Sequential Dutch Auction Markets Specification and Implementation

Oighty, Zeus, PottedMeat, indigo
Bond Protocol
OlympusDAO

September 2022

Abstract

We formalize the sequential dutch auction market model for the exchange of ERC20 tokens (aka, Olympus-style bond markets) and analyze the integer rounding errors in the Bond Protocol implementation.

1 Introduction

We provide a formal treatment of a novel decentralized finance mechanism, which we call a Sequential Dutch Auction Market (SDAM). We formalize the pricing model of the mechanism and analyze its implementation in the Bond Protocol system to verify that the approximations introduced by integer arithmetic optimize for desired rounding behaviors and allowed value ranges.

2 Sequential Dutch Auction Markets

A dutch auction, also known as an open-outcry descending-price auction, is an auction in which the asking price of an asset gradually lowers through the market duration until it is purchased. A market maker can utilize a dutch auction to sell over-the-counter (OTC) assets to bidders, creating markets in conditions of minimal liquidity with no slippage.

Consider an OTC market offering ("market") created by a user (the "maker") to buy one type of token (quote token) with another type of token (payout token). The market has a defined capacity and is offered from the time it is created to the configured conclusion timestamp. The goal of the market is to expend the capacity evenly over the duration of the market, but not necessarily at a constant rate, using a sequence of dutch auctions for variable amounts of payout tokens. The first auction begins when the market is created with the initial price configured by the maker. As time progresses, the price decreases at a linear rate until a purchase is made by a user (the "taker"). The taker is able to purchase an amount of payout tokens up to a maximum amount calculated by the market with no slippage. The purchase triggers a state transition, which updates the price and other variables. Then, a new dutch auction begins with the new state immediately after the transition. This sequence continues until either the capacity is expended or the market expires. We call this a Sequential Dutch Auction Market (SDAM).

3 Model Definition

When the taker exchanges quote tokens, they receive payout tokens at the current price of the SDAM, which is expressed as a ratio of quote tokens per payout token. Price is derived from two

variables: the debt of the SDAM, which increases by the payout amount of each purchase and decays over time to implement the sequential dutch auction pricing model, and the control variable, which is an independent variable adjusted based on market activity to achieve the goal of expending capacity evenly over the market duration. We formally define the price as:

Definition 1. $\Phi(t) \in \mathbb{R}^+$ is the price at time $t \in [T_0, T]$. $\Phi(t)$ is determined by the minimum price set by the maker, $\Phi_{min} \in \mathbb{R}^+$, the debt $D(t) \in \mathbb{R}^+$, and the control variable $\Gamma(t) \in \mathbb{R}^+$, such that:

$$\Phi(t) = max(\Phi_{min}, D(t) \times \Gamma(t))$$

3.1 Continuous Temporal Values and Discrete State Transitions

SDAMs have a defined conclusion. Therefore, they operate over a finite amount of time. Additionally, many logical components rely on the passage of time to ensure the SDAM's goals are achieved. We begin by defining the start, conclusion, and duration of a SDAM as follows:

Definition 2. Let $T_0, T \in \mathbb{R}^+$, $T_0 < T$, and L be the duration of the SDAM. Then, $L = T - T_0$.

SDAMs use stored values at a specific timestamp to calculate price and other variables at the time of a purchase. A state transition occurs whenever a purchase is made. Under certain conditions, the SDAM will optimize price with respect to the SDAM's capacity target by tuning the non-decaying price component, the control variable Γ. If a tune also occurs, a second state transition will follow immediately after.

Therefore, the system can be said to have up to three different states at a specific timestamp. To denote the differences between these states and aid in understanding the order of computation, asterisks are used to denote the values calculated at a timestamp after a purchase is made, e.g. $D(t)$ is the amount of debt prior to a purchase at time t whereas $D^*(t)$ is the amount of debt after a purchase is made at time t, and $\Gamma(t)$ is the value of the control variable prior to a tune at time t whereas $\Gamma^{**}(t)$ is the value of the control variable after a tune at time t. If no purchase is made at time t, the values will be the same.

We formally define the following set notation to aid our analysis of our continuous functions in the context of the discrete state transitions:

Definition 3. Let $B \subset \mathbb{R}^+$ be the set of all timestamps when purchases are made on the SDAM and $G \subset \mathbb{R}^+$ be the set of all timestamps when the SDAM is tuned such that $\forall s \in B$ and $\forall s \in G, T_0 \leq s \leq T$. SDAMs can only be tuned on a purchase; therefore, $G \subseteq B$. For $t \in [T_0, T]$, we define B_t as the set of all timestamps when purchases are made on the SDAM prior to t, and B_t^* as the set of all timestamps when purchases are made on the SDAM up to and including t such that:

$$B_t = [T_0, t) \bigcap B$$

$$B_t^* = [T_0, t] \bigcap B$$

Similarly, we define G_t as the set of all timestamps when the SDAM is tuned prior to t, and G_t^* as the set of all timestamps the SDAM is tuned up to and including t such that:

$$G_t = [T_0, t) \bigcap G$$

$$G_t^{**} = [T_0, t] \bigcap G$$

Additionally, we have:

- $B_t \neq B_t^* \iff t \in B$,

- $G_t \neq G_t^{**} \iff t \in G$
- $G_t \subseteq B_t$,
- and $G_t^{**} \subseteq B_t^*$.

3.2 Capacity and Payouts

The goal of the SDAM is to uniformly expend the defined token capacity over its duration. In order to do this, the SDAM needs to keep track of the capacity at any given time because it does not rely on oracles and instead uses the activity on the market to adjust its price. Thus, we provide notation and formulas for tracking this information to be used in state transitions.

SDAMs are created with a fixed capacity of tokens to buy or sell. Capacity can be specified in the quote (buy) or payout token (sell). In this paper, we examine the case where capacity is in the payout token. However, if specified in quote token, the formulas work the same after the amounts are converted into payout token units using the current price. In our case, capacity is the limit of payouts that can be made. The SDAM will end when token capacity is expended, the max debt circuit breaker is tripped, or the conclusion timestamp is reached, whichever is first. Therefore, the total of all payouts will be less than or equal to the initial capacity.

3.2.1 Receipts, Fees, and Payouts

We must account for the individual purchase amounts for each transaction at the discrete points in time they are made in order to properly track the amount of capacity distributed over the SDAM duration. Additionally, the purchase amounts and timing directly affect the price of the SDAM. Let's formally define the amount of tokens received and paid out at a specific time as follows:

Definition 4. For $t \in [T_0, T]$, let $q(t) \in \mathbb{R}^+$ be the amount of quote tokens bought and $p(t) \in \mathbb{R}^+$ be the amount of payout tokens sold at time t. If $t \in B$, $q(t) > 0$ and $p(t) > 0$, otherwise $q(t) = 0$ and $p(t) = 0$.

The SDAM may take a fee on the tokens received from the taker before they are sent to the maker. Let's define the amount of tokens received and taken as a fee as follows:

Definition 5. Let $\epsilon \in [0, 1)$ be a percentage fee taken on purchases in the quote token. Then, for $t \in [T_0, T]$, we define the amount of quote tokens received by the maker as $r(t) \in \mathbb{R}^+$ and the amount of quote tokens taken as fee as $f(t) \in \mathbb{R}^+$ such that:

$$r(t) = q(t) \times (1 - \epsilon)$$

$$f(t) = q(t) \times \epsilon$$

3.2.2 Capacity

Now that we have defined the amounts received and paid out by a SDAM, we can formally define capacity:

Definition 6. Assume capacity is defined by the maker in payout tokens. Let $t \in [T_0, T]$, $C(t) \in \mathbb{R}^+$ be the capacity prior to any purchases at time t, and $C^*(t) \in \mathbb{R}^+$ be the capacity following any purchases at time t. Then,

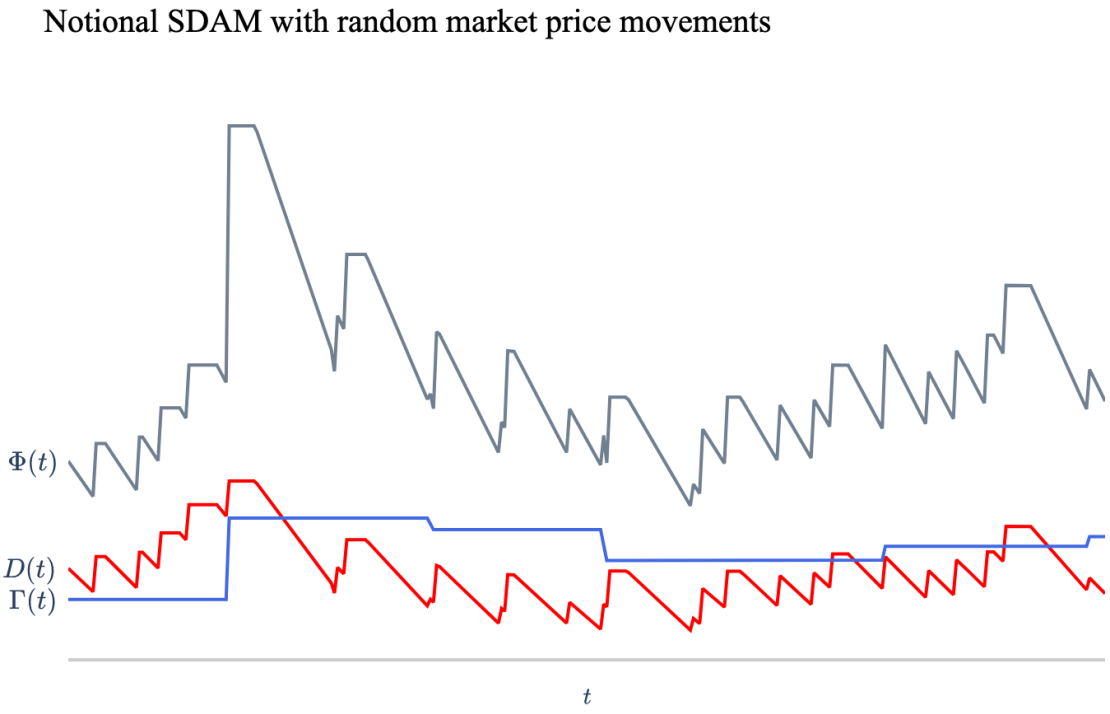
$$C(t) = C_0 - \sum_{s \in B_t} p(s)$$

$$C^*(t) = C(t) - p(t)$$

where C_0 is the initial capacity, $p(t)$ is the amount of payout tokens paid out at time t, and B_t is the set of timestamps where purchases are made up to time t.

3.3 Price Components

We previously defined the price of an SDAM as the product of the debt variable and control variables. This section further develops those components. The below chart shows the price and component variables over the duration of a notional SDAM with regular purchases to provide an intuition for the two variables as we expand the formal definitions.



3.3.1 Debt and Price Decay

We model a sequential dutch auction pricing model by establishing a debt variable, $D(t)$, derived from the capacity in the payout token. Debt decays linearly to zero over the debt decay interval, I_D. On a purchase, debt increases by the amount of the payout $p(t)$. We scale the capacity by the ratio of the debt decay interval to the total duration, L, so that the payout increases the debt proportionally to the max payout, $p_{max}(t)$. Debt can be a range of values after a purchase since payouts are variable, but we want the price, and therefore the debt variable, to decay at the same speed over the course of the SDAM. Thus, we introduce the last decay time, $T_D(t)$, as a reference point to calculate decay from at any time $t \in [T_0, T]$ and update it on each purchase. $T_D(t)$ increases on a purchase by a percentage of the debt decay interval derived from the size of $p(t)$ relative to the target debt, $\delta(t)$, as calculated at the last control variable tune (discussed in the following section). We formally define $T_D(t)$ and $D(t)$ as:

Definition 7. $T_D(t) \in \mathbb{R}^+$ is the debt decay reference time at time $t \in [T_0, T]$, prior to any purchases at time t. The SDAM is initialized with the debt decay reference set to the start time, $T_D(T_0) = T_0$. $T_D^*(t)$ is the debt decay reference time at time $t \in [T_0, T]$, following any purchases at time t. Assume

that a purchase is made at time t, i.e. $t \in B_t^*$. Then,

$$T_D^*(t) = T_D(t) + I_D \frac{p(t)}{\delta(t)}$$

where I_D is the debt decay interval, $p(t)$ is the payout for purchases at time t, and $\delta(t)$ is the target debt at the last control variable tune.

From this definition, we can induce a generalized result for the debt decay reference time, $T_D(t)$.

Lemma 1. *If B_t is the set of timestamps that purchases are made prior to time $t \in [T_0, T]$, then*

$$T_D(t) = T_0 + I_D \sum_{s \in B_t} \frac{p(s)}{\delta(s)}$$

We now formally define the debt variable and model its decay over the duration of the SDAM.

Definition 8. Let $t \in [T_0, T]$. $D(t) \in \mathbb{R}^+$ is the debt prior to any purchases at time t. Debt is initialized to the initial capacity scaled by the ratio of the debt decay interval and the duration. $D^*(t) \in \mathbb{R}^+$ is the debt following any purchases at time t. Debt increases by the amount of the payout when a purchase is made at time t. Then,

$$D(T_0) = C_0 \frac{I_D}{L}$$

$$D^*(t) = D(t) + p(t)$$

Additionally, let $\Delta t \in (0, T - t]$ be a small time increment such that there aren't any purchases from t to $t + \Delta t$, i.e. $B_t^* = B_{t+\Delta t}$, and where $\Delta t < I_D$. Then,

$$D(t + \Delta t) = D^*(t) \left(1 - \frac{max(0, t + \Delta t - T_D(t + \Delta t))}{I_D}\right)$$

where I_D is the debt decay interval and $T_D(t)$ is the debt decay reference time at time t.

We can then induce a generalized result for the debt variable, $D(t)$:

Lemma 2. *Let $t \in [T_0, T]$. Recall that B_t is the set of timestamps when purchases are made from $[T_0, t)$. Then,*

$$D(t) = C_0 \frac{I_D}{L} \prod_{s \in B_t} \left(1 - \frac{max(0, s - T_D(s))}{I_D}\right) + \sum_{s \in B_t} \left(p(s) \prod_{u \in B_t \bigcap [s,t)} \left(1 - \frac{max(0, u - T_D(u))}{I_D}\right)\right)$$

where I_D is the debt decay interval and $T_D(t)$ is the last debt decay reference time as defined above.

3.3.2 Max Debt Circuit Breaker

In order to limit makers' exposure to extreme market conditions, SDAMs are configured with a debt buffer from which a maximum debt value is calculated on creation. In the event that a quote token suddenly loses a large amount of value, e.g. a stable coin losing its peg or a smart contract hack is revealed, the maximum debt value serves as a circuit breaker to prevent takers from exchanging too many quote tokens for payout tokens at once. If the maximum debt is exceeded on a purchase, the SDAM ends immediately. We define the maximum debt as:

Definition 9. Let $b_D \in \mathbb{R}^+$ be the debt buffer provided by the maker and $D_{max} \in \mathbb{R}^+$ be the maximum debt. Then,

$$D_{max} = D(T_0)(1 + b_D)$$

where $D(T_0)$ is the initial debt.

3.3.3 Control Variable and Tuning

The second variable that the SDAM price depends on is the control variable, Γ. Debt is used to implement the continuous decay of a dutch auction pricing format. The control variable is an independent variable that converts the debt variable to a price.

The SDAM price is adjusted over time to be competitive to an external market price to optimize the sales target capacity. The SDAM offers takers a deal that they will accept instead of exchanging quote tokens for payout tokens via another liquidity source. Therefore, the control variable is adjusted, which we call "tuning", based on purchase activity to maintain a competitive price and happens in two cases:

1. The SDAM is undersold relative to the expected capacity at the time of the purchase and a specified amount of time since the last tuning, the tune interval I_Γ, has passed.

2. The SDAM is oversold relative to the expected capacity at the time of the purchase and a specified amount of capacity has been sold since the last tuning, the tune capacity C_Γ.

In order to track the time at which a SDAM can be tuned in the first case, we define the last tune time as:

Definition 10. Let $t \in [T_0, T]$. $T_\Gamma(t) \in \mathbb{R}^+$ is the last tune time at time t prior to the SDAM being tuned at t. Since the control variable is initialized on SDAM creation, the last tune time is initialized to the start of the SDAM, $T_\Gamma(T_0) = T_0$. $T_\Gamma^{**}(t)$ is the last tune time at time t following the SDAM being tuned at time t. Assume the SDAM is tuned at time t, i.e. $t \in G_t^*$. Then,

$$T_\Gamma^{**}(t) = t$$

Additionally, let $\Delta t \in (0, T - t]$ be a small time increment such that the SDAM is not tuned again from t to $t + \Delta t$, i.e. $G_t^* = G_{t+\Delta t}$. Then,

$$T_\Gamma(t + \Delta t) = T_\Gamma^{**}(t)$$

Recall G_t is the set of timestamps that the SDAM is tuned at from T_0 to t, including T_0. Then, we can alternatively express this as:

$$T_\Gamma(t) = max(G_t)$$
$$T_\Gamma^{**}(t) = max(G_t^*)$$

Time-neutral capacity is the expected capacity to be sold up to a point and measures whether the SDAM is undersold or oversold.If the time-neutral capacity is greater than the initial capacity, then the SDAM is undersold since it has more capacity remaining than expected. If the time-neutral capacity is less than the initial capacity, then the SDAM is oversold since it has less capacity remaining than expected. We formally define time-neutral capacity as:

Definition 11. Let $t \in [T_0, T]$. $\chi(t) \in \mathbb{R}^+$ is the time-neutral capacity at time t, prior to any purchases at time t. The SDAM is initialized with time-neutral capacity equal to the initial capacity, $\chi(T_0) = C_0$. $\chi^*(t)$ is the time-neutral capacity at time t, following any purchases at time t. Then,

$$\chi(t) = C_0 \frac{t - T_0}{L} + C(t)$$

$$\chi^*(t) = C_0 \frac{t - T_0}{L} + C^*(t)$$

where T_0 is the start time, L is the duration, $C(t)$ is the capacity prior to any purchases at time t, and $C^*(t)$ is the capacity following any purchases at time t. Additionally,

- A SDAM is **undersold** at time $t \iff \chi^*(t) > C_0$.
- A SDAM is **oversold** at time $t \iff \chi^*(t) < C_0$.

Time-neutral capacity prevent takers from manipulating the tuning process, such as trying to force the SDAM to tune when it does not make economic sense to make a purchase. Time-neutral capacity is also used to calculate the target amount of debt for the SDAM to sell its remaining capacity by the SDAM conclusion time. This target debt can then be converted to a target control variable, since that is the variable we can adjust, by dividing the price at the time of the purchase with the target debt. We formally define target debt as:

Definition 12. Let $t \in [T_0, T]$. $\delta(t) \in \mathbb{R}^+$ is the target debt at time t, prior to any purchases made at time t. The SDAM is initialized with target debt equal to the initial debt variable, $\delta(T_0) = D(T_0)$. $\delta^*(t)$ is the target debt at time t, following any purchases made at time t. Then,

$$\delta(t) = \chi(t)\frac{I_D}{L}$$

$$\delta^*(t) = \chi^*(t)\frac{I_D}{L}$$

where I_D is the debt decay interval, L is the duration, and $\chi(t)$ and $\chi^*(t)$ are the time-neutral capacity values defined above.

Tuning the control variable directly changes the price of the SDAM. While this is the intent to follow market movements, large, abrupt, downward changes to the control variable could cause the SDAM to offer too low of a price to the market and result in poor execution for the maker. However, large upward changes to the control variable may protect from overselling if the current SDAM price is too low. Therefore, control variable adjustments are applied asymmetrically. If the target control variable calculated when tuning the SDAM is:

- Greater than the current control variable, then the update is applied immediately.
- Less than the current control variable, then the update is applied gradually over the tune adjustment delay I_α.

In the case of a downward tuning adjustment, we define the control variable adjustment at a specific time as the remaining reduction to be applied to the control variable.

We formally define the control variable and the control variable adjustment as:

Definition 13. Let $t \in [T_0, T]$. $\Gamma(t) \in \mathbb{R}^+$ is the control variable prior to being tuned at time t. The SDAM is initialized with the control variable equal to the initial price divided by the initial target debt. Γ^{**} is the control variable after being tuned at time t. Assume that the SDAM is tuned at time t. Then,

$$\Gamma(T_0) = \frac{\Phi_0}{\delta(T_0)}$$

$$\Gamma^{**}(t) = max\left(\Gamma(t), \frac{\Phi(t)}{\delta^*(t)}\right)$$

Definition 14. Let $t \in [T_0, T]$. $\alpha(t) \in \mathbb{R}^+$ is the control variable adjustment remaining to be applied prior to the SDAM being tuned at time t. The SDAM is initialized with no adjustment to be applied. $\alpha^{**}(t)$ is the control variable adjustment remaining to be applied after the SDAM is tuned at time t. Assume that the SDAM is tuned at t. Then,

$$\alpha(T_0) = 0$$

$$\alpha^{**}(t) = max(\Gamma(t) - \Gamma^{**}(t), 0)$$

where $\Gamma(t)$ and $\Gamma^{**}(t)$ are the control variable values at time t as defined above. Additionally, let $\Delta t \in (0, T-t]$ be a small time increment such that the SDAM is not tuned again from t to $t + \Delta t$ and where $\Delta t < I_\alpha$. Then,

$$\alpha(t + \Delta t) = \alpha(t)\left(1 - \frac{\Delta t}{I_\alpha}\right)$$

We can then induce a generalized result for $\alpha(t)$:

Lemma 3. *Let $t \in [T_0, T]$. Then,*

$$\alpha(t) = \begin{cases} 0 & \text{if } t - T_\Gamma(t) \geq I_\alpha \text{ or } \Gamma(T_\Gamma(t)) \leq \frac{\Phi(T_\Gamma(t))}{\delta^*(T_\Gamma(t))} \\ \left(\Gamma(T_\Gamma(t)) - \frac{\Phi(T_\Gamma(t))}{\delta^*(T_\Gamma(t))}\right)\left(1 - \frac{t - T_\Gamma(t)}{I_\alpha}\right) & \text{if } t - T_\Gamma(t) < I_\alpha \text{ and } \Gamma(T_\Gamma(t)) > \frac{\Phi(T_\Gamma(t))}{\delta^*(T_\Gamma(t))} \end{cases}$$

where I_α is the tune adjustment delay, $T_\Gamma(t)$ is the last tune time, $\Phi(t)$ is the SDAM price, $\delta(t)$ is the target debt, $\Gamma(t)$ is the control variable at time t.

After defining the control variable adjustment, we define the control variable over an incremental timestamp as follows:

Definition 15. Let $t \in [T_0, T]$. Assume the SDAM is tuned at time t. Let $\Delta t \in (0, T-t]$ be a small time increment such that the SDAM is not tuned again from t to $t + \Delta t$. Then,

$$\Gamma(t + \Delta t) = \Gamma^{**}(t) - (\alpha^{**}(t) - \alpha(t + \Delta t))$$

where $\Gamma^{**}(t)$ is the control variable after being tuned at time t, $\alpha^{**}(t)$ is the control variable adjustment calculated when the SDAM was tuned at time t, and $\alpha(t + \Delta t)$ is the control variable adjustment remaining at time $t + \Delta t$ as defined above.

4 Implementation

The Ethereum Virtual Machine (EVM) compatible implementation of the SDAM model [1] requires approximation of the defined model using integer arithmetic. Let $\mathbb{U} = \mathbb{Z} \bigcap [0, 2^{256} - 1]$ represent the range of allowable unsigned, 256-bit integer values used to approximate the real-valued system in the implementation.

Due to the potential range of token decimals and differences in token value (and therefore, their relative price) of a SDAM, certain operations could be prone to under- or overflows for either the final result or intermediate computations and would cause the transaction to revert. Additionally, approximating the real-valued system using integer arithmetic requires scaling values by a constant when performing integer division to avoid loss of precision. To address both issues, we introduce a scaling constant S for each SDAM, calculated from its inputs on creation.

In this section, we also formalize the state transition functions *purchase* and *tune*, and the *debtDecay*, *controlDecay*, and *marketPrice* operations that they depend on. We first formalize their mathematical definitions, $debtDecay_{spec}$, $controlDecay_{spec}$, $marketPrice_{spec}$, $purchase_{spec}$, and $tune_{spec}$, that use real arithmetic from the model defined previously. Then, we formalize their implementation, $debtDecay_{code}$, $controlDecay_{code}$, $marketPrice_{code}$, $purchase_{code}$, and $tune_{code}$, that uses the integer arithmetic, and analyze the approximation errors due to the integer rounding. We also analyze the required input bounds and scaling value to avoid integer under/overflows in the defined functions.

4.1 Creating SDAMs

4.1.1 Scaling Values

A common practice in Ethereum applications is to use the same precision for the price variable as the token has configured for its decimals, making calculations simpler for developers interfacing with the system. Consider the following example for a SDAM in which both the quote token and payout token have 18 decimals of precision and we use the same precision for the price. This example illustrates how using a scaling value in the system calculations avoids precision loss with simple values that we can also abstract to a more general case.

Assume that the quote token has a price of \$1 and the payout token has a price of \$5. Therefore, the price ratio is 5 quote tokens to 1 payout token. If the price is configured with 18 decimals of precision, then we could set the initial price as $\Phi_0 = 5 \times 10^{18}$ and a scale factor $S = 10^{18}$. Let initial capacity $C_0 = 20,000 \times 10^{18}$, duration $L = 432,000$ (5 days), and the debt decay interval $I_D = 259,200$ (3 days). Then, then we can calculate the debt at T_0 as:

$$D(T_0) = C_0 \times \frac{I_D}{L} = 20,000 \times 10^{18} * \frac{259,200}{432,000} = 12,000 \times 10^{18}$$

Finally, we calculate the initial control variable as:

$$\Gamma(T_0) = \Phi_0 \times S \div D(T_0) = 5 \times 10^{18} \times 10^{18} \div (12,000 \times 10^{18}) = 4.166.. \times 10^{14}$$

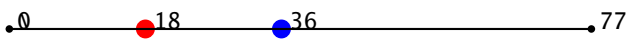
We can use the same scaling to calculate the payout amount for exchanging quote tokens. Assume quote tokens received $r(t) = 100 \times 10^{18}$, price $\Phi(t) = 5 \times 10^{18}$, and scale factor $S = 10^{18}$, then we have:

$$p(t) = r(t) \times S \div \Phi(t) = 100 \times 10^{18} \times 10^{18} \div (5 \times 10^{18}) = 20 \times 10^{18}$$

The above approach is valid within the implementation constraints for the trivial case of the tokens having the same decimal precision and prices that are similar orders of magnitude. However, it does not generalize to a large range of potential inputs.

Let's now consider a more complex example. Assume that the quote token has a price of \$100,000 and the payout token has a price of \$0.000001. Therefore, the price ratio is 0.00000000001 (10^{-10}) quote tokens to 1 payout token. Additionally, assume that the quote token has 18 decimals of precision and the payout token has 9 decimals of precision. If we update the price to account for the difference in precision, then we have a ratio of 10^{-19} quote tokens to payout tokens. If the price is configured with 18 decimals of precision, then we would try to set the initial price as $\Phi_0 = 10^{-19} \times 10^{18} = 0.1$, which rounds to zero with integer arithmetic. In addition to not having enough precision in the price variable, there are other possible scenarios where the price is in bounds, but we have under- or overflows with the control variable calculation. As a result, we need to construct a scaling system that allows for the widest range of practical values to have the correct precision.

The range of allowable base 10 values for a 256-bit unsigned integer is $[0, 1.15 \times 10^{77}]$. The midpoint of this range on a logarithmic scale is roughly 10^{38}. Our intuition is that setting the precision of our price variable around this midpoint in the base case allows for the greatest range of shifts for token decimal and price differences while avoiding under- and overflows. We choose to use 10^{36} as our midpoint since it is a multiple of most common decimal values, e.g. 6, 9, and 18, and is close to the midpoint.



We scale calculations between the debt and control variables to avoid loss of precision for price calculations. The scale variable S serves two purposes:

1. Scale quote token units and payout token units when converting between them with the price variable.

2. Converting between price, debt, and control variable values.

The scale factor should be adjusted from the midpoint to properly convert between the tokens' decimal precisions at a wide range of potential input values. We must account for the difference in the precision of the payout token and the quote token. We also account for half the difference of their price decimals in the scale factor, which can offset or add to the decimal precision difference. Ideally, we would account for the entire difference in price decimals on the price variable, but it limits the range of inputs that the SDAM can support.

We define the scale adjustment and scale factor as follows:

Definition 16. Let s be the scale adjustment from the midpoint and S be the scale factor for the SDAM. Then,

$$s = d_p - d_q - \frac{d_{\phi_p} - d_{\phi_q}}{2}$$

and

$$S = 10^{36+s}$$

where d_p is the configured decimals for the payout token, d_q is the configured decimals for the quote token, d_{ϕ_p} is the price decimals of the payout token relative to a common unit (such as dollars), and d_{ϕ_q} is the price decimals of the quote token relative to a common unit.

The scale adjustment s must be bound to a certain range to avoid under- and overflows of the scale factor S and values derived from it. Therefore, the Bond Protocol implementation requires that $s \in \mathbb{Z} \bigcap [-24, 24]$, and $d_q, d_p \in \mathbb{Z} \bigcap [6, 18]$. Therefore, the scale factor $S \in \mathbb{U} \bigcap [10^{12}, 10^{60}]$. These bounds allow the system to support SDAMs with a broad range of relative token prices such that:

$$|d_q - d_p| - 24 \leq |d_{\phi_q} - d_{\phi_p}| \leq 24 - |d_q - d_p|$$

4.1.2 Initializing Price, Debt, and Control Variable

We use the scale adjustment defined in the previous section to determine an optimal precision for the price variable. Ideally, our price variable stays around our midpoint to allow a broad range of values for the debt and control variables. However, slight adjustments are required based on differences in the price decimals of tokens to avoid under- and overflow issues with token unit conversions.

We formally define the initial price and the initialization of the debt and control variables.

Definition 17. Let Φ_0 be the initial price of the SDAM. Then,

$$\Phi_0 = \frac{\phi_p}{\phi_q} \times 10^{36+s+d_q-d_p+d_{\phi_p}-d_{\phi_q}}$$

where s is the scale adjustment as defined above, d_p is the configured decimals for the payout token, d_q is the configured decimals for the quote token, d_{ϕ_p} is the price decimals of the payout token relative to a common unit (such as dollars), d_{ϕ_q} is the price decimals of the quote token relative to a common unit, ϕ_p is the price coefficient of the payout token price relative to a common unit, and ϕ_q is the price coefficient of the quote token relative to a common unit.

Φ_0 is calculated and supplied by the maker, therefore the approximated value is equal to the real value.

Definition 18. Let T_0 be the start time, $D_{spec}(T_0)$ be the initial real value of the debt, and $D_{code}(T_0)$ be the initial integer approximation of the debt. Then,

$$D_{spec}(T_0) = C_0 \frac{I_D}{L}$$

and

$$D_{code}(T_0) = \left\lfloor \frac{C_0 \times I_D}{L} \right\rfloor$$

where C_0 is the initial capacity (in payout token units), I_D is the debt decay interval, and L is the duration.

Definition 19. Let T_0 be the start time, $\Gamma_{spec}(T_0)$ be the initial real value of the control variable, and $\Gamma_{code}(T_0)$ be the initial integer approximation of the control variable. Then,

$$\Gamma_{spec}(T_0) = \frac{\Phi_0 \times S}{D_{spec}(T_0)}$$

$$\Gamma_{code}(T_0) = \left\lfloor \frac{\Phi_0 \times S}{D_{code}(T_0)} \right\rfloor$$

where $\Phi(T_0)$ is the initial price, S is the scale factor, and $D(T_0)$ is the initial debt.

The initial control variable approximation $\Gamma_{code}(T_0)$ is rounded down with floor division to ensure that the approximated price value by the SDAM (which takes a ceiling value as described below) equals the initial value provided by the maker, i.e. $\Phi_0 = \Phi_{code}(T_0)$.

4.2 Purchases

Users primarily interact with a SDAM when purchasing payout tokens with quote tokens and trigger state transitions. The state transition depends on the decay of debt and the control variable to update the price prior to the state transition. Here we evaluate the implementation of the *debtDecay*, *controlDecay*, *marketPrice*, *purchase*, and *tune* operations using integer arithmetic and compare the resultant values to the specification with real values.

4.2.1 $debtDecay_{spec}$

As stated previously, the time-based dutch auction price discount is implemented by decaying the debt variable. *debtDecay* calculates the amount to reduce the stored value of debt by at the current timestamp without changing the state. Here, we define the specification of *decayDebt* with real values.

Definition 20. Let $debtDecay_{spec} \in \mathbb{R}^+$ be the amount of decay to apply to the stored debt value at $t \in [T_0, T]$. Then,

$$debtDecay_{spec} = D^*(T_p(t)) \times \frac{max(t - T_D(t), 0)}{I_D}$$

such that,

$$D_{spec}(t) = D^*(T_p(t)) - debtDecay_{spec} = D^*(T_p(t)) \left(1 - \frac{max(t - T_D(t), 0)}{I_D}\right)$$

where $T_p(t)$ is the time of the last purchase prior to t, $T_D(t)$ is the debt decay reference time prior to any purchases at time t, I_D is the debt decay interval, $D_{spec}(t)$ is the debt prior to any purchases at time t, and $D^*(t)$ is the debt following any purchases at time t.

4.2.2 $decayDebt_{code}$

We now define the approximation of *decayDebt* with integer arithmetic.

Definition 21. Let $debtDecay_{code} \in \mathbb{U}$ be the approximated amount of decay to apply to the stored debt value at $t \in [T_0, T]$. Then,

$$debtDecay_{code} = \left\lfloor \frac{D^*(T_p(t)) \times max(t - T_D(t), 0)}{I_D} \right\rfloor$$

such that,

$$D_{code}(t) = D^*(T_p(t)) - debtDecay_{code} = D^*(T_p(t)) - \left\lfloor \frac{D(T_p(t)) \times max(t - T_D(t), 0)}{I_D} \right\rfloor$$

where $T_p(t)$ is the time of the last purchase prior to t, $T_D(t)$ is the debt decay reference time prior to any purchases at time t, I_D is the debt decay interval, $D_{code}(t)$ is the debt prior to any purchases at time t, and $D^*(t)$ is the debt following any purchases at time t.

Theorem 4. *Let $D_{spec}(t) = D^*(T_p(t)) - debtDecay_{spec}$ and $D_{code}(t) = D^*(T_p(t)) - debtDecay_{code}$. Then, we have:*

1. $debtDecay_{spec} \geq debtDecay_{code}$
2. $D_{spec}(t) \leq D_{code}(t)$

The integer implementation has the desirable property that the *debtDecay* value is less than or equal to the real value. Therefore, the computed debt value is greater than or equal to the real value.

4.2.3 $controlDecay_{spec}$

The model specifies that downward tuning adjustments to the control variable happen over the tune adjustment interval instead of immediately. *controlDecay* calculates the amount to reduce the stored control variable by at the current timestamp without changing the state. We now define the specification of *controlDecay* using real values.

Definition 22. Let $controlDecay_{spec} \in \mathbb{R}^+$ be the amount of adjustment to apply to the stored control variable value at $t \in [T_0, T]$. Then,

$$controlDecay_{spec} = \alpha^{**}(T_\Gamma(t)) \times \frac{min(t - T_\Gamma(t), I_\alpha)}{I_\alpha}$$

such that,

$$\Gamma_{spec}(t) = \Gamma^{**}(T_\Gamma(t)) - controlDecay_{spec} = \Gamma^{**}(T_\Gamma(t)) - \alpha^{**}(T_\Gamma(t)) \times \frac{min(t - T_\Gamma(t), I_\alpha)}{I_\alpha}$$

where $T_\Gamma(t)$ is the time of the last tune prior to any purchases at time t, I_α is the control variable adjustment interval, $\Gamma_{spec}(t)$ is the control variable prior to a tune at time t, $\Gamma^{**}(t)$ is the control variable following a tune at time t, and α^{**} is the control variable adjustment following a tune at time t.

4.2.4 $controlDecay_{code}$

We now define the approximation of *controlDecay* using integer arithmetic.

Definition 23. Let $controlDecay_{code} \in \mathbb{U}$ be the approximated amount of adjustment to apply to the stored control variable value at $t \in [T_0, T]$. Then,

$$controlDecay_{code} = \left\lfloor \frac{\alpha^{**}(T_\Gamma(t)) \times min(t - T_\Gamma(t), I_\alpha)}{I_\alpha} \right\rfloor$$

such that,

$$\Gamma_{code}(t) = \Gamma^{**}(T_\Gamma(t)) - controlDecay_{code} = \Gamma^{**}(T_\Gamma(t)) - \left\lfloor \frac{\alpha^{**}(T_\Gamma(t)) \times min(t - T_\Gamma(t), I_\alpha)}{I_\alpha} \right\rfloor$$

where $T_\Gamma(t)$ is the time of the last tune prior to any purchases at time t, I_α is the control variable adjustment interval, $\Gamma_{code}(t)$ is the control variable prior to a tune at time t, $\Gamma^{**}(t)$ is the control variable following a tune at time t, and α^{**} is the control variable adjustment following a tune at time t.

Theorem 5. *Let* $\Gamma_{spec}(t) = \Gamma^{**}(T_\Gamma(t)) - controlDecay_{spec}$ *and* $\Gamma_{code}(t) = \Gamma^{**}(T_\Gamma(t)) - controlDecay_{code}$. *Then, we have:*

1. $controlDecay_{spec} \geq controlDecay_{code}$
2. $\Gamma_{spec}(t) \leq \Gamma_{code}(t)$

The integer implementation has the desirable property that the *controlDecay* value is less than or equal to the real value. Therefore, the computed control variable value is greater than or equal to the real value.

4.2.5 $marketPrice_{spec}$

marketPrice calculates the current price of the SDAM without changing the state. It uses the values provided by *debtDecay* and *controlDecay* to calculate the current value. We use the calculated scale factor for comparison to the integer approximation. We define the specification of *marketPrice* using real values.

Definition 24. Let $marketPrice_{spec} \in \mathbb{R}^+$ be the SDAM price at $t \in [T_0, T]$. Then,

$$marketPrice_{spec} = max\left(\frac{D_{spec}(t) \times \Gamma_{spec}(t)}{S}, \Phi_{min}\right)$$

where $D_{spec}(t)$ is the debt prior to any purchases at time t as defined above, $\Gamma_{spec}(t)$ is the control variable prior to a tune at time t as defined above, S is the scale factor, and Φ_{min} is the minimum price.

4.2.6 $marketPrice_{code}$

We now define the approximation of *marketPrice* in the system using integer arithmetic.

Definition 25. Let $marketPrice_{code} \in \mathbb{U}$ be the approximated SDAM price at $t \in [T_0, T]$. Then,

$$marketPrice_{code} = max\left(\left\lceil \frac{D_{code}(t) \times \Gamma_{code}(t)}{S} \right\rceil, \Phi_{min}\right)$$

where $D_{code}(t)$ is the approximated value of the debt prior to any purchases at time t as defined above, $\Gamma_{code}(t)$ is the approximated value of the control variable prior to a tune at time t as defined above, S is the scale factor, and Φ_{min} is the minimum price.

Theorem 6. *Let $t \in [T_0, T]$, $\Phi_{spec}(t) = marketPrice_{spec}$, and $\Phi_{code}(t) = marketPrice_{code}$. Then, we have:*

$$\Phi_{spec}(t) \leq \Phi_{code}(t)$$

The integer implementation has the desirable property that the approximated price from the decay of stored values is greater than or equal to the real value of the price calculated from the stored values, which protects makers from selling tokens at a lower price than expected.

4.2.7 $purchase_{spec}$

purchase implements the state transition logic associated with a taker purchase on the SDAM. Here, we define the specification of *purchase* with real values.

Definition 26. $purchase_{spec}$ takes inputs id i, amount of quote tokens in q, and minimum amount out of payout tokens p_{min}, where $i \in \mathbb{Z}_0^+$ and $(q, p_{min}) \in \mathbb{R}^+ \times \mathbb{R}^+$, and updates the state at time $t \in (T_0, T]$, as follows:

$$(C, D, T_p, T_D) \xrightarrow{purchase_{spec}(i,q,p_{min})} (C^*, D^*, T_p^*, T_D^*)$$

with intermediate calculations

$$p(t) = \frac{q \times S}{\Phi(t)}$$

$$r(t) = q \times (1 - \epsilon)$$

$$f(t) = q \times \epsilon$$

such that

$$C^*(t) = C(t) - p(t)$$

$$D^*(t) = D(t) + p(t)$$

$$T_p^*(t) = t$$

$$T_D^*(t) = T_D(t) + I_D \frac{p(t)}{\delta(t)}$$

where $\Phi(t)$ is the price prior to the state transition at time t.

Lemma 7. *Let $(C, D, T_p, T_D) \xrightarrow{purchase_{spec}(i,q,p_{min})} (C^*, D^*, T_p^*, T_D^*)$. Then, we have the following:*

1. $0 \leq C^* < C \leq C_0$
2. $0 \leq D < D^*$
3. $T_0 \leq T_p < T_p^* \leq T$
4. $T_0 < T_D < T_D^*$

4.2.8 $purchase_{code}$

We now define the approximation of *purchase* with integer arithmetic and establish the rounding behaviors of the implementation.

Definition 27. $purchase_{code}$ takes inputs id $i \geq 0$, amount in (quote token) $q > 0$, and minimum amount out of payout tokens $p_{min} > 0$, where $i, q, p_{min} \in \mathbb{U}$, and updates the state at time $t \in \mathbb{U}$, where $T_0 \leq t \leq T$, as follows:

$$(C, D, T_p, T_D) \xrightarrow{purchase_{code}(i,q,p_{min})} (C^{'}, D^{'}, T_p^{'}, T_D^{'})$$

with intermediate calculations

$$p^{'}(t) = \left\lfloor \frac{q \times S}{\Phi(t)} \right\rfloor$$

$$r^{'}(t) = \lceil q \times (1 - \epsilon) \rceil$$

$$f^{'}(t) = \lfloor q \times \epsilon \rfloor$$

such that

$$C^{'}(t) = C(t) - p^{'}(t)$$

$$D^{'}(t) = D(t) + p^{'}(t) + 1$$

$$T_p^{'}(t) = t$$

$$T_D^{'}(t) = T_D(t) + \left\lceil \frac{I_D \times p(t)}{\delta(t)} \right\rceil$$

where $\Phi(t)$ is the price prior to the state transition at time t.

Theorem 8. *Let* $(C, D, T_p, T_D) \xrightarrow{purchase_{spec}(i,q,p_{min})} (C^*, D^*, T_p^*, T_D^*)$, $(C, D, T_p, T_D) \xrightarrow{purchase_{code}(i,q,p_{min})} (C^{'}, D^{'}, T_p^{'}, T_D^{'})$, $\Phi^*(t)$ *be the price following* $purchase_{spec}$*, and* $\Phi^{'}(t)$ *be the price following* $purchase_{code}$*. Then, we have the following:*

1. $0 < p^{'} \leq p < p^{'} + 1$
2. $0 < r \leq r^{'}$
3. $0 \leq f^{'} \leq f$
4. $0 \leq C^* \leq C^{'} < C \leq C_0$
5. $0 \leq D < D^* < D^{'}$
6. $T_0 \leq T_p^{'} = T_p^* \leq T$
7. $T_0 < T_D^* \leq T_D^{'}$

Further, this implies that:

$$\Phi^*(t) \leq \Phi^{'}(t)$$

With integer arithmetic, we have the desirable property that the payout $p^{'}$ will be less than or equal to the real value p. Therefore, the SDAM will not pay out more tokens than expected for a given purchase or in total. The issuer also receives $r^{'}$ quote tokens, which is at least as much as expected by the real value r. This is at the expense of the fee recipient since the fee $f^{'}$ will be at most the real value f. We also have the desirable property that the approximated debt value $D^{'}$ is greater than the real value D^*, which implies the approximated price $\Phi^{'}$ is greater than the real value Φ^* following the state transition.

4.2.9 $tune_{spec}$

A purchase can also trigger a second state transition to tune the price based on whether the SDAM is under- or oversold, as described previously. *tune* calculates the updated control variable and associated values in this second state transition. Here we define the specification of *tune* using real values.

Definition 28. $tune_{spec}$ takes inputs id i, time t, and price Φ, where $i \in \mathbb{Z}_0^+$, $t \in [T_0, T]$, and $\Phi \in \mathbb{R}^+$, and updates the state, as follows:

$$(\Gamma, \alpha, T_\Gamma) \xrightarrow{tune_{spec}(i,t,\Phi)} (\Gamma^{**}, \alpha^{**}, T_\Gamma^{**})$$

with intermediate calculations

$$\chi^{**}(t) = C_0 \frac{t - T_0}{L} + C^*(t)$$

$$\delta^{**}(t) = \chi^{**}(t) \frac{I_D}{L}$$

such that

$$\Gamma^{**}(t) = max\left(\Gamma(t), \frac{\Phi(t) \times S}{\delta^{**}(t)}\right)$$

$$\alpha^{**}(t) = max\left(\Gamma(t) - \frac{\Phi(t) \times S}{\delta^{**}(t)}, 0\right)$$

$$T_\Gamma^{**}(t) = t$$

Lemma 9. *Let* $(\Gamma, \alpha, T_\Gamma) \xrightarrow{tune_{spec}(i,t,\Phi)} (\Gamma^{**}, \alpha^{**}, T_\Gamma^{**})$. *Then, we have the following:*

1. $0 \leq \Gamma \leq \Gamma^{**}$
2. $0 \leq \alpha^{**}$
3. $0 \leq T_\Gamma < T_\Gamma^{**} \leq T$

4.2.10 $tune_{code}$

We now define the approximation of *tune* using integer arithmetic and establish the rounding behaviors of the implementation.

Definition 29. $tune_{code}$ takes inputs id i, time t, and price $\Phi(t)$, where $i \in \mathbb{U}$, $t \in [T_0, T] \bigcap \mathbb{U}$, and $\Phi \in \mathbb{U}$, and updates the state, as follows:

$$(\Gamma, \alpha, T_\Gamma) \xrightarrow{tune_{code}(i,t,\Phi)} (\Gamma^{''}, \alpha^{''}, T_\Gamma^{''})$$

with intermediate calculations

$$\chi^{''}(t) = \left\lfloor \frac{C_0 \times (t - T_0)}{L} \right\rfloor + C^{'}(t)$$

$$\delta^{''}(t) = \left\lfloor \frac{\chi^{''}(t) \times I_D}{L} \right\rfloor$$

such that

$$\Gamma^{''}(t) = max\left(\Gamma(t), \left\lceil \frac{\Phi(t) \times S}{\delta^{''}(t)} \right\rceil\right)$$

$$\alpha^{''}(t) = max\left(\Gamma(t) - \left\lceil \frac{\Phi(t) \times S}{\delta^{''}(t)} \right\rceil, 0\right)$$

$$T_\Gamma^{''}(t) = t$$

Theorem 10. *Let* $(\Gamma, \alpha, T_\Gamma) \xrightarrow{tune_{spec}(i,t,\Phi)} (\Gamma^{**}, \alpha^{**}, T_\Gamma^{**})$ $(\Gamma, \alpha, T_\Gamma) \xrightarrow{tune_{code}(i,t,\Phi)} (\Gamma^{''}, \alpha^{''}, T_\Gamma^{''})$, $\Phi^{**}(t)$ *be the price following* $tune_{spec}$*, and* $\Phi^{''}(t)$ *be the price following* $tune_{code}$*. Then, we have the following:*

1. $0 \leq \Gamma \leq \Gamma^{**} \leq \Gamma^{''}$
2. $0 \leq \alpha^{''} \leq \alpha^{**}$
3. $0 \leq T_\Gamma < T_\Gamma^{''} = T_\Gamma^{**} \leq T$

Further, this implies that:

$$\Phi^{**} \leq \Phi^{''}$$

With the implementation using integer arithmetic, we have the desirable properties that the control variable $\Gamma^{''}$ will be at least as large as the real value Γ^{**}, which implies that the approximated price $\Phi^{''}$ will be greater than or equal to the real value Φ^{**} following the state transition. Additionally, the approximated control variable adjustment $\alpha^{''}(t)$ is at most the real value $\alpha^{**}(t)$ which means the control variable, and, therefore price, will not be adjusted down more than desired.

5 Acknowledgements

The system described in this paper is the work of several individuals over multiple iterations. The paper was written primarily by Oighty, but relied on the previous work of Zeus, PottedMeat, and indigo. Additionally, several contributors to the OlympusDAO Policy and Olympus Pro teams have provided valuable ideas, feedback, and suggestions over the course of their development. Specific individuals we would like to thank are:

- Tex - Olympus Pro and Bond Protocol lead. Provided requirements, reviewed solutions, and provided feedback on this paper.
- 0xEvan - Provided thorough review and edits on the structure and content of this paper.
- JFry - Provided feedback based on OlympusDAO's bonding requirements and reviewed this paper.
- JeffX - Implemented previous versions of the Olympus bond system and contributed to the current design discussions.
- spaceturtleship - Provided front-end application feedback on the system implementation and helped troubleshoot bugs on testnet.

References

[1] Oighty et al. *Bond Protocol smart contracts*. Bond Protocol. 2022. URL: `https://github.com/Bond-Protocol/bond-contracts`.