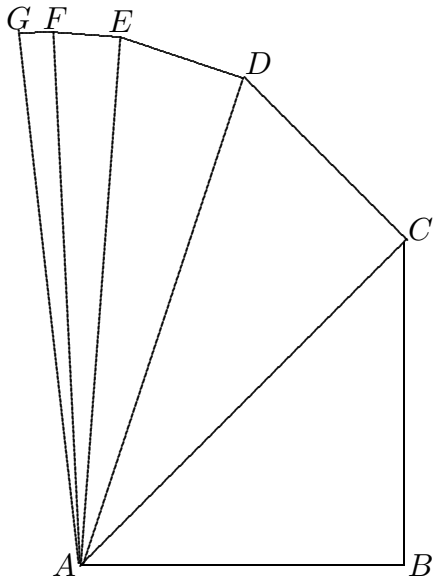
Elementary Functions and Calculators

Richard Parris, Phillips Exeter Academy

The mathematical content of these pages is not original. Despite being accessible in journal articles for several years, however, it seems not to be well known. Teachers of trigonometry and calculus thus may be surprised to learn what really happens when they ask their calculators to find square roots, cosines, and logarithms. It is hoped that this paper will help dispel a myth — that the calculus of infinite power series is at work — and that it will instead rekindle interest in the mathematics that lies at the heart of the versatile CORDIC algorithm — namely, trigonometric addition formulas and hyperbolic functions. It will also be seen that improvements are needed. Refining this algorithm is a topic of current research.

A Trigonometric Fan

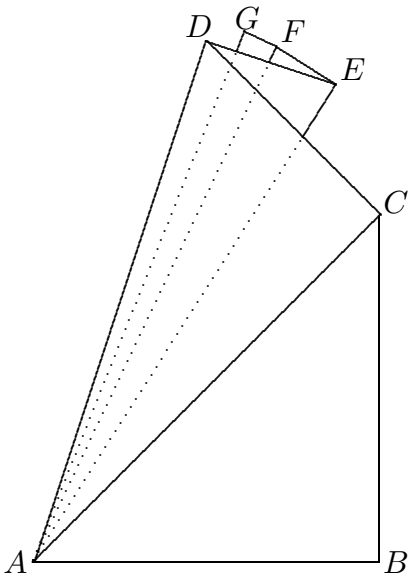
Imagine a fan built from a sequence of right triangles. The longest leg of each triangular section is hinged to the hypotenuse of the preceding section. The angles that meet at the vertex of the fan are all acute, and decrease steadily toward zero. The sizes of the first five angles are approximately 45.00°, 26.57°, 14.04°, 7.13°, and 3.58°. The diagram shows these sections. Notice that each angle is about half as large as its predecessor. This is a consequence of the precise construction of these angles. What is actually being halved at each stage is the *tangent ratio*: $BC = BA$, $CD = \frac{1}{2}CA$, $DE = \frac{1}{4}DA$, $EF = \frac{1}{8}EA$, $FG = \frac{1}{16}FA$, and so on. The resulting infinite sequence of angles is geometric in a limiting sense.



The examples that follow employ only the first two dozen terms of this sequence. In principle every term is useful, however, for the accuracy of CORDIC is set by the size of the last angle in the fan. The scope of the algorithm is, as you might have guessed, limited to those angles covered by the fan, which include all the acute ones. The first four angles alone bring the total to more than 90 degrees. No matter how many of these sections are included, the total never quite reaches 100 degrees, by the way.

To apply this fan to the computation of trigonometric values, the initial segment $\overline{AB}$ is placed on the positive x-axis, with A at the origin. The terminal segment (which would be $\overline{AG}$ if the fan consisted of only five sections, as in the diagram) points into the first or second quadrant, depending on how the hinges are folded. To say that the scope of the fan includes the first quadrant means that the terminal segment can be made to point in any first-quadrant direction, to within a small tolerance. This is accomplished by folding the fan along

its hinges. In the diagram you see the result of folding the third section over the second (along hinge $\overline{AD}$), and then the fourth and fifth sections over the third (along hinge $\overline{AE}$). If the goal had been to create a 70° angle with the fan, these would have been the first five stages. Because the first two angles have a sum of more than 70°, the third angle is subtracted by folding it back over the second. This makes the subtotal 57.53°, so the fourth and fifth angles must be added to bring the subtotal to 68.23°.



It is plausible that this process can be continued until the terminal segment makes an angle with $\overline{AB}$ that is as close to 70° as desired. In fact, it can be shown that the terminal angle differs from the target angle by no more than the last angle in the fan (as long as the target angle lies within the scope of the fan). This is a property of the special angles used to build the fan.

From now on, all examples are based on a 24-section fan. This guarantees that each angle in the first quadrant will be approximated by the folding process to within 0.00000683° (the size of the 24th angle, whose tangent is 2^{-23}). The table below shows step-by-step how the special angles are combined so that the terminal segment of the fan is always folded toward 70°.

Angle	Subtotal	Angle	Subtotal
45.00000000(+)	45.00000000	0.01398823(+)	70.00614325
26.56505118(+)	71.56505118	0.00699411(−)	69.99914914
14.03624347(−)	57.52880771	0.00349706(+)	70.00264619
7.12501635(+)	64.65382406	0.00174853(−)	70.00089767
3.57633437(+)	68.23015843	0.00087426(−)	70.00002340
1.78991061(+)	70.02006904	0.00043713(−)	69.99958627
0.89517371(−)	69.12489533	0.00021857(+)	69.99980484
0.44761417(+)	69.57250950	0.00010928(+)	69.99991412
0.22381050(+)	69.79632000	0.00005464(+)	69.99996876
0.11190568(+)	69.90822568	0.00002732(+)	69.99999608
0.05595289(+)	69.96417857	0.00001366(+)	70.00000974
0.02797645(+)	69.99215502	0.00000683(−)	70.00000291

Next to each of the 24 special angles is shown an indication (+ or −) of how the angle contributes to the folding process, as well as the subtotal that results. Notice that every subtotal differs from 70° by less than the size of its special angle, by the way. The covering property of the fan can be described another way: There are 23 hinges, each of which can be folded in one of two ways. This means that there are $2^{23} = 8388608$ ways of folding the fan, hence 8388608 positions for the terminal segment. These 8388608 terminal segments

are scattered throughout the first quadrant and part of the second, uniformly enough that any radial segment makes an angle less than 0.00000683° with one of them.

A key property of the special angles is that each one of them is exceeded by the sum of those that follow it in the list. This means that every section of the fan can be covered by the (smaller) sections that follow it (except possibly for a gap that is no larger than the smallest angle).

A familiar property of powers of 2

Take any group of consecutive terms from the infinite series

$$\cdots + 16 + 8 + 4 + 2 + 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

and calculate its sum. The result equals the difference between the last term and the term that precedes the first term. For instance, $8 + 4 + 2 + 1 + \frac{1}{2} + \frac{1}{4} = 16 - \frac{1}{4}$. In other words, any term in the series is equal to the sum of any block of terms that follows it, the last term in the block counting twice. (If infinite series are allowed, then any term equals the sum of *all* the terms that follow it.)

With the help of the Mean-Value Theorem from Calculus, this familiar result can be used to show that the special angles (whose tangents are powers of 2) used to build the fan have the necessary covering and accuracy properties. From now on, let A_n denote the acute angle whose tangent is 2^{-n}.

> For any nonnegative integers m and n, with $m < n$, A_m is at most equal to $A_{m+1} + A_{m+2} + \cdots + A_n + A_n$.
>
> Let T be a target value, with $0 < T < A_0 + A_1 + \cdots + A_n$. Define a sequence as follows: $T_0 = 0$ and, for $0 \le m$, let $T_{m+1} = T_m + A_m$ if $T_m < T$ and $T_{m+1} = T_m - A_m$ if $T \le T_m$. Then $|T_{n+1} - T| \le A_n$.

See [2] for proofs of these results, which formalize the angle-approximation scheme described above. Each T_m is the subtotal after special angle A_{m-1} has been included.

Trigonometric calculations

The time has come to actually calculate something. Let us find the sine and cosine of a 70-degree angle. The primary computational tools are the well known trigonometric addition formulas

$$\cos(u \pm v) = \cos(u)\cos(v) \mp \sin(u)\sin(v)$$
$$\sin(u \pm v) = \sin(u)\cos(v) \pm \cos(u)\sin(v)$$

Because 70° is approximated by $T_{24} = A_0+A_1-A_2+A_3+A_4+A_5-A_6+\cdots+A_{21}+A_{22}-A_{23}$, it is inviting to build up the desired answers in stages. First find

$$\begin{aligned}
\cos(A_0) &= 0.70710678\\
\sin(A_0) &= 0.70710678\\
\cos(A_1) &= 0.89442719\\
\sin(A_1) &= 0.44721360\\
&\vdots\\
\cos(A_{23}) &= 1.00000000\\
\sin(A_{23}) &= 0.00000012.
\end{aligned}$$

Incidentally, these values can be found by Pythagorean methods (the tangents of these angles are known, remember), and it is only necessary to calculate them once, for the same angles will be applied to every target angle.

Next use the addition formulas to calculate

$$\begin{aligned}
\cos(T_1) &= \cos(T_0 + A_0) = \cos(A_0) = 0.70710678\\
\sin(T_1) &= \sin(T_0 + A_0) = \sin(A_0) = 0.70710678,
\end{aligned}$$

$$\begin{aligned}
\cos(T_2) &= \cos(T_1 + A_1) = \cos(T_1)\cos(A_1) - \sin(T_1)\sin(A_1) = 0.31622777\\
\sin(T_2) &= \sin(T_1 + A_1) = \sin(T_1)\cos(A_1) + \cos(T_1)\sin(A_1) = 0.94868330,
\end{aligned}$$

then use the subtraction formulas to calculate

$$\begin{aligned}
\cos(T_3) &= \cos(T_2 - A_2) = \cos(T_2)\cos(A_2) + \sin(T_2)\sin(A_2) = 0.53687549\\
\sin(T_3) &= \sin(T_2 - A_2) = \sin(T_2)\cos(A_2) - \cos(T_2)\sin(A_2) = 0.84366149,
\end{aligned}$$

and so on. This naive, recursive approach will yield the desired sine and cosine values, *but at great cost.* Every round requires four multiplications and two additions, so the whole task requires 92 multiplications and 46 additions. This is time-consuming, even for a state-of-the-art electronic device. There is a better way — one that requires *no* multiplications!

No multiplications?

The clever trick is to rewrite the addition formulas as follows:

$$\begin{aligned}
\cos(u \pm v) &= \cos(v)\,[\cos(u) \mp \tan(v)\sin(u)]\\
\sin(u \pm v) &= \cos(v)\,[\sin(u) \pm \tan(v)\cos(u)]
\end{aligned}$$

The recursive calculations now take the form

$$\cos(T_1) = \cos(A_0),$$
$$\sin(T_1) = \cos(A_0),$$

$$\cos(T_2) = \cos(A_1)\left[\cos(T_1) - \frac{1}{2}\sin(T_1)\right] = \cos(A_1)\cos(A_0)\left[\frac{1}{2}\right],$$
$$\sin(T_2) = \cos(A_1)\left[\sin(T_1) + \frac{1}{2}\cos(T_1)\right] = \cos(A_1)\cos(A_0)\left[\frac{3}{2}\right],$$

$$\cos(T_3) = \cos(A_2)\left[\cos(T_2) + \frac{1}{4}\sin(T_2)\right] = \cos(A_2)\cos(A_1)\cos(A_0)\left[\frac{7}{8}\right],$$
$$\sin(T_3) = \cos(A_2)\left[\sin(T_2) - \frac{1}{4}\cos(T_2)\right] = \cos(A_2)\cos(A_1)\cos(A_0)\left[\frac{11}{8}\right],$$

and so on. The powers of $\frac{1}{2}$ are the tangent values of the special angles. Because the cosines have been factored out, the powers of $\frac{1}{2}$ occur where the sine values once were. It is easy to look ahead to the desired answers,

$$\cos(T_{24}) = \cos(A_{23})\cos(A_{22})\cdots\cos(A_1)\cos(A_0)[C_{24}],$$
$$\sin(T_{24}) = \cos(A_{23})\cos(A_{22})\cdots\cos(A_1)\cos(A_0)[S_{24}],$$

where C_{24} and S_{24} are *dyadic rationals*, calculated recursively as follows:

$$C_0 = 1 \quad \text{and} \quad S_0 = 0$$
$$C_{m+1} = C_m - 2^{-m} D_m S_m$$
$$S_{m+1} = S_m + 2^{-m} D_m C_m$$

Each D_m is either 1 or -1, depending on whether special angle A_m appears with a positive or negative sign in the approximation of the target angle. The variable names C_m and S_m are meant to suggest cosine and sine, respectively, even though their values are *not* $\cos(T_m)$ and $\sin(T_m)$.

It should now be clear why the tangent function was chosen to set up the fan. It is perhaps also clear that choosing to work with powers of two is about to pay a dividend.

Multiplying and dividing by 2 is easy for computers

Because numbers are usually represented in base-10 notation, it is an easy matter to divide a number like 0.70710678 by 10 — just shift the point one space to the left. The same principle applies to numbers expressed in base-2 notation, as is usually internally the case in electronic calculators. It is thus easy to divide $\cos(A_0) = 0.10110101000001001111001100 1\ldots$ by two=10 (but not by ten=1010).

In other words, the recursion above involves only addition, subtraction, and point-shifting — *no multiplication or division* — when done in binary scale.

This leaves only the final product $\cos(A_{23})\cos(A_{22})\cdots\cos(A_1)\cos(A_0)$ to worry about. It is eliminated next.

The fan revisited

One obvious and important feature of the fan is that the lengths of its hinges do not change, no matter how it is folded. These lengths are related in a simple way: First $AC = \sec(A_0)AB$, then $AD = \sec(A_1)AC = \sec(A_1)\sec(A_0)AB$, and so on — each one being larger than its predecessor. In particular, the length of the terminal segment of the 24-section fan is $\sec(A_{23})\sec(A_{22})\cdots\sec(A_1)\sec(A_0)AB$. This is useful information, because looking for cosine and sine values is equivalent to looking for x- and y-coordinates of points on the unit circle. In order that the terminal segment of the fan be of unit length, it is necessary that $AB = \cos(A_{23})\cos(A_{22})\cdots\cos(A_1)\cos(A_0)$. From now on, this *constant*, which is about 0.607252935, will be denoted by P. Whenever the 24-section fan is used to approximate angles, the value of P is essential, for it is used to seed the recursion. This is how multiplication is avoided entirely.

In other words, if the recursion is redefined

$$\begin{aligned} C_0 &= P \quad \text{and} \quad S_0 = 0 \\ C_{m+1} &= C_m - 2^{-m} D_m S_m \\ S_{m+1} &= S_m + 2^{-m} D_m C_m, \end{aligned}$$

then C_{24} and S_{24} will be $\cos(T_{24})$ and $\sin(T_{24})$, respectively. Making the length of fan segment $\overline{AB}$ exactly P thus eliminates all the multiplications from the process.

It remains only to define the numbers D_m carefully. These coefficients are determined by the target angle. For the final version of the algorithm, it is convenient to modify the angle-approximation recursion slightly. Instead of starting with $T_0 = 0$ and building toward the target value T, it is customary to start the process with the target value and reduce this value to zero. In other words, instead of checking to see whether T_m is less than T, we check to see whether $Z_m = T - T_m$ is greater than zero. This does not affect the calculations; it simply means that the process is guided by the sign of Z_m. Here is the final version of the algorithm:

$$\begin{aligned} Z_0 &= T \ , \ C_0 = P \ , \ S_0 = 0 \\ Z_{m+1} &= Z_m - D_m A_m \\ C_{m+1} &= C_m - 2^{-m} D_m S_m \\ S_{m+1} &= S_m + 2^{-m} D_m C_m, \end{aligned}$$

where $D_m = \pm 1$ is chosen to have the same sign as Z_m. We say that Z_m is *driven toward zero*. In the process, C_m and S_m are driven toward $\cos(T)$ and $\sin(T)$, respectively.

This is known as the CORDIC algorithm, which stands for *COordinate Rotation DIgital Computer*. The method was published in 1959 by J. Volder. See [2] for more details of its historical and mathematical evolution. It is still used today, because of its speed and efficiency. To calculate sine and cosine values for any of the 8388608 angles that can be exactly represented by the 24-section fan, it is only necessary to have twenty-five permanently stored constants (A_0, A_1, ..., A_{23}, and P), which are combined using only addition, subtraction, and point-shifting. These 8388608 sine and cosine values are used in place of the correct values for all other (nearby) angles.

The table below shows all the details of the (simultaneous) calculation of cos(70°) and sin(70°). Each row of the table shows the values of Z, C, S, and D at the end of a computation cycle. Each row of the table is produced by looking back at the previous row. Once the sign of D is set (by the sign of the old value of Z), all else follows. Except for A, each row of the table replaces the previous row in the calculator memory.

m	Z_m	C_m	S_m	D_m	A_m
0	70.00000000	0.60725294	0.00000000	1	45.00000000
1	25.00000000	0.60725294	0.60725294	1	26.56505118
2	−1.56505118	0.30362647	0.91087940	−1	14.03624347
3	12.47119229	0.53134632	0.83497279	1	7.12501635
4	5.34617594	0.42697472	0.90139108	1	3.57633437
5	1.76984157	0.37063778	0.92807700	1	1.78991061
6	−0.02006904	0.34163537	0.93965943	−1	0.89517371
7	0.87510467	0.35631755	0.93432137	1	0.44761417
8	0.42749050	0.34901816	0.93710510	1	0.22381050
9	0.20368000	0.34535760	0.93846846	1	0.11190568
10	0.09177432	0.34352465	0.93914298	1	0.05595289
11	0.03582143	0.34260752	0.93947846	1	0.02797645
12	0.00784498	0.34214879	0.93964575	1	0.01398823
13	−0.00614325	0.34191938	0.93972928	−1	0.00699411
14	0.00085086	0.34203410	0.93968754	1	0.00349706
15	−0.00264619	0.34197674	0.93970842	−1	0.00174853
16	−0.00089767	0.34200542	0.93969798	−1	0.00087426
17	−0.00002340	0.34201976	0.93969276	−1	0.00043713
18	0.00041373	0.34202693	0.93969015	1	0.00021857
19	0.00019516	0.34202334	0.93969146	1	0.00010928
20	0.00008588	0.34202155	0.93969211	1	0.00005464
21	0.00003124	0.34202066	0.93969243	1	0.00002732
22	0.00000392	0.34202021	0.93969260	1	0.00001366
23	−0.00000974	0.34201998	0.93969268	−1	0.00000683
24	−0.00000291	0.34202010	0.93969264		

It is sometimes suggested that the computational process could be improved by taking the sizes of A_m and Z_m into account. For example, in the 70-degree computation, it would seem advantageous to leave out special angles A_2, A_3, and A_4, for they are all much larger than the remainder Z_2. Even if one disregards the logical complications that such decision-making would introduce into the algorithm, the answer is clear: This would mean removing sections from the fan, thereby making it impossible to predict the length of the terminal segment in advance. It is an essential feature of the CORDIC method of building with special angles that *every special angle is used in every application.* All intermediate stages of the simultaneous computation of $\cos(70°)$ and $\sin(70°)$ are shown in the preceding table (in base-10 notation, of course). The 24 cycles of this algorithm are sufficient to guarantee better than 6-place accuracy. This is because the angle-approximation process creates an error of at most 0.00000683°, and because the sine and cosine functions themselves shrink this error further (to at most 0.00000012, as can be proved with calculus).

CORDIC has a reverse gear, too

Now that you have thoroughly examined the sine and cosine buttons on your calculator, turn your attention to some of the others. Because tangents are just ratios of sines to cosines, it is clear that the tangent button needs no further attention. The time has come to see what happens when CORDIC is run backwards. In other words, it is time to take up the inverse trigonometric functions.

As it turns out, the simplest one to understand is the inverse tangent. You might think that this is because the contruction of the fan features tangent ratios; actually, it is because tangent ratios provide the simplest way to describe an angle. As an example, consider the problem of finding the acute angle whose tangent is 2. This will illustrate the angle-finding capability of CORDIC.

You are now in the position of knowing the terminal ray of the fan, but of not knowing the combination of special angles that produced it. The obvious solution is to apply the angle-approximation algorithm again and keep an eye on the target angle. The difficulty is deciding how to deal with variable Z; how to seed it or how to know when it is close to its target value. The trick is to turn the fan over and place the initial $\overline{AB}$ on the terminal line. This in effect provides initial values for S and C. The folding process is now directed toward the positive x-axis. Variable Z will approach the desired angle, provided that it is initialized to be zero.

Which variable — C or S — should control the process? In other words, should C be driven toward 1 or S be driven toward 0? The correct choice is to drive S toward 0. The reason is that it is not necessary to worry about the size of the fan for this problem. The seed values for C and S should be of no concern individually — they need only be chosen so that $S/C = 2$. Thus, because C and S need not represent actual cosine and sine values, it would be incorrect to drive C toward 1.

All the intermediate stages of the computation of arctan(2) are shown below. Notice that the initial values $C_0 = 0.25$ and $S_0 = 0.5$ define the angle.

m	Z_m	C_m	S_m	D_m	A_m
0	0.00000000	0.25000000	0.50000000	−1	45.00000000
1	45.00000000	0.75000000	0.25000000	−1	26.56505118
2	71.56505118	0.87500000	−0.12500000	1	14.03624347
3	57.52880771	0.90625000	0.09375000	−1	7.12501635
4	64.65382406	0.91796875	−0.01953125	1	3.57633437
5	61.07748968	0.91918945	0.03784180	−1	1.78991061
6	62.86740029	0.92037201	0.00911713	−1	0.89517371
7	63.76257400	0.92051446	−0.00526369	1	0.44761417
8	63.31495983	0.92055559	0.00192783	−1	0.22381050
9	63.53877033	0.92056312	−0.00166809	1	0.11190568
10	63.42686465	0.92056638	0.00012989	−1	0.05595289
11	63.48281755	0.92056650	−0.00076910	1	0.02797645
12	63.45484109	0.92056688	−0.00031961	1	0.01398823
13	63.44085287	0.92056696	−0.00009486	1	0.00699411
14	63.43385875	0.92056697	0.00001751	−1	0.00349706
15	63.43735581	0.92056697	−0.00003867	1	0.00174853
16	63.43560728	0.92056697	−0.00001058	1	0.00087426
17	63.43473302	0.92056697	0.00000347	−1	0.00043713
18	63.43517015	0.92056697	−0.00000356	1	0.00021857
19	63.43495158	0.92056697	−0.00000004	1	0.00010928
20	63.43484230	0.92056697	0.00000171	−1	0.00005464
21	63.43489694	0.92056697	0.00000083	−1	0.00002732
22	63.43492426	0.92056697	0.00000039	−1	0.00001366
23	63.43493792	0.92056697	0.00000018	−1	0.00000683
24	63.43494475	0.92056697	0.00000007		

The entries in the S column are approaching 0; the entries in the Z-column are approaching arctan(2). It is left as an exercise for you to identify the number that is being approached by the entries in the C column. (The answer is useful and will appear in a few pages.) The recursive CORDIC approach to calculating arctan(T) is defined as follows:

$$Z_0 = 0 \ , \ S_0 = C_0 T$$
$$Z_{m+1} = Z_m - D_m A_m$$
$$C_{m+1} = C_m - 2^{-m} D_m S_m$$
$$S_{m+1} = S_m + 2^{-m} D_m C_m$$

where $D_m = \pm 1$ is chosen so that $D_m C_m$ and S_m have opposite signs. This drives S_m toward 0. In the process, Z_m is driven towards arctan(T). This algorithm can be applied to any angle within the scope of the fan, which includes all acute angles and some obtuse ones.

As for inverse sines and cosines, they can be dealt with similarly. In fact, any such problem can be converted into an inverse tangent problem by means of a preliminary Pythagorean computation. For instance, finding arcsin(0.6) is equivalent to finding arctan(0.75), and finding arccos(u) is equivalent to finding $\arctan\left(\dfrac{\sqrt{1-u^2}}{u}\right)$.

If this were all that CORDIC could perform for us, it would be a remarkable invention. There is still much more, however.

Exponents and logarithms

The utility and elegance of mathematics resides in its ability to exploit formal connections between apparently distinct problems. A case in point is the system of hyperbolic functions, namely sinh, cosh, tanh, csch, sech, and coth. Recall that

$$\cosh(u) = \frac{1}{2}\left(e^u + e^{-u}\right) \quad \text{and} \quad \sinh(u) = \frac{1}{2}\left(e^u - e^{-u}\right),$$

and that tanh(u) = sinh(u)/cosh(u), sech(u) = 1/cosh(u), and so on. Despite the appearance of these functions, their very names suggest that there are important similarities with the circular functions. For CORDIC, the most useful identities are the following:

$$\begin{gathered}[\cosh(u)]^2 - [\sinh(u)]^2 = 1 \\ \cosh(u \pm v) = \cosh(u)\cosh(v) \pm \sinh(u)\sinh(v) \\ \sinh(u \pm v) = \sinh(u)\cosh(v) \pm \cosh(u)\sinh(v)\end{gathered}$$

The calculator does not actually know of any geometric significance to the numbers A_m and P that it combines to evaluate circular functions. All it has to work with are the formal identities, applied in a recursive way. There is every reason to hope, therefore, that the CORDIC approach will be able to evaluate hyperbolic functions as easily as it evaluates circular functions. Define B_m provisionally by the equation $\tanh(B_m) = 2^{-m}$ for $m = 1, 2, \ldots, n$, let Q be the product $\cosh(B_1)\cosh(B_2)\cdots\cosh(B_n)$, and consider the equations

$$\begin{aligned} Z_1 &= T \ , \ C_1 = Q \ , \ S_1 = 0 \\ Z_{m+1} &= Z_m - D_m B_m \\ C_{m+1} &= C_m + 2^{-m-1} D_m S_m \\ S_{m+1} &= S_m + 2^{-m-1} D_m C_m \end{aligned}$$

where $D_m = \pm 1$ is always chosen to have the same sign as Z_m. It is necessary to look at the fine details of this proposal to see whether there are subtle flaws. You have probably already noted slight departures from the circular model. The sign change in the addition formula for $\cosh(u \pm v)$ has been incorporated into the recursion; it is not expected to cause any difficulty. On the other hand, because the equation $\tanh(u) = 1$ has no solutions, the list of special *arguments* B_m begins with the solution to $\tanh(B_1) = 1/2$. The indices in the recursion have been adjusted accordingly.

Approximating hyperbolic arguments is not routine

The table below shows the results of *trying* to approximate the value 0.549 by combining the special arguments B_1, B_2, ..., B_{24}:

Arg	Subtotal	Arg	Subtotal
0.54930614(+)	0.54930614	0.00012207(+)	0.54452168
0.25541281(−)	0.29389333	0.00006104(+)	0.54458271
0.12565721(+)	0.41955055	0.00003052(+)	0.54461323
0.06258157(+)	0.48213212	0.00001526(+)	0.54462849
0.03126018(+)	0.51339230	0.00000763(+)	0.54463612
0.01562627(+)	0.52901857	0.00000381(+)	0.54463994
0.00781266(+)	0.53683123	0.00000191(+)	0.54464184
0.00390627(+)	0.54073750	0.00000095(+)	0.54464280
0.00195313(+)	0.54269062	0.00000048(+)	0.54464327
0.00097656(+)	0.54366719	0.00000024(+)	0.54464351
0.00048828(+)	0.54415547	0.00000012(+)	0.54464363
0.00024414(+)	0.54439961	0.00000006(+)	0.54464369

The problem is that B_2 is greater (by more than 0.00466) than the sum of B_3, B_4, ..., B_{24}. This is a situation where it seems desirable to skip one of the special terms, for subtracting B_2 has left the subtotal much farther from the target than it was before B_1 was subtracted — so far, in fact, that the difference can not be closed with later terms.

The inadequacy of this sequence of special arguments casts some doubt on the ability of CORDIC to evaluate exponential functions. Because the special arguments B_m lack the essential covering property discussed earlier, the algorithm can not approximate arguments such as $T = 0.549$ with sufficient accuracy to permit computation of usable values for $\cosh(T)$ and $\sinh(T)$.

The remedy is simple, however — make *duplicate entries* in the list of special arguments. In other words, allow the algorithm to *reuse* certain arguments, in order that it be able to close all gaps such as in the example. This selection of duplicates must be made once and for all, so that the constant Q can be calculated and stored along with the arguments B_m. One solution is presented on the next page, together with a demonstration of its correctness. The proof consists of verifying that every argument in the list is at most equal to the sum of all the arguments that follow it in the list, except for an allowable gap equal to the last argument in the list.

The illustration below shows the eighteen special arguments $\tanh^{-1}(2^{-m})$ for $m = 1, 2, \ldots, 18$. Next to each is tabled the sum of all the special arguments that follow it in the list. Notice that every entry exceeds the corresponding sum by more than the allowed 0.00000381.

Argum	Sum	Argum	Sum
0.54930614	0.50615941	0.00097656	0.00097275
0.25541281	0.25074660	0.00048828	0.00048447
0.12565721	0.12508939	0.00024414	0.00024033
0.06258157	0.06250782	0.00012207	0.00011826
0.03126018	0.03124764	0.00006104	0.00005722
0.01562627	0.01562137	0.00003052	0.00002670
0.00781266	0.00780871	0.00001526	0.00001144
0.00390627	0.00390244	0.00000763	0.00000381
0.00195313	0.00194931	0.00000381	

The next table shows the effect of duplicating six chosen entries (which are marked). Now every special argument is exceeded by the sum of the arguments that follow it in the list. This verifies the covering property that is necessary for the CORDIC algorithm to approximate arguments to a predictable degree of accuracy.

Argum	Sum	Argum	Sum
0.54930614	0.57712204	0.00048828	0.00105286
0.25541281	0.32170922	0.00048828*	0.00056458
0.12565721	0.19605201	0.00024414	0.00032043
0.06258157	0.13347044	0.00012207	0.00019836
0.06258157*	0.07088887	0.00006104	0.00013733
0.03126018	0.03962869	0.00006104*	0.00007629
0.01562627	0.02400242	0.00003052	0.00004578
0.00781266	0.01618976	0.00001526	0.00003052
0.00781266*	0.00837710	0.00001526*	0.00001526
0.00390627	0.00447083	0.00000763	0.00000763
0.00195313	0.00251770	0.00000381	0.00000381
0.00097656	0.00154114	0.00000381*	

Every argument that lies within the scope of the table (from 0.0 to about 1.12642818) can be approximated to within 0.00000381 (the last entry in the list). The next page shows a new, successful attempt on the example $T = 0.549$.

How to accurately represent $T = 0.549$ as a combination of special arguments B_m:

Arg	Subtotal	Arg	Subtotal
0.54930614(+)	0.54930614	0.00048828(−)	0.54911683
0.25541281(−)	0.29389333	0.00048828(−)	0.54862855
0.12565721(+)	0.41955055	0.00024414(+)	0.54887269
0.06258157(+)	0.48213212	0.00012207(+)	0.54899476
0.06258157(+)	0.54471369	0.00006104(+)	0.54905580
0.03126018(+)	0.57597387	0.00006104(−)	0.54899476
0.01562627(−)	0.56034760	0.00003052(+)	0.54902528
0.00781266(−)	0.55253494	0.00001526(−)	0.54901002
0.00781266(−)	0.54472228	0.00001526(−)	0.54899476
0.00390627(+)	0.54862855	0.00000763(+)	0.54900239
0.00195313(+)	0.55058168	0.00000381(−)	0.54899858
0.00097656(−)	0.54960511	0.00000381(+)	0.54900239

Now the provisional definition of the list of special arguments can be put in a reliable form. The 24 special values will be denoted B_0, B_1, ..., B_{23} (in the interest of uniformity), even though no simple formula for $\tanh(B_m)$ is available (because of the duplications). All that needs to be stored is a list of binary-point shifts, however; see the example below. The CORDIC algorithm for the hyperbolic functions cosh and sinh is

$$\begin{aligned} &Z_0 = T \ , \ C_0 = Q \ , \ S_0 = 0 \\ &Z_{m+1} = Z_m - D_m B_m \\ &C_{m+1} = C_m + E_m D_m S_m \\ &S_{m+1} = S_m + E_m D_m C_m, \end{aligned}$$

where $D_m = \pm 1$ is always chosen to have the same sign as Z_m (thus driving Z_m toward 0), Q is the constant $\cosh(B_0)\cosh(B_1)\cdots\cosh(B_{23}) = 1.20753406...$, and $E_m = \tanh(B_m)$ is an integral power of 1/2 (represented by the shift entry listed below).

Tabled on the next page is a table of all the intermediate stages of a 24-cycle computation of cosh(1.0) and sinh(1.0), using this algorithm.

All the intermediate stages of a 24-cycle CORDIC computation of cosh(1.0) and sinh(1.0) are shown below.

m	Z_m	C_m	S_m	D_m	$Shift$	B_m
0	1.00000000	1.20753406	0.00000000	1	1	0.54930614
1	0.45069386	1.20753406	0.60376703	1	2	0.25541281
2	0.19528104	1.35847581	0.90565054	1	3	0.12565721
3	0.06962383	1.47168213	1.07546002	1	4	0.06258157
4	0.00704226	1.53889838	1.16744015	1	4	0.06258157
5	-0.05553931	1.61186339	1.26362130	−1	5	0.03126018
6	-0.02427913	1.57237523	1.21325057	−1	6	0.01562627
7	-0.00865286	1.55341819	1.18868221	−1	7	0.00781266
8	-0.00084020	1.54413161	1.17654613	−1	7	0.00781266
9	0.00697245	1.53493984	1.16448260	1	8	0.00390627
10	0.00306618	1.53948860	1.17047846	1	9	0.00195313
11	0.00111306	1.54177469	1.17348527	1	10	0.00097656
12	0.00013649	1.54292067	1.17499091	1	11	0.00048828
13	-0.00035179	1.54349440	1.17574429	−1	11	0.00048828
14	0.00013649	1.54292030	1.17499063	1	12	0.00024414
15	-0.00010765	1.54320717	1.17536732	−1	13	0.00012207
16	0.00001442	1.54306369	1.17517894	1	14	0.00006104
17	-0.00004661	1.54313542	1.17527312	−1	14	0.00006104
18	0.00001442	1.54306368	1.17517894	1	15	0.00003052
19	-0.00001609	1.54309955	1.17522603	−1	16	0.00001526
20	-0.00000083	1.54308162	1.17520248	−1	16	0.00001526
21	0.00001442	1.54306368	1.17517894	1	17	0.00000763
22	0.00000679	1.54307265	1.17519071	1	18	0.00000381
23	0.00000298	1.54307713	1.17519659	1	18	0.00000381
24	-0.00000083	1.54308162	1.17520248			

The values C_{24} and S_{24} are good approximations to cosh(1.0) and sinh(1.0), respectively. Their sum is 2.71828410, which has a familiar look to it:

CORDIC evaluates e^x by calculating cosh(x)+sinh(x).

The approximation of arguments within the scope of this algorithm is accurate to within 0.00000381. The cosh and sinh functions can magnify any error by as much as 1.4 or 1.7, respectively, so the results are accurate only to within 0.00000534 or 0.00000648. This in turn makes the calculation of e^x reliable only to within 0.00001182 (although the value for e found above was actually within 0.0000023 of being correct).

Remember that the above calculations, when carried out in base 2 arithmetic, involve addition, subtraction, and point-shifting only — there is no multiplication or division.

Back into reverse gear

Now that you have seen how CORDIC produces exponential values, it is time to explore the very interesting inverse process. As you would expect, the function of central importance is the inverse hyperbolic tangent. The illustration below shows the computation of $\tanh^{-1}(1/3)$, which is initialized by setting $Z_0 = 0$, $C_0 = 1$, and $S_0 = 3$. As S_m is driven toward 0, Z_m is driven toward the answer.

m	Z_m	C_m	S_m	D_m	$Shift$	B_m
0	0.00000000	3.00000000	1.00000000	−1	1	0.54930614
1	0.54930614	2.50000000	−0.50000000	1	2	0.25541281
2	0.29389333	2.37500000	0.12500000	−1	3	0.12565721
3	0.41955055	2.35937500	−0.17187500	1	4	0.06258157
4	0.35696898	2.34863281	−0.02441406	1	4	0.06258157
5	0.29438740	2.34710693	0.12237549	−1	5	0.03126018
6	0.32564758	2.34328270	0.04902840	−1	6	0.01562627
7	0.34127385	2.34251663	0.01241460	−1	7	0.00781266
8	0.34908651	2.34241964	−0.00588631	1	7	0.00781266
9	0.34127385	2.34237366	0.01241385	−1	8	0.00390627
10	0.34518012	2.34232516	0.00326395	−1	9	0.00195313
11	0.34713325	2.34231879	−0.00131090	1	10	0.00097656
12	0.34615669	2.34231751	0.00097652	−1	11	0.00048828
13	0.34664497	2.34231703	−0.00016719	1	11	0.00048828
14	0.34615669	2.34231695	0.00097652	−1	12	0.00024414
15	0.34640083	2.34231671	0.00040466	−1	13	0.00012207
16	0.34652290	2.34231666	0.00011873	−1	14	0.00006104
17	0.34658393	2.34231665	−0.00002423	1	14	0.00006104
18	0.34652290	2.34231665	0.00011873	−1	15	0.00003052
19	0.34655342	2.34231665	0.00004725	−1	16	0.00001526
20	0.34656868	2.34231665	0.00001151	−1	16	0.00001526
21	0.34658393	2.34231665	−0.00002423	1	17	0.00000763
22	0.34657631	2.34231665	−0.00000636	1	18	0.00000381
23	0.34657249	2.34231665	0.00000258	−1	18	0.00000381
24	0.34657631	2.34231665	−0.00000636			

The recursive CORDIC approach to calculating $\tanh^{-1}(T)$ is

$$
\begin{aligned}
&Z_0 = 0 \ , \ S_0 = C_0 T \\
&Z_{m+1} = Z_m - D_m B_m \\
&C_{m+1} = C_m + E_m D_m S_m \\
&S_{m+1} = S_m + E_m D_m C_m
\end{aligned}
$$

where $D_m = \pm 1$ is chosen so that $D_m C_m$ and S_m have opposite signs. This drives S_m toward 0. In the process, Z_m is driven toward $\tanh^{-1}(S_0/C_0)$.

Our interest in the values of the inverse hyperbolic tangent stems from its connection with logarithms:

$$\tanh^{-1}(u) = \frac{1}{2}\ln\frac{1+u}{1-u}$$

This can be rewritten in a form more useful to CORDIC:

$$\ln(x) = 2\tanh^{-1}\frac{x-1}{x+1}$$

In particular, the calculation above shows that $2\tanh^{-1}(1/3) = 0.69315262$, a good approximation to $\ln(2) = 0.69314718\ldots$ This is how CORDIC directly evaluates logarithms that lie within the scope of the above algorithm.

Square roots

A few pages back, a question was raised about the inverse tangent algorithm; namely, what is the eventual value of the C-variable? (Recall that Z is driven toward the desired angle while S is driven toward 0.) The answer is that C is driven towards 1/P times the initial value of $\sqrt{C_0^2 + S_0^2}$. The is because the 24-step folding process *always* magnifies the length of the initial segment by a factor of $1/P$.

A similar question can be raised about the CORDIC algorithm for the inverse hyperbolic tangent; namely, what is the significance of the terminal value $C_{24} = 2.34231665$ in the preceding table? It should come as no surprise that it is just $1/Q$ times the initial value of $\sqrt{C_0^2 - S_0^2}$ (which is $\sqrt{8}$). There is no geometric model (fan) to make this clear, but it is not hard to verify that

$$C_{m+1}^2 - S_{m+1}^2 = \frac{C_m^2 - S_m^2}{[\cosh(B_m)]^2}$$

by using the hyperbolic identities; this provides an inductive proof.

In other words, $\sqrt{8} = C_{24}Q$. This is in fact how CORDIC evaluates square roots, as a *byproduct* of the calculation of inverse hyperbolic tangents:

> If the inverse hyperbolic tangent algorithm is applied to the seed values $C_0 = x + 0.25$ and $S_0 = x - 0.25$, then $\sqrt{x} = C_{24}Q$, because $C_0^2 - S_0^2 = x$.

In contrast to many of the other evaluations of elementary functions, it is interesting that this application of CORDIC *does* require a postmultiplication to complete.

Summary

The preceding examples demonstrate the utility and efficiency of the three recursive equations

$$Z_{m+1} = Z_m - D_m A_m$$
$$C_{m+1} = C_m - kE_m D_m S_m$$
$$S_{m+1} = S_m + E_m D_m C_m$$

for evaluating the standard elementary functions. In the above display, the letter k stands for either 1 (circular function mode) or -1 (hyperbolic function mode). The meanings of the symbols A_m depend on the setting of the mode indicator k. Each set of values is permanently stored in the hardware, along with the product of all the cosine values of these special arguments. The meaning of the symbols E_m also depends on the value of k; in either case, however, they are powers of $1/2$, and thus represent binary-point shifting in the actual arithmetic. The remaining numbers are variables whose values depend on the immediate application.

The actual implementation of this theory requires considerable refinement, and there are features that no doubt vary from one calculator to another. For example, achieving satisfactory accuracy requires that the calculator process more than just two dozen terms; how many of course depends on the number of significant digits the calculator is designed to display.

Another issue that affects the implementation is the conversion of numerical data from decimal form to binary form or from binary form to decimal form. The time that this takes can be viewed as the price (not insignificant) that must be paid for the ease of using the above algorithms. There are other options, however. For instance, it is not necessary to work in binary scale. The preceding approach could be reworked using powers of $1/10$ instead of powers of $1/2$; it is only necessary (as was done for the hyperbolic examples) to work with lists that contain many duplicate entries.

Another detail that has been largely ignored in the whole discussion is the question of scope. As described, CORDIC can only handle requests for trigonometric values that are found in the first quadrant. This is seen to be an easy hurdle to get over, thanks to (among other things) the periodicity of the circular functions. The question becomes conspicuous, however, when the mode is switched to hyperbolic. In the simple version described above, CORDIC can evaluate ex only for x-values between 0.0 and 1.1264, and $\ln(x)$ only for x-values between 1.0 and 9.5149. It is thus clear that there must be additional algorithms at work when a calculator evaluates $\ln(100) = 4.605170186$ or when it evaluates $e^{10} = 22026.46579$ These algorithms are beyond the scope of this paper.

Extending the range of convergence of CORDIC is a significant challenge. As reference [3] illustrates, it is a topic of current research. The optimal solution has not yet been found.

A final surprise

There is no obvious reason for doing so, but setting the mode variable k equal to 0 has interesting consequences, if the associated list of special arguments is given by $A_m = 2^{-m}$. The recursive equations now look like

$$\begin{aligned} Z_{m+1} &= Z_m - 2^{-m} D_m \\ C_{m+1} &= C_m \\ S_{m+1} &= S_m + 2^{-m} D_m C_m \end{aligned}$$

Suppose that initial values C_0 and Z_0 are given, S_0 is initialized at 0, and that the D_m are chosen (as usual) to drive Z_m to 0. In the process, S_m will be driven toward $Z_0 C_0$. This is how CORDIC does ordinary multiplication! In reverse gear, Z_0 is initialized at 0 and values C_0 and S_0 are given. As S_m is driven toward 0, Z_m is driven toward S_0/C_0. This is how CORDIC does ordinary division!

It is not hard to see why this works. Take multiplication, for example: The result of driving Z_m toward zero is to generate a binary expansion

$$Z_{24} = D_0 2^0 + D_1 2^{-1} + \cdots + D_{23} 2^{-23}$$

in which the coefficients D_m are all ± 1. As above, this approximates Z_0 to a known degree of accuracy. Meanwhile, the algorithm is generating the value

$$S_{24} = D_0 C_0 2^0 + D_1 C_0 2^{-1} + \cdots + D_{23} C_0 2^{-23} = Z_{24} C_0$$

which approximates $Z_0 C_0$.

Thus the CORDIC mode $k = 0$ (which might be called the *linear* mode, in contrast to the circular mode $k = 1$ and the hyperbolic mode $k = -1$) also provides the basic arithmetic services of multiplication and division. For further details (and additional references), see the references.

Richard Parris
Phillips Exeter Academy
Exeter, NH 03833-2460

References

1. James Kropa, "Calculator Algorithms", *Mathematics Magazine*, volume 51 (March 1978), pages 106-109.
2. Charles Schelin, "Calculator Function Approximation", *The American Mathematical Monthly*, volume 90 (May 1983), pages 317-325.
3. Xiaobo Hu, Ronald Harber, and Steven Bass, "Expanding the Range of Con- vergence of the CORDIC Algorithm", *IEEE Transactions on Computers*, volume 40 (January 1991), pages 13-21.

