ChatGPT

Bond Market Making Simulation

Executive Overview

We are building a simulated bond market environment to develop and test bond **market-making** strategies. Real bond trading data can be costly and hard to obtain, so we generate realistic synthetic data that mimics how bond prices and trading activity behave over time. This project allows us to:

- Understand how a market maker quotes prices and manages inventory risk in a bond market.
- Experiment with how trading activity responds to price spreads and market conditions.
- Develop and backtest our own algorithm for making markets in bonds under controlled, simulated conditions.

This simulation provides a simplified foundation that we will continue to refine with added complexity and realism.

Interest Rate Model and Bond Pricing

At the core of our simulation is an **interest rate model** driving bond prices. We use the **Cox–Ingersoll–Ross (CIR)** short-rate model for realism:

- **Short Rate Dynamics (CIR Model)** – The short-term interest rate `r(t)` follows the stochastic differential equation:
  `dr(t) = a [b - r(t)] dt + σ √r(t) dW(t)`
  where **a** is the speed of mean reversion, **b** is the long-run mean rate, and **σ** is the volatility of interest rate fluctuations. We calibrated these parameters to market data (e.g. zero-coupon bond yields) to ensure realistic behavior (for example, $a \approx 0.10$, $b \approx 0.06$, $\sigma \approx 0.028$ based on one calibration, implying the rate tends to revert toward ~6% at 10% speed). The stochastic term `√r(t) dW(t)` ensures rates remain non-negative. We simulate the short rate path at a high frequency (per second) using an **Euler discretization** of this SDE.

- **10-Year Bond Mid-Price** – Our **mid-price** for the bond is derived from the short rate using the CIR model's **zero-coupon bond pricing formula**. Given the current short rate `r` and time to maturity `T` (here $T=10$ years initially, declining over time), the theoretical bond price is:
  `P(r, T) = A(T) * exp(-B(T) * r)`
  where *A(T)* and *B(T)* are functions of the model parameters (a, b, σ) and maturity. This closed-form solution comes from the CIR model, and we use it to compute a realistic price for a 10-year zero-coupon bond at each time step. As the simulation progresses, we decrease the time to maturity slightly each second (so a bond initially 10 years from maturity will be ~9 years by the end of a one-year simulation). The initial short rate *r(0)* is set such that the initial bond price matches observed market prices (from the calibration data).

Synthetic Market Data Generation

Using the above model, we generate **tick-level bond market data** for a series of trading days. Each trading day is simulated at one-second intervals from 8:00 AM to 5:00 PM (9 hours of trading). For example, we can simulate 10 trading days (or more, e.g. ~252 days for a full year) of second-by-second data. Key components of the synthetic data are:

- **Mid-Price (Bond)** – The mid price evolves according to the 10-year bond price computed from the CIR short-rate model. This introduces realistic trends and mean-reversion in the price dynamics, as opposed to a simple random walk. The mid-price will exhibit volatility driven by interest rate changes (from the CIR simulation).

- **Bid-Ask Spread** – We simulate a **market spread** that varies randomly each second to reflect changing liquidity. At each second, the spread is drawn from a uniform distribution in a reasonable range (for example, `spread ~ U(0.01, 0.04)` in price terms). This means the typical full bid-ask spread is between 1 and 4 cents (if price is around 1.0, or scaled accordingly if price is different).

- **Bid and Ask Prices** – Given the mid-price and the spread for that second, we set the **bid** and **ask** quotes as:
  `bid_price = mid_price - spread/2`
  `ask_price = mid_price + spread/2`
  This yields a symmetric quotation around the mid. (In our simulation, we ensure `bid_price` never exceeds the mid, and `ask_price` never falls below the mid.)

- **Order Arrival Intensity** – We model the arrival of buy/sell orders hitting the quotes as a **Poisson process** that depends on the spread. Specifically, the **trade arrival rate** λ (lambda) decreases as the bid-ask spread widens (customers trade less when the market is wide). We assume a functional form:
  `λ = A * exp(-k * spread)`
  where **A** is a baseline arrival rate (when spread → 0) and **k** is a sensitivity factor that makes volume drop off with wider spreads. We introduce variability in *k* to reflect changing market conditions: in the simulation *k* is drawn each second from a normal distribution (for example, `k ~ N(60, 2)`, capped between 30 and 90). This means on average, a 1-cent increase in spread reduces the trade rate by a factor of `e^{-60*0.01} ≈ 0.55` (45% drop), though the exact sensitivity fluctuates over time. We set *A* to achieve a realistic activity level – for instance, around 5 trades per second when the spread is near zero (split between buys and sells).

- **Trade Volume per Second** – Given the rate λ for that second, we sample the **number of trades** in that second from a Poisson distribution with mean λ. Each trade is considered a buyer-initiated or seller-initiated hit on the quotes. In the basic data generation, we can treat this as total trade count (since the spread is small, roughly half the trades will be buys and half sells on average). This yields a column `volume` (trades per second) in our simulated dataset. The randomness of the Poisson draw produces realistic variability in per-second trading volume.

- **Timestamp Index** – Each data point is time-stamped at one-second intervals during market hours. We structure the data with one row per second of the trading day (e.g. 8:00:00 to 16:59:59), and continue this for each simulated day. This produces a realistic time series dataset of prices and volumes. For example, one full trading day yields 9 hours * 3600 seconds = **32,400 rows** of data. Multiple days are concatenated for longer simulations.

All the simulated data (mid prices, bid, ask, spread, and volume per second) is saved for use in strategy backtesting. By construction, this dataset captures key dynamics: mean-reverting price trends (driven by interest rates), variable spreads, and a realistic inverse relation between spread and trading activity.

Market-Making Strategy Backtest

Using the synthetic market data, we implement a simple **market-making strategy** to evaluate performance. The strategy involves a simulated market maker who continuously posts bids and asks and manages an inventory of bonds. The goal is to earn the bid-ask spread while controlling risk. Our backtest simulates how this market maker would perform against the incoming order flow in the synthetic market. Below is an outline of the strategy and simulation process:

1. **Rolling Liquidity Estimation** – The market maker *observes* recent trading activity to gauge current liquidity. At each time step (each second), the agent looks at the past window (e.g. last 1000 seconds of data) and re-estimates the parameters *k* and *A* based on the observed `spread` vs `volume` relationship. This is done by fitting a model of the form `log(volume) ≈ log(A) - k * spread` over that window. The result gives an estimated *k* (spread sensitivity) and *A* (baseline intensity) for the current market conditions. This lets the agent adapt to any shifts in the simulated market's behavior over time (for instance, if overall trading slows down or becomes more sensitive to spreads).

2. **Quote Placement** – The market maker sets its **bid and ask quotes** for the current second. We start with a base half-spread (e.g. 0.02, meaning a 4 cent total spread) as the minimum spread the market maker wants to quote. Then we adjust the quotes based on the current **inventory** to manage risk:

3. If the agent has a positive inventory (long position in the bond), it will quote a **lower bid and ask** to encourage selling (and thus reduce inventory). Specifically, we subtract an amount `γ * inventory` from the bid and ask prices (where γ is an inventory sensitivity coefficient). This effectively widens the quoted spread when holding a long inventory, making the quotes more attractive to buyers and less attractive to sellers.
4. If the agent has a negative inventory (short position), it does the opposite, raising its bid and ask (narrowing the spread from the mid) to encourage buying bonds back.

In formula terms, if `mid_t` is the current mid price and `inv_t` is the current inventory:
`bid_t = mid_t - (base_spread/2) - γ * inv_t`
`ask_t = mid_t + (base_spread/2) - γ * inv_t`
We also enforce that `bid_t` never exceeds `mid_t` (if the formula would push it above mid, we cap it at mid minus a tiny epsilon) and similarly `ask_t` is kept at or above `mid_t`. This ensures a valid quote (bid <= mid <= ask). The parameter *γ* is chosen to balance inventory control with profitability (for example, γ might be a small number like 0.1, so that 10 extra bonds in inventory would adjust quotes by about 1 point).

1. **Trade Execution Model** – Once the market maker posts its quotes, we simulate the **incoming trades** for that second against those quotes. Using the previously estimated *A* and *k*, we calculate the **arrival rates** for buy and sell orders hitting the market maker:

2. `λ_bid = 0.5 * A * exp(-k * (mid_t - bid_t))` is the per-second rate of buy orders hitting our bid (note: `mid_t - bid_t` is essentially half the spread plus any inventory adjustment). The factor 0.5 splits the total intensity roughly in half for buys vs sells (assuming symmetric behavior around the mid price). If our bid is very close to mid (tight spread), `λ_bid` will be high (approaching 0.5 * A); if our bid is far from mid (wide spread), `λ_bid` drops significantly.
3. Similarly, `λ_ask = 0.5 * A * exp(-k * (ask_t - mid_t))` is the rate of sell orders hitting our ask. (Here `ask_t - mid_t` is also roughly half the quoted spread adjusted for inventory.)

We then **sample Poisson random variables** for the number of buys and sells in the current second: e.g. `N_bid ~ Poisson(λ_bid)` and `N_ask ~ Poisson(λ_ask)`. These `N_bid` and `N_ask` represent how many trades actually execute against our bid and ask quotes in that one-second interval.

1. **Inventory and Cash Update** – Each trade execution will change the market maker's position:
2. For each buy order that hits our bid (counted in `N_bid`), we **sell one bond** from inventory at the bid price. So we decrease our inventory by 1 and **increase cash** by the bid price (since we sold a bond and received cash). If we didn't have any inventory (inventory could go negative, representing a short position – we assume the market maker can short if necessary).
3. For each sell order that hits our ask (counted in `N_ask`), we **buy one bond** at the ask price. We increase our inventory by 1 (now holding an extra bond) and **decrease cash** by the ask price (we paid money to buy the bond).

Essentially, buys by others cause us to sell and reduce inventory, while sells by others cause us to buy and increase inventory. The cash account tracks the cumulative profit/loss from trades: selling at bid adds cash, buying at ask spends cash. The difference between what we sold and bought for (the spread) is our trading profit, while the inventory position carries risk if prices move.

1. **Iteration and Performance** – The above steps repeat for each second in the simulation. Over time, we obtain a time series of the market maker's **inventory** and **cash P&L**. We can analyze this output to evaluate performance: for example, computing the **profit** (realized P&L in cash plus any unrealized P&L on inventory), the **inventory level** over time (to see if it stays within reasonable bounds or tends to blow up), and **risk metrics**. We also examine how the strategy responds to changing market conditions (since k and A vary) and the effect of the inventory adjustment parameter γ. This simple strategy serves as a baseline – ideally, a successful market maker will earn small profits on the spread while keeping inventory risks low.

Through this backtest, we can observe whether the simulated market maker is able to profit from the bid-ask spread under our model assumptions and how sensitive its performance is to market conditions and strategy parameters. The entire framework is modular, allowing us to adjust assumptions (e.g. different interest rate models, varying spread distributions, more complex order arrival processes) and incorporate more sophisticated market-making strategies in future iterations. This updated simulation and strategy give us a more **realistic testbed** for algorithmic bond market-making than our initial version, while still being controlled and reproducible.
