A General Theory of Liquidity Provisioning for Prediction Markets

Adithya Bhaskara, Rafael Frongillo, and Maneesha Papireddygari

{adithya,raf,maneesha.papireddygari}@colorado.edu

July 29, 2025

Abstract

In Decentralized Finance (DeFi), automated market makers typically implement liquidity provisioning protocols. These protocols allow third-party liquidity providers (LPs) to provide assets to facilitate trade in exchange for fees. This paper introduces a general framework for liquidity provisioning for cost-function prediction markets with any finite set of securities. Our framework is based on the idea of running several market makers "in parallel"; we show formally that several notions of parallel market making are equivalent to ours. The most general protocol therefore allows LPs to submit an arbitrary cost function, which specifies their liquidity over the entire price space, and determines the deposit required. We justify the need for this flexibility by demonstrating the inherent high dimensionality of liquidity. We also give several restricted protocols which are more computationally feasible.

Furthermore, we show that our protocol recovers several existing DeFi protocols in the 2-asset case. Our work also contributes to the DeFi literature by giving a fully expressive protocol for any number of assets.

1 Introduction

The concept of an automated market maker was originally introduced by Hanson (2003) to solve thin market problems in combinatorial prediction markets. In contrast to order books and continuous double-auctions, where buyers are matched to sellers, automated market makers are central authorities willing to price any bundle of assets to buy or sell. More recently, automated market makers have gained in popularity in the context of decentralized finance (DeFi) as a low-gas way to implement a market on a blockchain (Base, 2025; Bartoletti et al., 2022; Xu et al., 2023; Mohan, 2022). Along with this trend came the innovation of decoupling the roles of the market mechanism, which facilitates trade, and liquidity providers, which take on risk to stabilize prices. With this decoupling, the DeFi market mechanism exposes another interface to potential liquidity providers (LPs). LPs, now distinct from the market maker, deposit assets in exchange for a cut of the fees charged on the trades using those assets.

Thus far, however, the design of liquidity provisioning interfaces has been somewhat ad-hoc and only focused on the case of two assets. In the Uniswap V2 interface, LPs must deposit funds proportional to the current reserves, a natural yet restrictive interface. Zetlin-Jones et al. (2024) show how these restrictions lead LPs to actively trade against the market—that is, themselves and other LPs—leading to potential inefficiencies. Uniswap V3 adds significant flexibility, but the interface is still somewhat cumbersome: LPs must contend with discrete "buckets" in the price space to allocate their funds. Throughout, the full design space of liquidity provisioning protocols has been far from clear.

This gap is especially large in the case of prediction markets, as they often exchange more than two securities. For example, an election market might offer a security for each potential candidate, or even a combinatorial market for the outcomes per state. Offering multiple independent markets for each pair of securities not only leads to information loss, it creates large arbitrage opportunities that increases the risk of providing liquidity (Dudik et al., 2012). Thus, effective liquidity provisioning protocols for more than two assets could have a significant impact on the performance and prevalence of prediction markets. Despite all these considerations, no liquidity provisioning protocol for prediction markets has been proposed.

We fill this gap by introducing a general framework for liquidity provisioning protocols for cost-function prediction markets trading any number of securities (§ 4). Our framework is designed on the idea of LPs running market makers "in parallel"; we show that several notions of parallel market making are equivalent, including scoring rule markets of Hanson (2003) (§ 5). Our protocol allows LPs to submit an arbitrary cost function, specifying essentially any liquidity allocation over the price space, and determines the deposit required to indeed provide that liquidity. One could view our protocol as theoretically formalizing Minswap (Nguyen, 2021), which is designed on Cardano to accomodate multiple LP pools. When working with 3 or more assets, we argue that the liquidity at a given price must be described by a full matrix, allowing one to assess the liquidity in each "direction", i.e., for each possible trade (§ 3). As a corollary, any fully general liquidity provisioning protocol must allow liquidity between all assets simultaneously, rather than only allowing liquidity in pairwise markets. We show how our protocol can recover existing DeFi protocols in the case of two assets, while also giving rise to expressive DeFi protocols with any number of assets that are more computationally feasible (§ 6). We conclude in § 7 with a discussion of fees in $n \geq 3$ securities and open directions.

1.1 Related work

We direct readers to Chen and Pennock (2007); Abernethy et al. (2013) for an overview of the literature on automated market makers for prediction markets, and to Angeris and Chitra (2020) for those in DeFi. Our work heavily relies on Frongillo et al. (2023), which establishes the equivalence of (non-parallelized) automated market makers as used for prediction markets to those used in DeFi. One can view our analysis of the trade split $\mathbf{r} = \sum_i \mathbf{r}^i$ as a special case of optimal routing problems stated in Angeris et al. (2022); Diamandis et al. (2023). We provide a closed form solution to this special case not yet seen in the literature. This work is also related to recent explorations of running parallel LMSRs in Dudík et al. (2021) and of geometric aspects of automated market makers (Angeris et al., 2023). In particular, the Minkowski sum operations in latter paper can be seen as implicitly computing an infimal convolution, which they also view as a combined market maker. However, their connections to implementing liquidity provisioning are not explored beyond a very restricted setting.

Perhaps closest to our general framework is Milionis et al. (2023), which asks LPs for their "demand curves" to be aggregated into a two-asset market maker. Our framework can be thought of as a similar, but more generalized, way of thinking about LPs running several markets in parallel. Their demand curves $h(p)$ (denoted $g(p)$ in their paper) are related to our market scoring rule with $h(p) = S(p, 1)$ and $-\int p dh(p) = S(p, 0)$. While demand curves are well rooted in micro-economic foundations (Milionis et al., 2024), we demonstrate in § 3 that cost functions (or some other suitable higher-dimensional notion of demand curves) are necessary to give a fully general liquidity provisioning protocol for more than two assets, a crucially important case for prediction markets as we describe above. They are also instrumental in helping us propose a closed-form construction of aggregate CFMMs and in helping us realize and prove different equivalent interpretations (Theorem 1).

2 Background

2.1 Notation and convex analysis

Vectors are denoted in bold, e.g. $\mathbf{q} \in \mathbb{R}^n$, and $q_i \in \mathbb{R}$ denotes the ith coordinate of $\mathbf{q}$. The all-zeros vector is $\mathbf{0} = (0, \ldots, 0)$ and the all-ones vector is $\mathbf{1} = (1, \ldots, 1)$. We define the indicator vector $\boldsymbol{\delta}^i$ by $\delta^i_i = 1$ and $\delta^i_j = 0$ for $j \neq i$. Comparison between two vectors is pointwise, e.g. $\mathbf{q} \succ \mathbf{q}'$ if $q_i > q'_i$ for all $i = 1, \ldots, n$, and similarly for $\succeq$. We say $\mathbf{q} \gneqq \mathbf{q}'$ when $q_i \geq q'_i$ for all i and $\mathbf{q} \neq \mathbf{q}'$. Define $\mathbb{R}^n_{\geq 0} = \{\mathbf{q} \in \mathbb{R}^n \mid \mathbf{q} \succeq \mathbf{0}\}$, $\mathbb{R}^n_{>0} = \{\mathbf{q} \in \mathbb{R}^n \mid \mathbf{q} \succ \mathbf{0}\}$, et cetera. Finally, we denote the probability simplex by $\Delta_n = \{\mathbf{p} \in \mathbb{R}^n_{\geq 0} \mid \langle \mathbf{p}, \mathbf{1} \rangle = 1\}$.

Let $f : \mathbb{R}^n \to \mathbb{R}$. We use the following conditions.

- *convex*: $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n, \lambda \in [0, 1], f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y})$.
- $\mathbf{1}$*-invariant*: $f(\mathbf{q} + \alpha \mathbf{1}) = f(\mathbf{q}) + \alpha$ for all $\mathbf{q} \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$.
- *1-homogeneous (on* $\mathbb{R}^n_{\geq 0}$*)*: $f(\alpha \mathbf{q}) = \alpha f(\mathbf{q})$ for all $\mathbf{q} \succeq \mathbf{0}$ and $\alpha > 0$.

We use f' to indicate differentiation when f is 1-dimensional.

Definition 1 (Convex conjugate)**.** *For a function* $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *we define its* convex conjugate $f^* : \mathbb{R}^n \to \mathbb{R} \cup \{\infty, -\infty\}$ *by* $f^*(\mathbf{x}^*) = \sup_{\mathbf{x} \in \mathbb{R}^n} \langle \mathbf{x}^*, \mathbf{x} \rangle - f(\mathbf{x})$.

Definition 2 (Subgradient)**.** *For a function* $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *and* $\mathbf{x} \in \mathbb{R}^n$ *we define the set of* subgradients *of* f *at* $\mathbf{x}$ *by* $\partial f(\mathbf{x}) = \{\mathbf{x}^* \in \mathbb{R}^n \mid \forall \mathbf{x}' \in \mathbb{R}^n\, f(\mathbf{x}') - f(\mathbf{x}) \geq \langle \mathbf{x}^*, \mathbf{x}' - \mathbf{x} \rangle\}$.

Definition 3 (Infimal convolution)**.** *For functions* $f_i : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ *we define their* infimal convolution $f = \bigwedge_i f_i$ *by* $f(\mathbf{x}) = \inf\left\{\sum_i f_i(\mathbf{x}^i) \mid \sum_i \mathbf{x}^i = \mathbf{x}\right\}$*, where the* $\mathbf{x}^i$ *range over* $\mathbb{R}^n$.

Definition 4 (1-homogeneous extension $\overline{f}$)**.** *Given* $f : \Delta_n \to \mathbb{R}$*, we define its* 1-homogeneous extension $\overline{f} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}$ *by* $\overline{f}(\mathbf{x}) := \|\mathbf{x}\|_1 f(\mathbf{x}/\|\mathbf{x}\|_1)$ *for* $\mathbf{x} \neq \mathbf{0}$ *and* $\overline{f}(\mathbf{0}) = 0$.

2.2 Cost functions and prediction markets

Automated market makers (AMMs) are mechanisms that are always willing to trade a bundle of n securities for some price. In contrast to traditional order book settings, where traders are waiting to be matched with sellers, traders can trade with AMMs directly. Prediction markets are AMM mechanisms that seek to elicit probability distributions over future events by allowing traders to buy and sell securities. Some common instantiations of prediction markets can be seen in horse betting, Iowa electronic markets, and more recently, Manifold markets (Manifold, 2022).

Suppose a random variable Y about a future event takes values from set $\mathcal{Y}$, which contains n mutually exclusive and exhaustive outcomes. A trader holding an Arrow-Debreu (AD) security associated with $y \in \mathcal{Y}$ gets paid \$1 when outcome y happens and \$0 otherwise. A security market is *complete* if it trades n independent AD securities, one for each outcome. While our Protocol 1 works for incomplete markets, our proofs in § 5.3 rely heavily on the market being complete. For this reason and for ease of exposition, we restrict our attention in this paper to complete markets.

Chen and Pennock (2007); Chen et al. (2013); Abernethy et al. (2013) characterize prediction markets, and Abernethy et al. (2013) show that they should be implemented by a cost function-based market maker satisfying certain conditions in order to satisfy certain information elicitation axioms. We define these below.

Definition 5 (Cost function-based market maker)**.** *A cost function-based market maker with* n *securities is one that prices each security via a differentiable potential function* $C : \mathbb{R}^n \to \mathbb{R}$*. Suppose a trader wants to purchase a bundle of securities* $\mathbf{r}' \in \mathbb{R}^n_{\geq 0}$*; that is,* r_i *shares of security* i*. Then, the trader must pay* $C(\mathbf{q} + \mathbf{r}') - C(\mathbf{q})$ *in cash to the market maker.*

As shown by Abernethy et al. (2013), prediction markets that elicit information well are precisely cost function-based prediction markets with C convex and $\mathbf{1}$-invariant.

Remark (From Frongillo et al. (2023))**.** *For a net trade*[1] $\mathbf{r} \in \mathbb{R}^n$*, the cost function satisfies* $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$.

To see why this is true, we use the observation that in a complete AD securities market, holding one of each security i.e., $\mathbf{1}$ is equivalent to holding \$1 cash. If the trader requests the market maker a trade of $\mathbf{r}' \in \mathbb{R}^n_{\geq 0}$, they have to pay $C(\mathbf{q} + \mathbf{r}') - C(\mathbf{q})$ cash. The net trade is hence $\mathbf{r} = \mathbf{r}' - (C(\mathbf{q}+\mathbf{r}') - C(\mathbf{q}))\mathbf{1}$. Using the $\mathbf{1}$-invariant property of C, we can see that $C(\mathbf{q}+\mathbf{r}) = C(\mathbf{q})$.

Cost function-based markets always maintain a *liability* $\mathbf{q} \in \mathbb{R}^n$ of securities, q_i of security i that the market has sold so far. The term liability comes from the notion that $C(\mathbf{q})$ is the amount of cash wagered in the market, that the market maker is liable for.

[1] Note that while it is not customary for predictions to deal with the net trade, we find it easier to, especially as the comparisons to the DeFi literature are easier to make, as we see later.

2.3 Scoring rules

Scoring rules were introduced by Brier (1950) to score forecasts of a random variable such as Y above. In this setting, we seek to design a score $S(\mathbf{p}, y)$ that determines the quality of prediction $\mathbf{p} \in \Delta_{\mathcal{Y}}$ upon observing the outcome $y \in \mathcal{Y}$, with the property that $\mathbb{E}_{\mathbf{p}} S(\mathbf{p}', Y)$ is maximized at $\mathbf{p}' = \mathbf{p}$. The full characterization of such "proper" scoring rules take the form

$$S_G(\mathbf{p}, y) = G(\mathbf{p}) + \langle \mathbf{d}G_{\mathbf{p}}, \delta_y - \mathbf{p} \rangle$$

where $G : \Delta_{\mathcal{Y}} \to \mathbb{R}$ is a convex function (Gneiting and Raftery, 2007). When $\mathcal{Y} = \{0, 1\}$, we can write $p \in [0, 1]$ to be the predicted probability that $Y = 1$, and write $S_g(p, y) = g(p) + g'(p)(y - p)$ for $g : [0, 1] \to \mathbb{R}$ convex.

Hanson (2003) showed that scoring rules could be used to design AMMs in a form we call a *scoring rule market*; see Protocol 3. The basic idea is to pay traders according to a difference of scoring rules, with the latest trader's prediction acting as the current market price. It was later shown that this formulation is equivalent in a strong sense to the cost-function market makers described above (Abernethy et al., 2013). Specifically, the scoring rule market for S_G is equivalent to the cost-function market maker with cost function $C = G^*$, the convex conjugate of G.

A corollary of these connections, leveraged in Frongillo et al. (2023), is that one can use scoring rules as vectors to convert between the market price vector and the current liability/reserve vector. Specifically, let $S_G(\mathbf{p}, \cdot) = (S_G(\mathbf{p}, y))_{y \in \mathcal{Y}} \in (\mathbb{R} \cup \{\infty\})^n$ be the scoring rule vector for price $\mathbf{p}$. Then, up to a uniform shift, the liability vector of a cost-function market maker with cost function $C = G^*$ at price $\mathbf{p}$ is simply $S_G(\mathbf{p}, \cdot)$. We will use this correspondence throughout the paper, as well as the 2-outcome version $S_g(p, \cdot) = (g'(p), 0) + (g(p) - p \cdot g'(p))\mathbf{1} \in (\mathbb{R} \cup \{\infty\})^2$.

2.4 Automated market makers in decentralized finance

Recently, AMMs have been implemented in a widespread manner for decentralized finance (DeFi). There, the goal is not information elicitation, as in prediction markets, but rather the exchange of assets–namely cryptocurrencies. Analogous to how cost function markets maintain a liability vector $\mathbf{q}$, AMMs maintain a reserve vector $\mathbf{x} = -\mathbf{q}$ of assets.

The framework of constant function market makers (CFMMs) is prominent in the DeFi literature. Frongillo et al. (2023); Angeris and Chitra (2020); Schlegel et al. (2022) argue that for various restrictions on their design, CFMMs satisfy desirable market making axioms. We define these market makers below.

Definition 6 (Constant function market maker, CFMM)**.** *A constant function market maker (CFMM) is a market maker based on a potential function $\varphi : \mathbb{R}^n \to \mathbb{R}$ that maintains a liability $\mathbf{q} \in \mathbb{R}^n$. At the current liability, the set of trades $\mathbf{r}$ available are those that satisfy $\varphi(\mathbf{q} + \mathbf{r}) = \varphi(\mathbf{q})$. After a trade, the liability vector updates to $\mathbf{q} \leftarrow \mathbf{q} + \mathbf{r}$.*

Consistent with this definition, as we soon again discuss, in this paper, we adopt the sign convention that trades are always oriented toward the trader. For example, a trade $\mathbf{r} = (1, -3)$ corresponds to giving the trader 1 unit of asset 1 in exchange for 3 units of asset 2. As explored in Frongillo et al. (2023) and other works, the classic cost function market makers commonly employed in prediction markets are special cases of CFMMs that retain the full flexibility of general potential functions φ.

The relationship between cost function prediction market makers and CFMMs is thoroughly explored in Frongillo et al. (2023). The two objects are different, but give rise to equivalent characterizations of markets. We use cost functions throughout, even when discussing CFMMs, as they have more mathematical structure without loss of expressiveness. For example, while for any potential φ, one can take ratios of partial derivatives to compute relative prices, this task is even easier for cost functions, as prices are normalized. In the context of prediction markets, normalization means that prices $\mathbf{p} \in \partial C(\mathbf{q})$ can be thought of as a probability distribution over outcomes. With or without this interpretation, we frequently use the fact that $\partial C(\mathbf{q}) \subseteq \Delta_n$ for all $\mathbf{q} \in \mathbb{R}^n$.

Some examples of CFMMs include the logarithmic market scoring rule (LMSR) with potential function $\varphi(\mathbf{q}) = b \log\left(\sum_{i=1}^n e^{q_i/b}\right)$ for $b > 0$. Another example is the constant product market maker which has a potential that is not a cost function: $\varphi(\mathbf{q}) = q_1 \cdot q_2 \cdots q_n$. The cost function equivalent to the constant product market maker in two assets is given by

$$C(\mathbf{q}) = \frac{1}{2}\left(q_1 + q_2 + \sqrt{4\alpha^2 + (q_1 - q_2)^2}\right)$$

as found in Chen and Pennock (2007); Frongillo et al. (2023). We also have $G(\mathbf{p}) = C^*(\mathbf{p}) = -2\sqrt{\alpha p_1 p_2}$ and more generally $G(\mathbf{p}) = -n(\alpha p_1 \cdot p_2 \cdots p_n)^{1/n}$.

A major AMM innovation in DeFi is the introduction of *liquidity provisioning*. In traditional AMMs, the market maker took on an additional risk of price fluctuations of the reserves for the ability to run a market always willing to price a bundle of assets. The DeFi implementations of AMMs, though, have outsourced provisioning these reserves, and hence liquidity, to external parties called liquidity providers (LPs). The AMMs typically define trade dynamics when liquidity is fixed. In this setting, traders can exchange assets with the market maker in a way that keeps the reserves/liability on the same invariant curve of φ or C. Decentralized finance protocols like Uniswap V2 and Uniswap V3 also allow liquidity providers (LPs) to change the market's liquidity while keeping the price $\mathbf{p}$ invariant (Adams et al., 2020, 2021). LPs may either add, or *mint*, liquidity to the market or remove, or *burn*, liquidity from the market. LPs make it easier for the AMM to conduct trades by absolving the market maker of the risk of providing liquidity. As compensation for taking on the risk, LPs are rewarded using trading fees, which are skimmed off along with the trade requested. This provides a pool to be distributed proportionally to LPs as the liquidity they allocated is used.

3 Defining liquidity in automated market makers

Informally, liquidity is the extent to which assets/securities can be exchanged. Below we quantify this notion, locally around a given price, in the context of automated market makers.

3.1 Liquidity as price insensitivity

One way to capture liquidity, locally, is to quantify the extent to which the price stays stable during a transaction. In other words, the lower the rate of change of the price, the higher the liquidity.

Thus far in the prediction market literature, even for large numbers of securities, liquidity is often captured by a single parameter. The most popular approach is to consider some base cost function C and define C_η via the *perspective transform* $C_\eta(\mathbf{q}) = \eta C(\mathbf{q}/\eta)$, where η corresponds to the liquidity of the market (Othman et al., 2013; Li and Vaughan, 2013; Abernethy et al., 2014;

Othman and Sandholm, 2011). Another approach is to define the liquidity of C to be some function of its Hessian $\nabla^2 C$ such as the inverse of its norm (Abernethy et al., 2013). A noted exception is Dudík et al. (2014), where liquidity is acknowledged to depend on which specific subset of securities are under consideration. Our approach is closest to the latter: as we argue soon in § 3.2, liquidity is indeed inherently multidimensional. Any subspace of securities could have any degree of liquidity. It is true that for very restricted protocols such as Uniswap V2, which indeed corresponds to the perspective transform, liquidity has a fixed shape that can be scaled by a single real parameter. But, in general, liquidity must be captured by a higher-dimensional object.

To make the above discussion concrete, let us begin with the case of 2 securities. Here we take advantage of the fact that by $\mathbf{1}$-invariance, a 2-dimensional cost function is really specified by only a 1-dimensional function. More formally, given any $\mathbf{1}$-invariant cost function $C : \mathbb{R}^n \to \mathbb{R}$, we may write $C(\mathbf{q}) = c(q_1 - q_2) + q_2$ for some convex $c : \mathbb{R} \to \mathbb{R}$ given by $c(q) = C((q, 0))$. Letting $\mathbf{q} = (q, 0)$, then, the price of security 1 is $\nabla C(\mathbf{q})_1 = c'(q)$.

Appealing to the above local notion of liquidity, let us define the liquidity at price p to be the reciprocal of the rate of change of the price when the price is p. While not required in general, for the purposes of this intuitive derivation, suppose that $c'' > 0$ everywhere. Then formally, we can define the liquidity at price $p \in [0, 1]$ as $\ell(p) = 1/c''(q) > 0$, where $p = c'(q)$. Since ℓ is strictly positive, we find that $\ell(p) = g''(p)$ for some convex function $g : [0, 1] \to \mathbb{R}$. This relationship is in essence a special case of convex conjugate duality: we may simply take $g = c^*$. From this duality, we have $g' = (c')^{-1}$, which is well-defined as c' is strictly monotone; by the inverse function theorem, we could equivalently derive ℓ as $\ell(p) = ((c')^{-1})'(p) = g''(p)$.

In higher dimensions, we can analogously define the liquidity at price $\mathbf{p}$ to be $\ell(\mathbf{p}) = (\nabla^2 C(\mathbf{q}))^+$ at any vector $\mathbf{q}$ with price $\nabla C(\mathbf{q}) = \mathbf{p}$, where A^+ is the pseudoinverse of A.[2] Here, liquidity is a matrix, which specifies the (inverse) rate of change of the price in any direction (or more generally, subspace) of interest. Again appealing to convex duality, we can write $\mathbf{q} = \nabla \overline{G}(\mathbf{p})$, where $\overline{G}$ is the 1-homogenous extension of the dual function $G = C^*$. Thus, we have $\ell(\mathbf{p}) = \left(\nabla^2 C(\nabla \overline{G}(\mathbf{p}))\right)^+$, or equivalently, $\ell(\mathbf{p}) = \nabla^2 \overline{G}(\mathbf{p})$.

In some cases it may be more natural to *start* with a liquidity function and arrive at a cost function. In examples, we often work with 2 assets, where one can start with some $\ell : [0, 1] \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ and arrive at the 1-dimensional cost $c = (\iint \ell)^*$. [3] As these integrals are definite, $g = \iint \ell$ is only determined by ℓ up to an affine function, and similarly c up to a shift and translation. This flexibility can be utilized to determine the "most efficient" cost function dual. This can be achieved by only allowing for the minimum deposit that ensures no-liability condition (discussed in §4.4), which imposes $g(0) = g(1) = 0$. Hence the "optimal" g is given by

$$g(p) = \int_0^p \int_0^t \ell(s)\, ds\, dt - p \int_0^1 \int_0^t \ell(s)\, ds\, dt\ . \tag{1}$$

We use g and ℓ heavily in examples, especially in § 6.

[2] The pseudoinverse is needed as the Hessian $\nabla^2 C$ is rank-deficient since C is always flat in the $\mathbf{1}$ direction. This observation also explains why we can express liquidity between 2 securities in one real number, since there is only 1 free parameter in $\nabla^2 C$ in that case.

[3] We include ∞ as a possible liquidity level, as often we will want $\ell(0) = \ell(1) = \infty$, as in the LMSR and constant product market makers. The latter is used significantly in decentralized finance.

3.2 Liquidity as a Hessian matrix

Above we argued that liquidity is captured by an entire matrix, the inverse Hessian of the cost function C. In the 2-asset case, this matrix is rank 1 and only has one free parameter, aligning with our intuition that liquidity is 1-dimensional in that case. In higher dimensions, however, one may wonder if the extra expressivity of a matrix is needed. Let us see why it is with an example.

Consider two cost function duals on 3 assets, both based on the constant product market maker. The first is the dual of the usual constant product market maker, and the second the sum of pairwise constant products for all 3 pairs of assets:

$$G^{(1)}(\mathbf{p}) = -3(p_1p_2p_3)^{1/3}\ , \quad G^{(2)}(\mathbf{p}) = -2\sqrt{p_1p_2} - 2\sqrt{p_2p_3} - 2\sqrt{p_1p_3}\ .$$

We will examine their liquidity as a function of $\mathbf{p}$. Both functions are written in a 1-homogenous form already, so we may simply compute their Hessians to obtain

$$\nabla^2\overline{G}^{(1)} = \frac{1}{3(p_1p_2p_3)^{2/3}}\begin{bmatrix} \frac{2p_2p_3}{p_1} & -p_3 & -p_2 \\ -p_3 & \frac{2p_1p_3}{p_2} & -p_1 \\ -p_2 & -p_1 & \frac{2p_1p_2}{p_3} \end{bmatrix}\ ,$$

$$\nabla^2\overline{G}^{(2)} = \frac{1}{2(p_1p_2p_3)^{1/2}}\begin{bmatrix} \frac{p_2\sqrt{p_3}+\sqrt{p_2}p_3}{p_1} & -\sqrt{p_3} & -\sqrt{p_2} \\ -\sqrt{p_3} & \frac{p_1\sqrt{p_3}+\sqrt{p_1}p_3}{p_2} & -\sqrt{p_1} \\ -\sqrt{p_2} & -\sqrt{p_1} & \frac{p_1\sqrt{p_2}+\sqrt{p_1}p_2}{p_3} \end{bmatrix}\ .$$

These matrices are rank 2, with a 0 eigenvalue in direction $\mathbf{p}$.

Now let us imagine a trader wishing to purchase asset 1 in exchange for asset 3. Letting $\mathbf{v} = (1, 0, -1)^\top$, the liquidity in these two markets in this direction, meaning the price insensitivity as a trader purchases asset 1 for 3, can be calculated simply as

$$\mathbf{v}^\top\nabla^2\overline{G}^{(1)}\mathbf{v} = \frac{2p_2\left(\frac{p_1}{p_3}+\frac{p_3}{p_1}+1\right)}{3(p_1p_2p_3)^{2/3}} = p_2^{1/3}\frac{2\left(\frac{p_1}{p_3}+\frac{p_3}{p_1}+1\right)}{3(p_1p_3)^{2/3}}\ ,$$

$$\mathbf{v}^\top\nabla^2\overline{G}^{(2)}\mathbf{v} = \frac{1}{\sqrt{p_1p_3}} + \frac{\sqrt{p_1p_2}+\sqrt{p_1p_3}}{2p_1^2} + \frac{\sqrt{p_1p_3}+\sqrt{p_2p_3}}{2p_3^2}\ .$$

We can now see that as $p_2 \to 0$, the price of asset 2 approaches 0, the liquidity between the other two assets is also driven to 0 for $G^{(1)}$. As first observed by Grugett (2023), this behavior can be undesirable, and is alleviated by considering $G^{(2)}$ instead, since there the liquidity is lower bounded by $1/\sqrt{p_1p_3}$ regardless of the price of asset 2. Simply summarizing the liquidity or "depth" of these markets by a single real value could miss this nuance.

4 Liquidity provisioning protocol for prediction markets

In this section, we first give intuition for why LPs can be thought of as running "parallel" markets. We then provide the general liquidity provisioning protocol that can be implemented in prediction markets. Next, discuss some key technical aspects of the protocol, which are essential for gaining a deeper understanding of the design space. In Section 5.3 we show how this protocol leads to several equivalent ways of thinking about LPs running markets in parallel. All these intuitive interpretations not only enhance our understanding of how liquidity provisioning can be implemented in prediction markets, but they also justify the framework we propose.

4.1 Liquidity provisioning as competing market makers

In traditional financial markets, such as continuous double-auctions, a market maker is an entity that offers both to buy and sell an asset. Typically the buy price is lower than the sell price; the difference comprises the *bid-ask spread*. Market makers earn a profit equal to the bid-ask spread whenever both buy and sell orders are executed, while remaining even with respect to the asset. In essence, market making is all about providing liquidity for a small premium, or "fee", as given by the spread. In these traditional markets, liquidity provisioning happens naturally, as often multiple market makers coexist. Rational traders will only buy or sell from the most favorable price offered, switching at will between different market makers.

The key idea behind our protocol therefore is to implement liquidity provisioning in the same manner, with multiple coexisting automated market makers. That is, we seek a protocol which implements an LP as simply another "competing" market maker, and we let traders interact with them all at once, i.e., in parallel. How could one implement such a protocol?

There are at least two natural ways to imagine this parallel transaction proceeding (See § 5 for more). First, a trader could select a valid trade $\mathbf{r}^i$ for the automated market maker of LP i, resulting in a net trade $\mathbf{r} = \sum_i \mathbf{r}^i$. As detailed in § 2.4, these valid trades can be expressed as those satisfying $C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)$ given a convex cost function C_i and current liability vector $\mathbf{q}^i$ for LP i. Second, a trader could start trading in infinitesimal amounts with each LP, each time taking the most favorable price, and stopping at some point, yielding a net trade $\mathbf{r}$. Perhaps surprisingly, by fundamental results in convex analysis, these two approaches are identical. Taken together, we can see that any Pareto-optimal trade leaves the combined market in a coherent state, with the price of each LP matching the global market price.

At first glance, it might appear that a major downside of our approach is the need for traders to interact directly with each LP, increasing the complexity of interaction required. Fortunately, one can simplify the interface: there always exists a single aggregate cost function C that captures the available net trades. Specifically, given the cost functions C_i defining each market maker, the valid trades in the combined market are exactly those of their *infimal convolution* $C = \bigwedge_i C_i$. Thus, the trader can simply choose any trade satisfying $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$, and behind the scenes, the split $\mathbf{r} = \sum_i \mathbf{r}^i$ can be computed along with the corresponding fees.

To check that the infimal convolution yields a sensible protocol, let us ask how the total liquidity in the market relates to that of the individual LPs. As justified above, we can quantify this liquidity in the matrix $(\nabla^2 C)^+$, or dually, $\nabla^2 \overline{G}$ where $G = C^*$. By results in convex analysis, the dual of an infimal convolution is the sum of the duals, giving $G = C^* = (\bigwedge_i C_i)^* = \sum_i C_i^* = \sum_i G_i$. As a consequence, the total liquidity of the combined market is $\nabla^2 \overline{G} = \sum_i \nabla^2 \overline{G}_i$. In other words, just as one would hope, adding another parallel market maker C_i literally adds the corresponding liquidity $\nabla^2 \overline{G}_i$ to the pool.

Another way to see this operation of adding liquidity is by considering the equivalent market scoring rule (MSR) formulation of cost functions. Here running MSRs "in parallel" is the same as adding their scoring rules S_{G_i} together, which is equivalent to using a single scoring rule S_G generated by the sum of the generating functions $G = \sum_i G_i$. See § 5.

For example, in the two-security case, suppose two LPs want to come together and provide liquidity using the LMSR cost functions with liquidity parameters b_1 and b_2. This gives a aggregate cost function of $C(\mathbf{q}) = (C_1 \wedge C_2)(\mathbf{q}) = (b_1 + b_2) \log \left(\exp\left(\frac{q_1}{(b_1+b_2)}\right) + \exp\left(\frac{q_2}{(b_1+b_2)}\right) \right)$. The aggregate cost function is indeed equivalent to the convex conjugate of the sum of $G_1(\mathbf{p}) = -b_1 H(\mathbf{p}) = C_1^*(\mathbf{p})$

Protocol 1 General protocol as parallel market makers

global constant $\mathcal{C}_{\text{init}} \subseteq \mathcal{C}_n^*$, $\mathcal{C}_{\text{LP}} \subseteq \mathcal{C}_n$, $\mathsf{fee}(), \mathsf{fee}_i() \in \mathbb{R}_{\geq 0}$.
global variables $k \in \mathbb{N}$, $\{\mathbf{q}^i \in \mathbb{R}^n\}_{i=0}^k$, $\{C_i \in \mathcal{C}_{\text{LP}}\}_{i=0}^k$
$\mathsf{liability}(C) := \mathbf{q} \in \mathbb{R}^n$ s.t. $\nabla C_0(\mathbf{q}^0) \in \partial C(\mathbf{q})$ and $C(\mathbf{q}) = 0$ ▷ Price matching, no-liability

function $\mathsf{Initialize}(\mathbf{q} \in \mathbb{R}^n, C \in \mathcal{C}_{\text{init}})$
    $(k, \mathbf{q}^0, C_0) \leftarrow (0, \mathbf{q}, C)$
    check $\mathbf{q}^0 = \mathsf{liability}(C_0)$

function $\mathsf{RegisterLP}(i = k + 1)$
    $(k, \mathbf{q}^i, C_i) \leftarrow (k + 1, 0, \max)$

function $\mathsf{ModifyLiquidity}(i \in \mathbb{N}, C \in \mathcal{C}_{\text{LP}})$
    request $\mathbf{r}^i = \mathbf{q}^i - \mathsf{liability}(C)$ from LP i
    $(\mathbf{q}^i, C_i) \leftarrow (\mathbf{q}^i - \mathbf{r}^i, C)$

function $\mathsf{ExecuteTrade}(\mathbf{r} \in \mathbb{R}^n)$
    $\mathbf{q} \leftarrow \sum_{i=0}^k \mathbf{q}^i$
    check $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$ where $C = \wedge_{i=0}^k C_i$
    trader pays $\mathsf{fee}(\mathbf{r}, \mathbf{q}, C)$ cash in fees
    give $\mathbf{r}$ to trader
    write $\mathbf{r} = \sum_{i=0}^k \mathbf{r}^i$ s.t. $\forall i$, $C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)$
    for each LP i **do**
        LP i gets $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, \{C_i\}_{i=1}^k)$ fees
        $\mathbf{q}^i \leftarrow \mathbf{q}^i + \mathbf{r}^i$

and $G_2(\mathbf{p}) = -b_2 H(\mathbf{p}) = C_2^*(\mathbf{p})$ where $H(\mathbf{p}) = p_1 \log p_1 + p_2 \log p_2$. Alternately, the corresponding MSRs for the log scores are $b_1 S$, $b_2 S$ where $S = (\log p_1, \log p_2)$. The net MSR of the combined market is hence $(b_1 + b_2)S$. This special case of cost functions in the same perspective transform family is the analogy to Uniswap V2 in DeFi; see § 6.2. We give a more involved example in § 4.3.

4.2 General protocol

Our proposed protocol for liquidity provisioning in prediction markets is described in Protocol 1. At a high level, the protocol works as follows. Let n be the number of securities. The market creator acts as the initial LP, giving reserves $\mathbf{q}^0$ to the liquidity pool and specifying the initial cost function C_0. When an additional LP i enters, their liability vector and cost function are initialized to the trivial values $\mathbf{q}^i = \mathbf{0}$ and $C_i(\mathbf{q}) = \max(\mathbf{q}) := \max_j q_j$, so that they initially provide no liquidity.[4] The ModifyLiquidity function handles an LP adding, removing, or otherwise altering their deposited liquidity: they simply replace their cost function with a different one, and are charged up-front the minimal deposit to ensure *no-liability*, i.e., that they will never owe the market maker in any future state. We provide more discussion on the no-liability condition in the next section (§4.4). When removing all liquidity, the LP simply sets $C_i = \max$ once again, and is

[4] To see why this choice is correct, note that the "bid-ask spread" of C_i at $\mathbf{q}^i = 0$ is maximal; every price vector in Δ_n is consistent, and any trade occurs at the worst feasible price. More technically, adding the max function to the infimal convolution $C = \wedge_{i=0}^k C_i$ does not change the result. Dually, the conjugate of max is the convex indicator of Δ_n, so this choice adds liquidity $G_i = 0$; see § 4.

given back their entire deposit. ExecuteTrade checks if a trade $\mathbf{r}$ is an allowed trade with the overall cost function $C = \wedge_{i=0}^{k} C_i$, and if so, requests an additional fee of $\mathsf{fee}(\mathbf{r}, \mathbf{q}, C)$ cash from the trader. Under the hood, it then finds the optimal split $\mathbf{r} = \sum_i \mathbf{r}^i$ into smaller trades, executing each with the corresponding LP and doling out $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, \{C_i\}_{i=1}^k)$ in fees.

We leave axioms for the design of the fee functions for future work; see Appendix § D. One reasonable suggestion would be to take $\mathsf{fee}(\mathbf{r}, \mathbf{q}, C) = \beta\|\mathbf{r}\|$ and $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, \{C_i\}_{i=1}^k) = \beta\|\mathbf{r}\| \frac{\|\mathbf{r}_i\|}{\sum_j \|\mathbf{r}\|_j}$ for some norm $\|\cdot\|$ and $\beta > 0$. The form of fee_i here is to ensure budget balance of the fees, so that the market maker does not owe LPs more than the trader pays.

Several questions arise from this protocol: A key step in this protocol is the computation of liabilities. In particular, if an LP wishes to provide liquidity using C, and the current price is $\mathbf{p}$, what do they need to deposit? Does the required split $\mathbf{r} = \sum_i \mathbf{r}^i$ always exist? Can the various quantities be computed efficiently? Is there a sensible $\mathsf{fee}()$ function? It turns out that we have an affirmative answer to all of these questions, at least in the case $n = 2$, i.e., with only two outcomes, which we discuss over the upcoming sections [5].

4.3 Example run of the protocol

In Figure 1, we show visually how we can determine $g_i = \iint \ell_i$ in the case of $n = 2$ outcomes and 2 LPs. LP 1, in red, provides constant liquidity on $[0, 0.6]$, while LP 2, in blue, provides constant liquidity on $[0.4, 1]$. We may integrate twice (see eq 1) to find the optimal g_i to be

$$g_1(p) = \begin{cases} \frac{5}{2}p^2 - 2.1p & p \in [0, 0.6] \\ \frac{9}{10}(p-1) & p \in [0.6, 1] \end{cases}, \quad g_2(p) = \begin{cases} -1.8p & p \in [0, 0.4] \\ 5\left(p - \frac{2}{5}\right)^2 - 1.8p & p \in [0.4, 1] \end{cases}.$$

Taking the respective convex conjugates, we have

$$c_1(q) = \begin{cases} 0 & q \in (-\infty, -2.1] \\ \frac{(q+2.1)^2}{10} & q \in [-2.1, 0.9] \\ q & q \in [0.9, \infty) \end{cases}, \quad c_2(q) = \begin{cases} 0 & \text{if } q \in (-\infty, -1.8] \\ \frac{5q^2+58q+88.2}{100} & q \in [-1.8, 4.2] \\ q & q \in [4.2, \infty) \end{cases}.$$

Now, we step through Protocol 1 at a high-level with the fee scheme $\mathsf{fee}(\mathbf{r}, \mathbf{q}, C) = \beta||\mathbf{r}||_1$ for $\beta = 0.1$. We round decimal values to three places.

Let the market be initialized with price $p = 0.2$ and LP 1 has cost function $C_1(\mathbf{q}) = c_1(q_1 - q_2) + q_2$. So, at first, the market holds initial liability $\mathbf{q}^0 = \mathbf{q}^1 = (-1.2, -0.1)$. Say a trader would like to make a trade $\mathbf{r} = \mathbf{r}^1 = (-0.975, 0.525)$. The price becomes $p = 0.5$ after the trade and since there is only one LP active, LP 1, they get fees \$0.15. Then, $\mathbf{q}^1 \leftarrow (-0.225, -0.625)$.

Now, suppose LP 2 wants to provide liquidity, by initializing ModifyLiquidity with $C_2(\mathbf{q}) = c_2(q_1 - q_2) + q_2$. LP 2 is required to deposit $(1.25, 0.45)$ into the market maker to ensure liability $\mathbf{q}^2 \leftarrow (-1.25, -0.45)$, and the aggregate market has $\mathbf{q} = \mathbf{q}^1 + \mathbf{q}^2 = (-0.225, -0.625) + (-1.25, -0.45) = (-1.475, -1.075)$. At this point, the market has computed $C(\mathbf{q}) = (C_1 \wedge C_2)(\mathbf{q})$ under the hood. Now, whenever traders wish to exchange securities, they will trade with the aggregate market maker C; the trade split between each LP is calculated automatically. Say a trader wants to make the trade $\mathbf{r} = (-1.025, 1.475)$. The protocol computes the new price as $p = 0.7$. A part

[5] The computational efficiency will depend on the computational properties of the constituent cost functions, but in § 6 we will see how to implement the protocol efficiently in practice for several examples.

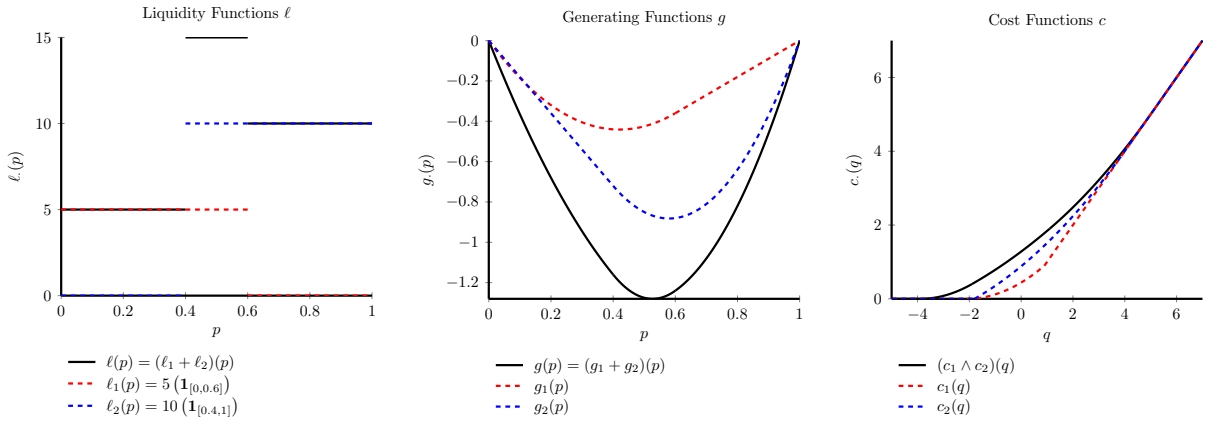

Figure 1: Liquidity functions and their associated generating functions.

of the trade $\mathbf{r}^1 = (-0.225, 0.275)$ is carried out by LP 1 for which it gets a \$0.05 fee, and LP 2 carries out the remainder, i.e. $\mathbf{r}^2 = (-0.8, 1.2)$ for a \$0.20 fee. We have the new liabilities as $\mathbf{q}^1 \leftarrow (-0.225, -0.625) - (-0.225, 0.275) = (0, -0.9)$, and $\mathbf{q}^2 \leftarrow (-1.25, -0.45) - (-0.8, 1.2) = (-0.45, -1.65)$. The aggregate market has $\mathbf{q} \leftarrow \mathbf{q}^1 + \mathbf{q}^2 = (-0.45, -2.55)$. What this means is that if outcome 2 realizes, LP 1 is paid out \$0.9 for its initial deposit of $(1.2, 0.1)$. Note that the LP deposits in this protocol are themselves outcome-contingent, but they change as trades occur. Thus, the LP deposits do not express a belief on the outcome, but rather beliefs about the volume of trade near the prices where liquidity was allocated.

4.4 Ensuring no liability

To capture no-liability, we require that the amount LP i owes the market maker of each security should be nonpositive. Given the cost function C for LP i, we must ask i to deposit some assets $\mathbf{x} \in \mathbb{R}^n_{\geq 0}$, therefore setting its liability to $\mathbf{q}^i = -\mathbf{x}$, so that $C(\mathbf{q}) = C(\mathbf{q}^i)$ implies $\mathbf{q} \preceq 0$ for all possible future states $\mathbf{q}$.

In principle, if an LP provides C where some level set satisfies no-liability, the protocol could compute it and request a corresponding deposit to cover the liability. More formally, if $C^{-1}(\alpha) := \{\mathbf{q} \in \mathbb{R}^n \mid C(\mathbf{q}) = \alpha\} \subseteq \mathbb{R}^n_{\leq 0}$, i.e. every liability vector in the α-level set $C^{-1}(\alpha)$ is nonpositive, the protocol could compute this α and request (minus) the element $\mathbf{q} \in C^{-1}(\alpha)$ such that $\nabla C(\mathbf{q}) = \mathbf{p}$, the current price. Yet it seems more natural for the LP to specify this α. Or equivalently, we could insist that the LP encode the valid trades in the zero level set, and the LP could submit $\hat{C} := C - \alpha$, so that $C^{-1}(\alpha) = \hat{C}^{-1}(0)$; this is the approach we take in Protocol 1.

Recall that for two outcomes, the constant product potential $\varphi(\mathbf{q}) = q_1 q_2$ has an equivalent cost function (Chen and Pennock, 2007; Frongillo et al., 2023) given by $C(\mathbf{q}) = \frac{1}{2}\left(q_1 + q_2 + \sqrt{4\alpha^2 + (q_1 - q_2)^2}\right)$. Taking the 0-level set, we have $q_1 + q_2 + \sqrt{4\alpha^2 + (q_1 - q_2)^2} = 0$ which implies $q_1 + q_2 < 0$. Then $(q_1 + q_2)^2 = 4\alpha^2 + (q_1 - q_2)^2$, which reduces to $q_1 q_2 = \alpha^2$. By the first observation, we must have $\mathbf{q} \prec \mathbf{0}$, giving no liability. While one could take $\hat{C}(\mathbf{q}) = C(\mathbf{q}) + 1$ as well, to arrive at the same liquidity level, one can check that C gives rise to the minimal deposit required for that level.

4.5 Practical considerations

When implementing Protocol 1 in practice, there is a tradeoff between LPs' expressiveness and the computational cost of running the protocol. This tradeoff is a strong consideration in DeFi, as all computations have to be done on-chain; see § 6. As prediction markets are typically not so constrained, much more expressive protocols are possible. For example, one could allow LPs to specify their liquidity functions via polynomials of bounded degree, or weighted sums of basis functions, or any computationally convenient class of functions that well approximate any possible liquidity allocation.

5 Equivalence of interpretations

As we have argued informally, one can regard liquidity provisioning as (a) recruiting multiple market makers, which then (b) process trades in parallel. We now study (b) formally, how multiple market makers process trades in parallel, showing that four natural ways to interpret this parallelism are all equivalent. The equivalence of (a) in these interpretations, the process of recruiting market makers and securing deposits, is then straightforward (§ A.3).

5.1 Four interpretations of parallel market making

To begin, we revisit the interpretation of liquidity provisioning as running several market makers in parallel, and show that four natural interpretations of this idea lead to equivalent protocols. For convenience,[6] we outline Protocol 3 in Appendix A, which is equivalent to our main Protocol 1, but more closely resembles a scoring rule market.

In interpretations 1-3, each market maker i is specified by a cost function C_i, and a state $\mathbf{q}^i$. In 3, market makers instead each have a scoring rule S_i generated by a convex function $G_i = C_i^*$, and maintain a price $\mathbf{p}^i$. In all cases we assume the trader behaves rationally, in the sense that the overall trade is Pareto optimal: if $\mathbf{r}, \mathbf{r}'$ are both valid trades, and $\mathbf{r} \succeq \mathbf{r}'$, the trader chooses $\mathbf{r}$. (Recall that trades are oriented toward the trader, so here $\mathbf{r}$ gives the trader weakly more of each security.)

In each interpretation, we capture the market state by the collection of liability vectors $\{\mathbf{q}^i \in \mathbb{R}^n\}_i$. After a trade $\mathbf{r} = \sum_i \mathbf{r}^i$, the state updates to $\{\mathbf{q}^i + \mathbf{r}^i\}_i$. The set of consistent market prices are defined to be $\partial C_i(\mathbf{q}^i)$ for interpretations 1-3, and an analogous definition for 4. We say the overall market state is *coherent* if there is a consistent price $\mathbf{p} \in \operatorname{relint} \Delta_n$ for all market makers simultaneously.

1. **The trader selects a valid trade from each market maker and executes them all.** Formally, for each market maker i the trader selects $\mathbf{r}^i$ such that $C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)$, for a total trade of $\mathbf{r} = \sum_i \mathbf{r}^i$.

2. **The trader continuously trades at the most favorable price and at some point stops.** Recall that we can interpret a cost function C_i as quoting a cost $C_i(\mathbf{q}^i + \mathbf{v}^i) - C_i(\mathbf{q}^i)$ for

[6] In many ways, for blockchain applications, Protocol 3 is the more practically appealing of the two. Instead of specifying a trade directly, the "inversion" from proposed trades to implied prices is handled off-chain by the trader. In practice, the price function from Protocol 4 is computationally expensive; an implementation in line with Protocol 3 where traders specify a new price would alleviate this burden on the market maker, while still giving just as flexible an interface. The remaining computation, of the implied liability vectors, is more straightforward.

each bundle of assets/securities $\mathbf{v}^i \in \mathbb{R}^n$. Formally, in this interpretation, the trader specifies a direction $\mathbf{v} \in \mathbb{R}^n$, and a stopping point α, and continuously purchases $\mathbf{v}dt$ for the smallest price $C_i'(\mathbf{q}^i; \mathbf{v})$ over all i, for α units of time. Here $C_i'(\mathbf{q}^i; \mathbf{v}) := \lim_{h \to 0^+} \frac{C_i(\mathbf{q}^i + h\mathbf{v}) - C_i(\mathbf{q}^i)}{h}$ is the directional derivative of C_i. In other words, the trades are of the form $(\mathbf{v} - C_i'(\mathbf{q}^i; \mathbf{v})\mathbf{1})dt$, where we recall that the numeraire is simply $\mathbf{1}$. Crucially, we also allow the trader to take advantage of any arbitrage opportunity that arises from this continuous trade: after the α units of time, the trader may place trades $\{\hat{\mathbf{r}}_i\}_i$ if they have negative net cost and $\sum_i \hat{\mathbf{r}}_i = \mathbf{0}$.

3. **A fully centralized market maker using the infimal convolution.** This interpretation corresponds to the rules for trade in Protocol 1. Formally, the trader selects any $\mathbf{r} \in \mathbb{R}^n$ such that $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$, where $C = \bigwedge_i C_i$ and $\mathbf{q} = \sum_i \mathbf{q}^i$. The central market maker first gives $\mathbf{r}$ to the trader. Behind the scenes, it then computes a split $\mathbf{r} = \sum_i \mathbf{r}^i$ such that $C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)$, whose existence we establish below, and executes these trades in each constituent market maker.

4. **Each market maker uses a market scoring rule, and the trader is paid according to the sum of them.** Formally, each market maker has a scoring rule $S_i(\mathbf{p}, y) = G_i(\mathbf{p}) + \langle \mathbf{d}G_{\mathbf{p}}, \boldsymbol{\delta}^y - \mathbf{p} \rangle$ where $G_i = C_i^*$ and $\{\mathbf{d}G_{\mathbf{p}} \in \partial G(\mathbf{p}) \mid \mathbf{p} \in \operatorname{relint} \Delta_n\}$ is an arbitrary selection of subgradients. Market maker i maintains a price vector $\mathbf{p}^i \in \operatorname{relint} \Delta_n$, and the trader may choose any $\hat{\mathbf{p}}^i \in \operatorname{relint} \Delta_n$, resulting in the trade $\mathbf{r}^i = S_i(\hat{\mathbf{p}}^i, \cdot) - S_i(\mathbf{p}^i, \cdot) \in \mathbb{R}^n$. See Protocol 3. For the purposes of comparing interpretations, define the set of consistent prices as $\{\mathbf{p} \in \operatorname{relint} \Delta_n \mid \forall i \; S_i(\mathbf{p}, \cdot) = S_i(\mathbf{p}^i, \cdot)\}$.

5.2 Technical definitions

Definition 7 (Smoothness of G)**.** *We say a convex function $G : \Delta_n \to \mathbb{R}$ is* smooth *if its 1-homogenous extension $\overline{G} : \mathbb{R}^n_{\geq 0}$ is differentiable.*

The key class of the functions G we restrict to is as follows.

Definition 8 (Generating function)**.** *We say $G : \Delta_n \to \mathbb{R}$ is a* generating function *if it is convex, smooth, and bounded on Δ_n.*

The following term was coined in Abernethy et al. (2013) and used similarly to our setting: ensuring that the market price remains in the relative interior of the simplex.

Definition 9 (Pseudobarrier)**.** *A generating function G is a* pseudobarrier *if for any sequence $\{\mathbf{p}^j \in \operatorname{relint} \Delta_n\}_j$ converging to the relative boundary of Δ_n, and $\{\mathbf{q}^j \in \partial G(\mathbf{p}^j)\}_j$, then $\|\mathbf{q}^j\| \to \infty$.*

A common example of a pseudobarrier is (negative) Shannon entropy $G(\mathbf{p}) = \sum_y p_y \log p_y$. Another is the dual of the constant product market maker $G(\mathbf{p}) = -n(\prod_y p_y)^{1/n}$.[7]

We now give the sets of cost and generating functions used in the general protocols. Let $\mathcal{G}_n$ be the set of nonpositive generating functions $G : \Delta_n \to \mathbb{R}_{\leq 0}$, and $\mathcal{G}_n^* \subseteq \mathcal{G}_n$ those which are pseudobarriers.[8] Let $\mathcal{C}_n$ and $\mathcal{C}_n^*$ be the sets of conjugates of $\mathcal{G}_n$ and $\mathcal{G}_n^*$, respectively.

[7] To see that this dual is correct, one can observe that it is 1-homogeneous, and thus $S(\mathbf{p}, y) = -(\prod_{y'} p_{y'})^{1-1/n}(\prod_{y' \neq y} p_y)$. Now letting $\mathbf{x} = -S(\mathbf{p}, \cdot)$ be the corresponding reserve vector, and computing the product, we have $\prod_y q_y = \prod_y (\prod_{y'} p_{y'})^{1-1/n}(\prod_{y' \neq y} p_y) = 1$.

[8] Given any bounded generating function G, to obtain the optimal liability, we can simply replace it by the function $\mathbf{p} \mapsto G(\mathbf{p}) - \langle \mathbf{p}, \mathbf{q} \rangle$ where $q_i = G(\boldsymbol{\delta}^i)$.

5.3 Equivalence of the interpretations

Before stating the equivalence of these interpretations, we must address an important technical point. By our assumptions on G, the resulting scoring rule vectors are unique for each price $\mathbf{p} \in \operatorname{relint} \Delta_n$; see Lemma 5 in § A. This statement can fail to hold, however, on the relative boundary of the simplex Δ_n. Taking $G(\mathbf{p}) = \|\mathbf{p}\|_2^2$, or any LP that does not provide infinite liquidity at the boundary of Δ_n, the resulting G will not have a unique subgradient (even modulo $\mathbf{1}$) at those boundary points, meaning we will have $\mathbf{q}, \mathbf{q}' \in \mathbb{R}^n$ with $\mathbf{p} \in \partial C(\mathbf{q}) \cap \partial C(\mathbf{q}')$, but with $\mathbf{q}' - \mathbf{q} \neq \alpha \mathbf{1}$ for any α. The scoring rule S must pick just one of these vectors, meaning the scoring rule market (interpretation 4) will be strictly less expressive than the others. Somewhat conversely, consider G to be negative Shannon entropy, which gives rise to the log scoring rule $S(\mathbf{p}, y) = \log p_y$. Here the liquidity does become infinite on the boundary, and consequently the scoring rule vectors have infinite entries. These vectors cannot be captured by any $\mathbf{q} \in \mathbb{R}^n$, only in the limit.

For these two reasons, we restrict to $\operatorname{relint} \Delta_n$ in Lemma 5. The first issue is somewhat surmountable, however: if one defines $\mathbf{d}G_{\mathbf{p}}$ to be the most favorable $\mathbf{q}$ (modulo $\mathbf{1}$) tangent to G at a boundary point $\mathbf{p}$, then all of the Pareto optimal trades will still be available to the scoring rule market trader. Indeed, the only trades missing are those at the maximum possible price, which is Pareto-suboptimal for the trader anyway—consider a two outcome example when the price of the first security is 1, the maximum possible, so that purchasing the first security at this price is weakly worse than simply refraining from trade. Thus, while in Theorem 1 we assume there is a "log-like" LP, typically the market creator, which keeps the price away from the boundary, in principle one could generalize this statement using the ideas above to the case where liquidity runs out.

Theorem 1. *Let $C_i = G_i^*$ for generating functions G_i, where at least one G_i is a pseudobarrier. Then the interpretations 1-4 above are equivalent in the following sense: given a coherent market state, the set of valid trades is identical, and the resulting market state is coherent.*

Implicit in the proof of Theorem 1 is that, in interpretations 1, 2, and 4, if the market state is not initially coherent, it becomes coherent after a sufficiently large trade.

6 Recovering and extending DeFi protocols

6.1 Conventional differences

There are several conventional differences to note between the AMM literatures in prediction markets and those in DeFi; we discuss them here as they will help readers of either literature understand protocols in the other. As noted previously, we consider all trades to be oriented towards the trader, even though in the DeFi literature, trades are typically oriented toward the market maker. A trade $\mathbf{r} \in \mathbb{R}^n$ oriented towards the trader means that the positive coordinates of the n-dimensional vector signify the amounts of assets given to the trader and negative coordinates signify the amounts of assets taken away from the trader and given to the AMM.

CFMMs in DeFi tend to track the reserves held by the market maker, whereas cost function prediction markets typically track their liabilities as a function of the eventual outcome. Hence, a vector $\mathbf{x}$ of reserves corresponds to a vector $-\mathbf{q}$ of liabilities. We have attempted to use both conventions, clearly marked, to keep the protocols as close to their respective literatures as possible.

With regard to prices, DeFi typically uses an "exchange-rate" version of the contract price: the rate at which one can exchange one asset for another. As alluded to above, taking advantage

Protocol 2 Uniswap V2

global constants β.
global variables $\mathbf{x} \in \mathbb{R}^2$, $k \in \mathbb{N}$, $\{\alpha^i \in \mathbb{R}_{\geq 0}\}_{i=0}^k$.
$\mathsf{price}(\mathbf{x}) := \frac{x_2}{x_1+x_2}$

function $\mathsf{Initialize}(\mathbf{x}^0 \in \mathbb{R}^2, \beta)$
    $(k, \beta) \leftarrow (0, \beta)$
    $\alpha^0 \leftarrow \sqrt{x_1^0 \cdot x_2^0}$

function $\mathsf{RegisterLP}(i = k+1)$
    $(k, q^i, \alpha^i) \leftarrow (k+1, 0, 0)$

function $\mathsf{ModifyLiquidity}(i \in \mathbb{N}, \alpha' \geq 0)$
    $p = \mathsf{price}(\mathbf{x})$
    request $\mathbf{x}' = \left((\alpha' - \alpha^i)\sqrt{\frac{1-p}{p}}, (\alpha' - \alpha^i)\sqrt{\frac{p}{1-p}}\right)$ from LP i.
    $(\mathbf{x}, \alpha^i) \leftarrow (\mathbf{x} + \mathbf{x}', \alpha')$

function $\mathsf{ExecuteTrade}(\mathbf{r} \in \mathbb{R}^2)$
    check $\varphi(\mathbf{x}) = \varphi(\mathbf{x} - \mathbf{r})$, where $\varphi(\mathbf{x}) = x_1 x_2$.
    pay $\frac{\beta\alpha^i}{\alpha}(-\mathbf{r})_+$ to LP i, where $\alpha = \sum_i \alpha^i$.
    $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{r}$.

of the structure of cost functions, we instead adopt a *normalized price* convention. One can view normalized prices $\mathbf{p} \in \Delta_n$ as an exchange rate between assets and the "grand bundle" $\mathbf{1}$ of all assets. That is, p_i denotes the instantaneous price, in units of $\mathbf{1}$, to purchase asset i.

Converting between the two conventions is straightforward. Given normalized prices $\mathbf{p}$, one can simply define the exchange rate between i and j as $\hat{p}_{ij} = p_i/p_j$. Conversely, given pairwise exchange rates, one can define $\mathbf{x} = (1, \hat{p}_{21}, \hat{p}_{31}, \ldots, \hat{p}_{n1})$ and take $\mathbf{p} = \mathbf{x}/\|\mathbf{x}\|_1$. The conversion simplifies in the case of two assets, as $\hat{p}_{12} = \frac{p}{1-p}$ and $p = \hat{p}_{12}/(\hat{p}_{12}+1)$.

6.2 Uniswap V2

Most DeFi protocols use AMMs to facilitate trades between only two assets due to transaction gas costs. AMMs handling more assets are computationally expensive. Hence in this subsection and the next, we similarly restrict our attention to two assets/outcomes. Uniswap V2, introduced by Adams et al. (2020), is a commonly used AMM in the Ethereum ecosystem to trade assets and has the functional invariant $\varphi_\alpha(\mathbf{x}) = x_1 x_2 = \alpha^2$. In line with DeFi conventions, we track the reserve vector $\mathbf{x}$, so that x_1 and x_2 represent the amount of assets 1 and 2, respectively, held by the market maker. We will still use normalized prices, so protocols mentioned here slightly differ from the exchange rate model of price commonly seen in the DeFi literature. The normalized price of the first asset can be computed as $p = \frac{x_2}{x_1+x_2}$, while the exchange rate model has $\hat{p} = \frac{x_2}{x_1}$. We refer the reader to Fan et al. (2022, 2023) for a detailed breakdown of Uniswap V2 mechanics.

We state the Uniswap V2 protocol and show that it is a special case of Protocol 4. Uniswap V2 restricts how a LP can add or remove their liquidity by constraining them to use the same base function $g_0 = -2\sqrt{p(1-p)}$ and only express their liquidity using parameter α^i where $g_i = \alpha^i g_0$. Recall that from Fan et al. (2022), the bundle required to change liquidity from α^i to α' while

keeping the price invariant is $\left(\frac{\alpha'-\alpha^i}{\sqrt{\hat{p}}}, (\alpha'-\alpha^i)\sqrt{\hat{p}}\right)$. Using normalized prices, this is represented as $\left((\alpha'-\alpha^i)\sqrt{\frac{1-p}{p}}, (\alpha'-\alpha^i)\sqrt{\frac{p}{1-p}}\right)$ in our protocol. We note that instead of skimming γ from $(-\mathbf{r})_+$ for trading fees, we ask for $\beta(-\mathbf{r})_+$ from the trader when they request the trade $\mathbf{r}$. These two fee schemes are equivalent when $\beta = \frac{\gamma}{1+\gamma}$.

Lemma 1. *(informal) Protocol 2 is a special case of Protocol 4 for specific restrictions on* $\mathcal{G}_{\text{init}}$.

We defer the formal statements and proofs to Appendix B. There, we show that the liability vectors from the latter indeed satisfy the constant product invariant for our choice of generating functions using Propositions 4,5.

6.3 Uniswap V3 and general bucketing

Uniswap V2 requires LPs to provide liquidity on the entire price space, and while this restriction may look intuitive, it is suboptimal since liquidity allocations far from the current price may not be used. For example, a market that trades securities on a sports game might not benefit from having liquidity at prices in the $(0, 0.1)$ range, say. Moreover, when LPs provide liquidity, they take on the risk of price volatility, and ideally we would like to allow them to bound that risk. Yet on the other hand, it is computationally challenging to maintain an infinitely flexible LP protocol.

Motivated by these concerns, DeFi's Uniswap V3 protocol, (Adams et al., 2021), partitions the price space, allowing each LP i to contribute a proportion α^{ij} of liquidity on any price bucket $[a_j, b_j]$ of their choosing. We refer the reader to Fan et al. (2022, 2023) for a detailed analysis of Uniswap V3. In Appendix B.4 we show that Uniswap V3, given by Protocol 5, can be written as a special case of our general protocol.

In this section, we *generalize* the idea of bucketing with regards to an arbitrary cost function C. For ease of exposition, let us restrict again to two outcome setting. We do so as this allows for LPs to be more expressive, depending on the number of price intervals and also keeps the complexity of implementation from blowing up. Let $G(\mathbf{p})$ indicate the corresponding dual where $p_2 = 1 - p_1$. Let $\ell^{(j)}$ be its liquidity function restricted to the j-th price interval $[a_j, b_j]$ of p_1 and be given by

$$\ell^{(j)} = \nabla^2 \overline{G} \mathbf{1}_{p_1 \in [a_j, b_j]}$$

Let the liability vector associated with the corresponding cost function dual for $\ell^{(j)}$ be given below. We show the workings of this in Appendix B.2.

$$\mathsf{liability}(C^{(j)}) = \begin{pmatrix} S_g(\max\{a_j, p_1\}, 1) - S_g(\max\{b_j, p_1\}, 1) \\ S_g(\min\{b_j, p_1\}, 0) - S_g(\min\{a_j, p_1\}, 0) \end{pmatrix}$$

where $\mathbf{p} = (p_1, p_2)$ is the current price, $S_g(p, y) = g(p) + g'(p)(y - p)$ and $G(\mathbf{p}) = g(p_1)$ as discussed in § 2.3,3.1 and $C^{(j)} = (\iint \ell^{(j)})^*$.

In Table 1, we use the above general liability vector to state the liability vectors for Uniswap V3, as well as new bucketing protocols where we use $G(\mathbf{p}) = p_1 \log p_1 + p_2 \log p_2$ from LMSR and $S_g(p, y) = -(p - y)^2$ of the Brier scoring rule as the base "shapes." We give detailed workings for the LMSR bucketing protocol in Appendix B.3.

	Uniswap V3	LMSR	Brier
$p < a_j$	$\alpha_j \begin{pmatrix} \sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-a_j}{a_j}} \\ 0 \end{pmatrix}$	$\begin{pmatrix} \log \frac{a_j}{b_j} \\ 0 \end{pmatrix}$	$\begin{pmatrix} (1-b_j)^2 - (1-a_j)^2 \\ 0 \end{pmatrix}$
$p \in [a_j, b_j]$	$\alpha_j \begin{pmatrix} \sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-p}{p}} \\ \sqrt{\frac{a_j}{1-a_j}} - \sqrt{\frac{p}{1-p}} \end{pmatrix}$	$\begin{pmatrix} \log \frac{p}{b_j} \\ \log \frac{1-p}{1-a_j} \end{pmatrix}$	$\begin{pmatrix} (1-b_j)^2 - (1-p)^2 \\ a_j^2 - p^2 \end{pmatrix}$
$p > b_j$	$\alpha_j \begin{pmatrix} 0 \\ \sqrt{\frac{a_j}{1-a_j}} - \sqrt{\frac{b_j}{1-b_j}} \end{pmatrix}$	$\begin{pmatrix} 0 \\ \log \frac{1-b_j}{1-a_j} \end{pmatrix}$	$\begin{pmatrix} 0 \\ a_j^2 - b_j^2 \end{pmatrix}$

Table 1: Liability vectors for bucketing liquidity protocols.

6.4 New protocols

In some cases, the generalized bucketing scheme we provide in § 6.3 is still overly restrictive on the expressivity of LPs, depending on the size of the price intervals. It may also be unnatural for LPs to specify their liquidity allocation via discontinuous liquidity functions. Fortunately, our general protocol easily allows one to generate more expressive restricted protocols, for particular families of C (equivalently g for two assets) functions which still allow for efficient computations. For example, one could use "soft" liquidity buckets where liquidity continuously "fades" in and out around a target price a_j. We describe this protocol and others in Appendix C.

In the case of n assets, one can generalize the idea of buckets, though now the task of partitioning the $d = n - 1$ dimensional price space becomes nontrivial. One promising approach would be to partition into a cell complex of convex polytopes, specifically a power diagram, (Aurenhammer, 1987). One could define indicator functions as above to allow LPs to allocate liquidity uniformly (or according to some base shape) across each of these regions. In Appendix C we also give a piecewise linear protocol which could be similarly adapted to these polyhedral regions, where the linear pieces align with the regions

7 Discussion and open directions

We have given a general protocol for liquidity provisioning in prediction markets, and more broadly any automated market making setting including decentralized finance (DeFi). In a sense that we formalize in § 3 and § 5, this protocol is maximally expressive, though as discussed in § 4.5 and § 6 it can be restricted for computational convenience. One natural direction for future work is to further explore specific restrictions, and study the tradeoffs in expressiveness and computation, and the implications for market efficiency.

One such restriction relevant to prediction markets is to restrict to an *incomplete market*, where not every outcome has its own Arrow–Debreu security. This too is a special case of our protocol, where one restricts to cost functions that are flat outside of the incomplete subspace of possible bundles of securities.

Beyond these directions, one particularly pressing future direction is to study the design of the fee function. As discussed in § 6, fee functions in DeFi take the form $\beta \mathbf{r}_+$ for a constant $\beta > 0$, where $\mathbf{r}_+$ denotes the vector with only the positive coordinates of $\mathbf{r}$. In particular, this fee is itself a bundle of assets, which in prediction markets corresponds to a bundle of outcome-contingent

securities. As we describe in § D, however, for more than 2 assets, this commonly used fee breaks the key budget-balance property that the trader's fee covers the sum of all the LP fees. The norm-based fee we propose in § 4.2 does balance the budget, but is complex for LPs and may not behave well. Designing fees with useful properties thus remains an important line of future work.

Acknowledgments

This material is based upon work supported by the Ethereum Foundation, and the National Science Foundation under Grant Nos. IIS-2045347 and DMS-1928930, the latter while the second author was in residence at the Mathematical Sciences Research Institute in Berkeley, California, during the Fall 2023 semester. Part of the research was conducted when second author was an intern at Ethereum Foundation. We thank Davide Crapis, James Grugett, Ciamac Moallemi, Alex Solleiro, and Bo Waggoner for several interesting discussions and ideas. We also thank Guillermo Angeris for suggestions regarding the piecewise linear market maker.

References

Jacob Abernethy, Yiling Chen, and Jennifer Wortman Vaughan. Efficient market making via convex optimization, and a connection to online learning. *ACM Transactions on Economics and Computation*, 1(2):12, 2013. URL http://dl.acm.org/citation.cfm?id=2465777.

Jacob D. Abernethy, Rafael M. Frongillo, Xiaolong Li, and Jennifer Wortman Vaughan. A General Volume-parameterized Market Making Framework. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, pages 413–430, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2565-3. doi: 10.1145/2600057.2602900. URL http://doi.acm.org/10.1145/2600057.2602900.

Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap v2 core, 2020. URL https://uniswap.org/whitepaper.pdf.

Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core, 2021. URL https://uniswap.org/whitepaper-v3.pdf.

Guillermo Angeris and Tarun Chitra. Improved price oracles: Constant function market makers. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 80–91, 2020.

Guillermo Angeris, Alex Evans, Tarun Chitra, and Stephen Boyd. Optimal routing for constant function market makers. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, EC '22, page 115–128, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391504. doi: 10.1145/3490486.3538336. URL https://doi.org/10.1145/3490486.3538336.

Guillermo Angeris, Tarun Chitra, Theo Diamandis, Alex Evans, and Kshitij Kulkarni. The geometry of constant function market makers. *arXiv preprint arXiv:2308.08066*, 2023.

F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987. doi: 10.1137/0216006. URL https://doi.org/10.1137/0216006.

Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of automated market makers in defi. *Logical Methods in Computer Science*, Volume 18, Issue 4, 2022. ISSN 1860-5974. doi: 10.46298/lmcs-18(4:12)2022. URL http://dx.doi.org/10.46298/lmcs-18(4:12)2022.

Base. Gas use in ethereum transactions, 2025. URL https://docs.base.org/base-learn/docs/introduction-to-ethereum/gas-use-in-eth-transactions/. Retrieved 2/8/2025.

G.W. Brier. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78 (1):1–3, 1950. ISSN 1520-0493.

Y. Chen and D.M. Pennock. A utility framework for bounded-loss market makers. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, pages 49–56, 2007.

Yiling Chen, Mike Ruberry, and Jenn Wortman Vaughan. Cost function market makers for measurable spaces. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 785–802, 2013. URL http://dl.acm.org/citation.cfm?id=2482608.

Theo Diamandis, Max Resnick, Tarun Chitra, and Guillermo Angeris. An efficient algorithm for optimal routing through constant function market makers, 2023. URL https://arxiv.org/abs/2302.04938.

Miroslav Dudik, Sebastien Lahaie, and David M. Pennock. A tractable combinatorial market maker using constraint generation. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, page 459–476, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450314152. doi: 10.1145/2229012.2229047. URL https://doi.org/10.1145/2229012.2229047.

Miroslav Dudík, Rafael Frongillo, and Jennifer Wortman Vaughan. Market Making with Decreasing Utility for Information. *Conference on Uncertainty in Artificial Intelligence*, 2014.

Miroslav Dudík, Xintong Wang, David M. Pennock, and David M. Rothschild. Log-time prediction markets for interval securities, 2021. URL https://arxiv.org/abs/2102.07308.

Zhou Fan, Francisco Marmolejo-Cossío, Ben Altschuler, He Sun, Xintong Wang, and David C. Parkes. Differential liquidity provision in uniswap v3 and implications for contract design, 2022. URL https://arxiv.org/abs/2204.00464.

Zhou Fan, Francisco Marmolejo-Cossío, Daniel J. Moroz, Michael Neuder, Rithvik Rao, and David C. Parkes. Strategic liquidity provision in uniswap v3, 2023. URL https://arxiv.org/abs/2106.12033.

Rafael Frongillo, Maneesha Papireddygari, and Bo Waggoner. An axiomatic characterization of cfmms and equivalence to prediction markets, 2023. URL https://arxiv.org/abs/2302.00196.

Tilmann Gneiting and Adrian E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

James Grugett. Multiple choice markets, 2023. URL https://news.manifold.markets/p/multiple-choice-markets. Retrieved 11/14/2023.

R. Hanson. Combinatorial Information Market Design. *Information Systems Frontiers*, 5(1):107–119, 2003.

Xiaolong Li and Jennifer Wortman Vaughan. An axiomatic characterization of adaptive-liquidity market makers. In *ACM EC*, 2013.

Manifold. Maniswap. https://www.notion.so/Maniswap-ce406e1e897d417cbd491071ea8a0c39, 2022.

Jason Milionis, Ciamac C Moallemi, and Tim Roughgarden. Complexity-approximation trade-offs in exchange mechanisms: Amms vs. lobs. In *International Conference on Financial Cryptography and Data Security*, pages 326–343. Springer, 2023.

Jason Milionis, Ciamac C. Moallemi, and Tim Roughgarden. A Myersonian Framework for Optimal Liquidity Provision in Automated Market Makers. In Venkatesan Guruswami, editor, *15th Innovations in Theoretical Computer Science Conference (ITCS 2024)*, volume 287 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81:1–81:19, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-309-6. doi: 10.4230/LIPIcs.ITCS.2024.81. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITCS.2024.81.

Vijay Mohan. Automated market makers and decentralized exchanges: a defi primer. *Financial Innovation*, 8(20), 2022. doi: 10.1186/s40854-021-00314-5. URL http://dx.doi.org/10.1145/3570639.

Long Nguyen. Minswap - multi-pool decentralized exchange on cardano. URL https://docs.minswap.org/get-started/whitepaper. 2021.

A Othman and T Sandholm. Liquidity-Sensitive Automated Market Makers via Homogeneous Risk Measures, 2011.

Abraham Othman, David M Pennock, Daniel M Reeves, and Tuomas Sandholm. A practical liquidity-sensitive automated market maker. *ACM Transactions on Economics and Computation (TEAC)*, 1(3):1–25, 2013.

Evgeni Y. Ovcharov. Existence and uniqueness of proper scoring rules. *J. Mach. Learn. Res.*, 16: 2207–2230, 2015. ISSN 1532-4435,1533-7928.

Evgeni Y. Ovcharov. Proper scoring rules and Bregman divergence. *Bernoulli*, 24(1):53 – 79, 2018. doi: 10.3150/16-BEJ857. URL https://doi.org/10.3150/16-BEJ857.

R.T. Rockafellar. *Convex analysis*, volume 28 of *Princeton Mathematics Series*. Princeton University Press, 1997.

Jan Christoph Schlegel, Mateusz Kwaśnicki, and Akaki Mamageishvili. Axioms for constant function market makers, 2022. URL https://arxiv.org/abs/2210.00048.

Thomas Strömberg. *A study of the operation of infimal convolution*. PhD thesis, Lule tekniska universitet, 1994.

Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols. *ACM Computing Surveys*, 55(11):1–50, February 2023. ISSN 1557-7341. doi: 10.1145/3570639. URL http://dx.doi.org/10.1145/3570639.

Ariel Zetlin-Jones, Bryan Routledge, and Yikang Shen. Automated exchange economies. 2024. URL https://www.andrew.cmu.edu/user/azj/files/rsz_slides.pdf.

Protocol 3 General protocol as parallel scoring rule markets

global constant $\mathcal{G}_{\text{init}} \subseteq \mathcal{G}^*$, $\mathcal{G}_{\text{LP}} \subseteq \mathcal{G}$, $\mathsf{fee}(), \mathsf{fee}_i() \in \mathbb{R}_{\geq 0}$.
global variables $k \in \mathbb{N}$, $\mathbf{p} \in \operatorname{relint} \Delta_n$, $\{G_i \in \mathcal{G}_{\text{LP}}\}_{i=0}^{k}$
$\mathsf{price}(G, \mathbf{q}) := (\nabla \overline{G})^{-1}(\mathbf{q})$
$\mathsf{liability}(G, p) := S_G(p, \cdot)$ where $S_G(p, \cdot) = \mathbf{d}G_{\mathbf{p}} + (G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p} \rangle)\mathbf{1}$.

function $\mathsf{Initialize}(\mathbf{p} \in \operatorname{relint} \Delta_n, G \in \mathcal{G}_{\text{init}})$
    $(k, \mathbf{p}, G_0) \leftarrow (0, \mathbf{p}, G)$
    market creator holds $-S_{G_0}(\mathbf{p}, \cdot)$ in reserve

function $\mathsf{RegisterLP}(i = k + 1)$
    $(k, G_k) \leftarrow (k + 1, 0)$

function $\mathsf{ModifyLiquidity}(i \in \mathbb{N}, G' \in \mathcal{G}_{\text{LP}})$
    request $S_{G_i}(\mathbf{p}, \cdot) - S_{G'}(\mathbf{p}, \cdot)$ from LP i
    $G_i \leftarrow G'$

function $\mathsf{ExecuteTrade}(\hat{\mathbf{p}} \in \operatorname{relint} \Delta_n)$
    $\mathbf{r} \leftarrow S_G(\hat{\mathbf{p}}, \cdot) - S_G(\mathbf{p}, \cdot)$ where $G = \sum_i G_i$
    pay $\mathsf{fee}(\mathbf{r}, \mathbf{p}, G)$ cash
    execute trade $\mathbf{r}$
    for each LP i **do**
        $\mathbf{r}^i \leftarrow S_{G_i}(\hat{\mathbf{p}}, \cdot) - S_{G_i}(\mathbf{p}, \cdot)$.
        pay $\mathsf{fee}_i(\mathbf{r}, \mathbf{p}, \{G_i\}_{i=1}^{k})$ in fees to LP i
        $\mathbf{q}^i \leftarrow \mathbf{q}^i + \mathbf{r}^i$

A Proofs From Section 5 and an alternative protocol

As some readers might be more familiar with Scoring Rule Markets (Hanson, 2003), we give Protocol 3 as an alternative to Protocol 1. Here we keep track of the market price $\mathbf{p}$ instead of liability vector $\mathbf{q}$ and use scoring rules instead of cost functions. We prove in Appendix A.3 that these two protocols are essentially equivalent under some mild conditions.

A.1 Technical lemmas

We begin with some standard facts from convex analysis.

Proposition 1 (Rockafellar (1997, Theorem 23.5))**.** *Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a closed convex function and f^* its conjugate. Then for all $\mathbf{x}, \mathbf{x}^* \in \mathbb{R}^n$ the following are equivalent:*

1. $\mathbf{x}^* \in \partial f(\mathbf{x})$
2. $\mathbf{x} \in \partial f^*(\mathbf{x}^*)$
3. $f(\mathbf{x}) + f^*(\mathbf{x}^*) = \langle \mathbf{x}^*, \mathbf{x} \rangle$

Proposition 2. *Let $f_i : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be convex for $i \in \{1, \ldots, k\}$ such that $\bigcap_i \operatorname{relint} \operatorname{dom} f_i \neq \emptyset$. Let $f = \sum_i f_i$. Then $f^* = \bigwedge_i f_i^*$, and the infimum in $\wedge$ in the definition of f^* is always attained. Moreover, for $\mathbf{v} \in \operatorname{relint} \operatorname{dom} f^*$, and any split $\mathbf{v} = \sum_i \mathbf{v}^i$, we have $f^*(\mathbf{v}) = \sum_i f_i^*(\mathbf{v}^i)$ if and only if $\bigcap_i \partial f_i(\mathbf{v}^i) \neq \emptyset$.*

Proof. The first statement follows from Rockafellar (1997, Theorem 16.4); we will prove the second. First suppose $f^*(\mathbf{v}) = \sum_i f_i^*(\mathbf{v}^i)$ for $\mathbf{v} \in \operatorname{relint} \operatorname{dom} f^*$ and $\mathbf{v} = \sum_i \mathbf{v}^i$. From Rockafellar (1997, Theorem 23.4), $\partial f^*(\mathbf{v}) \neq \emptyset$. Now Strömberg (1994, Theorem 3.6) gives $\bigcap_i \partial f_i^*(\mathbf{v}^i) = \partial f^*(\mathbf{v}) \neq \emptyset$. [9]

For the converse, let $\mathbf{x} \in \bigcap_i \partial f_i^*(\mathbf{v}^i)$ and define $\mathbf{v} = \sum_i \mathbf{v}^i$. From Proposition 1, $\mathbf{v}^i \in \partial f_i(\mathbf{x})$ for all i. Now Rockafellar (1997, Theorem 23.8) gives $\mathbf{v} = \sum_i \mathbf{v}^i \in \partial f(\mathbf{x})$. Proposition 1 again implies gives $f^*(\mathbf{v}) = \langle \mathbf{x}, \mathbf{v} \rangle - f(\mathbf{x}) = \sum_i \langle \mathbf{x}, \mathbf{v}^i \rangle - \sum_i f_i(\mathbf{x}) = \sum_i f_i^*(\mathbf{v}^i)$. □

We now prove several technical lemmas we use in the proof of Theorem 1.

Lemma 2. *Let G be a pseudobarrier and $C = G^*$. Then for all $\mathbf{q}, \hat{\mathbf{q}} \in \mathbb{R}^n$ we have $\partial C(\mathbf{q}) \subseteq \operatorname{relint} \Delta_n$ and $\hat{\mathbf{q}} \not\geq \mathbf{q} \implies C(\hat{\mathbf{q}}) > C(\mathbf{q})$.*

Proof. Let $\mathbf{p} \in \partial C(\mathbf{q})$. By Proposition 1, $\mathbf{q} \in \partial G(\mathbf{p})$. If $\mathbf{p} \notin \operatorname{relint} \Delta_n$, then we have an interior sequence $\{\mathbf{p}^j\}_j$ such that $\mathbf{p}^j \to_j \mathbf{p}$ and subgradients $\mathbf{q}^j \to_j \mathbf{q}$, violating the definition of a pseudobarrier. For this $\mathbf{p}$, we have $p_y > 0$ for all $y \in \{1, \dots, n\}$. As $\hat{\mathbf{q}} \not\geq \mathbf{q}$, we have $\hat{q}_y \geq q_y$ for all y, with at least one inequality strict. Thus by the subgradient inequality, $C(\hat{\mathbf{q}}) - C(\mathbf{q}) \geq \langle \mathbf{p}, \hat{\mathbf{q}} - \mathbf{q} \rangle > 0$, as desired. □

The following lemma is essentially a restatement of results due to Ovcharov (2018, 2015). It says that subgradients of generating functions G are unique modulo $\mathbf{1}$.

Lemma 3. *Let G be a generating function. Then for all $p \in \Delta_n$, and all $\mathbf{q}, \mathbf{q}' \in \partial G(\mathbf{p})$, there exists $\alpha \in \mathbb{R}$ such that $\mathbf{q}' = \mathbf{q} + \alpha \mathbf{1}$.*

Proof. Let $\mathbf{q} \in \partial G(\mathbf{p})$. We will show $\hat{\mathbf{q}} \in \partial \overline{G}(\mathbf{p})$, where $\hat{\mathbf{q}} = \mathbf{q} + (G(\mathbf{p}) - \langle \mathbf{q}, \mathbf{p} \rangle)\mathbf{1}$. By construction, $\langle \hat{\mathbf{q}}, \mathbf{p} \rangle = \langle \mathbf{q}, \mathbf{p} \rangle + G(\mathbf{p}) - \langle \mathbf{q}, \mathbf{p} \rangle = G(\mathbf{p})$. For all $\mathbf{x} \in \mathbb{R}^n_{\geq 0} \setminus \{\mathbf{0}\}$, we have

$$\begin{aligned}
\overline{G}(\mathbf{x}) &= \|\mathbf{x}\|_1 G(\mathbf{x}/\|\mathbf{x}\|_1) \\
&\geq \|\mathbf{x}\|_1 \left(G(\mathbf{p}) + \langle \mathbf{q}, \mathbf{x}/\|\mathbf{x}\|_1 - \mathbf{p} \rangle \right) \\
&= \|\mathbf{x}\|_1 \left(G(\mathbf{p}) + \langle \hat{\mathbf{q}}, \mathbf{x}/\|\mathbf{x}\|_1 - \mathbf{p} \rangle \right) \\
&= \|\mathbf{x}\|_1 \left(\langle \hat{\mathbf{q}}, \mathbf{x}/\|\mathbf{x}\|_1 \rangle + G(\mathbf{p}) - \langle \hat{\mathbf{q}}, \mathbf{p} \rangle \right) \\
&= \langle \hat{\mathbf{q}}, \mathbf{x} \rangle \\
&= \overline{G}(\mathbf{p}) - \langle \hat{\mathbf{q}}, \mathbf{p} \rangle + \langle \hat{\mathbf{q}}, \mathbf{x} \rangle \\
&= \overline{G}(\mathbf{p}) + \langle \hat{\mathbf{q}}, \mathbf{x} - \mathbf{p} \rangle \ .
\end{aligned}$$

Thus, every element of $\partial G(\mathbf{p})$, up to a shift by $\mathbf{1}$, is an element of $\partial \overline{G}(\mathbf{p})$. As the latter is a singleton set, by assumption on G, the result follows. □

Lemma 4. *Let $C = \bigwedge_i C_i$ where C_i^* are generating functions.*

Then for all $\mathbf{q} \in \mathbb{R}^n$, the infimum in the definition of $C(\mathbf{q})$ is attained, and $C(\mathbf{q}) = \sum_i C_i(\mathbf{q}^i)$ if and only there exists $\mathbf{p} \in \Delta_n$ such that $p \in \partial C_i(\mathbf{q}^i)$ for all i.

Proof. We have $\operatorname{dom} C_i^* = \Delta_n$ and $\operatorname{dom} C_i = \mathbb{R}^n$ for all i, so Proposition 2 applies.

□

[9] As needed for that result to apply, the assumption that there exists some $\mathbf{x} \in \bigcap_i \operatorname{relint} \operatorname{dom} f_i$ implies that f_i^* is bounded from below by the same affine function, namely one with gradient $\mathbf{x}$.

Lemma 5. *Let $C = G^*$ where G is a generating function. For $\mathbf{p} \in \operatorname{relint}\Delta_n$, let $S(\mathbf{p}, y) = G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \boldsymbol{\delta}^y - \mathbf{p}\rangle$, where $\{\mathbf{d}G_{\mathbf{p}} \in \partial G(\mathbf{p}) \mid \mathbf{p} \in \operatorname{relint}\Delta_n\}$ is a selection of subgradients.*

Then for all $\mathbf{p} \in \operatorname{relint}\Delta_n$ and $\mathbf{q} \in \mathbb{R}^n$, we have $(\mathbf{p} \in \partial C(\mathbf{q}) \wedge C(\mathbf{q}) = 0) \iff \mathbf{q} = S(\mathbf{p}, \cdot)$. In particular, $\{S(\mathbf{p}, \cdot) \mid \mathbf{p} \in \operatorname{relint}\Delta_n\} = \{\mathbf{q} \in C^{-1}(0) \mid \partial C(\mathbf{q}) \cap \operatorname{relint}\Delta_n \neq \emptyset\}$.

Proof. Let $\mathbf{q} = S(\mathbf{p}, \cdot)$. Then $C(\mathbf{q}) = C(\mathbf{d}G_{\mathbf{p}} + (G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle)\mathbf{1}) = G(\mathbf{d}G_p) + G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle = 0$ by Proposition 1. Furthermore, by the same theorem, $\mathbf{d}G_{\mathbf{p}} \in \partial G(\mathbf{p}) \iff \mathbf{p} \in \partial C(\mathbf{d}G_{\mathbf{p}})$, and by $\mathbf{1}$-invariance, $\partial C(\mathbf{d}G_{\mathbf{p}}) = \partial C(S(\mathbf{p}, \cdot))$. Thus, $\mathbf{p} \in \partial C(\mathbf{q})$.

Now let $\mathbf{q}$ such that $C(\mathbf{q}) = 0$, and take $\mathbf{p} \in \partial C(\mathbf{q})$; we will show $\mathbf{q} = S(\mathbf{p}, \cdot)$. By Proposition 1, $\mathbf{q} \in \partial G(\mathbf{p})$. From Lemma 3, we have $\partial G(\mathbf{p}) = \{\mathbf{d}G_{\mathbf{p}} + \alpha\mathbf{1} \mid \alpha \in \mathbb{R}\}$. Thus $\mathbf{q} = \mathbf{d}G_{\mathbf{p}} + \alpha\mathbf{1}$ for some $\alpha \in \mathbb{R}$. Now $0 = C(\mathbf{q}) = C(\mathbf{d}G_{\mathbf{p}} + \alpha\mathbf{1}) = C(\mathbf{d}G_{\mathbf{p}}) + \alpha = \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle - G(\mathbf{p}) + \alpha$ by Proposition 1. Thus $\alpha = G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle$, and we have $\mathbf{q} = \mathbf{d}G_{\mathbf{p}} + (G(\mathbf{p}) - \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle)\mathbf{1} = S(\mathbf{p}, \cdot)$, as desired. □

A.2 Proof of Theorem 1

Proof. We will show that 1 and 2 are each equivalent to 3, and 4 is equivalent to 1. For each, let $\{\mathbf{q}^i\}_i$ be the current market state, $\mathbf{q} = \sum_i \mathbf{q}^i$, $\mathbf{p}$ a consistent price, and $C = \bigwedge_i C_i$. From Lemma 4, price consistency implies $C(\mathbf{q}) = \sum_i C_i(\mathbf{q}^i)$, i.e., the $\mathbf{q}^i$ vectors achieve the infimum in the definition of the infimal convolution.

1. A trade for 3 satisfies the conditions for 1, so we only need to show that this choice is Pareto optimal for the trader; coherence will then follow by Lemma 4. More formally, let $\{\mathbf{r}^i\}_i$ satisfy $C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)$ for all i. Letting $\mathbf{r} = \sum_i \mathbf{r}^i$, from the definition of infimal convolution, $C(\mathbf{q} + \mathbf{r}) \leq \sum_i C_i(\mathbf{q}^i + \mathbf{r}^i) = \sum_i C_i(\mathbf{q}^i) = C(\mathbf{q})$. We wish to show that $\mathbf{r}$ is Pareto optimal if and only if $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$, or equivalently, that $\mathbf{r}$ is Pareto suboptimal if and only if $C(\mathbf{q} + \mathbf{r}) < C(\mathbf{q})$.

   Suppose first that we had some $\hat{\mathbf{r}} \not\geq \mathbf{r}$ such that $C_i(\mathbf{q}^i + \hat{\mathbf{r}}^i) = C_i(\mathbf{q}^i)$ where $\hat{\mathbf{r}} = \sum_i \hat{\mathbf{r}}^i$. From the same argument as above, we have $C(\mathbf{q} + \hat{\mathbf{r}}) \leq \sum_i C_i(\mathbf{q}^i + \hat{\mathbf{r}}^i) = \sum_i C_i(\mathbf{q}^i) = C(\mathbf{q})$. By Lemma 2, $C(\mathbf{q}+\hat{\mathbf{r}}) > C(\mathbf{q}+\mathbf{r})$, giving $C(\mathbf{q}+\mathbf{r}) < C(\mathbf{q})$. Conversely, suppose $C(\mathbf{q}+\mathbf{r}) \neq C(\mathbf{q})$, which from the inequality above implies $C(\mathbf{q}+\mathbf{r}) < C(\mathbf{q})$. Let $\hat{\mathbf{r}} = \mathbf{r} + (C(\mathbf{q}) - C(\mathbf{q}+\mathbf{r}))\mathbf{1} \not\geq \mathbf{r}$. By $\mathbf{1}$ invariance, we have $C(\mathbf{q} + \hat{\mathbf{r}}) = C(\mathbf{q})$. From part (3) of the proof below, there exists a split $\hat{\mathbf{r}} = \sum_i \hat{\mathbf{r}}^i$ such that of $C_i(\mathbf{q}^i + \hat{\mathbf{r}}^i) = C_i(\mathbf{q}^i)$. Thus, $\mathbf{r}$ was not a Pareto-optimal total trade.

2. Let $\mathbf{r}$ be a trade from interpretation 3, so that $C(\mathbf{q} + \mathbf{r}) = C(\mathbf{q})$ where $C = \bigwedge_i C_i$. Set $\mathbf{v} = \mathbf{r}$ and $\alpha = 1$ for interpretation 2. Let $\alpha_i \geq 0$ be the amount of $\mathbf{r}$ purchased from cost function i, and $\beta_i \in \mathbb{R}$ the total cost, so that the net trade from cost function i is $\mathbf{r}_i = \alpha_i\mathbf{r} - \beta_i\mathbf{1}$. Then we have $\sum_i \alpha_i = 1$, so that the net trade is $\mathbf{r} - \beta\mathbf{1}$ where $\beta = \sum_i \beta_i$. By definition of $\mathbf{r}_i$, the cost of trades, and the $\mathbf{1}$-invariance of C_i, we have $C_i(\mathbf{q}_i + \mathbf{r}_i) = C_i(\mathbf{q}_i)$.

   If $\beta = 0$ we are done. Otherwise, as $\mathbf{r}$ is Pareto optimal from part (1) above, we must have $\beta > 0$.

   From part (3) of the proof below, there exists a set of trades $\{\hat{\mathbf{r}}_i\}_i$ with $\sum_i \hat{\mathbf{r}}_i = \mathbf{r}$ such that $C_i(\mathbf{q}_i + \hat{\mathbf{r}}_i) = C_i(\mathbf{q}_i) = C_i(\mathbf{q}_i + \mathbf{r}_i - \beta_i\mathbf{1})$ for all i. Thus, from state $\{\mathbf{q}'_i\}_i := \{\mathbf{q}_i + \mathbf{r}_i - \beta_i\mathbf{1}\}_i$,

the trades $\{\mathbf{r}'_i\}_i := \{\hat{\mathbf{r}}_i - \mathbf{r}_i\}_i$ are an arbitrage, with $\sum_i \mathbf{r}'_i = \mathbf{r} - \mathbf{r} = \mathbf{0}$ and net cost

$$\begin{aligned}\sum_i C_i(\mathbf{q}'_i + \mathbf{r}'_i) - C_i(\mathbf{q}'_i) &= \sum_i C_i(\mathbf{q}_i + \hat{\mathbf{r}}_i - \beta_i\mathbf{1}) - C_i(\mathbf{q}_i + \mathbf{r}_i - \beta_i\mathbf{1}) \\ &= \sum_i C_i(\mathbf{q}_i + \hat{\mathbf{r}}_i - \beta_i\mathbf{1}) - C_i(\mathbf{q}_i + \hat{\mathbf{r}}_i) \\ &= -\sum_i \beta_i = -\beta < 0 \ .\end{aligned}$$

For the converse, let $\mathbf{r}$ be any net trade from the continuous trading process, and $\beta \leq 0$ the optimal net cost from any arbitrage. By part (1) and the argument above, we have $\beta = C(\mathbf{q}) - C(\mathbf{q}+\mathbf{r})$; taking $\hat{\mathbf{r}} = \mathbf{r} + (C(\mathbf{q}) - C(\mathbf{q}+\mathbf{r}))\mathbf{1}$,

we again split $\hat{\mathbf{r}}$ into $\{\hat{\mathbf{r}}_i\}_i$ and construct an arbitrage $\{\mathbf{r}'_i\}_i := \{\hat{\mathbf{r}}_i - \mathbf{r}_i\}_i$ which achieves net cost $C(\mathbf{q}+\mathbf{r}) - C(\mathbf{q}) = -\beta$.

3. Let $\mathbf{r}$ such that $C(\mathbf{q}+\mathbf{r}) = C(\mathbf{q})$. We must show that there exist $\{\mathbf{r}^i\}_i$ such that $C_i(\mathbf{q}^i+\mathbf{r}^i) = C_i(\mathbf{q}^i)$ and $\mathbf{r} = \sum_i \mathbf{r}^i$. From the definition of infimal convolution, $C(\mathbf{q}+\mathbf{r}) = \inf\{\sum_i C_i(\mathbf{v}^i) \mid \sum_i \mathbf{v}^i = \mathbf{q}+\mathbf{r}\}$. By Lemma 4, this infimum is attained by some $\{\mathbf{v}^i\}_i$. Define $\mathbf{r}^i := \mathbf{v}^i - \mathbf{q}^i + (C_i(\mathbf{q}^i) - C_i(\mathbf{v}^i))\mathbf{1}$. For the first condition, $C_i(\mathbf{q}^i+\mathbf{r}^i) = C_i(\mathbf{v}^i + (C_i(\mathbf{q}^i) - C_i(\mathbf{v}^i))\mathbf{1}) = C_i(\mathbf{q}^i)$. For the second,

$$\begin{aligned}\sum_i \mathbf{r}^i &= \sum_i \mathbf{v}^i - \sum_i \mathbf{q}^i + \sum_i (C_i(\mathbf{q}^i) - C_i(\mathbf{v}^i))\mathbf{1} \\ &= (\mathbf{q}+\mathbf{r}) - \mathbf{q} + \left(\sum_i C_i(\mathbf{q}^i) - \sum_i C_i(\mathbf{v}^i)\right)\mathbf{1} \\ &= \mathbf{r} + (C(\mathbf{q}) - C(\mathbf{q}+\mathbf{r}))\mathbf{1} = \mathbf{r} \ .\end{aligned}$$

Coherence again follows from Lemma 4.

4. We will show equivalence to interpretation 1.

Let $\alpha_i = C_i(\mathbf{q}^i)$ for all i, so that $C(\mathbf{q}^i - \alpha\mathbf{1}) = 0$. By Lemma 5, we may therefore write $\mathbf{q}^i = S(\mathbf{p}^i, \cdot) + \alpha_i\mathbf{1}$. From Lemma 4, we again have $C(\mathbf{q}) = \sum_i C(\mathbf{q}^i)$. From Lemma 5 again, and $\mathbf{1}$-invariance,

$$\begin{aligned}&\{\mathbf{r}^i \in \mathbb{R}^n \mid C_i(\mathbf{q}^i + \mathbf{r}^i) = C_i(\mathbf{q}^i)\} \\ &= \{\mathbf{r}^i \in \mathbb{R}^n \mid C_i(\mathbf{q}^i + \mathbf{r}^i - \alpha_i\mathbf{1}) = C_i(\mathbf{q}^i - \alpha_i\mathbf{1})\} \\ &= \{\mathbf{r}^i \in \mathbb{R}^n \mid C_i(S_i(\mathbf{p}^i, \cdot) + \mathbf{r}^i) = C_i(S_i(\mathbf{p}^i, \cdot))\} \\ &= \{\mathbf{r}^i \in \mathbb{R}^n \mid C_i(S_i(\mathbf{p}^i, \cdot) + \mathbf{r}^i) = 0\} \\ &= \{\hat{\mathbf{q}}^i - S(\mathbf{p}^i, \cdot) \mid C_i(\hat{\mathbf{q}}^i) = 0\} \\ &= \{S(\hat{\mathbf{p}}^i, \cdot) - S(\mathbf{p}^i, \cdot) \mid \hat{\mathbf{p}}^i \in \operatorname{relint}\Delta_n\} \ .\end{aligned}$$

We conclude that the possible trades $\{\mathbf{r}^i\}_i$ in interpretation 1, such that $C_i(\mathbf{q}^i + \hat{\mathbf{r}}^i) = C_i(\mathbf{q}^i)$ for all i, are exactly the same as the trades $\{S_i(\hat{\mathbf{p}}^i, \cdot) - S_i(\mathbf{p}^i, \cdot)\}_i$ allowed in interpretation 4; we have simply reparameterized the trades by $\{\hat{\mathbf{p}}^i\}_i$. □

A.3 Equivalence of the full protocols and practical considerations

Theorem 1 tells us that the process of trading in Protocols 1 and 3 are the same. The equivalence of the rest of the protocol follows from Lemma 5, as $\mathsf{liability}(C) = S_G(\mathbf{p}, \cdot)$ where $\mathbf{p} \in \partial C(\mathbf{q})$.

The two-outcome case is similar; we need only verify the translation from G and C to thei 1-dimensional counterparts. Letting $G(\mathbf{p}) = g(p_1)$, we have $\overline{G}(\mathbf{x}) = (x_1 + x_2)g(x_1/(x_1 + x_2))$ which is differentiable. Lemma 3 now gives $\partial G(\mathbf{p}) = \{(g'(p_1), 0) + \alpha\mathbf{1} \mid \alpha \in \mathbb{R}\}$ and thus $S_G(\mathbf{p}, \cdot) = \mathbf{d}G_{\mathbf{p}} + (G(\mathbf{p}) + \langle \mathbf{d}G_{\mathbf{p}}, \mathbf{p}\rangle)\mathbf{1} = (g'(p_1), 0) + (g(p_1) - p_1 g'(p_1))\mathbf{1} = \mathsf{liability}(g, p_1)$. Computing the conjugate, we have

$$\begin{aligned} G^*(\mathbf{q}) &= \sup_{\mathbf{p}\in\Delta_2} \langle \mathbf{p}, \mathbf{q}\rangle - G(\mathbf{p}) \\ &= \sup_{p\in[0,]1} pq_1 + (1-p)q_2 - g(p) \\ &= \left(\sup_{p\in[0,1]} p(q_1 - q_2) - g(p)\right) + q_2 \\ &= g^*(q_1 - q_2) + q_2 \ , \end{aligned}$$

as desired. Finally, to verify $\mathsf{price}(\cdot)$, note that $c' = (g')^{-1}$ whenever both derivatives are defined. By assumption on $\mathcal{G}_{\text{init}}$, any argument to price is both differentiable and strictly convex, and thus c is differentiable. We have now established Proposition 3.

B Proofs and additional working from Section 6

B.1 Proofs related to Uniswap V2 in Section 6

Before we analyze Uniswap, we introduce a simpler version of Protocol 1 for the two outcome case. It helps us reason about liquidity functions in a much convenient way. The above discussed observations in Appendix A aids us in writing Protocol 4. Reminder that for two outcome case, we use g, c instead of G, C. Refer to section 3.1 for a revision. The fact that $C(S_g(p, \cdot)) = 0$ combined with the fact that $\nabla C(S_g(p, \cdot)) = (p, 1-p)$ means that one can replace $\mathsf{liability}(C)$ in Protocol 1 line 3 with $\mathsf{liability}(g, p) := S_g(p, \cdot)$, where p is the current price of asset 1. Similarly, the trade check (line 14) and optimal split (line 17) in Protocol 1 can be done more straightforwardly using $S_g(p, \cdot)$, without the need to compute infimal convolutions explicitly. We skip the formal proof of the equivalence of Protocols 1 and 4 to § 5, as it readily follows from a slightly weaker equivalence between the corresponding n-asset protocols (Theorem 1).

Let $\mathcal{G}$ be the set of functions $g : [0, 1] \to \mathbb{R}_{\leq 0}$ which are convex, continuously differentiable, and bounded. Let $\mathcal{G}^* \subseteq \mathcal{G}$ be those which additionally have $|g'(p)| \to \infty$ as $p \to 0$ or $p \to 1$.

Proposition 3. *Let $\mathcal{G}_{\text{init}} \subseteq \mathcal{G}^*$ be a set of functions which are strictly convex. Then for $n = 2$ Protocol 1 is equivalent to Protocol 4 for the choices $c_i = g_i^*$ where $C_i(\mathbf{q}) = c_i(q_1 - q_2) + q_2$.*

Proposition 4. *For $\mathcal{G}_{\text{init}} = \{\alpha g_0 \mid \alpha > 0\}$, $\mathcal{G}_{\text{LP}} = \{\alpha g_0 \mid \alpha \geq 0\}$ where $g_0(p) = -2\sqrt{p(1-p)}$ in Protocol 4, $\mathsf{liability}(g, p) = -\alpha\left(\sqrt{\frac{1-p}{p}}, \sqrt{\frac{p}{1-p}}\right)^\top$ for $g = \alpha g_0$. The vector $\mathbf{x}$ of reserves in Protocol 2 satisfies $x_1 \cdot x_2 = \alpha^2$ for some $\alpha > 0$ iff $\mathbf{q} = \mathsf{liability}(g, \mathsf{price}(g, p))$, where $\mathbf{x} = -\mathbf{q}$.*

Protocol 4 General two-asset protocol via liquidity functions

global constant $\mathcal{G}_{\text{init}} \subseteq \mathcal{G}^*$, $\mathcal{G}_{\text{LP}} \subseteq \mathcal{G}$, $\mathsf{fee}(), \mathsf{fee}_i() \in \mathbb{R}_{\geq 0}$
global variables $k \in \mathbb{N}$, $\{\mathbf{q}^i \in \mathbb{R}^2\}_{i=0}^k$, $\{g_i \in \mathcal{G}_{\text{LP}}\}_{i=0}^k$
$\mathsf{price}(g, \mathbf{q}) := (g')^{-1}(q_1 - q_2)$
$\mathsf{liability}(g, p) := S_g(p, \cdot)$ where $S_g(p, \cdot) = (g'(p), 0) + (g(p) - p \cdot g'(p))\mathbf{1}$.

function $\mathsf{Initialize}(\mathbf{q} \in \mathbb{R}^2, g \in \mathcal{G}_{\text{init}})$ ▷ Equivalently, the market creator can specify $\ell = g''$
    $(k, \mathbf{q}^0, g_0) \leftarrow (0, \mathbf{q}, g)$
    check $\mathbf{q}^0 = \mathsf{liability}(g_0, \mathsf{price}(g_0, \mathbf{q}))$

function $\mathsf{RegisterLP}(i = k + 1)$
    $(k, \mathbf{q}^i, g_i) \leftarrow (k + 1, 0, 0)$

function $\mathsf{ModifyLiquidity}(i \in \mathbb{N}, g \in \mathcal{G}_{\text{LP}})$ ▷ Equivalently, the LP can specify $\ell = g''$
    request $\mathbf{r}^i = \mathbf{q}^i - \mathsf{liability}(g, \mathsf{price}(g_0, \mathbf{q}^0))$ from LP i
    $(\mathbf{q}^i, g_i) \leftarrow (\mathbf{q}^i - \mathbf{r}^i, g)$

function $\mathsf{ExecuteTrade}(\mathbf{r} \in \mathbb{R}^2)$
    $p' \leftarrow \mathsf{price}(\sum_{j=0}^k g_j, \sum_{i=0}^k \mathbf{q}^i + \mathbf{r})$ ▷ The price after this trade
    check $\sum_{i=0}^k \mathbf{q}^i + \mathbf{r} = \mathsf{liability}(\sum_{j=0}^k g_j, p')$
    trader pays $\mathsf{fee}(\mathbf{r}, \mathbf{q}, g)$ cash in fee
     Give r to trader
    for each LP i **do**
        $\mathbf{r}^i \leftarrow \mathsf{liability}(g_i, p') - \mathbf{q}^i$.
        LP i gets $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, \{g_i\}_{i=1}^k)$ fees
        $\mathbf{q}^i \leftarrow \mathbf{q}^i + \mathbf{r}^i$

Proof. Observe that any change in liability vector in the ModifyLiquidity or Initialize phases results in a liability vector that takes the form $\mathsf{liability}(g, p)$ for some $g \in \mathcal{G}_{\text{LP}}$, $p \in [0, 1]$. Let this $g(p) = \alpha g_0(p) = -2\alpha\sqrt{p(1-p)}$ and thereby $g'(p) = -\alpha\frac{1-2p}{\sqrt{p(1-p)}}$. So,

$$\begin{aligned}\mathsf{liability}(g, p) &= (g(p) - p \cdot g'(p)\mathbf{1} + \begin{pmatrix} g'(p) \\ 0 \end{pmatrix} \\ &= \alpha\left(\left(-2\sqrt{p(1-p)} + p\frac{1-2p}{\sqrt{p(1-p)}}\right)\mathbf{1} - \begin{pmatrix} \frac{1-2p}{\sqrt{p(1-p)}} \\ 0 \end{pmatrix}\right) = \alpha\begin{pmatrix} -\sqrt{\frac{1-p}{p}} \\ -\sqrt{\frac{p}{1-p}} \end{pmatrix}.\end{aligned}$$

Hence if $\mathbf{q} = \mathsf{liability}(g, \mathsf{price}(g, p))$,

$$\begin{pmatrix} q_1 \\ q_2 \end{pmatrix} = \mathsf{liability}(g, \mathsf{price}(g, p)) = -\alpha\begin{pmatrix} \sqrt{\frac{1-\mathsf{price}(g,p)}{\mathsf{price}(g,p)}} \\ \sqrt{\frac{\mathsf{price}(g,p)}{1-\mathsf{price}(g,p)}} \end{pmatrix}$$

which implies $q_1 \cdot q_2 = \alpha^2 = x_1 \cdot x_2$.

For the if direction, price of market at q is given by $(g')^{-1}(q_1 - q_2)$, call this p.

$$q_1 - q_2 = g'(p) = -\alpha\frac{1-2p}{\sqrt{p(1-p)}}$$

Solving this, we get that $p = \frac{q_2}{q_1+q_2}$. So,

$$\begin{aligned}\mathsf{liability}(g, \mathsf{price}(g,p)) &= \mathsf{liability}(g, \frac{q_2}{q_1+q_2}) \\ &= \alpha \begin{pmatrix} -\sqrt{\frac{q_1}{q_2}} \\ -\sqrt{\frac{q_2}{q_1}} \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \end{pmatrix}.\end{aligned}$$

The last line uses the fact that $q_1 \cdot q_2 = \alpha^2$. □

Proposition 5. *Protocol 2 is equivalent to Protocol 4 for* $\mathcal{G}_{\text{init}} = \{\alpha g_0 \mid \alpha > 0\}$, $\mathcal{G}_{\text{LP}} = \{\alpha g_0 \mid \alpha \geq 0\}$ *where* $g_0(p) = -2\sqrt{p(1-p)}$, $\mathsf{fee}(\mathbf{r}, \mathbf{q}, g) = \beta \mathbf{r}_+$ *and* $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, \{g_i\}_i) = \frac{\beta\alpha_i}{\alpha}\mathbf{r}_+$ *for* $\beta > 0$.

Proof. Showing that Protocol 4 gives Protocol 2, for the choices of g specified, involves showing the equivalence of three specific components. That is, we wish to show that the initialization check in Line 7 of Protocol 4 is satisfied and that the request vectors in ModifyLiquidity and ExecuteTrade routines match.

First, we note that modifying $\mathbf{x}$ preserves the invariant $x_1 \cdot x_2 = \alpha^2$ for some α in Protocol 2. As shown in the proof of Proposition 4, $(g')^{-1}(q_1 - q_2) = \mathsf{price}(g_0, \mathbf{q}) = \frac{q_2}{q_1+q_2}$ where $\mathbf{q} = -\mathbf{x}$, showing that the Uniswap price function is a special case.

For the initialization phase, $x_1^0 \cdot x_2^0 = (\alpha^0)^2$ and Proposition 4 give us that $\mathsf{liability}(g_0, \mathsf{price}(g_0, q^0)) = \mathbf{q}^0$ hence satisfying the check. By induction, this statement holds for all states $\mathbf{q}$ i.e. $\mathbf{q} = \mathsf{liability}(g, \mathsf{price}(g, \mathbf{q}))$.

A liquidity change of α^i to α' at price p reflects a change of g_i from $-2 \cdot \alpha^i\sqrt{p(1-p)}$ to $\hat{g} = -2 \cdot \alpha'\sqrt{p(1-p)}$ in Uniswap V2. The quantity of assets requested by Protocol 4 is given by

$$\begin{aligned}r^i = -(\mathsf{liability}(\hat{g}, p) - \mathbf{q}^i) &= -(\mathsf{liability}(\hat{g}, p) - \mathsf{liability}(g_i, p)) \\ &= (\alpha' - \alpha^i) \begin{pmatrix} \sqrt{\frac{1-p}{p}} \\ \sqrt{\frac{p}{1-p}} \end{pmatrix} = x'\end{aligned}$$

Now, we show that the check in Line 19 of Protocol 4 for the given g gives us the condition $x_1 \cdot x_2 = (x_1 - r_1) \cdot (x_2 - r_2)$ that appears in Uniswap V2.

Let $\mathbf{q} = \sum_i \mathbf{q}^i$ and $\mathbf{r} = \sum_i \mathbf{r}^i$. The check in Protocol 4 can be rewritten as

$$\begin{aligned}\mathbf{q} + \mathbf{r} &= \mathsf{liability}\left(\sum_{j=0}^{k} g_j, p'\right) \\ &= \mathsf{liability}\left(\sum_{j=0}^{k} g_j, \mathsf{price}\left(\sum_{j=0}^{k} g_j, \mathbf{q} + \mathbf{r}\right)\right)\end{aligned}$$

From this, by applying Proposition 4, $(q_1 + r_1)(q_2 + r_2) = (-x_1 + r_1)(-x_2 + r_2) = \alpha^2 = x_1 \cdot x_2$. Again the note the difference in sign conventions for liability and reserves.

The last fact we want to show to complete the proof is that $r^i = \frac{\alpha^i}{\alpha} r$ for Uniswap V2. This can be obtained by observing that $r = \mathsf{liability}(g, p') - \mathbf{q} = \mathsf{liability}(g, p') - \mathsf{liability}(g, p)$ from Proposition 4 and $g = \frac{\alpha^i}{\alpha} g_i$ being true for Uniswap V2. □

B.2 Liquidity vectors for general bucketing mechanism

We are given, for a specified g, an $\ell^{(j)}$ function of the form below,

$$\ell^{(j)}(p) = g''(p)\mathbf{1}_{[a_j,b_j]}(p) = \begin{cases} 0 & p < a_j \\ g''(p) & p \in [a_j, b_j] \\ 0 & p > b_j \end{cases}.$$

Then, $(\hat{g}^{(j)})'(p)'$ is given by,

$$(\hat{g}^{(j)})'(p) = \int_0^p \ell^{(j)}(s)\, ds = \begin{cases} 0 & p < a_j \\ g'(p) - g'(a_j) & p \in [a_j, b_j] \\ g'(b_j) - g'(a_j) & p > b_j \end{cases}.$$

We integrate from 0 to p to see that

$$\hat{g}^{(j)}(p) = \int_0^p (\hat{g}^{(j)})'(s)ds = \begin{cases} 0 & p < a_j \\ g(p) - g(a_j) - g'(a_j)(p - a_j) & p \in [a_j, b_j] \\ (g'(b_j) - g'(a_j))(p - b_j) + g(b_j) - g(a_j) - g'(a_j)(b_j - a_j) & p > b_j \end{cases}.$$

Then,

$$\begin{aligned} g^{(j)}(p) &= \hat{g}^{(j)}(p) - p\hat{g}^{(j)}(1) \\ &= \begin{cases} p(g(a_j) - g(b_j) - g'(a_j)(a_j - 1) + g'(b_j)(b_j - 1)) & p < a_j \\ g(p) + g(a_j)(p-1) - pg(b_j) - a_j g'(a_j)(p-1) + pg'(b_j)(b_j - 1) & p \in [a_j, b_j] \\ (p-1)(g(a_j) - g(b_j) - a_j g'(a_j) + b_j g'(b_j)) & p > b_j \end{cases}. \end{aligned}$$

From this we can calculate the liability vector as follows

$$\begin{aligned} \mathsf{liability}(g^{(j)}, p) &= (g'(p), 0) + (g(p) - pg'(p))\mathbf{1} \\ &= \begin{cases} \begin{pmatrix} g(a_j) - g(b_j) - g'(a_j)(a_j - 1) + g'(b_j)(b_j - 1) \\ 0 \end{pmatrix} & p < a_j \\ \begin{pmatrix} g(p) - g(b_j) - g'(p)(p-1) + g'(b_j)(b_j - 1) \\ g(p) - g(a_j) - pg'(p) + a_j g'(a_j) \end{pmatrix} & p \in [a_j, b_j] \\ \begin{pmatrix} 0 \\ g(b_j) - g(a_j) + a_j g'(a_j) - b_j g'(b_j) \end{pmatrix} & p > b_j \end{cases} \\ &= \begin{cases} \begin{pmatrix} S(a_j, 1) - S(b_j, 1) \\ 0 \end{pmatrix} & p < a_j \\ \begin{pmatrix} S(p, 1) - S(b_j, 1) \\ S(p, 0) - S(a_j, 0) \end{pmatrix} & p \in [a_j, b_j] \\ \begin{pmatrix} 0 \\ S(b_j, 0) - S(a_j, 0) \end{pmatrix} & p > b_j \end{cases} \\ &= \begin{pmatrix} S_g(\max\{a_j, p\}, 1) - S_g(\max\{b_j, p\}, 1) \\ S_g(\min\{b_j, p\}, 0) - S_g(\min\{a_j, p\}, 0) \end{pmatrix} \end{aligned}$$

where $S(p, y) = g(p) + g'(p)(y - p)$.

B.3 Bucketing scheme for logarithmic market scoring rule (LMSR)

Here, we apply the techniques of the previous section to consider an interesting new protocol. What if we used g from LMSR but with a bucketing scheme similar to Uniswap V3? That is, LPs can deposit according to $g(p) = p\log p + (1-p)\log(1-p)$ on discrete buckets analogous to what we see in Uniswap V3. We see that $g'(p) = \log p - \log(1-p) = \log\left(\frac{p}{1-p}\right)$, so

$$g^{(j)}(p) = \begin{cases} p\log\frac{a_j}{b_j} & p < a_j \\ p\log\frac{p}{b_j} + (1-p)\log\frac{(1-p)}{(1-a_j)} & p \in [a_j, b_j] \\ (1-p)\log\frac{(1-b_j)}{(1-a_j)} & p > b_j \end{cases}.$$

$$\begin{aligned} \mathsf{liability}(g^{(j)}, p) &= (g'(p), 0) + (g(p) - pg'(p))\mathbf{1} \\ &= \begin{cases} \begin{pmatrix} \log\frac{a_j}{b_j} \\ 0 \end{pmatrix} & p < a_j \\ \begin{pmatrix} \log\frac{p}{b_j} \\ \log\frac{(1-p)}{(1-a_j)} \end{pmatrix} & p \in [a_j, b_j] \\ \begin{pmatrix} 0 \\ \log\frac{(1-a_j)}{(1-b_j)} \end{pmatrix} & p > b_j \end{cases}. \end{aligned}$$

B.4 Discussion on Uniswap V3

Readers familiar with the original protocol may recognize Protocol 5 as Uniswap V3 mechanics but with minor changes coming from using normalized prices. Line 14 comes from Fan et al. (2022)'s analysis of Uniswap V3, and Line 19 comes from the shifted reserve curve characteristic of Uniswap V3 as seen in both Fan et al. (2022) and Adams et al. (2021). For clarity, we want to reiterate that Fan et al. (2022) uses an exchange rate price $\hat{p}$, and we use its normalized version p. The two quantities are related by $\hat{p} = \frac{p}{1-p}$.

Proposition 6. *For* $\mathcal{G}_{\text{init}} = \{\sum_j \alpha_j g^{(j)} \mid \alpha_j > 0\}$, $\mathcal{G}_{\text{LP}} = \{\sum_j \alpha_j g^{(j)} \mid \alpha_j \geq 0\}$ *where*

$$g^{(j)}(p) = \begin{cases} p(\sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-a_j}{a_j}}) & \text{if } p \leq a_j \\ -2\sqrt{p(1-p)} + p\sqrt{\frac{1-b_j}{b_j}} + (1-p)\sqrt{\frac{a_j}{1-a_j}} & \text{if } a_j \leq p \leq b_j \\ (1-p)(\sqrt{\frac{a_j}{1-a_j}} - \sqrt{\frac{b_j}{1-b_j}}) & \text{if } p \geq b_j \end{cases},$$

the vector $\mathbf{q}$ *of liability in Protocol 4 always satisfies* $\left(x_1 + \alpha_j\sqrt{\frac{1-b_j}{b_j}}\right) \cdot \left(x_2 + \alpha_j\sqrt{\frac{a_j}{1-a_j}}\right) = \alpha_j^2$ *for some* $\alpha_j > 0$, *where* $\mathbf{x} = -\mathbf{q}$.

Proof. Observe that any change in liability vector in the ModifyLiquidity or Initialize phases results in a liability vector that results in the vector taking the form $\mathsf{liability}(g, p)$ for some $g \in \mathcal{G}_{\text{LP}}$, $p \in [0, 1]$. Let this g be $\alpha_j g^{(j)}(p)$ where α_j is the total liquidity in jth price interval which is $\sum_i \alpha^{ij}$

Protocol 5 Uniswap V3

global constants β, m, $\{B^j = [a_j, b_j]\}_{j=0}^{m}$.
global variables $\mathbf{x} \in \mathbb{R}^2$, $k \in \mathbb{N}$, $\{\alpha^{ij} \in \mathbb{R}_{\geq 0}\}_{i \in \{0,\ldots,k\}, j \in \{0,\cdots,m\}}$
function price($\mathbf{x} \in \mathbb{R}^2$)
    return $\frac{x_2 + \alpha_j \sqrt{\frac{a_j}{1-a_j}}}{x_1 + \alpha_j \sqrt{\frac{1-b_j}{b_j}} + x_2 + \alpha_j \sqrt{\frac{a_j}{1-a_j}}}$
function Initialize($\mathbf{x} \in \mathbb{R}^2, \alpha > 0, \beta$)
    $(k, \beta) \leftarrow (0, \beta)$
    ModifyLiquidity$(0, \mathbf{x}, \alpha, [0,1])$ ▷ We technically have to split this into function call for each price interval.
function RegisterLP()
    $k \leftarrow k + 1$
    $\alpha^{kj} \leftarrow 0, \forall j \in \{0, \ldots, m\}$
    return k ▷ ID of the new LP
function ModifyLiquidity($i \in \mathbb{N}, \alpha' \geq 0, j \in \{0, \ldots, m\}$)
    $p =$ price$(\mathbf{x})$
    request $\mathbf{x}' = \begin{cases} \left((\alpha' - \alpha^{ij})\left(\sqrt{\frac{1-a_j}{a_j}} - \sqrt{\frac{1-b_j}{b_j}}\right), 0\right) & \text{if } p < a_j \\ \left(0, (\alpha' - \alpha^{ij})\left(\sqrt{\frac{b_j}{1-b_j}} - \sqrt{\frac{a_j}{1-a_j}}\right)\right) & \text{if } p > b_j \\ \left((\alpha' - \alpha^{ij})\left(\sqrt{\frac{1-p}{p}} - \sqrt{\frac{1-b_j}{b_j}}\right), (\alpha' - \alpha^{ij})\left(\sqrt{\frac{p}{1-p}} - \sqrt{\frac{a_j}{1-a_j}}\right)\right) & \text{if } p \in [a_j, b_j] \end{cases}$
    $(\mathbf{x}, \alpha^{ij}) \leftarrow (\mathbf{x} + \mathbf{x}', \alpha')$
function ExecuteTrade($\mathbf{r} \in \mathbb{R}^2$)
    Let $p =$ price$(\mathbf{x})$, $p' =$ price$(\mathbf{x} - \mathbf{r})$ [10]
    Let l, u be such that $a_l \leq p \leq b_l$ and $a_u \leq p' \leq b_u$.
    check

$$\frac{1}{(\sum_{i=0}^{k} \alpha^{il})^2}\left(x_1 + \sum_{i=0}^{k} \alpha^{il}\sqrt{\frac{1-b_l}{b_l}}\right)\left(x_2 + \sqrt{\frac{a_l}{1-a_l}}\sum_{i=0}^{k} \alpha^{il}\right) = \frac{1}{(\sum_{i=0}^{k} \alpha^{iu})^2}\left(x_1 - r_1 + \sum_{i=0}^{k} \alpha^{iu}\sqrt{\frac{1-b_u}{b_u}}\right)\left(x_2 - r_2 + \sqrt{\frac{a_u}{1-a_u}}\sum_{i=0}^{k} \alpha^{iu}\right)$$

    pay $\beta \frac{\sum_j \alpha^{ij}}{\sum_j \sum_o \alpha^{oj}}(-\mathbf{r})_+$ to each LP i where j sums over buckets in $[B^l, B^u]$.▷ WLOG assume that the B^u bucket comes later than B^l
    $\mathbf{x} \leftarrow \mathbf{x} - \mathbf{r}$

in Protocol 5. We have

$$g'(p) = \begin{cases} \alpha_j(\sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-a_j}{a_j}}) & \text{if } p \le a_j \\ \alpha_j(-\frac{1-2p}{\sqrt{p(1-p)}} + \sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{a_j}{1-a_j}}) & \text{if } a_j \le p \le b_j \\ (\sqrt{\frac{b_j}{1-b_j}} - \sqrt{\frac{a_j}{1-a_j}}) & \text{if } p \ge b_j \end{cases}$$

Solving for $\mathsf{liability}(g,p) = (g(p) - p \cdot g'(p)\mathbf{1} + \begin{pmatrix} g'(p) \\ 0 \end{pmatrix}$ for these three cases gives us

$$\mathsf{liability}(g,p) = \begin{cases} \alpha_j \begin{pmatrix} \sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-a_j}{a_j}} \\ 0 \end{pmatrix} & \text{if } p \le a_j \\ \alpha_j \begin{pmatrix} -\sqrt{\frac{1-p}{p}} + \sqrt{\frac{1-b_j}{b_j}} \\ -\sqrt{\frac{p}{1-p}} + \sqrt{\frac{a_j}{1-a_j}} \end{pmatrix} & \text{if } a_j \le p \le b_j \\ \alpha_j \begin{pmatrix} 0 \\ \sqrt{\frac{a_j}{1-a_j}} - \sqrt{\frac{b_j}{1-b_j}}) \end{pmatrix} & \text{if } p \ge b_j \end{cases}.$$

In each of these cases, with a bit of algebra we can see that

$$\left(q_1 - \alpha_j\sqrt{\frac{1-b_j}{b_j}}\right) \cdot \left(q_2 - \alpha_j\sqrt{\frac{a_j}{1-a_j}}\right) = \left(x_1 + \alpha_j\sqrt{\frac{1-b_j}{b_j}}\right) \cdot \left(x_2 + \alpha_j\sqrt{\frac{a_j}{1-a_j}}\right) = \alpha_j^2,$$

as $\mathbf{q}$ in Protocol 4 is liability which is negative of reserves in Protocol 5. □

Proposition 7. *Protocol 5 is equivalent to Protocol 4 for* $\mathcal{G}_{\text{init}} = \{\sum_j \alpha_j g^{(j)} \mid \forall j\, \alpha_j > 0\}$, $\mathcal{G}_{\text{LP}} = \{\sum_j \alpha_j g^{(j)} \mid \forall j\, \alpha_j \ge 0\}$ *where*

$$g^{(j)}(p) = \begin{cases} p(\sqrt{\frac{1-b_j}{b_j}} - \sqrt{\frac{1-a_j}{a_j}}) & \text{if } p \le a_j \\ -2\sqrt{p(1-p)} + p\sqrt{\frac{1-b_j}{b_j}} + (1-p)\sqrt{\frac{a_j}{1-a_j}} & \text{if } a_j \le p \le b_j \\ (1-p)(\sqrt{\frac{a_j}{1-a_j}} - \sqrt{\frac{b_j}{1-b_j}}) & \text{if } p \ge b_j \end{cases}.$$

Proof. To show that Protocol 4 gives us Protocol 5, for the choices of g specified, involves showing three specific components are equivalent. They include showing that the initialization check satisfies and the request vectors in ModifyLiquidity and ExecuteTrade routines match.

We saw α_j to mean total liquidity in jth interval i.e. $\sum_{i=0}^{k} \alpha^{ij}$ and let $g(p) = \alpha_j g^{(j)}$. Firstly, to show the initialization check, we derive that

$$(g')^{-1}(q_1 - q_2) = (g')^{-1}(x_2 - x_1) = \frac{x_2 + \alpha_j\sqrt{\frac{a_j}{1-a_j}}}{x_1 + \alpha_j\sqrt{\frac{1-b_j}{b_j}} + x_2 + \alpha_j\sqrt{\frac{a_j}{1-a_j}}}.$$

[10] We also note that price in Uniswap V3 is not calculated on demand like we do here but is a state variable thats maintained throughout the implementation. Uniswap V3 also implements the Line 19 by passing through all price ranges consecutive to p and checking which price interval satisfies this check. We abstract away from this to avoid being caught up in technicalities as this is not the main problem we tackle.

We consider only the case when this price falls in a bucket j as for all other buckets, $(g')^{-1}$ would not give a definitive result.

$$\begin{aligned}
\mathsf{liability}(g_0, \mathsf{price}(g_0, q^0)) &= \alpha_j \begin{pmatrix} -\sqrt{\frac{x_1^0+\alpha_j\sqrt{\frac{1-b_j}{b_j}}}{x_2^0+\alpha_j\sqrt{\frac{a_j}{1-a_j}}}} + \sqrt{\frac{1-b_j}{b_j}} \\ -\sqrt{\frac{x_2^0+\alpha_j\sqrt{\frac{a_j}{1-a_j}}}{x_1^0+\alpha_j\sqrt{\frac{1-b_j}{b_j}}}} + \sqrt{\frac{a_j}{1-a_j}} \end{pmatrix} \\
&= \alpha_j \begin{pmatrix} -\frac{x_1^0+\alpha_j\sqrt{\frac{1-b_j}{b_j}}}{\alpha_j} + \sqrt{\frac{1-b_j}{b_j}} \\ -\frac{x_2^0+\alpha_j\sqrt{\frac{a_j}{1-a_j}}}{\alpha_j} + \sqrt{\frac{a_j}{1-a_j}} \end{pmatrix} \\
&= \begin{pmatrix} -x_1^0 \\ -x_2^0 \end{pmatrix} = \begin{pmatrix} q_1^0 \\ q_2^0 \end{pmatrix}
\end{aligned}$$

The first two steps are a result of Proposition 6.

Now, we show that the quantity of assets requested from a LP to change the liquidity level from α^{ij} to α' given by Line 14 in Protocol 5 is equivalent to Line 11 of Protocol 4.

We first note that the price p, does not change in this operation, as liquidity must be added by keeping the price constant. Another way to see it is that $\mathsf{price}(g_0, \mathbf{q}^0)$ remains unchanged. A liquidity change of α^{ij} to α' at price $p \in B_j$ reflects a change of g_i from $\alpha^{ij} g^{(j)}$ to $\hat{g} = \alpha' g^{(j)}$ in Uniswap V3. The quantity of asset requested by Protocol 4 is given by

$$\begin{aligned}
-(\mathsf{liability}(\hat{g}, p) - \mathbf{q}^i) &= \mathsf{liability}(g_i, p) - \mathsf{liability}(\hat{g}, p) \\
&= \begin{cases} \alpha^{ij}\left(-\sqrt{\frac{1-a_j}{a_j}} + \sqrt{\frac{1-b_j}{b_j}}, 0\right) - \alpha'\left(-\sqrt{\frac{1-a_j}{a_j}} + \sqrt{\frac{1-b_j}{b_j}}, 0\right) & \text{if } p < a_j \\ \alpha^{ij}\left(0, \sqrt{\frac{a_j}{1-a_j}} + \sqrt{\frac{b_j}{1-b_j}}\right) - \alpha'\left(0, \sqrt{\frac{a_j}{1-a_j}} + \sqrt{\frac{b_j}{1-b_j}}\right) & \text{if } p > b_j \\ (\alpha^{ij} - \alpha')\left(-\sqrt{\frac{1-p}{p}} + \sqrt{\frac{1-b_j}{b_j}}, \sqrt{\frac{p}{1-p}} + \sqrt{\frac{a_j}{1-a_j}}\right) & \text{if } p \in [a_j, b_j] \end{cases} \\
&= \begin{cases} \left((\alpha' - \alpha^{ij})\left(\sqrt{\frac{1-a_j}{a_j}} - \sqrt{\frac{1-b_j}{b_j}}\right), 0\right) & \text{if } p < a_j \\ \left(0, (\alpha' - \alpha^{ij})\left(\sqrt{\frac{b_j}{1-b_j}} - \sqrt{\frac{a_j}{1-a_j}}\right)\right) & \text{if } p > b_j \\ \left((\alpha' - \alpha^{ij})\left(\sqrt{\frac{1-p}{p}} - \sqrt{\frac{1-b_j}{b_j}}\right), (\alpha' - \alpha^{ij})\left(\sqrt{\frac{p}{1-p}} - \sqrt{\frac{a_j}{1-a_j}}\right)\right) & \text{if } p \in [a_j, b_j] \end{cases}
\end{aligned}.$$

Now, we show that the check in Line 15 of Protocol 4 for the given g gives us the condition $\left(x_1 + \alpha_l\sqrt{\frac{1-b_l}{b_l}}\right)\left(x_2 + \sqrt{\frac{a_l}{1-a_l}}\alpha_l\right) = \left(x_1 - r_1 + \alpha_u\sqrt{\frac{1-b_u}{b_u}}\right)\left(x_2 - r_2 + \sqrt{\frac{a_u}{1-a_u}}\alpha_u\right)$ where $\alpha_x = \sum_{i=0}^{k} \alpha^{ix}$ that appears in Uniswap V3.

Let $\mathbf{q} = \sum_i \mathbf{q}^i$ and $\mathbf{r} = \sum_i \mathbf{r}^i$. The check in Protocol 4 can be rewritten as

$$\begin{aligned}\mathbf{q} + \mathbf{r} &= \mathsf{liability}\left(\sum_{i=0}^{k} g_i, p'\right) \\ &= \mathsf{liability}\left(\sum_{i=0}^{k} \alpha^{iu} g^{(j)}, p'\right) \\ &= \sum_{i=0}^{k} \alpha^{iu} \mathsf{liability}(g^{(j)}, p').\end{aligned}$$

From Proposition 6, we can see that

$$\left(x_1 - r_1 + \left(\sum_{i=0}^{k} \alpha^{iu}\right)\sqrt{\frac{1-b_u}{b_u}}\right) \cdot \left(x_2 - r_2 + \left(\sum_{i=0}^{k} \alpha^{iu}\right)\sqrt{\frac{a_u}{1-a_u}}\right) = \left(\sum_{i=0}^{k} \alpha^{iu}\right)^2$$

and

$$\left(x_1 + \sum_{i=0}^{k} \alpha^{il}\sqrt{\frac{1-b_l}{b_l}}\right) \cdot \left(x_2 + \sum_{i=0}^{k} \alpha^{il}\sqrt{\frac{a_l}{1-a_l}}\right) = \left(\sum_{i=0}^{k} \alpha^{il}\right)^2,$$

proving what we need.

The last fact we want to show to complete the proof is that $r^i = \frac{\alpha^i}{\alpha} r$ for Uniswap V3, where $\alpha_i = \sum_j \alpha^{ij}$, $\alpha = \sum_j \sum_{o=0}^{k} \alpha^{oj}$ for j summing over baskets B^l to B^u. We see that

$$\begin{aligned}\mathbf{r}^i &= \mathsf{liability}(g_i, p') - \mathbf{q}^i \\ &= \alpha^i(\mathsf{liability}(g^{(j)}, p') - \mathsf{liability}(g^{(j)}, p)) \\ &= \frac{\alpha^i}{\alpha}\left(\mathsf{liability}(g, p') - \mathbf{q}\right) \quad \text{where } g = \sum_{i=0}^{k} g_i \text{ and } \mathbf{q} = \sum_{i=0}^{k} \mathbf{q}_i \\ &= \frac{\alpha^i}{\alpha}\mathbf{r},\end{aligned}$$

as desired. □

C Detailed discussion on new protocols from section 6

C.1 Soft buckets allowing richer functions than discrete buckets

Define a set $a_0 < a_1 = 0 < a_2 < \cdots < a_k = 1 < a_{k+1}$. Our "buckets" B_j will be supported on the interval $[a_{j-1}, a_{j+1}]$. To capture the continuous fading, define the triangular function $T^{(j)} : [0,1] \to [0,1]$ as $T^{(j)}(p) = \left(\frac{p-a_{j-1}}{a_j - a_{j-1}}\right)\mathbf{1}_{p\in[a_{j-1},a_j]} + \left(\frac{a_{j+1}-p}{a_{j+1}-a_j}\right)\mathbf{1}_{p\in[a_j,a_{j+1}]}$. The corresponding base liquidity functions are then $\ell^{(j)} = (\ell T^{(j)})(p)$ where $\ell(p) = 2(p(1-p))^{-\frac{3}{2}}$, to again use the same base "shape" as the constant product invariant $\varphi_\alpha(\mathbf{x}) = x_1 x_2 = \alpha^2$. Let $g^{(j)}$ then be the corresponding base generating function for $\ell^{(j)}$.

Now letting $\mathcal{G}_{\text{init}} = \{\sum_j \alpha_j g^{(j)} \mid \forall j\, \alpha_j > 0\}$ and $\mathcal{G}_{\text{LP}} = \{\sum_j \alpha_j g^{(j)} \mid \forall j\, \alpha_j \geq 0\}$, Protocol 4 gives a new liquidity provisioning protocol. While the corresponding computations appear to

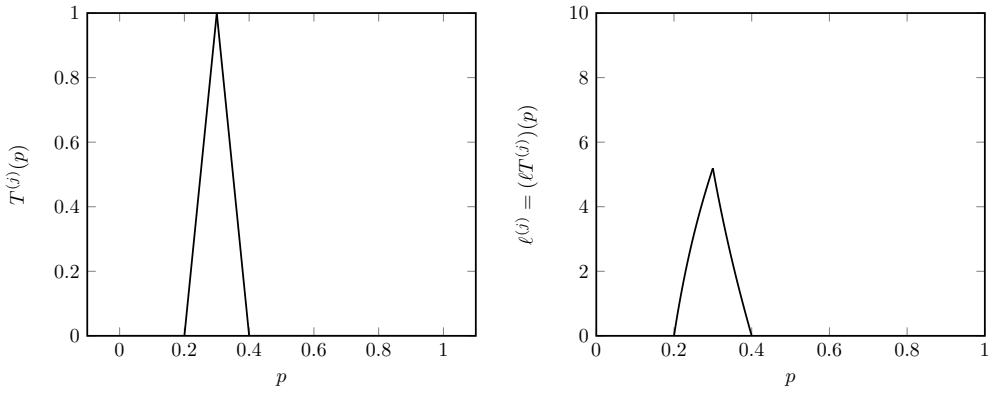

Figure 2: $T^{(j)}(p)$ and $\ell^{(j)}(p)$ for $a_j = 0.3$

be essentially as light-weight as Uniswap V3, the liquidity function is better behaved. We can characterize the possible liquidity functions $\{g'' \mid g \in \mathcal{G}_{\text{LP}}\}$ as the functions $f\ell$ where f is an arbitrary nonnegative continuous function that is affine on each interval $[a_j, a_{j+1}]$. (Simply take $\alpha_j = f(a_j)$.) Thus, we have in particular that liquidity is always continuous in the price in this new protocol. One can additionally innovate from here, adding more flexibility, while taking care to keep the various computations manageable.

C.2 Piecewise linear market maker

To demonstrate the robustness of our protocol, in this section we consider a market where the liquidity curves ℓ are not well-defined. Yet, we show that our general scheme still gives rise to a sensible market maker.

Consider a market maker that restricts the possible prices to $\{p_i\}_{i=1}^m$ where $0 < a_1 < \cdots < a_m < 1$. Consider a $(g^{(j)})'$ of the form -

$$(g^{(j)})'(p) = \begin{cases} a_j - 1 & p \le a_j \\ [a_j - 1, a_j] & p = a_j \\ a_j & p \ge a_j \end{cases}.$$

$g'(p) = \sum_j \alpha_j (g^{(j)})'(p)$ where $\alpha_j = \sum_i \alpha^{ij}$.

Observe that liquidity is infinity at each price. As prices only move discretely in this market, the market needs to be parameterized by reserves held / liability vector $\mathbf{q}$. Hence the state of the market is $\{\alpha_{ij}\}_{i\in\{0,\cdots,k\},j\in\{1,\cdots,m\}}, \{q_i\}_{i\in\{0,\cdots,k\}}$.

If the current market reserves are q, the price in this market can be derived from the below formula

$$\mathsf{price}(g, q) = a_{j^*} \text{ where } j^* = \text{argmax}_{j'} \text{ s.t. } \left\{ q - \sum_{j=1}^{m} \alpha_j (a_j - 1) + \sum_{j=1}^{j'-1} \alpha_j \ge 0 \right\}$$

For a given q, let there exist $y \in [0,1)$ such that $q = \sum_j \alpha_j a_j - \sum_{j=j^*}^m \alpha_j + y\alpha_{j^*}$. We can maintain this relative liquidity in a bucket after changing the liquidity curves.

For the ModifyLiquidity function, let LP i wants to change its liquidity levels from α_{ij} to α'_{ij} for price bucket j. The new reserves that LP i needs to deposit is given by $q' - q = (\alpha'_{ij} - \alpha_{ij})(a_j - \mathbf{1}_{j \geq j^*} + y\mathbf{1}_{j=j^*})$

ExecuteTrade takes in a trade r, computes the new price p' corresponding to the new liability vector $q + r$ using the above given formula.

D Fees and budget balancedness

We show an example when the fees used in DeFi are not budget-balanced: the market maker may not charge enough in fees to cover those owed to the LPs. To illustrate why, consider a 3 outcome market with two LPs $G = G_1 + G_2$ where $G_1(\mathbf{p}) = -2\sqrt{p_1 p_2}$ and $G_2(\mathbf{p}) = -2\sqrt{p_2 p_3}$. Per Protocol 3, a trade $\mathbf{p} \to \mathbf{p}'$ is given by $\mathbf{r} = S_G(\mathbf{p}', \cdot) - S_G(\mathbf{p}, \cdot)$, where

$$S_G(\mathbf{p}, \cdot) = \nabla \overline{G}(\mathbf{p}) = \nabla \overline{G}_1(\mathbf{p}) + \nabla \overline{G}_2(\mathbf{p}) = \left(\sqrt{p_1/p_2}, \sqrt{p_2/p_1} + \sqrt{p_2/p_3}, \sqrt{p_3/p_2}\right) .$$

The trade is then split among the two LPs, as

$$\mathbf{r}^1 = \nabla G_1(\mathbf{p}') - \nabla G_1(\mathbf{p}) = \left(\sqrt{p'_1/p'_2} - \sqrt{p_1/p_2}, \sqrt{p'_2/p'_1} - \sqrt{p_2/p_1}, 0\right) ,$$

$$\mathbf{r}^2 = \nabla G_2(\mathbf{p}') - \nabla G_2(\mathbf{p}) = \left(0, \sqrt{p'_2/p'_3} - \sqrt{p_2/p_3}, \sqrt{p'_3/p'_2} - \sqrt{p_3/p_2}\right) .$$

Let us start the market at the uniform price $\mathbf{p} = (1/3, 1/3, 1/3)$, and consider a trader wishing to purchase security 1 in exchange for security 3, as above. Intuitively, as there is liquidity between securities 1 and 2 (provided by LP 1) and between securities 2 and 3 (provided by LP 2), there should be "combined" liquidity between 1 and 3. And indeed that is the case: if the trader selects $\mathbf{p}' = \left(\frac{3/2+\sqrt{2}}{3+\sqrt{2}}, \frac{1}{3+\sqrt{2}}, \frac{1}{2(3+\sqrt{2})}\right)$, an expression chosen for arithmetic convenience, we have a resulting trade $\mathbf{r} = \nabla \overline{G}(\mathbf{p}') - \nabla \overline{G}(\mathbf{p}) = (\sqrt{3/2} - 1, 0, \sqrt{1/2} - 1)$. The split $\mathbf{r} = \mathbf{r}^1 + \mathbf{r}^2$ between the LPs is also roughly as one would expect, each $\mathbf{r}^i$ being between the corresponding pair of securities:

$$\mathbf{r}^1 = \nabla G_1(\mathbf{p}') - \nabla G_1(\mathbf{p}) = \left(0, \sqrt{2} - 1, \sqrt{1/2} - 1\right) ,$$

$$\mathbf{r}^2 = \nabla G_2(\mathbf{p}') - \nabla G_2(\mathbf{p}) = \left(\sqrt{3/2} - 1, 1 - \sqrt{2}, 0\right) .$$

Using the standard trading fee used in DeFi, i.e., $\mathsf{fee}(\mathbf{r}, \mathbf{q}, C) = \beta \mathbf{r}_+$, $\mathsf{fee}_i(\mathbf{r}, \mathbf{q}, C_{i=1}^k) = \beta(\mathbf{r}_i)_+$, presents an issue. The fee charged to the trader, $\beta(-\mathbf{r})_+ = \beta(0, 0, 1 - \sqrt{1/2})$, ignores the fact that LP 2 provided liquidity that facilitated the trade. Indeed, looking at the fees paid to LPs, we see this same fee $\beta(-\mathbf{r}^1)_+ = \beta(0, 0, 1 - \sqrt{1/2})$ paid to LP 1, plus an additional fee of $\beta(-\mathbf{r}^2)_+ = \beta(0, \sqrt{2} - 1, 0)$ to LP 2.

Fortunately, this issue is not present in 2-asset protocols like Protocol 4, but clearly it can emerge beyond 2 assets/outcomes. Moreover, it seems to emerge precisely when there is "synergy" among the LPs, enabling trades that fruitfully combine their liquidity. While one could easily fix this issue by directly charging the trader for the sum of the fees to the LPs, doing so may be problematic. For example, this proposal would break the abstraction barrier, in the sense that the fees would depend intimately on the LP profile, not just their combined liquidity. Resolving this issue adequately is an interesting direction for future work.