An Optimal Liquidation Algorithm Based on Monte Carlo Method



Dawei CAI

Linacre College

University of Oxford

A thesis submitted for the degree of

MSc of Mathematical Computational Finance

Trinity 2011

Acknowledgements

I would like to thank my supervisor Dr. Lajos Gergely Gyurko for all the assistance, support and guidance that he provided in the duration of this dissertation, which enabled me to develop an understanding of this subject.

Contents

1 Monte Carlo Algorithm Framework **3**

1.1 Financial Market Setup 3

1.2 Problem Formation 3

1.3 Numerical Approach Algorithm 1 5

1.3.1 Dynamic Programming Formulation 5

1.3.2 Conditional Expectation Approximation 5

1.3.3 Regression Coefficients Estimation 8

1.3.3.1 MC estimation of $B_{V,\psi}$ and B_{ψ} 8

1.3.3.2 $\mathcal{G}_t^{xy}$ Implementation 8

1.3.4 The Plain Algorithm of LSR-based MC estimation 11

1.4 Price Impact Model 12

1.4.1 Price Dynamics 12

1.4.2 Temporary Price Impact 13

1.4.3 Permanent Price Impact 15

1.5 Numerical Algorithm Approach 2 16

2 Numerical Algorithm Implementation and Results **19**

2.1 Basic Test and Observation 19

2.1.1 Test of Regression Component 20

2.1.2 Test of the Plain Algorithm Implementation 21

2.2 Artificial Market Set-up and Implementation 22

2.2.1 Weekly Execution 23

2.2.2 Daily Execution 23

2.2.3 Comment on Scenario Results 25

2.3 Optimal Execution Strategy 27

2.4 Further Manipulation and Improvement 29

2.4.1 Goal Function 29

2.4.2 Basis Function 29

2.4.3 A Low-biased estimate 30
2.4.4 Calibration to Real Market 30

3 Conclusion and Final Remark **32**

References **33**

A Appendix MATLAB Code **35**

List of Figures

1.1 $\mathcal{G}_t^{xy} = \mathcal{G}_t^x \times \mathcal{G}_t^y$ (10000 points in total, 20 y-items per each x item) . . 10
1.2 $\mathcal{G}_t^{xy}$ grid, y points generated rank-1 lattice rule (4000 points in total, 8 y-item per each x item) 10
1.3 Price Impact Sketch as Jump from Trajectories 17
1.4 Conditional Expectation Illustration 18

2.1 Comparison of Effective Payoff with $\sigma = 0.2$ and $\sigma = 0.02$, with $X_0 = 42.5$ 22
2.2 Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid 23
2.3 Optimal Effective Payoff at Different Time Step with $X_0 = 70$ 24
2.4 Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid 25
2.5 Optimal Effective Payoff at Different Time Step with $X_0 = 70$ 26
2.6 Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid 26
2.7 Weekly Trading Strategy Simulation 28
2.8 Daily Trading Strategy Simulation 28
2.9 Difference Between LSR-based MC estimation Estimate and Low-biased Estimate 30

List of Tables

2.1 Test of regression coefficient estimation, MC expectation, 100 samples 21
2.2 Test of regression coefficient estimation, MC variance, 100 samples . 21
2.3 Parameters Specification 21
2.4 Parameter Specification: Artificial Market 1 24
2.5 Parameter Specification: Artificial Market 2 25

Introduction

Algorithmic trade execution has become a standard technique for institutional market players in recent years, particularly in the equity market where electronic trading is most prevalent. A trade execution scheme typically seeks to execute a trade decision while optimally fulfilling specific goal function upon receiving inputs from a human trader.

Transaction costs are widely recognized as a large determinant of execution scheme performance. For large trades, the execution decision itself has tremendous impact on the market, causing deviation from the desired execution price. Compared with explicitly costs such as commissions and fees, these implicit costs are more interesting and play a critical role in improving the execution scheme.

A common form of optimality goal function seeks to strike a balance between minimizing pricing impact and minimizing timing risk. Several approaches have been suggested to quantify the minimization of pricing impact and timing risk. The first and most intuitive approach is to maximize the expected revenues while satisfying a risk criterion, for example, variance, quadratic variation or value-at-risk. The second approach maximizes the expected value of a utility function of revenues, for example, a power-law function or an exponential function. The third approach seeks to minimize the expected execution cost.

To model this problem, while a large body of literature focusing on solving a dynamic programming problem using trees or the associated Hamilton-Jacobi-Bellman equation with stochastic control and PDE techniques, we concentrate on developing Monte Carlo techniques to provide numerical solution for a particular dynamic programming problem.

In this paper, we concentrate on an algorithm based on Monte Carlo techniques to present a solution for optimal liquidation problem. We first conclude our problem into a dynamic programming framework. Then with the assistance of regression we derive a plain algorithm. Next we introduce a price impact function and incorporate it into our algorithm. As the construction of the algorithm is presented, we focus on

assessing and interpreting some basic test results and its performance under artificial markets. We estimate the optimal goal function value at initial time and simulate the corresponding optimal trade execution strategy. Finally we make some final remarks and discuss further interesting extensions, such as calibration to a real market.
The paper is structured as follows, chapter 1 starts with problem formulation, then it goes through the dynamic programming, conditional expectation, regression method and finally present a plain algorithm. The rest of this chapter focuses on price impact and how to incorporate it into our algorithm. At the end of chapter 1 we derive the final algorithm. Chapter 2 starts with general tests of numerical implementation. Then we set up artificial markets with a parameter setting recommended from some references to evaluate performance of our algorithm and try to interpret it. After that we focus on how to create optimal trade execution strategy given a market setup and regression coefficients we have derived earlier. We finish chapter 2 with some supplement explanation of interesting issues and further discussion over calibration to a real market.

Chapter 1

Monte Carlo Algorithm Framework

1.1 Financial Market Setup

The general setup includes a finite, discrete-time horizon $\tau = \{t_1, \ldots, t_M\}$ and a Markov stochastic process X_t on a complete filtered probability space $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$, where $\mathbb{F} = (\mathcal{F}_t)_{t\in\tau}$ is the augmented filtration generated by the underlying price process $(X_t)_{t\in\tau}$. We take $(X_t)_{t\in\tau}$ to be an $\mathbb{R}^d$-valued discrete time Markov chain representing the price of the underlying assets. We assume the sample space is the set of all possible realizations of the stochastic process X_t over τ, and $\mathbb{P}$ is the natural probability measure.

1.2 Problem Formation

Our problem is the abstraction from a real world problem depicted as follows: Suppose an institutional trader needs to execute a large order at a desired price level. However, the order cannot be liquidated completely at a fixed price level given the fact that a huge amount of shares traded will deplete market supply. The price impact hence will lead to losses when selling a huge portfolio. Nevertheless, because of the temporary nature of the price impact, splitting up the trade into smaller chunks and extending the trade time horizon enables investors to avoid huge losses. While the order is liquidated within a time interval, there exist two factors causing deviation from the desired execution price: the intrinsic volatility of price process and the reverse price impact caused by each split execution.

Our main concern here is to find a trade scheme, given a dynamics of the price process and a price impact model, for example, Geometric Brownian Motion with temporary and permanent price impact, to fulfill a particular optimality criterion, for example, expected value of utility function of terminal wealth.

First we introduce a model in section 1.3 that does not directly capture the permanent price impact. Later in section 1.5, we extend this model and incorporate the permanent price impact.

- $(\boldsymbol{\Omega, \mathcal{F}, (\mathbb{F}_t)_{t\in\tau}, \mathbb{P}})$ Filtered probability space defined in the previous section.
- $\boldsymbol{(h_t)_{t\in\tau} \in \mathbb{R}^l}$ Amount of shares to be traded at time t subject to certain constraints, that is, h_{t_k} is the amount of shares traded within time interval $[t_k, t_{k+1}]$.
- $\boldsymbol{(X_t)_{t\in\tau}}$ The underlying price process defined in the previous section. It is worth to notice that at this phase,h_t has no impact on he evolution of underlying state X_t.
- $\boldsymbol{\pi_t = (h_t, \ldots, h_T), h_t \in \mathbb{R}^l}$ Exercise strategy adapted to filtration $\mathcal{F}_t$.
- $\boldsymbol{H_t(X_t, h_t), t \in \tau}$ The effective payoff function at time t. Notice it is a function of underlying process $(X_t)_{t\in\tau}$ and exercise amount $(h_t)_{t\in\tau}$. Also note here this is the effective payoff that incorporates the temporary price impact which we will specify in section 1.4.
- $\boldsymbol{(y_t)_{t\in\tau} \in \mathbb{R}^l}$ Control process. In this context y_t means the total amount of shares left to be liquidated at time t, from the definition we have $y_{t+1} := y_t - h_t$ and $y_0 = N$, N to be the number of share to sell/but at time $t = 0$.
- $\boldsymbol{K_t(X_t, y_t), t \in \tau}$ Set-valued function on $\tau \times \mathbb{R}^d \times \mathbb{R}^l$ whose output is a set of subsets of $\mathbb{R}^l$. It is the admissible exercise decisions available at time t. Notice it could depend on the state of underlying X_t and the state of control process y_t.
- $\boldsymbol{\mathcal{P}_{k,y,t}}$ Set of K-admissible exercise policies starting at time t with $y_t = y$. $\pi_t = (h_t, \ldots h_T) \in \mathcal{P}_{k,y,t}$ if (1) h_s is $\mathcal{F}_s$-measurable for $s = t, \ldots, T$ and (2) $h_s(\omega) \in K_s(X_s(\omega), y_s)$ for all $s = t, \ldots, T$ and for all ω in a set of probability one.

The value of an exercise strategy at time t is hence given by

$$V_t^{K,\pi}(x, y_t) = \mathbb{E}\Big[\sum_{s=t}^{T} H_s(X_s, h_s) | X_t = x\Big] \tag{1.1}$$

We make a remark here that for simplicity, through out this chapter, interest rate is assumed to be 0 and no discounting factor is involved. Note in practice, the

assumption is valid if the trade is executed within one trading day. However, non-zero overnight interest rate exists if the trade time horizon is extended longer, in which case the discount factor should be applied to the model.

Now we are ready to define the value function

$$V_t^{K,*}(x,y) = \sup_{\pi \in \mathcal{P}(K,y,t)} V_t^{K,\pi}(x,y) = \sup_{\pi \in \mathcal{P}_{K,y,t}} \mathbb{E}\left[\sum_{s=t}^{T} H_s(X_s, h_s) | X_t = x \right] \tag{1.2}$$

$$(x,y) \in \mathbb{R}^d \times \mathbb{R}^l$$

In the following section we try to utilize Monte-Carlo method to determine $\pi(h_0, \ldots, h_T)$ and value of value function at time $t = 0$. A trading strategy is hence constructed and the expected effective payoff at time $t = 0$ can be predicted.

1.3 Numerical Approach Algorithm 1

1.3.1 Dynamic Programming Formulation

To tackle this stochastic control problem, we use Dynamic Programming, one of the most popular and effective approaches. The main idea was originated from Bellman's principle, which states

An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The value function satisfies dynamic principle, which is specified as follow:

$$V_t^{K,*}(x,y) = \begin{cases} \sup_{h \in K_T(x,y)} H_T(x,h) & \text{if } t = T \\ \sup_{h \in K_t(x,y)} \{ H_T(x,h) + \mathbb{E}[V_{t+1}^{K,*}(X_{t+1}, y-h) | X_t = x] \} & \text{if } 0 \le t \le T-1 \end{cases} \tag{1.3}$$

It is worth noting that unlike some exotic option pricing scheme with similar techniques, for example, swing option in [5], the control process h_t here is selected within a continuous interval rather than a discrete set with finite values.

1.3.2 Conditional Expectation Approximation

To continue the calculation, we need to deal with the conditional expectation term $\mathbb{E}[V_{t+1}^{K,*}(X_{t+1}, y-h) | X_t = x]$ at each time step. We introduce a slightly different version of least squares regression-base Monte Carlo method stated by Longstaff and Schwartz [11].

The main idea is to consider the projection of the function $x \to \mathbb{E}[V_n(X_n)|X_{n-1} = x]$ onto the subspace spanned by a finite set of independent basis function $\{\psi_1, \ldots, \psi_Q\}$ which has a linear combination form. This assumption can be formally justified, for example, when the conditional expectation is an element of the L^2 space of square-integrable functions relative to some measure. As a Hilbert space, L^2 has a countable orthonormal basis and the conditional expectation can be represented as a linear function of the basis elements.

To proceed with our numerical approximation, first we define the continuation value function

$$G_t(x, y) := \mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x] \tag{1.4}$$

Similar to the classical standard least squares regression based approach, we approximate the continuation value function $G_t(x, y)$ by an orthogonal projection onto a function space spanned by a set of basis functions $\{\psi_1, \ldots, \psi_Q\}$. However, note that here we choose basis functions depending on two variables rather than one, hence unlike original version, for $q = 1, \ldots, Q$, ψ_q is defined on $\mathrm{Dom}(G(\cdot, \cdot)), \mathbb{R}^d \times \mathbb{R}^l$. We have the approximation:

$$\hat{G}_t(x, y) = \sum_{q=1}^{Q} \gamma_{t,q}\psi_q(x, y) \approx G_t(x, y) = \mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x] \tag{1.5}$$

Since an extra variable is incorporated into basis functions, the orthogonal projection at time t is determined by both X_t and control variable rather than solely by X_t. We define the projection of the function to minimize

$$\mathbb{E}_{X_t,Y}\left(\mathbb{E}[V_{t+1}(X_{t+1}, Y)|X_t] - \sum_{q=1}^{Q} \gamma_{t,q}\psi_q(X_t, Y)\right)^2 \tag{1.6}$$

Where the control variable Y and underlying process X are independent random variables. Note the minimization of error term (1.6) is in $L^2 -$ norm sense where given the joint distribution of X_t and Y, the norm is defined as $\|\cdot\| := \mathbb{E}_{X_t,Y}|\cdot|^2$. Here we assume Y to be uniformly distributed on a boundary set. The distribution of X_t conditioned on some particular values of X_0 will be specified when implementing the algorithm based on the particular setup. If not emphasized otherwise, the law of X_0 is assumed to be uniform on a boundary set.

In terms of basis function choices, according to reference [11], we could use Hermite, Legendre, Chebyshev and Jacobi polynomials etc. The numerical result in [11]

indicates remarkably robustness to the choice of different basis functions. Here for bivariate basis functions, an example set of basis functions could be

$$\psi_1(x,y) = 1 \qquad \psi_2(x,y) = x \qquad \psi_3(x,y) = y \tag{1.7}$$

$$\psi_4(x,y) = x^2 \qquad \psi_5(x,y) = y^2 \qquad \psi_6(x,y) = xy \qquad \dots$$

The formulation of orthogonal projection suggests that, by increasing the number of basis functions $\psi_1, \psi_2, \dots$, denoted by q, $\{\hat{G}_t^q(x,y)\}$ converges to the conditional expectation

$$(x,y) \to \mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x]$$

where $\hat{G}_t^q(x,y)$ represents the approximation corresponding to a particular q at time t.

To determine the regression coefficient $\gamma_{t,q}$, for $q = 1, \dots, Q$, based on least squares optimization, we set the derivative with respect to $\gamma_{t,q}$ equal to zero.

$$\frac{\partial}{\partial \gamma_{t,r}} \mathbb{E}_{X_t,Y} \left(\mathbb{E}[V_{t+1}(X_{t+1}, Y)|X_t] - \sum_{q=1}^{Q} \gamma_{t,q} \psi_q(X_t, Y) \right)^2$$

$$= 2\mathbb{E}_{X_t,Y} \left[\mathbb{E}[V_{t+1}(X_{t+1}, Y)|X_t] \psi_r(X_t, Y) \right] - 2\mathbb{E}_{X_t,Y} \left[\sum_{q=1}^{Q} \gamma_{t,q} \psi_q(X_t, Y) \psi_r(X_t, Y) \right]$$

$$= 2\mathbb{E}_{X_t,Y}[V_{t+1}(X_{t+1}, Y)\psi_r(X_t, Y)] - 2\sum_{q=1}^{Q} \gamma_{t,q} \mathbb{E}_{X_t,Y}[\psi_q(X_t, Y)\psi_r(X_t, Y)] = 0$$

Set

$$B_{V,\psi} := \mathbb{E}_{X_t,Y}[V_{t+1}(X_{t+1}, Y)\psi(X_t, Y)]$$

$$B_{\psi} := \mathbb{E}_{X_t,Y}[\psi(X_t, Y)\psi(X_t, Y)^T]$$

Hence for $\gamma_t = (\gamma_{t,1}, \dots, \gamma_{t,Q})^T$, we have

$$B_{V,\psi} = B_{\psi}\gamma_t \tag{1.8}$$

Where $\psi(x,y) = (\psi_1(x,y), \dots, \psi_Q(x,y))^T$. Note $B_{v,\psi}$ and B_{ψ} both depend on t.

1.3.3 Regression Coefficients Estimation

1.3.3.1 MC estimation of $B_{V,\psi}$ and B_ψ

Define a set $\mathcal{G}_0^x$ of distinct initial values of X and generates independent trajectories x^i, $i = 1, \ldots, N$, of the process X, with $x_0^i = x$ for each $x \in \mathcal{G}_0^x$. Define $\mathcal{G}_t^x = \{x_t^i | 1 \leq i \leq N\}$ for $t = 1, \ldots, T$, where $N = |\mathcal{G}_0^x|$. Furthermore, for $t = 1, \ldots, T$, define a finite set

$$\mathcal{G}_t^{xy} \subseteq \mathrm{Dom}(V_t(\cdot, \cdot)) \subseteq \mathbb{R}^d \times \mathbb{R}^l \tag{1.9}$$

such that for all $(x, y) \in \mathcal{G}_t^{xy}$, $x \in \mathcal{G}_t^x$.

In practice, to estimate regression coefficient, we replace $B_{V,\psi}$, B_ψ with their Monte Carlo estimates. In order to estimate $B_{V,\psi}$, B_ψ using Monte Carlo techniques, for each t, we need the joint distribution of (X_t, X_{t+1}, Y).

To deal with this, we simulate trajectories of the X process as what we did in the first paragraph. By doing so, for any i and t, the distribution of (x_t^i, x_{t+1}^i) is sampled from the distribution of (X_t, X_{t+1}).

Then for a given t, we assume Y to be independent of the process X and uniformly distributed in a boundary interval. We define the grid $\mathcal{G}_t^{xy} = \{(x_t^i, y^i) | i = 1, \ldots N, t = 0, \ldots, T\}$ where y^i is an sample uniformly distributed on the boundary interval and independent of the sampled x trajectories.

Hence we have the MC estimations of $B_{V,\psi}$, B_ψ at each time step t:

$$\hat{B}_{V,\psi} := \frac{1}{|\mathcal{G}_{t+1}^{xy}|} \sum_{(x_{t+1}^i, y) \in \mathcal{G}_{t+1}^{xy}} V_{t+1}(x_{t+1}^i, y)\psi(x_t^i, y) \tag{1.10}$$

$$\hat{B}_\psi := \frac{1}{|\mathcal{G}_{t+1}^{xy}|} \sum_{(x_t^i, y) \in \mathcal{G}_t^{xy}} \psi(x_t^i, y)\psi(x_t^i, y)^T \tag{1.11}$$

1.3.3.2 $\mathcal{G}_t^{xy}$ Implementation

The choice of $\mathcal{G}_t^{xy}$ and $\mathcal{G}_0^x$, $t = 0, \ldots, T$, is a critical factor that determines the performance of least square regression estimates. How to improve simulation accuracy with limited computational cost is of our interest.

To begin with, we consider $\mathcal{G}_0^x$ to be randomly sampled from the law of X_0. However if there is no particular preference, we set $\mathcal{G}_0^x$ to be uniformly distributed along x-axis within an interval set by the endpoints obtained from real world data. In addition, other manipulations are acceptable according to different situation, for example, we could set more trajectories generating from a particular interval either

because the physical probability in this region is higher or we are more interested in these cases.

An important remark is that here we must guarantee that only one trajectory is generated from each point in $\mathcal{G}_0^x$. This is because if we allow otherwise cases, it is possible that matrix B_ψ in equation (1.8) would become singular which dramatically impairs the accuracy of our regression estimates. If we really are interested in the cases with more than one paths for each point in $\mathcal{G}_0^x$, one solution is to generate a discrete neighborhood of a point from which set of points could be selected. If more trajectories are required, we could simply refine our grid to satisfied a level of magnitude.

With $\mathcal{G}_0^x$ defined, each x_t from $\mathcal{G}_t^{xy}$ is randomly sampled from the conditional distribution $X_t|X_0$.

In terms of control variable y, we set it to be independent of x_t and randomly sampled from the uniform distribution on the support of y_t. Then we generate a small number of y for each x_t and construct the set $\mathcal{G}_t^{xy}$.

We consider two construction modes. One natural way is to define $\mathcal{G}_t^{xy} = \mathcal{G}_t^x \times \mathcal{G}_t^y$. That is, we first define a set of $\mathcal{G}_t^y$, for example, uniformly distributed grid within interval $[a, b]$. Then each x value is collocated with this same set $\mathcal{G}_t^y$ and form a pair $(x, y) \in \mathcal{G}_t^{xy}$. However, according to reference [8], in order to achieve a desired accuracy, an excessively large $\mathcal{G}_t^y$ is required, resulting in intractable computational cost. An preferable alternative is, instead of providing the same set $\mathcal{G}_t^y$ for each x value, we use some quasi-Monte Carlo method, such as rank-1 lattice rule, sobel sequence, etc, to tweak y vector a little while remain its distribution law and then allocate different $\{\mathcal{G}_t^y\}_n$ to each x value. If necessary, some random offset could be implemented to gain a confidence interval or to proceed with MC error estimation. $\mathcal{G}_t^{xy}$ constructed this way better covered the whole space rather than merely x-axis. To achieve the same level of accuracy, the grid size is significantly smaller. We demonstrate the difference between these two modes in figure 1.1 and 1.2.

As we can see, $\mathcal{G}_t^{xy}$ generated from rank-1 lattice rule better represent the whole space than $\mathcal{G}_t^x \times \mathcal{G}_t^y$.

Similar multivariate regression has been mentioned in some papers. [11] suggested the same method while the basis function was restricted to solely depend on factor X. We argue that indeed computational cost would boost as grid extended to 2-dimension with basis functions depending on two variables. However, with careful selection method of $\mathcal{G}_t^{xy}$ as promoted above, we can improve the estimate accuracy significantly compared with the former provided the same computational cost as mentioned in [8].

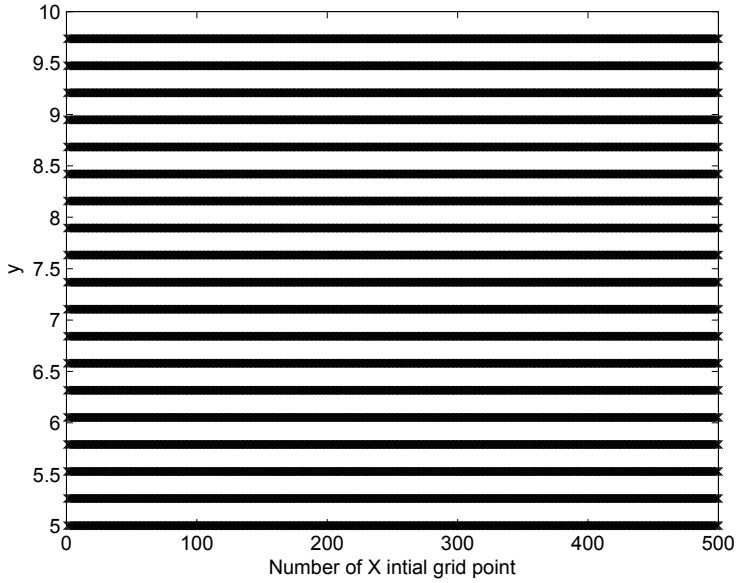


Figure 1.1: $\mathcal{G}_t^{xy} = \mathcal{G}_t^x \times \mathcal{G}_t^y$ (10000 points in total, 20 y-items per each x item)

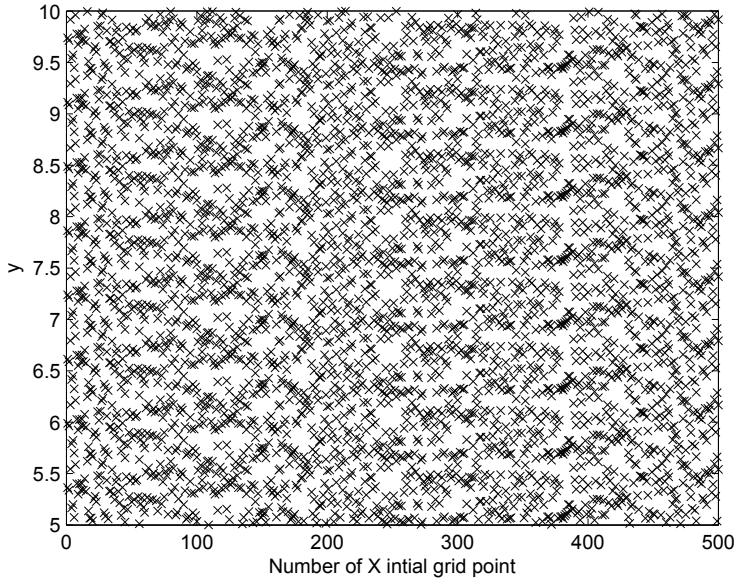


Figure 1.2: $\mathcal{G}_t^{xy}$ grid, y points generated rank-1 lattice rule (4000 points in total, 8 y-item per each x item)

1.3.4 The Plain Algorithm of LSR-based MC estimation

LSR-based MC estimation here stands for least squares regressions-based Monte Carlo estimation. From the construction of algorithm stated below, we see that it uses later information which makes it a high biased estimation in some sense. However, high biased as it is called, it is still uncertain that this estimation is an upper boundary of actual value. This is because various sources of errors should be considered. Apart from high biased error generated when estimating $G_t(x, y)$ in equation (1.5), there are Monte Carlo estimation error when calculating $\hat{B}_{V,\psi}, \hat{B}_{\psi}$ and error caused by discretization of domain of x, y. Hence the result may not fully represent its high biased feature although it is called so.

Given regression framework and construction of $\mathcal{G}_t^{xy}$, we are ready to present recursive valuation algorithm using dynamic principle formulation (1.3).

Set $\hat{y} = y - h$, from (1.3) we get,

$$V_t^{K,*}(x, y) = \begin{cases} \sup_{h \in K_T(x,y)} H_T(x, h) & \text{if } t = T \\ \sup_{y-\hat{y} \in K_t(x,y)} \{H_T(x, y - \hat{y}) + \mathbb{E}[V_{t+1}^{K,*}(X_{t+1}, \hat{y}) | X_t = x]\} & \text{if } 0 \leq t \leq T-1 \end{cases}$$

Notice $\hat{y}$ and y correspond to time period $t+1$ and t respectively.

The valuation scheme proceeds as follow

1. At time t=T, define
$$V_T(x, y) = \sup_{h \in K_T(x,y)} H_T(x, h)$$

2. Set time $t+1 \to t$, we seek $V_t(x, y)$ for every $(x, y) \in \mathcal{G}_t^{xy}$

3. Recall continuation value function (1.4). Given the set,
$$\{(y, x_t, x_{t+1}, V_{t+1}(x_{t+1}, y)) | (x_{t+1}, y) \in \mathcal{G}_{t+1}^{xy}\}$$
we derive the regression coefficient in estimation formula (1.5) using (1.10) (1.11) and (1.8). Hence define $\hat{G}_t(x, y)$ on $\mathrm{Dom}(\hat{G}_t(x, \hat{y}))$.

4. For each $(x, y) \in \mathcal{G}_t^{xy}$, solve the optimization problem
$$V_t(x, y) = \sup_{y-\hat{y} \in K_t(x,y)} \{H_t(x, y - \hat{y}) + \hat{G}_t(x, \hat{y})\}$$
That is, we fix each $x \in \mathcal{G}_t^x$ and search for $\hat{y} \in \mathcal{G}_{t+1}^y$ to maximize $\{H_t(x, y - \hat{y}) + \hat{G}_t(x, \hat{y})\}$.

5. If $t \geq 0$, continue with step 2; otherwise, we derive $V_0(x, y)$ as the LSR-based MC estimation.

Note that this valuation scheme yields an estimate of regression coefficient and value of goal function at time t, for $t = 1, \ldots, T$. Since we use identical trajectories to calculate both, this may yield a high biased value. To avoid this, we could generate another set of trajectories and implement forward valuation scheme using regression coefficients derived here. It yields a low biased value since the solution would be merely sub optimal. Therefore it provides a standard to confirm the performance of our algorithm as the low biased and high biased value ought to behave similarly. We'll further discuss this issue in chapter 2.

1.4 Price Impact Model

Now that the valuation algorithm has been constructed, in the following sections, we proceed to introduce a price impact model and incorporate it into the trading scheme construction.

In the last few years, many paper referring to the construction of different price impact models for different markets and the calibration of parameters were published. It is widely agreed that price impact can be separated into two terms, temporary impact and permanent impact.

Temporary component refers to the temporary price movements away from equilibrium resulting from the temporary imbalanced supply-demand relationship. This component is believed to be strongly sensitive to time horizon and trading scheme. Moreover, market event and release of information have significant influence on the calibration of parameters.

Permanent component, on the other hand, refers to the information transmission to the market by the buy/sell imbalance. This component is believed to be roughly independent of the time horizon and trading scheme. It should persist within the life of our liquidation and would not fade away easily.

1.4.1 Price Dynamics

The price process X_t evolves according to two factors, volatility and drift, which are assumed to fit the market structure and independently of the trading scheme. In terms of the dynamics of X_t, naturally we choose a standard geometric Brownian motion

$$dX(t) = \mu X(t)dt + \sigma X(t)d\mathbb{W}(t) \tag{1.12}$$

Where μ is the drift and σ is the volatility.

However, according to [2], though it is important to consider geometric Brownian motion over long-term or extremely volatile market due to the fact that σ is in scale with $X(t)$, over short-term time horizon, the total fractional price changes are small, hence we could use the arithmetic Brownian motion as approximation. Which is

$$dX(t) = \mu X(0)dt + \sigma X(0)d\mathbb{W}(t)$$

We present the temporary price impact and permanent price impact under both dynamics in the following section.

1.4.2 Temporary Price Impact

To understand temporary price impact, we first present a brief introduction of order book. The order book is the list of all buy and sell limit orders[1], with their corresponding price and volume, at a given instant of time. When a new order appears(for example a sell order), it either adds to the book if it is above the bid price, or generates a trade at the bid if it is below or equal to the bid price[2]. If we try to execute a trade with rather huge amount of units at a given price level in a short time interval, it first deplete all the best offer available in the order book and liquidate part of the total amounts. The rest looks for second best offer and it goes on until the order is completely liquidated. Hence the temporary price impact reduces our effective payoff. However, due to the resilience feature[3], it could be optimized if we split our trade into several chucks and extend the execution time horizon. We make the assumption that in discretized model, temporary impact persists within each time interval. The equilibrium price is re-established when time proceeds to a new interval.

In our framework, as the definition of variable on page 4 suggested, we first define the rate of execution v within time interval $[t_k, t_{k+1}]$ to be $v_{t_k} = \frac{h_{t_k}}{\triangle t}$, $t_k \in \tau$. The assumption implied is that within each time interval $[t_k, t_{k+1}]$, $k = 0, \ldots, M-1$, execution rate remains constant and the impact fades away when entering into a new time interval.

[1]Limit order means an order placed with a brokerage to buy or sell a set number of shares at a specified price or better. Limit orders also allow an investor to limit the length of time an order can be outstanding before being canceled.

[2]These orders are called market orders.

[3]Resilience means in the order book, the supply at a given price level will be replenished in a short time interval if depleted.

Recall our definition of $H_t(X_t, h_t)$ on page 4, here to incorporate the temporary impact, we replace $X_{realized}(t)$ with X_t as the input for H_t. For example, if H_t is the effective payoff $h_t X_t$, then $H_t(X_{Realized}(t), h_t) = h_t X_{realized}(t)$.

According to [6], in the GBM case, model can be constructed as follow

$$X^G_{Realized}(v,t) \quad = \quad f(v)X(t) \tag{1.13}$$

where

$$f(v) = (1 + \kappa_s sgn(v)) \exp[\kappa_t sgn(v)|v|^\beta] \tag{1.14}$$

$$\begin{cases} \kappa_s = \text{bid-ask spread parameter} \\ \kappa_t = \text{temporary price impact factor} \\ \beta = \text{ the price impact exponent} \end{cases}$$

Here we assume $\kappa_s < 1$ to guarantee $X_{Realized}(v,t) \geq 0$.

Hence H_t (if defined as effective payoff) under this model is specified as $H_t(X_{Realized}(t), h_t) = X_t f(\frac{h_t}{\triangle t}) \times h_t$.

In the ABM case, model can be constructed as follow,

$$X^A_{Realized}(v,t) = X(t) + X_0 l(v) \tag{1.15}$$

$$l(v) = \kappa_s sgn(v) + \kappa_t sgn(v)v \tag{1.16}$$

where $\hat{X}$ represents post-trade adjusted price compared to original X from trajectories. See reference () for detail.

[14] suggested some relationship between (1.13), (1.14) and (1.15), (1.16) which is displayed as follow. If we assume $\beta = 1$, $\kappa_t|v| \ll 1$ and $\kappa_t\kappa_s \ll 1$, (1.14) is approximately

$$f(v) \approx 1 + \kappa_s sgn(v) + \kappa_t sgn(v)v$$

by Taylor's expansion. Hence for $X(t) \approx X(0)$,

$$\begin{aligned} X^G_{Realized}(v,t) - X(t) \quad &= \quad f(v)X(t) - X(t) = X(t)[\kappa_s sgn(v) + \kappa_t sgn(v)v] \\ &\approx \quad X(0)[\kappa_s sgn(v) + \kappa_t sgn(v)(v)] \end{aligned}$$

$$X^A_{Realized}(v,t) \approx X^G_{Realized}(v,t)$$

1.4.3 Permanent Price Impact

As [14] suggested, in a continuous setting, we introduce a permanent price impact as an additional drift term incorporated into the dynamics.

Let the price $X_t = x$ at time t. If there is no trade, the price dynamics can be given by the SDE

$$dX_s^{x,t} = \mu X_s^{x,t} ds + \sigma X_s^{x,t} dW_s, \quad for\ s \in [t, t + \Delta t].$$

where the superscript denotes the initial condition at t.

If there is a trade with intensity v, then the drift changes to $\mu + g(v)$, hence the actual price process will satisfy

$$dZ_s = (\mu + g(v))Z_s ds + \sigma Z_s dW_s, \quad Z_t = x. \tag{1.17}$$

where the trading rate v is defined as a function of current state. Note Z_t denotes it is the underlying price incorporated with the permanent price impact. In this context, it depends on h_t

For linear permanent price impact, we set $g(v)$ to have the from

$$g(v) = \kappa_p v$$

Where κ_p is the permanent price impact factor.

To discretize the above model, note if we assume trading rate v is constant during each time interval $[t_k, t_{k+1}]$, $v_{t_k} = \frac{h_{t_k}}{\triangle t}$ becomes the discretized form of v_t. If the time-step is short enough, then

$$Z_{t+\Delta t} \approx Z_t(1 + g(v)\Delta t) + \int_t^{t+\Delta t} \mu Z_s ds + \sigma Z_s dW_s$$

that is

$$Z_{t+\Delta t} \approx X_{t+\Delta t}^{x+g(v)\Delta t, t} \tag{1.18}$$

From formula (1.18), we can see the adjusted underlying price is $Z_{t+\Delta t}$ approximately the original trajectory starting from another initial point $x + g(v)\Delta t$. We would refer to this result in section 1.5.

The explicit form of adjusted underlying price is

$$Z_{t+\triangle t} \approx X_{t+\triangle t} + h_t \kappa_p x \tag{1.19}$$

Note the permanent price impact for GBM and ABM dynamics are identical.

Note here in this context, h_t is negative if the action is sell. To be compatible with defnition on page 4, where h_t and y_t are defined to be positive, we change the sign in equation (1.14) and $sgn(v)$ in equation (1.19) to represent different action type[4].

1.5 Numerical Algorithm Approach 2

In this section, we extend our algorithm derived in section 1.3 by incorporating the price impact model introduced in section 1.4. By default we assume the action is sell[5] and h_t, y_t to be positive through the execution. Recall our algorithm in section 1.3.4. The recursive feature of this algorithm relies on the fact that given a trajectory, we could look back from $X_{t+1}(\omega)$ to $X_t(\omega)$. However, taking price impact into account, apart from $X_{t+1}(\omega)$, another variable, control process y_t also has influence on $X_t(\omega)$ which complicates our algorithm. After all, the price impact is more naturally to be incorporated in a forward scheme, where given $X_t(\omega)$ and h_t, the trajectory is adjusted correspondingly from t_k to t_{k+1}.

While intuitively this seems tricky, from section 1.4.3, particularly formula (1.19) we notice that the mere difference between these two is a shift of value at time $t+1$. Hence if we assume the grid is fine enough, we can regard price impact as a jump from one trajectory to another. When we move to a next interval, we merely shift the initial value corresponding to the price impact rather than modifying trajectories.

As shown in figure 1.3, three trajectories are generated. The red dotted line indicates one adjusted trajectory. At time $t = 1$, it shifts from $X_1(\omega_1)$ (red) to $X_1(\omega_3)$ (green). At time $t = 2$, it again shifts from $X_2(\omega_3)$ (green) to $X_2(\omega_2)$ (blue). Note actually it is a jump from different (x, y) pairs rather than different x values as our continuation function depend on two variables.

Next we rigorously adjust our algorithm. Modification for permanent price impact is displayed as follow. In terms of temporary price impact, due to its characteristic, we could simply substitute realized price (1.13) for X_t in the payoff function as we explained.

Figure 1.4 further display how equation (1.18) should work. The red line, both dotted and solid indicate the same transition density corresponding to the intrinsic dynamic of the underlying price. The green line indicates with price impact, X_{t+1} is shifted by $\text{ppi}(h_t)$ to Z_{t+1} which is specified in (1.19) except that we use minus

[4]That is positive to represent buy order and negative to represent sell order.

[5]For buy order, simply change the sign of $\text{ppi}(h)$

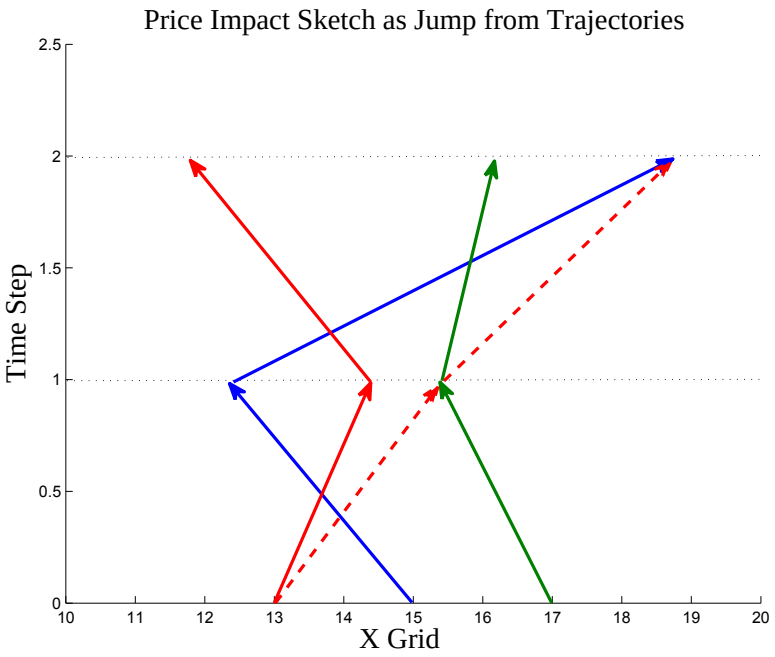


Figure 1.3: Price Impact Sketch as Jump from Trajectories

instead of plus to represent sell action. Hence to adjust for the replacement of Z_{t+1} with X_{t+1} in $V_{t+1}(Z_{t+1}, y_{t+1}(y_t, h_t))$, we simply need to shift X_t to Z_t by $\text{ppi}(h_t)$ to preserve the same transition density.

Hence the conditional expectation we need to estimate now can be written as

$$\mathbb{E}[V_{t+1}(Z_{t+1}, y_{t+1}(y_t, h_t))|Z_t - \text{ppi}(h_t) = x - \text{ppi}(h_t)] \tag{1.20}$$

where $y_{t+1} = y_t - h_t$, $\text{ppi}(h)$ is the permanent price impact function. h_t is positive and there is a minus in front of $\text{ppi}(h_t)$ to represent the sell action. Note Z_t depends on h_t from its definition. From formula (1.19), we define $\text{ppi}(h)$ to be,

$$\text{ppi}(h_t) = h_t \kappa_p X_t$$

Our continuation value function (1.4).

$$G_t(x, y) := \mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x]$$

It is worth noting that the transition density between $Z_t - \text{ppi}(h_t)$ and Z_{t+1} is the same as transition density between X_t and X_{t+1}. Hence the least squares regression method still work in this setting.

To calculate (1.20), instead of $G(x, y - h)$, now we estimate $G(x - \text{ppi}(h), y - h)$.

Finally we present the modified algorithm.

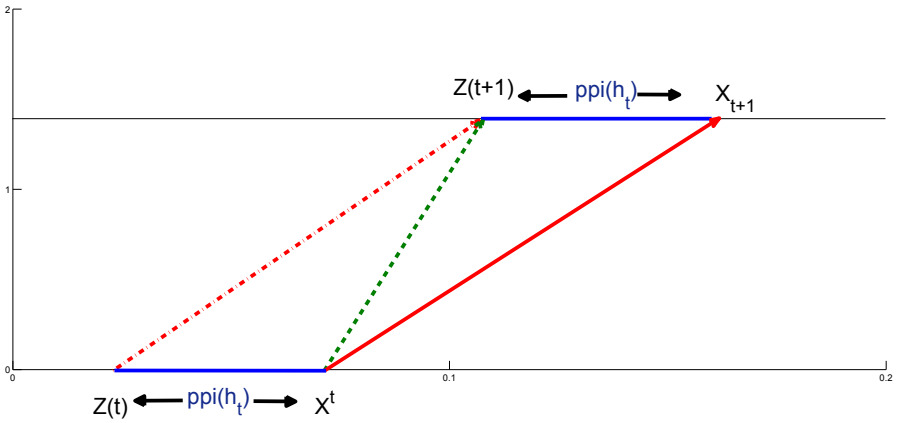


Figure 1.4: Conditional Expectation Illustration

1. At time t=T, define
$$V_T(x, y) = \sup_{h \in K_T(x,y)} H_T(x, h)$$
Note here x is the realized price. However while at time $t = T$, $h_T = 0$ from its definition, temporary price impact is zero, hence x remains the same.

2. Set time $t + 1 \to t$, we seek $V_t(x, y)$ for every $(x, y) \in \mathcal{G}_t^{xy}$

3. Recall continuation value function (1.4). Given the set,
$$\{(y, x_t, x_{t+1}, V_{t+1}(x_{t+1}, y))|(x_{t+1}, y) \in \mathcal{G}_{t+1}^{xy}\}$$
we derive the regression coefficient in estimation formula (1.5) using (1.10) (1.11) and (1.8). Hence define $\hat{G}_t(x, y)$ on $\text{Dom}(\hat{G}_t(x, \hat{y}))$. Note though $\text{Dom}(\hat{G}_t(x, \hat{y}))$ is a discrete set when implementing numerical approach, we can do some interpolation extending it to a continuous domain with regression coefficient we derived.

4. For each $(x, y) \in \mathcal{G}_t^{xy}$, solve the optimization problem
$$V_t(x, y) = \sup_{h \in K_t(x,y)} \{H_t(x, h) + \hat{G}_t(x - \text{ppi}(h), \hat{y})\}$$
That is, we fix each $(x, y) \in \mathcal{G}_t^{xy}$ and search for $\hat{y} \in \mathcal{G}_{t+1}^{y}$, $h = y - \hat{y}$ to maximize $\{H_t(x, h) + \hat{G}_t(x - \text{ppi}(h), \hat{y})\}$. Note x is post-trade price described in equation (1.13) to represent temporary price impact.

5. If $t \geq 0$, continue with step 2; otherwise, we derive $V_0(x, y)$ as the LSR-based MC estimation.

Chapter 2

Numerical Algorithm Implementation and Results

In this chapter we present the numerical implementation and some results based on the algorithm described in chapter 1. The algorithm is implemented using MATLAB, which is provided in Appendix A section of this thesis.

We assume a geometric Brownian motion

$$dX_t = rX_t dt + \sigma X_t d\mathbb{W}_t$$

as the dynamics of underlying price. Where the interest rate r and volatility σ are both constant. X_t denotes the underlying price process and $\mathbb{W}_t$ is a Brownian motion. In order to construct trajectories we use the explicit formula

$$X_{t_{k+1}} = X_{t_k} e^{(r-\frac{1}{2}\sigma^2)\triangle t + \sigma\sqrt{\triangle t} Z_k}$$

where $\triangle t = t_{k+1} - t_k$ is constant and same for every k and $Z_k \sim \mathcal{N}(0,1)$.

2.1 Basic Test and Observation

In terms of interest rate r, as we discussed in chapter 1, through this chapter we assume $r = 0$. As for daily trade scenario, discount factor is negligible. However in terms of weekly trade or trade within longer time interval, discount factor may have influence over our strategy. In addition this assumption should also be modified if trader believes there is an obvious upward or downward trend of the price dynamics or some major event occurs, driving price move toward certain direction.

Through out this chapter we assume sell action, effective payoff goal function $H(X_t, h_t) = h_t X_t$ and interest rate $r = 0$.

2.1.1 Test of Regression Component

One critical component of our algorithm implementation is to calculate the regression coefficients based on the formula

$$\mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x] \approx \sum_{q=1}^{Q} \gamma_{t,q}\psi_q(x, y)$$

To test the accuracy and robustness of our regression coefficient approximation, set $X_t = \mathbb{W}_t$, $\mathbb{W}_t$ is Brownian motion. From the definition and property of Brownian motion, we have

$$\begin{aligned}
\mathbb{E}(X_{t+1}|X_t) &= \mathbb{E}(X_{t+1} - X_t + X_t|X_t) = X_t \\
\mathbb{E}(X_{t+1}^2|X_t) &= \mathbb{E}[(X_{t+1} - X_t)^2 - X_t^2 + 2X_tX_{t+1}|Xt] \\
&= \mathbb{E}[(X_{t+1} - X_t)^2] + 2\mathbb{E}[X_t(X_{t+1} - X_t + X_t)|X_t] - X_t^2 \\
&= \mathbb{E}[X_{t+1}^2 + X_t^2 - 2X_tX_{t+1}] + X_t^2 \\
&= X_t^2 + t + 1 + t - 2t = X_t^2 + 1
\end{aligned}$$

If V_{t+1} has the form $V_{t+1} = a+bx+cy+dx^2+ey^2+fxy$. Then $\mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x]$ should have the form

$$\mathbb{E}[V_{t+1}(X_{t+1}, y)|X_t = x] = a + d + bx + cy + dx^2 + ey^2 + fxy$$

Hence $\{\gamma_q\}_{q=1,\ldots,6}$ should approximately be $\{a + d, b, c, d, e, f\}$.

We make some observations via assigning different values to various parameters. Through this section we use the same basis function as suggested in chapter 1. Our first observation is that the magnitude of number of trajectories N has dramatic influence over the coefficients accuracy. To balance our computational cost and accuracy, we suggest the magnitude of N to be 10^4. In addition, computational cost is much more sensitive to the number of grid points of control process y. Acceptable grid point number Y should be within 500 otherwise the computational cost is intractable. Another observation is that coefficient accuracy is closely related to the relative magnitudes of y and X, for example, if the price process is of magnitude 10, it's impossible to have a reasonable result given y of magnitude 10^6, no matter how we increase N and Y. Hence in practice, how we choose the measure units for different parameters should be given careful consideration.

Here are some test results in table 2.1 and 2.2, given $X_t = \mathbb{W}_t$, $a + d = 4$, $b = 6$, $c = 12$, $d = 1$, $e = 4$, $f = 7$, $N = 10000$, $Y = 100$. We can see our algorithm accomplishes rather accurate approximation within this framework.

	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	$\hat{e}$	$\hat{f}$
t=10	3.9830	6.0014	12.0016	1.0021	4.0000	7.0004
t=50	3.9724	5.9977	11.9919	1.0002	4.0000	6.9993
t=100	3.9607	6.0019	11.9933	1.0006	4.0000	7.0008

Table 2.1: Test of regression coefficient estimation, MC expectation, 100 samples

	$\hat{a}$	$\hat{b}$	$\hat{c}$	$\hat{d}$	$\hat{e}$	$\hat{f}$
t=10	0.3095	0.0015	0.0051	0.0030	0.0000	0.0006
t=50	0.3081	0.0012	0.0054	0.0001	0.0000	0.0001
t=100	0.3270	0.0012	0.0055	0.0000	0.0000	0.0001

Table 2.2: Test of regression coefficient estimation, MC variance, 100 samples

2.1.2 Test of the Plain Algorithm Implementation

Now that we have confirmed the effectiveness of our regression component, next we examine our algorithm framework with all price impact parameter set in a way that the market is free from price impact, that is, we assume our trading volume is relative small.

Underlying price dynamic is assumed to follow a geometric Brownian motion and the goal function $H(X_t, h_t)$ simply to be the effective payoff $h_t X_t$. All parameters specification are presented in table 2.3

Note from the observation made in section 2.1.1, we set $N = 10000$, $Y = 100$, and domain of X_t, y_t with same level of magnitude. Also note this is a two time step simulation and the interest rate is set to 0.

Figure 2.1 displays effective payoff at time $t = 0$, and $t = 1$, for a particular trajectory, given different values of σ. Note it is perfect linear and the effective payoff value does coincide with our goal function format. Optimal value for selling y amount of units at time $t = 0$ is approximately $y_t X_0$. The comparison also indicates that as

Number of trajectories $\boldsymbol{N}$	10000	Interest rate $\boldsymbol{r}$	0
Number of time steps $\boldsymbol{M}$	2	Volatility $\boldsymbol{\sigma}$	0.02
Time scale $\boldsymbol{T}$	1	X_0 lower bound $\boldsymbol{X_{0Lower}}$	40
Number of basis function $\boldsymbol{q}$	6	X_0 upper bound$\boldsymbol{X_{0Upper}}$	45
Number of y grid point $\boldsymbol{Y}$	100	y lower bound $\boldsymbol{y_1}$	0
Number of refined y grid point $\boldsymbol{Yfine}$	100	y upper bound $\boldsymbol{y_2}$	15
Permanent price impact factor $\boldsymbol{\kappa_p}$	0	Bid-ask spread factor $\boldsymbol{\kappa_s}$	0
Temporary price impact factor $\boldsymbol{\kappa_t}$	0	Price exponent factor $\boldsymbol{\beta}$	1

Table 2.3: Parameters Specification

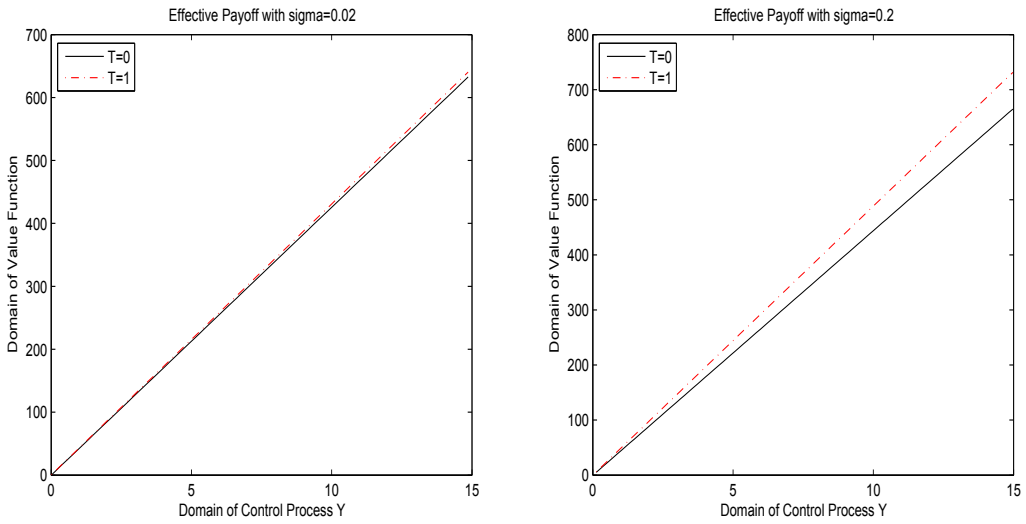


Figure 2.1: Comparison of Effective Payoff with $\sigma = 0.2$ and $\sigma = 0.02$, with $X_0 = 42.5$

σ increase, difference between global optimal ($t = 0$) and local optimal ($t = 1$) value becomes larger which is in line with our intuition. In addition, the global optimal value with $\sigma = 0.2$ is larger , due to the fact that the higher volatility it has, the more advantage we could take than trivial strategy.

Figure 2.2 displays a 3-dimension surface of effective payoff. From figure 2.1 and 2.2 we can see the algorithm framework perform satisfactorily within this naive set-up.

2.2 Artificial Market Set-up and Implementation

In this section we set up and simulate two artificial markets to evaluate our algorithm performance.

We obtain the general sense of parameter scale and magnitude according to [2]. We assume 250 trading days per year and 420 minutes as daily trading time. [6] suggests for liquidity market, daily $\sigma \approx 0.07$ corresponding to 1154 bps standard variation of $X(T)$ per year and for illiquid market, daily $\sigma \approx 0.06$ correspond to 632 bps standard variation of $X(T)$ per year. According to [14], $\bar{\kappa}_t$, the temporary price impact parameter correspond to liquidating 1% of the daily volume traded of a typical large-cap liquid stock, has a level of magnitude between 10^{-6} and 10^{-7}. If we set $\beta = 1$, and the executed amount of units accounts for $Z\%$ of the total daily traded volume , we can set κ_t as $\kappa_t = (\bar{\kappa}_t)Z$.

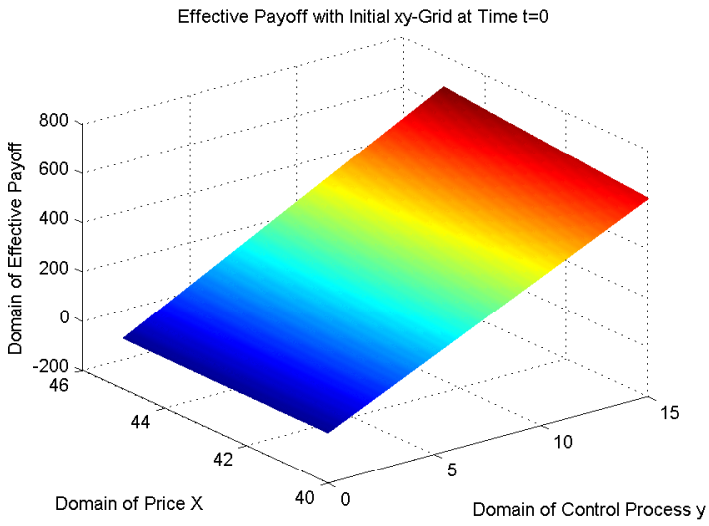


Figure 2.2: Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid

2.2.1 Weekly Execution

For our first scenario, trade is supposed to be liquidated within a week. We use $T = 5$, T is measured by day. Time step M is also five, meaning we want to know how many units to be executed per day. Daily volatility is set to be$\sigma = 0.05$. For price impact parameter, it is worth noticing

$$dt = \frac{T}{M} \qquad v = \frac{h}{dt} \Longrightarrow v = \frac{hM}{T}$$

Since v is the primary input for price impact evaluation, scale of κ strongly depends on scale of T. For daily trading, T could be set either to be $\frac{1}{250}$, implying year scale, or to be1, implying daily scale. κ with different magnitude should be adopted for these two cases, . Control process y denotes bp of daily trading volume, the upper bound indicates liquidating 2% of daily traded volume. Action type is sell, which means execution would drag down ideal price level.

Table 2.5 specifies all parameters in this scenario.

Figure 2.5 and 2.6 display the numerical results.

2.2.2 Daily Execution

For our second scenario, trade is supposed to be liquidated within a trade day. We use $T = 1$ and time step M is approximately $420/5 \approx 80$, which follows the assumption that we take five minutes interval as measure unit and a typical trading day has 420 trading minutes. Interval volatility is set to be $\bar{\sigma} = 0.05/\sqrt{90} \approx 0.005$.

Number of trajectories $\boldsymbol{N}$	10000	Interest rate $\boldsymbol{r}$	0
Number of time steps $\boldsymbol{M}$	5	Volatility $\boldsymbol{\sigma}$	0.05
Time scale $\boldsymbol{T}$	5	X_0 lower bound $\boldsymbol{X_{0Lower}}$	40
Number of basis function $\boldsymbol{q}$	6	X_0 upper bound $\boldsymbol{X_{0Upper}}$	100
Number of y grid point $\boldsymbol{Y}$	100	y lower bound $\boldsymbol{y_1}$	0
Number of refined y grid point $\boldsymbol{Yfine}$	200	y upper bound $\boldsymbol{y_2}$	200
Permanent price impact factor $\boldsymbol{\kappa_p}$	0.002	Bid-ask spread factor $\boldsymbol{\kappa_s}$	0
Temporary price impact factor $\boldsymbol{\kappa_t}$	2×10^{-6}	Price exponent factor $\boldsymbol{\beta}$	1

Table 2.4: Parameter Specification: Artificial Market 1

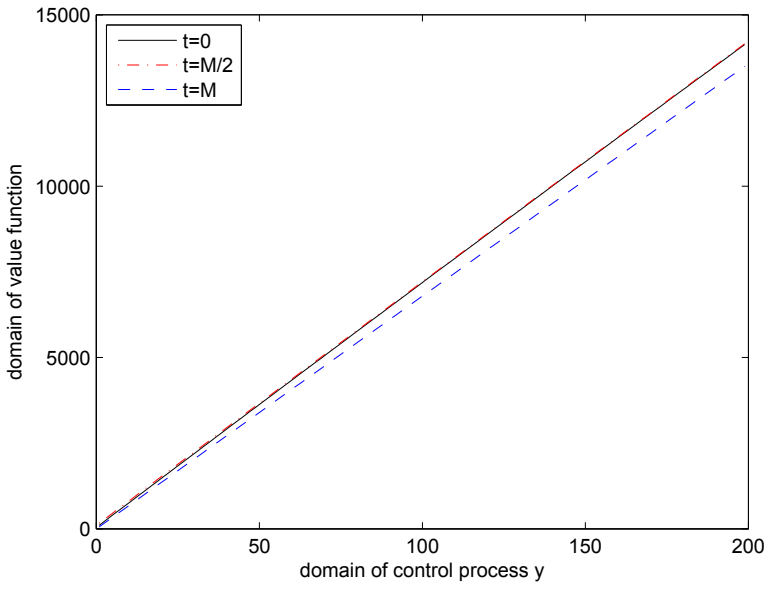


Figure 2.3: Optimal Effective Payoff at Different Time Step with $X_0 = 70$

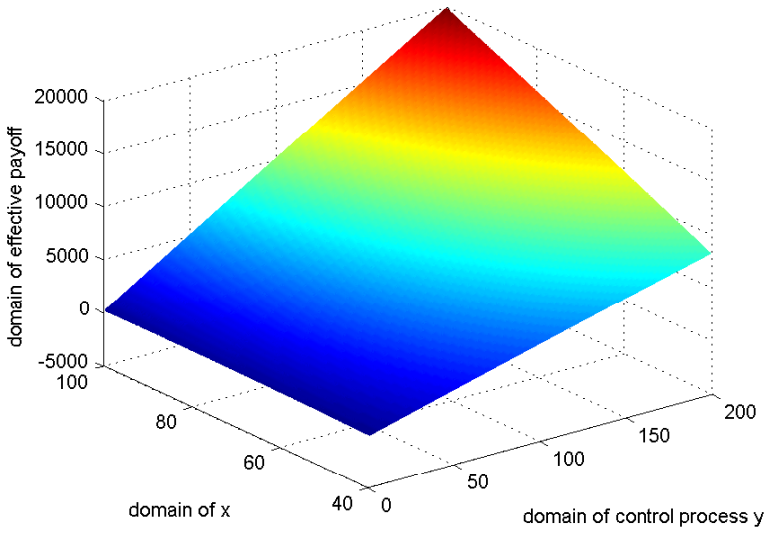


Figure 2.4: Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid

Number of trajectories $\boldsymbol{N}$	10000	Interest rate $\boldsymbol{r}$	0
Number of time steps $\boldsymbol{M}$	80	Volatility $\boldsymbol{\sigma}$	0.05
Time scale $\boldsymbol{T}$	1	X_0 lower bound $\boldsymbol{X_{0Lower}}$	40
Number of basis function $\boldsymbol{q}$	6	X_0 upper bound $\boldsymbol{X_{0Upper}}$	100
Number of y grid point $\boldsymbol{Y}$	100	y lower bound $\boldsymbol{y_1}$	0
Number of refined y grid point $\boldsymbol{Yfine}$	200	y upper bound $\boldsymbol{y_2}$	200
Permanent price impact factor $\boldsymbol{\kappa_p}$	0.002	Bid-ask spread factor $\boldsymbol{\kappa_s}$	0
Temporary price impact factor $\boldsymbol{\kappa_t}$	2×10^{-6}	Price exponent factor $\boldsymbol{\beta}$	1

Table 2.5: Parameter Specification: Artificial Market 2

Table 2.5 specifies all parameters in this scenario.

Figure 2.5 and 2.6 display the numerical results.

2.2.3 Comment on Scenario Results

Compare figure 2.3 and 2.5, we notice difference between optimal value at various time-step is much smaller for daily trade scenario, which coincides with our intuition that the single day volatility would be smaller than the 5-day volatility. Compare figure 2.4 and 2.6, we notice that the absolute effective payoff is much smaller for daily trade scenario, which coincides with our intuition that when executing a certain order, the shorter our trading horizon is, the larger the temporary price impact would be.

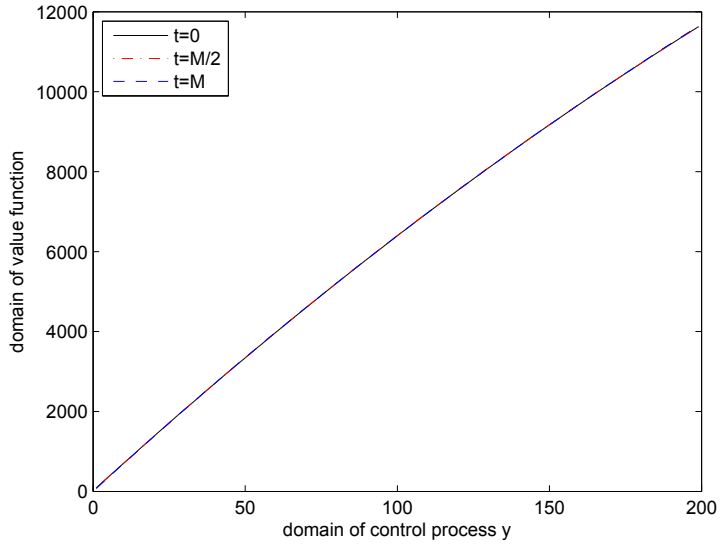


Figure 2.5: Optimal Effective Payoff at Different Time Step with $X_0 = 70$

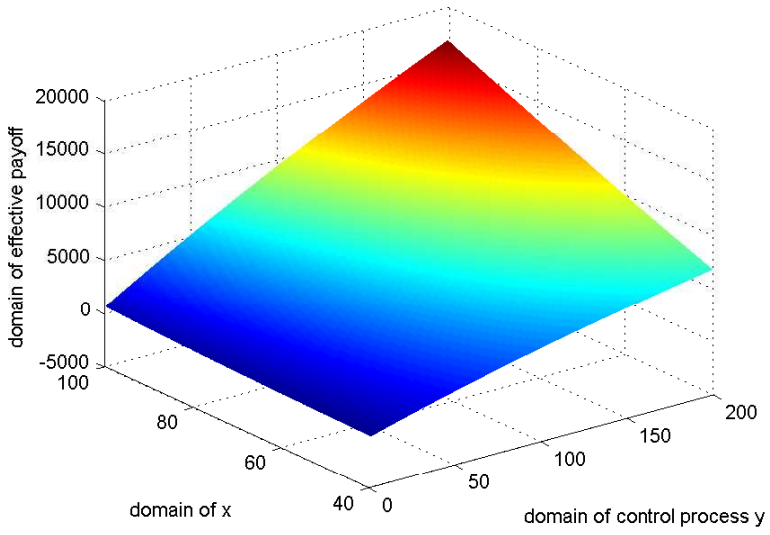


Figure 2.6: Effective Payoff at time $T = 0$ with Initial $\mathcal{G}_0^{xy}$-grid

2.3 Optimal Execution Strategy

Above discussion mainly focused on estimating value of our value function at time $t = 0$. However in practice we are more interested in deriving a trading strategy as this is the solution to our original real world problem.

There are two kinds of strategies, the first is static or piecewise static strategy. It means based on the information we obtained at initial time, a global optimal strategy could be built. The other type is dynamic strategy where the execution amount at time $t+1$ is decided based on information obtained at time t rather $t = 0$. For certain kind of price impact function and goal function, the optimal trading strategy could be static, however how to determine whether the strategy is static or dynamic falls beyond the scope of this thesis.

This algorithm framework could easily be extended to derive both kind of strategies. For static strategy, we simply need to focus on the global optimal solution and the corresponding $\{h_t(\omega)\}_{t\in\tau}$ which constitutes a trading strategy. For dynamic strategy, we need to create a variable to store each optimal $\{h_t(\omega)\}_{t\in\tau,\omega\in\Omega}$ in the process of maximizing our target function. From step to step, we look up corresponding $(x, y) \in \mathcal{G}_t^{xy}$ that fits the information from filtration $\mathcal{F}_t$ and select each h_t. $\{h_t\}_{t\in\tau}$ is our final dynamic trading strategy.

Here we demonstrate some results using the set of parameters as in table 2.5 and 2.5. All parameters remain the same except we change the upper bound and lower bound of the initial value X_0 to be 65 and 75. We use these parameters to derive a matrix of regression coefficients γ with which we try to simulate the trade execution strategy given a particular trajectory starting from $X_0 = 70$ and various y, denoting the volume need to be executed at time $t = 0$.

Figure 2.7 and figure 2.8 respectively demonstrate the daily and weekly trade execution scheme. Here x-axis indicates different time step, y-axis indicates different initial amount to be executed. The initial price for both cases is fixed at $X_0 = 70$. z-axis indicates the residual amount need to be executed at each time step. We make the constraint that the trade is completely liquidated within the time horizon. For detailed setup, see section 2.2. From the results, an important observation is that trade is likely to be liquidated earlier in weekly trade setting than daily trade setting. This is because for the same level of volatility, if the time horizon is longer, we are more likely to suffer from the underlying price volatility if we spend too much time waiting for the price impact to diminish.

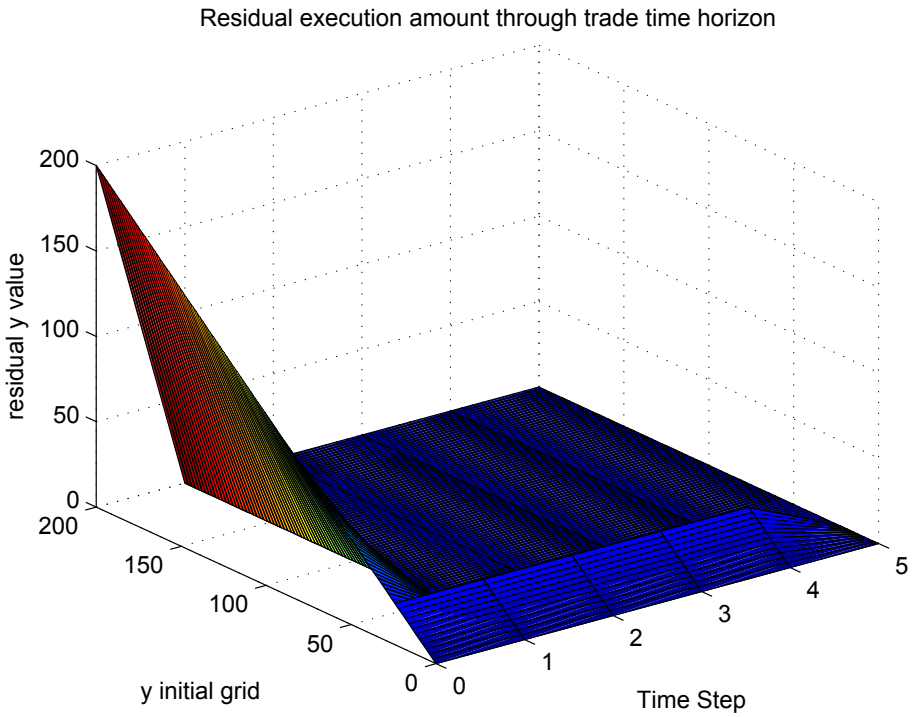


Figure 2.7: Weekly Trading Strategy Simulation

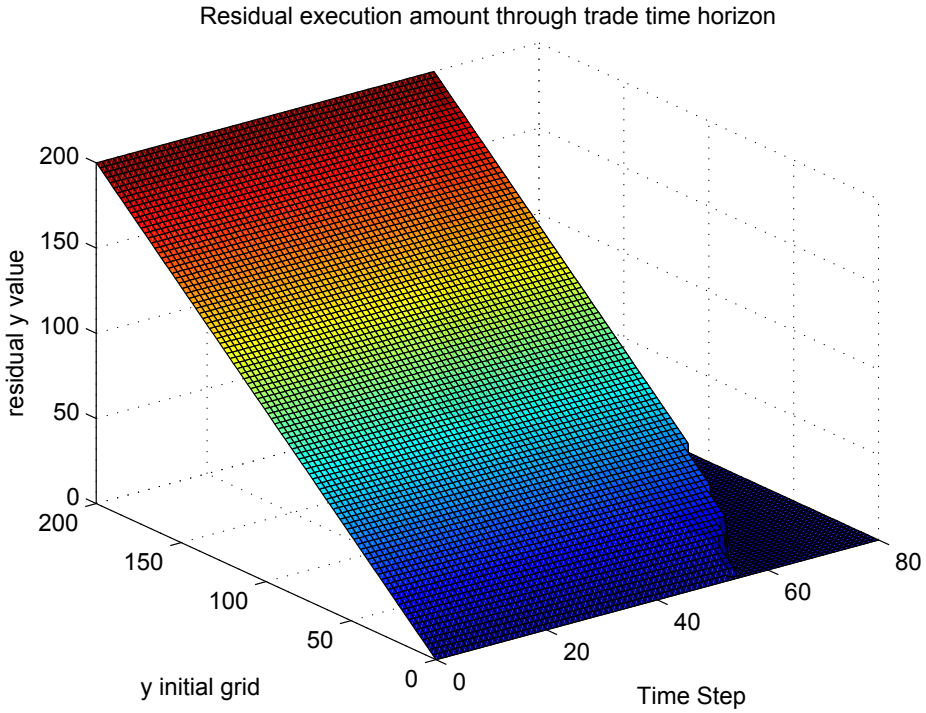


Figure 2.8: Daily Trading Strategy Simulation

We could use control variate method to manipulate various variables and compare different outcomes. For example, we could manipulate volatility σ and the price impact parameters. Since the optimal strategy in some sense is an optimal balance between these two aspects of influence, manipulating their value will create different optimal trade execution strategy. In terms of accuracy improvement, we could increase number of y grid points or implement other methods to improve the accuracy of regression coefficients.

2.4 Further Manipulation and Improvement

In this section, we discuss some alternative manipulation and interesting issue within our algorithm framework.

2.4.1 Goal Function

All our implementation so far focus on the simple goal function, namely effective payoff. However according to [14], several types of optimal criteria exist. A common form of optimality criterion seeks to maximize expected revenue while minimize a risk criterion, for example, variance, quadratic variation, or value-at-risk. A second approach maximize the expected value of utility function of revenues, for example, a power-law or exponential utility function. The third approach minimize expected execution cost through modeling dynamic distribution of bid and ask orders in a limit order book. For all these different approaches, different goal functions should be designed. Here we stick to our effective payoff due to its linear feature and intuitive criterion by which we can easily judge the legitimacy of our model.

2.4.2 Basis Function

To increase accuracy, as discussed in chapter 1, we can increase number of polynomial functions. Another interesting manipulation, suggested by [8], is that we could use polynomial on patches instead of global polynomial functions. In addition, for certain patches we could increase the number of grid points and number of polynomial functions. Results in [8] prove this method results in a dramatic improvement when applied to gas storage problem which shares a similar problem formation features with ours.

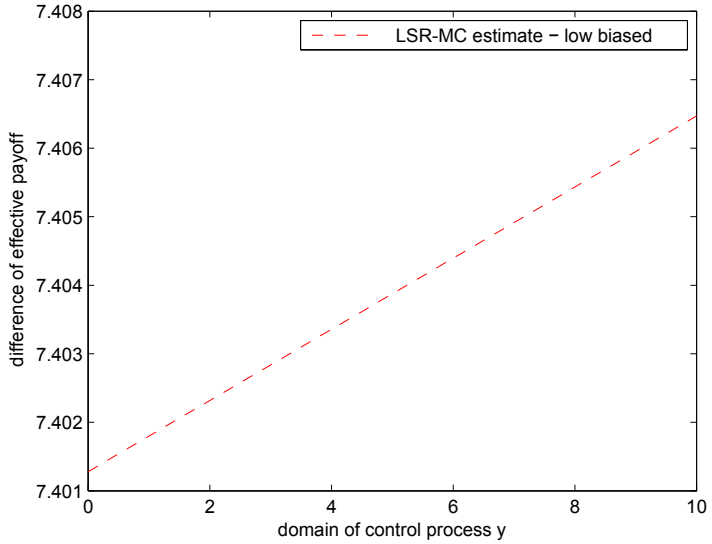


Figure 2.9: Difference Between LSR-based MC estimation Estimate and Low-biased Estimate

2.4.3 A Low-biased estimate

Apart from the LSR-based MC estimation which might be a high-biased one, we could also derive a low-biased estimate. The method is simple, we store the regression coefficients derived from one set of trajectories and implement the algorithm again with the derived coefficients and another set of trajectories. Since we are not using later information this time, the solution should be suboptimal hence it is a low-biased estimate of value function. Note here low-biased as it is called, it could be higher than actual value because of the various sources of errors

Since the difference is subtle compared to effective payoff value, we display the absolute difference instead of two estimates in figure 2.9.

2.4.4 Calibration to Real Market

To implement this algorithm in real market, the estimation of different parameter is of paramount importance. Though calibration to real market falls beyond the scope of this thesis, we briefly discussed it here and recommend some reference.

While calibration of σ is not a demanding task, the choice of price impact model and calibration of price impact parameter could be really tricky. Function $g(v)$ and $f(v)$ may be chosen to reflect any preferred model of market microstructure, subject only to certain natural convexity conditions. Even with the model selected, the calibration of parameter could be difficult. [3] discusses the calibration of parameters

within CITI proprietary trading data set. [10] suggests a particular limit order book model for resilient market. [12] displays some statistic property and calibration tips. Most calibration involves statistics techniques and data cleaning method. There could be an interesting further extension of our algorithm framework based on a specific data set, catering to particular market.

Chapter 3

Conclusion and Final Remark

In this thesis, we develop a Monte Carlo based algorithm to solve the optimal liquidation problem through extending the least squares regression method to calculate conditional expectation within a dynamic programming problem setup. We evaluate the performance of algorithm implementation through some basic testing and simulation within a reasonable artificial market. Moreover we discuss some manipulation and possible improvement of algorithm implementation.

Optimal liquidation via Monte Carlo method is an alternative to continuous modeling via HJB equation and PDE approach. Many papers, such as [4], [7], [13], [1] focus on this issue and obtain some interesting results. These approaches, however, are more applicable to higher dimension problems such as price impact with moderately long memory. In terms of Monte Carlo based algorithm, with careful calibration of parameters to real market, it provides good simulation of trading strategy and initial value of value function.

As mentioned at the end of chapter 2. A possible extension of this work would be to calibrate the model and do some in-sample and out-of-sample test and evaluate the optimal trading strategy. Another possible extension of this work is to focus on error estimation. Throughout the thesis we omit the error estimation part and concertrate on model construction. Finally, one can implement MC variance reduction techniques, such as importance sampling or control variables method within this framework to improve accuracy of the result.

References

[1] N. Aleksandrov and B. M. Hambly. Liquidity modelling and optimal liquidation in bond markets. *preprint*, 2010.

[2] R. Almgren and N. Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, 3:540, 2001.

[3] R. Almgren, C. Thum, E. Hauptmann, and H. Li. Direct estimation of equity market impact. *Risk*, 18(7):5862, 2005.

[4] Clive Bowsher. Modelling security market events in continuous time: Intensity based, multivariate point process models. October 2006.

[5] Stavros Christodoulou. Monte carlo based numerical pricing of multiple Strike-Reset options. http://eprints.maths.ox.ac.uk/1372/, June 2011.

[6] P. A Forsyth, J. S Kennedy, S. T. Tse, and H. Windcliff. Optimal trade execution: a mean quadratic variation approach. *Quantitative Finance*, 2009.

[7] Peter A. Forsyth. A hamilton jacobi bellman approach to optimal trade execution. *Applied Numerical Mathematics*, 61(2):241–265, February 2011.

[8] Lajos Gergely Gyurko, Ben Hambly, and Jan Hendrik Witte. Monte carlo methods via a dual approach for some discrete time stochastic control problems. *eprint arXiv:1112.4351*, December 2011.

[9] Paul Glasserman. *Monte Carlo Methods in Financial Engineering (Stochastic Modelling and Applied Probability)*. Springer, 1 edition, August 2003.

[10] J. Large. Measuring the resiliency of an electronic limit order book. *Journal of Financial Markets*, 10(1):125, 2007.

[11] F. A. Longstaff and E. S. Schwartz. Valuing american options by simulation: A simple Least-Squares approach. *Review of Financial Studies*, 14(1):113–147, January 2001.

[12] Marc Potters and Jean-Philippe Bouchaud. More statistical properties of order books and price impact. *arXiv:cond-mat/0210710*, October 2002.

[13] Alexander Schied and Torsten Schöneborn. Risk aversion and the dynamics of optimal liquidation strategies in illiquid markets. *Finance and Stochastics*, 13(2):181–204, 2009.

[14] S. T. Tse, P. A. Forsyth, J. S. Kennedy, and H. Windcliff. Comparison between the mean variance optimal and the mean quadratic variation optimal trading strategies. *Preprint*, 2011.

[15] J. Wang and P. A. Forsyth. Comparison of mean variance like strategies for optimal asset.

Appendix A

Appendix MATLAB Code

In this appendix we present the MATLAB code used for algorithm implementation and results obtained in chapter 2.

Function PathGenerator takes initial grid S_0, interest rate r, time scale T, volatility σ and number of time steps M as input. It generates a price matrix with each row representing a trajectory starting from a particular initial point, and each column corresponding to time steps. Note here we use geometric Brownian motion and its explicit solution to generate trajectories.

```
% This function outputs the trajectories and Brownian increments
% for each intial point in S0, N number of trajectories is
  generated from this initial value
% S0          intial stock price grid at t=0, a size N vector
% M           time scale of the trajectory
% r           risk-free rate
% T           time boundary
% sigma       volatility in model
% The output S is a (N*n,M+1) matrix, i*N row stores the N
  trajectories generated given the ith initial point in S0, each
  column is corredponding to a different time step

function [S]=PathGenerator(S0,r,T,sigma,M)
N=length(S0);
dt=T/M;
vol=sigma*sqrt(dt);
S=zeros(N,M+1);
S(:,1)=S0;
Z=zeros(N,1);
drift=dt*(r-0.5*sigma^2);
for i=1:M
Z=randn(N,1)*vol;
S(:,i+1)=S(:,1).*exp(drift+Z);
end
```

YGrid function generates $\mathcal{G}_t^y$ use quasi-Monte Carlo method. Given upper and lower boundary of Y, it generates a matrix with each row corresponding to a set of

$\mathcal{G}_t^y$. The matrix has the same number of row as trajectory matrix generated above.

```
% This function generate y grid use rank-1 lattice rule, the
  output is (N,Y) matrix
% Y          number of grid point needed
% y1,y2      boundary of parameter y
% N          total number of trajectories

function ygrid=YGrid(N,Y,y1,y2)
y2=y2-(y2-y1)/Y;
ygrid=((y2-y1)/(Y-1)*rand(N,1))*ones(1,Y)+ones(N,1)*linspace(y1,y2
  ,Y);
```

BasisFunction takes all trajectories at a particular time step and a y matrix as input. It uses the form depicted in chapter 1 as basis function to calculate its value.

```
% This function is to generate value of basis function given
  specific x,y
% Test function are 1,x,y,x^2,y^2,x*y
% x          (N,1) vector, N could be 1
% y          (N,Y) matrix, N could be 1, Y could be 1, N should be
  the same
% The output is (N,Y,q) matrix, q is the number of basis function

function basis_function=BasisFunction(x,y,q)
N=size(y,1); Y=size(y,2); basis_function=zeros(N,Y,q);
xextend=repmat(x,1,Y);
basis_function(:,:,1)=ones(N,Y);
basis_function(:,:,2)=xextend;
basis_function(:,:,3)=y;
basis_function(:,:,4)=xextend.*xextend; basis_function(:,:,5)=y.*y
  ;
basis_function(:,:,6)=xextend.*y;
```

BasisFunction_PriceImpact has the same structure as BasisFunction. The only difference is that it incorporates permanent price impact with parameter κ_p. This function is use to calculate $G(x, y)$.

```
function basis_function=BasisFunction_PriceImpact(x,y,q,h,kapa_p)

N=size(y,1);
Y=size(y,2);
basis_function=zeros(N,Y,q);
xextend=repmat(x,1,Y)-repmat(x,1,Y).*h*kapa_p;
basis_function(:,:,1)=ones(N,Y);
basis_function(:,:,2)=xextend;
basis_function(:,:,3)=y;
basis_function(:,:,4)=xextend.*xextend;
basis_function(:,:,5)=y.*y;
basis_function(:,:,6)=xextend.*y;
```

GoalFunctionSimple takes temporary price impact factor as input and use effective payoff as goal function. Note temporary price impact is incorporated in this function,

denoted by f.

```
% This function is to generate value of the goal function given x,
  h
% x          (N,1) vector, N could be 1
% h          (N,Y) matrix, N could be 1, N should be the same
% The output is (N,Y) matrix
% Here the value of goal function has already incorporated
  temporary price impact

function goal_function=GoalFunctionSimple(x,h,dt,kapa_s,kapa_t,
  beta,lambda)
N=size(x,1);
Y=size(h,2);
v=-h/dt;
% here f is the matrix of temporary factor, (N,Y)
f=(1+kapa_s*sign(v)).*exp(kapa_t*sign(v).*(abs(v).^beta));
% reassign post-trade underlying value to x
goal_function=repmat(x,1,Y).*f.*h;
```

RegressionCoefficient implements algorithm described in chapter 1. It takes $\mathcal{G}_t^{xy}$, V_{t+1} and S_t as input and generates regression coefficient used to calculate V_t recursively.

```
% This function is to calculate the regression coefficient for
  each time step, the output gamma, regression coefficient, is (1,q
  ) vector
% basis_function (q,1) vector, basis function for regression
% value_function V function in formula,a number given x,y
% Sx       generated trajectories of price x,(N,2) matrix, first
  column contains price at t, second              price at t+1
% ygrid      generated by quasi-MC(eg, lattice rule), (N,Y)matrix,Y
   is grid number for variable y
% V1         Value function at time t+1,(N,Y) matrix
% q          number of basis function, fixed according to
  BasisFunction

function [gamma]=RegressionCoefficient(ygrid,Sx,q,V1)
BVphi=zeros(q,1);
Bphi=zeros(q,q);
N=size(Sx,1);
Y=size(ygrid,2);
basis_function=BasisFunction(Sx(:,1),ygrid,q);
for i=1:q
    BVphi(i)=sum(sum(V1.*basis_function(:,:,i)))/(N*Y);
    for j=1:q
        Bphi(i,j)=sum(sum(basis_function(:,:,i).*basis_function
          (:,:,j)))/(N*Y);
    end
end
gamma=(Bphi\BVphi)';
```

Function Main is the numerical algorithm framework described in chapter 1. It

takes all the variables and generates a 3-dimension value function matrix, with z denoting each time step and $xy-plane$ displaying value for all points at $\mathcal{G}_t^{xy}$ at particular time step t. It also stores regression coefficient gamma at each time step to implement further low biased estimate.

```
% This is the main function to calculate value funtion from time T
   to 0
% S0          intial stock price grid at t=0, a size n vector
% M           time scale of the trajectory
% r           risk-free rate
% T           time boundary
% sigma       volatility in model
% yf          y fine grid number
% y1,y2       boundary of y
% q           number of basis function
% kapa_p      permanent price impact parameter

function [V gamma]=Main(S0,r,T,sigma,M,Y,yf,y1,y2,q,kapa_p,kapa_s,
  kapa_t,beta,lambda)
S=PathGenerator(S0,r,T,sigma,M);
N=size(S0,2); ygrid=YGrid(N,Y,y1,y2);
yfinegrid=linspace(y1,y2,yf);
V=zeros(N,Y,M+1);
V(:,:,M+1)=GoalFunctionSimple(S(:,M+1),zeros(N,Y),T/M,kapa_s,
  kapa_t,beta,lambda);
gamma=zeros(M,q);
% loop backward for time scale for
i=M:-1:1
    % take Vt+1 value from value matrix, input for regression
    V1=V(:,:,i+1);
    % take St and St+1 value from price function, input for other
      function
    Sx=S(:,i:i+1);
    % get regression coefficient,(1,q)
    gamma(i,:)=RegressionCoefficient(ygrid,Sx,q,V1);
    % at each time step, loop for y grid to find sup according to
      algorithm
    for j=1:Y
    % calculate h_input for goal function, i.e. y-y_hat, here use
      the fine grid, h_input match the  input h of GoalFunction, (N
      ,yf)
    h_input=repmat(ygrid(:,j),1,yf)-min(repmat(ygrid(:,j),1,yf),
      repmat(yfinegrid,N,1));
    % calculate value of basis function given vector of x, and
      matrix of x, (N,yf,q)
    phi=BasisFunction_PriceImpact(Sx(:,1),min(repmat(ygrid(:,j),1,
      yf),repmat(yfinegrid,N,1)),q,h_input  ,kapa_p);
    % generate G(x,y) by incorporating gamma and phi,(N,yf)
    G=zeros(N,yf);
    for k=1:q
        G=G+gamma(i,k)*phi(:,:,k);
    end
    % targetfuntion=G(x_PI,y)+H(x,h), (N,yf)
```

```
    target=G+GoalFunctionSimple(Sx(:,1),h_input,T/M,kapa_s,kapa_t,
      beta,lambda);
    % find sup for each row, (N,1)
    V(:,j,i)=max(target,[],2);
    end
end

figure (1);
i=5000;
plot(ygrid(i,:),V(i,:,1),'-k',ygrid(i,:),V(i,:,floor(M/2)),'-.r',
  ygrid(i,:),V(i,:,M),'--b');
xlabel ('domain of control process y')
ylabel ('domain of value function')
legend ('t=0','t=M/2','t=M','location','northwest')

figure (2);
surf(ygrid(i,:),S0,V(:,:,1));
shading interp;
ylabel('domain of x')
xlabel('domain of control process y')
zlabel('domain of effective payoff')
```

Function Main_ht is to generate optimal trade execution scheme given regression coefficients. It takes all the variables together with regression coefficients and generate optimal trade execution strategy for a fixed inital X_0 and various y which denotes the amount of trade to be executed at intial time.

```
% This is the main function to simulate the optimized trade
  strategy
% S0          intial stock price grid at t=0, a size n vector
% M           time scale of the trajectory
% r           risk-free rate
% T           time boundary
% sigma       volatility in model
% yf          y fine grid number
% y1,y2       boundary of y
% q           number of basis function
% kapa_p      permanent price impact parameter

function [res,H,V]=Main_ht(S0,r,T,sigma,M,Y,y1,y2,q,kapa_p,kapa_s,
  kapa_t,beta,lambda,gamma)
S=PathGenerator(S0,r,T,sigma,M);
N=length(S0);
yf=Y;
ygrid=YGrid(N,Y,y1,y2);
yfinegrid=linspace(y1,y2,yf);
H=zeros(N,Y,M);
V=zeros(N,Y,M+1);
V(:,:,M+1)=GoalFunctionSimple(S(:,M+1),zeros(N,Y),T/M,kapa_s,
  kapa_t,beta,lambda);
% loop backward for time scale
for i=M:-1:1
```

```
    % take St and St+1 value from price function, input for other
      function
    Sx=S(:,i:i+1);
    % get regression coefficient,(1,q)
    % at each time step, loop for y grid to find sup according to
      algorithm
    for j=1:Y
    % calculate h_input for goal function, i.e. y-y_hat, here use
      the
    % fine grid, h_input match the input h of GoalFunction, (N,yf)
    h_input=repmat(ygrid(:,j),1,yf)-min(repmat(ygrid(:,j),1,yf),
      repmat(yfinegrid,N,1));
    % calculate value of basis function given vector of x, and
      matrix of x, (N,yf,q)              phi=
      BasisFunction_PriceImpact(Sx(:,1),min(repmat(ygrid(:,j),1,yf)
      ,repmat(yfinegrid,N,1)),q,h_input  ,kapa_p);
    % generate G(x,y) by incorporating gamma and phi,(N,yf)
    G=zeros(N,yf);
    for k=1:q
        G=G+gamma(i,k)*phi(:,:,k);
    end
    % targetfuntion=G(x_PI,y)+H(x,h), (N,yf)
    target=G+GoalFunctionSimple(Sx(:,1),h_input,T/M,kapa_s,kapa_t,
      beta,lambda);
    % find sup for each row, (N,1)
    [V(:,j,i),pos]=max(target,[],2);
    H(:,j,i)=pos;
    end
end
res_h=zeros(Y,M+1);
for m=Y:-1:1
    res=zeros(1,M+1);
    loc=m;
    res(1)=ygrid(loc);
    for l=2:M
        loc=H(:,loc,l-1);
        res(l)=yfinegrid(loc);
    end;
    res_h(m,:)=res;
end;
figure (3);
surf((0:1:M),ygrid,res_h);
xlabel('Time Step')
ylabel('y initial grid')
zlabel('residual y value')
title ('Residual execution amount through trade time horizon')
end
```