C++ Implementation

Pieter P

Dividing by Powers of 2

The factor α in the difference equation of the Exponential Moving Average filter is a number between zero and one. There are two main ways to implement this multiplication by α: Either we use floating point numbers and calculate the multiplication directly, or we use integers, and express the multiplication as a division by $1/\alpha > 1$.
Both floating point multiplication and integer division are relatively expensive operations, especially on embedded devices or microcontrollers.

We can, however, choose the value for α in such a way that $1/\alpha = 2^k, k \in \mathbb{N}$.
This is useful, because a division by a power of two can be replaced by a very fast right bitshift:

$$\alpha \cdot x = \frac{x}{2^k} = x \gg k$$

We can now rewrite the difference equation of the EMA with this optimization in mind:

$$\begin{aligned} y[n] &= \alpha x[n] + (1-\alpha) y[n-1] \\ &= y[n-1] + \alpha\left(x[n] - y[n-1]\right) \\ &= y[n-1] + \frac{x[n]-y[n-1]}{2^k} \\ &= y[n-1] + \left(x[n] - y[n-1]\right) \gg k \end{aligned}$$

Negative Numbers

There's one caveat though: this doesn't work for negative numbers. For example, if we try to calculate the integer division $-15/4$ using this method, we get the following answer:

$$\begin{aligned} -15/4 &= -15 \cdot 2^{-2} \\ -15 \gg 2 &= 0b11110001 \gg 2 \\ &= 0b11111100 \\ &= -4 \end{aligned}$$

This is not what we expected! Integer division in programming languages such as C++ returns the quotient truncated towards zero, so we would expect a value of -3. The result is close, but incorrect nonetheless.

This means we'll have to be careful not to use this trick on any negative numbers. In our difference equation, both the input $x[n]$ and the output $y[n]$ will generally be positive numbers, so no problem there, but their difference can be negative. This is a problem. We'll have to come up with a different representation of the difference equation that doesn't require us to divide any negative numbers:

$$\begin{aligned} y[n] &= y[n-1] + \alpha\left(x[n] - y[n-1]\right) \\ y[n] &= y[n-1] + \frac{x[n]-y[n-1]}{2^k} \\ 2^k y[n] &= 2^k y[n-1] + x[n] - y[n-1] \\ z[n] &\triangleq 2^k y[n] \Leftrightarrow y[n] = 2^{-k} z[n] \\ z[n] &= z[n-1] + x[n] - 2^{-k} z[n-1] \end{aligned}$$

We now have to prove that $z[n-1]$ is greater than or equal to zero. We'll prove this using induction:

Base case: $n - 1 = -1$

The value of $z[-1]$ is the initial state of the system. We can just choose any value, so we'll pick a value that's greater than or equal to zero: $z[-1] \geq 0$.

Induction step: n

Given that $z[n-1] \geq 0$, we can now use the difference equation to prove that $z[n]$ is also greater than zero:

$$z[n] = z[n-1] + x[n] - 2^{-k} z[n-1]$$

We know that the input $x[n]$ is always zero or positive.
Since $k > 1 \Rightarrow 2^{-k} < 1$, and since $z[n-1]$ is zero or positive as well, we know that $z[n-1] \geq 2^{-k} z[n-1] \Rightarrow z[n-1] - 2^{-k} z[n-1] \geq 0$.
Therefore, the entire right-hand side is always positive or zero, because it is a sum of two numbers that are themselves greater than or equal to zero. $\square$

Rounding

A final improvement we can make to our division algorithm is to round the result to the nearest integer, instead of truncating it towards zero.
Consider the rounded result of the division a/b. We can then express it as a flooring of the result plus one half:

$$\begin{aligned}\left\lfloor\frac{a}{b}\right\rceil &= \left\lfloor\frac{a}{b}+\frac{1}{2}\right\rfloor \\ &= \left\lfloor\frac{a+\frac{b}{2}}{b}\right\rfloor\end{aligned}$$

When b is a power of two, this is equivalent to:

$$\begin{aligned}\left\lfloor\frac{a}{2^k}\right\rceil &= \left\lfloor\frac{a}{2^k}+\frac{1}{2}\right\rfloor \\ &= \left\lfloor\frac{a+\frac{2^k}{2}}{2^k}\right\rfloor \\ &= \left\lfloor\frac{a+2^{k-1}}{2^k}\right\rfloor \\ &= (a+1 \ll (k-1)) \gg k\end{aligned}$$

Implementation in C++

We now have everything in place to write an implementation of the EMA in C++:

```
#include <stdint.h>

template <uint8_t K, class uint_t>
class EMA {
  public:
    uint_t filter(uint_t x) {
        z += x;
        uint_t y = (z + (1 << (K - 1))) >> K;
        z -= y;
        return y;
    }

  private:
    uint_t z = 0;
};
```

Note how we save $z[n] - 2^{-k}z[n]$ instead of just $z[n]$. Otherwise, we would have to calculate $2^{-k}z[n]$ twice (once to calculate $y[n]$, and once on the next iteration to calculate $2^{-k}z[n-1]$), and that would be unnecessary.