Next Silicon: CM Home Assignment

Dorde Zivanovic
LinScale

April 24, 2025

Contents

1 Introduction

This report contains the responses to the tasks stated in the home project pdf. It is accompanied with the repository that contains reproducible solutions with the installation instructions alongside: experiments and tests according to the task requirements.

2 Code Analysis and Documentation

In this section we analyze the existing code shown in Algorithm 1.

2.1 The Existing Implementation: Code Drawbacks

The code is written for C and follows the following bad practices:

1. Reusing the variable multiple times, `(float)M_PI` and `2.0f * (float)M_PI`, (lines 3, 4, 6, 6, 5, 7 of Algorithm 1);
2. Not using auto in order to automatically deduce types since the results of all the statements are known;
3. Cleaning if loop to be more understandable: `fmodf` returns the result in the range $(-2\pi, 2\pi)$. Then, now it is obvious that one checks whether the number is outside of the range $[-\pi, \pi]$, and then updates **x** for 2π period, so the method works from the number in the range $[-\pi, \pi]$;
4. Adding more verbosity ();
5. Reusing variable names -> more verbose names should be used in order to improve readability of the code. The c ompiler will optimize for the least number of variables/registers to be used;
6. Renaming function names and migrating these functions to the corresponding headers and sources that would contain the custom maths functions.

2.2 Drawbacks: numerical and implementation issues

Here, I will give state several main drawbacks in terms of the implementation and numerical accuracy. The division by In the next subsection, I will list the drawbacks related to the method itself.

2.3 Mathematical Analysis

Let us state the general Taylor series formula that is implemented in Algorithm 1.

1: **Input:** A float (IEEE-754) number
2: **Output:** A float (IEEE-754) sine value of this number computed using Taylor Series.
3: **Steps:**

```
float fp32_custom_sine(float x)
{
    x = fmodf(x, 2.0f * (float)M_PI);
    if (x > (float)M_PI)
        x -= 2.0f * (float)M_PI;
    else if (x < -(float)M_PI)
        x += 2.0f * (float)M_PI;
    float result = 0.0f;
    float term = x;
    float x_squared = x * x;
    int sign = 1;
    for (int n = 1; n <= 7; n += 2)
    {
        result += sign * term;
        sign = -sign;
        term = term * x_squared;
        term = term / (float)(n + 1);
        term = term / (float)(n + 2);
    }
    return result;
}
```

Algorithm 1: Algorithm with Code Listing

Theorem 2.1 (Theorem 5.19 from [1, p. 113]): Let f be a function having finite n-th derivative $f^{(n)}$ everywhere in an open interval (a, b) and assume that $f^{(n-1)}$ is continuous on the closed interval $[a, b]$. Then, for every x in $[a, b], x \neq c$, there exists a point x_1 interior to the interval joining x and c such that

$$f(x) = f(c) + \sum_{k=1}^{n-1} \frac{f^{(k)}(c)(x-c)^{(k)}}{k!} + \frac{f^{(n)}(x_1)}{n!}(x-c)^n.$$

A corollary of Theorem 2.1 when we set $c = 0$ is a Maclaurin Series.

Corollary 2.2 (Maclaurin Series): Let f be a function having finite n-th derivative $f^{(n)}$ everywhere in an open interval (a, b) and assume that $f^{(n-1)}$ is continuous on the closed interval $[a, b]$. Assume that $c \in [a, b]$. Then, for every x in $[a, b], x \neq 0$, there exists a point x_1 interior to the interval joining x and 0 such that

$$f(x) = f(0) + \sum_{k=1}^{n-1} \frac{f^{(k)}(0)(x)^{(k)}}{k!} + \frac{f^{(n)}(x_1)}{n!}x^n.$$

Let us now set $a = -\pi, b = \pi$ and $f(x) = sin(x)$. The Maclaurin Series becomes for sin function:

Corollary 2.3: For $x \in \mathbb{R}, x \neq 0$ and $n \in \mathbb{N}$, and $0 < |x_1| < |x|$ the approximation of a degree n is

$$sin(x) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} (-1)^i \frac{x^{2i+1}}{(2i+1)!} + L(x, n).$$

where

$$L(x,n) = \begin{cases} \frac{f^{(n+2)}x_1}{(n+2)!}x^{n+2}, & \text{if } n \bmod 2 = 0, \\ \frac{f^{(n+1)}x_1}{(n+1)!}x^{n+1}, & \text{if } n \bmod 2 = 1. \end{cases}$$

Thus, the method stated in Algorithm 1 has an algorithmic error $L(x, 7)$ which is smaller than $\frac{x^8}{8!}$. Now, we obtain

2.4 Failures

Based on Corollary 2.3, we can notice that Algorithm 1 is incorrect for $x = 0$. Similar, the farther the x is from 0, the error is larger since x^8 grows exponentially. There are several ways to solve these problems:

1. Have another Taylor series expansion for numbers around π and $-\pi$. In this case we would manually calculate the $\sin x$ for π and $-\pi$.
2. Add more terms in Taylor series expansion. It is important to find the optimal degree for the balance of accuracy and performance.
3. Implement another method.

2.5 Test Plan

3 Conclusion

Summarize your findings or thoughts here. [1]

References

[1] Tom M Apostol. Mathematical analysis. Narosa Publishing House Pvt. Ltd., 1985.