

Math and Trig Functions in Embedded Microcontrollers

John S. Jacob

April 2011

www.wallingup.com

www.johnsjacob.com

www.Smart-Inventor.com

www.youtube.com/TheInnovationGuru

Embedded microcontrollers are an incredibly useful technology, capable of making almost any ordinary mechanical machine or device vastly more capable, efficient, reliable, precise, or safe. The value added is not in the processor itself, but in how good the software is that runs on it.

Often in robotics, machine control, automation or in any number of applications, it is desirable to use trigonometric, exponential, logarithmic, or other non-algebraic functions (meaning other than the four basic operations: +, -, *, /). But many embedded microprocessors do not have full math libraries built-in. Programmers are limited to basic integer and floating-point arithmetic operations.

Often in my career I have encountered situations where a highly desirable control algorithm could not be implemented because the programmers thought they were limited to the mathematical functions available in processor function libraries.

Damn ignoramuses! Where do they find these people? How does one manage to get a university degree in Computer Science without knowing even the most rudimentary facts of Numerical Analysis? Right here is an example of a compact, efficient and highly effective Sine function that I doodled onto the back of an envelope in an idle moment. It can be implemented on any processor with floating point capability. It serves as an example of how non-algebraic functions such as trig functions can be implemented on devices with limited clock cycle and memory budgets.

This algorithm is presented in “pseudo-code” which is a way of communicating concepts in easily-readable code that is also readily adaptable to many specific programming languages. I mention this because some “programmers” I have known have questioned the specific syntax rather than using their brains for something.

I present the algorithm first and then explain the mathematics of why it works. I will also describe its performance characteristics and applications.

The input argument x can be any real number, but it's faster if x is in the range $-\pi \leq x \leq \pi$. Because $\text{Sine}(x)$ is periodic, x can be readily adjusted from any value down to an equivalent value within this range. In other words, $\text{Sine}(x) = \text{Sine}(x - N \cdot 2\pi)$ for any integer value of N . If x is really big, simply subtract whole integer multiples of 2π until x is within the optimal range. Note that x physically represents an angular variable in radians (not degrees). If you are working with degrees, convert to radians using the factor $\pi/180$. Set the value of Error to the desired level of precision.

```

Function Sine(x)
real Term = x; integer 2n = 0; real Error = 0.00001;
real Sine = x; real sqx = x*x;
While Term > Error
    2n = 2n + 2;
    Term = Term*(-1)*sqx/(2n*2n + 2n);
    Sine = Sine + Term;
End

```

That's it! This short function uses only 3 additions, 2 multiplications, one floating point division and one floating point comparison per iteration. Even better, two of the additions and one of the multiplications are integer operations, which require even fewer clock cycles.

But how many iterations are needed to get a useful result? In the worst case, with x in the vicinity of 3.14, just 14 iterations are needed to get an absurdly precise 15 decimal places of accuracy. Usually in practice only 5 places of precision would be needed, achieved in just 7 iterations. For values of x less than 1 and Error = 0.00001, this algorithm terminates in 4 iterations or less.

What makes it so efficient? It is based on the series expansion of the trigonometric Sine function:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

The secret to making it efficient is to factor each term out of each subsequent term. This has the impact that only the minimum number of operations is performed for a given accuracy requirement and makes sure that calculations once performed are never repeated. In other words, no term takes any longer to compute than any previous term, no matter how long it goes on. Without this trick, imagine trying to calculate the 10th term: $(-1)^{10} * x^{21} / 21!$ (!' Means factorial, e.g. $5! = 5 * 4 * 3 * 2 * 1$) Each additional term would take exponentially longer to compute, making such an algorithm utterly impractical.

This optimized algorithm is also careful to perform common operations (like squaring x) one time only, storing the result in a variable. The derivation of the formula is trivial, and the result is this:

$$Next\ Term = Previous\ Term * (-1) * \frac{x^2}{2n(2n + 1)};$$

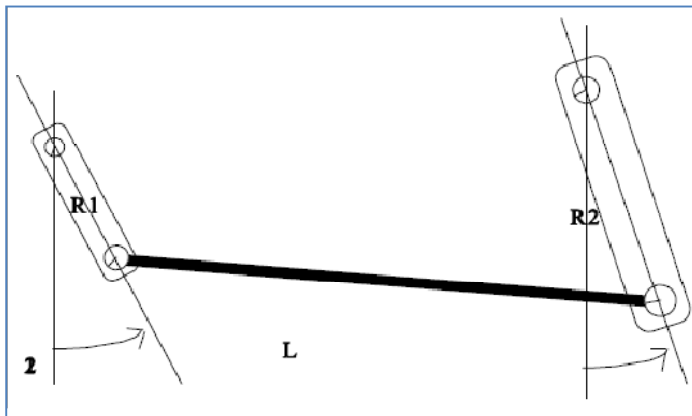
where n is the number of iterations. Sine(x) is the running total of these successively smaller terms. The starting term is simply equal to the input argument x. When the term becomes smaller than the required precision, further contributions to the total can be ignored and the iteration stops.

Occasionally it is essential to keep the function from running longer than a certain number of clock cycles. In a real-time controller like a PLC device, the processor has a fixed amount of time in which to come up with its next control signal output. If this is the case and one is unsure about the rate of convergence, it may be advisable to implement a "hard stop" in the iterative loop. An example of a

“hard stop” would be when the number of iterations reaches a pre-set limit, the function terminates and outputs its current best-guess of the required value and (optionally) an estimate of the remaining error (which can be worked out mathematically).

Application Example

The need for trig functions in embedded controllers arises for example when a mechanism converts the rotary motion of an actuator into linear motion, as in the diagram below. A high-performing controller would be able to estimate the amount of rotation that it needs to create on arm 1 that will result in a required amount of rotation on arm 2.



The means of predicting the results of the controller’s action comes from the equation

$$R_1 \sin(\theta_1) = R_2 \sin(\theta_2).$$

This is approximately valid when the length L of the connecting rod is much, much longer than either of the lever arms.

In a typical application, the controller would be called on to move lever 2 by say, 5 degrees, by adjusting the position of lever 1. Assume that the controller can measure the angle of lever 2. It rotates lever 1, hunting around, until the angle of lever 2 is finally correct. That strategy might be OK for ordinary applications, but in order to vastly improve critical performance, the controller needs to predict by how much it should rotate lever 1 in order to create the desired effect, and only then rely on feedback from the sensor on lever 2 to fine-tune the position. And for that, it needs to be able to compute sine functions very quickly.

Many other examples exist in areas such as controlling internal combustion engines, positioning solar collectors, steering and feathering wind turbines, controlling stationary and mobile robots, and satellite attitude control just to name a few.

This is merely one example of how a non-algebraic function can be implemented on an embedded microprocessor controller without the benefit of a complete maths library loaded into memory. Most of the trig functions and their inverses can be implemented in a similarly efficient and compact way. Logarithmic, exponential and even more esoteric functions like Bessel functions, Riemann Zeta functions and so on can similarly be implemented if there is some performance benefit in doing so.

All it takes is the ability to massage a series expansion into the most computationally effective and rapidly converging format. It is usually beneficial if not essential to assure the input argument is well-formed within the optimal range, and to create a bullet-proof termination condition.

References and useful resources:

Jacob J.S., "**Conversion and Control of an Amphibious ATV for use as an Autonomous Mobile Robot**," Master's Thesis, Utah State University, 1998. (available from www.wallingup.com)

Beyer W.H. ed., "**Standard Math Tables**," 27th edition, CRC Press, 1984.

Gradshteyn I.S. and Ryzhik I.M., "**Table of Integrals, Series, and Products**," corrected and enlarged edition, Academic Press, 1980.

Abramowitz M. and Stegun I.A., eds., "**Handbook of Mathematical Functions**," 9th printing, Dover, 1972.

Kincaid D. and Cheney W., "**Numerical Analysis**," 2nd ed., Brooks/Cole Publishing, 1996.

Goodwin G.C. and Sin K.S., "**Adaptive Filtering, Prediction and Control**," Prentice-Hall, 1984.

Ogata K., "**Modern Control Engineering**," 2nd ed., Prentice Hall, 1990.