Efficient Algorithms for calculating fixed-point logarithm of large numbers

Classical Algos:

1. **Binary logarithm algorithm** - http://www.claysturner.com/dsp/BinaryLogarithm.pdf

   Algorithm:

   1) Initialize result to 0: y = 0.
   2) Initialize mantissa-bit decimal value to 0.5: b = 1/2.
   3) While x < 1, x = 2x, y = y - 1.
   4) While x > 2, x = x/2, y = y + 1.
   5) Go to Step 3 and repeat until 1 <= x < 2.
   6) Square: x = x*x.
   7) If x > 2, x = x/2, y = y + b.
   8) Scale for next bit: b = b/2.
   9) Go to Step 6 and repeat until the desired number of mantissa bits are found.
   10) Final log(x) value: y

   To start, we simply desire to find y = log(x).
   Thus we can invert this and find $x = 2^y$
   our log algorithm assumes x is in 1 <= x < 2.
   So if x>=2, use log(x) = log(x/2) + log(2)
   Else if x<1, use log(x) = log(x*2) - log(2). Since log(2) = 1, we can simply add or subtract the required number of 1s upon division or multiplication. This gives us the characteristic(integer part) of our answer

   Now $1 \leq x < 2$
   $So\ 0 \leq log(x) < 1\ ie\ 0 \leq y < 1$
   $Let\ y = 0.y_1y_2\ldots\ (in\ binary)$
   $y = y_1 * 2^{-1} + y_2 * 2^{-2} + y_3 * 2^{-3} + \ldots$
   $y = 2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y3 + \ldots)))$
   $x = 2^y = 2^{2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + \ldots)))}$

$$x^2 = 2^{2y} = 2^{2 * 2^{-1}(y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + \ldots)))}$$

$$x^2 = 2^{y_1 + 2^{-1}(y_2 + 2^{-1}(y_3 + \ldots))}$$

$$= 2^{y_1} * 2^{2^{-1}(y2 + 2^{-1}(y3 + \ldots))}$$

$$= 2^{y_1} * some\ number\ k \qquad where\ 1 <= k < 2$$

$If\ y_1 = 0$:

$$2^{y_1} = 1\ and\ 1 \leq x^2 < 2$$

$Else$:

$$y_1 = 1$$

$$2^{y_1} = 2\ and\ 2 \leq x^2 \leq 4$$

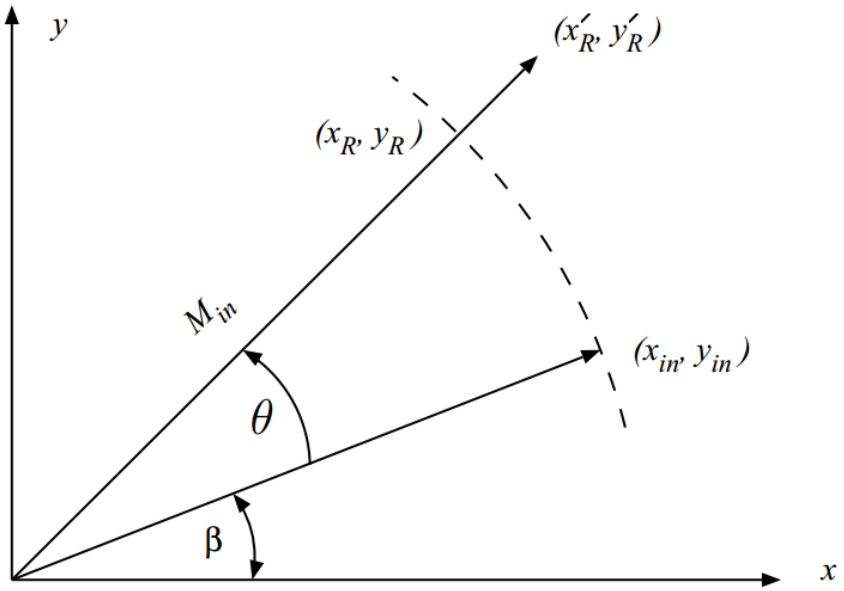
$So\ if\ x^2 \prec 2\ then\ y_1 = 0.\ Otherwise\ y_1 = 1$ and thus we have found the first digit of our logarithm's mantissa(decimal part)

We then repeat these steps for $2^{2^{-1}(y_2+2^{-1}(y_3+\ldots))}$ for as many bits as we want

2. CORDIC Algo

Uses only addition, subtraction, bit-shifts
Efficient for hardware implementation of log

Consider the point (x_0, y_0)
If we rotate this point clockwise by an angle Θ about the origin, then



Source:
https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/

$x' = xcos\Theta - ysin\Theta$ and
$y' = xsin\Theta + ycos\Theta$

This set of equations can be written as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \cos\theta \begin{bmatrix} 1 & -\tan\theta \\ \tan\theta & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The CORDIC algorithm resorts to two fundamental ideas to achieve rotation without multiplication. The first fundamental idea is that rotating the input vector by an arbitrary angle θ is equal to rotating the vector by several smaller angles, θ_i, i=0,1,…,n, provided

$$\theta = \sum_{i=0}^{n} \theta_i$$

For example, a rotation of 57.535° is the same as three successive rotations by $45^\circ(=\tan^{-1}(2^0))$, $26.565^\circ(=\tan^{-1}(2^{-1}))$, and $-14.03^\circ(=\tan^{-1}(-2^{-2}))$.

we can choose the small elementary angles in a way that $\tan(\theta_i)=2^{-i}$ for i=0,1,…,n. In this way, multiplication by $\tan(\theta_i)$ can be achieved by a bitwise shift which is a much faster operation compared to multiplication

Each rotation also has a scaling factor of cosΘ. it is possible to ignore the cos(θ) term and take the scaling factor into account at the end of the algorithm. As we proceed with the algorithm, the angle of rotation rapidly becomes smaller and smaller. Hence, cos(θ) tends toward unity.

For example, with i=6, θ_i becomes $\tan^{-1}(2^{-6}) = 0.895^\circ$ which leads to a scaling factor of $\cos(0.895^\circ)=0.99987$. As a result, if the algorithm is designed to have more than six iterations, looking at the four significant figures, we obtain the following scaling factor:

$K\approx\cos(45^\circ)\cos(26.565^\circ)\times\cdots\times\cos(0.895^\circ)=0.6072$

In summary, we can ignore the cos(θ) term and apply a scaling factor of approximately 0.6072 at the end of the process regardless of the desired rotation angle.

So the final equations become

$x_{i+1} = x_i - \delta_i y_i tan\Theta_i$ and

$y_{i+1} = y_i + \delta_i x_i tan\Theta_i$ where δ_i determines the sign of the elementary angle Θ_i

Approximating $tan\Theta_i$ = 2^{-i}

$x_{i+1} = x_i - \delta_i y_i 2^{-i}$ and

$y_{i+1} = y_i + \delta_i x_i 2^{-i}$

δ_i = -1, if If the desired rotation is smaller than previously achieved rotation(so we try to bring it down to the desired value). Else δ_i = 1

Error in the angle rotation is given by

$$\theta_{error} = \theta - \sum_{i=0}^{n} \theta_i$$

This can be expressed using another equation

$z_{i+1} = z_i - \delta_i tan^{-1}(2^{-i})$ where $z_0 = \Theta$

Applications of this approach:

(i) If initial values $x_0 = 1 \; and \; y_0 = 0$ then

$x_f = cos\Theta$ and

$y_f = sin\Theta$

Hence we find sinΘ and cosΘ using the above method

(ii) consider the vector magnitude $\sqrt{{x_0}^2 + {y_0}^2}$.

To achieve this, we only need to rotate the input vector so that it is aligned with the x-axis. In this way, y_f=0(final y component) and the x_f will give the vector magnitude

More generally

$x_{i+1} = x_i - m\,\delta_i y_i 2^{-i}$ and

$y_{i+1} = y_i + \delta_i x_i 2^{-i}$

The curves of constant radius for the circular (m=1), linear (m = 0), and hyperbolic (m= -1) coordinate systems are shown

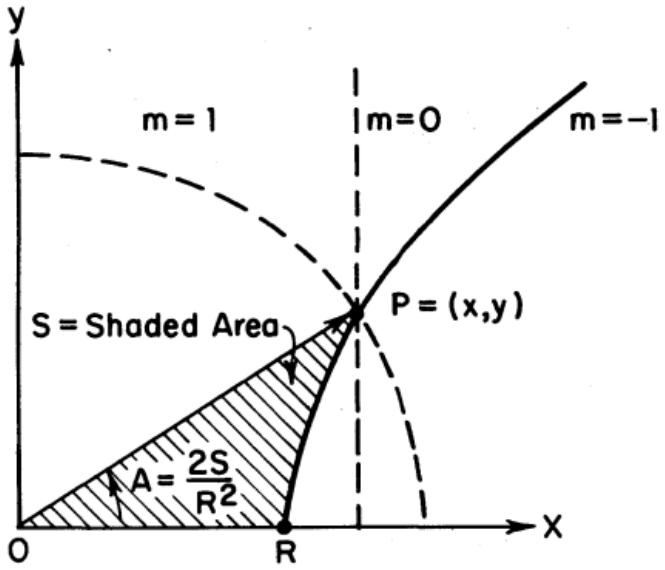


Figure 1—Angle A and Radius R of the vector $P=(x, y)$

To compute log we use m=-1, the hyperbolic CORDIC algorithm

The equations are

$$X_{i+1} = X_i + \delta_i Y_i 2^{-i}$$

$$Y_{i+1} = Y_i + \delta_i X_i 2^{-i}$$

$$Z_{i+1} = Z_i - \delta_i tanh^{-1}(2^{-i})$$

We use the vectoring mode and hence δ_i = -1 if $x_i y_i$ >= 0 and +1 otherwise
Thus for sufficiently large N, the quantities tend to

$$X_N \leftarrow A_N\sqrt{X_0{}^2 - Y_0{}^2}$$

$$Y_N \leftarrow 0$$

$$Z_N \leftarrow Z_0 + \tanh^{-1}\left(\frac{Y_0}{X_0}\right)$$

$$A_N \leftarrow \prod_{i=1}^{N}\sqrt{1-2^{-2i}}$$

$$Since\ \ln(a) = 2\tanh^{-1}\left(\frac{a-1}{a+1}\right),$$

$$\ln(a) = 2\cdot Z_N\ provided\ Z_0 = 0,\ X_0 = a+1\ and\ Y_0 = a-1$$

Range of values this method supports:

$$\left|\tanh^{-1}\left(\frac{Y_0}{X_0}\right)\right| \le \theta_N + \sum_{i=1}^{N}\theta_i$$

$$\left|\tanh^{-1}\left(\frac{Y_0}{X_0}\right)\right| \le 1.1182\ for\ N \to \infty$$

$$\left|\frac{Y_0}{X_0}\right|_{max} \approx 0.80694\ \ for\ N \to \infty$$

$$\left|\ln(a)_{\ max}\right| = \left|2\tanh^{-1}\left(\frac{a-1}{a+1}\right)\right| = 2.2364$$

$$Since\ X_0 = a+1\ and\ Y_0 = a-1\ then$$

$$\left|\frac{a-1}{a+1}\right| \le 0.80694$$

$$\Rightarrow 0.106843 \le a \le 9.35947$$

This can be tackled by modifying the basic hyperbolic CORDIC algorithm, by including additional (M+1) iterations for negative indexes i: (i = 0, - 1, - 2, - 3, … - M)

$$\theta_i = Tanh^{-1}\left(1 - 2^{i-2}\right),\ for\ i \le 0 \qquad (12)$$

Therefore, the modified algorithm results:

$$For\ i \le 0 \quad \begin{cases} X_{i+1} = X_i + \delta_i\left(1 - 2^{i-2}\right)Y_i \\ Y_{i+1} = Y_i + \delta_i\left(1 - 2^{i-2}\right)X_i \\ Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(1 - 2^{i-2}\right) \end{cases} \qquad (13)$$

$$For\ i > 0 \quad \begin{cases} X_{i+1} = X_i + \delta_i Y_i 2^{-i} \\ Y_{i+1} = Y_i + \delta_i X_i 2^{-i} \\ Z_{i+1} = Z_i - \delta_i Tanh^{-1}\left(2^{-i}\right) \end{cases} \qquad (14)$$

X_N, Y_N, and Z_N are as indicated in (4). δ_i is as indicated in (3). But the quantity A_n, described in (5), is redefined as follows:

$$A_n \leftarrow \left[\prod_{i=-M}^{0}\sqrt{1-\left(1-2^{(i-2)}\right)^2}\right]\left[\prod_{i=1}^{N}\sqrt{1-2^{-2i}}\right] \qquad (15)$$

The range of convergence, now becomes:

$$\left|Tanh^{-1}\left(Y_0 / X_0\right)\right| \le \theta_{\max} \qquad (16)$$

$$\text{Where: } \theta_{\max} = \sum_{i=-M}^{0} Tanh^{-1}\left(1 - 2^{i-2}\right) + \left[Tanh^{-1}\left(2^{-N}\right) + \sum_{i=1}^{N} Tanh^{-1}\left(2^{-i}\right)\right] \qquad (17)$$

θ_{max} is the maximum value of $Z \leftarrow Tanh^{-1}$ (if $Z_0=0$), and imposes a limitation to X_0 and Y_0, and therefore to α. The values of θ_{max} are tabulated for M between 0 an 10 and shown in Table 1.

with M = 2 (θ_{max}=5.16215), the range of $Tanh^{-1}(\Theta)$ is [-5.16215, +5.16215]. Then $\alpha \in [3.2825 \times 10^{-5}, 30463.9711]$, which is a far larger domain of ln(α) than that of the original hyperbolic CORDIC algorithm. It is clear that the expansion scheme does work. The more domain of ln(α) is needed, the more the iterations (M+1) that must be executed.

References:

a) https://www.researchgate.net/publication/230668515_A_fixed-point_implementation_of_the_natural_logarithm_based_on_a_expanded_hyperbolic_CORDIC_algorithm (main paper)

b) https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/ (basic introduction to CORDIC algo)
c) https://dl.acm.org/doi/10.1145/1478786.1478840 (for the different types of CORDIC)

3. In C, log is implemented by a call to an assembly instruction -> hardware dependent
   intel IA64, apparently they use Taylor series combined with a table. - https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=CD2F3C5DA9CA809C0A1700930D20A611?doi=10.1.1.24.5177&rep=rep1&type=pdf

Some resources(don't know if they are helpful)

4. https://www.ams.org/journals/mcom/1954-08-046/S0025-5718-1954-0061464-9/S0025-5718-1954-0061464-9.pdf
5. https://www.brics.dk/RS/04/17/BRICS-RS-04-17.pdf
6. https://stackoverflow.com/questions/11376288/fast-computing-of-log2-for-64-bit-integers

General Observations: In the classical world, most researches(based on our search) have been focussing on implementing hardware for logarithm and the most efficient algorithm for a given computer depends on the hardware. Most techniques used in making the algorithm efficient include using fast operations like bit shift instead of multiplications and deal with parallelism, branch costs, etc…

Quantum Algos:

Found algos only for discrete log problem(finding integer k such that $b^k = a$ where a and b are elements of a multiplicative group)