1 Token pair execution

1.1 Summay

This document describes an algorithm for matching a set of orders between a pair of tokens, maximizing our current objective function. Its primary motivation is to help understanding the code in the open solver, so that it can be truly *open* (further audited/extended by anyone).

1.2 Notation

Notation is based on Tom's paper. The problem consists of a set of n orders. An order $o_i = \left(b_i, s_i, \overline{y}_i, \pi_i\right)$, $0 < i \leq n$, is defined by the buy and sell tokens b_i, s_i, the maximum sell amount $\overline{y}_i$, and maximum exchange rate π_i. The actual executed buy and sell amounts for order o_i are denoted as x_i and y_i, respectively. An executed order o_i is called *unfilled*, *partial*, or *filled* if $y_i = 0$, $y_i \in \left]0, \overline{y}_i\right[$, or $y_i = \overline{y}_i$, respectively.

The current problem considers only two tokens, τ_1, τ_2, and thus the set of orders can be partitioned in two: the set B of orders that buy τ_1 and the set S of orders that sell τ_1 (or equivalently, that buy τ_2). That is, $B \cup S = \{1, \ldots, n\}$.

Let $r_i = y_i / x_i$ be the exchange rate associated with order o_i. For simplicity, the fee is not considered in this document.

1.3 Problem

1.3.1 Structural constraints

uniform clearing price All orders are executed according to a unique exchange rate r. More formally, the exchange rate r for the problem satisfies $r_i = r, \forall_{i \in B}$ and $1/r_j = r, \forall_{j \in S}$.

limited sell amount For every order o_i, condition $y_i \leq \overline{y}_i$ must hold.

limit price For every executed order o_i, i.e. where $y_i > 0$, condition $r_i \leq \pi_i$ must hold.

token balance The amounts of token τ_2 bought and sold must be equal for every order,

$$\sum_{i \in B} y_i = \sum_{j \in S} x_j$$

1.3.2 Objective

The function to optimize, known as *disregarded utility over all orders*, is defined as

$$f(r,\mathbf{y}) = p_1 \sum_{i \in B} \frac{(2y_i - \overline{y}_i)(\pi_i - r)}{\pi_i r}[r \le \pi_i] + p_1 \sum_{j \in S} \frac{(2y_j - \overline{y}_j)(\pi_j r - 1)}{\pi_j r}\left[r \ge 1/\pi_j\right] \tag{1.1}$$

Note that this function is non-linear - the sell amounts y_i, exchange rate r, and price p_1 of token τ_1 are unkowns. It is also non-differentiable due to the conditions in square brackets.

1.4 Optimal execution for a given exchange rate

An easier problem is to find the executed amounts $\mathbf{y}$ when the exchange rate r is known. In that case, the objective function simplifies to,

$$f(\mathbf{y}) = p_1 \sum_{i \in B'} \frac{(2y_i - \overline{y}_i)(\pi_i - r)}{\pi_i r} + p_1 \sum_{j \in S'} \frac{(2y_j - \overline{y}_j)(\pi_j r - 1)}{\pi_j r} \tag{1.2}$$

where $B' = \{i | i \in B \wedge r \le \pi_i\}$, and $S' = \left\{j | j \in S \wedge r \ge 1/\pi_j\right\}$, are constants.

1.4.1 Intuition

Recall that the only variables in the above expression are the y_i. Since, for any function g,

$$\begin{aligned} \arg_x \max g(c + x) &= \arg_x \max g(x) \\ \arg_x \max g(c\,x) &= \arg_x \max g(x), \forall c > 0 \end{aligned}$$

equation 1.2 can be further simplified to,

$$f(\mathbf{y}) = \sum_{i \in B'} a_i y_i + \sum_{j \in S'} b_j y_j$$

where $a_i = (1 - r/\pi_i)$ and $b_i = (r - 1/\pi_i)$ are non-negative constants (guaranteed by the definition of B' and S').

It is clear that, in order to maximize f above, the y_i should be maximized as much as possible. Due to the maximum sell amount constraint, the amount that may be assigned to each y_i is of course bounded. Additionally, due to the token balance constraint, an y_i increment of an order $i \in B'$ must be balanced by some y_j increment of one (or more) orders $j \in S'$.

It helps to consider a greedy approach to solving what looks like a kind of a resource allocation problem. Imagine any y_i can be incremented by some amount, that is $y_i < \overline{y}_i$ for all orders. Which order o_i should be chosen? Naturally, to maximize f, the y_i for which its constant coefficient (a_i or b_i) is the largest should be selected. Any positive amount that can be added to that y_i with will have a higher impact on maximizing f than if it was assigned to any other order. This intuitively determines the sequence of execution: by decreasing value

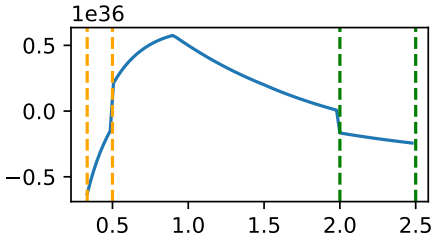

Figure 1.1: Token pair matching with 4 orders: $S = \{1,2\}$, and $B = \{3,4\}$. The blue curve shows the value of the objective function (yy) as a function of the exchange rate (xx), assuming an optimal order execution (i.e. $\mathbf{y}$) for that exchange rate. The two dashed green vertical lines on the right signal the limit prices of the buy orders, $\pi_3 = 2$ and $\pi_4 = 2.5$. The orange dashed vertical lines on the left signal the inverse of limit prices of the sell orders, $1/\pi_1 = 1/3$ and $1/\pi_2 = 0.5$. The optimal exchange rate is close to 1.

of a_i, b_i, coefficients[1]. Furthermore remark that ordering orders by decreasing value of a_i, b_i is equivelent to ordering orders by decreasing value of π_i.

The above reasoning also provides an intuition about how much should each y_i be incremented - as much as possible. Roughly, the limit is a function of the minimum of $\overline{y}_i$ and maximum buying capacity on the "other side".

Example 1. Figure 1.1 illustrates a matching between two orders and two counter-orders on a token pair. Assume that the exchange rate $r = 1$ is given, and therefore $S' = S$, and $B' = B$. The algorithm first selects the order with highest limit price, which is the leftmost sell order with $\pi_1 = 3$. Then matches this order with the buy order with highest limit price, which is the rightmost buy order with $\pi_4 = 2.5$. The traded amounts y_1 and y_2 can be obtained from the sell amount limits of both orders, taking into consideration the token balance constraint and the fact that r is known. For example, $y_1 \leftarrow \min\left(\overline{y}_1, \overline{y}_2 r\right)$. After this step either o_1 or o_4 (or both) is fully executed, which dictates which order should be considered next: whichever is partially executed, or if none, the order with the highest limit price. This ping-pong continues, in the direction from the outer orders to the inner orders, until no more orders can be selected.

1.4.2 More formally

Lemma 2. *There is an optimal solution* $\mathbf{y}^\star = \arg_{\mathbf{y}} \max f\left(y\right)$ *where either a) all buy orders* B' *are filled, or b) all sell orders* S' *are filled, or c) all orders are filled.*

Proof. Increasing y_i of an order can only increase $f\left(y\right)$, that is $\partial f / \partial y_i > 0$ (see sec.1.4.1). Therefore if there is both a buy order and a sell order in the solution that are not completely filled, the solution cannot be optimal. □

[1]Side note: if f was volume then these coefficients would not exist. That is why optimizing for volume objective does not guarantee that orders with higher limit prices will be executed first.

Lemma 3. *For the optimal solution* $\mathbf{y}^\star = \arg_{\mathbf{y}} \max f(\mathbf{y})$*, it must be the case that* $y_j > 0 \Rightarrow y_i = \overline{y}_i$ *for all pair of buy orders* o_i, o_j *such that* $\pi_i > \pi_j$*. Similarly for the sell orders.*

Proof. The only *active* constraints involved in the subproblem obtained when r is given are the *token balance* and *limited sell amount* constraints (all others are satisfied by construction). The previous lemma asserts that either the buy orders B' are completely filled, or the sell orders S' are completely filled, or both. Assuming the first, the token balance constraint becomes,

$$\sum_{i \in B'} \overline{y}_i = \sum_{j \in S} y_j r$$

The problem is therefore reduced to maximize f subject to $\sum_{j \in S} y_j = c$ where c is a constant. The result follows since $\partial f / \partial y_j \propto \pi_j$ (see sec.1.4.1). Assuming instead that the sell orders S' are completely filled leads to the same conclusion. □

Lemma 4. *Finding the optimal solution to* $\mathbf{y}^\star = \arg_{\mathbf{y}} \max f(\mathbf{y})$ *can be done in* $O\left(n \log n\right)$*, where* n *is the number of orders.*

Proof. The previous lemma determines the order of execution. It requires sorting buy and sell orders, and then a linear traversal to actually compute the traded amounts. □

1.5 Finding the optimal exchange rate

The problem of executing all orders in a token pair when the exchange rate r is not known is slightly more complex. The main idea behind the algorithm is to split the domain of the optimal exchange rate,

$$r^\star \in \left[\min_{j \in S} \frac{1}{\pi_j}, \max_{i \in B} \pi_i \right] \tag{1.3}$$

into a finite set of intervals $\mathscr{C}$ that cover the entire domain, but where the local maximum of f can be computed analytically. The choice of $\mathscr{C}$ is natural: the minimal set of intervals that, by turning the parts in square brackets of eq. 1.1 into constants, make f differentiable. That is, the intervals determined by consecutive limit prices.

Example 5. For the example of fig. 1.1,

$$\mathscr{C} = \{[1/\pi_1, 1/\pi_2], [1/\pi_2, \pi_3], [\pi_3, \pi_4]\}$$

1.5.1 Local optima for a given interval

For each interval $[a, b] \in \mathscr{C}$, the objective function f reduces to something very similar to eq. 1.2,

$$f(r, \mathbf{y}) = p_1 \sum_{i \in B'} \frac{\left(2y_i - \overline{y}_i\right)(\pi_i - r)}{\pi_i r} + p_1 \sum_{j \in S'} \frac{\left(2y_j - \overline{y}_j\right)\left(\pi_j r - 1\right)}{\pi_j r} \tag{1.4}$$

where $B' = \{i | i \in B \wedge b \leq \pi_i\}$, and $S' = \left\{j | j \in S \wedge a \geq 1/\pi_j\right\}$, are constants, and $r \in [a,b]$.

To simplify this problem up to a point where it is possible to solve it analytically, orders are partioned into three subsets U, P, F according if they are unfilled, partially filled, or completely filled. Assume for now that we somehow know these subsets. Then the previous equation can be rewritten as,

$$\begin{aligned} f(r,\mathbf{y}) = & p_1 \sum_{i \in B' \cap P} \frac{(2y_i - \overline{y}_i)(\pi_i - r)}{\pi_i r} + p_1 \sum_{i \in B' \cap U} \frac{-\overline{y}_i(\pi_i - r)}{\pi_i r} + p_1 \sum_{i \in B' \cap F} \frac{\overline{y}_i(\pi_i - r)}{\pi_i r} + \\ & + p_1 \sum_{j \in S' \cap P} \frac{(2y_j - \overline{y}_j)(\pi_j r - 1)}{\pi_j r} + p_1 \sum_{j \in S' \cap U} \frac{-\overline{y}_j(\pi_j r - 1)}{\pi_j r} + p_1 \sum_{j \in S' \cap F} \frac{\overline{y}_j(\pi_j r - 1)}{\pi_j r} \end{aligned}$$

Lemma 6. *Any solution to $r^\star, \mathbf{y}^\star = \arg_{r,\mathbf{y}} \max f(r,\mathbf{y})$ for an optimal exchange rate $r^\star \in [a,b] \in \mathscr{C}$ contains at least one filled order and at most one partial order.*

Proof. Given that the domain of $r^\star$ is not empty (eq. 1.3) and since all orders from one side must be completely filled (lemma 1.2) then at least one order must be fully filled. Lemma 3 asserts that can be at most one partially filled counter on the other side. □

From the previous lemma we know there is at most one partially filled buy order o_k or a partially filled sell order o_l in the optimal solution. Using this fact, and removing constant terms and positive multiplicative factors, f can be rewritten as,

$$f(r,\mathbf{y}) = \frac{(2y_k - \overline{y}_k)(\pi_k - r)}{\pi_k r} + \frac{(2y_l - \overline{y}_l)(\pi_l r - 1)}{\pi_l r} + \frac{c}{r} \tag{1.5}$$

with constant

$$c = -\sum_{i \in B' \cap U} \overline{y}_i + \sum_{i \in B' \cap F} \overline{y}_i + \sum_{j \in S' \cap U} \overline{y}_j/\pi_j - \sum_{j \in S' \cap F} \overline{y}_j/\pi_j$$

The full constrained optimization problem that needs to be solved analytically is to maximize eq. 1.5 subject to,

$$\begin{aligned} y_k + \textstyle\sum_{i \in B' \cap F} \overline{y}_i - r(y_l + \sum_{j \in S' \cap F} \overline{y}_j) = 0 \quad & \text{(token balance)} \\ y_k \leq \overline{y}_k \quad & \text{(limit sell amount)} \\ y_l \leq \overline{y}_l \quad & \text{(limit sell amount)} \\ 1/\pi_l \leq r \leq \pi_k \quad & \text{(limit price)} \end{aligned}$$

The problem can be solved using the method of Lagrange multipliers. That is, the local maxima of the above optimization problem are the solutions of the following system of equations,

$$\begin{cases} \nabla f(r,\mathbf{y},\mathbf{s}) - \sum_{i=1}^{5} \lambda_i \nabla g_i(r,\mathbf{y},\mathbf{s}) & = 0 \\ \bigwedge_{i=1}^{5} g_i(r,\mathbf{y},\mathbf{s}) & = 0 \end{cases} \tag{1.6}$$

where a set $\mathbf{s}$ of *slack variables* are introduced to rewrite the constraint inequations into equations. The five g_i correspond the transformed constraints:

$$
\begin{aligned}
g_1(r,\mathbf{y},\mathbf{s}) &= y_k + \sum_{i\in B'\cap F} \overline{y}_i - r(y_l + \sum_{j\in S'\cap F} \overline{y}_j) \\
g_2(r,\mathbf{y},\mathbf{s}) &= y_k + s_1^2 - \overline{y}_k \\
g_3(r,\mathbf{y},\mathbf{s}) &= y_l + s_2^2 - \overline{y}_l \\
g_4(r,\mathbf{y},\mathbf{s}) &= r - \pi_k + s_3^2 \\
g_5(r,\mathbf{y},\mathbf{s}) &= \frac{1}{r} - \pi_l + s_4^2
\end{aligned}
$$

The system of equations in eq. 1.6 contains three *structural* variables (r, y_k, y_l), four *slack* variables $s_1,\ldots,s_4$, and five lagrange multipliers $\lambda_1,\ldots,\lambda_5$. Its five real-valued roots can be determined analytically:

$$
\begin{aligned}
r &= \pi_k \\
r &= 1/\pi_l \\
r &= \frac{4\left(\overline{y}_k + a\right)}{\pi_l\left(3\overline{y}_k + 2a + c\right) + y_l + 2b} \\
r &= \frac{\sqrt{\pi_k\pi_l\left(\overline{y}_k + a - c\right) + \pi_k\overline{y}_l}}{\sqrt{2}\sqrt{\pi_l\overline{y}_l + \pi_l b}} \\
r &= \frac{\overline{y}_k + a}{\overline{y}_l + b}
\end{aligned}
$$

with constants $a = \sum_{i\in B'\cap F} \overline{y}_i$ and $b = \sum_{j\in S'\cap F} \overline{y}_j$.

The first and second roots occur when the limit price constraints are at their boundaries. The third root occurs when all B' orders are completely filled, and there is one order in S' only partially filled. Inversely, the fourth root occurs when all S' orders are completely filled, and there is one order in B' only partially filled. Finally, the fifth root occurs when all orders are totally filled.

1.5.2 Partitioning orders

It is implicit in the previous section that the sets of orders U, P, F that are unfilled, partial filled, or complete filled, are known in advance. Of course, this is not the case, since they are determined by the value of variable r we're interested in finding. The algorithm thus needs to make several guesses, fortunately not many:

Lemma 7. *The set of solutions to* $r^\star, \mathbf{y}^\star = \arg_{r,\mathbf{y}} \max f(r,\mathbf{y})$ *for an optimal exchange rate* $r^\star \in [a,b]$ *partition the set of orders into unfilled, partially filled, and completely filled in at most* $O(n)$ *different ways.*

Proof. Given $r^\star \in [a,b]$ orders can be split into B' and S' as described in the previous section. By lemma 2 it is known that all orders from one of the sides will be completely filled. From lemma 3 it is known that orders on the other side must partition into a subset of unfilled orders, at most one partial order, and a subset of filled orders. The same lemma shows that these subsets are ordered by increasing limit price of the order. There are at most $n-1$ orders on the each side, which determine the number of ways a side can be partitioned into. □

1.5.3 Complete algorithm description

Roughly, the algorithm for matching all orders of a token pair takes the following steps:

1. Sort orders by limit price π_i.
2. For each interval $[a,b]$ where a and b are consecutive limit exchange rates given by the limit prices of the orders (π_i for buy orders, $1/\pi_i$ for sell orders), and for each possible partition U,P,F of orders, compute and store r for local optima to the resulting optimization problem, as described in section 1.5.1.
3. For each collected r corresponding to a local optimum, compute the traded amounts as described in section 1.4, and evaluate the objective. Return solution corresponding to the global optimum.

Lemma 8. *The previously described algorithm runs in time $O\left(n^2 \log n\right)$, where n is the number of orders.*

Proof. There are at most $n-1$ limit exchange rate intervals to be considered. For each interval there are $O(n)$ partitions to consider according to lemma 7. The computation of the roots runs in time $O(1)$. Steps 1 and 2 thus runs in $O\left(n^2\right)$. The number of roots is proportional to the number of orders n (5 per interval). By lemma 4, executing orders for one root takes $O\left(n \log n\right)$. Therefore, step 3 runs in $O\left(n^2 \log n\right)$. □

1.6 Differences to the implementation

There are some significant differences between the exposition here and the implementation:

- The implementation also correctly handles the fee. This would make the formulas here terribly more complex.
- The implementation skips many intervals and combinations of U,P,F order subsets that can be proven suboptimal efficiently.
- The implementation also handles side constraints: maximum number of orders, minimum tradable amount, and economic viability. The don't change the worst case runtime complexity of the algorithm except for the latter - increases it by a factor of n. Unfortunately, with these constraints the solutions obtained are no longer guaranteed to be optimal.

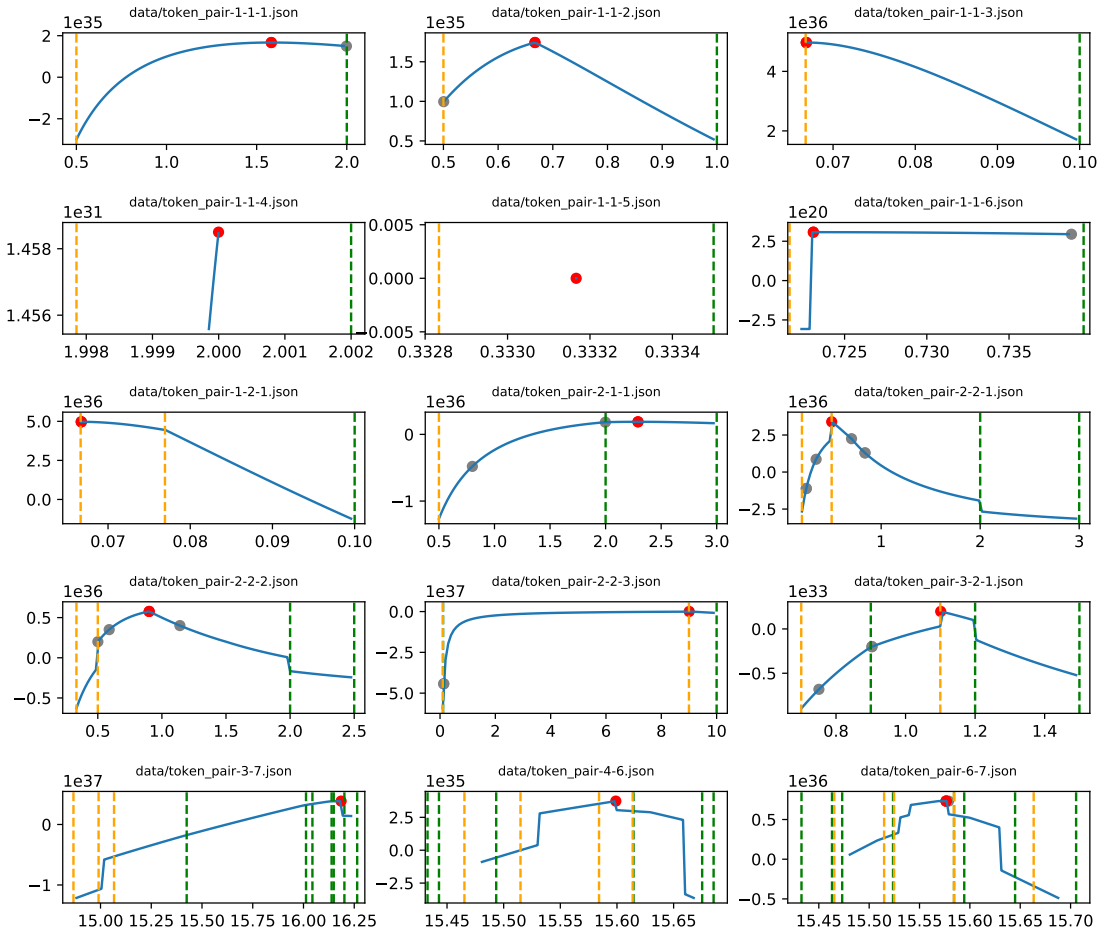

Figure 1.2: All examples in "repo/data" folder. The red dot is the global optimum, the grey dots are the local optima that were actually computed and evaluated (those that were trivially excluded are not shown). Note also that these examples also take the fee into consideration.

1.7 Open questions

It seems that there is always exactly one local optimum, which is the global optimum, in all instances of this problem we've inspected so far. Is this true in general? How to prove it?

1.8 All examples

There are a handfull of examples in the "repo/data" folder, which were used to verify the math above and the code (fig. 1.2).