Analysis of Stochastic Processes with Neural Networks

Molnár András
Supervisor: Dr. Lukács András

2026 May

1 Introduction and Motivation

In modern financial markets, algorithmic trading and high-frequency trading are important parts of price formation and liquidity provision. Many classical models describe the price of an asset with continuous stochastic processes, for example with geometric Brownian motion. These models are mathematically very useful, but they do not describe directly how prices change because of individual market orders, limit orders, cancellations, and changes in market liquidity.

The aim of this project is to study a more microstructure-based model, where price changes are generated by discrete events in the order book. The main reference of the project is the paper by Bank, Cartea, and Körber [1]. In this framework, the price is not driven by an external fundamental Brownian motion. Instead, price changes are caused by market order flow, and the size of the price impact depends on the current liquidity level of the market.

This semester I worked on understanding the full model in the paper and implementing its main parts in Python. The focus was not only on simulating the uncontrolled price process, but also on implementing the controlled trader problem. This includes signal-based trades, state-based trades, transaction costs, price impact, trading halts, terminal auction effects, and the numerical solution of the Hamilton–Jacobi–Bellman (HJB) Quasi-Variational Inequality (QVI). After implementing the HJB solver, I also used the resulting benchmark policy inside the simulator and checked with Monte Carlo (MC) simulations whether the simulated expected utility is consistent with the value function. Here, the value function represents the maximal expected utility that the trader can achieve when starting from a given state.

The longer-term goal is to use this simulator as a reinforcement learning environment. In this way, a learning agent could be trained to trade in the same market model, and its performance could be compared to the HJB benchmark solution.

2 The Market Model

The paper studies a reduced-form model of a financial market where price changes are generated by order flow. Here, the price moves only when market orders arrive, and the size of the price move depends on the current level of market liquidity.

The main state variable of the market is the liquidity process λ_t. It measures how much liquidity is available in the market. Liquidity increases when new limit orders are posted, and it decreases when market orders or cancellations consume liquidity. The liquidity dynamics are given by

$$d\lambda_t = dL_t - |dM_t|,$$

where L_t denotes the cumulative limit order and cancellation flow, while M_t denotes the cumulative market order flow. Positive limit order flow increases liquidity, while market orders always reduce liquidity, independently of whether they are buy or sell market orders.

Both order flows are modeled using a marked Poisson point process. The dynamics of limit and market orders are

$$dL_t = \int_{E\times R_+} \mathbf{1}_{\{y\le f(\lambda_{t-})\}}\rho(e)N(dt,de,dy),$$

and

$$dM_t = \int_{E\times R_+} \mathbf{1}_{\{y\le g(\lambda_{t-})\}}\eta(e)N(dt,de,dy).$$

Here N is a marked Poisson point process on $[0,\infty)\times E\times R_+$. The mark $e\in E$ determines the size and type of the possible order, while the auxiliary variable y is used for thinning. The functions $f(\lambda)$ and $g(\lambda)$ are state-dependent arrival intensities. The function f controls the arrival rate of limit orders and cancellations, while g controls the arrival rate of market orders. This thinning representation means that candidate Poisson points are generated first, and then an order is accepted only if the auxiliary variable y is below the corresponding intensity. For example, a candidate limit order is accepted if $y \le f(\lambda_{t-})$ and a candidate market order is accepted if $y \le g(\lambda_{t-})$. The functions $\rho(e)$ and $\eta(e)$ determine the size and direction of the orders. If $\rho(e) > 0$, then a new limit order is posted and liquidity increases. If $\rho(e) < 0$, then a cancellation occurs and liquidity decreases. Similarly, if $\eta(e) > 0$, then a buy market order arrives, while if $\eta(e) < 0$, then a sell market order arrives.

2.1 Price Impact

The price process is driven by market orders. Limit orders and cancellations change liquidity, but they do not directly move the asset price. The price dynamics are

$$dP_t = I(\Delta_t M, \lambda_{t-}),$$

where $\Delta_t M = M_t - M_{t-}$ is the jump size of the market order flow at time t. The function I is the price impact function. The price impact of a market order of size Δ at liquidity level λ is defined as

$$I(\Delta,\lambda) = \operatorname{sgn}(\Delta)\int_0^{|\Delta|} \iota(\lambda - z)\,dz.$$

The function ι is the marginal price impact function. It describes how much an additional small unit of order size moves the price. Since ι is decreasing in liquidity, market orders have larger price impact when liquidity is low. This formulation is important because the impact of a large order is not simply linear in its size. Instead, as the order consumes liquidity, the remaining part of the order is executed at a lower liquidity level. Therefore, the integral form of I naturally captures nonlinear price impact.

2.2 The Controlled Trader

After defining the uncontrolled market, the paper introduces an active trader. The trader uses market orders to execute a position over a finite time horizon $[0,T]$. Her trades affect the market in the same way as external market orders: they consume liquidity and move the price through the same impact function.

The controlled state process is

$$S_t^C := (\lambda_t^C, Q_t^C, P_t^C, X_t^C), \qquad t \ge 0.$$

Here λ_t^C is the controlled liquidity process, Q_t^C is the trader's inventory, P_t^C is the asset price, and X_t^C is the trader's cash position. The superscript C indicates that these quantities depend on the trader's control process (strategy) C. The trader pays transaction costs. If she trades Δ shares when the pre-trade price is p, then the cash position changes by the value of the trade, the half-spread cost, and the nonlinear impact cost. The impact cost is

$$\Xi(\Delta,\lambda) = \int_0^{|\Delta|} I(z,\lambda)\,dz.$$

This is the total cost of walking through the book when executing the order.

2.3 Trade Signals and Timing of Events

A central feature of the paper is that the trader may receive a short-term signal about the upcoming external order flow. The signal is denoted by Z_t. It is generated from the same marked Poisson point process that drives the external order arrivals:

$$Z_t = \int_{\{t\}\times E\times R_+} z(e,y)N(ds,de,dy).$$

The function $z(e,y)$ determines what information the trader receives about the candidate order. The important point is that the signal arrives before the external order is fully revealed to the market. Therefore, the trader can use the signal to trade just before the external order changes the liquidity and price. This creates a precise timing structure at each event time:

$$\text{pre-trade state} \to \text{signal-based trade} \to \text{external order} \to \text{state-based trade}.$$

In the benchmark case, the signal is simple. A signal value $Z_t = -1$ means that a liquidity-taking event is coming. This may be a market order or a cancellation. A signal value $Z_t = 1$ means that a liquidity-providing event is coming, so a positive limit order will arrive. The trader does not know the exact size of the incoming order, only whether the next event will increase or decrease liquidity.

This is economically meaningful. If the trader knows that liquidity will be taken away soon, she may want to trade before liquidity becomes worse. On the other hand, if the trader knows that liquidity provision is coming, she may prefer to wait, because future price impact will be lower.

2.4 Portfolio Value

The paper defines the trader's realizable portfolio value as

$$W_t^C := w(X_t^C, Q_t^C, P_t^C, \lambda_t^C), \qquad t \in [0,T],$$

where

$$w(x,q,p,\lambda) = x + qp - \left(\zeta|q| + \Xi(q,\lambda)\right).$$

The term $x+qp$ is the usual mark-to-market value of the cash and inventory position. The term $\zeta|q|+\Xi(q,\lambda)$ subtracts the transaction costs and impact costs that would be paid if the current inventory were liquidated immediately. Therefore, w is not just the book value of the position, but the value after accounting for liquidation costs.

2.5 Trading Halt Mechanism

The model also includes a trading halt mechanism. The purpose of this is to prevent liquidity from becoming too low and to avoid degenerate strategies. Since the trader's own market orders affect liquidity and future order arrivals, without a liquidity lower bound the model could allow unrealistic or manipulation-like behavior.

The market has a lower liquidity threshold, denoted here by $\lambda_{\min}$. If a liquidity-taking order would push liquidity below this threshold, then the order is only partially executed, using the available liquidity above the threshold. After this, trading is halted. The paper denotes the controlled state process with trading halt by

$$\widetilde{S}_t^C = (\widetilde{\lambda}_t^C, \widetilde{Q}_t^C, \widetilde{P}_t^C, \widetilde{X}_t^C).$$

Before the halt time, the halted and non-halted state processes coincide. Once the halt is triggered, no further normal trading takes place.

2.6 Terminal Wealth

At terminal time, the trader must liquidate her remaining inventory. If there is enough liquidity, this can be done normally through the liquidation value w. If not, or if trading has been halted, the remaining part of the position is executed in an auction. The paper models the auction price uncertainty by an independent random shock Y.

The terminal wealth is

$$\widetilde{W}^C_{T+} = w(\widetilde{X}^C_{T+}, \widetilde{Q}^C_{T+}, \widetilde{P}^C_{T+}, \widetilde{\lambda}^C_{T+}) + Yr(\widetilde{Q}^C_{T+}, \widetilde{\lambda}^C_{T+}).$$

Here $r(q, \lambda) = (|q| - (\lambda - \lambda_{\min})_+)_+$ is the part of the inventory that cannot be liquidated with the available liquidity. This residual quantity is exposed to the auction shock Y. In the benchmark case, Y is normally distributed with mean zero. This adds an extra risk component when the trader reaches terminal time with too much inventory relative to available liquidity.

2.7 Utility Maximization and Value Function

The trader's objective is to maximize expected utility of terminal wealth. The paper uses exponential utility,

$$U(w) = -\exp(-\alpha w),$$

where $\alpha > 0$ is the risk aversion parameter. The value function is defined by taking the supremum over all admissible trading strategies:

$$v(T, s) = v(T, \lambda, q, p, x) := \sup_{C \in \widetilde{\mathcal{C}}(\lambda, q)} E\left[U(\widetilde{W}^C_{T+})\right].$$

Thus, the value function tells us the best achievable expected utility when the initial state is $s = (\lambda, q, p, x)$. This value function is the central object of the optimal control problem. The paper derives the HJBQVI for this value function.

3 Implementation of the Path Simulator

Compared to the previous semester, the simulator was extended significantly. Previously, the main focus was on generating the uncontrolled market dynamics and studying sample paths of liquidity and price. During this semester, I implemented the controlled part of the model as well. This means that the simulator now includes not only the external market and limit order flow, but also an active trader who can send market orders, react to signals, change inventory, pay transaction costs, and trigger trading halts.

To make the simulator useful for testing different strategies, I implemented a general policy structure. A policy can decide separately about signal-based trades and state-based trades. This made it possible to compare different simple benchmark strategies before using the HJB policy or reinforcement learning.

The simplest benchmark is the no-trade policy. This policy never sends any order, so the trader keeps the initial inventory until terminal liquidation.

The second benchmark is a Time-Weighted Average Price (TWAP) policy. In this implementation, the policy gradually moves the inventory towards a target level, usually zero, over the trading horizon. If the trader has a positive inventory, the TWAP policy sells small amounts over time. This is a simple and commonly used execution rule, but it does not use the signal information in an optimal way.

The third simple strategy is a signal-aware toy policy. This policy uses the signal in a basic heuristic way. For example, when the trader has a positive inventory and receives a signal that liquidity will be taken away, the policy may sell before the external event arrives.

4 HJBQVI Solver and Benchmark Policy

The most important new part of the work this semester was the implementation of the HJBQVI numerical solver. The paper derives a reduced HJB equation for the value function. Because of the exponential utility, the value function has the multiplicative structure

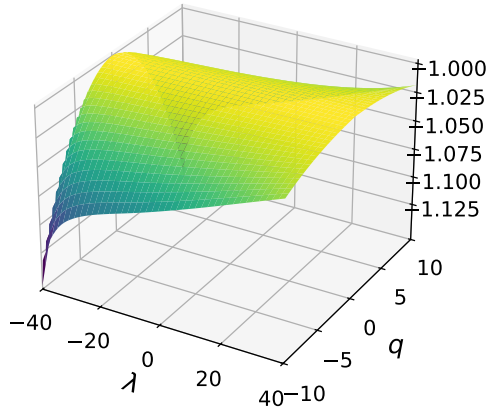
$$v(T, \lambda, q, p, x) = \exp(-\alpha(x + pq))v_0(T, \lambda, q).$$

This reduces the numerical problem to the variables time-to-go, liquidity, and inventory.

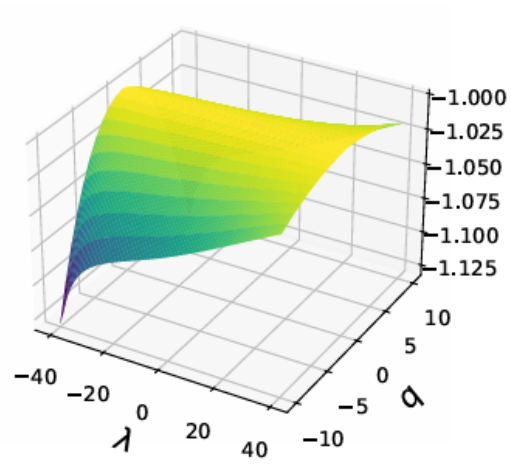
The numerical scheme uses grids for liquidity and inventory. In my implementation, liquidity is discretized on a grid from the halt bucket to the maximum liquidity level, and inventory is discretized on a bounded grid. The action space is also discretized, so trades are multiples of the inventory grid size dq. The code stores not only the value function, but also the optimal state-based trades and the optimal signal-based trades for liquidity-taking and liquidity-providing signals.

After solving the HJBQVI, I wrote a policy adapter called the HJB benchmark policy. This policy takes the current simulator state, converts calendar time into time-to-go, finds the closest grid points in liquidity and inventory, and reads out the optimal action from the stored HJB policy arrays. In this way the HJB solution can be used directly inside the event-driven simulator.

As a qualitative check of the numerical implementation, I also compared the value function surface produced by my HJBQVI solver with the corresponding figure from the paper. The first plot shows my numerical result, while the second plot is the reference figure from the article.



(a) Value function surface produced by my HJBQVI solver.



(b) Reference value function surface from Bank, Cartea and Körber [1].

Figure 1: Comparison of the value function surface for $p = x = 0$. Panel (a) shows the result of my implementation, while panel (b) shows the corresponding figure from the original paper.

5 Monte Carlo Validation of the HJB Solver

After implementing the HJB solver and the policy adapter, I tested whether the HJB value function is consistent with Monte Carlo simulation. If the HJB policy is optimal and the simulator implements the same model, then the expected utility from simulated paths should be close to the HJB value,

$$v(0, \lambda_0, q_0, p_0, x_0) \approx \frac{1}{N} \sum_{i=1}^{N} -\exp(-\alpha W_T^{(i)}),$$

where the paths are simulated using the HJB benchmark policy.

For the initial state[1]

$$\lambda_0 = 0, \qquad q_0 = 8, \qquad p_0 = 100, \qquad x_0 = -800,$$

I ran 5000 Monte Carlo paths. The HJB value and the Monte Carlo mean utility were very close, and the HJB value was inside the 95 percent Monte Carlo confidence interval. This supports that the implemented simulator, the policy adapter, and the HJBQVI solver are consistent with each other.

[1]For numerical readability, I used a shifted initial cash position and set $x_0 = -p_0 q_0$. This removes the large initial mark-to-market value from the utility scale, so the utility values are around -1 instead of being of order 10^{-35}.

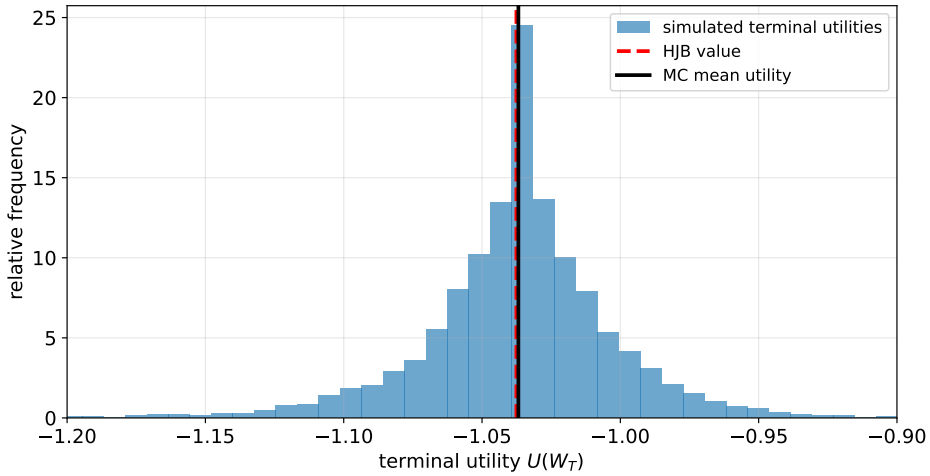


Figure 2: Monte Carlo validation of the HJB benchmark value. The histogram shows the simulated terminal utilities under the HJB benchmark policy, while the vertical lines indicate the HJB value function and the Monte Carlo mean.

As an additional robustness check, I repeated the validation on a grid of initial states with different λ_0 and q_0 values. Out of the 25 tested initial states, the HJB value was inside the 95 percent Monte Carlo confidence interval in 24 cases.

6 Policy Comparisons

After validating the HJB policy, I compared it to simpler policies. The tested policies were the HJB benchmark policy, a no-trade policy, a TWAP policy, and a simple signal-aware toy policy. For each policy, I simulated many paths and calculated the average expected utility. This was then compared to the HJB value. Since the utility is negative, a less negative value is better.

The HJB policy was very close to the HJB value, as expected. The no-trade policy was clearly worse. The TWAP policy and the toy signal policy were closer to the HJB policy, but still below it in expected utility. This showed that the difference between policies is not huge in the benchmark parameter setting, but it is measurable. TWAP is already a reasonable execution strategy, so it is not surprising that it is not extremely far from the HJB policy.

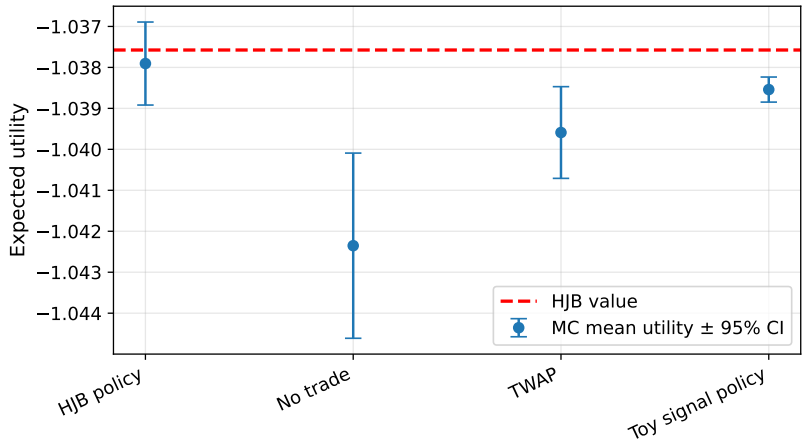


Figure 3: Comparison of different trading policies against the HJB benchmark value. The points show Monte Carlo estimates of expected utility with 95% confidence intervals, while the dashed line represents the HJB benchmark value obtained from the HJBQVI solver.

7 Next Step: Reinforcement Learning

The next step of the project is to use reinforcement learning (RL) to train a learning agent in this environment and compare it to the HJB benchmark. The simulator is already structured in a way that makes this possible. The state of the agent can include time-to-go, liquidity, inventory, price, cash, and the observed signal. The action is the trade size. The reward can be the terminal utility.

There are several possible RL approaches that could be tested in the next semester. One option is a value-based method with a discretized action space, where the agent learns the value of different trade sizes in each state. Another option is a policy-gradient or actor-critic method, where the trading policy is learned directly. In this setting, an actor-critic method such as proximal policy optimization (PPO) seems especially natural, because the state variables are continuous and the action is the chosen trade size. The learned agent could then be compared to the HJB benchmark, the TWAP policy, and the simple signal-aware benchmark.

Code Availability

The Python implementation used in this project, including the simulator, the HJBQVI solver, the benchmark policies, and the plotting scripts, is available in the following GitHub repository:
https://github.com/MolnarAndras2718/Own-projeect---Trade-Signal-model

References

[1] Peter Bank, Álvaro Cartea, and Laura Körber. Optimal execution and speculation with trade signals. *arXiv preprint arXiv:2306.00621*, 2024. https://arxiv.org/abs/2306.00621.