Improved Taylor Approximation of Sine and Cosine

Wayne Anderson

24/05/2024

1. Introduction

Sine and Cosine values are widely calculated in some specific fields like digital signal processing and simulations. For many general-purpose processors, Sine and Cosine are not implemented in hardware that can be calculated by calling one specific instruction. Instead, many applications, e.g., standard C math library, implement their own Sine and Cosine functions. However, these math libraries do not cover all aspects, even though they are compatible with many contexts.

DECX is a high-performance scientific computational library, including modules like linear algebra (BLAS), digital signal processing (DSP), computer vision (CV) and neural network (NN). DECX is designed for wide broad of platforms such as x86-64 series and Arm64 series. Lots of implemented functions in DECX involve SIMD (Single Instruction Multiple Data). This article introduces an algorithm based on Taylor approximation on calculating Sine and Cosine values in a fast and accurate way, which is also especially designed for SIMD cases.

2. Method

2.1 Taylor approximation of Cosine and Sine

Taylor series at $x = 0$ of Sine and Cosine are shown in equation (2.1.1) and (2.1.2):

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} \tag{2.1.1}$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^{n-1} x^{2n}}{(2n)!} \tag{2.1.2}$$

Taylor approximation is straight forward and easy to implemented. The graph of Sine and Cosine functions and their approximations with $n = 6$ are given in Figure 2.1.1 and 2.1.2. However, the approximated curves of both Sine and Cosine diverge from the desired values outside one period near the origin, since the approximations are taken at point $x = 0$.

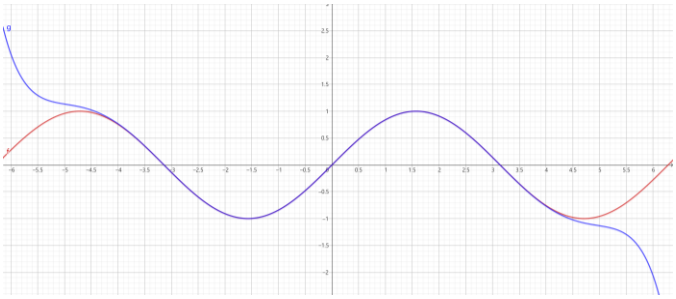

Figure 2.1.1 Sine and its $n = 6$ approximation

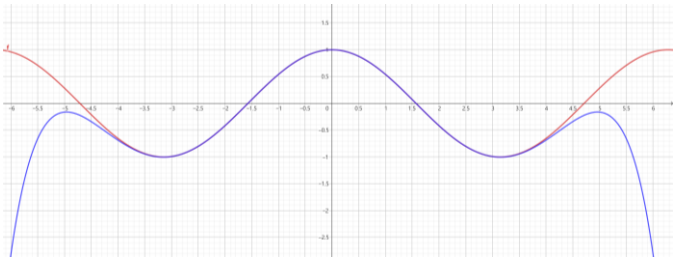

Figure 2.1.2 Cosine and its $n = 6$ approximation

Whereas, both Sine and Cosine are periodic functions, before the approximation, the input angle should be normalized within one period. Taylor approximation usually gives the most accurate result at the approximated point, with accuracy decreased as moving further from that point. Hence, to preserve the accuracy as possible, the angle should be limited in range $[-\pi/2, \pi/2]$.

However, in practice, reducing the angle to such range takes up more additional machine cycles, and the mapping of the result is complicated. Hence, the angle is normalized to range $[0, \pi]$ instead. Equations (2.1.3) to (2.1.5) describe how the angle is normalized and how the result is mapped in Cosine cases:

$$K = floor\left(\frac{|x|}{\pi}\right) \tag{2.1.3}$$

$$x' = |x| - \pi K \tag{2.1.4}$$

$$\cos(x) = \begin{cases} -\hat{g}(x'), & K \text{ is odd} \\ \hat{g}(x'), & K \text{ is even} \end{cases} \tag{2.1.5}$$

Where $\hat{g}(x')$ denotes the approximated result of the normalized angle by the Taylor polynomial; $floor(x)$ rounds value x towards zero (rounding down).

A MATLAB script is created to measure the accuracy of direct Taylor approximation. The input angles are presented as an array, ranging from -2π to 2π. They are increasing with step of $4\pi/1920$. To simulate the performance in single precision floating point number (fp32) cases, the data in MATLAB is converted to fp32. The estimated Cosine values and the error are shown in Figure 2.1.3.

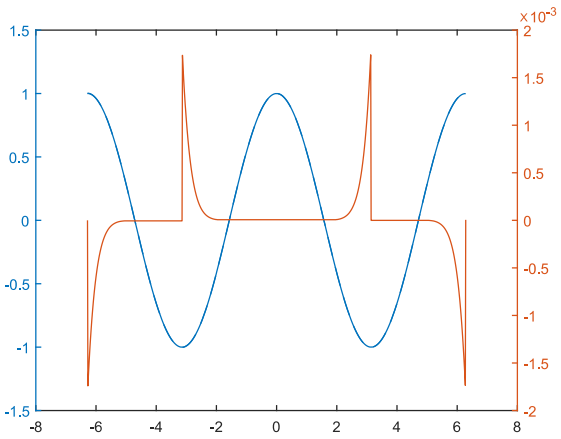

Figure 2.1.3 Estimated Cosine and its error

The total error is obtained by the net error:

$$\chi^2 = \sum_{i=1}^{N} |y - \hat{y}|^2 \tag{2.1.6}$$

The total error is 0.1324 in this case. The result is unsatisfying when demanding for high precision, such as Fourier transform of a long signal. The figure sees four significant peaks in the error curve, at points $x = \pm 3.129, y = 1.1741 \times 10^{-2}$ and $x = \pm 6.27, y = -1.1741 \times 10^{-2}$, respectively.

2.2 CORDIC

CORDIC algorithm [1] is a new iterative approach to calculate Sine and Cosine values. Different from Taylor approximation, CORDIC is more about geometrical perspective. The procedure rotates the angle iteratively to approach the desired angle. Both its Sine and Cosine values are updated during the process.

In mathematical expression, the process starts from $v_0 = [1,0]$, as shown in Figure 2.2.1. The vector rotates

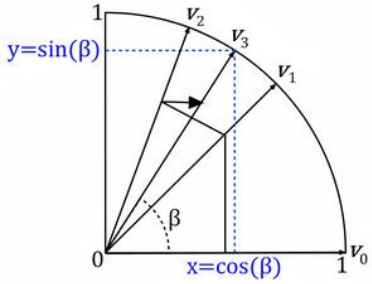

Figure 2.2.1 CORDIC algorithm

gradually towards the desired angle step by step. Let γ_i denotes the angle (step) rotated at the i[th] step. The rotation matrix is:

$$R_i = \begin{bmatrix} \cos(\gamma_i) & -\sin(\gamma_i) \\ \sin(\gamma_i) & \cos(\gamma_i) \end{bmatrix} \tag{2.2.1}$$

and can be written as:

$$R_i = \cos(\gamma_i) \begin{bmatrix} 1 & -\tan(\gamma_i) \\ \tan(\gamma_i) & 1 \end{bmatrix} \tag{2.2.2}$$

Hence, the x and y component of the vector in the next iteration will be:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \cos(\gamma_i) \begin{bmatrix} 1 & -\tan(\gamma_i) \\ \tan(\gamma_i) & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{2.2.3}$$

The process rotates angle γ_i each time with γ_i satisfies $\tan(\gamma_i) = 2^{-i}$, so that the iteration is converged. Additionally, to ensure the rotation orientation is always towards the desired angle, the orientation sign should be considered. Substituting $\tan(\gamma_i)$ with 2^{-i}, equation (2.2.3) can be written as:

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = \cos(\gamma_i) \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} \tag{2.2.4}$$

$$\sigma_i = \begin{cases} +1, \beta_i < \theta \\ -1, \beta_i \geq \theta \end{cases}$$

$$\beta_{i+1} = \beta_i + \sigma_i \gamma_i$$

The final value of the iteration, is obtained by iterating n times.

$$\begin{bmatrix} x_n \\ y_n \end{bmatrix} = K \prod_{i=1}^{n} \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{2.2.5}$$

$$K = \prod_{i=1}^{n} \cos(\gamma_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{1 + 2^{-2i}}} \tag{2.2.6}$$

CORDIC algorithm avoids tons of multiplications of floating-point numbers, so that it's especially efficient in hardware design and some embedded contexts. Because multiplying or division by 2 can be implemented using bit shifting. However, for many modern CPUS, multiplication of floating-point numbers is not expensive, since they are equipped with FPUs (floating-point processing unit).

Additionally, CORDIC converges slower than Taylor approximation to obtain the same precision. A MATLAB script is created to test the accuracy of CORDIC with different iteration times. The results are given in Table 2.2.1.

Iteration times	Total error	Peak error (Absolute value)
6	9.4993	0.0295
12	0.1534	4.7952×10^{-4}
16	0.00983	0.00983
24	5.7961×10^{-5}	2.8639×10^{-7}
32	4.8029×10^{-5}	2.8625×10^{-7}

Table 2.2.1 Performances of CORDIC

Compared to the direct Taylor approximation, to achieve the same accuracy, CORDIC algorithm should iterate 12 times, whereas Taylor only needs to iterate 6 times (calculates until the sixth term of the polynomial). Conclusively, for the CPUs equipped with FPUs, which calculate FP multiplication almost as fast as the other operations, CORDIC has no advantage over Taylor approximation.

2.3 Improved Taylor approximation of Cosine

Refer to Figure 2.1.3, the error begins to rise significantly in range of

$$x \in [\pm\left(\tfrac{3\pi}{4} + N\pi\right), \pm(\pi + N\pi)],\ N = 0,1,2,3\ldots$$

The error sharply decreases at the beginning of the next period. Mapping to the normalized angle, the maximum error happens around the points where the normalized angle falls in range

$$\left[\frac{3\pi}{4}, \pi\right] \qquad (2.3.1)$$

Taylor approximation gives the most accurate result at the approximated point, and the accuracy decreases as moving further from that point. Hence, the trend of the error curve is reasonable.

Cosine waveform is not only axially symmetrical but also centrally symmetrical. DECX maps the angle values that falls into the range given in (2.3.1) to range $\left[0, \frac{\pi}{2}\right]$. The process can be described by equation (2.3.2):

$$\cos(x) = -\cos(\pi - x) \qquad (2.3.2)$$

Hence, the angle is mapped closer to x=0, which gives a more accurate result. The results and the corresponding errors are given in Figure 2.3.1:

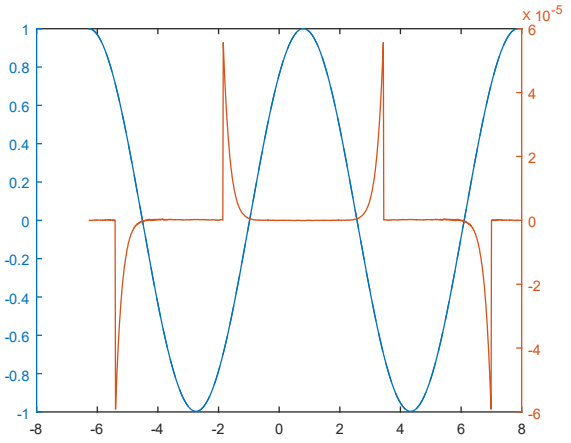

Figure 2.3.1 Estimated Cosine and its error

The total error is 0.0033, which is 40 times less than the direct approximation. However, peaks still exist around points $x = \pm\left(\frac{\pi}{2} + N\pi\right)$, $N = 0,1,2,3\ldots$, where the approximation gives the least accurate results. However, if regard these points as where the Sine values are approximated, they can be calculated in a more accurate way, that is using Taylor approximation of Sine function at $x = 0$. The mapping can be expressed by equation (2.3.3):

$$\cos(\mathrm{x}) = -\sin\left(\mathrm{x} - \frac{\pi}{2}\right) \qquad (2.3.3)$$

In DECX, the normalized angles which satisfy inequality (2.3.4) will be mapped to new values and calculated with Taylor polynomial of Sine.

$$\left|x' - \frac{\pi}{2}\right| < \frac{\pi}{4} \qquad (2.3.4)$$

The estimated curve and error are given in Figure 2.3.2. The total error is 5.8726×10^{-5}, which is significantly less than the method with compensating only at range $[\pm\left(\frac{3\pi}{4} + N\pi\right), \pm(\pi + N\pi)]$. The peak error is 3×10^{-7} approximately, which is capable to give an accurate enough result in high-precision cases.

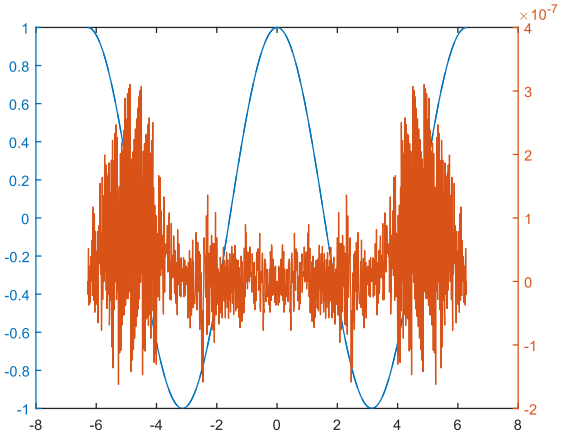

Figure 2.3.2 Estimated Cosine and its error

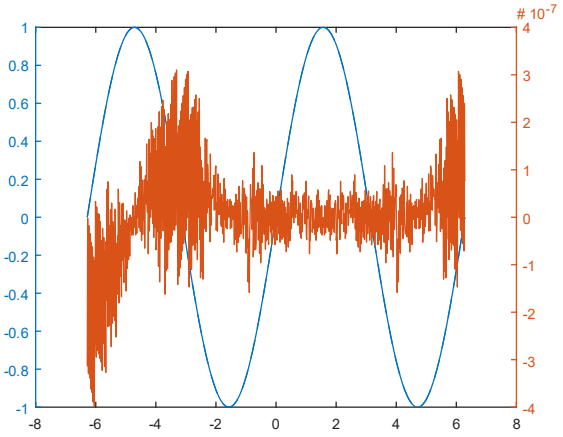

Figure 2.3.3 Estimated Sine and its error

One of the advantages of the improved version of Taylor approximation is that, the approximating points are distributed evenly within a period, that is 0_+, $\pi/2$, and π_-. The subscripts x_+ and x_- mean the point lies infinitely near in front of and behind point x, respectively.

Hence, the Sine value can be directly obtained by:

$$\sin(x) = \cos\left(x - \frac{\pi}{2}\right) \qquad (2.3.5)$$

without worrying any precision loss. The results and errors of estimated Sine values are show in Figure 2.3.3. The total error is 6.1966×10^{-5}, and the absolute value of the peak error is 4×10^{-7}, indicating that Cosine and Sine have the similar performance in the improved Taylor approximation method.

3. Implementation with SSE4 and AVX2

SSE4 [2] processes data in 128bit XMM registers. In fp32 mode, each instruction processes 4 numbers. Although in Intel context, Intel short vector math library (SVML) [3] is provided, the SSE4 and AVX2 [4] version of Sine and Cosine are provided, in some cases, implementing these functions from scratch benefits a lot. The accuracy level can be customized and bargain with executing speed. Moreover, the ideas are the same across all similar SIMD contexts. The implementations are similar and only a few instructions should be changed when migrating codes from SSE4 to AVX2 or Arm NEON [5], for example.

From the analysis in section 2, the pseudocodes are shown below:

```
cos_fp32(angle) PROC
    angle = abs(angle);
    K = floor(angle / pi);
    angle = angle - K * pi;
    if angle > pi*3/4
        angle_iter = pi - angle;
        res = cos_taylor(angle_iter, n=6);
    end
    if abs(angle-pi/2) < pi/4
        angle_iter -= pi/2;
        res = sin_taylor(angle_iter, n=6);
    end
    if abs(angle-pi/2) < pi/4 or mod(K,2)==1
        res = -res;
    end
ENDP
```

Recall the equations (2.1.1) and (2.1.2) for Taylor approximation of Sine and Cosine, ignoring the factors of each term in the polynomials, the procedures of Taylor approximation of Sine and Cosine are the same, but the Sine one should multiply a x to its result in the

end. That is, equation (2.1.1) can be written as:

$$\sin(x) = x\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n}}{(2n+1)!} \tag{3.1}$$

Hence, the Taylor approximation steps of Sine and Cosine can be integrated into one, and the factor of each term can be selected by intrinsics *_mm_blendv_ps()* or its corresponding instruction *blendvps*.

The period number K is an integer. In the implementation, K is presented as packed 4-int32 numbers. The conditional sign inversion of the results can be done by:

1. Bitwise-and K with a *__m128i* vector that four elements are all 1 to extract the LSB of each value in K.
2. If the value(s) in K is(are) odd, the bitwise-and result will be 0x01, 0x00 otherwise.
3. Left shift the LSB of the result all the way to the MSB. The shifted result then bitwise-xor with the results obtained from the Taylor approximation. According to IEEE754 standard for fp32 number, the MSB denotes its sign and 1 for negative, 0 for positive, this procedure inverses the sign of the results conditionally.
4. Conditional inversion according to whether it's from Sine or Cosine is similar with step 3. By performing inequality (2.3.4) by *_mm_cmp_ps()* or *vcmpps*, which involved AVX2 instruction set, a mask with each element either 0x00 or 0xFFFFFFFF can be obtained. The mask then bitwise and with a vector with 4 0x80000000 numbers to obtained the indication at MSB, just like step 3.

4. References

[1] Volder, J. (1959, March). The CORDIC computing technique. In Papers presented at the the March 3-5, 1959, western joint computer conference (pp. 257-261).

[2] SSE4. (2024, May 24). In Wikipedia. Retrieved July 2, 2024, from https://en.wikipedia.org/wiki/SSE4

[3] Intel Corporation. (2021). Intrinsics for short vector math library operations. Retrieved May 24, 2024, from https://www.intel.com/content/www/us/en/docs/cpp-compiler/developer-guide-reference/2021-8/intrinsics-for-short-vector-math-library-ops.html

[4] Advanced Vector Extensions. (2024, May 24). In Wikipedia. Retrieved May 24, 2024, from https://en.wikipedia.org/wiki/Advanced_Vector_Extensions

[5] Arm Limited. (2024). Neon. Retrieved May 24, 2024, from https://developer.arm.com/Architectures/Neon