

Closed-form solutions for generic N -token AMM arbitrage

Matthew Willetts and Christian Harrington

QuantAMM.fi

February 2024

Abstract

Convex optimisation has provided a mechanism to determine arbitrage trades on automated market makers (AMMs) since almost their inception. Here we outline generic closed-form solutions for N -token geometric mean market maker pool arbitrage, that in simulation (with synthetic and historic data) provide better arbitrage opportunities than convex optimisers and is able to capitalise on those opportunities sooner. Furthermore, the intrinsic parallelism of the proposed approach (unlike convex optimisation) offers the ability to scale on GPUs, opening up a new approach to AMM modelling by offering an alternative to numerical-solver-based methods. The lower computational cost of running this new mechanism can also enable on-chain arbitrage bots for multi-asset pools.

1 Introduction

Automated market makers (AMMs), of which most decentralized exchanges (DEXs) are examples, are some of the most important Decentralized Finance (DeFi) infrastructure this is. In these systems Liquidity providers deposit assets (LPs) that is used to enable traders to exchange tokens against the deposited capital. These systems provide close-to-market prices in their trades as there is an economic incentive for them to do so: quoted prices depend inversely on the pool's reserves, meaning that if the quoted price for an asset is too high a trader can gain by extracting that asset out of the pool (exchanging something else in) and vice versa if the quoted price is too low the trader can gain from trading that asset out of the pool. These trades are *arbitrage trades*, are carried out by the pool's arbitrageurs, and have the effect of bringing the pool's quoted prices closer to the current market prices.

We are interested in modelling the arbitrage opportunities that arise on multi-token (i.e. $N > 2$) geometric mean market makers, in particular finding the optimum trade for the arbitrageur to perform to maximise their return at current market prices. This takes us beyond swaps, as in general the arbitrage opportunity requires the trading of multiple different assets into or out of the pool, or both.

This problem can be attacked numerically using linear programming/convex optimisation [1]. Here we take a different approach, deriving closed-form analytical expressions for the optimal arbitrage trade. Fundamentally, these closed-form expressions for the optimal multi-asset arbitrage trade enable new and fine-grained mathematical analysis of the properties of the trade. These closed form expressions are particularly useful in the context of Temporal-function Market Making (TFMM) [2], where pools' weights change with time, as they enable modelling of the arbitrage trade (and resulting changes in pool reserves) *through which one can take gradients for the purpose of optimisation*.

Unlike a linear program, these closed form expressions require an *a priori* known amount of arithmetic operations to calculate, and for reasonable numbers of assets ($N \lesssim 7$) are substantially quicker to perform than convex optimisation (see Figure 2).

2 Optimal Multi-Asset Trades in the presence of fees

The pool contains N tokens, so a trader has a vector Δ of token amounts being traded to the pool for another set of tokens Λ , where entries in each are ≥ 0 . Where $\Delta_i > 0$, $\Lambda_i = 0$: tokens cannot be traded for themselves.¹ With fees $(1 - \gamma)$, for a trade to be accepted it must be the case that

$$\prod_{i=1}^N (R_i + \gamma \Delta_i - \Lambda_i)^{w_i} \geq k. \quad (1)$$

where $\mathbf{R} = \{R_i\}_{i=1,\dots,N}$, $\mathbf{w} = \{w_i\}_{i=1,\dots,N}$, $\forall i, 0 < w_i < 1$, $\sum_{i=1}^N w_i = 1$ and $k = \prod_{i=1}^N R_i^{w_i}$. While any trade that increases $\prod_{i=1}^N (R_i + \gamma \Delta_i - \Lambda_i)^{w_i}$ above k is accepted, a trader does better if their trade maintain k .² In constructing the optimal arbitrage trade, first we have to know *which* tokens to trade in and *which* to withdraw. We will return to that question in Section 3.2.

It is notationally simpler if we combine the inwards and outwards trade legs into one variable $\Phi := \Delta - \Lambda$. If token i is not being traded, $\Phi_i = 0$. We introduce a *trade signature* \mathbf{s} of length N , $s_i \in \{-1, 0, 1\}$ which encodes whether a token is being extracted from the pool, not being traded, or is being added to the pool. We can index over the *active* tokens in the pool, those with $s_i \neq 0$. We define the set $\mathcal{A} = \{i \in [N] | s_i \neq 0\}$, and we define an active-token normalised set of weights $\check{w}_i := w_i / \sum_{j \in \mathcal{A}} w_j$. We also introduce an auxiliary variable $\mathbf{d} = \mathbb{I}_{\mathbf{s}=1}$ for application of fees.³

We can rewrite Eq (1) as

$$\prod_{i \in \mathcal{A}} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{\check{w}_i} = 1, \quad (2)$$

a reduced form of the trading function (Appendix B). The we can form the multi-asset objective as

$$\mathcal{L} = - \sum_{i \in \mathcal{A}} (m_{p,i} \Phi_i) - \lambda \left(\prod_{j \in \mathcal{A}} \left(1 + \frac{\gamma^{d_j} \Phi_j}{R_j}\right)^{\check{w}_j} - 1 \right). \quad (3)$$

Taking partial derivatives w.r.t. Φ_i and setting to 0

$$\frac{\partial \mathcal{L}}{\partial \Phi_i} = -m_{p,i} - \lambda \frac{\check{w}_i \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{\check{w}_i} \gamma^{d_i} \prod_{j \in \mathcal{A}, j \neq i} \left(1 + \frac{\gamma^{d_j} \Phi_j}{R_j}\right)^{\check{w}_j}}{\left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)} = 0 \quad (4)$$

$$\Rightarrow -m_{p,i} - \lambda \frac{\check{w}_i \gamma^{d_i} \prod_{j=1}^N \left(1 + \frac{\gamma^{d_j} \Phi_j}{R_j}\right)^{\check{w}_j}}{\left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)} = 0 \quad (5)$$

$$\Rightarrow \lambda = - \frac{m_{p,i} R_i \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)}{\check{w}_i \gamma^{d_i}} \quad (6)$$

This gives us $|\mathcal{A}|$ non-degenerate equations each solving for λ . Putting any two into equality,

$$\frac{m_{p,i} R_i \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)}{\check{w}_i \gamma^{d_i}} = \frac{m_{p,j} R_j \left(1 + \frac{\gamma^{d_j} \Phi_j}{R_j}\right)}{\check{w}_j \gamma^{d_j}} \quad (7)$$

$$\Rightarrow \left(1 + \frac{\gamma^{d_j} \Phi_j}{R_j}\right) = \frac{\check{w}_j \gamma^{d_j}}{\check{w}_i \gamma^{d_i}} \frac{m_{p,i} R_i}{m_{p,j} R_j} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) \quad (8)$$

¹Equivalently, $\exists N \in \mathbb{N}, N \geq 2 : \forall \Lambda \in (\mathbb{R}_0^+)^N, \Delta \in (\mathbb{R}_0^+)^N, \Delta \cdot \Lambda = 0$.

²The inequality trading constraint is of use, however, as it enables convex optimisation (which demanding equality does not), with the obtained solution lying on the level set for which equality is maintained [1].

³This turns our problem into a mixed-integer convex program if we have our trading-function invariance imposed via an inequality, c.f. the above footnote.

We can now sub in Eq (8) to Eq (2) for all $j \neq i$ and rearrange (Appendix A) to obtain

$$\forall i \in \mathcal{A}, \Phi_i = \gamma^{-d_i} \left(\check{k} \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i}} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j}}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} - R_i \right), \quad (9)$$

where $\check{k} = \prod_{i \in \mathcal{A}} R_i^{\check{w}_i}$. This gives the optimal arbitrage trade for a multi-token/basket trade for geometric mean market makers.

Comment on derivation When first looking at these problems of optimal trades, it can seem that closed-form solutions are not possible as we have to impose inequality constraints on the trade (e.g. that $\Phi_i > 0$ where $s_i = 1$ in Eq (2)) meaning we have (Karush–Kuhn–Tucker) KKT conditions.

Our derivations works because a) we externally chose the trade signature, b) the signature allows for tokens to be untouched in the trade and c) the solutions to the full problem satisfy complementary slackness. Together these mean that we do not have to follow a KKT formulation, instead we get closed-form expressions via Lagrange multipliers.

Finally, note that in the case that fees are not present, the modelling task is much simpler [2]. Various approaches and results can then be layered on top of the zero-fee-case model (e.g. the bound in [3]). Here we are interested in full treatment of the effect of fees.

2.1 Finding the optimal trade signature

To use our results, it seems we need *a priori* the arbitrage trade’s signature \mathbf{s} . There is a simple fix: run through all possible variants of \mathbf{s} calculating $\Phi(\mathbf{s})$, and for each $\Phi(\mathbf{s})$ calculate its return to the arbitrageur $-\sum_{i=1}^N (m_{p,i} \Phi_i)$ and check that it fulfils the trade invariant Eq (2). The best such trade is then the optimal. If no checked trade fulfils Eq (2) while having a return > 0 , the best trade is $\Phi = 0$ and we are in the no-arb region.

This computationally-intensive parts of this approach (calculating $\Phi(\mathbf{s})$, $-\sum_{i=1}^N (m_{p,i} \Phi_i)$ and checking Eq (2)) are embarrassingly-parallel over different settings of \mathbf{s} .

What are the sets of \mathbf{s} that one has to check for a given pool size N ? We denote set of valid trade signatures $\mathcal{S}(N)$. The upper limit for $|\mathcal{S}(N)|$, the number of signatures a pool can have, is 3^N (each entry s_i can be one of $\{-1, 0, 1\}$), but not all combinations are valid. A signature \mathbf{s} has to have at least one entry of 1 and at least one -1 (at least one token has to be traded for at least one other token). For pools with $N < 6$ or so, these restrictions mean the number of valid signatures is $< 80\%$ of the naive 3^N calculation. We plot $|\mathcal{S}(N)|$ as a fraction of 3^N in Figure 1. As N increases, this ratio tends to one. For reference, $|\mathcal{S}(N = 3)| = 12$ and $|\mathcal{S}(N = 4)| = 50$.

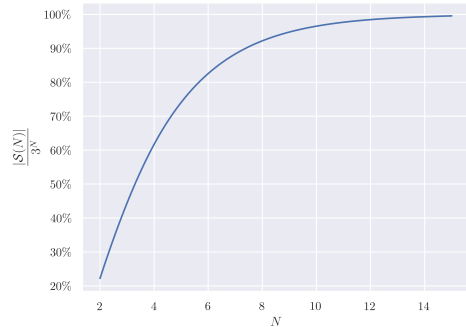


Figure 1: Number of valid trade signatures as a fraction of all possible unconstrained signatures.

Another possibility is create a computationally-less-intensive heuristic we can use to calculate the trade signature/trade direction—though for low N -token pool running through all variants is still considerably faster than convex optimisation even prior to any other speedups. An example of such a heuristic is provided in Appendix A.1.

3 Simulation results

Fundamentally, arbitrage trades are how AMM pools' reserves update. In modelling our results in silico and benchmarking against existing (numerical) approaches we are interested in both the run time of our approach and in how high-fidelity it is in finding arb opportunities. Both these areas matter whether one is interested in using our approach for simulating AMM pool rebalancing (for example in a backtest) or for using it to find and act on arbitrage opportunities in real time.

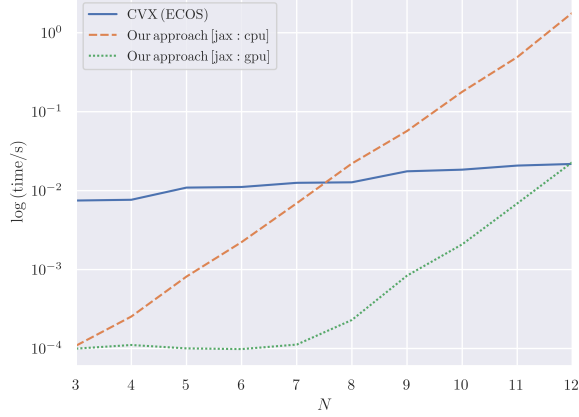


Figure 2: Logarithmic plot of the time taken to calculate an optimal trade as a function of number of pool assets N . All experiments were run in python, using CVXPY for convex optimisation. We used a MacBook Air M2 for all CPU results and a workstation with a GPU with compute capability 8.9.

3.1 Run time

For large values of N , standard convex solvers scale better than the closed form solutions, given the naive brute force methodology for finding the signature of the optimal trade requires roughly 3^N computations (though new heuristics may remove the advantage). As demonstrated in Figure 2, the scalability of the closed form solution can be drastically helped because unlike current convex solutions, it can be parallelised (for both CPUs and GPUs). While this does only delay the point at which convex solvers become faster, even with a single GPU it delays the point at an N -token number that is rarely, if at all, seen in contemporary AMMs.

3.2 Effectiveness

We create simulated, independent, potential arbitrage opportunities. For each we start with a pool initially at equilibrium (pool prices match market prices) and then randomly sample new market prices that deviate slightly, which have a chance of pushing the pool out of the no-arb region.

$$m_{p,i} \sim \text{Uniform}(0, 1), \quad \mathbf{R}_{\text{initial}} = V_0 \frac{\mathbf{w}}{\mathbf{m}_p}, \quad u_i \sim \text{Uniform}(0, 1), \quad \mathbf{m}_p^* = \mathbf{m}_p + a\mathbf{u},$$

where V_0 is the initial value of the pool in the numéraire of \mathbf{m}_p , each trial's \mathbf{w} is chosen to be close to uniform weights $w_i = \frac{1}{N}$ (as convex optimisation is less stable when \mathbf{w} is close to one-hot) and $\gamma = 0.05$.

We perform 120,000 independent random trials, and for each record the result of convex optimisation and our approach. Figure 3 compares the N -token closed to convex results. This demonstrate the superiority of the N -token approach across pools to find arbitrage opportunities.

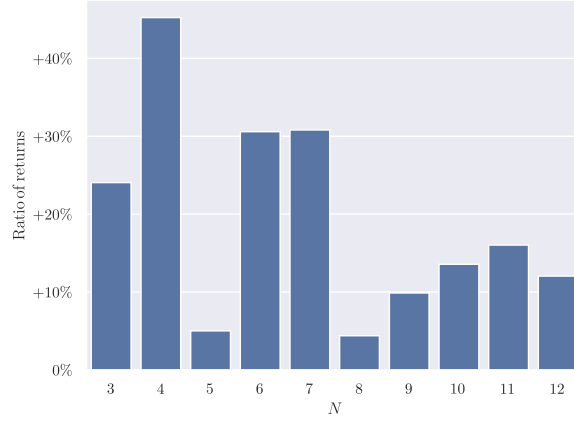


Figure 3: Comparing arbitrage returns from our approach as a function of the number of pool assets against numerical convex optimisation.

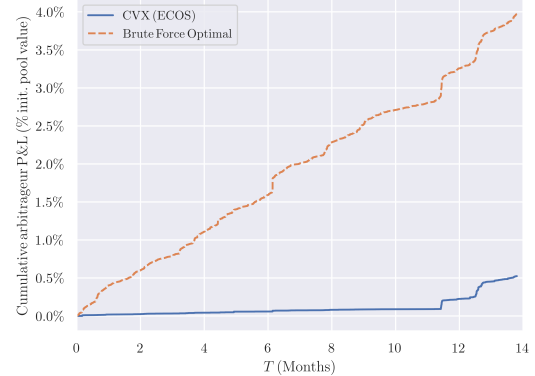
3.3 Duelling arbitrageurs

If you run multiple separate simulations of an N -token pool over time, in each updating the pool according to the arbitrage trades produced by a particular arbitrage algorithm, the algorithm that is *less good* at constructing arbitrage trades will often show *greater profit*. Why is this unexpected result found? In a less-than-ideal system, the arbitrage opportunity is only detected and acted on further away from the no-arb region of the pool.

As there is no competition between arbitrageurs (we are comparing separate runs), the less-good algorithm wins by ‘farming’ the pool, letting the arbitrage opportunity build up and up before eventually being reaped.



(a) Asset prices for the basket over time (plotted indexed to initial prices).



(b) Cumulative arbitrageur profits as a fraction of initial pool value.

Figure 4: Comparing arbitrage returns from competition between a simulated arbitrageur our approach and one using numerical solutions from convex optimisation. After every new market price comes in the convex optimiser arbitrageur gets priority (for free) in trading with the pool yet performs worse.

So in comparing multiple arbitrage algorithms over time we have to have them compete. Here we compare our closed-form approach to convex optimisation (performed using CVXPY) on a historical backtest of a CFMM pool with $N = 3$ run on a basket of ETH, BTC and DAI with uniform weights $w_i = \frac{1}{3}$ from June 2021 to July 2022. $\gamma = 0.003$ and initial pool value was 1m USD. We plot the results in Figure 4.

To give the convex-solver-arbitrageur an edge over the closed form algorithm, the convex-solver-arb is provided the top transaction in the block every time, for free. The closed form algorithm reaps the arbitrage opportunity a considerable number of times before convex-optimisation arbitrageur can.

4 Conclusions

We have described the closed-form expressions for the optimal arbitrage trade for N -token pools. In a range of experiments we have demonstrated the speed and arbitrage performance advantages of this new closed-form approach over current approaches.

Coupling this with the ability to run the approach at even higher speeds on GPUs (due to its parallelisability), and to use take gradients through the process (enabling more advanced machine learning techniques for AMM strategy training [2]) means that this approach may find use both for practitioners interested in performing arbitrage trades as well as those interested primarily in simulating & modelling AMM pools.

There are on-chain implications too that are significant, given how this technique could be used not only for on-chain arbitrage bots that have the direct ability to structure the arbitrage trade (via access to the current state of the pool) in the same transaction that the trade is performed, but also for DEX order routers and DEX aggregator routers. TSTORE advances, intelligent caching, and also signature detection heuristics being developed could each significantly reduce the computational cost of obtaining such a solution on-chain.

References

- [1] Guillermo Angeris, Akshay Agrawal, Alex Evans, Tarun Chitra, and Stephen Boyd. Constant function market makers: Multi-asset trades via convex optimization. 2021.
- [2] QuantAMM team. Temporal-function market making, 2023.
- [3] Guillermo Angeris, Alex Evans, and Tarun Chitra. When does the tail wag the dog? curvature and market making. 2020.
- [4] QuantAMM team. Temporal-function market making litepaper, 2023.

Technical Appendix

A Derivation of multi-asset optimal arbitrage trade

We sub in Eq (8) to Eq (2) for all $j \neq i, i, j \in \mathcal{A}$:

$$1 = \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{\check{w}_j \gamma^{d_j}}{\check{w}_i \gamma^{d_i}} \frac{m_{p,i} R_i}{m_{p,j} R_j} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) \right)^{\check{w}_j} \quad (\text{A.10})$$

$$\Rightarrow 1 = \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,i} R_i}{\check{w}_i \gamma^{d_i}} \right)^{\check{w}_j} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{\check{w}_j \gamma^{d_j}}{m_{p,j} R_j} \right)^{\check{w}_j} \quad (\text{A.11})$$

$$\Rightarrow 1 = \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) \left(\frac{m_{p,i} R_i}{\check{w}_i \gamma^{d_i}} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{\check{w}_j \gamma^{d_j}}{m_{p,j} R_j} \right)^{\check{w}_j} \quad (\text{A.12})$$

$$\Rightarrow \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) = \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i} R_i} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j} R_j}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} \quad (\text{A.13})$$

$$\Rightarrow \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) = \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i} R_i} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j} R_j}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} \quad (\text{A.14})$$

$$\Rightarrow \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) = \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i} R_i} \right)^{1-\check{w}_i} \frac{\check{k}}{R_i^{\check{w}_i}} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j}}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} \quad (\text{A.15})$$

$$\Rightarrow \frac{\gamma^{d_i} \Phi_i}{R_i} = \frac{\check{k}}{R_i} \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i}} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j}}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} - 1 \quad (\text{A.16})$$

$$\Rightarrow \forall i \in \mathcal{A}, \Phi_i = \gamma^{-d_i} \left(\check{k} \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i}} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j}}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} - R_i \right), \quad (\text{A.17})$$

where $\check{k} := \prod_{i \in \mathcal{A}} R_i^{\check{w}_i}$. There is also a slightly more symmetric form, which however is slightly less appropriate for implementation in a resource-constrained setting:

$$\forall i \in \mathcal{A}, \Phi_i = \gamma^{-d_i} R_i \left(\left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i} R_i} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j} R_j}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} - 1 \right) \quad (\text{A.18})$$

Amplified Liquidity This approach also naturally extends to multi-token pools that make use of amplified liquidity [4]. There a pool's *virtual reserves* are $\check{R}_i = \nu R_i$ where ν is a (potentially varying from block to block) scalar and the trading invariant is that $\prod_i (\nu R_i + \gamma \Delta_i - \Lambda)^{w_i} = \nu \prod_i (R_i)^{w_i}$. Applying the mapping $R_i \rightarrow \nu R_i$ to Eq (A.17), we get the optimal arbitrage trade for these pools:

$$\forall i \in \mathcal{A}, \Phi_i = \gamma^{-d_i} \left(\check{\nu} \check{k} \left(\frac{\check{w}_i \gamma^{d_i}}{m_{p,i}} \right)^{1-\check{w}_i} \prod_{\substack{j \neq i \\ j \in \mathcal{A}}} \left(\frac{m_{p,j}}{\check{w}_j \gamma^{d_j}} \right)^{\check{w}_j} - \nu R_i \right), \quad (\text{A.19})$$

where $\check{\nu} := \nu^{\sum_{i \in \mathcal{A}} \check{w}_i}$, with the analogue of Eq (A.18) similarly following.

A.1 Heuristic for Trade Signature

A heuristic that works reasonably well for finding a workable trade signature \mathbf{s} in the close vicinity of the no-arb-region boundary is to look for, in effect, swap-level arb opportunities between each pair within the pool and use those to form \mathbf{s} . Do as follows:

Find the zeros Construct the ratio between the (zero-fees) quoted prices of the pool (in the same numéraire as the market prices \mathbf{m}_p) and the market prices: $\ell := V \frac{\mathbf{w}}{\mathbf{R}\mathbf{m}_p}$ (where all operations are done elementwise) and $V = \sum_{i=1}^N m_{p,i} R_i$. Then construct the ‘price quotient matrix’ Γ , where each entry is $\Gamma_{i,j} = \ell_i / \ell_j$. For any tokens i for which $\sum_{j=1}^N \mathbb{I}_{\Gamma_{i,j} > \gamma^{-1}} = \sum_{j=1}^N \mathbb{I}_{\Gamma_{i,j} < \gamma} = 0$, $s_i = 0$.

Active trade direction For tokens i with $s_i \neq 0$, simply have $s_i = 1$ if $\ell_i > 1$ and $s_i = -1$ if $\ell_i < 1$.

B Reduced form of Trading Function for G3Ms

Starting with Eq (1) and making the substitutions from $\{\Delta, \Lambda\}$ to $\{\Phi, \mathbf{d}\}$:

$$\prod_{i=1}^N (R_i + \gamma^{d_i} \Phi_i)^{w_i} \geq k. \quad (\text{B.20})$$

The most profitable trade will be when equality is reached. Divide by $\prod_{i=1}^N (R_i)^{w_i} = k$ to obtain

$$\prod_{i=1}^N \frac{(R_i + \gamma^{d_i} \Phi_i)^{w_i}}{(R_i)^{w_i}} = 1 \quad (\text{B.21})$$

$$\prod_{i=1}^N \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{w_i} = 1. \quad (\text{B.22})$$

As $\Phi_i = 0$ for $i \notin \mathcal{A}$, $\left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right) = 1$ for $i \notin \mathcal{A}$, so in the product we can consider only the set of indices \mathcal{A} , giving us

$$\Rightarrow \prod_{i \in \mathcal{A}} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{w_i} = 1. \quad (\text{B.23})$$

We can now take the $\left(\sum_{j \in \mathcal{A}} w_j\right)^{\text{th}}$ root, so we obtain

$$\Rightarrow \prod_{i \in \mathcal{A}} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{\frac{w_i}{\sum_{j \in \mathcal{A}} w_j}} = 1 \quad (\text{B.24})$$

$$\Rightarrow \prod_{i \in \mathcal{A}} \left(1 + \frac{\gamma^{d_i} \Phi_i}{R_i}\right)^{\check{w}_i} = 1 \quad (\text{B.25})$$

as required.