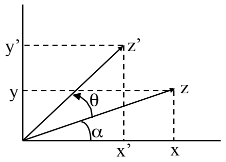
# Cálculos trigonométricos: CORDIC

COordinate Rotation DIgital Computer
Objetivo: Calcular el seno de un ángulo de modo iterativo y sencillo

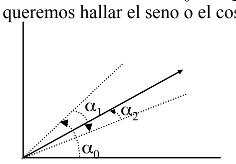



z=(x,y)
x = |z|cos α
y = |z|sen α

Z'=(x',y')
X' = |z'|cos(α+ θ) = |z|(cos α cosθ - sen α senθ) = x cos θ - y sen θ
Si la rotación hubiese sido horaria cambiaría el signo de y sen θ

X' = x cos θ - y sen θ = cos θ (x - y tan θ)
y' = y cos θ + x sen θ = cos θ (y + x tan θ)
Si tan θ es una potencia de 2 => las coordenadas de z' pueden calcularse mediante sumas/restas y desplazamientos.

Partiendo de un vector $z_0$ de argumento $0^0$, hacer rotaciones sucesivas hasta alcanzar el ángulo θ, del que queremos hallar el seno o el coseno.




$\theta = \alpha_0 \pm \alpha_1 \pm \alpha_2 \pm \ldots \pm \alpha_n$
Siendo $\alpha_i = \mathrm{atan}(2^{-i})$

$|Z''| = |Z'| / \cos\theta$

| Arco | tan |
|---|---|
| 45 | $2^0$ |
| 26,56 | $2^{-1}$ |
| 14,03 | $2^{-2}$ |
| 7,12 | $2^{-3}$ |
| ... | |

20

# Cálculos trigonométricos: CORDIC

Proceso iterativo:
Se parte de : $z_0 = (x_0, y_0) = (x_0, 0)$
Al hacer sucesivas rotaciones, llegaremos a vectores $z_{i+1}$ con coordenadas:

$$\left.\begin{aligned} x_{i+1} &= x_i \pm y_i * 2^{-i} \\ y_{i+1} &= y_i \pm x_i * 2^{-i} \end{aligned}\right\} \Rightarrow$$

El cálculo sólo implica sumas/ restas y desplazamientos

Después de n iteraciones se dispondrá de un vector $z_n$ tal que:

$$|z_n| = \frac{1}{\cos\alpha_{n-1}}|z_{n-1}| = \frac{1}{\cos\alpha_{n-1}}\frac{1}{\cos\alpha_{n-2}}|z_{n-2}| = \ldots = \frac{1}{\prod_{i=0}^{n-1}\cos\alpha_i}|z_0|$$

Pero: $$\lim_{n\to\infty}\left(\prod_{i=0}^{n-1}\cos\alpha_i\right) = 0{,}6073$$

Por ello, eligiendo $z_0=(0{,}6073,0)$, para n suficientemente grande se obtendrá un vector $z_n$ con módulo 1, y con argumento θ, y por tanto:
$x_n = \cos\theta$
$y_n = \mathrm{sen}\,\theta$

CALCULO de sen() y cos() por CORDIC en binario con 10 bits

Tabla de referencia

| i | $2^{-i}$ | Arctag ($2^{-i}$) | En rad | En grad |
|---|---|---|---|---|
| 0 | 1.0000000000 | 0.1100100100 | 0,785 | 45 |
| 1 | 0.1000000000 | 0.0111011011 | 0,464 | 26,565 |
| 2 | 0.0100000000 | 0.0011111011 | 0,245 | 14,036 |
| 3 | 0.0010000000 | 0.0001111111 | 0,124 | 7,125 |
| 4 | 0.0001000000 | 0.0001000000 | 0,062 | 3,576 |
| 5 | 0.0000100000 | 0.0000100000 | 0,031 | 1,790 |
| 6 | 0.0000010000 | 0.0000010000 | 0,016 | 0,896 |
| 7 | 0.0000001000 | 0.0000001000 | 0,008 | 0,448 |
| 8 | 0.0000000100 | 0.0000000100 | 0,004 | 0,224 |
| 9 | 0.0000000010 | 0.0000000010 | 0,002 | 0,112 |
| 10 | 0.0000000001 | 0.0000000001 | 0,001 | 0,056 |

Constante 1/K, valor inicial de cos() = 0,6074 = .1001101110

21

## Suma/resta en coma flotante

**Algoritmo:**
Sean e1 y e2 los exponentes de los dos operandos y m1, m2 las mantisas con el bit oculto.

**1.-** Si e1<e2 **intercambiar** los operandos. Esto garantiza que la diferencia de exponentes d=e1-e2 es siempre positiva.
Poner como exponente tentativo del resultado e=e1.

**2.-** Si los signos de los operandos difieren, reemplazar m2 por C2(m2).

**3.-** Colocar m2 en un registro de **p** bits y desplazarlo a la derecha **d** posiciones (introduciendo unos si se ha complementado m2) para igualar los exponentes.
De los bits desplazados fuera de **p**, poner el más significativo en un biestable **g**, el siguiente más significativo en un biestable **r**, y la "o-lógica" del resto de bits en un biestable **s** (sticky).

**4.-** Calcular m como la suma de m1 y el contenido de **p** (m2 modificado).
Si (signo de a1 ≠ signo de a2) and (bit_mas_significativo(m)=1) and (no carry-out) entonces m es negativo.
Reemplazar m con C2(m).
Esto sólo puede ocurrir si d=0.



## Suma/resta en coma flotante

**5.-** Normalización y ajuste de r y s :

Caso 1. Si (signo de a1 = signo de a2) and (carry-out) desplazar m una posición a la derecha, introduciendo un 1.
- s=(g or r or s). r=bit que sale de m al desplazar.

Caso 2. Si no estamos en el caso 1, desplazar a la izquierda hasta que el número esté normalizado.
(En el primer desplazamiento a la izquierda se introduce el contenido de g, en los restantes se introduce 0).
- Si no se ha necesitado desplazar => s=r or s r=g
- Si se ha desplazado una posición => r=r s= s
- Si se ha desplazado más de una posición => r=s=0 (esto sólo ocurre si a1 y a2 tienen signos opuestos y el mismo exponente, por lo que la suma ha sido exacta).

En ambos casos, ajustar el exponente adecuadamente.

**6.-** Redondear según la tabla que vimos anteriormente. Si se produce carry-out, desplazar a la derecha y ajustar el exponente.

**7.-** Signo del resultado:
Si (signo de a1 = signo de a2) este es el signo del resultado.
Si (signo de a1 ≠ signo de a2) el signo depende de cuál de los dos operandos fuera negativo, de si en el paso 1 se intercambiaron y de si en el paso 4 se reemplazó por su c2, como se muestra en la tabla.

| *Intercambio* | *C2* | *signo(a1)* | *signo(a2)* | *signo(resultado)* |
|---|---|---|---|---|
| si | | + | - | - |
| si | | - | + | + |
| no | no | + | - | + |
| no | no | - | + | - |
| no | si | + | - | - |
| no | si | - | + | + |

Significa que los exponentes son iguales y a2>a1