HOW DO CALCULATORS CALCULATE?

Bruce H. Edwards
Robert T. Jackson
Jeremy M. Underwood

Department of Mathematics
358 Little Hall
University of Florida
Gainesville, FL 32611
(352) 392-0281
be@math.ufl.edu

ABSTRACT

How do calculators quickly produce the values of transcendental functions? For instance, how does the TI-85 calculate $\tan 1.5$? The calculator does not use series or polynomial approximation, but rather the so-called CORDIC method. We will present the main ideas of the CORDIC method along with some elementary examples.

1. Introduction.

The CORDIC method for approximating function values is used on all popular graphing calculators, including the TI-81, TI-82, TI-85 and HP-48G. This algorithm is not based on calculus nor polynomial approximation, but instead involves a more elementary system of iterative equations. The CORDIC (**CO**ordinate **R**otation **DI**igital **C**omputer) algorithm is relatively new, introduced in 1959 by Volder [4] to calculate trigonometric function values. It was later extended to hyperbolic, logarithmic, and other functions by Walther in 1971 [5].

The hardware restrictions of calculators require that the elementary functions be computed using only additions, subtractions, digit shifts, comparisons, and the recall of a small number of stored constants. As we will see, the CORDIC method only uses these five basic operations. It is interesting to note that even multiplication and division can be computed by the CORDIC method.

Because it is time-consuming to convert from base 10 to base 2, the CORDIC algorithm is programmed in base 10. However, it is easier to understand the essential ideas using binary arithmetic. Hence, multiplication by 2^k, where k is an integer, is nothing more than a digit shift. See the UMAP module [1] for an elementary treatment of the CORDIC method using base 10 arithmetic.

We will not address in full detail the important theoretical issue of the convergence properties of the CORDIC method, and the interested reader is referred to the literature [2]. In general, the algorithm consists of performing n iterations of the three equations given below, and the convergence theorems guarantee that the resulting answer is within a certain error bound that depends on n.

2. The CORDIC Algorithm.

The general CORDIC algorithm consists of the following three iterative equations.

$$\begin{aligned} x_{k+1} &= x_k - m\delta_k y_k 2^{-k} \\ y_{k+1} &= y_k + \delta_k x_k 2^{-k} \\ z_{k+1} &= z_k - \delta_k \sigma_k \end{aligned}$$

The constants m, δ_k, and σ_k depend on the specific computation being performed, as explained below.

1. m is either 0, 1, or -1. $m = 1$ is used for trigonometric and inverse trigonometric functions. $m = -1$ is used for hyperbolic, inverse hyperbolic, exponential, and logarithmic functions, as well as square roots. Finally, $m = 0$ is used for multiplication and division.

2. δ_k is one of the following two signum functions:

$$\delta_k = \operatorname{sgn}(z_k) = \begin{cases} 1, & z_k \geq 0 \\ -1, & z_k < 0 \end{cases} \qquad \text{or} \qquad \delta_k = -\operatorname{sgn}(y_k) = \begin{cases} 1, & y_k < 0 \\ -1, & y_k \geq 0 \end{cases}$$

The first is often called the *rotation* mode, in which the z values are driven to zero, whereas the second is the *vectoring* mode, in which the y values are driven to zero. Note that δ_k requires nothing more than a comparison.

3. The numbers σ_k are stored constants which depend on the value of m. For $m = 1$, $\sigma_k = \tan^{-1} 2^{-k}$, for $m = 0$, $\sigma_k = 2^{-k}$, and for $m = -1$, $\sigma_k = \tanh^{-1} 2^{-k}$ (with some minor modifications to be discussed later).

To use these equations, appropriate starting values x_0, y_0, and z_0 must be given. One of these inputs, say z_0, might be the number whose sine we wish to approximate, $\sin z_0$. Or two of these inputs, say y_0 and x_0, might be the quotient we wish to approximate, y_0/x_0. In all cases, the starting values must be restricted to a certain interval about the origin in order to ensure convergence [5]. As we shall see in the following two examples, one of the variables tends to zero while another variable approaches the desired approximation.

Example: Division.

Let $m = 0$, $\delta_k = -\operatorname{sgn}(y_k)$, and $\sigma_k = 2^{-k}$. Then the following equations will approximate the quotient y_0/x_0 for $|y_0/x_0| \leq 2$. Notice that these equations consist of only additions, subtractions, comparisons and binary shifts.

$$\begin{aligned} x_{k+1} &= x_0 \\ y_{k+1} &= y_k + \delta_k x_0 2^{-k} \\ z_{k+1} &= z_k - \delta_k 2^{-k} \qquad\qquad (\sigma_k = 2^{-k}) \end{aligned}$$

The starting values are the given numbers x_0, y_0 and $z_0 = 0$. Here are the first five and last five iterations for the specific example $x_0 = 5$ and $y_0 = 7$ ($n = 50$ iterations):

	x	y	z
0	5.00000000000000	7.00000000000000	0
1	5.00000000000000	2.00000000000000	1.00000000000000
2	5.00000000000000	−0.50000000000000	1.50000000000000
3	5.00000000000000	0.75000000000000	1.25000000000000
4	5.00000000000000	0.12500000000000	1.37500000000000
5	5.00000000000000	−0.18750000000000	1.43750000000000
⋮			
46	5.00000000000000	−0.00000000000003	1.40000000000001
47	5.00000000000000	0.00000000000004	1.39999999999999
48	5.00000000000000	0.00000000000001	1.40000000000000
49	5.00000000000000	−0.00000000000001	1.40000000000000
50	5.00000000000000	−0.00000000000000	1.40000000000000

Notice how the y values have been driven towards zero while the z values approximate the quotient $y_0/x_0 = 1.4$.

It is not difficult to see why the sequence z_k will approach y_0/x_0. Note that the second and third equations imply that

$$y_{n+1} = y_0 + \sum_{k=0}^{n} \delta_k x_0 2^{-k} \qquad \text{and} \qquad z_{n+1} = -\sum_{k=0}^{n} \delta_k 2^{-k},$$

respectively. Hence,

$$\frac{y_0}{x_0} = \frac{y_{n+1} - \sum_{k=0}^{n} \delta_k x_0 2^{-k}}{x_0} = \frac{y_{n+1}}{x_0} - \sum_{k=0}^{n} \delta_k 2^{-k} = \frac{y_{n+1}}{x_0} + z_{n+1},$$

which gives

$$\left| z_{n+1} - \frac{y_0}{x_0} \right| = \left| \frac{y_{n+1}}{x_0} \right|.$$

Since y_{n+1} is close to zero, $z_{n+1} \approx y_0/x_0$.

Example: Sine and Cosine.

To approximate the sine and cosine of a number $z_0 = \theta$, $-\pi/2 \le \theta \le \pi/2$, we use $m = 1$, $\delta_k = \operatorname{sgn}(z_k)$, and $\sigma_k = \tan^{-1} 2^{-k}$.

$$\begin{aligned} x_{k+1} &= x_k - \delta_k y_k 2^{-k} \\ y_{k+1} &= y_k + \delta_k x_k 2^{-k} \\ z_{k+1} &= z_k - \delta_k \tan^{-1} 2^{-k} \qquad (\sigma_k = \tan^{-1} 2^{-k}) \end{aligned}$$

The starting values are $x_0 = K = \prod_{j=0}^{n} \cos(\sigma_j)$, $y_0 = 0$, and $z_0 = \theta$, the given angle. Because of the way the z's are constructed, z_k is forced towards zero. As we see in the following example, the x's will tend to $\cos\theta$ and the y's to $\sin\theta$. A precise proof of convergence depends on elementary trigonometric identities and can be found in [2].

The following MATLAB program does $n = 47$ iterations of the CORDIC algorithm to approximate the sine and cosine of the angle θ, $-\pi/2 \le \theta \le \pi/2$. We show the output for the first five and last five iterations for the calculation of the sine and cosine of $\theta = 1$. Notice how the z values are being driven to zero, while the x values approach $\cos 1$ and the y values approach $\sin 1$.

```
function cordtrig(t)
n=47;x=zeros(n,1);y=x;z=x;
K=cos(atan(1));
for j=1:n-1,
K=K*cos(atan(2^(-j)));
end
x(1)=K; y(1)=0; z(1)=t;
for j=1:n+1,
del=sign(z(j)); if del == 0 del =1; end;
x(j+1) = x(j) - del*y(j)*2^(-j+1);
y(j+1) = y(j) + del*x(j)*2^(-j+1);
z(j+1) = z(j) - del*atan(2^(-j+1));
end
answer = [x y z]
```

	x	y	z
0	0.60725293500888	0	1.00000000000000
1	0.60725293500888	0.60725293500888	0.21460183660255
2	0.30362646750444	0.91087940251332	−0.24904577239825
3	0.53134631813277	0.83497278563721	−0.00406710927139
4	0.63571791633742	0.76855449587062	0.12028788527537
5	0.58768326034551	0.80828686564170	0.05786907527941
⋮			
42	0.54030230586817	0.84147098480788	0.00000000000004
43	0.54030230586798	0.84147098480800	−0.00000000000019
44	0.54030230586808	0.84147098480794	−0.00000000000008
45	0.54030230586812	0.84147098480791	−0.00000000000002
46	0.54030230586815	0.84147098480789	0.00000000000001
47	0.54030230586814	0.84147098480790	−0.00000000000000

3. Concluding Remarks.

The above examples show some of the versatility of the CORDIC method. In the following table we give a more complete summary of the CORDIC options.

	$\delta_k = \text{sgn}(z_k) \qquad (z_k \to 0)$	$\delta_k = -\text{sgn}(y_k) \qquad (y_k \to 0)$
$m = 0$ $\sigma_k = 2^{-k}$	x_0, z_0 given, $y_0 = 0$ gives $y_n \approx x_0 z_0$	x_0, y_0 given, $z_0 = 0$ gives $z_n \approx y_0/x_0$
$m = 1$ $\sigma_k = \tan^{-1} 2^{-k}$	$x_0 = K, z_0 = \theta$, $y_0 = 0$ gives $x_n \approx \cos\theta, y_n \approx \sin\theta$	x_0, y_0 given, $z_0 = 0$ $z_n \approx \tan^{-1}(y_0/x_0)$
$m = -1$ $\sigma_k = \tanh^{-1} 2^{-k}$ (some σ_k repeated)	$x_0 = K', z_0 = \theta$, $y_0 = 0$ gives $x_n \approx \cosh\theta, y_n \approx \sinh\theta$ $e^\theta \approx x_n + y_n$	x_0, y_0 given, $z_0 = 0$ $z_n \approx \tanh^{-1}(y_0/x_0)$ $x_n \approx \sqrt{x_0^2 - y_0^2}/K'$

In this table, $K = \prod_{j=0}^{n} \cos(\sigma_j)$ and K' is a similar product using the hyperbolic cosine function. It should be noted that the key convergence theorem [2] for the CORDIC algorithm requires that the constants σ_k satisfy the inequality

$$\sigma_k \leq \sum_{j=k+1}^{n} \sigma_j + \sigma_n, \qquad \text{for } \; 0 \leq k \leq n.$$

It is an easy exercise to show that the constants $\sigma_k = 2^{-k}$ and $\sigma_k = \tan^{-1} 2^{-k}$ satisfy this property. However, for the hyperbolic functions, $m = -1$, the constants $\sigma_k = \tanh^{-1} 2^{-k}$ do not satisfy this property for all k. Hence, it is necessary to repeat certain σ_k values, the details of which can be found in [5].

To calculate $\ln w$, you can use the identity $\tanh^{-1} t = \frac{1}{2} \ln \frac{1+t}{1-t}$ and the CORDIC hyperbolic case, starting with $x_0 = w + 1, y_0 = w - 1$, to obtain

$$z_n \approx \tanh^{-1}(y_0/x_0) = \frac{1}{2} \ln\left(\frac{1 + y_0/x_0}{1 - y_0/x_0}\right) = \frac{1}{2} \ln\left(\frac{x_0 + y_0}{x_0 - y_0}\right) = \frac{1}{2} \ln w.$$

Thus, $\ln w \approx 2z_n$. Finally, it can be shown that using $x_0 = w + 0.25$ and $y_0 = w - 0.25$ you obtain $\sqrt{w} \approx K' x_n$.

Bibliography

1. Richard J. Pulskamp and James A. Delaney, **Computer and Calculator Computation of Elementary Functions**, UMAP Module 708, 1991.

2. Charles W. Schelin, **Calculator Function Approximation**, American Mathematical Monthly 90(5), 1983, pp 317-325.

3. **Transcendental Function Algorithms**, Post from Texas Instruments to Graph-TI mailing list, March 8, 1993.

4. Jack E. Volder, **The CORDIC Trigonometric Computing Technique**, IRE Transactions on Electronic Computers, volume EC-8, No. 3, September 1959, pp. 330-334.

5. J. S. Walther, **A Unified Algorithm for Elementary Functions**, Joint Computer Conference Proceedings, Spring 1971, pp 379-385.