



WIKIPEDIA  
The Free Encyclopedia

[Main page](#)

[Contents](#)

[Featured content](#)

[Current events](#)

[Random article](#)

[Donate to Wikipedia](#)

[Wikipedia store](#)

Interaction

[Help](#)

[About Wikipedia](#)

[Community portal](#)

[Recent changes](#)

[Contact page](#)

Tools

[What links here](#)

[Related changes](#)

[Upload file](#)

[Special pages](#)

[Permanent link](#)

[Page information](#)

[Wikidata item](#)

[Cite this page](#)

Print/export

[Create a book](#)

[Download as PDF](#)

[Printable version](#)

Languages

[Català](#)

[Čeština](#)

[Deutsch](#)

[Español](#)

[Français](#)

[Português](#)

[Русский](#)

[Türkçe](#)

[Українська](#)

[Edit links](#)

[Create account](#) [Log in](#)

Article

[Talk](#)

[Read](#)

[Edit](#)

[View history](#)



# CORDIC

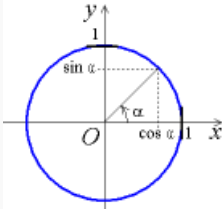
From Wikipedia, the free encyclopedia

**CORDIC** (for **CO**ordinate **R**otation **D**igital **C**omputer), also known as the **digit-by-digit method** and **Volder's algorithm**, is a simple and efficient [algorithm](#) to calculate [hyperbolic](#) and [trigonometric functions](#). It is commonly used when no [hardware multiplier](#) is available (e.g. in simple [microcontrollers](#) and [FPGAs](#)) as the only operations it requires are addition, subtraction, [bitshift](#) and [table lookup](#).

## Contents

- 1 Origins
- 2 Applications
  - 2.1 Hardware
  - 2.2 Software
- 3 Mode of operation: rotation mode
- 4 Mode of operation: vectoring mode
- 5 Software implementation
- 6 Hardware implementation
- 7 Related algorithms
- 8 History
- 9 Notes
- 10 References
- 11 External links

## Trigonometry



[Outline](#) · [History](#) · [Usage](#)  
[Functions \(inverse\)](#)  
[Generalized trigonometry](#)

## Reference

[Identities](#) · [Exact constants](#) · [Tables](#)

## Laws and theorems

[Sines](#) · [Cosines](#) · [Tangents](#) · [Cotangents](#)  
[Pythagorean theorem](#)

## Calculus

[Trigonometric substitution](#)  
[Integrals \(inverse functions\)](#)  
[Derivatives](#)

v · t · e

## Origins

The modern CORDIC algorithm was first described in 1959 by **Jack E. Volder**. It was developed at the aerelectronics department of [Convair](#) to replace the [analog resolver](#) in the [B-58 bomber's navigation](#) computer.<sup>[1]</sup>

Although CORDIC is similar to mathematical techniques published by [Henry Briggs](#) as early as 1624, it is optimized for low complexity finite state CPUs.

**John Stephen Walther** at [Hewlett-Packard](#) further generalized the algorithm, allowing it to calculate [hyperbolic](#) and [exponential functions](#), [logarithms](#), [multiplications](#), [divisions](#), and [square roots](#).<sup>[2]</sup> The CORDIC [subroutines](#) for trigonometric and hyperbolic functions can share most of their code.

Originally, CORDIC was implemented using the [binary numeral system](#). In the 1970s, [decimal](#) CORDIC became widely used in pocket [calculators](#), most of which operate in [binary-coded decimal](#) (BCD) rather than binary. This change in the input and output format did not alter CORDIC's core calculation algorithms. CORDIC is particularly well-suited for handheld calculators, in which low cost – and thus low chip gate count – is much more important than speed.

CORDIC has been implemented in the cores of some [x86](#) CPUs for some kinds of [floating point](#) instructions, mainly as a way to reduce the gate counts (and complexity) of the [FPU](#) subsystem.<sup>[*citation needed*]</sup>

## Applications

CORDIC uses simple shift-add operations for several computing tasks such as the calculation of trigonometric, hyperbolic and logarithmic functions, real and complex multiplications, division, square-root calculation, solution of linear systems, eigenvalue estimation, singular value decomposition, QR factorization and many others. As a consequence, CORDIC has been used for applications in diverse areas such as signal and image processing, communication systems, robotics and 3-D graphics apart from general scientific and technical computation.<sup>[3][4]</sup>

## Hardware

CORDIC is generally faster than other approaches when a hardware multiplier is not available (e.g., a

microcontroller), or when the number of gates required to implement the functions it supports should be minimized (e.g., in an [FPGA](#)).

On the other hand, when a hardware multiplier is available (e.g., in a DSP microprocessor), table-lookup methods and [power series](#) are generally faster than CORDIC. In recent years, the CORDIC algorithm has been used extensively for various biomedical applications, especially in [FPGA](#) implementations.

## Software [\[edit\]](#)

Many older systems with integer-only CPUs have implemented CORDIC to varying extents as part of their IEEE Floating Point libraries. As most modern general-purpose CPUs have floating-point registers with common operations such as add, subtract, multiply, divide, sin, cos, square root, log10, natural log, the need to implement CORDIC in them with software is nearly non-existent. Only microcontroller or special safety and time-constrained software applications would need to consider using CORDIC.

## Mode of operation: rotation mode [\[edit\]](#)

CORDIC can be used to calculate a number of different functions. This explanation shows how to use CORDIC in *rotation mode* to calculate the sine and cosine of an angle, and assumes the desired angle is given in radians and represented in a [fixed point](#) format. To determine the sine or cosine for an angle  $\beta$ , the y or x coordinate of a point on the [unit circle](#) corresponding to the desired angle must be found. Using CORDIC, we would start with the vector  $v_0$ :

$$v_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In the first iteration, this vector is rotated 45° counterclockwise to get the vector  $v_1$ . Successive iterations rotate the vector in one or the other direction by size-decreasing steps, until the desired angle has been achieved. Step  $i$  size is  $\arctan(1/(2^{i-1}))$  for  $i = 1, 2, 3, \dots$

More formally, every iteration calculates a rotation, which is performed by multiplying the vector  $v_{i-1}$  with the [rotation matrix](#)  $R_i$ :

$$v_i = R_i v_{i-1}$$

The rotation matrix is given by:

$$R_i = \begin{bmatrix} \cos \gamma_i & -\sin \gamma_i \\ \sin \gamma_i & \cos \gamma_i \end{bmatrix}$$

Using the following two [trigonometric identities](#):

$$\cos \gamma_i = \frac{1}{\sqrt{1 + \tan^2 \gamma_i}}$$

$$\sin \gamma_i = \frac{\tan \gamma_i}{\sqrt{1 + \tan^2 \gamma_i}}$$

the rotation matrix becomes:

$$R_i = \frac{1}{\sqrt{1 + \tan^2 \gamma_i}} \begin{bmatrix} 1 & -\tan \gamma_i \\ \tan \gamma_i & 1 \end{bmatrix}$$

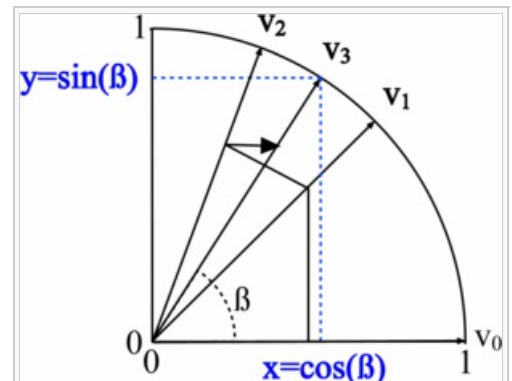
The expression for the rotated vector  $v_i = R_i v_{i-1}$  then becomes:

$$v_i = \frac{1}{\sqrt{1 + \tan^2 \gamma_i}} \begin{bmatrix} 1 & -\tan \gamma_i \\ \tan \gamma_i & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$

where  $x_{i-1}$  and  $y_{i-1}$  are the components of  $v_{i-1}$ . Restricting the angles  $\gamma_i$  so that  $\tan \gamma_i$  takes on the values  $\pm 2^{-i}$ , the multiplication with the tangent can be replaced by a division by a power of two, which is efficiently done in digital computer hardware using a [bit shift](#). The expression then becomes:

$$v_i = K_i \begin{bmatrix} 1 & -\sigma_i 2^{-i} \\ \sigma_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1} \\ y_{i-1} \end{bmatrix}$$

where



An illustration of the CORDIC algorithm in progress. □

$$K_i = \frac{1}{\sqrt{1 + 2^{-2i}}}$$

and  $\sigma_i$  can have the values of  $-1$  or  $1$ , and is used to determine the direction of the rotation; if the angle  $\beta_i$  is positive then  $\sigma_i$  is  $+1$ , otherwise it is  $-1$ .

$K_i$  can be ignored in the iterative process and then applied afterward with a scaling factor:

$$K(n) = \prod_{i=0}^{n-1} K_i = \prod_{i=0}^{n-1} 1/\sqrt{1 + 2^{-2i}}$$

which is calculated in advance and stored in a table, or as a single constant if the number of iterations is fixed. This correction could also be made in advance, by scaling  $\mathbf{v}_0$  and hence saving a multiplication. Additionally it can be noted that:

$$K = \lim_{n \rightarrow \infty} K(n) \approx 0.6072529350088812561694^{[5]}$$

to allow further reduction of the algorithm's complexity.

After a sufficient number of iterations, the vector's angle will be close to the wanted angle  $\beta$ . For most ordinary purposes, 40 iterations ( $n = 40$ ) is sufficient to obtain the correct result to the 10th decimal place.

The only task left is to determine if the rotation should be clockwise or counterclockwise at each iteration (choosing the value of  $\sigma$ ). This is done by keeping track of how much the angle was rotated at each iteration and subtracting that from the wanted angle; then in order to get closer to the wanted angle  $\beta$ , if  $\beta_{n+1}$  is positive, the rotation is clockwise, otherwise it is negative and the rotation is counterclockwise.

$$\beta_i = \beta_{i-1} - \sigma_i \gamma_i. \quad \gamma_i = \arctan 2^{-i},$$

The values of  $\gamma_n$  must also be precomputed and stored. But for small angles,  $\arctan(\gamma_n) = \gamma_n$  in fixed point representation, reducing table size.

As can be seen in the illustration above, the sine of the angle  $\beta$  is the  $y$  coordinate of the final vector  $\mathbf{v}_n$ , while the  $x$  coordinate is the cosine value.

## Mode of operation: vectoring mode [\[edit\]](#)

The rotation-mode algorithm described above can rotate any vector (not only a unit vector aligned along the  $x$  axis) by an angle between  $-90^\circ$  and  $+90^\circ$ . Decisions on the direction of the rotation depend on  $\beta_i$  being positive or negative.

The vectoring-mode of operation requires a slight modification of the algorithm. It starts with a vector the  $x$  coordinate of which is positive and the  $y$  coordinate is arbitrary. Successive rotations have the goal of rotating the vector to the  $x$  axis (and therefore reducing the  $y$  coordinate to zero). At each step, the value of  $y$  determines the direction of the rotation. The final value of  $\beta_i$  contains the total angle of rotation. The final value of  $x$  will be the magnitude of the original vector scaled by  $K$ . So, an obvious use of the vectoring mode is the transformation from rectangular to polar coordinates.

## Software implementation [\[edit\]](#)

The following is a [MATLAB/GNU Octave](#) implementation of CORDIC that does not rely on any transcendental functions except in the precomputation of tables. If the number of iterations  $n$  is predetermined, then the second table can be replaced by a single constant. The two-by-two [matrix multiplication](#) represents a pair of simple shifts and adds. With MATLAB's standard double-precision arithmetic and "format long" printout, the results increase in accuracy for  $n$  up to about 48.

```
function v = cordic(beta,n)
% This function computes v = [cos(beta), sin(beta)] (beta in radians)
% using n iterations. Increasing n will increase the precision.

if beta < -pi/2 || beta > pi/2
    if beta < 0
        v = cordic(beta + pi, n);
    else
        v = cordic(beta - pi, n);
    end
v = -v; % flip the sign for second or third quadrant
return
end
```

```

% Initialization of tables of constants used by CORDIC
% need a table of arctangents of negative powers of two, in radians:
% angles = atan(2.^(-(0:27)));
angles = [ ...
    0.78539816339745    0.46364760900081    0.24497866312686    0.12435499454676 ...
    0.06241880999596    0.03123983343027    0.01562372862048    0.00781234106010 ...
    0.00390623013197    0.00195312251648    0.00097656218956    0.00048828121119 ...
    0.00024414062015    0.00012207031189    0.00006103515617    0.00003051757812 ...
    0.00001525878906    0.00000762939453    0.00000381469727    0.00000190734863 ...
    0.00000095367432    0.00000047683716    0.00000023841858    0.00000011920929 ...
    0.00000005960464    0.00000002980232    0.00000001490116    0.00000000745058 ];

% and a table of products of reciprocal lengths of vectors [1, 2^-2j]:
% Kvalues = cumprod(1./abs(1 + 1j*2.^(-(0:23))))
Kvalues = [ ...
    0.70710678118655    0.63245553203368    0.61357199107790    0.60883391251775 ...
    0.60764825625617    0.60735177014130    0.60727764409353    0.60725911229889 ...
    0.60725447933256    0.60725332108988    0.60725303152913    0.60725295913894 ...
    0.60725294104140    0.60725293651701    0.60725293538591    0.60725293510314 ...
    0.60725293503245    0.60725293501477    0.60725293501035    0.60725293500925 ...
    0.60725293500897    0.60725293500890    0.60725293500889    0.60725293500888 ];

Kn = Kvalues(min(n, length(Kvalues)));

% Initialize loop variables:
v = [1;0]; % start with 2-vector cosine and sine of zero
poweroftwo = 1;
angle = angles(1);

% Iterations
for j = 0:n-1;
    if beta < 0
        sigma = -1;
    else
        sigma = 1;
    end
    factor = sigma * poweroftwo;
    R = [1, -factor; factor, 1];
    v = R * v; % 2-by-2 matrix multiply
    beta = beta - sigma * angle; % update the remaining angle
    poweroftwo = poweroftwo / 2;
    % update the angle from table, or eventually by just dividing by two
    if j+2 > length(angles)
        angle = angle / 2;
    else
        angle = angles(j+2);
    end
end

% Adjust length of output vector to be [cos(beta), sin(beta)]:
v = v * Kn;
return
endfunction

```

## Hardware implementation [\[edit\]](#)

The number of [logic gates](#) for the implementation of a CORDIC is roughly comparable to the number required for a multiplier as both require combinations of shifts and additions. The choice for a multiplier-based or CORDIC-based implementation will depend on the context. The multiplication of two [complex numbers](#) represented by their real and imaginary components (rectangular coordinates), for example, requires 4 multiplications, but could be realized by a single CORDIC operating on complex numbers represented by their polar coordinates, especially if the magnitude of the numbers is not relevant (multiplying a complex vector with a vector on the unit circle actually amounts to a rotation). CORDICs are often used in circuits for telecommunications such as [digital down converters](#).

## Related algorithms [\[edit\]](#)

CORDIC is part of the class of "shift-and-add" algorithms, as are the logarithm and exponential algorithms derived from Henry Briggs' work. Another shift-and-add algorithm which can be used for computing many

elementary functions is the [BKM algorithm](#), which is a generalization of the logarithm and exponential algorithms to the complex plane. For instance, BKM can be used to compute the sine and cosine of a real angle  $x$  (in radians) by computing the exponential of  $0 + ix$ , which is  $\cos x + i \sin x$ . The BKM algorithm is slightly more complex than CORDIC, but has the advantage that it does not need a scaling factor (K).

## History [\[edit\]](#)

Volder was inspired by the following formula in the 1946 edition of the [CRC Handbook of Chemistry and Physics](#):

$$K_n R \sin(\theta \pm \phi) = R \sin(\theta) \pm 2^{-n} R \cos(\theta)$$

$$K_n R \cos(\theta \pm \phi) = R \cos(\theta) \mp 2^{-n} R \sin(\theta)$$

with  $K_n = \sqrt{1 + 2^{-2n}}$ ,  $\tan(\phi) = 2^{-n}$ .<sup>[1]</sup>

Some of the prominent early applications of CORDIC were in the Convair navigation computers CORDIC I to CORDIC III,<sup>[1]</sup> the Hewlett-Packard [HP-9100](#) and [HP-35](#) calculators,<sup>[6]</sup> the [Intel 80x87](#) coprocessor series until [Intel 80486](#), and [Motorola 68881](#).<sup>[7]</sup>

Decimal CORDIC was first suggested by Hermann Schmid and Anthony Bogacki.<sup>[8]</sup>

## Notes [\[edit\]](#)
















- ↑  a b c J. E. Volder, "The Birth of CORDIC", J. VLSI Signal Processing **25**, 101 (2000). [↗](#)
- ↑ J. S. Walther, "The Story of Unified CORDIC", J. VLSI Signal Processing **25**, 107 (2000). [↗](#)
- ↑ P. K. Meher, J. Valls, T-B Juang, K. Sridharan, and K. Maharatna, '50 Years of CORDIC: Algorithms, Architectures and Applications,' IEEE Transactions on Circuits & Systems-I: Regular Papers, vol.56, no. 9, pp. 1893–1907, September 2009
- ↑ P. K. Meher and S. Y. Park, 'CORDIC Designs for Fixed Angle of Rotation,' IEEE Transactions on VLSI Systems, vol.21, no.2, pp. 217–228, February 2013.
- ↑ J.-M. Muller, *Elementary Functions: Algorithms and Implementation*, 2nd Edition (Birkhäuser, Boston, 2006), p. 134. [↗](#)
- ↑ D. Cochran, "Algorithms and Accuracy in the HP 35", Hewlett Packard J. **23**, 10 (1972).
- ↑ R. Nave, "Implementation of Transcendental Functions on a Numerics Processor", Microprocessing and Microprogramming **11**, 221 (1983).
- ↑ H. Schmid and A. Bogacki, "Use Decimal CORDIC for Generation of Many Transcendental Functions", EDN Magazine, February 20, 1973, p. 64.

## References [\[edit\]](#)

- Jack E. Volder, *The CORDIC Trigonometric Computing Technique*, IRE Transactions on Electronic Computers, pp. 330–334, September 1959 [↗](#)
- Daggett, D. H., *Decimal-Binary conversions in CORDIC*, IRE Transactions on Electronic Computers, Vol. EC-8 #5, pp. 335–339, IRE, September 1959
- John S. Walther, *A Unified Algorithm for Elementary Functions*, Proc. of Spring Joint Computer Conference, pp. 379–385, May 1971 [↗](#)
- P. K. Meher, J. Valls, T-B Juang, K. Sridharan, and K. Maharatna, '50 Years of CORDIC: Algorithms, Architectures and Applications,' IEEE Transactions on Circuits & Systems-I: Regular Papers, vol.56, no.9, pp. 1893–1907, September 2009. [↗](#)
- P. K. Meher and S. Y. Park, 'CORDIC Designs for Fixed Angle of Rotation,' IEEE Transactions on VLSI Systems, vol.21, no.2, pp. 217–228, February 2013. [↗](#)
- J. E. Meggitt, *Pseudo Division and Pseudo Multiplication Processes*, IBM Journal, April 1962 [↗](#)
- Vladimir Baykov, *Problems of Elementary Functions Evaluation Based on Digit by Digit (CORDIC) Technique*, PhD thesis, Leningrad State Univ. of Electrical Eng., 1972 [↗](#)
- Schmid, Hermann, *Decimal computation*. New York, Wiley, 1974
- V.D.Baykov,V.B.Smolov, *Hardware implementation of elementary functions in computers*, Leningrad State University, 1975, 96p. [↗](#)\*Full Text [↗](#)
- Senzig, Don, *Calculator Algorithms*, IEEE Compcon Reader Digest, IEEE Catalog No. 75 CH 0920-9C, pp. 139–141, IEEE, 1975.
- V.D.Baykov,S.A.Seljutin, *Elementary functions evaluation in microcalculators*, Moscow, Radio & svjaz,1982,64p.
- Vladimir D.Baykov, Vladimir B.Smolov, *Special-purpose processors: iterative algorithms and structures*, Moscow, Radio & svjaz, 1985, 288 pages [↗](#)

- M. E. Frerking, *Digital Signal Processing in Communication Systems*, 1994
- Vitit Kantabutra, *On hardware for computing *exponential* and *trigonometric functions**, IEEE Trans. Computers 45 (3), 328–339 (1996)
- Andraka, Ray, *A survey of CORDIC algorithms for FPGA based computers* 
- Henry Briggs, *Arithmetica Logarithmica*. London, 1624, folio
- *CORDIC Bibliography Site* , Shaoyun Wang, July 2011
- *The secret of the algorithms* , Jacques Laporte, Paris 1981
- *Digit by digit methods* , Jacques Laporte, Paris 2006
- Ayan Banerjee, *FPGA realization of a CORDIC based FFT processor for biomedical signal processing* , Kharagpur, 2001
- *CORDIC Architectures: A Survey* , B. Lakshmi and A. S. Dhar, Journal: VLSI Design, January 2010
- *Implementation of a CORDIC Algorithm in a Digital Down-Converter* , C. Cockrum, Fall 2008

## External links

- CORDIC Bibliography Site 
- Another USENET discussion 
- BASIC Stamp, CORDIC math implementation 
- CORDIC as implemented in the ROM of the HP-35 – Jacques Laporte (step by step analysis, simulator running the real ROM with breakpoints and trace facility) 
- CORDIC implementation in verilog 
- CORDIC information 
- CORDIC Vectoring with Arbitrary Target Value 
- PicBasic Pro, Pic18 CORDIC math implementation 
- Python CORDIC implementation 
- Simple C code for fixed-point CORDIC 
- The CORDIC Algorithm 
- Tutorial and MATLAB Implementation – Using CORDIC to Estimate Phase of a Complex Number 
- USENET discussion 
- Descriptions of hardware CORDICs in Arx with testbenches in C++ and VHDL 
- Usenet discussion about CORDIC 

Categories: [Numerical analysis](#) | [Trigonometry](#)

This page was last modified on 10 July 2015, at 16:45.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

