Cairo University
Electronics and Communications Department

Computer Arithmetic: Elementary functions

Hossam A. H. Fahmy

What are the elementary functions

- For mathematicians, they are elementary. For hardware people, they are *Higher level functions*.
- Originally, calculators and computers only had: $\sin(x)$, $\log(x)$, $\sqrt[n]{x}$, $\tanh(x)$, ...
- Now, some DSPs may include other operations such as the gamma function.
- Simply, they are any functions that can be easily tabulated or represented in a series, a polynomial, or a ratio of polynomials.

Implementation steps

The approximation of an elementary function is more efficient (time delay and area) if the argument is constrained in a small interval.

Hence, there are three main steps in any elementary function calculation:

1. range reduction,
2. approximation, and
3. reconstruction.

Reduction and reconstruction

- The range reduction and reconstruction steps are related and they are function-dependent.
- There is no single reduction and reconstruction technique that is applicable to all functions.
- The modular reduction is applicable to the exponential and sinusoidal functions, the case of the logarithm is even simpler.

$$\begin{aligned}
e^x &= e^{N\ln(2)+y} = 2^N \times e^y \\
\sin(x) &= \sin(N \times \tfrac{\pi}{2} + y) \\
\sin(x) &= \quad \sin(y), \quad N\mathbf{mod}_4 = 0 \\
\sin(x) &= \quad \cos(y), \quad N\mathbf{mod}_4 = 1 \\
\sin(x) &= -\sin(y), \quad N\mathbf{mod}_4 = 2 \\
\sin(x) &= -\cos(y), \quad N\mathbf{mod}_4 = 3 \\
\log(x) &= \log(2^{exp} \times 1.f) = exp\ \log(2) + \log(1.f)
\end{aligned}$$

The approximation step

The approximation algorithms are classified as:

1. digit recurrence techniques,
2. functional recurrence techniques,
3. polynomial approximation techniques, and
4. rational approximation techniques.

The digit and functional recurrence

Digit recurrence techniques:

- converge linearly.
- use addition, subtraction, shift, and single digit multiplication.
- restoring/non-restoring division, SRT, Cordic, Briggs and DeLugish, . . .

Functional recurrence techniques:

- converge quadratically (or better for higher orders).
- use addition, subtraction, multiplication, and table lookup.
- Newton-Raphson of any order.

The polynomial approximation

Polynomial approximation techniques:

- depending on the function and the implementation details, may converge directly to the required precision.
- use addition, subtraction, multiplication, and table lookup.
- divide the interval of the argument to a number of sub-intervals where the elementary function is approximated by a polynomial of a suitable degree. One or more tables contain the coefficients of the polynomials.

The rational approximation

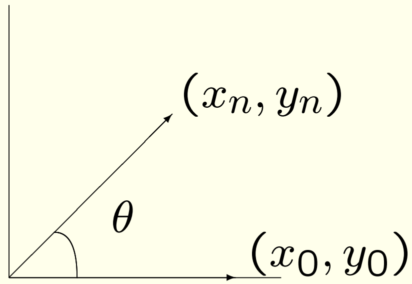
Rational approximation techniques:

- depending on the function and the implementation details, may converge directly to the required precision.
- use addition, subtraction, multiplication, tables, and *division*.
- for each sub-interval, approximate the given function by a rational function (a polynomial divided by another polynomial).

Digit recurrence: Cordic

- J. Volder in 1959 developed a digit by digit algorithm to compute all the trigonometric functions with minimal hardware support.
- The generalized algorithm calculates also the hyperbolic and the arc functions.
- Cordic has been widely used in calculators and in some processors.

Basic idea of Cordic



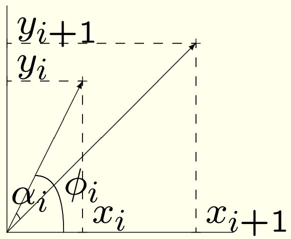
To reach an angle of θ, we rotate the initial vector in each iteration by a small angle $\alpha_i = \pm\tan^{-1} 2^{-i}$ and watch the error $z_i = \theta - \sum_{j=0}^{i} \alpha_j$.

At the end (when $z_n \approx 0$), we reach

$$\begin{aligned} x_n &= x_0 \cos\theta \\ y_n &= x_0 \sin\theta. \end{aligned}$$

Setting $x_0 = 1$, we directly get the cosine and sine functions.

Derivation of Cordic



We rotate the current vector (x_i, y_i) by an angle $\alpha_i = \pm\tan^{-1} 2^{-i}$.

$$\begin{aligned} x' &= R\cos(\phi_i + \alpha_i) \\ &= R(\cos\phi_i \cos\alpha_i - \sin\phi_i \sin\alpha_i) \\ &= x_i \cos\alpha_i - y_i \sin\alpha_i \\ \frac{x'}{\cos\alpha_i} &= x_i - y_i \tan\alpha_i \\ x_{i+1} &= x_i - d_i y_i 2^{-i} \end{aligned}$$

The simple Cordic iteration

Volder's algorithm is based on

$$\begin{aligned} x_{i+1} &= x_i - d_i y_i 2^{-i} \\ y_{i+1} &= y_i + d_i x_i 2^{-i} \\ z_{i+1} &= z_i - d_i \tan^{-1}(2^{-i}) \\ d_i &= 1 \text{ if } z_i \geq 0, \\ d_i &= -1 \text{ otherwise.} \end{aligned}$$

Notice that, with each iteration,

- the magnitude of z_i is decreasing by $|\alpha_i|$ and
- the magnitude of the vector is increasing due to the division by $\cos\alpha_i$.

i	0	1	2	3	4	5	6	7	8	9
α_i (degrees)	45	26.6	14	7.1	3.6	1.8	0.9	0.4	0.2	0.1

Compensation

Since $\alpha_i = \pm \tan^{-1} 2^{-i}$ then $\frac{1}{\cos \alpha_i} = \sqrt{1 + 2^{-2i}}$. Let us define

$$k = \prod_{i=0}^{\infty} \sqrt{1 + 2^{-2i}} = 1.646760258 \cdots$$

and start from

$$\begin{aligned} x_0 &= \frac{1}{k} = 0.60725293 \cdots \\ y_0 &= 0 \\ z_0 &= \theta. \end{aligned}$$

Is there a maximum for θ? What is it? What if you want to calculate for a larger angle?

Concluding Cordic

- What we have just explained is the *rotation mode* of the *circular* type. There is also a *vectoring mode* and two other types:*linear*, and *hyperbolic*.
- With the generalized Cordic, it is possible to compute many functions with minimal hardware support (three additions and a comparison).
- Cordic is slow but area efficient. It has been used in calculators and in the 8087 coprocessor.

Polynomial approximations

- To improve the precision, it is better to divide the domain of the input to sub-intervals and to have a specific polynomial for each sub-interval.
- Many different polynomials may approximate the same function, how do we choose the best? $\Rightarrow$ How do we define *best*?
- How do we actually make the calculation? What is the *best* way?

Different polynomials

- Taylor series are available for most functions if we know their derivatives. However, such series do not provide the minimum error term.
- Chebyshev polynomials minimize the maximum error (mini-max) in the domain of the approximation. However, the calculation of the coefficients of the polynomial may take some effort. Note that we actually use *truncated* coefficients so we need to compensate for that.
- Instead of saving the coefficients in a table, we can save the values of the function at various points and interpolate.
  - How many points? Which points?
  - How to interpolate between those points?

Evaluating a polynomial

It is possible to optimize the evaluation of a polynomial $f(x) = c_n x^n + c_{n-1}x^{n-1} + \cdots + c_0$ in order to minimize the hardware or to increase the speed.

For Hardware: use Horner's rule

$$f(x) = (\cdots(((c_n x + c_{n-1})x + c_{n-2})x + \cdots)x + c_0)$$

and iterate. This might be even driven and controlled by software.

For Speed:

- use parallel powering units to reach the required precision in one iteration if possible.
- use the PPA of a multiplier.

Rational approximations

If a polynomial approximation requires too many terms, there might be a rational function $R(x) = \frac{P(x)}{Q(x)}$ of a lower degree that gives a good approximation.

- To reduce the calculation order, you may use $R_{m,n} = x\frac{P_m(x^2)}{Q_n(x^2)}$.
- Typically, $R_{4,4}$ is enough for over sixty bits of precision. However, some functions require higher degrees.
- Rational approximations are "better" for double or extended precisions, Cordic is preferred for single precision.

Using the PPA of a multiplier

In 1993, Eric Schwarz proposed to use the PPA to evaluate the reciprocal. It is possible to generalize the idea for any polynomial.

- Write the operation in terms of the bits and expand the polynomial symbolically.
- Group the terms with the same power of 2 numerical weight and write them in columns.
- The resulting matrix is similar to the partial products. Hence, we add the rows of this matrix using the reduction tree and the carry propagate adder of the multiplier.

Reciprocal with PPA

If we want to calculate q such that $b \times q = 0.1111\cdots \approx 1.0$ where $b = 0.1b_2b_3b_4\cdots$ we write

$$\begin{array}{cccccccccccc}
 & & & & 0. & 1 & b_2 & b_3 & b_4 & b_5 & \cdots & = b \\
 & & & \times & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & \cdots & = q \\
\hline
 & & & & & & \vdots & & & & & \\
 & & & & & q_4 & q_4b_2 & q_4b_3 & q_4b_4 & q_4b_5 & \cdots & \\
 & & & & q_3 & q_3b_2 & q_3b_3 & q_3b_4 & q_3b_5 & \cdots & & \\
 & & & q_2 & q_2b_2 & q_2b_3 & q_2b_4 & q_2b_5 & \cdots & & & \\
 & & q_1 & q_1b_2 & q_1b_3 & q_1b_4 & q_1b_5 & \cdots & & & & \\
 & q_0 & q_0b_2 & q_0b_3 & q_0b_4 & q_0b_5 & \cdots & & & & & \\
\hline
0. & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \approx 1
\end{array}$$

and use *redundant* digits for the q_i to make each column independent. Hence,

$$\begin{aligned}
q_0 &= 1 \\
q_1 + q_0b_2 &= 1 \\
q_2 + q_1b_2 + q_0b_3 &= 1
\end{aligned}$$

Steps of PPA implementation

- Solve the equations:

$$\begin{aligned} q_0 &= 1 \\ q_1 &= 1 - q_0 b_2 = 1 - b_2 \\ q_2 &= 1 - q_1 b_2 - q_0 b_3 = 1 - b_3 \end{aligned}$$

- Put in a PPA form:

q_0	q_1	q_2	q_3	q_4	$\cdots$
			$-b_4$	$-b_5$	
			$-b_3$	$2b_2b_4$	$\cdots$
			$2b_2b_3$	$-b_4$	
	$-b_2$	$-b_3$	$-b_2$	$-b_2b_3$	
1	1	1	1	1	

Reduction rules for the PPA evaluation

1. Any $M \times a \Rightarrow (\sum k_i 2^i)a$. For example $5a \Rightarrow (a, 0, a)$ over three columns.
2. Algebraic reductions. For example, $2a - a \Rightarrow a$.
3. Boolean reductions:
   - $a - ab = a(1-b) \Rightarrow a\bar{b}$.
   - $a + b - ab = a + \bar{a}b \Rightarrow a\,OR\;b$.
   - $a + b - 2ab \Rightarrow a \oplus b$.

The rest of the steps

After applying the reduction rules,

- Compensate for any approximation errors to improve the accuracy.
- Complement the negative elements and subtract one (remember that $\bar{a} = 1 - a$).
- Reduce all the constants.

q_0	q_1	q_2	q_3	q_4	$\cdots$
				1	
				$\bar{b}_5$	$\cdots$
			1	$\bar{b}_2b_3b_4$	
	$\bar{b}_2$		$\bar{b}_2\bar{b}_4$	$\bar{b}_4$	
	1	$(b_2 OR\; \bar{b}_3)$	$\bar{b}_3$	$(\bar{b}_2 OR\; \bar{b}_3)$	

Reducing all of this, we get a *non-redundant* representation of the quotient.

Other functions using the PPA

If the function is expressed as a polynomial, we represent the coefficients and the parameter using their bits and expand symbolically. For example, let us evaluate $P(h) = c_0 + c_1 h + c_2 h^2$ where

$$\begin{aligned} h &= h_1 2^{-5} + h_2 2^{-6} \\ c_0 &= c_{00} + c_{01} 2^{-1} \\ c_1 &= c_{10} + c_{11} 2^{-1} \\ c_2 &= c_{20} + c_{21} 2^{-1} \end{aligned}$$

We expand $P(h)$ symbolically and group the terms:

$$\begin{aligned} P(h) &= c_{00} + c_{01}2^{-1} + c_{10}h_1 2^{-5} + (c_{10}h_2 + c_{11}h_1)2^{-6} \\ &+ c_{11}h_2 2^{-7} + (c_{20}h_1 + c_{20}h_1h_2)2^{-10} + (c_{21}h_1 + c_{21}h_1h_2)2^{-11} \\ &+ c_{20}h_2 2^{-12} + c_{21}h_2 2^{-13} \end{aligned}$$

then write them in the form of a partial product array:

c_{00}	c_{01}	0	0	0	$c_{10}h_1$	$c_{10}h_2$	$c_{11}h_2$	0	0	$c_{20}h_1$	$c_{21}h_1$	$c_{20}h_2$	$c_{21}h_2$
						$c_{11}h_1$				$c_{20}h_1h_2$	$c_{21}h_1h_2$		

Conclusion on the use of PPA

- The PPA of a multiplier is modified by adding a multiplexer and some logic gates to generate the required bit patterns for the function.
- That minimal hardware allows us to compute many functions at the speed of a multiplication.
- Almost all the elementary functions can be computed (usually to within 12–20 bits of precision).
- The reconfiguration of the multiplier may be done in less than a clock cycle.

Summary of elementary functions

- We presented a general classification of how to implement them.
- What is "best" depends on the goal of the specific unit.
- It is possible to have hardware intensive and extremely fast evaluation. On the opposite spectrum, it is possible to delegate the computations to software.