# An Efficient Algorithm for Optimal Routing Through Constant Function Market Makers

Theo Diamandis[1], Max Resnick[2], Tarun Chitra[3], and Guillermo Angeris[4]

[1] Massachusetts Institute of Technology, tdiamand@mit.edu
[2] Risk Harbor, max@riskharbor.com
[3] Gauntlet Networks, tarun@gauntlet.network
[4] Bain Capital Crypto, gangeris@baincapital.com

**Abstract** Constant function market makers (CFMMs) such as Uniswap have facilitated trillions of dollars of digital asset trades and have billions of dollars of liquidity. One natural question is how to optimally route trades across a network of CFMMs in order to ensure the largest possible utility (as specified by a user). We present an efficient algorithm, based on a decomposition method, to solve the problem of optimally executing an order across a network of decentralized exchanges. The decomposition method, as a side effect, makes it simple to incorporate more complicated CFMMs, or even include 'aggregate CFMMs' (such as Uniswap v3), into the routing problem. Numerical results show significant performance improvements of this method, tested on realistic networks of CFMMs, when compared against an off-the-shelf commercial solver.

## Introduction

Decentralized Finance, or DeFi, has been one of the largest growth areas within both financial technologies and cryptocurrencies since 2019. DeFi is made up of a network of decentralized protocols that match buyers and sellers of digital goods in a trustless manner. Within DeFi, some of the most popular applications are decentralized exchanges (DEXs, for short) which allow users to permissionlessly trade assets. While there are many types of DEXs, the most popular form of exchange (by nearly any metric) is a mechanism known as the constant function market maker, or CFMM. A CFMM is a particular type of DEX which allows anyone to propose a trade (*e.g.*, trading some amount of one asset for another). The trade is accepted if a simple rule, which we describe later in §1.1, is met.

The prevalence of CFMMs on blockchains naturally leads to questions about routing trades across networks or aggregations of CFMMs. For instance, suppose that one wants to trade some amount of asset A for the greatest possible amount of asset B. There could be many 'routes' that provide this trade. For example, we may trade asset A for asset C, and only then trade asset C for asset B. This routing problem can be formulated as an optimization problem over the set of CFMMs available to the user for trading. Angeris et al. [Ang+22b] showed that the general problem of routing is a convex program for concave utilities, ignoring blockchain transactions costs, though special cases of the routing problem have been studied previously [Wan+22; DKP21].

*This paper.* In this paper, we apply a decomposition method to the optimal routing problem, which results in an algorithm that easily parallelizes across all DEXs. To solve the subproblems of the algorithm, we formalize the notions of swap markets, bounded liquidity, and aggregate CFMMs (such as Uniswap v3) and discuss their properties. Finally, we demonstrate that our algorithm for optimal routing is efficient, practical, and can handle the large variety of CFMMs that exist on chain today.

## 1 Optimal routing

In this section, we define the general problem of optimal routing and give concrete examples along with some basic properties.

*Assets.* In the optimal routing problem, we have a global labeling of $n$ assets which we are allowed to trade, indexed by $j = 1, \dots, n$ throughout this paper. We will sometimes refer to this 'global collection' as the *universe* of assets that we can trade.

*Trading sets.* Additionally, in this problem, we have a number of markets $i = 1, \dots, m$ (usually constant function market makers, or collections thereof, which we discuss in §1.1) which trade a subset of the universe of tokens of size $n_i$. We define market $i$'s behavior, at the time of the trade, via its *trading set* $T_i \subseteq \mathbb{R}^{n_i}$. This trading set behaves in the following way: any trader is able to propose a *trade* consisting of a basket of assets $\Delta_i \in \mathbb{R}^{n_i}$, where positive entries of $\Delta_i$ denote that the trader receives those tokens from the market, while negative values denote that the trader tenders those tokens to the market. (Note that the baskets here are of a subset of the universe of tokens which the market trades.) The market then *accepts* this trade (*i.e.*, takes the negative elements in $\Delta_i$ from the trader and gives the positive elements in $\Delta_i$ to the trader) whenever

$$\Delta_i \in T_i.$$

We make two assumptions about the sets $T_i$. One, that the set $T_i$ is a closed convex set, and, two, that the zero trade is always an acceptable trade, *i.e.*, $0 \in T_i$. All existing DEXs that are known to the authors have a trading set that satisfies these conditions.

*Local and global indexing.* Each market $i$ trades only a subset of $n_i$ tokens from the universe of tokens, so we introduce the matrices $A_i \in \mathbb{R}^{n \times n_i}$ to connect the local indices to the global indices. These matrices are defined such that $A_i \Delta_i$ yields the total amount of assets the trader tendered or received from market $i$, in the global indices. For example, if our universe has 3 tokens and market $i$ trades the tokens 2 and 3, then

$$A_i = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Written another way, $(A_i)_{jk} = 1$ if token $k$ in the market's local index corresponds to global token index $j$, and $(A_i)_{jk} = 0$ otherwise. We note that the ordering of tokens in the local index does not need to be the same as the global ordering.

*Network trade vector.* By summing the net trade in each market, after mapping the local indices to the global indices, we obtain the *network trade vector*

$$\Psi = \sum_{i=1}^{m} A_i \Delta_i.$$

We can interpret $\Psi$ as the net trade across the network of all markets. If $\Psi_i > 0$, we receive some amount of asset $i$ after executing all trades $\{\Delta_i\}_{i=1}^m$. On the other hand, if $\Psi_i < 0$, we tender some of asset $i$ to the network. Note that having $\Psi_i = 0$ does not imply we do not trade asset $i$; it only means that, after executing all trades, we received as much as we tendered.

*Network trade utility.* Now that we have defined the network trade vector, we introduce a utility function $U : \mathbb{R}^n \to \mathbb{R} \cup \{-\infty\}$ that gives the trader's utility of a net trade $\Psi$. We assume that $U$ is concave and increasing (*i.e.*, we assume all assets have value with potentially diminishing returns). Furthermore, we will use infinite values of $U$ to encode constraints; a trade $\Psi$ such that $U(\Psi) = -\infty$ is unacceptable to the trader. We can choose $U$ to encode several important actions in markets, including liquidating or purchasing a basket of assets and finding arbitrage. See [Ang+22a, §5.2] for several examples.

*Optimal routing problem.* The *optimal routing problem* is then the problem of finding a set of valid trades that maximizes the trader's utility:

$$\begin{array}{ll} \text{maximize} & U(\Psi) \\ \text{subject to} & \Psi = \displaystyle\sum_{i=1}^{m} A_i \Delta_i \\ & \Delta_i \in T_i, \qquad i = 1, \dots, m. \end{array} \tag{1}$$

The problem variables are the network trade vector $\Psi \in \mathbb{R}^n$ and trades with each market $\Delta_i \in \mathbb{R}^{n_i}$, while problem data are the utility function $U : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, the matrices $A_i \in \mathbb{R}^{n \times n_i}$, and the trading sets $T_i \subseteq \mathbb{R}^{n_i}$, where $i = 1, \dots, m$. Since the trading sets are convex and the utility function is concave, this problem is a convex optimization problem. In the subsequent sections, we will use basic results of convex optimization to construct an efficient algorithm to solve problem (1).

## 1.1 Constant function market makers

Most decentralized exchanges, such as Uniswap v2, Balancer, Curve, among others, are currently organized as *constant function market makers* (CFMMs, for short) or collections of CFMMs (such as Uniswap v3) [AC20; Ang+22a]. A constant function market maker is a type of permissionless market that allows anyone to trade baskets of, say, $r$, assets for other baskets of these same $s$ assets, subject to a simple set of rules which we describe below.

*Reserves and trading functions.* A constant function market maker, which allows $r$ tokens to be traded, is defined by two properties: its *reserves* $R \in \mathbb{R}^r_+$, where $R_j$ denotes the amount of asset $j$ available to the CFMM, and a *trading function* which is a concave function $\varphi : \mathbb{R}^r_+ \to \mathbb{R}$, which specifies the CFMM's behavior and its *trading fee* $0 < \gamma \leq 1$.

*Acceptance condition.* Any user is allowed to submit a trade to a CFMM, which is, from before, a vector $\Delta \in \mathbb{R}^r$. The submitted trade is then accepted if the following condition holds:

$$\varphi(R - \gamma\Delta_- - \Delta_+) \geq \varphi(R), \tag{2}$$

and $R - \gamma\Delta_- - \Delta_+ \geq 0$. Here, we denote $\Delta_+$ to be the 'elementwise positive part' of $\Delta$, *i.e.*, $(\Delta_+)_j = \max\{\Delta_j, 0\}$ and $\Delta_-$ to be the 'elementwise negative part' of $\Delta$, *i.e.*, $(\Delta_-)_j = \min\{\Delta_j, 0\}$ for every asset $j = 1, \dots, r$. The basket of assets $\Delta_+$ may sometimes be called the 'received basket' and $\Delta_-$ may sometimes be called the 'tendered basket' (see, *e.g.*, [Ang+22a]). Note that the trading set $T$, for a CFMM, is exactly the set of $\Delta$ such that (2) holds,

$$T = \{\Delta \in \mathbb{R}^r \mid \varphi(R - \gamma\Delta_- - \Delta_+) \geq \varphi(R)\}. \tag{3}$$

It is clear that $0 \in T$, and it is not difficult to show that $T$ is convex whenever $\varphi$ is concave, which is true for all trading functions used in practice. If the trade is accepted then the CFMM pays out $\Delta_+$ from its reserves and receives $-\Delta_-$ from the trader, which means the reserves are updated in the following way:

$$R \leftarrow R - \Delta_- - \Delta_+.$$

The acceptance condition (2) can then be interpreted as: the CFMM accepts a trade only when its trading function, evaluated on the 'post-trade' reserves with the tendered basket discounted by $\gamma$, is at least as large as its value when evaluated on the current reserves.

It can be additionally shown that the trade acceptance conditions in terms of the trading function $\varphi$ and in terms of the trading set $T$ are equivalent in the sense that every trading set has a function $\varphi$ which generates it [AC20], under some basic conditions.

*Examples.* Almost all examples of decentralized exchanges currently in production are constant function market makers. For example, the most popular trading function (as measured by most metrics) is the product trading function:

$$\varphi(R) = \sqrt{R_1 R_2},$$

originally proposed for Uniswap [ZCP18] and a 'bounded liquidity' variation of this function:

$$\varphi(R) = \sqrt{(R_1 + \alpha)(R_2 + \beta)}, \tag{4}$$

used in Uniswap v3 [Ada+21], with $\alpha, \beta \geq 0$. Other examples include the weighted geometric mean (as used by Balancer [MM19])

$$\varphi(R) = \prod_{i=1}^{r} R_i^{w_i}, \tag{5}$$

where $r$ is the number of assets the exchange trades, and $w \in \mathbb{R}^r_+$ with $\mathbf{1}^T w = 1$ are known as the weights, along with the Curve trading function

$$\varphi(R) = \alpha \mathbf{1}^T R - \left( \prod_{i=1}^{r} R_i^{-1} \right),$$

where $\alpha > 0$ is a parameter set by the CFMM [Ego]. Note that the 'product' trading function is the special case of the weighted geometric mean function when $r = 2$ and $w_1 = w_2 = 1/2$.

*Aggregate CFMMs.* In some special cases, such as in Uniswap v3, it is reasonable to consider an *aggregate CFMM*, which we define as a collection of CFMMs, which all trade the same assets, as part of a single 'big' trading set. A specific instance of an aggregate CFMM currently used in practice is in Uniswap v3 [Ada+21]. Any 'pool' in this exchange is actually a collection of CFMMs with the 'bounded liquidity' variation of the product trading function, shown in (4). We will see that we can treat these 'aggregate CFMMs' in a special way in order to significantly improve performance.

## 2 An efficient algorithm

A common way of solving problems such as problem (1), where we have a set of variables coupled by only a single constraint, is to use a decomposition method [DW60; Ber16]. The general idea of these methods is to solve the original problem by splitting it into a sequence of easy subproblems that can be solved independently. In this section, we will see that applying a decomposition method to the optimal routing problem gives a solution method which parallelizes over all markets. Furthermore, it gives a clean programmatic interface; we only need to be able to find arbitrage for a market, given a set of reference prices. This interface allows us to more easily include a number of important decentralized exchanges, such as Uniswap v3.

### 2.1 Dual decomposition

To apply the dual decomposition method, we first take the coupling constraint of problem (1),

$$\Psi = \sum_{i=1}^{m} A_i \Delta_i,$$

and relax it to a linear penalty in the objective, parametrized by some vector $\nu \in \mathbb{R}^n$. (We will show in §2.2 that the only reasonable choice of $\nu$ is a market clearing price, sometimes called a no-arbitrage price, and that this choice actually results in a relaxation that is tight; *i.e.*, a solution for this relaxation also satisfies the original coupling constraint.) This relaxation results in the following problem:

$$\begin{array}{ll} \text{maximize} & U(\Psi) - \nu^T(\Psi - \sum_{i=1}^m A_i \Delta_i) \\ \text{subject to} & \Delta_i \in T_i, \quad i = 1, \dots, m, \end{array}$$

where the variables are the network trade vector $\Psi \in \mathbb{R}^n$ and the trades are $\Delta_i \in \mathbb{R}^{n_i}$ for each market $i = 1, \dots, m$. Note that this formulation can be viewed as a family of problems parametrized by the vector $\nu$.

A simple observation is that this new problem is actually separable over all of its variables. We can see this by rearranging the objective:

$$\begin{array}{ll} \text{maximize} & U(\Psi) - \nu^T\Psi + \sum_{i=1}^m (A_i^T \nu)^T \Delta_i \\ \text{subject to} & \Delta_i \in T_i, \quad i = 1, \dots, m. \end{array} \tag{6}$$

Since there are no additional coupling constraints, we can solve for $\Psi$ and each of the $\Delta_i$ with $i = 1, \dots, m$ separately.

*Subproblems.* This method gives two types of subproblems, each depending on $\nu$. The first, over $\Psi$, is relatively simple:

$$\text{maximize} \quad U(\Psi) - \nu^T\Psi, \tag{7}$$

and can be recognized as a slightly transformed version of the Fenchel conjugate [BV04, §3.3]. We will write its optimal value (which depends on $\nu$) as

$$\bar{U}(\nu) = \sup_{\Psi} \left( U(\Psi) - \nu^T\Psi \right).$$

The function $\bar{U}$ can be easily derived in closed form for a number of functions $U$. Additionally, since $\bar{U}$ is a supremum over an affine family of functions parametrized by $\nu$, it is a convex function of $\nu$ [BV04, §3.2.3]. (We will use this fact soon.) Another important thing to note is that unless $\nu \geq 0$, the function $\bar{U}(\nu)$ will evaluate to $+\infty$. This can be interpreted as an implicit constraint on $\nu$.

The second type of problem is over each trade $\Delta_i$ for $i = 1, \dots, m$, and can be written, for each market $i$, as

$$\begin{array}{ll} \text{maximize} & (A_i^T \nu)^T \Delta_i \\ \text{subject to} & \Delta_i \in T_i. \end{array} \tag{8}$$

We will write its optimal value, which depends on $A_i^T \nu$, as $\mathbf{arb}_i(A_i^T \nu)$. Problem (8) can be recognized as the *optimal arbitrage problem* (see, *e.g.*, [Ang+22a]) for market $i$, when the external market price, or reference market price, is equal to $A_i^T \nu$. Since $\mathbf{arb}_i(A_i^T \nu)$ is also defined as a supremum over a family of affine functions of $\nu$, it too is a convex function of $\nu$. Solutions to the optimal arbitrage problem are known, in closed form, for a number of trading functions. (See appendix A for some examples.)

*Dual variables as prices.* The optimal solution to problem (8), given by $\Delta_i^\star$, is a point $\Delta_i^\star$ in $T_i$ such that there exists a supporting hyperplane to the set $T_i$ at $\Delta_i^\star$ with slope $A_i^T \nu$ [BV04, §5.6]. We can interpret these slopes as the 'marginal prices' of the $n_i$ assets, since, letting $\delta \in \mathbb{R}^{n_i}$ be a small deviation from the trade $\Delta_i^\star$, we have, writing $\tilde{\nu} = A_i^T \nu$ as the weights of $\nu$ in the local indexing:

$$\tilde{\nu}^T(\Delta_i^\star + \delta) \leq \tilde{\nu}^T \Delta_i^\star,$$

for every $\delta$ with $\Delta_i^\star + \delta \in T_i$. (By definition of optimality.) Canceling terms, we find:

$$\tilde{\nu}^T \delta \leq 0.$$

If, for example, $\delta_i$ and $\delta_j$ are the only two nonzero entries of $\delta$, we would have

$$\delta_i \leq -\frac{\tilde{\nu}_j}{\tilde{\nu}_i} \delta_j,$$

so the exchange rate between $i$ and $j$ is at most $\tilde{\nu}_i / \tilde{\nu}_j$. This observation lets us interpret the dual variables $\tilde{\nu}$ (and therefore the dual variables $\nu$) as 'marginal prices', up to a constant multiple.

## 2.2 The dual problem

The objective value of problem (6), which is a function of $\nu$, can then be written as

$$g(\nu) = \bar{U}(\nu) + \sum_{i=1}^m \mathbf{arb}_i(A_i^T \nu). \tag{9}$$

This function $g : \mathbb{R}^n \to \mathbb{R}$ is called the *dual function.* Since $g$ is the sum of convex functions, it too is convex. The *dual problem* is the problem of minimizing the dual function,

$$\text{minimize} \quad g(\nu), \tag{10}$$

over the dual variable $\nu \in \mathbb{R}^n$, which is a convex optimization problem since $g$ is a convex function.

*Dual optimality.* While we have defined the dual problem, we have not discussed how it relates to the original routing problem we are attempting to solve, problem (1). Let $\nu^\star$ be a solution to the dual problem (10). Assuming that the dual function is differentiable at $\nu^\star$, the first order, unconstrained optimality conditions for problem (10) are that

$$\nabla g(\nu^\star) = 0.$$

(The function $g$ need not be differentiable, in which case a similar, but more careful, argument holds using subgradient calculus.) It is not hard to show that if $\bar{U}$ is differentiable at $\nu^\star$, then its gradient must be $\nabla \bar{U}(\nu^\star) = -\Psi^\star$, where $\Psi^\star$ is the solution to the first subproblem (7), with $\nu^\star$. (This follows from the fact that the gradient of a maximum, when differentiable, is the gradient of the argmax.) Similarly, the gradient of $\mathbf{arb}_i$ when evaluated at $A_i^T \nu^\star$ is $\Delta_i^\star$, where $\Delta_i^\star$ is a solution to problem (8) with marginal prices $A_i^T \nu^\star$, for each market $i = 1, \dots, m$. Using the chain rule, we then have:

$$0 = \nabla g(\nu^\star) = -\Psi^\star + \sum_{i=1}^{m} A_i \Delta_i^\star. \tag{11}$$

Note that this is exactly the coupling constraint of problem (1). In other words, when the linear penalties $\nu^\star$ are chosen optimally (*i.e.*, chosen such that they minimize the dual problem (10)) then the optimal solutions for subproblems (7) and (8) automatically satisfy the coupling constraint. Because problem (6) is a relaxation of the original problem (1) for any choice of $\nu$, any solution to problem (6) that satisfies the coupling constraint of problem (1) must also be a solution to this original problem. All that remains is the question of finding a solution $\nu^\star$ to the dual problem (10).

### 2.3 Solving the dual problem

The dual problem (10) is a convex optimization problem that is easily solvable in practice, even for very large $n$ and $m$. In many cases, we can use a number of off-the-shelf solvers such as SCS [O'D+16], Hypatia [CKV21], and Mosek [ApS19]. For example, a relatively effective way of minimizing functions when the gradient is easily evaluated is the L-BFGS-B algorithm [Byr+95; Zhu+97; MN11]: given a way of evaluating the dual function $g(\nu)$ and its gradient $\nabla g(\nu)$ at some point $\nu$, the algorithm will find an optimal $\nu^\star$ fairly quickly in practice. (See §5 for timings.) By definition, the function $g$ is easy to evaluate if the subproblems (7) and (8) are easy to evaluate. Additionally the right hand side of equation (11) gives us a way of evaluating the gradient $\nabla g$, essentially for free, since we typically receive the optimal $\Psi^\star$ and $\Delta_i^\star$ as a consequence of computing $\bar{U}$ and $\mathbf{arb}_i$.

*Interface.* In order for a user to specify and solve the dual problem (10) (and therefore the original problem) it suffices for the user to specify (a) some way of evaluating $\bar{U}$ and its optimal $\Psi$ for problem (7) and (b) some way of evaluating the arbitrage problem (8) and its optimal trade $\Delta_i^\star$ for each market $i$ that the user wishes to include. New markets can be easily added by simply specifying how to arbitrage them, which, as we will see next, turns out to be straightforward for most practical decentralized exchanges. The Julia interface required for the software package described in §4 is a concretization of the interface described here.

## 3 Swap markets

In practice, most markets trade only two assets; we will refer to these kinds of markets as *swap markets*. Because these markets are so common, the performance of our algorithm is primarily governed by its ability to solve (8) quickly on these two asset markets. We show practical examples of these computations in appendix A. In this section, we will suppress the index $i$ with the understanding that we are referring to a specific market $i$.

## 3.1 General swap markets

Swap markets are simple to deal with because their trading behavior is completely specified by the *forward exchange function* [Ang+22a] for each of the two assets. In what follows, the forward trading function $f_1$ will denote the maximum amount of asset 2 that can be received by trading some fixed amount $\delta_1$ of asset 1, *i.e.*, if $T \subseteq \mathbb{R}^2$ is the trading set for a specific swap market, then

$$f_1(\delta_1) = \sup\{\lambda_2 \mid (-\delta_1, \lambda_2) \in T\}, \quad f_2(\delta_2) = \sup\{\lambda_1 \mid (\lambda_1, -\delta_2) \in T\}.$$

In other words, $f_1(\delta_1)$ is defined as the largest amount $\lambda_2$ of token 2 that one can receive for tendering a basket of $(\delta_1, 0)$ to the market. The forward trading function $f_2$ has a similar interpretation. If $f_1(\delta_1)$ is finite, then this supremum is achieved since the set $T$ is closed.

*Trading function.* If the set $T$ has a simple trading function representation, as in (3), it is not hard to show that the function $f_1$ is the unique (pointwise largest) function that satisfies

$$\varphi(R_1 + \gamma\delta_1, R_2 - f_1(\delta_1)) = \varphi(R_1, R_2). \tag{12}$$

whenever $\varphi$ is nondecreasing, which may be assumed for all CFMMs [AC20], and similarly for $f_2$. (Note the equality here, compared to the inequality in the original definition (2).)

*Properties.* The functions $f_1$ and $f_2$ are concave, since the trading set $T$ is convex, and nonnegative, since $0 \in T$ by assumption. Additionally, we can interpret the directional derivative of $f_j$ as the current marginal price of the received asset, denominated in the tendered asset. Specifically, we define

$$f_j'(\delta_j) = \lim_{h \to 0^+} \frac{f_j(\delta_j + h) - f_j(\delta_j)}{h}. \tag{13}$$

This derivative is sometimes referred to as the price impact function [ACE22]. Intuitively, $f_1'(0)$ is the current price of asset 1 quoted by the swap market before any trade is made, and $f_1'(\delta)$ is the price quoted by the market to add an additional $\varepsilon$ units of asset 1 to a trade of size $\delta$, for very small $\varepsilon$. We note that in the presence of fees, the marginal price to add to a trade of size $\delta$, *i.e.*, $f_1'(\delta)$, will be lower than the price to do so after the trade has been made [AC20].

*Swap market arbitrage problem.* Equipped with the forward exchange function, we can specialize (8). Overloading notation slightly by writing $(\nu_1, \nu_2) \geq 0$ for

$A_i^T\nu$ we define the swap market arbitrage problem for a market with forward exchange function $f_1$:

$$\begin{array}{ll} \text{maximize} & -\nu_1\delta_1 + \nu_2 f_1(\delta_1) \\ \text{subject to} & \delta_1 \geq 0, \end{array} \tag{14}$$

with variable $\delta_1 \in \mathbb{R}$ We can also define a similar arbitrage problem for $f_2$:

$$\begin{array}{ll} \text{maximize} & \nu_1 f_2(\delta_2) - \nu_2\delta_2 \\ \text{subject to} & \delta_2 \geq 0, \end{array}$$

with variable $\delta_2 \in \mathbb{R}$. Since $f_1$ and $f_2$ are concave, both problems are evidently convex optimization problems of one variable. Because they are scalar problems, these problems can be easily solved by bisection or ternary search. The final solution is to take whichever of these two problems has the largest objective value and return the pair in the correct order. For example, if the first problem (14) has the highest objective value with a solution $\delta_1^\star$, then $\Delta^\star = (-\delta_1^\star, f(\delta_1^\star))$ is a solution to the original arbitrage problem (8). (For many practical trading sets $T$, it can be shown that at most one problem will have strictly positive objective value, so it is possible to 'short-circuit' solving both problems if the first evaluation has positive optimal value.)

*Problem properties.* One way to view each of these problems is that they 'separate' the solution space of the original arbitrage problem (8) into two cases: one where an optimal solution $\Delta^\star$ for (8) has $\Delta_1^\star \leq 0$ and one where an optimal solution has $\Delta_2^\star \leq 0$. (Any optimal point $\Delta^\star$ for the original arbitrage problem (8) will never have both $\Delta_1^\star < 0$ and $\Delta_2^\star < 0$ as that would be strictly worse than the 0 trade for $\nu > 0$, and no reasonable market will have $\Delta_1^\star > 0$ and $\Delta_2^\star > 0$ since the market would be otherwise 'tendering free money' to the trader.) This observation means that, in order to find an optimal solution to the original optimal arbitrage problem (8), it suffices to solve two scalar convex optimization problems.

*Optimality conditions.* The optimality conditions for problem (14) are that, if

$$\nu_2 f_1'(0) \leq \nu_1 \tag{15}$$

then $\delta_1^\star = 0$ is a solution. Otherwise, we have

$$\delta_1^\star = \sup\{\delta \geq 0 \mid \nu_2 f_1'(\delta) \geq \nu_1\}.$$

Similar conditions hold for the problem over $\delta_2$. If the function $f_1'$ is continuous, not just semicontinuous, then the expression above simplifies to finding a root of a monotone function:

$$\nu_2 f_1'(\delta_1^\star) = \nu_1. \tag{16}$$

If there is no root and condition (15) does not hold, then $\delta_1^\star = \infty$. However, the solution will be finite for any trading set that does not contain a line, *i.e.*, the market does not have 'infinite liquidity' at a specific price.

*No-trade condition.* Note that using the inequality (15) gives us a simple way of verifying whether we will make any trade with market $T$, given some prices $\nu_1$ and $\nu_2$. In particular, the zero trade is optimal whenever

$$f_1'(0) \leq \frac{\nu_1}{\nu_2} \leq \frac{1}{f_2'(0)}.$$

We can view the interval $[f_1'(0), 1/f_2'(0)]$ as a type of 'bid-ask spread' for the market with trading set $T$. (In constant function market makers, this spread corresponds to the fee $\gamma$ taken from the trader.) This 'no-trade condition' lets us save potentially wasted effort of computing an optimal arbitrage trade as, in practice, most trades in the original problem will be 0.

*Bounded liquidity.* In some cases, we can easily check not only when a trade will not be made (say, using condition (15)), but also when the 'largest possible trade' will be made. (We will define what this means next.) Markets for which there is a 'largest possible trade' are called bounded liquidity markets. We say a market has *bounded liquidity in asset 2* if there is a finite $\delta_1$ such that $f_1(\delta_1) = \sup f_1$, and similarly for $f_2$. In other words, there is a finite input $\delta_1$ which will give the maximum possible amount of asset 2 out. A market has *bounded liquidity* if it has bounded liquidity on both of its assets. A bounded liquidity market then has a notion of a 'minimum price'. First, define

$$\delta_1^- = \inf\{\delta_1 \geq 0 \mid f_1(\delta_1) = \sup f_1\},$$

*i.e.*, $\delta_1^-$ is the smallest amount of asset 1 that can be tendered to receive the maximum amount the market is able to supply. We can then define the *minimum supported price* as the left derivative of $f_1$ at $\delta_1^-$:

$$f_1^-(\delta_1^-) = \lim_{h \to 0^+} \frac{f(\delta_1^-) - f(\delta_1^- - h)}{h}.$$

The first-order optimality conditions imply that $\delta_1^-$ is a solution to the scalar optimal arbitrage problem (14) whenever

$$f_1^-(\delta_1^-) \geq \frac{\nu_1}{\nu_2}.$$

In English, this can be stated as: if the minimum supported marginal price we receive for $\delta_1^-$ is still larger than the price being arbitraged against, $\nu_1/\nu_2$, it is optimal to take all available liquidity from the market. Using the same definitions for $f_2$, we find that the only time the full problem (14) needs to be solved is when the price being arbitraged against $\nu_1/\nu_2$ lies in the interval

$$f_1^-(\delta_1^-) < \frac{\nu_1}{\nu_2} < \frac{1}{f_2^-(\delta_2^-)}. \qquad (17)$$

(It may be the case that $f_2^-(\delta_2^-) = 0$ in which case we define the right hand side to be $\infty$.) We will call this interval of prices the *active interval* for a bounded liquidity market.

*Example.* In the case of Uniswap v3 [Ada+21], we have a collection of, say, $i = 1, \dots, s$ bounded liquidity product functions (4), where the parameters $\alpha_k, \beta_k > 0$ are chosen such that all of the active price intervals, as defined in (17), are disjoint. (An explicit form for this trading function is given in the appendix, equation (18).) Solving the arbitrage problem (14) over this collection of CFMMs is relatively simple. Since all of the intervals are disjoint, any price $\nu_1/\nu_2$ can lie in at most one of the active intervals. We therefore do not need to compute the optimal trade for any interval, except the single interval where $\nu_1/\nu_2$ lies, which can be done in closed form. We also note that this 'trick' applies to any collection of bounded liquidity markets with disjoint active price intervals.

## 4 Implementation

We implemented this algorithm in `CFMMRouter.jl`, a Julia [Bez+17] package for solving the optimal routing problem. Our implementation is available at

`https://github.com/bcc-research/CFMMRouter.jl`

and includes implementations for both weighted geometric mean CFMMs and Uniswap v3. In this section, we provide a concrete Julia interface for our solver.

### 4.1 Markets

*Market interface.* As discussed in §2.3, the only function that the user needs to implement to solve the routing problem for a given market is

`find_arb!(Δ, Λ, mkt, v).`

This function solves the optimal arbitrage problem (8) for a market `mkt` (which holds the relevant data about the trading set $T$) with dual variables `v` (corresponding to $A_i^T \nu$ in the original problem (8)). It then fills the vectors $\Delta$ and $\Lambda$ with the negative part of the solution, $-\Delta^\star_-$, and positive part of the solution, $\Delta^\star_+$, respectively.

For certain common markets (*e.g.*, geometric mean and Uniswap v3), we provide specialized, efficient implementations of `find_arb!`. For general CFMMs where the trading function, its gradient, and the Hessian are easy to evaluate, one can use a general-purpose primal-dual interior point solver. For other more complicated markets, a custom implementation may be required.

*Swap markets.* The discussion in §3 and the expression in (16) suggests a natural, minimal interface for swap markets. Specifically, we can define a swap market by implementing the function `get_price(Δ)`. This function takes in a vector of inputs $\Delta \in \mathbb{R}^2_+$, where we assume that only one of the two assets is being tendered, *i.e.*, `Δ₁Δ₂ == 0`, and returns $f_1'(\Delta_1)$, if $\Delta_1 > 0$ or $f_2'(\Delta_2)$ if $\Delta_2 > 0$. With this *price impact* function implemented, one can use bisection to compute the solution to (16). When price impact function has a closed form and is readily differentiable by hand, it is possible to use a much faster Newton method to solve this problem. In the case where the function does not have a simple closed form, we can use automatic differentiation (*e.g.*, using `ForwardDiff.jl` [RLP16]) to generate the gradients for this function.

*Aggregate CFMMs.* In the special case of aggregate, bounded liquidity CFMMs, the price impact function often does not have a closed form. On the other hand, whenever the active price intervals are disjoint, we can use the trick presented in §3.1 to quickly arbitrage an aggregate CFMM. For example, a number of Uniswap v3 markets are actually composed of many thousands of bounded liquidity CFMMs. Treating each of these as their own market, without any additional considerations, significantly increases the size and solution complexity of the problem.

In this special case, each aggregate market 'contains' $s$ trading sets, each of which has disjoint active price intervals with all others. We will write these intervals as $(p_i^-, p_i^+)$ for each trading set $i = 1, \dots, s$, and assume that these are in sorted order $p_{i-1}^+ \le p_i^- < p_i^+ \le p_{i+1}^+$. Given some dual variables $\nu_1$ and $\nu_2$ for which to solve the arbitrage problem (8), we can then run binary search over the sorted intervals (taking $O(\log(s))$ time) to find which of the intervals the price $\nu_1/\nu_2$ lies in. We can compute the optimal arbitrage for this 'active' trading set, and note that the remaining trading sets all have a known optimal trade (from the discussion in §3.1) and require only constant time. For Uniswap v3 and other aggregate CFMMs, this algorithm is much more efficient from both a computational and memory perspective when compared with a direct approach that considers all $s$ trading sets separately.

*Other functions.* If one is solving the arbitrage problem multiple times in a row, it may be helpful to implement the following additional functions:

1. `swap!(cfmm,` $\Delta$`)`: updates `cfmm`'s state following a trade $\Delta$.
2. `update_liquidity!(cfmm, [range,] L)`: adds some amount of liquidity `L` $\in \mathbb{R}_+^2$, optionally includes some interval `range = (p1, p2)`.

### 4.2 Utility functions.

Recall that the dual problem relies on a slightly transformed version of the Fenchel conjugate, which is the optimal value of problem (7). To use LBFGS-B (and most other optimization methods), we need to be able to evaluate this function $\bar{U}(\nu)$ and its gradient $\nabla\bar{U}(\nu)$, which is the solution $\Psi^\star$ to (7) with parameter $\nu$. Thus, utility functions are implemented as objects that implement the following interface:

- `f(objective, v)` evaluates $\bar{U}$ at `v`.
- `grad!(g, objective, v)` evaluates $\nabla\bar{U}$ at `v` and stores it in `g`.
- `lower_limit(objective)` returns the lower bound of the objective.
- `upper_limit(objective)` returns the upper bound of the objective.

The lower and upper bounds can be found by deriving the conjugate function. For example, for the 'total arbitrage' objective $U(\Psi) = c^T\Psi - I(\Psi \geq 0)$, where a trader wants to tender no tokens to the network, but receive any positive amounts out with value proportional to some nonnegative vector $c \in \mathbb{R}^n_+$, has $\bar{U}(\nu) = 0$ if $\nu \geq c$ and $\infty$ otherwise. Thus, we have the bounds $c \leq \nu < \infty$, and gradient $\nabla\bar{U}(\nu) = 0$. We provide implementations for arbitrage and for basket liquidations in our Julia package. (See [Ang+22b, §3] for definitions.)

## 5 Numerical results

We compare the performance of our solver against the commercial, off-the-shelf convex optimization solver Mosek, accessed through JuMP [DHL17; Leg+21]. In addition, we use our solver with real, on-chain data to illustrate the benefit of routing an order through multiple markets rather than trading with a single market. Our code is available at

`https://github.com/bcc-research/router-experiments.`

*Performance.* We first compare the performance of our solver against Mosek [ApS19], a widely-used, performant commercial convex optimization solver. We generate $m$ swap markets over a global universe of $2\sqrt{m}$ assets. Each market is randomly generated with reserves uniformly sampled from the interval between 1000 and 2000, denoted $R_i \sim \mathcal{U}(1000, 2000)$, and is a constant product market with probability 0.5 and a weighted geometric mean market with weights $(0.8, 0.2)$ otherwise. (These types of swap markets are common in protocols such as Balancer [MM19].) We run arbitrage over the set of markets, with 'true prices' for each asset randomly generated as $p_i \sim \mathcal{U}(0, 1)$. For each $m$, we use the same parameters (markets and price) for both our solver and Mosek. Mosek is configured with default parameters. All experiments are run on a MacBook Pro with a 2.3GHz 8-Core Intel i9 processor. In figure 1, we see that as the number of

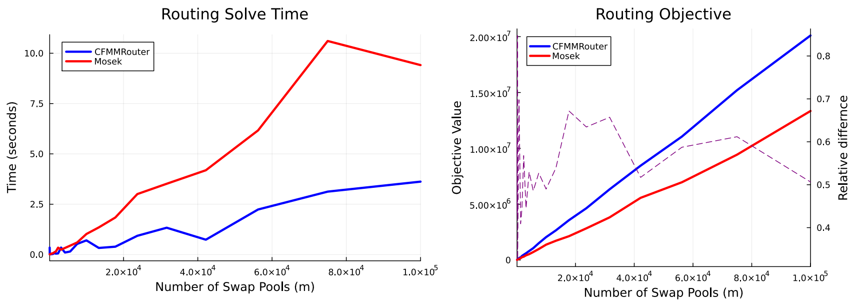

**Figure 1:** Solve time of Mosek vs. `CFMMRouter.jl` (left) and the resulting objective values for the arbitrage problem, with the dashed line indicating the relative increase in objective provided by our method (right).

pools (and tokens) grow, our method begins to dramatically outperform Mosek and scales quite a bit better. We note that the weighted geometric mean markets are especially hard for Mosek, as they must be solved as power cone constraints. Constant product markets may be represented as second order cone constraints, which are quite a bit more efficient for many solvers. Furthermore, our method gives a higher objective value, often by over 50%. We believe this increase stems from Mosek's use of an interior point method and numerical tolerances. The solution returned by Mosek for each market will be strictly inside the associated trading set, but we know that any rational trader will choose a trade on the boundary.

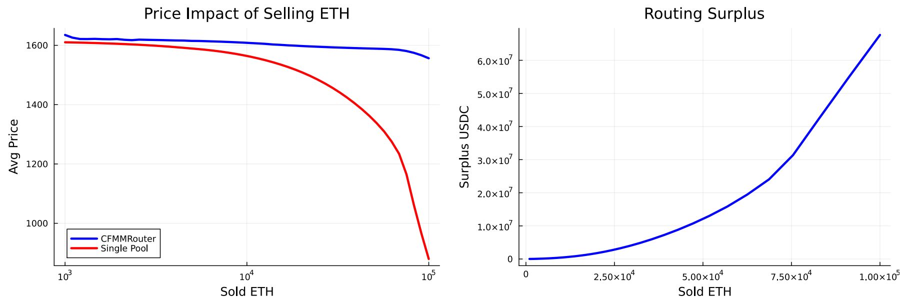

**Figure 2:** Average price of market sold ETH in routed vs. single-pool (left) and routed vs. single-pool surplus liquidation value (right).

*Real data: trading on chain.* We show the efficacy of routing by considering a swap from WETH to USDC (*i.e.*, using the basket liquidation objective to sell WETH for USDC). Using on-chain data from the end of a recent block, we show in figure 2 that as the trade size increases, routing through multiple pools gives an increasingly better average price than using the Uniswap v3 USDC-WETH .3% fee tier pool alone. Specifically, we route orders through the USDC-WETH .3%, WETH-USDT .3%, and USDC-USDT .01% pools. This is the simplest example in which we can hope to achieve improvements from routing, since two possible routes are available to the seller: a direct route through the USDC-WETH pool; and an indirect route that uses both the WETH-USDT pool and the USDC-USDT pool.

## 6 Conclusion

We constructed an efficient algorithm to solve the optimal routing problem. Our algorithm parallelizes across markets and involves solving a series of optimal arbitrage problems at each iteration. To facilitate efficient subproblem solutions, we introduced an interface for swap markets, which includes aggregate CFMMs.

We note that we implicitly assume that the trading sets are known exactly when the routing problem is solved. This assumption, however, ignores the realities of trading on chain: unless our trades execute first in the next block, we are not guaranteed that the trading sets for each market are the same as those in the last block. Transactions before ours in the new block may have changed prices (and reserves) of some of the markets we are routing through. This observation naturally suggests *robust routing* as a natural direction for future research. Furthermore, efficient algorithms for routing with fixed transaction costs (*e.g.*, gas costs) are another interesting direction for future work (see [Ang+22b, §5] for the problem formulation).

## Acknowledgements

We thank Francesco Iannelli and Jiahao Song for contributing to the package's documentation and the Financial Cryptography 2023 reviewers for helpful comments on this paper.

## References

[AC20] Guillermo Angeris and Tarun Chitra. "Improved Price Oracles: Constant Function Market Makers." In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. AFT '20: 2nd ACM Conference on Advances in Financial Technologies. New York NY USA: ACM, Oct. 21, 2020, pp. 80–91. ISBN: 978-1-4503-8139-0. DOI: `10.1145/3419614.3423251`. (Visited on 02/17/2021).

[ACE22] Guillermo Angeris, Tarun Chitra, and Alex Evans. "When Does The Tail Wag The Dog? Curvature and Market Making." In: *Cryptoeconomic Systems* 2.1 (June 2022). Ed. by Reuben Youngblom.

[Ada+21] Hayden Adams et al. "Uniswap v3 Core." In: (2021). URL: `https://uniswap.org/whitepaper-v3.pdf`.

[Ang+20] Guillermo Angeris et al. "An Analysis of Uniswap Markets." In: *Cryptoeconomic Systems* (Nov. 25, 2020). In collab. with Reuben Youngblom. DOI: `10.21428/58320208.c9738e64`. URL: `https://cryptoeconomicsystems.pubpub.org/pub/angeris-uniswap-analysis` (visited on 07/08/2021).

[Ang+22a] Guillermo Angeris et al. "Constant Function Market Makers: Multi-asset Trades via Convex Optimization." In: *Handbook on Blockchain*. Ed. by Duc A. Tran, My T. Thai, and Bhaskar Krishnamachari. Cham: Springer International Publishing, 2022, pp. 415–444. ISBN: 978-3-031-07535-3. DOI: `10.1007/978-3-031-07535-3_13`.

[Ang+22b] Guillermo Angeris et al. "Optimal routing for constant function market makers." In: *Proceedings of the 23rd ACM Conference on Economics and Computation*. 2022, pp. 115–128.

[ApS19] MOSEK ApS. *MOSEK Optimizer API for Python 9.1.5*. 2019. URL: `https://docs.mosek.com/9.1/pythonapi/index.html`.

[Ber16] Dimitri Bertsekas. *Nonlinear Programming*. Third edition. Belmont, Massachusetts: Athena Scientific, 2016. 861 pp. ISBN: 978-1-886529-05-2.

[Bez+17] Jeff Bezanson et al. "Julia: A Fresh Approach to Numerical Computing." In: *SIAM Review* 59.1 (Jan. 2017), pp. 65–98. ISSN: 0036-1445, 1095-7200. DOI: `10.1137/141000671`. URL: `https://epubs.siam.org/doi/10.1137/141000671` (visited on 01/06/2020).

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 1st ed. Cambridge, United Kingdom: Cambridge University Press, 2004. 716 pp. ISBN: 978-0-521-83378-3.

[Byr+95] Richard H Byrd et al. "A limited memory algorithm for bound constrained optimization." In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[CKV21] Chris Coey, Lea Kapelevich, and Juan Pablo Vielma. *Solving natural conic formulations with Hypatia.jl*. 2021. arXiv: `2005.01136 [math.OC]`.

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization." In: *SIAM Review* 59.2 (Jan. 2017), pp. 295–320. ISSN: 0036-1445, 1095-7200. DOI: `10.1137/15M1020575`. URL: `https://epubs.siam.org/doi/10.1137/15M1020575` (visited on 01/06/2020).

[DKP21] Vincent Danos, Hamza El Khalloufi, and Julien Prat. "Global Order Routing on Exchange Networks." In: *Financial Cryptography and Data Security. FC 2021 International Workshops*. Ed. by Matthew Bernhard et al. Vol. 12676. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021, pp. 207–226. ISBN: 978-3-662-63957-3 978-3-662-63958-0. DOI: `10.1007/978-3-662-63958-0_19`.

[DW60] George B Dantzig and Philip Wolfe. "Decomposition principle for linear programs." In: *Operations research* 8.1 (1960), pp. 101–111.

[Ego] Michael Egorov. "StableSwap - Efficient Mechanism for Stablecoin Liquidity." In: (), p. 6. URL: `https://www.curve.fi/stableswap-paper.pdf`.

[Leg+21] Benoît Legat et al. "MathOptInterface: A Data Structure for Mathematical Optimization Problems." In: *INFORMS Journal on Computing* (Oct. 22, 2021), ijoc.2021.1067. ISSN: 1091-9856, 1526-5528. DOI: `10.1287/ijoc.2021.1067`. URL: `http://pubsonline.informs.org/doi/10.1287/ijoc.2021.1067` (visited on 02/08/2022).

[MM19] Fernando Martinelli and Nikolai Mushegian. "Balancer: A Non-Custodial Portfolio Manager, Liquidity Provider, and Price Sensor." In: (2019).

[MN11] José Luis Morales and Jorge Nocedal. "Remark on "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization".” In: *ACM Transactions on Mathematical Software (TOMS)* 38.1 (2011), pp. 1–4.

[O'D+16] Brendan O'Donoghue et al. “Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding.” In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068. ISSN: 0022-3239, 1573-2878. DOI: `10.1007/s10957-016-0892-3`. URL: `http://link.springer.com/10.1007/s10957-016-0892-3` (visited on 10/30/2020).

[RLP16] J. Revels, M. Lubin, and T. Papamarkou. “Forward-Mode Automatic Differentiation in Julia.” In: *arXiv:1607.07892 [cs.MS]* (2016). URL: `https://arxiv.org/abs/1607.07892`.

[Wan+22] Ye Wang et al. “Cyclic Arbitrage in Decentralized Exchanges.” In: *Companion Proceedings of the Web Conference 2022*. Virtual Event, Lyon France: ACM, Apr. 2022, pp. 12–19. ISBN: 978-1-4503-9130-6. DOI: `10.1145/3487553.3524201`.

[ZCP18] Yi Zhang, Xiaohong Chen, and Daejun Park. “Formal Specification of Constant Product (Xy=k) Market Maker Model and Implementation.” In: (2018).

[Zhu+97] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization.” In: *ACM Transactions on mathematical software (TOMS)* 23.4 (1997), pp. 550–560.

## A Closed form solutions

Here, we cover some of the special cases where it is possible to analytically write down the solutions to the arbitrage problems presented previously.

*Geometric mean trading function.* Some of the most popular swap markets, for example, Uniswap v2 and most Balancer pools, which total over \$2B in reserves, are geometric mean markets (5) with $n = 2$. This trading function can be written as

$$\varphi(R) = R_1^w R_2^{1-w},$$

where $0 < w < 1$ is a fixed parameter. This very common trading function admits a closed-form solution to the arbitrage problem (8). Using (12), we can write

$$f_1(\delta_1) = R_2 \left(1 - \left(\frac{1}{1 + \gamma\delta_1/R_1}\right)^\eta\right)$$

where $\eta = w/(1-w)$. (A similar equation holds for $f_2$.) Using (15) and (16), and defining

$$\delta_1 = \frac{R_1}{\gamma}\left(\left(\eta\gamma\frac{\nu_2}{\nu_1}\frac{R_2}{R_1}\right)^{1/(\eta+1)} - 1\right),$$

we have that $\delta_1^\star = \max\{\delta_1, 0\}$ is an optimal point for (14). Note that when we take $w = 1/2$ then $\eta = 1$ and we recover the optimal arbitrage for Uniswap given in [Ang+20, App. A].

*Bounded liquidity variation.* The bounded liquidity variation (4) of the product trading function satisfies the definition of bounded liquidity given in §3.1, whenever $\alpha, \beta > 0$. We can write the forward exchange function for the bounded liquidity product function (4), using (12), as

$$f_1(\delta) = \min\left\{R_2, \frac{\gamma\delta(R_2+\beta)}{R_1+\gamma\delta+\alpha}\right\}$$

The 'min' here comes from the definition of a CFMM: it will not accept trades which pay out more than the available reserves. The maximum amount that a user can trade with this market, which we will write as $\delta_1^-$, is when $f_1(\delta_1^-) = R_2$, *i.e.*,

$$\delta_1^- = \frac{1}{\gamma}\frac{R_2}{\beta}(R_1+\alpha).$$

(Note that this can also be derived by taking $f_1(\delta_1) = R_2$ in (12) with the invariant (4).) This means that

$$f_1^-(\delta_1^-) = \gamma\frac{\beta^2}{(R_1+\alpha)(R_2+\beta)},$$

is the minimum supported price for asset 1. As before, a similar derivation yields the case for asset 2. Writing $k = (R_1+\alpha)(R_2+\beta)$, we see that we only need to solve (14) if the price $\nu_1/\nu_2$ is in the active interval (17),

$$\frac{\gamma\beta^2}{k} < \frac{\nu_1}{\nu_2} < \frac{k}{\gamma\alpha^2}. \tag{18}$$

Otherwise, we know one of the two 'boundary' solutions, $\delta_1^-$ or $\delta_2^-$, suffices.