

b) Triple Angle Reduction:

- Moderate accuracy loss
- Formula: $\sin(3x) = 3\sin(x) - 4\sin^3(x)$
- When we solve for $\sin(x) = \sin(3x)/3 + 4\sin^3(x)/3$
- Loss occurs due to:
  - Division operations
  - Cube operation, which can magnify existing errors
  - Potential cancellation between terms

c) Taylor Series Evaluation:

- Highest accuracy loss
- Issues include:
  - Alternating series causing cancellation errors
  - Large powers causing loss of significant digits
  - Factorial terms in denominators causing underflow/overflow risks
  - Truncation error from the limited number of terms

The most significant accuracy loss typically occurs in the Taylor series evaluation (step c), primarily due to:

1. Catastrophic cancellation in alternating series
2. Loss of significance when computing high powers
3. Roundoff errors accumulate across many terms

Additional Optimization Techniques

Here are additional optimization techniques that can be considered for implementing the sin(x) functions.

1. Minimax Polynomial Approximation:
   - Replace Taylor series with minimax polynomial coefficients

- Provides better accuracy with fewer terms
- Reduces cancellation errors

2. Table-based Approach for Small Angles:
   - Use a lookup table for very small angles
   - Linear interpolation between table values
   - Reduces computation for common cases
3. Improved Argument Reduction:
   - Use multiplication/division by powers of 2 where possible
   - More accurate than naive modulo operation
   - Careful handling of edge cases
4. Better Constants:
   - Use extended precision constants
   - Store pre-computed values for standard angles
   - Reduce intermediate rounding errors
5. Specialized Handling:
   - Different algorithms for different ranges
   - Optimize for common cases
   - The balance between speed and accuracy
6. Error Analysis and Compensation:
   - Track and compensate for rounding errors
   - Use compensated summation for series
   - Guard digits in critical operations

This will not always provide better accuracy and performance than a Taylor series approach with period mapping and argument reductions. The key improvements are:

Recommendation

This is my recommendation for the sin(x) function. If we use sin(x) and input value from [1..0.00001], we get the table below for the number of Taylor terms as a function of the argument. As can be seen, there's a sweet spot around 0.001 where we get excellent accuracy without excessive operations. Going more aggressive than this doesn't significantly improve accuracy but does increase computational cost.

X	Number of Taylor terms	Approximate Error
1	9	0
0.1	5	1e-17
0.01	3	1e-18
0.001	3	0
0.0001	2	0
0.00001	2	0

1. Optimal Reduction Factor:
   - The sweet spot appears to be around 0.01 to 0.001
   - Below 0.0001, we start seeing:
     - Diminishing returns in accuracy
     - Increased computational cost

- Risk of underflow in floating-point operations

2. Trade-offs:
   - Each reduction step (dividing by 3) reduces the magnitude by ~1.6 bits
   - More aggressive reduction means:
     - Fewer Taylor series terms needed (huge accuracy benefit)
     - Less catastrophic cancellation
     - More triple-angle reduction steps
     - More multiplication operations
     - Risk of losing significance in very small numbers
3. Optimal Strategy:
   - Reduce until $|x| < 0.001$
   - Use only 3-4 Taylor series terms instead of the usual 7-8
   - This gives approximately 14-15 digits of accuracy for most inputs
   - The computational cost is lower than using more Taylor terms
4. Additional Considerations:
   - For very large inputs, you might want to use a two-phase reduction:

1. First, reduce to $[0..\pi/2]$ using modulo & function symmetry.
2. Then, apply aggressive triple-angle reduction
   - For values near zero, skip reduction entirely
   - Consider using table lookup for final evaluation when x is very small

The key insight is that the error from triple-angle reduction grows roughly linearly with the number of reductions. In contrast, the Taylor series error grows rapidly with larger inputs. This makes aggressive reduction highly advantageous.

Conclusion

These optimizations demonstrate significant improvements in evaluating the Taylor series for transcendental functions, particularly trigonometric and hyperbolic functions. The strategic use of period mapping, argument reduction, and efficient series evaluation enhances numerical stability and reduces computational overhead. These broadly applicable techniques make them important in environments requiring speed and precision, such as real-time systems and scientific simulations.

The interactive tool developed for this study provides users with hands-on insight into the Taylor series' behavior and the impact of optimization strategies. Future research could refine these methods by focusing on hybrid approximation techniques or enhanced floating-point error compensation. In the tool developed for exploring the Taylor series, all the formulas presented in this paper are implemented together with the argument reduction novelty for the Arcsinh(x) and Arccosh(x).

These techniques mentioned in this paper have been added to a web page you can use to explore the Taylor series. See https://www.hvks.com/Numerical/webTaylor.html

References

1. IEEE Standard for Floating-Point Arithmetic (IEEE 754)

2. David Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic"
3. Press, W. H., et al. (2007). Numerical Recipes: The Art of Scientific Computing (3rd ed.). Cambridge University Press.
4. Fast Computation of Math Constants in arbitrary precision. HVE Fast Gamma, Beta, Error, and Zeta functions for arbitrary precision.
5. Fast Exponential calculation for arbitrary precision math. HVE Fast Exp() calculation for arbitrary precision
6. Fast logarithm calculation for arbitrary precision math. HVE Fast Log() calculation for arbitrary precision
7. Fast Trigonometric function for arbitrary precision. HVE Fast Trigonometric calculation for arbitrary precision
8. Fast Hyperbolic functions for arbitrary precision. HVE Fast Hyperbolic calculation for arbitrary precision
9. Richard Brent & Paul Zimmermann, Modern Computer Arithmetic, Version 0.5.9 17 October 2010; http://maths-people.anu.edu.au/~brent/pd/mca-cup-0.5.9.pdf