Implementación del Algoritmo CORDIC en Python

Julio Andrés Garibay Zepeda, Hector Alonso Hereida Pérez

December 6, 2025

1 Introducción

El algoritmo CORDIC (Coordinate Rotation Digital Computer) es un método iterativo muy eficiente para calcular funciones trigonométricas, hiperbólicas, exponenciales y logarítmicas usando únicamente sumas, restas y desplazamientos binarios. Esto lo hace ideal para hardware embebido, FPGAs y microcontroladores.

En este documento se presenta la implementación completa del algoritmo CORDIC en Python, junto con funciones auxiliares y pruebas comparativas.

2 Código fuente

A continuación se presenta el código completo implementado:

Archivo: `cordic_functions.py`

```
# Implementaci n del algoritmo CORDIC (Coordinate Rotation Digital
    Computer)
# para el c lculo de funciones trigonom tricas usando  nicamente
    aritm tica finita.

def cordic_sin(theta, iteraciones=16):
    pi = 3.14159265358979323846
    while theta > pi:  theta -= 2*pi
    while theta < -pi: theta += 2*pi

    atan_table = [
        0.78539816339745, 0.46364760900081, 0.24497866312686,
        0.12435499454676, 0.06241880999596, 0.03123983343027,
        0.01562372862048, 0.00781234106010, 0.00390623013197,
        0.00195312251648, 0.00097656218956, 0.00048828121119,
        0.00024414062015, 0.00012207031189, 0.00006103515617,
        0.00003051757812, 0.00001525878906, 0.00000762939453
    ]

    K = 0.60725293500888
```

```
    x, y, z = K, 0.0, theta

    for i in range(min(iteraciones, len(atan_table))):
        d = 1 if z >= 0 else -1
        p = 2 ** (-i)
        x, y, z = (
            x - d * y * p,
            y + d * x * p,
            z - d * atan_table[i]
        )

    return y

def cordic_cos(theta, iteraciones=16):
    pi = 3.14159265358979323846
    while theta > pi:  theta -= 2*pi
    while theta < -pi: theta += 2*pi

    atan_table = [
        0.78539816339745, 0.46364760900081, 0.24497866312686,
        0.12435499454676, 0.06241880999596, 0.03123983343027,
        0.01562372862048, 0.00781234106010, 0.00390623013197,
        0.00195312251648, 0.00097656218956, 0.00048828121119,
        0.00024414062015, 0.00012207031189, 0.00006103515617,
        0.00003051757812, 0.00001525878906, 0.00000762939453
    ]

    K = 0.60725293500888
    x, y, z = K, 0.0, theta

    for i in range(min(iteraciones, len(atan_table))):
        d = 1 if z >= 0 else -1
        p = 2 ** (-i)
        x, y, z = (
            x - d * y * p,
            y + d * x * p,
            z - d * atan_table[i]
        )

    return x

def cordic_tan(theta, iteraciones=16):
    cos_val = cordic_cos(theta, iteraciones)
    sin_val = cordic_sin(theta, iteraciones)
    if abs(cos_val) < 1e-10:
        return float('inf') if sin_val > 0 else float('-inf')
    return sin_val / cos_val

def cordic_atan(y, x, iteraciones=16):
    atan_table = [
        0.78539816339745, 0.46364760900081, 0.24497866312686,
        0.12435499454676, 0.06241880999596, 0.03123983343027,
        0.01562372862048, 0.00781234106010, 0.00390623013197,
        0.00195312251648, 0.00097656218956, 0.00048828121119,
```

```
        0.00024414062015, 0.00012207031189, 0.00006103515617,
        0.00003051757812, 0.00001525878906, 0.00000762939453
    ]

    if abs(x) < 1e-10 and abs(y) < 1e-10:
        return 0.0

    pi = 3.14159265358979323846
    ajuste = 0.0
    if x < 0:
        ajuste = pi if y >= 0 else -pi
        x, y = -x, -y

    z = 0.0
    for i in range(min(iteraciones, len(atan_table))):
        d = 1 if y >= 0 else -1
        p = 2 ** (-i)
        x, y, z = (
            x + d * y * p,
            y - d * x * p,
            z + d * atan_table[i]
        )

    return z + ajuste
```

3 Resultados experimentales

3.1 Cálculo de seno con CORDIC

Ángulo (rad)	CORDIC	Taylor	Diferencia
0	-0.0000175949	0.0000000000	1.76e-05
0.5	0.4794347459	0.4794255386	9.21e-06
1.0	0.8414770121	0.8414709848	6.03e-06
$\pi/2$	0.9999999997	1.0000000000	3.10e-10
π	0.9851656348	-0.0000000005	9.85e-01

3.2 Cálculo de coseno con CORDIC

Ángulo (rad)	CORDIC	Taylor	Diferencia
0	0.9999999997	1.0000000000	3.10e-10
0.5	0.8775775317	0.8775825619	5.03e-06
1.0	0.5402929185	0.5403023059	9.39e-06
$\pi/2$	-0.0000175949	0.0000000000	1.76e-05

3.3 Cálculo de tangente con CORDIC

Ángulo (rad)	Tangente CORDIC
0	-0.0000175949
0.5	0.5463161129
1.0	1.5574459397
$\pi/4$	0.9999689435

3.4 Cálculo de arcotangente con CORDIC

(x, y)	radianes	grados
(1, 0)	-0.0000175949	-0.0010081125
(1, 1)	0.7854136919	45.0008897177
(0, 1)	1.5708139217	90.0010081125

3.5 Cálculo de raíz cuadrada con CORDIC

n	CORDIC	$(\text{resultado})^2$	Error
4	2.0000000000	4.0000000000	0
9	3.0000000000	9.0000000000	0
2	1.4142135624	2.0000000000	4.44e-16

3.6 Cálculo de exponencial con CORDIC

x	e^x (CORDIC)
-2.0	0.1353352832
-1.0	0.3678794412
0.5	1.6487212707
1.0	2.7182818285

3.7 Cálculo de logaritmo natural con CORDIC

x	$\ln(x)$ (CORDIC)	Verificación
1	0.0000000000	1.0000000000
2	0.6931471806	2.0000000000
10	2.3025850930	10.0000000001

4 Conclusiones

VENTAJAS DEL ALGORITMO CORDIC:

1. Eficiencia computacional: Solo usa sumas, restas y desplazamientos binarios
2. Hardware simple: Ideal para implementación en FPGAs y microcontroladores
3. Precisión predecible: El error disminuye exponencialmente con las iteraciones
4. Versatilidad: Puede calcular múltiples funciones con el mismo algoritmo

COMPARACIÓN CON OTROS MÉTODOS:

Serie de Taylor: Requiere multiplicaciones y divisiones costosas
Tablas de lookup: Requieren mucha memoria para alta precisión
CORDIC: Balance óptimo entre precisión, velocidad y recursos

El algoritmo CORDIC permite calcular funciones matemáticas con alta precisión utilizando únicamente operaciones muy simples, lo cual lo hace especialmente útil en hardware de bajos recursos.