1 Алгоритм CORDIC для вычисления угла вектора

В вычислительной технике часто возникает задача определения угла между вектором (x, y) и осью OX. Традиционное решение с использованием тригонометрических функций требует значительных вычислительных ресурсов, что не всегда приемлемо для встроенных систем с ограниченными возможностями. В этой статье мы рассмотрим эффективный алгоритм CORDIC (COordinate Rotation DIgital Computer), который позволяет вычислять угол, используя только элементарные операции: сложение, вычитание и битовые сдвиги.

1.1 Постановка задачи

Предположим, мы знаем координаты точки (x, y). Наша задача - найти угол между осью OX и вектором (x, y), используя только элементарные операции. Для упрощения будем считать, что точка (x, y) находится в первой четверти.

1.2 Матрица поворота

Вспомним, что для поворота вектора на угол ϕ можно умножить его на матрицу поворота:

$$\begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}$$

Эту матрицу можно представить в другом виде:

$$\cos(\phi) \cdot \begin{bmatrix} 1 & -\tan(\phi) \\ \tan(\phi) & 1 \end{bmatrix}$$

Она повернёт вектор (x, y) на угол ϕ по часовой стрелке.

1.3 Алгоритм CORDIC

Идея алгоритма CORDIC заключается в последовательном повороте вектора на заранее заданные углы $\frac{\pi}{4}, \frac{\pi}{8}, \frac{\pi}{16}$ и т.д. На каждом шаге мы проверяем знак координаты y после поворота: если y стал отрицательным, значит угол был меньше текущего шага, если положительным - больше. Этот процесс напоминает бинарный поиск.

Мы уже можем запустить наш алгоритм, предварительно предподсчитав значения $\tan(\frac{\pi}{4}), \tan(\frac{\pi}{8}), \ldots$

Но при повороте необходимо производить умнжение чисел, а нам бы хотелось этого избежать - очень ресурсозатратно

Для оптимизации вычислений выбираем углы ϕ_j такие, что $\tan(\phi_j) = \frac{1}{2^j}$. Это позволяет заменить умножение битовым сдвигом, что значительно ускоряет вычисления.

Примерная последовательность таких углов:

$$\pi \cdot 0.250, \quad \pi \cdot 0.147, \quad \pi \cdot 0.077, \quad \pi \cdot 0.039, \quad \pi \cdot 0.019$$

Заметим, что они очень позожи на углы $\frac{\pi}{4}, \frac{\pi}{8}, \frac{\pi}{16}$...

1.4 Реализация алгоритма

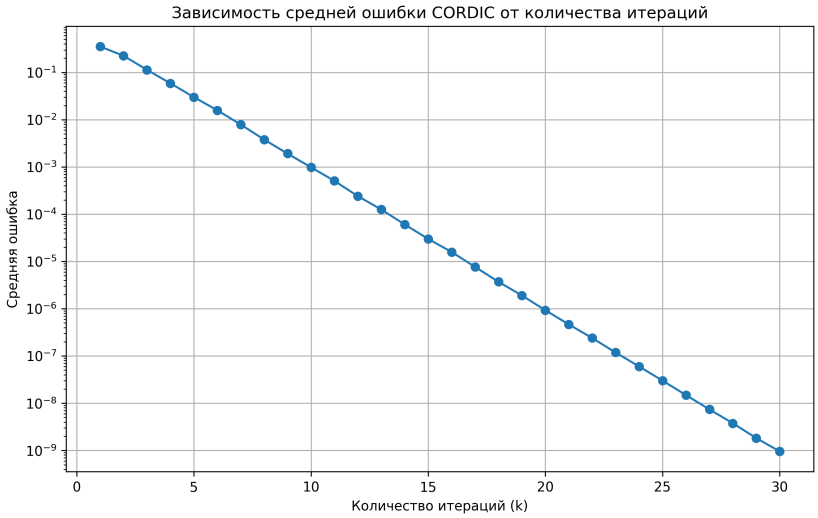
Приведём псевдокод алгоритма:

```
N = 16

angles = []

for j in range(N):
    angles.append(math.atan(1 / 2 ** j))

res_angle = 0

for i in range(N):
    t = 2 ** i
    if y > 0:
        res_angle += angles[i]
        x, y = x + y / t, - x / t + y
    else:
        res_angle -= angles[i]
        x, y = x - y / t, x / t + y
```

на github можно глянуть анимацию
https://github.com/gbu1at/cordic

1.5 Характеристики алгоритма

Взглянем на график зависимости средней по модулю ошибки от числа N.



Интересно, что график оказался линейным в логарифмическом масштабе :)

Как мы поняли, данный алгоритм достаточно быстро сходится к правильному ответу, используя очень мало памяти для предподсчёта и элементарные операции: битовый сдвиг и сложение. Его используют на аппаратном уровне для минимизации ресурсов.

Спасибо за прочтение!