Entendiendo getAmountOut y su Fórmula Mágica

Tu análisis de los parámetros de entrada (amountIn, reserveIn, reserveOut), el de salida (amountOut) y los require es **impecable y correcto**.

Ahora, vamos a la fórmula, que es donde está la genialidad y la confusión.

El Principio Básico: x * y = k (Sin Comisiones)

Imagina por un segundo que no hay comisiones. El principio de un pool de Uniswap es que el producto de las reservas de ambos tokens siempre debe ser constante (o aumentar).

- reserva_A * reserva_B = k (la constante de liquidez)

Cuando tú metes amountIn de Token A, la nueva reserva de A será reserva_A + amountIn. Para que la constante k se mantenga, el pool debe entregarte una cantidad amountOut de Token B, de modo que la nueva reserva de B sea reserva_B - amountOut.

La ecuación sería: (reserva_A + amountIn) * (reserva_B - amountOut) = reserva_A * reserva_B

Si despejaras amountOut de esa ecuación, obtendrías el resultado del intercambio. Pero esto no incluye la comisión.

Introduciendo la Comisión del 0.3%

Aquí es donde entran los números 997 y 1000.

Uniswap v2 cobra una comisión fija del **0.3%** en cada intercambio. Esta comisión se queda en el pool para recompensar a los proveedores de liquidez.

Como Solidity no usa decimales, no podemos hacer amountIn * 0.3 / 100. La forma de calcular un porcentaje con números enteros es a través de fracciones.

- Un 0.3% de comisión significa que del amountIn que tú entregas, solo el **99.7%** se usa realmente para el intercambio.
- En forma de fracción, 99.7% es **997 / 1000**.

Ahora, analicemos la fórmula que tienes, paso a paso:

1. **uint amountInWithFee = amountIn.mul(997);**
    - **Traducción:** "Toma la cantidad que el usuario quiere cambiar (amountIn) y calcula el 99.7% de ella."
    - Multiplicamos por 997 en lugar de 0.997 porque estamos trabajando con

enteros. Estamos preparando el cálculo para una división final.

2. **uint numerator = amountInWithFee.mul(reserveOut);**
    - **Traducción:** "El numerador de nuestra división será: la cantidad de entrada (con la comisión ya descontada) multiplicada por la reserva del token que queremos recibir (reserveOut)."
    - Hasta aquí, es bastante lógico. Es una regla de tres.
3. **uint denominator = reserveIn.mul(1000).add(amountInWithFee);**
    - **Traducción:** "El denominador será: la reserva del token que estamos entregando (reserveIn) **sumada a** la cantidad que estamos metiendo (el 99.7%)."
    - ¿Por qué? Porque después del intercambio, la nueva reserva del token de entrada no será la original, sino la original más la nueva cantidad que se ha añadido. La fórmula final del AMM es:
      amountOut = (reserveOut * amountIn_con_comisión) / (reserveIn + amountIn_con_comisión)
    - **¿Y por qué reserveIn.mul(1000)?** Este es el truco matemático para evitar decimales. Como amountInWithFee ya está multiplicado por 997, tenemos que poner reserveIn en la misma "escala" antes de sumarlos. Al multiplicar reserveIn por 1000, mantenemos la proporción correcta para la fórmula final.
4. **amountOut = numerator.div(denominator);**
    - **Traducción:** "Ahora sí, realiza la división final para obtener la cantidad de tokens que el usuario recibirá."

Ejemplo con Números

Imagina:

- Quieres cambiar **100** tokens A (amountIn).
- El pool tiene **10,000** de Token A (reserveIn).
- El pool tiene **20,000** de Token B (reserveOut).

1. amountInWithFee = 100 * 997 = 99,700
2. numerator = 99,700 * 20,000 = 1,994,000,000
3. denominator = (10,000 * 1000) + 99,700 = 10,000,000 + 99,700 = 10,099,700
4. amountOut = 1,994,000,000 / 10,099,700 = **197** (la división de enteros trunca los decimales).

Así que por tus 100 tokens A, recibirías 197 tokens B.

Espero que este desglose te haya aclarado la lógica. Es la pieza de matemática más importante del protocolo. ¡Entenderla te pone en un nivel muy superior!