Off-chain Limit Orders for the LMSR/LS-LMSR

Jack Peterson
www.augur.net

The user wants to make a buy limit order on outcome i. The user enters:

- Limit price $p(q_i)$.
- Total number of shares to buy, N.
- Price cap for the stop order, ξ. This is the maximum price the user is willing to accept for this order.

What is the maximum number of shares (n) the user can buy such that the price $p(q_i + n)$ does not rise above the price cap (ξ)?

$$p\left(q_i + n\right) = \xi \tag{1}$$

To solve Eq. 1 for n, the price function p must be specified. First, use the LMSR's simple price function:

$$p\left(q_i\right) = \frac{\mathrm{e}^{\beta q_i}}{\sum_j \mathrm{e}^{\beta q_j}}, \tag{2}$$

making the substitution $\beta \equiv b^{-1}$ for readability.[1] Plug Eq. 2 into Eq. 1 and rearrange to solve for n:

$$n = -q_i + \frac{1}{\beta} \log\left(\frac{\xi}{1-\xi} \sum_{j \neq i} \mathrm{e}^{\beta q_j}\right). \tag{3}$$

If $n \geq N$, this is just a stop order: it converts to a market order, is completely filled by the automated market maker, and nothing further happens. If $n < N$, a market order for n shares is submitted and filled by the market maker (bringing the price to ξ). This leaves $N - n$ total shares in the user's limit order. The order remains open until the market maker's price again drops to the limit price $p(q_i)$.

The LS-LMSR's price function can also be used, although since it is more complicated there is not a closed-form expression for n like Eq. 3. The LS-LMSR's price function is

$$p\left(q_i\right) = \alpha \log\left(\sum_j \mathrm{e}^{q_j / b(q_i)}\right) + \frac{\mathrm{e}^{q_i/b(q_i)} \displaystyle\sum_j q_j - \sum_j q_j \mathrm{e}^{q_j/b(q_i)}}{\displaystyle\sum_j q_j \sum_j \mathrm{e}^{q_j/b(q_i)}}, \tag{4}$$

where $b\left(q_i\right) \equiv \alpha \sum_j q_j$. Plug Eq. 4 into Eq. 1:

$$b\left(q_i + n\right) = b\left(q_i\right) + n \tag{5}$$

$$\alpha \log\left(\mathrm{e}^{\frac{q_i+n}{b(q_i)+n}} + \sum_{j \neq i} \mathrm{e}^{\frac{q_j}{b(q_i)+n}}\right) + \frac{\mathrm{e}^{\frac{q_i+n}{b(q_i)+n}} \displaystyle\sum_{j \neq i} q_j - \sum_{j \neq i} q_j \mathrm{e}^{\frac{q_j}{b(q_i)+n}}}{\left(n + \displaystyle\sum_j q_j\right)\left(\mathrm{e}^{\frac{q_i+n}{b(q_i)+n}} + \displaystyle\sum_{j \neq i} \mathrm{e}^{\frac{q_j}{b(q_i)+n}}\right)} = \xi \tag{6}$$

Eq. 6 is then numerically solved for n.

---

[1] $p(q_i)$ and $b(q_i)$ are written as univariate functions because only q_i is varied here.

Example implementation (Matlab)

```
% price cap
xi = 0.3;

% LMSR
beta = 1;
q = 10*ones(1,5);
i = 1;
qj = [q(1:i-1) q(i+1:end)];
n = -q(i) + log(xi*sum(exp(beta*qj))/(1 - xi)) / beta
q(i) = q(i) + n;
p_lmsr = exp(beta*q) / sum(exp(beta*q))

% LS-LMSR
clear n
a = 0.0079;
q = 10*ones(1,5);
i = 1;
qj = [q(1:i-1) q(i+1:end)];
F = @(n) a*log(exp((q(i) + n)/a/(n + sum(q))) + sum(exp(qj/a/(n + sum(q))))) + ...
    (exp((q(i) + n)/a/(n + sum(q)))*sum(qj) - sum(qj.*exp(qj/a/(n + sum(q))))) / ...
    ((n + sum(q))*(exp((q(i) + n)/a/(n + sum(q))) + sum(exp(qj/a/(n + sum(q)))))) - xi;
n0 = fsolve(F, 0.05)
q(i) = q(i) + n0;
b = a*sum(q);
p_lslmsr = a*log(sum(exp(q/b))) + ...
    (exp(q/b)*sum(q) - sum(q.*exp(q/b))) / sum(q) / sum(exp(q/b))
```

Example implementation (JavaScript)

```
var numeric = require("numeric");

// Newton's method parameters
var tolerance = 0.00000001;
var epsilon = 0.0000000000001;
var maxIter = 250;

// MSR parameters
var q = [10, 10, 10, 10, 10]; // shares
var i = 1;                    // outcome to trade
var a = 0.0079;               // LS-LMSR alpha
var xi = 0.3;                 // price cap

// Find roots of f using Newton-Raphson.
// http://www.turb0js.com/a/Newton%E2%80%93Raphson_method
function solve(f, fprime, x0) {
    var next_x0;
    for (var i = 0; i < maxIter; ++i) {
        var denominator = fprime(x0);
        if (Math.abs(denominator) < epsilon) return null;
        next_x0 = x0 - f(x0) / denominator;
        if (Math.abs(next_x0 - x0) < tolerance) return next_x0;
        x0 = next_x0;
    }
}

// LS-LMSR price function (Eq. 6)
function f(n) {
    var qj = numeric.clone(q);
    qj.splice(i, 1);
    var q_plus_n = n + numeric.sum(q);
    var b = a * q_plus_n;
    var exp_qi = Math.exp((q[i] + n) / b);
    var exp_qj = numeric.exp(numeric.div(qj, b));
    var sum_exp_qj = numeric.sum(exp_qj);
    return a*Math.log(Math.exp((q[i] + n)/b) + sum_exp_qj) +
        (exp_qi*numeric.sum(qj) - numeric.sum(numeric.mul(qj, exp_qj))) /
        (q_plus_n*(exp_qi + sum_exp_qj)) - xi;
};

// First derivative of f
function fprime(n) {
    return (f(n + 0.000001) - f(n - 0.000001)) / 0.000002;
};

console.log(solve(f, fprime, 0.05));
```