TSTE18 Digital Arithmetic
Seminar 11

Oscar Gustafsson

Approximation of elementary functions

- Errors
- Range reduction
- Tables
- (Piecewise) polynomial approximations
- Bi-/multi-partite tables
- **CORDIC**
- **Convergence-based approximation**

CORDIC

- The coordinate rotation digital computer (CORDIC) algorithm is a recursive algorithm to calculate elementary functions such as the trigonometric and hyperbolic (and their inverses) functions as well as magnitude and phase of complex vectors
- Consider a rotation of a complex number $X+jY$ by an angle θ

$$\begin{bmatrix} X_r \\ Y_r \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}. \tag{1}$$

- This angle can be decomposed into several subsequent rotations to obtain programmability

$$\theta = \sum_{k=0}^{\infty} d_k w_k \tag{2}$$

where in general w_k is a partial angle and d_k is used to control the rotation direction

CORDIC

- Now we get

$$\begin{bmatrix} X_r \\ Y_r \end{bmatrix} = \begin{bmatrix} \cos\left(\sum_{k=0}^{\infty} d_k w_k\right) & -\sin\left(\sum_{k=0}^{\infty} d_k w_k\right) \\ \sin\left(\sum_{k=0}^{\infty} d_k w_k\right) & \cos\left(\sum_{k=0}^{\infty} d_k w_k\right) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

$$\begin{bmatrix} \cos(d_0 w_0) & -\sin(d_0 w_0) \\ \sin(d_0 w_0) & \cos(d_0 w_0) \end{bmatrix} \cdots \begin{bmatrix} \cos(d_\infty w_\infty) & -\sin(d_\infty w_\infty) \\ \sin(d_\infty w_\infty) & \cos(d_\infty w_\infty) \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

- Design choices:
  - Make representation easy
  - Make rotation easy

CORDIC

- To simplify the representation we can use e.g.

$$\theta = \sum_{k=-1}^{B} b_k 2^{-k} \tag{3}$$

or

$$\theta = \sum_{k=0}^{B} b_k \pi 2^{-k} \tag{4}$$

- While this will lead to a simple mapping between the input and the rotation value, the rotations are still general

CORDIC

- To simplify the rotations we can first note

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \cos(\theta) \begin{bmatrix} 1 & -\frac{\sin(\theta)}{\cos(\theta)} \\ \frac{\sin(\theta)}{\cos(\theta)} & 1 \end{bmatrix}$$

$$= \cos(\theta) \begin{bmatrix} 1 & -\tan(\theta) \\ \tan(\theta) & 1 \end{bmatrix}$$

- This can of course be used for the previous representation, but still will require general rotations
- Now select angles such as $\tan(\theta)$ has a simple value

CORDIC

- In the original CORDIC algorithm the angles were selected such as

$$w_k = \arctan(2^{-k}) \tag{5}$$

- In this way the multiplication is a simple shift leading to a rotation of

$$\begin{bmatrix} 1 & -d_k 2^{-k} \\ d_k 2^{-k} & 1 \end{bmatrix} \tag{6}$$

when the scaling term $\cos(\theta)$ is neglected

- The problem now is to recode the input angle to determine the value of d_k

CORDIC

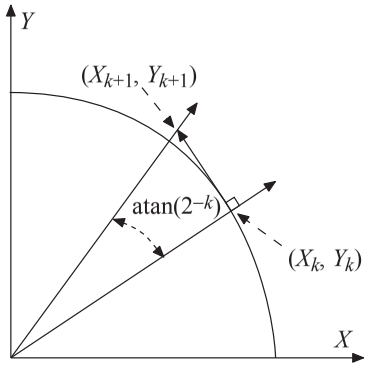
- In the original CORDIC $d_k = \pm 1$ so a rotation is always performed, the question is just in which direction
- Solution: Compare the angle rotated so far with the required angle and determine the rotation direction
- Introduce a third variable, Z_k, representing the angle remaining to be rotated
- Initialize $Z_0 = \theta$ and update as $Z_{k+1} = Z_k + d_k w_k$ where w_k is stored in a memory and

$$d_k = \begin{cases} 1 & Z_k \geq 0 \\ -1 & Z_k < 0 \end{cases} \tag{7}$$

- Hence, the variable Z_k will go towards zero on convergence

CORDIC

- The rotation does not have a correct scaling as the $\cos(\theta)$ term is neglected
- The corresponding operation is called a similarity



CORDIC

- In each iteration the gain is $\sqrt{1^2 + (2^{-k})^2}$
- The total gain is then

$$G(k) = \prod_{i=0}^{k} \sqrt{1 + 2^{-2i}} \tag{8}$$

  where for $k \to \infty$, $G \approx 1.6468$
- It is possible to compensate for this gain by multiplying both the real and imagniary parts with
  - $G(\infty)$ ($G(K)$ using K iterations) after all iterations
  - $1 - a_k 2^{-k}$ for a suitable sequence of $a_k \in \{-1, 0, 1\}$ after each iteration

CORDIC

- This type of CORDIC operation is performed in the so called rotation mode
- Using the rotation mode it is possible to compute $\sin(\theta)$ and $\cos(\theta)$ by initializing $X = 1$ and $Y = 0$ (or, better, $X = 1/G(\infty)$)
- It is also possible to rotate an arbitrary vector $X + jY$ and angle θ
- Furthermore, it is possible to determine θ of an angle by selecting the rotations such that the imaginary part Y_k converges to zero
- Then the result in Z_k will be the angle of $X + jY$
- This is called vectoring mode

Generalized CORDIC

- There are many more operations that can be perfomed using the concept of rotations
- A generalized CORDIC rotation can be written as

$$\begin{aligned} X_{k+1} &= X_k - m d_k Y_k 2^{-\sigma(k)} \\ Y_{k+1} &= Y_k + d_k X_k 2^{-\sigma(k)} \\ Z_{k+1} &= Z_k - d_k w_{\sigma(k)} \end{aligned} \tag{9}$$

  where an appropriate choice of m, d_k, w_k, and $\sigma(k)$ several different functions can be obtained

Generalized CORDIC

- Three different classes of CORDIC types: circular, linear and hyperbolic
- Each can be operated in rotation or vectoring mode

Type	Parameters	Rotation mode, $d_k = \mathrm{sign}(z_k)$
Circular	$m = 1$ $w_k = \arctan(2^{-k})$ $\sigma(k) = k$	$X_k \to G_k(X_0 \cos(Z_0) - Y_0 \sin(Z_0))$ $Y_k \to G_k(Y_0 \cos(Z_0) + X_0 \sin(Z_0))$ $Z_k \to 0$
Linear	$m = 0$ $w_k = 2^{-k}$ $\sigma(k) = k$	$X_k \to X_0$ $Y_k \to Y_0 + X_0 Z_0$ $Z_k \to 0$
Hyperbolic	$m = -1$ $w_k = \tanh^{-1}(2^{-k})$ $\sigma(k) = k - h_k$	$X_k \to \hat{G}_k(X_1 \cosh(Z_1) - Y_1 \sin(Z_1))$ $Y_k \to \hat{G}_k(Y_1 \cosh(Z_1) + X_1 \sinh(Z_1))$ $Z_k \to 0$

Generalized CORDIC

Type	Parameters	Vectoring mode, $d_k = -\mathrm{sign}(y_k)$
Circular	$m = 1$ $w_k = \arctan(2^{-k})$ $\sigma(k) = k$	$X_n \to G_k\sqrt{X_0^2 + Y_0^2}$ $Y_n \to 0$ $Z_n \to Z_0 + \arctan(\frac{Y_0}{X_0})$
Linear	$m = 0$ $w_k = 2^{-k}$ $\sigma(k) = k$	$X_n \to X_0$ $Y_n \to 0$ $Z_n \to Z_0 + \frac{Y_0}{X_0}$
Hyperbolic	$m = -1$ $w_k = \tanh^{-1}(2^{-k})$ $\sigma(k) = k - h_k$	$X_n \to \hat{G}_k\sqrt{X_1^2 - Y_1^2}$ $Y_n \to 0$ $Z_n \to Z_1 + \tanh^{-1}(\frac{Y_1}{X_1})$

CORDIC modification/improvements

- Allow zero rotations, $d_k \in \{-1, 0, 1\}$
  - New selection rule, rather straightforward
  - Scale factor will be angle dependent
- Unfolded algorithms
  - Naturally, it is possible to unfold the algorithm computing several rotations in one iteration
  - Sign selection will still depend on the previous result
- Higher-radix, i.e., determine several rotations in one
  - More complex selection rules
- Redundant representations
  - Make selection without exactly knowing the sign

Normalization/convergence-based approximation

- Consider an iterative algorithm

$$Y_k = Y_{k-1} P_k \tag{10}$$

- If $Y_0 = X$ and P_k is a sequence such that $Y_\infty \to 1$ we get

$$Y_\infty = \prod_{k=1}^{\infty} P_k X \to 1 \tag{11}$$

$$\frac{1}{X} = \prod_{k=0}^{\infty} P_k \approx \prod_{k=0}^{N} P_k \tag{12}$$

Normalization/convergence-based approximation

- When discussing reciprocals we saw a scheme giving quadratic convergence but rather complicated computations
- The simpler scheme $P_k = 1 + s_k 2^{-k}$ gives a linear convergence and

$$\frac{1}{X} \approx \prod_{k=0}^{N} P_k = \prod_{k=0}^{N} (1 + s_k 2^{-k}) \tag{13}$$

- Example: $X = 1.5 = P_0$, initialize remainder $R_k = 1$ and let $s_k \in \{-1, 1\}$

k	s_k	$Y_k = Y_{k-1} P_k$	$R_k = R_{k-1} P_k$
0		1.5	1
1	-1	$1.5(1 - 2^{-1}) = 0.75$	$1(1 - 2^{-1}) = 0.5$
2	1	$0.75(1 + 2^{-2}) = 0.9375$	$0.5(1 + 2^{-2}) = 0.625$
3	1	$0.9375(1 + 2^{-3}) \approx 1.0547$	$0.625(1 + 2^{-3}) \approx 0.7031$
4	-1	$1.0547(1 - 2^{-4}) \approx 0.9888$	$0.7031(1 - 2^{-4}) \approx 0.6592$
5	1	$0.9888(1 + 2^{-5}) \approx 1.0197$	$0.6592(1 + 2^{-5}) \approx 0.6798$
6	-1	$1.0197(1 - 2^{-6}) \approx 1.0037$	$0.6798(1 - 2^{-6}) \approx 0.6692$

Normalization/convergence-based approximation

- The problem is then to define the selection of s_k to guarantee convergence
- It can be shown that the following selection rule for s_j will work in the radix-2 case

$$s_k = \begin{cases} 1, & 0 \leq W_k \\ 0, & -0.5 \leq W_k < 0 \\ -1, & W_k < -0.5 \end{cases}$$

- Example: $X = 1.5 = 1 - W_0$, initialize remainder $R_k = 1$

k	s_k	$W_k = 2W_{k-1} - s_k + s_k W_{k-1} 2^{-(k-1)}$	$R_k = R_{k-1} P_k$
0		-0.5	1
1	0	$2(-0.5) = -1$	$1(1 + 0) = 1$
2	-1	$2(-1) + 1 + 1 \times 2^{-1} = -0.5$	$1(1 - 2^{-2}) = 0.75$
3	0	$2(-0.5) = -1$	$0.75(1 + 0) = 0.75$
4	-1	$2(-1) + 1 + 1 \times 2^{-3} = -0.875$	$0.75(1 - 2^{-4}) \approx 0.7031$
5	-1	$2(-0.875) + 1 + 0.875 \times 2^{-4} \approx -0.6953$	$0.7031(1 - 2^{-5}) \approx 0.6812$
6	-1	$2(-0.6953) + 1 + 0.6953 \times 2^{-5} \approx -0.3689$	$0.6812(1 - 2^{-6}) \approx 0.6705$
7	0	$2(-0.3689) \approx -0.7378$	$0.6705(1 + 0) \approx 0.6705$

Normalization/convergence-based approximation

- The sequence will converge to one while the error gets smaller and smaller
- From a representation point of view it make sense to define a scaled residual as

$$W_k = 2^k (1 - Y_k) \tag{14}$$

  i.e., only the error from one is represented and it is scaled to fall within a suitable range
- This gives a recurrence

$$\begin{aligned} Y_k &= Y_{k-1} P_k && (15) \\ Y_k &= Y_{k-1} \left(1 + s_k 2^{-k}\right) && (16) \\ 1 - 2^{-k} W_k &= \left(1 - 2^{-(k-1)} W_{k-1}\right) \left(1 + s_k 2^{-k}\right) && (17) \\ W_k &= 2W_{k-1} - s_k + s_k W_{k-1} 2^{-(k-1)} && (18) \end{aligned}$$

with $W_0 = 1 - X$

Normalization/convergence-based approximation

- Now consider the computation of $\log X$ (arbitrary logarithm base)
- With

$$\frac{1}{X} \approx \prod_{k=0}^{N} P_k = \prod_{k=0}^{N} (1 + s_k 2^{-k}) \tag{19}$$

  we get

$$\log X \approx -\log \left(\prod_{k=0}^{N} P_k \right) = -\sum_{k=0}^{N} \log P_k = \sum_{k=0}^{N} \log(1 + s_k 2^{-k}) \tag{20}$$

- Hence, it is possible to compute the logarithm of a value summing the values $\log(1 + s_k 2^{-k})$
- s_k must be determined in parallel based on the multiplicative convergence
- $N + 1$ values of $\log(1 + s_k 2^{-k})$ must be stored in a table

Normalization/convergence-based approximation

- Similarly the anti-logarithm/exponent can be computed using additive convergence by finding a sequence of suitable values such that

$$X - \sum_{k=1}^{N} \log B_k \to 0 \tag{21}$$

- Then we obtain, using base-L logarithms

$$L^X \approx \prod_{k=1}^{N} B_k \tag{22}$$

- Again we can restrict the term to be on the form $B_k = 1 + s_k 2^{-k}$
- This means that we can use the same look-up table for the values $\log(1 + s_k 2^{-k})$

Normalization/convergence-based approximation

- The residual recurrence is now

$$W_{k+1} = 2(W_k - 2^k \log(1 + s_k 2^{-k})) \tag{23}$$

with a selection function

$$s_k = \begin{cases} 1 & 0.5 \leq W_k \\ 0 & -0.5 \leq W_k < 0.5 \\ -1 & W_k < -0.5 \end{cases}$$

- For all these algorithms it is possible to use a redundant representation and only determine the two most significant bits
- Note that it is possible to approximation $\log(1 \pm P)$ with $\pm P$ for small P