CORDIC: How Hand Calculators Calculate

Article *in* The College Mathematics Journal · March 2009

DOI: 10.4169/193113409X469514

1 author:

Alan Sultan
City University of New York - Queens College
40 PUBLICATIONS **165** CITATIONS

CORDIC: How Hand Calculators Calculate

Alan Sultan

Alan Sultan is a professor of mathematics at Queens College of the City University of New York. He is the author of two books and several research articles, and is currently heavily involved in the math education of prospective high school math teachers. He enjoys singing bass in the Oratorio Society of Queens, and loves to see how the math we teach can be applied.

Almost every calculus teacher I ask answers the question, "How do calculators compute sines and cosines?" with the words, "Taylor Polynomials." It comes as quite a surprise to most that, though it is reasonable to use Taylor polynomials, it is really a method known as CORDIC that is used to compute these and other special functions. CORDIC was discovered by Jack Volder in 1959 while working for the Convair corporation and was developed to replace the analog resolver in the B-58 bomber's navigation computer. What was needed was a program that could compute trigonometric functions in real time without the use of much hardware. CORDIC, an acronym for COordinate Rotation DIgital Computer, was the solution. CORDIC is fast; much faster than Taylor series for the low level hardware used on a calculator, though when you first see it, it is hard to believe.

What is nice about CORDIC is that it connects mathematics usually learned in high school with computer hardware, making it especially interesting to students who wonder where the math they learn might be used. The method can be adapted to compute inverse trigonometric functions, hyperbolic functions, logarithms, exponentials, and can even be modified to do multiplication and division! (See [**2**] or [**3**] for more on this.) This amazing method is a genuinely key part of calculator technology and knowledge of it can benefit both teacher and student.

We present CORDIC in its original binary form even though now it uses binary-coded decimal (BCD). What we present is the heart of the algorithm, focusing on a special case—the computation of sines and cosines. (For the history of computing transcendental functions see [**4**]. For its actual implementation in hardware, see [**1**].)

The basics

What makes CORDIC fast are two facts: (1) When we take a binary number and multiply it by 2^n, we shift the binary point n places to the right and when we divide by 2^n we shift the binary point n places to the left. (2) The operations on a computer that are cheapest and fastest to perform are (A) addition and subtraction, (B) comparing numbers to see which is larger or smaller, (C) storing and retrieving numbers from memory, and (D) *shifting the binary point*.

Addition and subtraction are very fast, but not multiplication or division. The machine does compute multiplications and divisions by powers of 2 very quickly however, by just shifting the binary point. CORDIC exploits this, and thus uses *only* operations (A)–(D) to evaluate sines and cosines. The heart of the algorithm is a series of cleverly performed rotations.

Rotations

In high school, students learn to rotate a point $P = (x_0, y_0)$ through an angle of θ degrees. The rotated point has coordinates $P_1 = (x, y)$ where

$$x = x_0 \cos\theta - y_0 \sin\theta,$$
$$y = x_0 \sin\theta + y_0 \cos\theta,$$

which can be written in matrix form as $P_1 = R_\theta P_0$ where

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}. \tag{1}$$

Now suppose we rotate the point $(1, 0)$ by θ radians. Then the rotated point, $P_1 = (x, y)$ has coordinates $(\cos\theta, \sin\theta)$. However, we can't use (1) to rotate by the angle θ unless we already know $\sin\theta$ and $\cos\theta$. But finding them is our goal! So what are we to do?

CORDIC breaks down rotation by θ radians, into rotations by smaller angles $\theta_0, \theta_1, \theta_2, \theta_3, \ldots$. These are cleverly chosen so that these rotations can be calculated using angles hardwired into the computer. It then generates points P_1, P_2, P_3 etc. on the unit circle which approach the point $(\cos\theta, \sin\theta)$. (See Figure 1.)

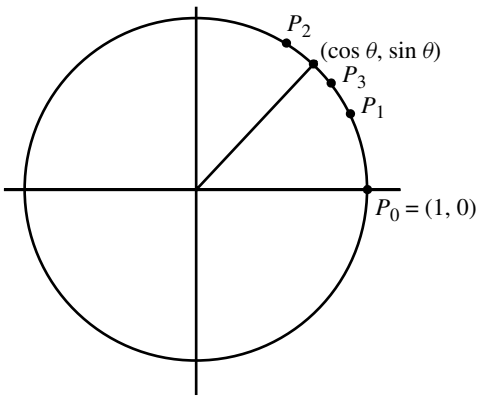

Figure 1.

Hard-wiring the machine

Which angles does CORDIC hard-wire? They are the angles whose tangents are of the form $\frac{1}{2^n}$, where $n = 0, 1, 2, 3, \ldots$.

Let us list these θ_i's:

$$\theta_0 = \tan^{-1}(1) = \frac{\pi}{4} \approx .785398,$$
$$\theta_1 = \tan^{-1}(1/2) \approx .463648 \approx \frac{1}{2},$$
$$\theta_2 = \tan^{-1}(1/4) \approx .244979 \approx \frac{1}{4},$$

$$\theta_3 = \tan^{-1}(1/8) \approx .124355 \approx \frac{1}{8},$$

$$\theta_4 = \tan^{-1}(1/16) \approx .062419 \approx \frac{1}{16}.$$

We observe that when θ is small, $\tan\theta \approx \theta$. Thus, we don't even need to store all the values θ_i, since eventually $\tan\theta$ and θ are the same for practical purposes.

A numerical example

We go through an example, to illustrate how CORDIC works. We set as our goal the angle $\theta = 1$ radian, and try to build it from the θ_i's we have chosen. We start with $z_0 = \theta_0 = .785398$. (The letters z_i will keep a running total of our rotation.) This is too small, so the algorithm now adds θ_1 attempting to bring the total up to 1 radian. We get

$$z_1 = \theta_0 + \theta_1 \approx .785398 + .463648 = 1.249046.$$

This is more than 1 radian, so the algorithm backs up. It subtracts θ_2, to get

$$z_2 = \theta_0 + \theta_1 - \theta_2 \approx 1.249046 - .244979 = 1.004067.$$

This is still more than 1, so the algorithm backs up again getting

$$z_3 = \theta_0 + \theta_1 - \theta_2 - \theta_3 \approx 1.004067 - .124355 = 0.879712.$$

This is less than 1, so we add $\theta_4 = .062419$, to get

$$z_4 = \theta_0 + \theta_1 - \theta_2 - \theta_3 + \theta_4 \approx .879773 + .062419 = 0.942192,$$

and so on. With 40 terms, which is what many machines use,

$$z_{39} = \theta_0 + \theta_1 - \theta_2 - \theta_3 + \theta_4 + \theta_5 + \theta_6 + \theta_7 + \theta_8 + \cdots - \theta_{39}. \tag{2}$$

(Notice the $-\theta_{39}$.) At each step, the machine checks to see if the running total z_i is more or less than the angle of 1 radian, and either adds or subtracts the next θ accordingly.

Simultaneously, the algorithm performs the corresponding rotations. So, following (2) we initially rotate $(1, 0)$ by θ_0, then by θ_1, then rotate by $-\theta_2$, then rotate by $-\theta_3$ and so on.

After 40 rotations, we get

$$\begin{aligned} P_{40} &= R_{-\theta_{39}} \cdot \cdots \cdot R_{-\theta_2} \cdot R_{\theta_1} \cdot R_{\theta_0} \cdot P_0 \\ &= \begin{pmatrix} \cos(-\theta_{39}) & -\sin(-\theta_{39}) \\ \sin(-\theta_{39}) & \cos(-\theta_{39}) \end{pmatrix} \cdots \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{pmatrix} \begin{pmatrix} \cos\theta_0 & -\sin\theta_0 \\ \sin\theta_0 & \cos\theta_0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \end{aligned} \tag{3}$$

Forty rotations may seem like a lot of work. We will see.

How good is our approximation?

For any iterative procedure to be useful, we must have an idea of how accurate our answers are. The following theorem tells us.

Theorem 1. *Using the method described in this paper, after $n+1$ iterations, the approximations obtained for* $\sin\theta$ *and* $\cos\theta$ *are within* $\frac{1}{2^n}$ *of the true values, provided* $-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$. *Thus, the CORDIC Method converges to* $(\cos\theta, \sin\theta)$.

A detailed proof is given in [**5**] and uses a combination of induction, trig identities and the Mean Value Theorem. If the inequality $-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$ is false, then we subtract multiples of 2π to yield an angle between 0 and 2π and then, using trig identities, replace θ with an angle between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$.

Theorem 1 guarantees that the error we make in computing sin 1 and cos 1 by computing P_{40} is not more than $\frac{1}{2^{39}} = 1.8190 \times 10^{-12}$. Thus, we have at least 10 places of decimal accuracy!

Taming equation (3)

Equation (3) certainly looks daunting. You must be thinking "This method is far more difficult than using Taylor polynomials." And so it is, unless we can multiply those matrices rapidly. In this section we show how to accomplish this.

Each matrix in the above product (3) is of the form

$$R_{\theta_i} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix}$$

which can be rewritten as

$$R_{\theta_i} = \cos\theta_i \begin{pmatrix} 1 & -\tan\theta_i \\ \tan\theta_i & 1 \end{pmatrix}, \tag{4}$$

and now we see how the tangent function comes into play. Since by design, $\tan\theta_i = \frac{1}{2^i}$, (4) becomes

$$R_{\theta_i} = \cos\theta_i \begin{pmatrix} 1 & -\frac{1}{2^i} \\ \frac{1}{2^i} & 1 \end{pmatrix}. \tag{5}$$

(Note: $R_{-\theta_i}$, is of the same form with off diagonal entries switched.) Finally, since for $-\frac{\pi}{2} \le \theta \le \frac{\pi}{2}$, $\cos\theta = \frac{1}{\sqrt{1+\tan^2\theta}} = \frac{1}{\sqrt{1+(2^{-i})^2}}$, (5) can be written as

$$R_{\theta_i} = \frac{1}{\sqrt{1+(2^{-i})^2}} \begin{pmatrix} 1 & -\frac{1}{2^i} \\ \frac{1}{2^i} & 1 \end{pmatrix}$$

Now substituting into (3), we get

$$P_{40} = \frac{1}{\sqrt{1+(2^{-39})^2}} \begin{pmatrix} 1 & \frac{1}{2^{39}} \\ -\frac{1}{2^{39}} & 1 \end{pmatrix} \cdots \frac{1}{\sqrt{1+(2^{-1})^2}} \begin{pmatrix} 1 & -\frac{1}{2^1} \\ \frac{1}{2^1} & 1 \end{pmatrix} \cdot \frac{1}{\sqrt{1+(2^0)^2}} \begin{pmatrix} 1 & -\frac{1}{2^0} \\ \frac{1}{2^0} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Pulling the constants, $\frac{1}{\sqrt{1+(2^{-i})^2}}$ to the front of this matrix product, we get that

$$P_{40} = K \begin{pmatrix} 1 & \frac{1}{2^{39}} \\ -\frac{1}{2^{39}} & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\frac{1}{2^1} \\ \frac{1}{2^1} & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2^0} \\ \frac{1}{2^0} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

where

$$K = \frac{1}{\sqrt{1+(2^{-39})^2}} \frac{1}{\sqrt{1+(2^{-38})^2}} \cdots \frac{1}{\sqrt{1+(2^{-1})^2}} \frac{1}{\sqrt{1+(2^{0})^2}},$$

and we are on our way. K is a constant! Once we compute it, we can, and do, hard-wire it into the machine. It turns out that $K \approx .607252935$. So finally (3) becomes

$$P_{40} \approx .607252935 \begin{pmatrix} 1 & \frac{1}{2^{39}} \\ -\frac{1}{2^{39}} & 1 \end{pmatrix} \cdots \begin{pmatrix} 1 & -\frac{1}{2^1} \\ \frac{1}{2^1} & 1 \end{pmatrix} \begin{pmatrix} 1 & -\frac{1}{2^0} \\ \frac{1}{2^0} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{6}$$

Observe that all entries in (6) are hard-wired into the machine. No computation is needed to obtain them. Then, as we shall see, multiplying the matrices is very fast.

The speed with which the matrices are multiplied

We observe that

$$\begin{pmatrix} 1 & -\frac{1}{2^n} \\ \frac{1}{2^n} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} a - \frac{1}{2^n} b \\ \frac{1}{2^n} a + b \end{pmatrix}. \tag{7}$$

Now $\frac{1}{2^n} b$ is computed by shifting the binary point in b. So our first entry in the product, $a - \frac{1}{2^n} b$, uses a shift and a subtraction (two fast operations). Our second entry, $\frac{1}{2^n} a + b$, uses a shift and an addition (another 2 fast operations). Thus, when we multiply the two matrices from (7) we do 4 very fast operations. We then store the two results for a total of 6 operations, all of which are among the computer's fastest.

Now consider all of the operations in (6). We must execute 40 matrix multiplications, each of which takes only 6 fast operations. The bottom line: this requires only $40 \times 6 = 240$ fast operations. (After we multiply all of the matrices in (6) we must not forget to multiply our final result by $K = .607252935$.)

If, for the example, $\theta = 1$ radian, we multiply just 11 matrices, and then multiply by K, we obtain

$$(\cos 1, \sin 1) \approx (0.5397, 0.84185).$$

According to Theorem (1) we should be within $\frac{1}{2^{10}} = .00097656$ of the correct results, and in fact,

$$\cos 1 \approx 0.5403 \quad \text{and} \quad \sin 1 \approx 0.84147. \tag{8}$$

Of course, if we multiply all 40 matrices together and then multiply by K, the approximations we get for $\sin\theta$ and $\cos\theta$ will be accurate to more than 10 places, as we have pointed out.

Final remarks

CORDIC is a robust and practical algorithm, preferred by calculator manufacturers because it requires relatively low level electronics. It is perfect for calculators, and good for the company's bottom line. What is amazing about CORDIC in addition, is that by just slightly modifying the method, we get *all* of the usual calculator functions. The details of this are technical and some are even proprietary. But one cannot help but be astounded by the fact that essentially one algorithm, and its variations, can be used to compute sines, cosines, tangents, inverse trig functions, hyperbolic functions, logarithms, exponentials, roots—even do multiplications and divisions. It is just mind-boggling!

References

1. R. Andraka, A survey of CORDIC algorithms for FPGA based computers, FPGA '98, *Proc. of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, Monterey CA, 1998, 191–200.
2. N. Eklund, CORDIC: Elementary function computation using recursive sequences, this JOURNAL **32** (2001) 330–333.
3. ———, CORDIC: Elementary function computation using recursive sequences, *Electronic Proceeding of the ICTCM* **11** (1998) C027.
4. C. W. Schelin, Calculator function approximation, *Amer. Math. Monthly* **90** (1983) 317–325.
5. J. Underwood and B. Edwards, How do calculators calculate trigonometric functions? *Educational Resources Information Center (ERIC)* document ED461519. Also available at `http://www.math.ufl.edu/~haven/papers/paper.pdf`.

On Achieving the Petaflop

(Sung to the tune of the Ode to Joy in Beethoven's Choral Symphony)

Hectic, hectic apoplectic
Life in the Computer Age

Things start fast
Then go yet faster

Soon we'll reach the final stage

Mega-tasks in nano-seconds
Glad we've achieved the Petaflop

If we are forced to go yet faster
Things will come to a

Dead
Stop

—David Seppala-Holtzman (dholtzman@sjcny.edu)

[David Seppala-Holtzman is a professor of mathematics at St. Joseph's College where he has taught for the past 27 years. His mathematical interests are many and varied, his love for the subject, in general, being the only constant.]