Finite-Horizon Optimal Execution with Discrete Price Bins: Dynamic Programming and Tabular Q-Learning

Florent Ouabo

December 11, 2025

Abstract

We study a discrete-time optimal execution problem in which a trader must liquidate a risky inventory over a finite horizon under uncertain price dynamics. Instead of assuming a parametric diffusion model, the price evolution is constructed directly from historical data: empirical increments are used to calibrate short-term volatility, and the resulting distribution is incorporated either through a discretized Markov chain or via Monte Carlo sampling. Execution generates nonlinear transaction costs, while any remaining inventory at the horizon incurs a forced-liquidation cost and a quadratic terminal penalty.

The problem is formulated as a finite-horizon Markov Decision Process (MDP). We examine three complementary numerical solution methods: (i) *transition-matrix dynamic programming*, which discretizes the price into bins and computes expectations exactly using a Gaussian kernel calibrated from data; (ii) *Monte Carlo dynamic programming*, which preserves the continuous price and approximates conditional expectations through simulation and interpolation; and (iii) *tabular Q-learning*, a fully model-free approach that learns optimal state–action values from simulated trajectories without relying on any transition model.

We present the mathematical formulation, describe the discretization and simulation schemes, and compare the resulting value functions and optimal liquidation policies. The experiments illustrate how model-based methods produce smooth and stable controls, while model-free reinforcement learning recovers the same qualitative structure but with higher variance and slower convergence. The combined analysis provides a unified benchmark for evaluating both classical stochastic control and modern reinforcement learning approaches to optimal execution.

1 Introduction

Optimal execution is a central problem in algorithmic trading and quantitative finance. A trader who holds a large inventory of a risky asset must unwind her position within a prescribed time window while facing uncertain prices and market impact. Liquidating too aggressively generates large trading costs, whereas liquidating too slowly leaves the trader exposed to price risk and, in many settings, regulatory or risk-management constraints on terminal inventory.

Classical models (e.g. Almgren–Chriss) treat optimal execution as a stochastic control problem and solve for continuous-time or discrete-time policies under stylized price dynamics. In practice, however, one often has only a discretized representation of the price process and may be interested in numerical schemes that can either:

- exploit a parametric model for the price transition law (model-based dynamic programming), or
- learn the optimal policy directly from data or simulation (model-free reinforcement learning).

The goal of this paper is to develop a simple yet expressive discrete-time framework that allows us to compare these approaches in a controlled setting. We consider a finite-horizon liquidation problem with discrete inventory and a Gaussian price process calibrated from historical data. The trader trades in discrete time, paying quadratic execution costs and a terminal penalty on any leftover inventory. The problem is formulated as a finite-horizon MDP and solved with:

1. a transition-matrix dynamic program on a grid of price bins;
2. a Monte Carlo dynamic program that approximates expectations in the Bellman recursion by simulation; and
3. a tabular Q-learning algorithm that does not require knowledge of the transition law.

Our numerical experiments show that the three methods produce very similar value functions and policies near the terminal time, and that the model-free Q-learning solution gradually converges toward the model-based benchmark as the number of episodes increases. The setting can be viewed as a toy laboratory for studying reinforcement learning in execution problems, with a clear ground truth.

2 Model Specification

2.1 Time and inventory

We consider a finite time grid

$$t = 0, 1, \ldots, T,$$

where $T \in \mathbb{N}$ is the horizon (in the experiments $T = 20$). The trader starts with inventory $x_0 = x_{\max}$ and trades in integer units. Inventory is modeled as

$$x_t \in \{0, 1, \ldots, x_{\max}\}, \qquad x_{\max} \in \mathbb{N},$$

with deterministic dynamics

$$x_{t+1} = x_t - a_t, \tag{1}$$

where the action a_t represents the number of units sold at time t. We impose the constraint

$$a_t \in \mathcal{A}(x_t) = \{0, 1, \ldots, \min(a_{\max}, x_t)\},$$

so that inventory cannot become negative and the trading rate per period is bounded by $a_{\max}$.

2.2 Price dynamics

Let S_t denote the (mid-)price of the asset at discrete time $t \in \{0, 1, \ldots, T\}$. Over the relatively short horizons relevant for optimal execution, we model the price as an additive Gaussian random walk,

$$S_{t+1} = S_t + \sigma Z_{t+1}, \qquad Z_{t+1} \sim \mathcal{N}(0, 1) \text{ i.i.d.}, \tag{2}$$

where $\sigma > 0$ is the per-step volatility parameter. Conditionally on S_t, this implies

$$S_{t+1} \mid S_t \sim \mathcal{N}(S_t, \sigma^2).$$

This specification captures short-term price uncertainty while deliberately ignoring drift and permanent price impact, so that the optimal execution problem focuses on the interplay between volatility, execution costs, and inventory risk.

Calibration of σ. The volatility σ is calibrated from a historical time series $\left(S_n^{\text{hist}}\right)_{n=0}^{N}$ sampled at the same time scale as the execution horizon. Defining the increments

$$\Delta S_n = S_{n+1}^{\text{hist}} - S_n^{\text{hist}}, \qquad n = 0, \dots, N-1,$$

we estimate σ as the sample standard deviation

$$\widehat{\sigma}^2 = \frac{1}{N-1} \sum_{n=0}^{N-1} \left(\Delta S_n - \overline{\Delta S}\right)^2, \qquad \overline{\Delta S} = \frac{1}{N} \sum_{n=0}^{N-1} \Delta S_n.$$

In the forward model (2) we set the drift to zero and use $\sigma = \widehat{\sigma}$, which is consistent with the empirical observation that short-horizon mean returns are small relative to volatility.

Motivation for the additive Gaussian model. The dynamics (2) can be interpreted as a discrete-time analogue of an arithmetic Brownian motion, but we use it purely as a *local* model for short-horizon price fluctuations. This choice is motivated by:

- *Execution focus:* the objective is to study optimal liquidation under noise-driven price movements, rather than to forecast long-term trends;
- *Numerical tractability:* conditional on S_t, the distribution of S_{t+1} is Gaussian, which allows conditional expectations in the Bellman recursion to be computed either via a discretized transition kernel or via Monte Carlo sampling;
- *Compatibility with discretization:* the additive structure is well suited to the price-binning and interpolation schemes used in our dynamic programming and reinforcement-learning algorithms.

Limitations and extensions. The model (2) omits several real-market features, such as: permanent price impact from the trader's own orders, stochastic volatility, jumps, and microstructure effects (spreads, queues, and hidden liquidity). These ingredients can be incorporated in more elaborate models (for example by adding an impact term or by making σ state dependent), but the present specification provides a clean baseline in which the numerical behavior of transition-matrix dynamic programming, Monte Carlo dynamic programming, and Q-learning can be compared within a unified stochastic control framework.

2.3 Reward structure and objective

When the trader sells a_t units at price S_t, she earns revenue $a_t S_t$ and pays a quadratic execution cost ca_t^2, where $c > 0$. The immediate reward is therefore

$$R_t(x_t, S_t, a_t) = a_t S_t - c a_t^2, \qquad t = 0, \dots, T-1. \tag{3}$$

At the terminal time T any remaining inventory x_T is liquidated at price S_T, incurring the same quadratic cost and an additional penalty λx_T^2 on the residual position:

$$R_T(x_T, S_T) = x_T S_T - c x_T^2 - \lambda x_T^2 = x_T S_T - (c + \lambda) x_T^2, \tag{4}$$

where $\lambda > 0$ controls the strength of the terminal inventory penalty.

We use an undiscounted objective ($\gamma = 1$). Given a policy π mapping states to actions, the performance criterion is

$$J^\pi(x_0, S_0) = \mathbb{E}^\pi\left[\sum_{t=0}^{T-1} R_t(x_t, S_t, a_t) + R_T(x_T, S_T) \;\middle|\; x_0, S_0\right]. \tag{5}$$

The (time-dependent) value function is

$$V_t(x, S) = \sup_\pi \mathbb{E}^\pi\left[\sum_{u=t}^{T-1} R_u(x_u, S_u, a_u) + R_T(x_T, S_T) \;\middle|\; x_t = x,\, S_t = S\right].$$

3 Transition-Matrix Dynamic Programming

We first approximate the price process by a finite-state Markov chain on a grid of price bins and solve the MDP by backward dynamic programming on the resulting discrete state space.

3.1 Price discretization and Markov chain

Let S_{init} be a reference level (e.g. the initial price). We construct an interval

$$[S_{\min}, S_{\max}] = \left[S_{\text{init}} - 3\sigma\sqrt{T}, S_{\text{init}} + 3\sigma\sqrt{T}\right],$$

which captures most of the mass of the Gaussian price distribution over the horizon. This interval is partitioned into N_S bins via a grid

$$S_{\min} = S^{(0)} < S^{(1)} < \cdots < S^{(N_S)} = S_{\max}.$$

We take the midpoint

$$\tilde{S}^{(s)} = \frac{S^{(s-1)} + S^{(s)}}{2}, \qquad s = 1, \ldots, N_S,$$

as the representative price in bin s.

Conditional on S_t lying in bin s, we approximate S_t by $\tilde{S}^{(s)}$ and define the one-step transition probabilities

$$P_{s,s'} = \mathbb{P}\big(S_{t+1} \in (S^{(s'-1)}, S^{(s')}) \,\big|\, S_t = \tilde{S}^{(s)}\big) = \Phi\left(\frac{S^{(s')} - \tilde{S}^{(s)}}{\sigma}\right) - \Phi\left(\frac{S^{(s'-1)} - \tilde{S}^{(s)}}{\sigma}\right), \tag{6}$$

where Φ is the standard normal cdf. Each row is renormalized so that $\sum_{s'} P_{s,s'} = 1$. The matrix $P \in \mathbb{R}^{N_S \times N_S}$ thus defines a discrete-time Markov chain for the price bin index.

3.2 State, actions, and Bellman recursion

The MDP has:

- **State:** (x, s) with $x \in \{0, 1, \ldots, x_{\max}\}$ (inventory) and $s \in \{1, \ldots, N_S\}$ (price bin index).
- **Action:** $a \in \mathcal{A}(x) = \{0, 1, \ldots, \min(a_{\max}, x)\}$.
- **Transition:** given (x, s) and a, the next inventory is $x' = x - a$ and the next price bin s' is drawn according to the row $P_{s,\cdot}$ of the transition matrix.

Let $V_t(x, s)$ denote the value function at time t for inventory x and price bin s. For $t = T-1, T-2, \ldots, 0$ the Bellman recursion reads

$$\begin{aligned} V_t(x, s) &= \max_{a \in \mathcal{A}(x)} \Big\{ a\tilde{S}^{(s)} - ca^2 + \mathbb{E}\big[V_{t+1}(x - a, s_{t+1})\big] \Big\} \\ &= \max_{a \in \mathcal{A}(x)} \Big\{ a\tilde{S}^{(s)} - ca^2 + \sum_{s'=1}^{N_S} P_{s,s'} V_{t+1}(x - a, s') \Big\}, \end{aligned} \tag{7}$$

as long as $t + 1 < T$ and $x - a > 0$. If $x - a = 0$, there is no future reward and the continuation term vanishes.

At the last decision time $t = T - 1$, the next step is T, where the only contribution is the terminal reward. In that case

$$V_{T-1}(x, s) = \max_{a \in \mathcal{A}(x)} \Big\{ a\tilde{S}^{(s)} - ca^2 + \mathbb{E}\big[R_T(x - a, S_T)\big] \Big\}, \tag{8}$$

with

$$\mathbb{E}\big[R_T(x-a,S_T)\big] = \sum_{s'=1}^{N_S} P_{s,s'}\Big((x-a)\tilde{S}^{(s')} - (c+\lambda)(x-a)^2\Big). \tag{9}$$

We use $V_T(x,s) = 0$ as terminal condition, since R_T is included explicitly at $t = T-1$. The optimal feedback control $a_t^*(x,s)$ is the maximizer in (7) or (8).

3.3 Numerical algorithm

We store the value function and policy in 3D arrays

$$V[t,x,s], \quad A[t,x,s],$$

with $t = 0,\dots,T-1$, $x = 0,\dots,x_{\max}$, $s = 1,\dots,N_S$. The algorithm is:

1. Calibrate σ from data; build the price grid, midpoints $\tilde{S}^{(s)}$, and the transition matrix P.
2. Set $V_T(x,s) = 0$ for all x, s.
3. For $t = T-1$ down to 0:
   (a) For each (x,s):
      i. Enumerate feasible actions $a \in \mathcal{A}(x)$.
      ii. For each a, compute the immediate reward $a\tilde{S}^{(s)} - ca^2$.
      iii. If $t+1 = T$, compute the expected terminal reward via (9); if $x - a = 0$, set the continuation term to zero; otherwise compute $\sum_{s'} P_{s,s'} V_{t+1}(x-a,s')$.
      iv. Sum the immediate reward and continuation value to obtain $Q_t(x,s,a)$.
      v. Choose $a^* = \arg\max_a Q_t(x,s,a)$ and set $V_t(x,s) = Q_t(x,s,a^*)$, $A_t(x,s) = a^*$.

The cost is $\mathcal{O}(T\, x_{\max}\, N_S\, a_{\max})$, which is manageable for moderate grids.

4 Numerical Results: Transition-Matrix DP

We now illustrate the structure of the value function and the optimal policy under the transition-matrix scheme with the parameter set

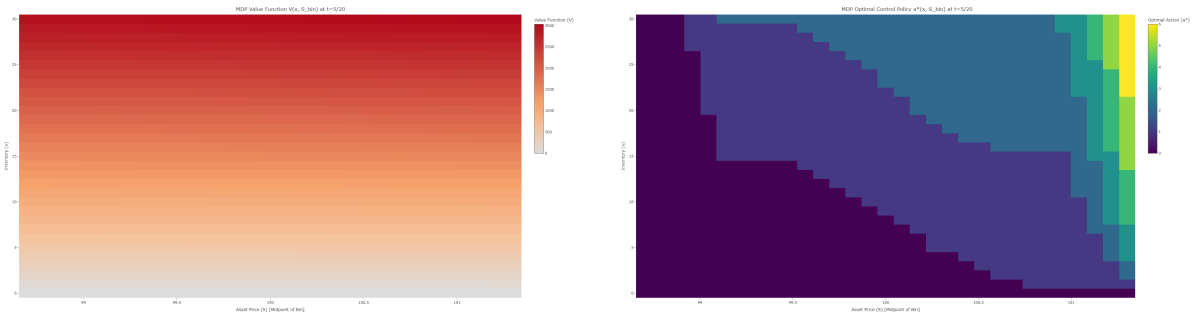
$$T = 20, \quad x_{\max} = 30, \quad a_{\max} = 10, \quad c = 0.02, \quad \lambda = 0.5.$$

The volatility σ is calibrated from a simulated random walk, and we use $N_S = 31$ price bins centered around $S_{\text{init}} = 100$.

Figures 1–3 show the value function and the optimal action at three different times. In each figure, the left panel displays $V_t(x,S)$ as a function of inventory x and price S (bin midpoint), while the right panel shows the corresponding optimal action $a_t^*(x,S)$.

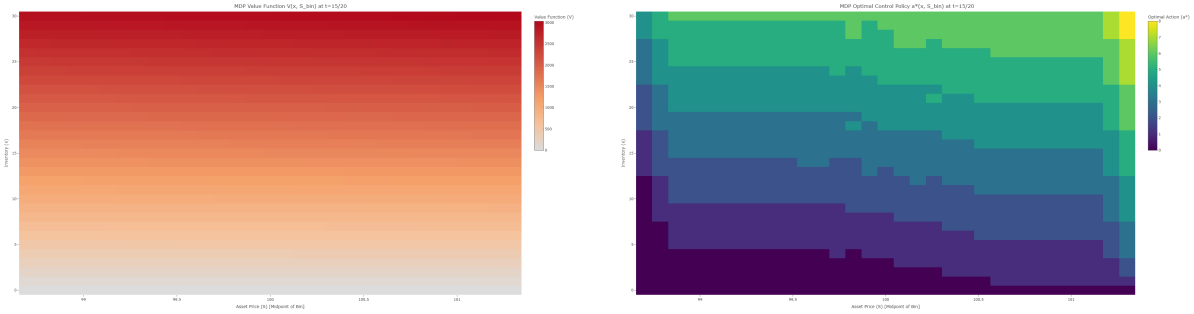
4.1 Intermediate horizon: $t = 5$

At this early stage, the value function is almost linear in inventory and monotone increasing: larger positions have higher expected liquidation value. Dependence on price is weak, because there is still ample time to respond to future price moves. The optimal policy is relatively conservative: for low inventories the agent often chooses not to trade ($a_t^* = 0$), while higher inventories and higher prices trigger positive selling rates.



(a) Value function $V_5(x, S)$ (b) Optimal policy $a_5^*(x, S)$

Figure 1: Transition-matrix DP: value function and optimal control at $t = 5/20$.



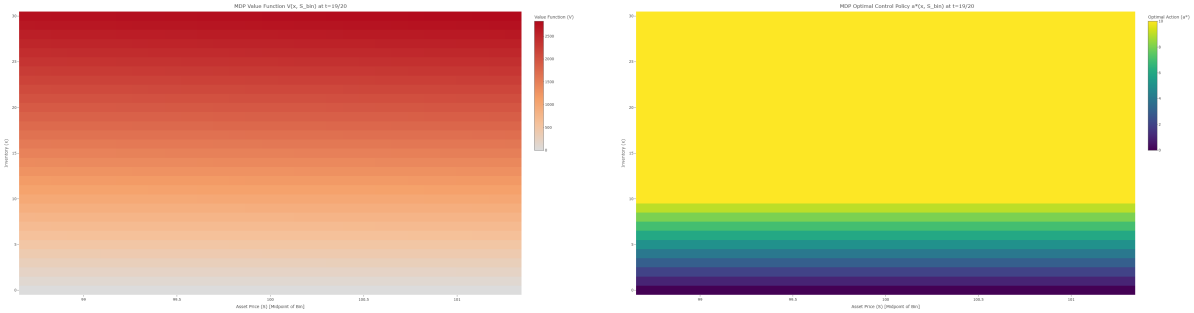
(a) Value function $V_{15}(x, S)$ (b) Optimal policy $a_{15}^*(x, S)$

Figure 2: Transition-matrix DP: value function and optimal control at $t = 15/20$.

4.2 Later in the horizon: $t = 15$

By $t = 15$, there is less time remaining. Inventory risk and the terminal penalty become more important, and the optimal policy is substantially more aggressive: for medium and high inventories, the trader sells several units per step, with only mild sensitivity to the current price.

4.3 Final decision time: $t = 19$



(a) Value function $V_{19}(x, S)$ (b) Optimal policy $a_{19}^*(x, S)$

Figure 3: Transition-matrix DP: value function and optimal control at $t = 19/20$.

At the last decision time, the policy is essentially bang–bang in inventory: for $x \geq a_{\max}$ the trader sells at the maximum rate $a_{\max}$, while for small inventories she sells everything. Price plays almost no role, because there is no opportunity to wait for favorable moves.

5 Monte Carlo Dynamic Programming

The transition-matrix approach requires explicit construction of the matrix P in (6). In more complex models this may be difficult. We therefore consider an alternative scheme that keeps the price continuous and approximates the conditional expectations in the Bellman equation by Monte Carlo sampling.

5.1 Continuous price and Bellman equation

We retain the inventory dynamics (1) and the price process (2), but now treat S_t as continuous. The Bellman equation becomes

$$V_t(x,S) = \max_{a\in\mathcal{A}(x)} \Big\{ aS - ca^2 + \mathbb{E}\big[V_{t+1}(x-a, S+\sigma Z)\big] \Big\}, \qquad t = 0,\dots,T-1, \tag{10}$$

with $Z \sim \mathcal{N}(0,1)$ and terminal condition $V_T(x,S) = 0$. At $t = T-1$ the terminal reward R_T is added explicitly as in the transition-matrix case. When $x - a = 0$, the continuation term vanishes.

The main difficulty is that the integral

$$\mathbb{E}\big[V_{t+1}(x-a, S+\sigma Z)\big] = \int_{\mathbb{R}} V_{t+1}(x-a, S+\sigma z)\,\varphi(z)\,dz$$

cannot be computed analytically, because V_{t+1} is only available at discrete grid points. We handle this via a combination of interpolation and Monte Carlo sampling.

5.2 Price grid and interpolation

We introduce a price grid

$$S^{(1)} < S^{(2)} < \cdots < S^{(J)}$$

covering a range $[S_{\min}, S_{\max}]$ (e.g. $\pm 3\sigma\sqrt{T}$ around S_{init}). For each inventory level $x_i \in \{0, 1, \dots, x_{\max}\}$ we store $V_t(x_i, S^{(j)})$ at each grid point.

For $S \in [S_{\min}, S_{\max}]$ we define the linear interpolation operator $\mathcal{I}_S[V_t](x,S)$ as follows: let j be such that $S^{(j)} \le S \le S^{(j+1)}$, and set

$$\mathcal{I}_S[V_t](x,S) = \lambda V_t(x, S^{(j)}) + (1-\lambda) V_t(x, S^{(j+1)}), \qquad \lambda = \frac{S^{(j+1)} - S}{S^{(j+1)} - S^{(j)}}. \tag{11}$$

For S outside the grid we clamp to the boundary: $\mathcal{I}_S[V_t](x,S) = V_t(x, S^{(1)})$ if $S < S^{(1)}$, and $\mathcal{I}_S[V_t](x,S) = V_t(x, S^{(J)})$ if $S > S^{(J)}$.

5.3 Monte Carlo approximation of the continuation value

Fix (t,x,S) and $a \in \mathcal{A}(x)$. To approximate $\mathbb{E}[V_{t+1}(x-a, S+\sigma Z)]$ we sample $Z^{(1)}, \dots, Z^{(M)} \sim \mathcal{N}(0,1)$ and define $S^{(m)}_{\text{next}} = S + \sigma Z^{(m)}$. We then compute

$$\widehat{V}^{(m)}_{t+1}(x-a, S) = \mathcal{I}_S[V_{t+1}]\big(x-a, S^{(m)}_{\text{next}}\big),$$

and set

$$\widehat{\mathbb{E}}\big[V_{t+1}(x-a, S+\sigma Z)\big] = \frac{1}{M}\sum_{m=1}^{M} \widehat{V}^{(m)}_{t+1}(x-a, S). \tag{12}$$

Substituting into (10) yields the Monte Carlo estimate

$$\widehat{Q}_t(x,S,a) = aS - ca^2 + \frac{1}{M}\sum_{m=1}^{M} \mathcal{I}_S[V_{t+1}]\big(x-a, S+\sigma Z^{(m)}\big), \tag{13}$$

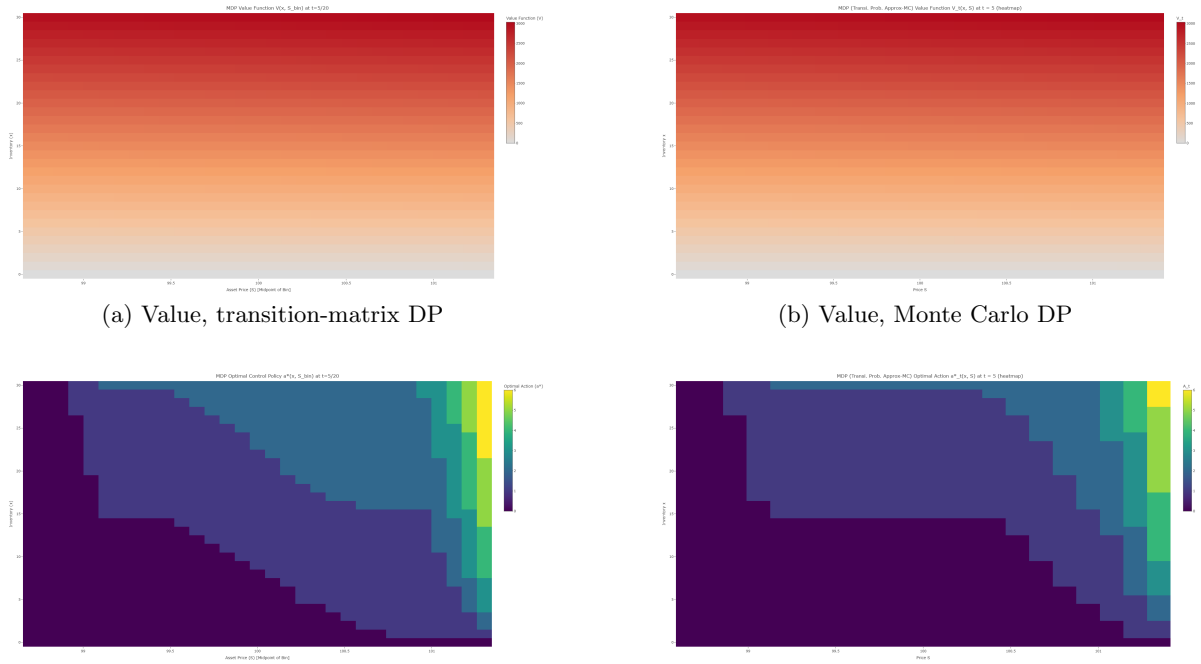
and

$$\widehat{V}_t(x,S) = \max_{a\in\mathcal{A}(x)} \widehat{Q}_t(x,S,a). \tag{14}$$

5.4 Algorithm and comparison with transition-matrix DP

The Monte Carlo DP algorithm proceeds backward in time, reusing the same Gaussian samples $Z^{(m)}$ across states at a given t to reduce variance. In the experiments we use $M = 2000$.

To compare with the transition-matrix DP, we compute value functions and policies on the same price grid and inventory grid. Figures 4–6 show side–by–side comparisons for $t/T \in \{5/20, 15/20, 19/20\}$.



(a) Value, transition-matrix DP

(b) Value, Monte Carlo DP

(c) Policy, transition-matrix DP

(d) Policy, Monte Carlo DP

Figure 4: Comparison of value function and optimal control at $t = 5/20$ between transition-matrix DP and Monte Carlo DP.

The value surfaces are nearly indistinguishable across methods and exhibit the same monotonicity in inventory and weak dependence on price, especially near the horizon. The optimal policies also agree qualitatively: a wedge-shaped trading region at earlier times, increasingly aggressive liquidation as t approaches T, and a bang–bang structure at $t = 19$. Differences are confined to the grid edges and are attributable to Monte Carlo noise and interpolation.

6 Tabular Q-Learning for Stochastic Execution Control

We now turn to a model-free approach based on tabular Q-learning. The key motivation is to study how a reinforcement-learning agent can recover the optimal execution policy when it does not know the transition kernel of the price process and can only interact with a simulator.

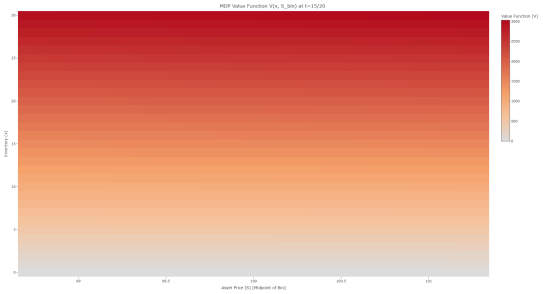
6.1 State, action, and reward

We reuse the model of Section 2. At each decision time $t \in \{0, \dots, T-1\}$ the state is

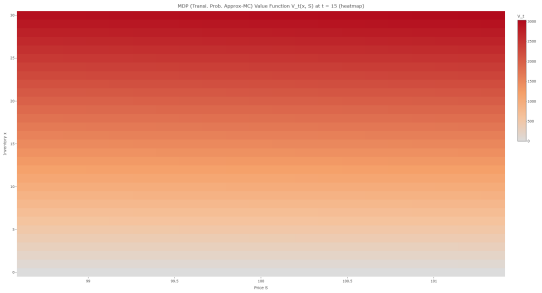
$$s_t = (t, x_t, S_t).$$

Time is included explicitly so that the problem remains finite-horizon. The action set is

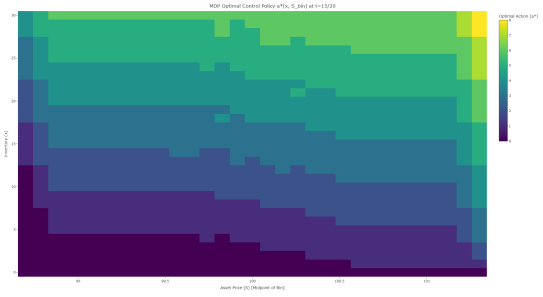
$$a_t \in \mathcal{A}(x_t) = \{0, 1, \dots, \min(a_{\max}, x_t)\},$$



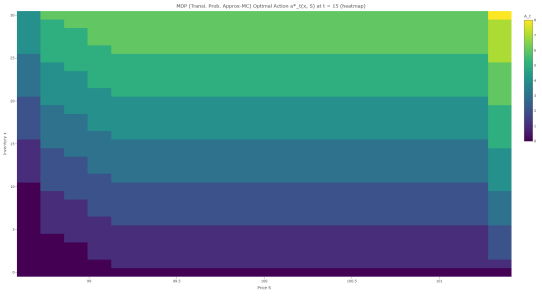
(a) Value, transition-matrix DP



(b) Value, Monte Carlo DP

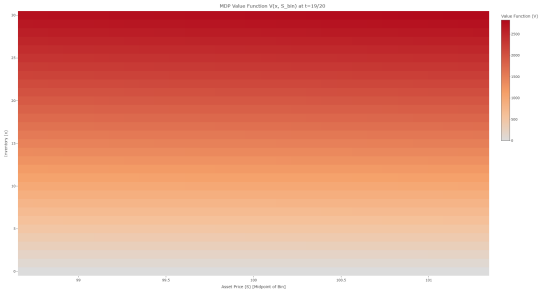


(c) Policy, transition-matrix DP

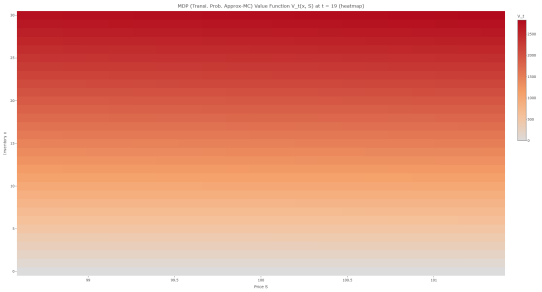


(d) Policy, Monte Carlo DP

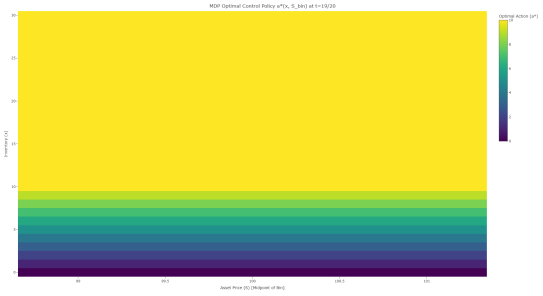
Figure 5: Comparison of value function and optimal control at $t = 15/20$ between transition-matrix DP and Monte Carlo DP.



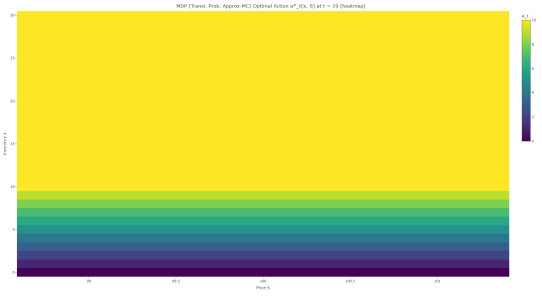
(a) Value, transition-matrix DP



(b) Value, Monte Carlo DP



(c) Policy, transition-matrix DP



(d) Policy, Monte Carlo DP

Figure 6: Comparison of value function and optimal control at $t = 19/20$ between transition-matrix DP and Monte Carlo DP.

and the immediate reward is $R_t(x_t, S_t, a_t)$ given by (3). At $t = T$ the environment returns the terminal reward (4) (through a final step in the simulator).

6.2 Discretization of the price dimension

To maintain a tabular representation, we discretize the price into the same N_S bins as in Section 3. We thus work with a finite state space

$$(t, x, s), \qquad t = 0, \dots, T-1,\ x = 0, \dots, x_{\max},\ s = 1, \dots, N_S,$$

where s is the index of the price bin containing S_t. The Q-function is stored as a four-dimensional tensor

$$Q[t, x, s, a],$$

indexed by time, inventory, price bin, and action ($a = 0, \dots, a_{\max}$).

6.3 Q-learning update

The optimal action-value function Q^* satisfies the Bellman optimality equation

$$Q^{*(t,x,S,a)} = R_t(x, S, a) + \mathbb{E}\left[\max_{a'} Q^{*(t+1,x',S',a')}\right],$$

with $x' = x - a$ and S' distributed according to the (unknown) transition law. Because the horizon is finite and time is part of the state, we take $\gamma = 1$.

Given a simulated transition $(t, x, S, a, r, t', x', S')$ with discretized bins s, s', the Q-learning update is

$$Q_{k+1}(t, x, s, a) = Q_k(t, x, s, a) + \alpha\Big(r + \max_{a'} Q_k(t', x', s', a') - Q_k(t, x, s, a)\Big),$$

where $\alpha > 0$ is the learning rate. Action selection follows an ε-greedy rule: with probability ε_k the agent chooses a random feasible action; otherwise it chooses the action that maximizes the current Q_k. The exploration parameter decays as

$$\varepsilon_{k+1} = \max(\varepsilon_{\min}, \varepsilon_k \rho), \qquad \rho < 1.$$

The environment simulator implements the inventory dynamics $x_{t+1} = x_t - a_t$, the price update $S_{t+1} = S_t + \sigma Z_{t+1}$ and the terminal liquidation rule. Episodes terminate either when t reaches T or when $x_t = 0$.

6.4 Behaviour of the learned policy

Figures 7–9 show the Q-learning policy and the associated value estimate at $t = 5, 15, 19$ after training over 500,000 episodes on the same parameter set as before.

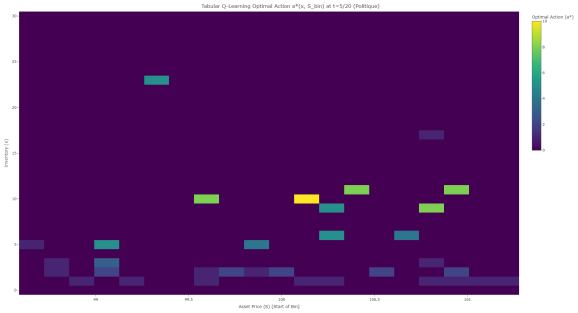
At $t = 5$ the learned policy is noisy and sparse. Many state–action pairs are rarely visited, and no structural prior (such as smoothness in price or inventory) is enforced. The resulting action map exhibits isolated trading regions and a value surface with noticeable irregularities.

By $t = 15$ the policy has become much more structured. High inventory levels lead to more aggressive liquidation, while low inventories tend to wait ($a = 0$). Price sensitivity is weaker than in the MDP benchmark, reflecting the fact that Q-learning only observes stochastic transitions and does not exploit the exact Gaussian kernel.

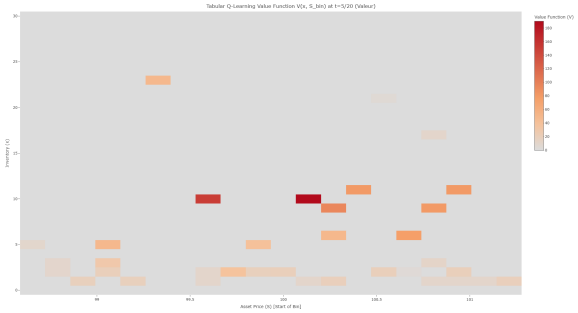
Near the terminal time, Q-learning and the MDP methods agree on the basic structure:

$$a^{*(t,x,S)} \approx x \wedge a_{\max},$$

that is, sell the maximum rate until inventory falls below $a_{\max}$ and then liquidate the remainder. The learned value function is almost linear in inventory and largely independent of price. The main differences with the MDP benchmark are residual irregularities in regions that are visited infrequently during training.

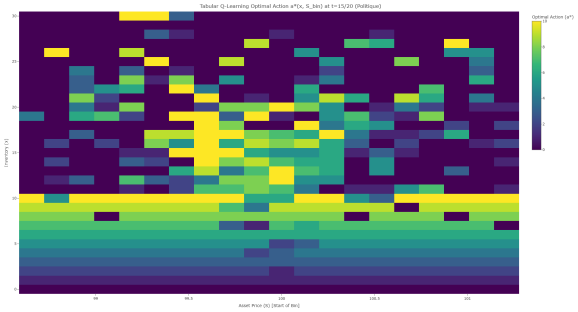


(a) Q-learning policy $a^{*(x,S)}$ at $t = 5/20$

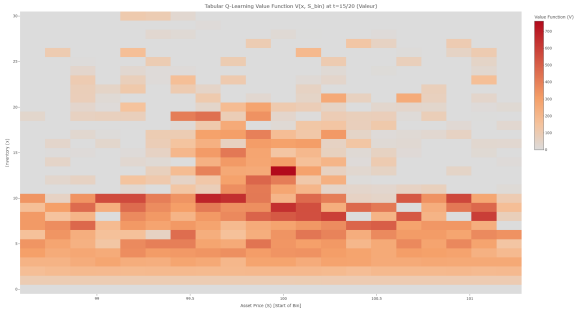


(b) Q-learning value $V(x, S)$ at $t = 5/20$

Figure 7: Tabular Q-learning results at $t = 5/20$.

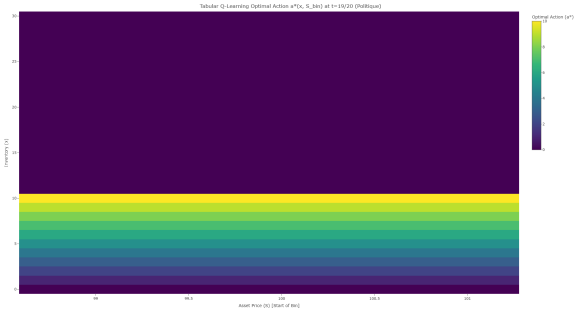


(a) Q-learning policy $a^{*(x,S)}$ at $t = 15/20$

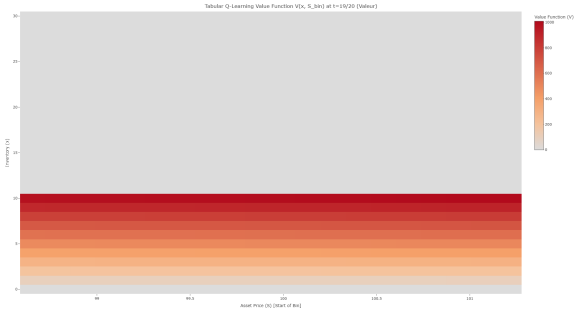


(b) Q-learning value $V(x, S)$ at $t = 15/20$

Figure 8: Tabular Q-learning results at $t = 15/20$.



(a) Q-learning policy $a^{*(x,S)}$ at $t = 19/20$



(b) Q-learning value $V(x, S)$ at $t = 19/20$

Figure 9: Tabular Q-learning results at $t = 19/20$.

6.5 Comparison with model-based methods

Table 1 summarizes the three numerical methods.

Method	Transition knowledge	Expectation	Smoothness	Convergence speed
Transition-matrix DP	Full analytic (Gaussian)	Exact integration	Very smooth	Fast
Monte Carlo DP	Parametric (Gaussian)	MC quadrature	Smooth (up to MC noise)	Medium
Tabular Q-learning	None (model-free)	Sample-based	High variance	Slow

Table 1: Comparison of the three solution methods.

Q-learning roughly recovers the order of magnitude and qualitative structure of the MDP value function, but requires many episodes, and the estimates remain noisy, especially away from frequently visited states. This highlights the trade-off between model-based and model-free methods: knowing the transition law allows for fast and smooth dynamic programming, whereas model-free learning is more flexible but statistically less efficient.

7 Neural Approximation of the Optimal Execution Policy

The dynamic–programming methods developed in the previous sections deliver a high–quality optimal policy $a_t^*(x, S)$ and value function $V_t(x, S)$ on a discrete grid of time, inventory and price. However, the resulting control is tabular: it is defined only on the grid points, requires interpolation for off–grid states, and can become expensive to update if the model or grid are refined.

In this section we construct a *neural network approximation* of the optimal policy. The idea is to treat the dynamic–programming solution as an *expert* (teacher) and train a neural network (student) to imitate its decisions. This is a form of *policy distillation* or *imitation learning*. Once trained, the network provides: (i) fast evaluation of the optimal action for any continuous state (t, x, S), (ii) smooth generalization between grid points, and (iii) a convenient parametrization that can be further fine–tuned with reinforcement learning if the market model changes.

7.1 State Representation

We represent each state $s_t = (t, x_t, S_t)$ by a normalized feature vector $\boldsymbol{s}_t \in \mathbb{R}^3$,

$$\boldsymbol{s}_t = \begin{bmatrix} t/T \\ x_t/x_{\max} \\ (S_t - S_{\text{init}})/(\sigma\sqrt{T}) \end{bmatrix}, \tag{15}$$

where T is the horizon, $x_{\max}$ is the maximum inventory, S_{init} is a reference price (for instance the initial price), and σ is the per–step volatility. This scaling brings all inputs into a comparable range and encodes the economic structure: time–to–maturity, inventory level, and price in volatility units.

Additional features (e.g. realized volatility, bid–ask spread, or volume) can be appended to (15) without changing the framework.

7.2 Supervised Distillation from the MDP Solution

Let $a_t^*(x, S)$ denote the optimal control obtained by dynamic programming for the discretized model. We sample a collection of states on the DP grid and possibly from simulated trajectories under the optimal policy. For each sampled state (t, x, S) we form the feature vector $\boldsymbol{s}_t$ in (15) and the associated target action

$$y_t = a_t^*(x, S) \in \{0, 1, \ldots, a_{\max}\}.$$

This yields a supervised dataset

$$\mathcal{D} = \{(\boldsymbol{s}_i, y_i)\}_{i=1}^{N}$$

on which we train a neural network to approximate the mapping $\boldsymbol{s} \mapsto a^*$.

Two related formulations are natural.

Policy network (classification). We parameterize a stochastic policy $\pi_\theta(a \mid \boldsymbol{s})$ by a neural network f_θ with a softmax output over discrete actions,

$$\pi_\theta(a \mid \boldsymbol{s}) = \mathrm{softmax}_a\big(f_\theta(\boldsymbol{s})\big), \qquad a \in \{0, 1, \ldots, a_{\max}\}. \tag{16}$$

The training objective is the cross–entropy loss

$$\mathcal{L}_{\mathrm{pol}}(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \pi_\theta\big(y_i \mid \boldsymbol{s}_i\big) + \lambda_{\mathrm{reg}}\|\theta\|_2^2, \tag{17}$$

where $\lambda_{\mathrm{reg}} \geq 0$ controls ℓ_2 regularization. At run time the recommended trade size is

$$a_t^{\mathrm{NN}}(\boldsymbol{s}_t) = \arg\max_{a \leq x_t} \pi_\theta(a \mid \boldsymbol{s}_t),$$

with the additional constraint $a \leq x_t$ to preserve feasibility.

Q–network (regression). Alternatively, we may approximate the state–action value function $Q_t^*(x, S, a)$. For each grid state (t, x, S) and each discrete action $a \in \{0, \ldots, a_{\max}\}$, we compute the dynamic–programming value $Q_t^*(x, S, a)$ and train a neural network $Q_\theta(\boldsymbol{s}, a)$ to minimize the squared error

$$\mathcal{L}_{\mathrm{Q}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{a=0}^{a_{\max}} \Big(Q_\theta(\boldsymbol{s}_i, a) - Q_i^*(a)\Big)^2 + \lambda_{\mathrm{reg}}\|\theta\|_2^2, \tag{18}$$

where $Q_i^*(a)$ denotes the tabular value at the corresponding grid point. The induced policy is then

$$a_t^{\mathrm{NN}}(\boldsymbol{s}_t) = \arg\max_{a \leq x_t} Q_\theta(\boldsymbol{s}_t, a).$$

For the present execution problem, the policy–network formulation (16)–(17) is usually sufficient and simpler to implement.

7.3 Network Architecture and Training

Because the state dimension is small, a standard multilayer perceptron (MLP) provides an adequate function class. A typical architecture is:

- Input layer of dimension d (here $d = 3$),
- Two or three hidden layers with 64–128 neurons each and ReLU activation,
- Output layer with $a_{\max} + 1$ units; softmax for the policy network, or linear outputs for the Q–network.

The parameters θ are optimized by stochastic gradient descent, e.g. using the Adam optimizer, on mini–batches drawn from $\mathcal{D}$. A validation set is held out to monitor overfitting and tune hyperparameters such as network width, learning rate and regularization. A practical performance metric is the *policy accuracy*,

$$\mathrm{Acc} = \frac{1}{N_{\mathrm{val}}}\sum_{(\boldsymbol{s},y)\in\mathcal{D}_{\mathrm{val}}} \mathbf{1}\{a^{\mathrm{NN}}(\boldsymbol{s}) = y\},$$

measuring the fraction of validation states on which the neural network reproduces the optimal DP action.

8 Conclusion

We have formulated a finite-horizon optimal execution problem as a discrete-time Markov Decision Process on a joint grid of inventory and price states, where the price follows an additive Gaussian process calibrated from historical data. Within this setting, we implemented and compared three numerical solution methods: (i) transition-matrix dynamic programming based on a discretized Markov chain approximation of the price; (ii) Monte Carlo dynamic programming, which retains a continuous price representation and approximates conditional expectations through simulation and interpolation; and (iii) tabular Q-learning, a model-free reinforcement learning algorithm that acquires optimal state–action values purely from sampled trajectories.

The two model-based approaches produced nearly identical value functions and optimal control surfaces, underscoring the numerical robustness of the discretization and Monte Carlo quadrature schemes. The tabular Q-learning agent, although slower to converge and subject to sampling noise, progressively recovered the core structural features of the optimal policy particularly near terminal time, where the liquidation rule becomes steep and nearly deterministic. These observations confirm that the execution environment provides a controlled and analytically tractable testbed for probing the behavior of reinforcement learning algorithms under known optimal benchmarks.

Beyond numerical comparison, this framework illustrates the conceptual continuum between stochastic control and reinforcement learning: dynamic programming relies on fully specified transition operators, Monte Carlo DP queries the transition via simulation, and Q-learning dispenses with the transition model entirely. This perspective clarifies where each method succeeds or fails as the dimensionality of the problem increases or market microstructure is enriched.

Future research directions naturally follow from this unified viewpoint. From a modelling perspective, one may incorporate permanent and transient price impact, stochastic volatility, regime-switching dynamics, or partial-information settings where drift and liquidity conditions must be filtered. On the computational side, a central avenue for extension is the *neural approximation of the optimal execution policy*: using deep networks to approximate either the value function, the Q-function, or the policy itself. Such parametrizations via deep Q-networks, fitted value iteration, actor–critic architectures, or supervised learning on synthetic MDP solutions offer a scalable path to handling high-dimensional state spaces and realistic market features. Moreover, neural models trained on a combination of simulated and historical data could serve as adaptive execution engines capable of generalizing to unseen market conditions.

Overall, the combination of classical dynamic programming, Monte Carlo methods, and reinforcement learning provides a versatile and extensible computational toolkit. It opens the door to increasingly realistic execution models and to the development of data-driven trading algorithms grounded in rigorous stochastic control principles.

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[2] Á. Cartea, S. Jaimungal, and J. Penalva. *Algorithmic and High-Frequency Trading*. Cambridge University Press, 2015.