A Pricing Formula for the Share Dispenser

Alethena*

February 21, 2019

1 Linear Pricing

Assume that a total of N shares are made available for sale, we can number them $1, ..., N$. The first share will be sold at price p_{min}, the last one at p_{max} with linear interpolation in-between. This means that the k-th share will be sold at price

$$P(k) = p_{min} + \frac{k-1}{N-1}(p_{max} - p_{min}).$$

The share dispenser smart contract needs to be able to quickly compute the cumulated price for buying shares m through l, which is easily possible as follows:

$$\begin{aligned} P(m,l) =& \sum_{k=m}^{l} P(k) \\ =& (l-m+1)p_{min} + \frac{p_{max} - p_{min}}{N-1} \sum_{k=m-1}^{l-1} k \\ =& (l-m+1)p_{min} + \frac{p_{max} - p_{min}}{2(N-1)} \Big(l(l-1) - (m-1)(m-2) \Big). \end{aligned}$$

This expression is quadratic in l which means that after fixing m as well as the cumulated price we can solve for l. This gives us the maximum number of shares that can be purchased starting from share number m given a fixed price.
In Solidity, the above can be implemented as follows (where m and l are called `first` and `last` respectively):

```
1 function helper(uint256 first, uint256 last) internal view returns (uint256)
      {
2     uint256 tempa = last.sub(first).add(1).mul(minPriceInXCHF);
3     uint256 tempb = maxPriceInXCHF.sub(minPriceInXCHF).div(
          initialNumberOfShares.sub(1)).div(2);
4     uint256 tempc = last.mul(last).add(first.mul(3)).sub(last).sub(first.mul
          (first)).sub(2);
5     return tempb.mul(tempc).add(tempa);
6     }
```

*Equility AG, Dammstrasse 16, CH-6300 Zug, Switzerland, e-mail: contact@alethena.com, website: www.alethena.com. Alethena is a brand of Equility AG, an independent legal entity registered in the Canton of Zug (registered under registration number CHE−460.255.304 with the Commercial Register of the Canton of Zug), Switzerland.

Assuming a more complex price dependency a closed inverse formula is highly unlikely and therefore it might be more efficient to use a binary search type algorithm to find l.

2 The Available Supply Can Exceed N

Furthermore, if a company decides to also provide XCHF to the share dispenser, a situation can occur where more than N shares are available for purchase. No share should be sold below the minimum price. Assume there are a total of $N + i$ shares available, then shares $N + i$ through $N + 1$ will be sold at price p_{min} and starting at share N the regular pricing formula takes over.
In Solidity, this can be implemented as follows (note that within the code N is labelled as `initialNumberOfShares`):

```
function getCumulatedPrice(uint256 amount, uint256 supply) public view
    returns (uint256){
     uint256 cumulatedPrice = 0;
     if (supply <= initialNumberOfShares) {
         uint256 first = initialNumberOfShares.add(1).sub(supply);
         uint256 last = first.add(amount).sub(1);
         cumulatedPrice = helper(first, last);
     }

     else if (supply.sub(amount) >= initialNumberOfShares) {
         cumulatedPrice = minPriceInXCHF.mul(amount);
     }

     else {
         cumulatedPrice = supply.sub(initialNumberOfShares).mul(
             minPriceInXCHF);
         uint256 first = 1;
         uint256 last = amount.sub(supply.sub(initialNumberOfShares));
         cumulatedPrice = cumulatedPrice.add(helper(first,last));
     }

     return cumulatedPrice;
     }
```