

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)?

$$p(q_i + n) = \xi \quad (1)$$

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

$$p(q_i) = \frac{e^{\beta q_i}}{\sum_j e^{\beta q_j}}, \quad (2)$$

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

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

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

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

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

$$b(q_i + n) = b(q_i) + n \quad (5)$$

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

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_ls_lmsr = 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)

```
#!/usr/bin/env node

var Decimal = require("decimal.js");
var fzero = require("fzero");

// MSR parameters
var q = [new Decimal(10),      // outcome 1 shares
         new Decimal(10),      // outcome 2 shares
         new Decimal(10),      // outcome 3 shares
         new Decimal(10),      // outcome 4 shares
         new Decimal(10)];     // outcome 5 shares
var i = 1;                     // outcome to trade
var a = new Decimal("0.0079"); // LS-LMSR alpha
var xi = new Decimal("0.3");   // price cap

// LS-LMSR objective function (Eq. 6)
function f(n) {
  n = new Decimal(n);
  var numOutcomes = q.length;
  var qj = new Array(numOutcomes);
  var sum_q = new Decimal(0);
  for (var j = 0; j < numOutcomes; ++j) {
    qj[j] = q[j];
    sum_q = sum_q.plus(q[j]);
  }
  qj.splice(i, 1);
  var q_plus_n = n.plus(sum_q);
  var b = a.times(q_plus_n);
  var exp_qi = q[i].plus(n).dividedBy(b).exp();
  var exp_qj = new Array(numOutcomes);
  var sum_qj = new Decimal(0);
  var sum_exp_qj = new Decimal(0);
  var sum_qj_x_expqj = new Decimal(0);
  for (j = 0; j < numOutcomes - 1; ++j) {
    sum_qj = sum_qj.plus(qj[j]);
    exp_qj[j] = qj[j].dividedBy(b).exp();
    sum_exp_qj = sum_exp_qj.plus(exp_qj[j]);
    sum_qj_x_expqj = sum_qj_x_expqj.plus(qj[j].times(exp_qj[j]));
  }
  return a.times(q[i].plus(n).dividedBy(b).exp().plus(sum_exp_qj).ln()).plus(
    exp_qi.times(sum_qj).minus(sum_qj_x_expqj).dividedBy(
      q_plus_n.times(exp_qi.plus(sum_exp_qj))
    ).minus(xi)
  );
}

console.log(fzero(f, [0.001, 10]).solution);
```