Introduction to elementary function algorithms

Jean-Michel Muller

June 2004

$\boxed{\text{Elementary functions}}$

- sine, cosine, arctangent, exponential, logarithm. . .
- most algorithms for approximating these functions give a result in a small domain only $\Rightarrow$ 3 steps : range reduction, approximation, computation of final result.

Desirable properties

– speed ;
– accuracy ;
– reasonable amount of resource (ROM/RAM, silicon area used by a dedicated hardware. . .) ;
– preservation of important mathematical properties such as *monotonicity*, and *symmetry*. Monotonicity failures can cause problems in evaluating divided differences ;
– range limits getting a sine larger than 1 may lead to unpleasant surprises, for instance, when computing

$$\sqrt{1 - \sin^2 x}.$$

Computing System	$\sin x$
Exact result	-0.852200849767188801772 7...
HP 48 GX	-0.852200849762
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
Silicon Graphics Indy	0.87402806...
SPARC	-0.8522008497671887 9
IBM RS/6000 AIX 3005	-0.852200849...
IBM 3090/600S-VF AIX 370	0.0
DECstation 3100	NaN
TI 89	Trig. arg. too large

TAB. 1 – $\sin(x)$ *for* $x = 10^{22}$

Polynomial approximations to functions

– assume we have FP addition and multiplication available in hardware ;
– $\pm$, $\times$, comparisons $\rightarrow$ the functions of one variable we can compute are piecewise polynomials .
– it is natural to try to approximate functions by polynomials.

Rational functions : sometimes interesting, but in most cases the delay of FP division and the fact that it is not pipelined makes evaluation of rational functions rather costly.

A few notations

- $\mathcal{P}_n$: set of the polynomials of degree less than or equal to n ;
- we want to approximate a function f by an element p^* of $\mathcal{P}_n$ on an interval $[a, b]$;
- there is much better do do than using Taylor approximations ;
- done by minimizing a “distance” $||p^* - f||$;
- minimax approximations : the distance is

$$||p^* - f||_\infty = \max_{a \le x \le b} |p^*(x) - f(x)|.$$

In 1885, Weierstrass proved that a continuous function can be approximated as accurately as desired by a polynomial.

Theorem 1 (Weierstrass, 1885) *Let f be a continuous function. For any $\epsilon > 0$ there exists a polynomial p such that $||p - f||_\infty \leq \epsilon$.*

Another theorem, due to Chebyshev, gives a characterization of the minimax approximations to a function.

Theorem 2 (Chebyshev) *p^* is the minimax degree-n approximation to f on $[a, b]$ if and only if there exist at least $n+2$ values*

$$a \leq x_0 < x_1 < x_2 < \ldots < x_{n+1} \leq b$$

such that :

$$p^*(x_i) - f(x_i) = (-1)^i \left[p^*(x_0) - f(x_0)\right] = \pm||f - p^*||_\infty.$$

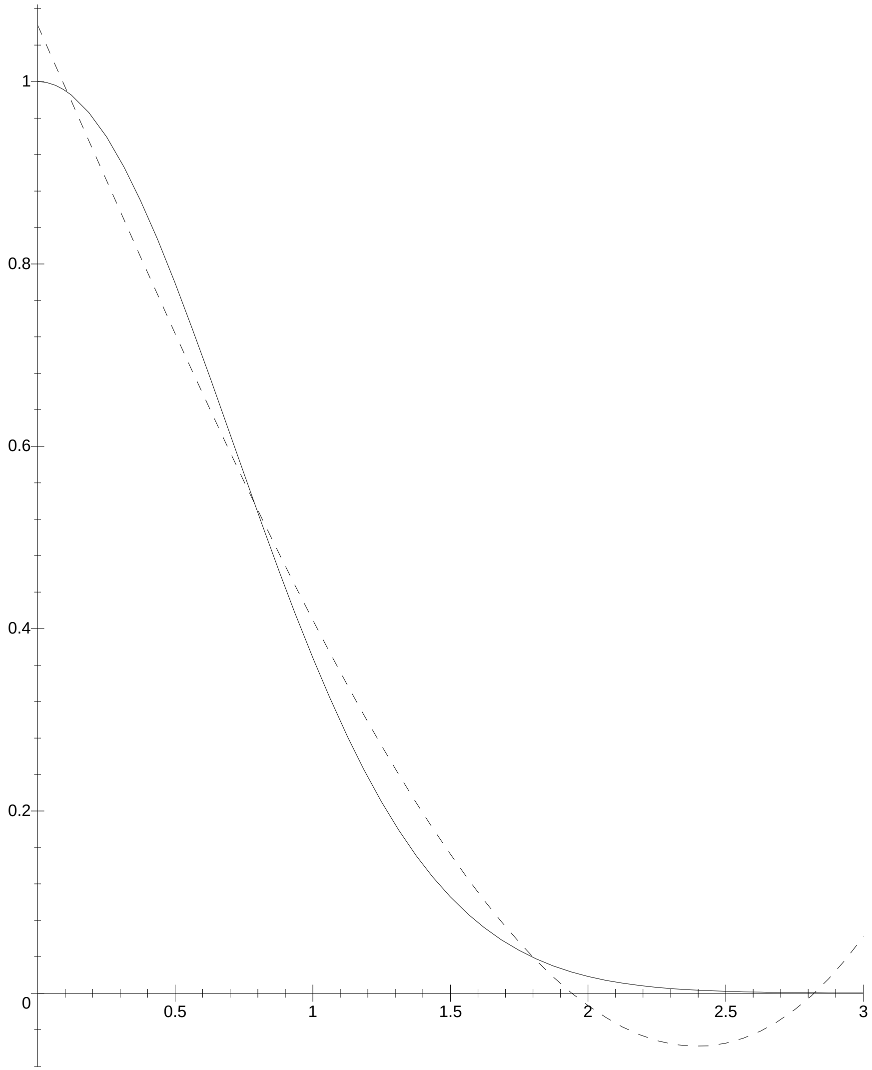

FIG. 1 – $\exp(-x^2)$ *and its degree-*3 *minimax approx. on* $[0, 3]$*.*

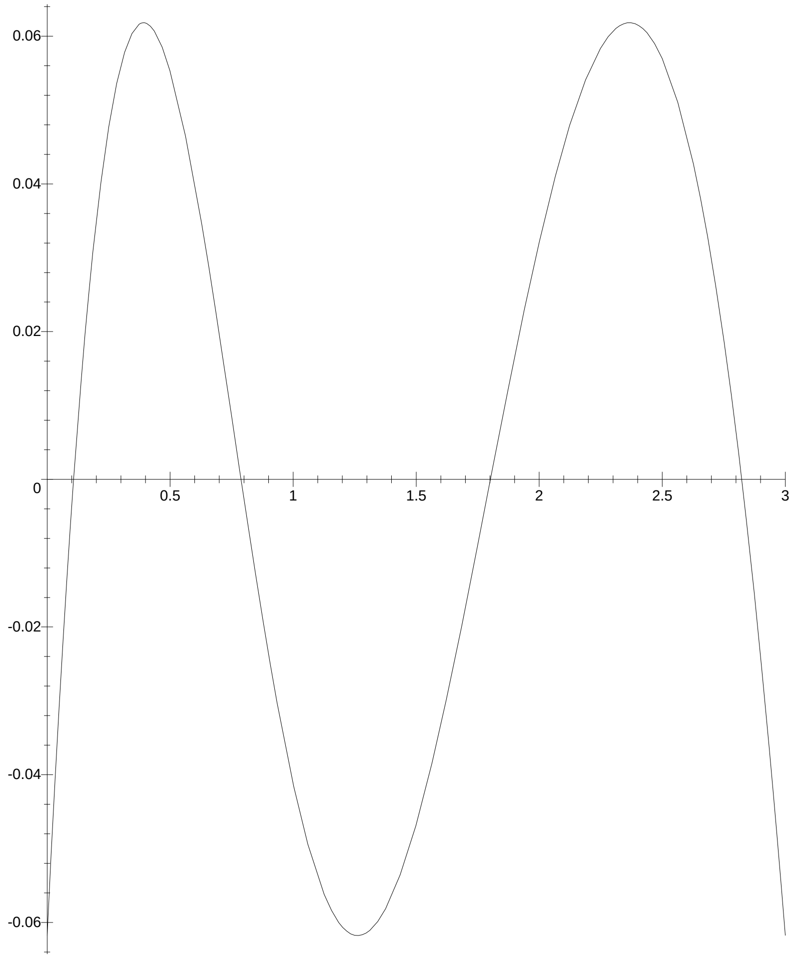

FIG. 2 – *Difference between* $\exp(-x^2)$ *and its degree-*3 *minimax approximation on* $[0, 3]$.

Getting the approximations

- Remes algorithm (1934) : minimax approximation to a function in a given interval ;
- difficult to predict accuracy vs degree : very function-dependent ;
- implemented in the numapprox package of Maple ;
- best “truncated” polynomial approximations : algorithm suggested by Brisebarre, Muller and Tisserand (2004).

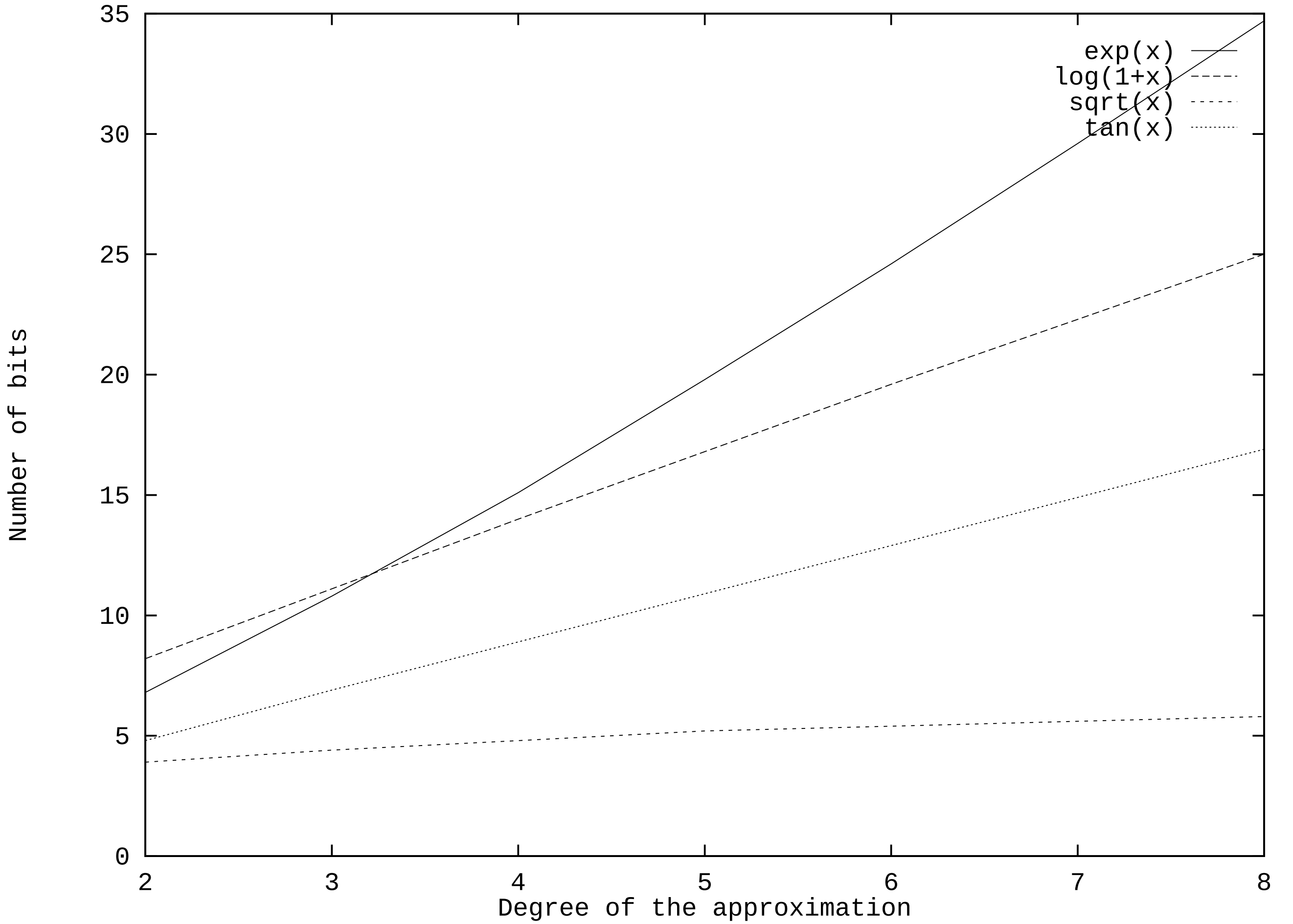

FIG. 3 – *Number of significant bits (obtained as* $-\log_2(error)$*) of the minimax polynomial approximations to various functions on* $[0, 1]$*.*

Introduction to shift and add algorithms

- algorithms that do not use $\times$ or $\div$;
- the most famous : CORDIC (Volder 59, Walther 71) : first large-scale implementations HP 35, Intel 8087, Motorola 68881 ;
- introduction to theoretical bakery $\Rightarrow$ link with elementary functions ;
- simple algorithms ;
- more efficient algorithms using redundancy.

Let us weigh a loaf of bread

Pair of scales, weights w_0, w_1, w_2,. . . that satisfy :

– $\forall i, w_i > 0$;

– the sequence w_i is decreasing and $\sum w_i < +\infty$;

– $\forall i, w_i \leq \sum_{k=i+1}^{\infty} w_k$.

first exercise the weights are either unused or put in the pan that does not contain the bread.

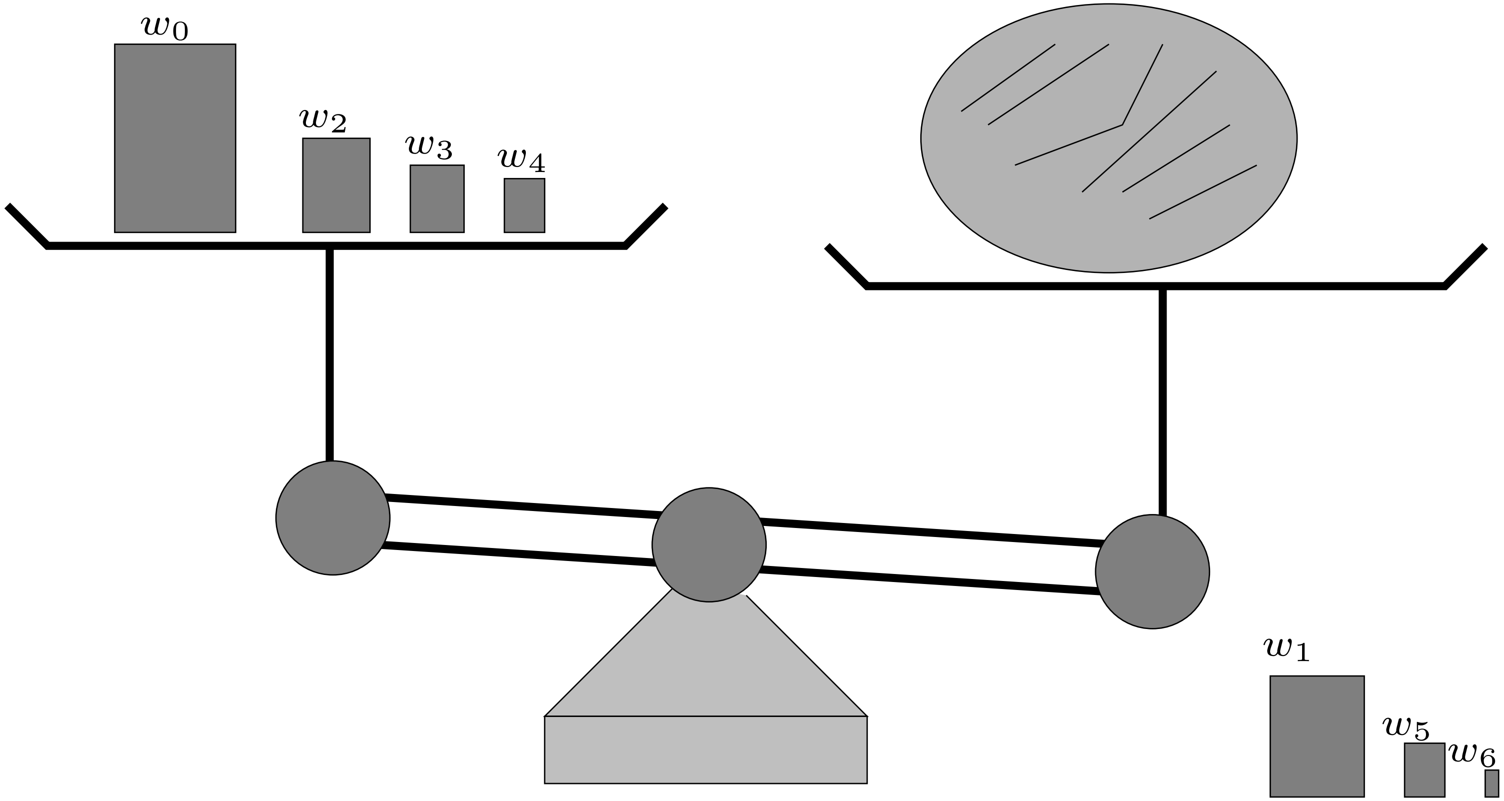

FIG. 4 – Restoring decomposition

Theorem 3 (Restoring decomposition) *Let* (w_n) *be a decreasing sequence of real numbers* > 0 *such that* $\sum_{i=0}^{\infty} w_i < \infty$*. If*

$$\forall n, w_n \leq \sum_{k=n+1}^{\infty} w_k \tag{1}$$

then $\forall t \in [0, \sum_{k=0}^{\infty} w_k]$*, the sequences* (t_n) *and* (d_n) *defined as*

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n w_n \\ d_n &= \begin{cases} 1 & \textit{if } \ t_n + w_n \leq t \\ 0 & \textit{otherwise} \end{cases} \end{aligned} \tag{2}$$

satisfy $t = \sum_{n=0}^{\infty} d_n w_n = \lim_{n \to \infty} t_n$.

Second exercise

Same pair of scales $\&$ weights, but now we must use all weights. However, they can be put in both pans.

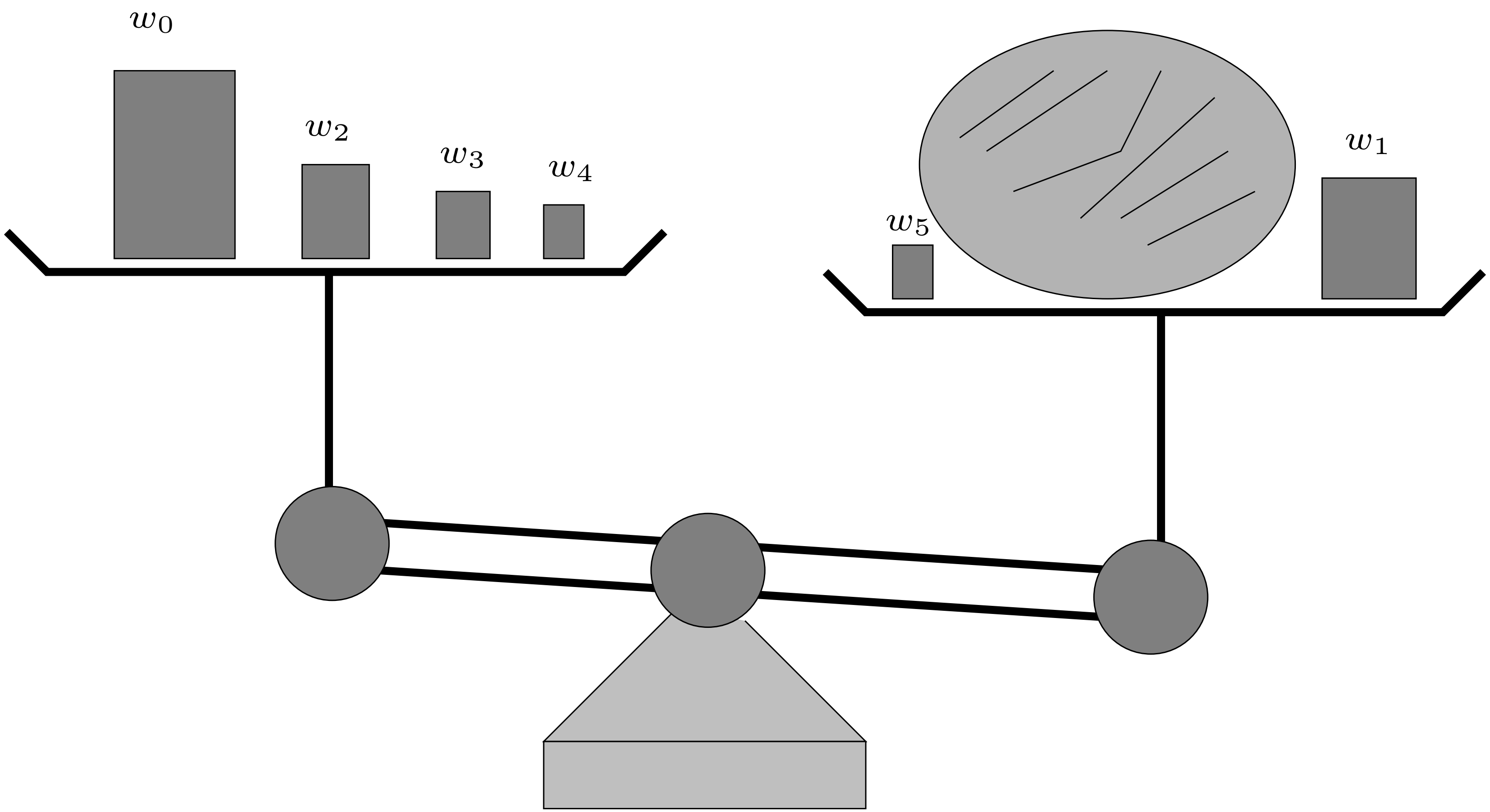

FIG. 5 – Non-restoring algorithm

Another greedy algorithm :

Theorem 4 (Non-restoring algorithm) *Let* (w_n) *be a sequence satisfying the conditions of Theorem 3.* $\forall t \in [-\sum_{k=0}^{\infty} w_k, \sum_{k=0}^{\infty} w_k]$, *the sequences* (t_n) *and* (d_n) *defined as*

$$\begin{aligned} t_0 &= 0 \\ t_{n+1} &= t_n + d_n w_n \\ d_n &= \begin{cases} 1 & \textit{if } t_n \le t \\ -1 & \textit{otherwise} \end{cases} \end{aligned} \tag{3}$$

satisfy $t = \sum_{n=0}^{\infty} d_n w_n = \lim_{n\to\infty} t_n$.

Theorem 5 *The sequences* $\ln(1+2^{-n})$ *and* $\arctan 2^{-n}$ *satisfy the conditions of Theorems 3 and 4*

From theoretical bakery to the exponential function

$w_n = \ln(1 + 2^{-n})$. Let $t \in [0, \sum_{k=0}^{\infty} w_k] = [0, 1.56 \cdots]$.

$$t_0 = 0 \qquad\qquad d_n = \begin{cases} 1 & \text{if } t_n + \ln\left(1 + 2^{-n}\right) \leq t \\ 0 & \text{otherwise} \end{cases}$$
$$t_{n+1} = t_n + d_n \ln\left(1 + 2^{-n}\right)$$

satisfy $t = \sum_{n=0}^{\infty} d_n \ln(1 + 2^{-n}) = \lim_{n \to \infty} t_n$.

Let E_n be such that $\forall n$, $E_n = e^{t_n}$

- $t_0 = 0 \Rightarrow E_0 = 1$.
- when $t_{n+1} \neq t_n$ (i.e., when $d_n = 1$), $t_{n+1} = t_n + \ln(1 + 2^{-n})$.
  $E_n = e^{t_n} \Rightarrow E_n$ multiplied by $\exp \ln(1 + 2^{-n}) = (1 + 2^{-n})$.

Since $t_n \to t$, $E_n \to e^t$.

Algorithm 1 (expo-1, inputs : t, N (nb of steps), output : E_N)

$t_0 = 0 \qquad E_0 = 1;$ **build t_n and E_n as follows**

$$\begin{array}{rcl} t_{n+1} & = & t_n + \ln\left(1 + d_n 2^{-n}\right) \\ E_{n+1} & = & E_n \left(1 + d_n 2^{-n}\right) = E_n + d_n E_n 2^{-n} \\ d_n & = & \begin{cases} 1 & \textit{if } t_n + \ln\left(1 + 2^{-n}\right) \leq t \\ 0 & \textit{otherwise.} \end{cases} \end{array} \tag{4}$$

This algorithm : only $+$, and $\times$ by powers of 2 (mere shifts). Constants $\ln\left(1 + 2^{-n}\right)$ precomputed and stored (n bits of accuracy $\Rightarrow \approx n$ constants).

Replace $\ln\left(1 + 2^{-n}\right)$ by $\log_a\left(1 + 2^{-n}\right) \longrightarrow$ algorithm for a^t.

From exponentials to logarithms

We want to compute $\ell = \ln(x)$. First assume ℓ is known (!!!) and compute its exponential (yes, I know it is x) using :

$$
\begin{array}{rcl}
t_0 = 0 & & E_1 = 1 \\
t_{n+1} & = & t_n + d_n \ln\left(1 + 2^{-n}\right) \\
E_{n+1} & = & E_n + d_n E_n 2^{-n}
\end{array}
\tag{5}
$$

with $d_n = \begin{cases} 1 & \text{if } t_n + \ln\left(1+2^{-n}\right) \le \ell \\ 0 & \text{otherwise.} \end{cases}$ $t_n \to \ell$, $E_n \to e^{\ell} = x$.

Cannot be used since needs ℓ... From «$E_n = \exp\left(t_n\right)$» that comparison can be replaced by $d_n = \begin{cases} 1 & \text{if } E_n \times \left(1+2^{-n}\right) \le x \\ 0 & \text{otherwise.} \end{cases}$

Same results, without requiring the knowledge of ℓ.

Algorithm 2 (logarithm-1)

– **inputs :** x, n, *with* $1 \leq x \leq \prod_{i=0}^{\infty}\left(1+2^{-i}\right) \approx 4.76$;

– **output :** $t_n \approx \ln x$.

$t_0 = 0, E_0 = 1$. **Build t_i and E_i as follows**

$$\begin{aligned}
t_{i+1} &= t_i + \ln\left(1 + d_i 2^{-i}\right) \\
E_{i+1} &= E_i\left(1 + d_i 2^{-i}\right) = E_i + d_i E_i 2^{-i} \\
d_i &= \begin{cases} 1 & \text{if } E_i + E_i 2^{-i} \leq x \\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{6}$$

Replace $\ln\left(1+2^{-n}\right)$ by $\log_a\left(1+2^{-n}\right) \rightarrow$ alg. for $\log_a$.

Trigonometric functions

- Non restoring decomposition (weights on both pans)
- Sequence $w_n = \arctan 2^{-n}$
- decomposition $\Rightarrow \theta = \sum_{k=0}^{\infty} d_k w_k, \quad d_k = \pm 1, \quad w_k = \arctan 2^{-k}.$

Rotation mode of CORDIC : perform a rotation of angle θ as a sequence of "micro-rotations" of angles $d_n w_n$. Start from (x_0, y_0). Get (x_{n+1}, y_{n+1}) from (x_n, y_n) by performing rotation of angle $d_n w_n$. Gives

$$\begin{array}{lcl} t_0 & = & 0 \\ t_{n+1} & = & t_n + d_n w_n \end{array} \qquad d_n = \begin{cases} 1 & \text{if } t_n \leq \theta \\ -1 & \text{otherwise;} \end{cases} \tag{7}$$

nth rotation

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} \cos(d_n w_n) & -\sin(d_n w_n) \\ \sin(d_n w_n) & \cos(d_n w_n) \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \tag{8}$$

$d_n = \pm 1 \Rightarrow \cos(d_n w_n) = \cos(w_n)$ and $\sin(d_n w_n) = d_n \sin(w_n)$.
Moreover, $\tan w_n = 2^{-n}$. Therefore :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \cos(w_n) \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}. \tag{9}$$

Radix-2 arithmetic $\rightarrow$ all operations are very simple, with one serious exception : product by $\cos(w_n) = 1/\sqrt{1 + 2^{-2n}}$.

Just ignore the problem and compute :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -d_n 2^{-n} \\ d_n 2^{-n} & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} \tag{10}$$

basic **CORDIC** iteration. Instead of a *rotation*, *similarity* of angle w_n & factor $1/\cos w_n = \sqrt{1+2^{-2n}}$.

Last modification $z_n = \theta - t_n$. Gives $z_0 = \theta$,

$$\left\{ \begin{array}{rcl} x_{n+1} & = & x_n - d_n y_n 2^{-n} \\ y_{n+1} & = & y_n + d_n x_n 2^{-n} \\ z_{n+1} & = & z_n - d_n \arctan 2^{-n}. \end{array} \right. \tag{11}$$

with $d_n = 1$ if $z_n \geq 0$, -1 otherwise.

$(x_n, y_n) \longrightarrow$ result of similarity of angle θ $\&$ factor $K = 1.646760258121 \cdots = \prod \sqrt{1 + 2^{-2i}}$ applied to (x_0, y_0).

$$\lim_{n\to\infty} \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = K \times \begin{pmatrix} x_0 \cos z_0 - y_0 \sin z_0 \\ x_0 \sin z_0 + y_0 \cos z_0 \\ 0 \end{pmatrix} \tag{12}$$

For instance, $x_0 = 1/K$ and $y_0 = 0$ give $x_n \rightarrow \cos(\theta)$ and $y_n \rightarrow \sin(\theta)$.

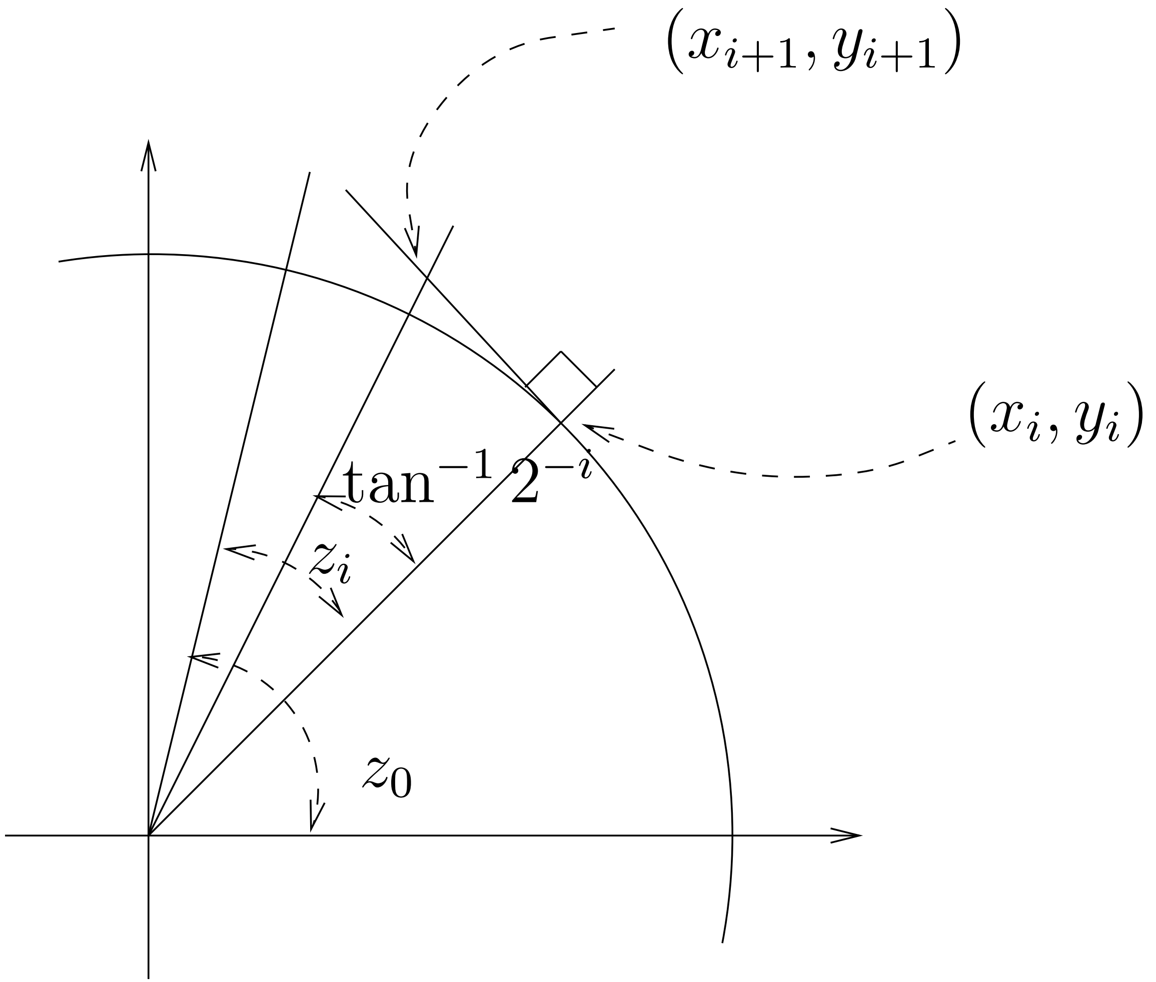

FIG. 6 – *One iteration of CORDIC.*

Generalized CORDIC

Due to John Walther, from HP. Implemented on HP 35, then Intel 8087. Basic iteration :

$$\left\{ \begin{array}{lcl} x_{n+1} & = & x_n - m d_n y_n 2^{-\sigma(n)} \\ y_{n+1} & = & y_n + d_n x_n 2^{-\sigma(n)} \\ z_{n+1} & = & z_n - d_n w_{\sigma(n)}, \end{array} \right. \tag{13}$$

$m = 1$ gives previous algorithm.

m	w_k	$d_n = \text{sign} z_n$ (Rotation Mode)	$d_n = -\text{sign} y_n$ (Vectoring Mode)
1	$\arctan 2^{-k}$	$x_n \to K\,(x_0 \cos z_0 - y_0 \sin z_0)$ $y_n \to K\,(y_0 \cos z_0 + x_0 \sin z_0)$ $z_n \to 0$	$x_n \to K\sqrt{x_0^2 + y_0^2}$ $y_n \to 0$ $z_n \to z_0 - \arctan \frac{y_0}{x_0}$
0	2^{-k}	$x_n \to x_0$ $y_n \to y_0 + x_0 z_0$ $z_n \to 0$	$x_n \to x_0$ $y_n \to 0$ $z_n \to z_0 - \frac{y_0}{x_0}$
-1	$\tanh^{-1} 2^{-k}$	$x_n \to K'\,(x_1 \cosh z_1 + y_1 \sinh z_1)$ $y_n \to K'\,(y_1 \cosh z_1 + x_1 \sinh z_1)$ $z_n \to 0$	$x_n \to K'\sqrt{x_1^2 - y_1^2}$ $y_n \to 0$ $z_n \to z_1 - \tanh^{-1} \frac{y_1}{x_1}$

TAB. 2 – *Fonctions computable with CORDIC.*

Trigo ($m = 1$)	$\sigma(n) = n$
Linear ($m = 0$)	$\sigma(n) = n$
Hyperbolic ($m = -1$)	$\sigma(n) = n - k$ where k is largest integer s.t. $3^{k+1} + 2k - 1 \leq 2n$

TAB. 3 – *Value of* $\sigma(n)$

Some references on CORDIC

– Some ideas go back to Briggs (1561-1631) ;
– CORDIC : Volder (1959) ;
– very similar ideas developed by Meggitt (1962) ;
– generalized version : Walther (1971). Implementations : HP35, Intel 8087, Motorola 68881.
– simple algorithms for log and exp : Specker (1965), DeLugish's PhD (1970) ;
– redundant versions : Takagi, Ercegovac & Lang, Lee & Lang, Duprat & Muller ;
– Special Issue on CORDIC in the *Journal of VLSI Signal Processing* (june 2000).

A few words on correct rounding

- In general, the result of an arithmetic operation on two FP numbers is not exactly representable in the same FP format $\Rightarrow$ must be rounded
- In a FP system that follows IEEE-754, the user can choose an *active rounding mode* from : rounding towards $-\infty$, $+\infty$, 0 and to the nearest even.
- The system should behave as if the results of $+$, $-$, $\div$, $\times$ and $\sqrt{x}$ were first computed exactly, and then rounded accordingly to the active rounding mode.
- Operations that satisfy this property are called correctly rounded (or exactly rounded).

What about the elementary functions ?

- No such requirement for the elementary functions
- Requiring correctly rounded results would not only improve the accuracy of computations : it would help to make numerical software more portable, help implementing interval arithmetic, and facilitate the preservation of properties such as monotonicity, symmetry, . . .

The Table Maker's Dilemma

- Let f be an elementary function and x a FP number.
- Unless x is a very special case – e.g., $\sin(0)$ –, $y = f(x)$ cannot be exactly computed. The only thing we can do is to compute an *approximation* y^* to y.
- Correctly rounded functions : we must know what the accuracy of this approximation should be to make sure that rounding y^* is always equivalent to rounding y.

The Table Maker's Dilemma (cont.)

– y^* and known bounds on the approximation error $\Rightarrow y$ belongs to some interval Y.

– **breakpoint :** a value z where the rounding changes :

$$t_1 < z < t_2 \Rightarrow \diamond(t_1) < \diamond(t_2)$$

where $\diamond$ is the rounding function ;

– "directed" rounding modes : the breakpoints are the FP numbers ;

– rounding to the nearest mode : they are the exact middle of two consecutive FP numbers.

When does the problem occur ?

If Y contains a breakpoint, we cannot provide $\diamond(y)$: computation must be carried again with larger accuracy. Two solutions :

– iteratively increase accuracy of approximation, until Y no longer contains a breakpoint. And yet, how many iterations will be necessary ?

– compute, once and in advance, the smallest nonzero mantissa distance between the image $f(x)$ of a FP number x and a breakpoint $\Rightarrow$ accuracy with which f must be approximated to make sure that rounding the approximation is equivalent to rounding the exact result.

Example

Worst case for natural logarithm in full double precision range :

$$x = 1.011000101010100010000110000100110110001010\;0110110110 \times 2^{678}$$

whose logarithm is

$$\log x = \overbrace{111010110.01000111100111110101\cdots110001}^{53 \text{ bits}}\;\underbrace{000000000000000000\cdots000000000000000}_{65 \text{ zeroes}}1110...$$

This is a "difficult case" in a directed rounding mode since it is very near a FP number.

TAB. 4 – *Worst cases for the exponential function in the full range.*

Interval	worst case (binary)
$[-\infty, -2^{-30}]$	$\exp(-1.1110110100110001100011101111101101100010011111101010 \times 2^{-27})$ $= 1.111111111111111111111111100 \cdots 0111000100 \;\; 1 \;\; 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.0000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.11111111111111 \cdots 11111111111111100 \;\; 0 \;\; 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.1111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.0000000000000000000000000000000000000000000000000000 \;\; 1 \;\; 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.0111111111111110011111111111111101110000000000100100 \times 2^{-32})$ $= 1.0000000000000000000000000000001011111111111111101000 \;\; 0 \;\; 0^{57}1101\ldots$
	$\exp(1.1000000000000001011111111111111101101111111111011100 \times 2^{-32})$ $= 1.0000000000000000000000000000001100000000000000010111 \;\; 1 \;\; 1^{57}0010\ldots$
	$\exp(1.1001111010011100101110111111110101100000100000001011 \times 2^{-31})$ $= 1.0000000000000000000000000000001100111101001110010111 \;\; 1 \;\; 0^{57}1010\ldots$
	$\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \;\; 0 \;\; 0^{57}1000\ldots$

Property 1 (Computation of exponentials) *Let y be the exponential of a double-precision number x. Let y^* be an approximation to y such that the mantissa distance between y and y^* is bounded by ϵ.*

– *for $|x| \geq 2^{-30}$, if $\epsilon \leq 2^{-53-59} = 2^{-112}$ then for any of the 4 rounding modes, rounding y^* is equivalent to rounding y ;*

– *for $|x| < 2^{-30}$, if $\epsilon \leq 2^{-53-104} = 2^{-157}$ then rounding y^* is equivalent to rounding y.*