NORTHWESTERN UNIVERSITY

Machine Learning in Option Markets

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Industrial Engineering and Management Sciences

By

Yaxiong Zeng

EVANSTON, ILLINOIS

September 2017

ABSTRACT

Machine Learning in Option Markets

Yaxiong Zeng

Machine learning has been widely applied to solve intricate problems in finance. Yet in options theory, machine learning methods are less visited due to the structural complexity of the derivatives market. This dissertation focuses on using machine learning algorithms to obtain optimal decisions for three distinct option-related problems. In the first chapter, we apply a reinforcement learning technique – approximate dynamic programming to model a real option of an investment in the renewable energy sector. The second chapter combines Q-learning and progressive hedging to attain optimal decisions for an option portfolio. In the third chapter, we model the implied volatility surface that exists in option markets by an online adaptive support vector regression algorithm. Each of the three chapters presents detailed case studies and numerical experiments that prove the effectiveness of our proposed models and algorithms.

Chapter 1 solves a real option problem of the investment timing of solar panels. Solar energy is rapidly emerging thanks to the decreasing installation cost of solar panels and the renewable portfolio standard imposed by state governments. Recently, third-party financing has become a common practice in solar panel investments. In this chapter, we discuss optimal timing for the host to potentially buy back the solar panels after being installed for a period of time and how to incorporate the optimal timing into a power purchase agreement between the host and the third-

party developer. By a modified real option structure, we model the buyback contract as a real option and solve it with an approximate dynamic program based Monte Carlo simulation method.

Chapter 2 focuses on financial options, in particular, a portfolio of American options. American options allow early exercise, which yields an additional challenge when optimizing a portfolio of American options, besides the weights of each option. We propose a reinforcement learning (Q-learning) algorithm for an American option portfolio, combining an iterative progressive hedging method and a quadratic approximation to Q-values by regression. By means of Monte Carlo simulation and empirical experiments we evaluate the quality of the algorithms proposed.

In Chapter 3, we design a machine learning based method – online adaptive primal support vector regression (SVR) – to model the implied volatility surface (IVS). The algorithm proposed is the first derivation and implementation of an online primal kernel SVR. It features enhancements that allow online adaptive learning by embedding the idea of local fitness and budget maintenance. To accelerate our algorithm, we implement its most computationally intensive parts in a Field Programmable Gate Arrays hardware. Using intraday tick data from the E-mini S&P 500 options market, we show that our algorithm outperforms two competing methods and the Gaussian kernel is a better choice than the linear kernel. Sensitivity analysis is also presented to demonstrate how hyper parameters affect the error rates and the number of support vectors in our models.

Acknowledgements

During the past five years, I am indeed fortunate to have received tremendous help from so many supportive and talented individuals. Hereby, I would like to pay special thanks to those who left positive influence on me throughout my years as a graduate student.

First and foremost, I would like to extend my sincere gratitude to my advisor, Professor Diego Klabjan. Under his guidance, I learnt how to conduct research and polished my way of critical thinking. His tolerance and patience of my missteps and his inspiration and wisdom that enlightened me along my PhD path were critical to the achievements that I accomplished during my PhD study. Without him, I would be nowhere near the fulfilment of PhD.

I would like to express my appreciation to Professor Vadim Linetsky, who led me into the world of finance. From his financial engineering courses, I absorbed plenty of ideas that are essential to my research projects. I would like to thank Professor Jorge Nocedal for his course machine learning, which opened a new door for me and stimulated my interest in pursuing machine learning applications in finance. I would like to thank my mentors during my internships, Eric Meschke, who showed me how exchange data was analyzed professionally and Tom Kalaway, who taught me how to develop ideas for trading. I am also grateful for my friends who enriched my life and provided me help: Dahai Liu, Yu Ang, Jiaming Miao, Lu Wang, Yi Gao, Yutian Nie.

Last but not least, I dedicate this dissertation to my parents – Jiangnan Song and Ming Zeng. I feel deeply indebted to them for their selfless support.

Table of Contents

List of Tables

List of Figures

CHAPTER 1

Distributed Solar Renewable Generation: Option Contracts with Renewable Energy Credit Uncertainty

1.1. Introduction

In recent years, global warming has been increasingly acknowledged as a threat to long-term human survival. Many countries have thus set up targets concerning emission limitations or reductions of greenhouse gases: the European Union aims at a 20% reduction below the 1990 baseline by 2020 (UNFCCC, 2008) and the United States offers a goal of a 17% reduction below the 2005 level by 2020 (US Department of Energy, 2009). To attain these goals, renewable energy technologies are being widely adopted to reduce the reliance of energy on fossil fuels.

In the United States, in 2011 renewable energy accounted for 9% of total primary energy consumption, with hydroelectric (35%), wood (22%), biofuels (21%) and wind power (13%) as major renewable sources (EIA, 2012). The solar market is now rapidly expanding as a result of historically high photovoltaic prices and by the financial incentives from the federal government, states and utilities. From the 2012 U.S Solar Market Insight report, photovoltaic installations totaled 3,313 MW, up 76% from 2011 with an estimated market value of $11.5 billion (SEIA/GTM Research, 2013).

One important aspect in the solar market is the Renewable Energy Credit (REC). As of 2013, the Renewable Portfolio Standard (RPS) is implemented in 30 U.S. states (including District of Columbia). Under such a policy, local utilities and load-serving entities are obligated to procure a specified fraction of their electricity as renewable energy. As a market response to RPS, the REC trading programs are initiated in most of the 30 states. Eligible renewable power producers receive a REC for each MWh of renewable energy generated. When electricity providers cannot meet the mandatory requirement from their own power facilities, they can in turn purchase RECs from renewable generators to comply with the RPS. The unit of REC price is $/MWh. Specifically, 17 out of the 30 states adopted detailed RPS targets to ensure solar power comprises a minimum fraction of the renewable mix, resulting in the creation and trading of the Solar Renewable Energy Credit (SREC). If the power supplier fails to obtain adequate credits, i.e. fails to meet the RPS, it is then subject to a penalty called the Alternative Compliance Payment (ACP) per MWh. In the solar case, the supplier is subject to the Solar Alternative Compliance Payment (SACP). Generally, SACP caps the SREC price. Otherwise, the obligated entity would prefer to pay the penalty, which is the mechanism of last resort to achieve compliance with the RPS. As the solar market continues to grow, the SREC trading program provides a driving incentive for home and business owners to install photovoltaic panels to satisfy their own electricity needs and financially benefit from selling SRECs.

In residential or commercial sites, direct ownership of solar panels is not a common practice. Instead, third party financing is taking off across the U.S. For instance, the 2012 U.S. Solar Insight Report pinpoints the ongoing third-party solar revolution. Specifically, over 50% of all

new residential installations in most major residential markets are from third-party-owned systems. The report also forecasts that the momentum will last. Usually, a third-party developer designs, installs, owns, operates and maintains solar panels on the user's roof and the user or host procures electricity from developer-owned solar panels. The user pays the developer according to a lease or Power Purchase Agreement (PPA). In the lease contract, home or business owners pay a monthly flat fee to the third-party developer. In the PPA, the host pays based on its electricity usage and according to a fixed rate or a rate with a fixed annual escalating factor. In both settings, the host does not pay for the panel installation or maintenance while it is the developer who incurs these costs. We particularly study the PPA setting, where the host buys all the electricity generated from the panels as negotiated in the contract.

Although the contract based on third-party financing can bring electricity with low and predictable costs, it prevents the host from making profits by selling RECs since they are owned by the third-party entity because they own the installation. The customer can buy back the panels at a later time (see NREL, 2009). However, when to buy back the panels has not yet been studied. Thus in this chapter, we discuss optimal timing of the buyback option by the host, in which the host buys the third-party owned solar panels at a particular time and price, and the REC ownership transfers from the developer to the host after panel buyback. The timing decision from our analysis is valuable to both the third-party developer and host. Grounded on the analysis, the two parties can develop the PPA and integrate the best timing decision into the contract. Ultimately, the buyback problem is a real option because it focuses on the exercise opportunity during a predetermined period.

Uncertainties involved in the buyback option result from fluctuations of REC prices, the electricity price, electricity demand by the user, and the value of the solar panels. As the REC markets continue to grow in the United States and worldwide, changing REC prices will have an increasing effect on the optimal investment timing decision. In this chapter, we first introduce a new financial model to forecast REC prices, which have a lower bound of zero and upper bound of the ACP value. Results show that our model outperforms the existing Geometric Brownian Motion (GBM) forecasting model. Forecasted REC prices are then incorporated in the cost-profit analysis of the solar investment. After the buyback option is exercised, the pattern of the cash flow changes due to REC sales. We thus model the investment timing problem as a real option by proposing a new option structure. We solve the model using a Monte Carlo simulation method based on approximate dynamic programming (ADP). In essence, our real option structure does not rely on sophisticated financial mathematics due to inherent complexity and can provide decision insights under different combinations of uncertainties.

In this chapter, we consider the host to be a non-power generating company. We only discuss solar projects, but the methodology can be adapted to other distributed renewable generations. For example, on-site wind generation is also applicable to our analytical framework, since it has an equivalent REC market and a similar third-party financing structure. The main contributions of this chapter are as follows.

1. A new REC forecasting method that specifically takes ACP values into account.

2. A new real option framework that can handle different patterns of cash flows before and after the option is exercised.
3. A developer-host contract that explicitly considers optimal buyback timing, the first of its kind in distributed renewable generation.
4. A case study that solves a problem of a real-world company.

The structure of this chapter is as follows. Section 1.2 provides a literature review. Section 1.3 introduces the new REC forecasting model and compares it with existing models. Section 1.4 exhibits the model of the investment timing problem as a real option and it provides the solution methodology by means of the least square ADP algorithm. Section 1.5 discusses the results from the Monte Carlo simulation with three case studies. Section 1.6 draws conclusions and suggests relevant future work.

1.2. Literature Review

An important model in our work is the discounted cash flow (DCF) model. When assessing an investment project, the DCF model is chosen traditionally. The model discounts the cash flows of the project to present time. If the obtained net present value (NPV) is positive, the project is economically viable. Due to its simplicity, it leads to rigid managerial decisions. It assumes the future cash flows have no variability, and the decision is simply to invest now or abandon the investment. These two drawbacks contradict the current investment styles characterized by uncertainties and dynamic decisions. As a remedy, the concept of real option has been proposed, which has been researched in a variety of disciplines during the last three decades. Essentially, it

serves as an extension to the DCF model. Several textbooks have been published to comprehensively introduce the concepts, theories and methods in real option studies (Dixit and Pindyck, 1994; Trigeorgis, 1996; Amram and Kulatilaka, 1999; Mun, 2002).

Before year 2000, in the energy sector, most of the real option literature applies to the oil industry. Because of the deregulation of the electricity market in the mid 1990s, real option principles began to be widely adopted in the analysis of electricity relevant topics such as electricity markets and power system investments. However, it is not until the last decade that the real option theory has been applied to the renewable energy field. The first paper by Venetsanos et al. (2002) presents a framework to assess renewable energy projects by real options and illustrates the possible uncertainties under deregulated energy markets. The paper models a Greek wind energy project as a real option and evaluates it by the Black-Scholes model, different from our ADP approach.

There are two popular methods in renewable real option analyses: the partial differential equation (PDE) method and the dynamic programming (DP) method. Usually, the PDE method requires an advanced understanding of stochastic models and financial mathematics. It is also not easy to analyze a real option that allows an early exercise using PDE, similar to an American call option. As another method, DP follows a recursive pattern to optimize decisions that influence future cash flows. Unlike PDE, the DP approach makes intermediate values and decisions readily available (Fernandes et al., 2011). In DP applications, a full stochastic DP is sometimes used, but the simplest and most frequently used model is the lattice model. By calculating risk neutral

probabilities and up and down factors, it is possible to almost always identify the basic structure from any real option and construct a corresponding lattice model (Mun, 2002). For example, Munoz et al. (2009) use a trinomial lattice model to evaluate the option to invest in a wind power generation project and demonstrate the probabilities of "invest now," "wait," and "abandon," while we only consider one option over the entire time horizon – in which year should we invest. The main problem with the lattice model is the difficulty to deal with multiple uncertainties. It needs multiple variables to represent different uncertainties, and thus the number of nodes grows exponentially with the dimension of uncertainties. Furthermore, to approximate the stochastic process accurately, it requires a high-dimension tree or infinitesimal time intervals. These two factors make it computationally expensive in the attempt to model complex problems with several uncertainties and to retain a satisfactory level of approximations of endogenous stochastic processes.

As a result, we adopt a Monte Carlo simulation and optimization method to solve our model. Unlike the PDE method, it requires less mathematical sophistication and can easily handle early exercise situations. In contrast to the lattice model, it better copes with stochastic processes, regardless of the dimension of uncertainties. In this study, we integrate Monte Carlo simulation with ADP. It differs from the lattice model mainly in the fact that ADP can cope with a large number of decisions and complex stochastic processes. Overall, Monte Carlo simulation is widely used in real option, but less so in renewable energy. Martínez-Cesena and Mutale (2011) propose an advanced real option methodology for renewable energy generation project planning using a simulation approach for a hydropower case study. In comparison, we focus on the

flexibility of the investment decision because the exercise year is to be decided, rather than the flexibility of a project design.

In financial theory, Longstaff and Schwartz (2001) propose a simple least-square approach to value American options. The key contribution is their estimation of continuation values, i.e. the conditional expectation of payoff if the option is held. In each time period, the option holder compares the value of exercising the option and expected payoff when holding the option. If the exercise value surpasses the holding value, the holder exercises; otherwise, the option is held and exercised later. The holding value is based on a conditional expectation function that is generated from the regression of ex post realized payoffs from continuation as functions of the values of the state variables. The proposed algorithm falls in the category of ADP and is called least square Monte Carlo (LSM). Applications of LSM in real options have been studied in Rodrigues and Armada (2006). However, such a method has not yet been used in renewable real options. In this chapter, we focus on the Monte Carlo simulation and we specifically incorporate LSM in finding the optimal exercise time.

Solar-related literature is the scarcest among renewable applications of real options (Fernandes et al., 2011). Four papers relate to our work in terms of the solar investment timing. The most relevant is Ashuri et al. (2011), who study the best timing to invest in photovoltaic panels on solar ready buildings. Ashuri et al. and our work both focus on the timing decision to install solar panels of a stakeholder such as a company or homeowner. They assume the price of solar panels decreases over time according to the experience curve, while we make a different assumption

that the price decreases according to market fair values. Ashuri et al. focus on savings from solar panels while we concentrate on the minimization of the total NPV over the entire time horizon. The uncertainty in their model is the electricity price, by which they calculate the savings from solar panels against procuring electricity from a utility, while our uncertainty is predominantly the REC price. They assume no cash flow before exercising the option, making it possible to use the lattice model. But cash flow is captured throughout the future horizon in our model, and has different patterns before and after the option exercise. They use a binomial lattice to forecast electricity prices and by means of decision trees they determine the optimal installation time, while we use a Monte Carlo simulation method combined with ADP. In summary, our model is more general and realistic under current market conditions.

Beliën et al. (2013) also consider the optimal timing for a solar panel investment and follow a similar method to Ashuri et al. (2011) to construct the model, and only differ in that they consider more cost and revenue factors, and solve the problem using the future value analysis. Sarkis and Tamarkin (2008) also consider savings as profit, not from electricity, but from greenhouse gas emissions. They also adopt the lattice model to solve the timing problem. Martinez-Cesena et al. (2013) from a savings point of view discuss about optimal timing using the economics concept of an indifference curve. These four papers have a similar cash flow structures and similar uncertainties. They differ mainly in the calculations of savings and the methods that solve the models. Unlike these studies, we propose a different real option structure, we combine multiple uncertainties with the focus on REC price volatility and solve the model with a different method.

An important building block of this chapter is REC price forecasting. To the best of our knowledge, only one paper, Tang et al. (2012), discusses a forecasting method. The paper provides three stochastic models for price processes: Geometric Brownian Motion (GBM), Jump Diffusion and NGARCH processes. By a statistical comparison they choose GBM as the appropriate model. When forecasted prices exceed the ACP value, they simply cap the price by the ACP. This can be problematic because it is possible for GBM to exceed the ACP for a long time; and thus during that period, REC prices are constants (equal to ACP), which obviously deviates from the market trend. In this chapter, we provide a forecasting model that in particular accounts for the ACP value.

1.3. REC Forecasting Model

REC is a market response to RPS. It helps electricity providers to meet the RPS by purchasing certificates from the REC market. As a provider of RECs, a company equipped with solar panels should forecast the REC prices and incorporate them in its cash flow (REC sales) calculation. Forecasting of REC prices is needed in our real option model explained in Section 1.4, but it also has value as a standalone model in NPV analyses.

REC prices should have the ACP as the upper bound and zero as a natural lower bound. The ACP value, functioning as the penalty payment that the utility incurs when it fails to meet the RPS, varies across states. For example, New Jersey has a mature SREC market and has designed its 20-year SACP plan so that by 2028 it would have 4.1% of its electricity from the solar power.

However, California just initiated a tradable REC market in 2011, with a promotional REC price cap of $50 that will expire at the end of 2013. We base our study on the NJ SREC market and use its available monthly-basis data from 2004 to the present. 81 historical monthly average prices from April 2004 to May 2011 are used to calibrate parameters of forecasting models, while the remaining 20 prices from June 2011 to January 2013 are used to perform out-of-sample tests. The ACP value decreases over time from $711 in 2009 to $239 in 2028. For years before 2004 and after 2028, we assume the ACP is the same as in 2009 and 2028, $711 and $239 correspondingly.

To feature the variations of ACP values overtime, we introduce the artificial prices by dividing the actual REC prices by the ACP value at that time. It is the artificial prices on which we base our parameter estimation and price forecasting. To further obtain the forecasted REC prices, we simply multiply the artificial prices by corresponding ACP values.

We first introduce the GBM model that has been used to forecast REC prices in Tang et al. (2012). The process $\{S_t,\ t \geq 0\}$ defined by

$$S_t = S_0 e^{\mu t + \sigma B_t},\ t \geq 0 \tag{1.1}$$

is called a Geometric Brownian Motion with drift μ and volatility σ. The standard Brownian Motion B_t is characterized by the independent normal distributions with mean 0 and variance $t,\ t \geq 0$. In finance, S_t is considered as the asset price with S_0 being the initial price. It can be also expressed by the stochastic differential equation (SDE):

$$dS_t = S_t\left(\mu dt + \sigma dB_t\right),\ t \geq 0\,. \tag{1.2}$$

In the SDE formulation, dS_t is the price increment, dt is the time increment and dB_t is the increment for standard Brownian Motion. In particular, dB_t also follows a normal distribution with mean 0 and variance dt . By Ito's formula, (1.1) and (1.2) are equivalent.

To simulate the GBM process, we need to discretize the time horizon. Suppose we want to simulate a sample path of an asset price from [0,*T*]. We first divide the time interval into *N* equal time steps with the interval length $\Delta t = \frac{T}{N}$ and simulate a sample path $\left\{S_{t_i},\ i = 0,1,...,N\right\}$ with $t_i = i\Delta t$ and a starting price S_0:

$$S_{i+1} = S_i e^{\left(\mu - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\varepsilon_{i+1}\sqrt{\Delta t}},\ i = 0,1,...,N-1,$$

where ε_i 's are i.i.d. random variables generated from the standard normal distribution with mean 0 and variance 1. In each step, we basically simulate the return process $\frac{S_{i+1}}{S_i}$, and multiply it by the previous asset price S_i . To obtain parameters μ and σ , we apply the widely used maximum likelihood estimation method, based on historical prices.

A drawback of the GBM model is that even if the parameters are estimated accurately, prices generated by this model can often exceed the ACP upper bound. To confirm this, we performed the following experiment. We first simulated 110 sample paths (with 100 prices in one sample path) generated by the GBM model with fitted parameters $\mu = 0.2424,\ \sigma = 0.4972$ using historical REC prices in New Jersey since 2004. If the generated artificial price is greater than 1,

then it is considered to exceed the ACP value. Figure 1.1 illustrates the number of sample paths in which the sample paths exceed the ACP different times. From the plot, we observe that most sample paths exceed the ACP value more than once, which is not acceptable as it contradicts the market fundamentals. Even worse, some sample paths show more than 80% of the prices over the ACP! There are only 17% of sample paths with no or only one exceeded value. This occurs because the GBM model does not have an upper bound that should be fundamental to a REC forecasting model.

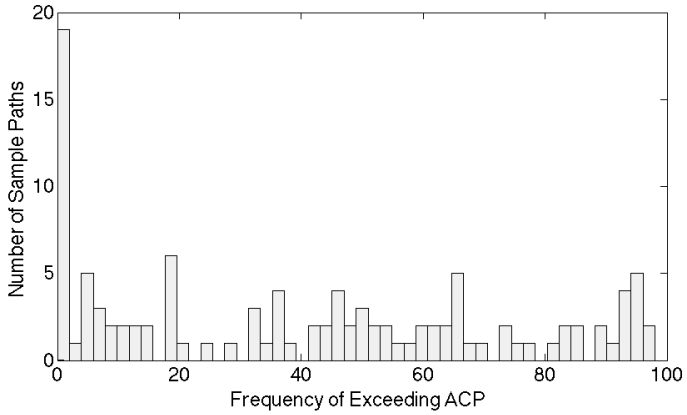


Figure 1.1: Histogram of the frequency of exceeding the ACP value

Next we propose a new forecasting model that is borrowed from financial engineering – Jacobi Diffusion (JD). Its stochastic differential equation is:

$$ds_t = -b(s_t - \beta)dt + \sigma\sqrt{s_t(1-s_t)}dB_t,$$

where $b > 0,\ \sigma > 0,\ 0 < \beta < 1$. Quantity b is the mean reverting parameter, β is the mean of the process, and σ is the volatility. Again, ds_t is the price increment, dt is the time increment and dB_t is the increment of standard Brownian Motion. In particular, s_t generated by this model lie between 0 and 1. To further obtain forecasted REC prices S_t, we simply multiply s_t by the ACP value. As

the ACP vary year by year, forecasted REC prices display different upper bounds (see Figure 1.2, whose details are discussed later).

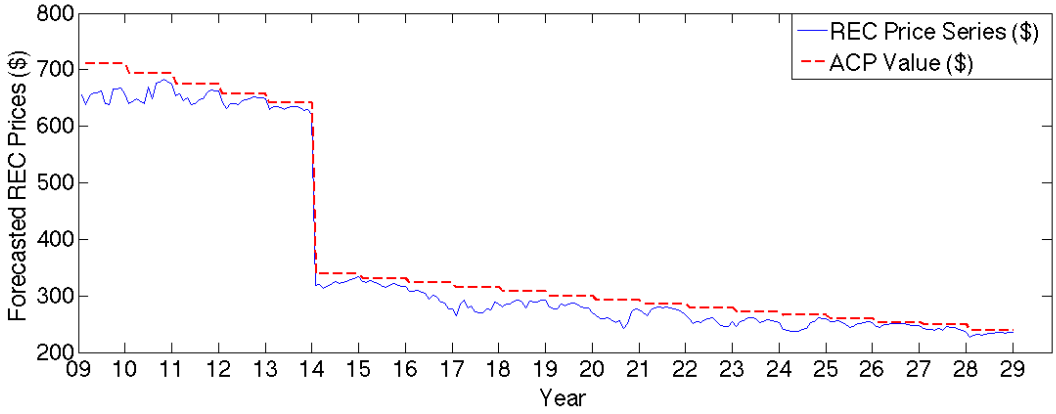


Figure 1.2: Simulated REC prices from the JD model

In simulation, discretization of the JD process is also needed. We follow the same structure introduced in the GBM part: $\Delta t = \frac{T}{N}$ and we simulate a sample path $\{S_{t_i},\ i = 0,1,...,N\}$ with $t_i = i\Delta t$ and starting price S_0 ($s_0 = S_0 / ACP_0$):

$$s_{i+1} = s_i - b(s_i - \beta)\Delta t + \sigma\sqrt{s_i(1-s_i)\Delta t}\,\varepsilon_{i+1},\ i = 0,1,...,N-1, \tag{1.3}$$

$$S_{i+1} = ACP_{i+1} \cdot s_{i+1},\ i = 0,1,...,N-1. \tag{1.4}$$

In essence, during each step we simulate the difference of the prices. Then we add the difference to the previous price and perform the multiplication to get the forecast. However, due to the discretization of the time, in a few cases s_i can be greater than 1 or less than 0, making it impossible to produce the next price. To prevent this, we force s_i to be 0.99 or 0.01 in respective cases. Finally, we follow the approximated maximum likelihood estimation method introduced in Gouriéroux and Valéry (2004) to estimate the parameters in the JD model. The fitted

parameters are $b = 0.11,\ \beta = 0.9085,\ c = 0.0808$ by using the same historical data in New Jersey as in the GBM case.

Unlike the GBM sample paths exceeding the ACP multiple times, the JD sample paths exceed the ACP at most once in all 100 sample paths due to the time discretization. To further assess the GBM and JD forecasts, we present the Quantile-Quantile (QQ) plot. The QQ plot is used to test how well the sample data fits a distribution. In Figure 1.3, the x-axis plots the quantiles of historical REC returns in NJ, and the y-axis plots the quantiles of the log returns of simulated data from GBM and JD processes. We use log returns to normalize the data and consequently we can compare the data over different time periods. From the QQ plots we observe that the points from the JD process lie closer to the straight line, indicating that the JD process fits the historical REC prices better.

As another piece of evidence, in an out-of-sample test, we find that the JD sample path behaves better than the GBM sample path. Out of the 101 historical NJ REC prices, we use the first 81 points to estimate parameters for the GBM and JD processes. Then we generate 20 prices separately from the two models and compare them with the remaining 20 historical prices. Because the price before the 81st month are near the ACP value, there is little room for the actual prices to go up and it is very likely that the price should exhibit a downward trend. The JD process captures this feature very well and shows a sharp contrast to the GBM model, which allows the prices to go up (Figures 1.4 and 1.5). All of the three comparisons show our forecasting method outperforms the GBM model.

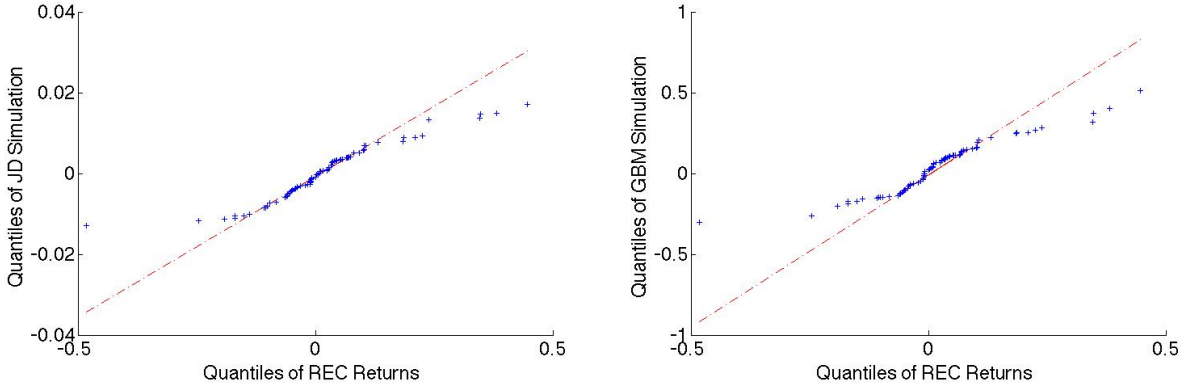


Figure 1.3: QQ plots of the JD and GBM models

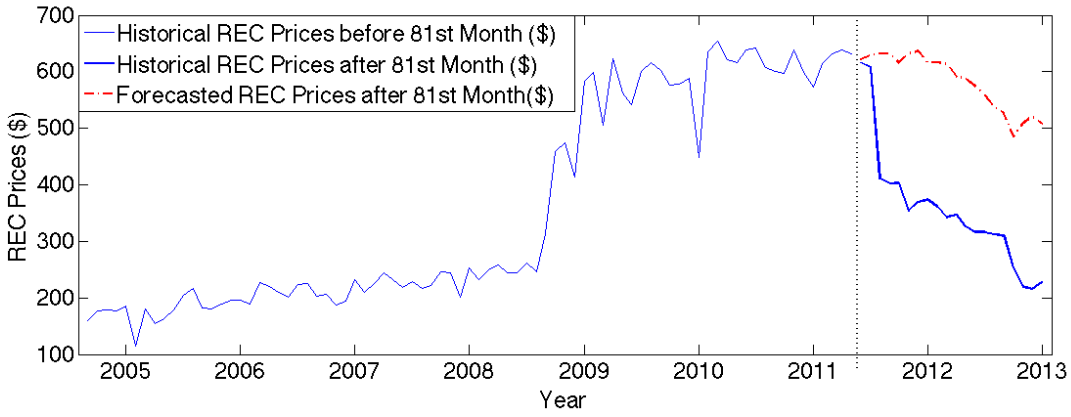


Figure 1.4: Out-of-sample test for the JD model

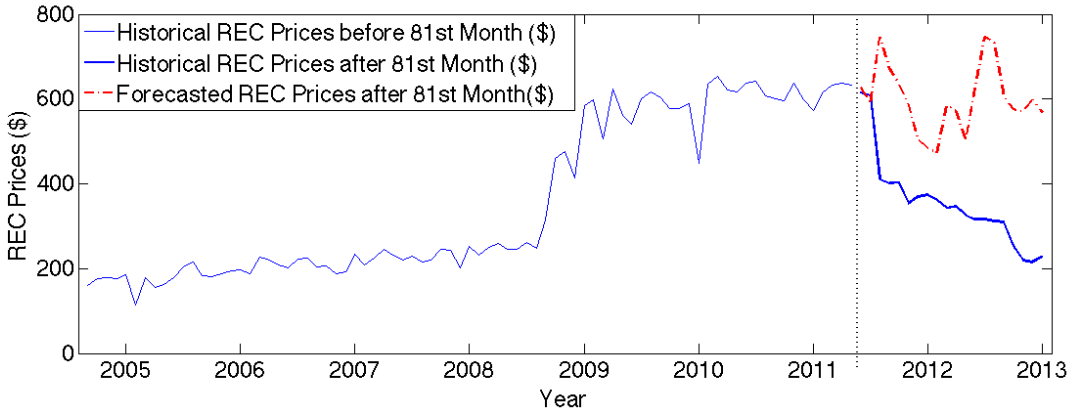


Figure 1.5: Out-of-sample test for the GBM model

1.4. Option Contract Modeling

As described in the introduction, we consider a host to be an industrial non-power generating company, considering a third-party developer to install, maintain and operate the panels on its property due to the high up-front investment cost of solar panels. In return, the company pays the charge based on the amount of electricity generated from the solar panels according to the PPA. If the panels cannot satisfy the company's power demand, it has to buy extra electricity from a utility. The third-party developer has a good reason to engage in such a contract: besides the regular profit by selling electricity to the company, the developer can also benefit from selling the RECs from the solar electricity and the tax incentives from the government. However, as time goes by, the value of solar panels declines. To minimize the company's overall cost, the company has an option to purchase the panels from the developer at a specified time in the future. We model this option as an investment contract. In the contract, the company has the option to buy the solar panels from the third-party developer for a predetermined price based on a buyback time. After the purchase, the company then owns the panels and associated RECs, and thus benefits from the REC market. The buyback option is analogous to the American call option. First, they both have a limited time horizon. The call option has an expiration date, while the buyback can only happen during the panels' lifespan. Second, they both emphasize the exercise opportunity. The call option exercises at a particular price prior to expiration while the company buys the panels at a given price before the panels turn obsolete. So the questions for the company are: (a) Should the panels be bought? (b) When should they purchase the panels in order to minimize the net cost NPV during the entire time horizon? To answer these questions, we model the buyback contract as a real option.

In a buyback contract for solar panels, the cash flows have different patterns before and after the purchase of panels. Before the purchase, the company has to incur the electricity cost from panels paid to the developer and to the utility for the remaining power. While after the purchase, the company only needs to pay the electricity cost to the utility, potentially pay the maintenance cost to the developer and receive profits by selling RECs and potential sales of redundant solar electricity. Specifically, we calculate all costs on a monthly basis, i.e. the time period is a month. We use the following notation

T - life span of solar panels (month)

G_m – electricity energy generated from solar panels in month m (kWh)

D_m – electricity energy demand in month m (kWh)

P_m – price of electricity from solar panels in month m (\$/kWh)

U_m – price of electricity from utility in month m (\$/kWh)

R_m – REC price in month m (\$/MWh)

W_m – one-time purchase price of solar panels in month m (\$)

M_m – maintenance cost of solar panels in month m (\$)

Usually, invertors of solar panels need to be replaced at the half of solar panels' life span and the replacement cost represents 10% of related investment cost. In our model, we incorporate the replacement cost into monthly maintenance cost with

$$M_m = \begin{cases} \overline{M}_m, & \text{if } m \neq \lfloor T/2 \rfloor \\ \overline{M}_m + M_{INV}, & \text{if } m = \lfloor T/2 \rfloor \end{cases},$$

where $\overline{M}_m$ is the regular maintenance cost and M_{INV} is the invertor replacement cost.

We make the following assumptions:

1. The life span T of solar panels is finite.
2. The amount G_m of electricity generated from the solar panels decreases on a yearly basis and we do not take into account any panel failures or variability of weather conditions that cause generation variability.
3. The price of solar electricity P_m escalates at a fixed rate per year.
4. The price of utility electricity U_m escalates at a fixed rate per year.
5. Panel purchase price W_m decreases according to the net profit the third-party developer expects to incur from the solar panels after the company's purchase, not according to experience curves.
6. The company is obligated to purchase all the generated electricity before owning the solar panels. If the generated electricity G_m exceeds the company's demand D_m, the current owner will sell it back to the grid at the current utility price U_m.
7. The ACP values are known in advance and thus deterministic.
8. After the buyback the company pays the developer to maintain the solar panels with maintenance cost M_m per month.
9. The company applies the buyback contract to mature factories or buildings so the electricity demand is deterministic over years (including peak demand).

10. The discount rate is constant. We use different discount rates when calculating NPVs for the company and the developer since they have dissimilar capital structures and thus different discount rates.

Note that G_m, P_m and U_m change on a yearly basis. For example, G_m decreases yearly based on fixed degradation rate a. It means that the value of G_m changes only for m representing January and in that month it decreases by $1 - a\%$ over the value in the previous year.

Under these assumptions, we next present the cost analysis. Before exercising the option, the company has to incur the following cost in month m:

$$C_m = G_m P_m + \left(D_m - G_m\right)^+ U_m .$$

After exercising the option, the company has to incur the following cost during each month:

$$B_m\left(R_m\right) = \left(D_m - G_m\right)U_m - G_m R_m + M_m .$$

At any time, the panel purchase price W_m is the NPV of the developer's future profit from selling RECs and sales of electricity to the company. Future maintenance costs after the purchase are subtracted from the purchase price in accordance with assumption 8, i.e.

$$W_m\left(\overline{R}_m\right) = E\left[\sum_{n=m+1}^{T} \gamma_d^{n-m}\left(\left(R_n + P_n\right)G_n - M_n\right)\middle| F_m\right], \tag{1.5}$$

where $\overline{R}_m = \left(R_{m+1}, R_{m+2}, ..., R_T\right)$, $\gamma_d^k = e^{-r_d k}$ is the continuous discount factor and r_d is the developer's discount rate. Note that r_d is used only here for the calculation of panel purchase price and the company discount rate is used elsewhere. By (1.5), the resulting optimal month can balance the profits of the developer and company.

Accordingly, the exercise value of the option in month *m* is

$$V_m^e\left(\overline{R}_m\right)=\sum_{n=1}^{m}\left(\gamma^n C_n\middle|F_m\right)+\gamma^m W_m\left(\overline{R}_m\right)+E\left[\sum_{n=m+1}^{T}\left(\gamma^n B_n\left(R_n\right)\middle|F_m\right)\right],$$

where $\gamma^k=e^{-rk}$ is the continuous discount factor, *r* is the company discount rate, R_k is F_k measurable, i.e. $F_k=\sigma\left(R_u, u\le k\right)$ and F_k contains the information up to time *k.* If R_m is a martingale, then the expectation can be computed by substituting the R_n 's by R_m and thus $V_m^e\left(\overline{R}_m\right)=V_m^e\left(R_m\right)$. Note that the value in month *m* is function of only $\overline{R}_m$, as we assume other variables can be computed without uncertainty. Let $V_m^h\left(\overline{R}_m\right)$ be the optimal cost if the optimal exercise time happens after month *m.* This implies that in month *m* the option is not exercised and thus this is the holding cost in month *m.* There does not exist a closed form expression for V_m^h.

The optimization problem that minimizes the overall cost is to find the latest month m_j^* for which

$$V_{m_j^*-1}^h\left(\overline{R}_{m_j^*-1}^j\right)\le V_{m_j^*}^e\left(\overline{R}_{m_j^*}^j\right)$$

for a realization $\overline{R}_{m_j^*-1}^j$ of REC prices. The selected final time is the average time of all m_j^* over all realizations *j*.

For such a cash flow form, it is difficult to apply traditional approaches of real options that treat cash flows in a consistent pattern. We propose a new option framework that addresses this difficulty.

We first define two artificial variables: the artificial strike price and the artificial stock price.

Artificial Strike Price K_m: - (total cost that has incurred before month m + panel purchase price in month m)

Artificial Stock Price S_m: + (total cost expected to incur after current month m)

The artificial strike price is known given the current month m while the artificial stock price is still undetermined. This resembles the structure of an American call option with fixed strike prices and volatile stock prices. In an American call option the option allows option holders to exercise the option, i.e. buy an agreed quantity of a particular stock from the seller at a predetermined price K at any time prior to and including its expiration date T. The value of exercising an American option is

$$(S-K)^+ = \max(0,S-K) = \begin{cases} 0, & S < K \\ S-K, & S \geq K \end{cases}.$$

The artificial value of our option is thus simply

(total cost expected to incur after current month m + total cost that has incurred before month m + panel purchase price in month m) = Total cost during the entire time horizon.

To summarize these concepts, we draw a comparison between the American Call option and our solar option in Table 1.1.

Table 1.1: Comparison of the American call option and the solar option

American Option	Solar Option
Maturity time T	Life span of the solar panel T
Stock price S_m at time m	Total cost the company expects to incur after month m
Strike price K_m at time m	Total cost the company has incurred before and in month m

Early exercising time m ($<T$)	Month to buy back the solar panels m ($<T$)
Option price if exercising at time m $=\max(0,S-K)$	Value of the project (the total cost during T) in month m $=V_m^e(R_m)$

Solving this model in traditional ways, such as by a PDE or the lattice model is not easy due to the inconsistent cash flow form. We thereby propose an ADP based Monte Carlo simulation method. The basic idea has been introduced in Longstaff and Schwartz (2001). The algorithm first simulates a series of realizations of sample paths and calculates relevant cash flows. Although exercise values can be easily computed, holding values in each month are difficult to obtain. The algorithm thus approximates holding values by least square regression and then performs DP backward recursion. The algorithm returns an optimal exercise time for each realization. Instead of pricing financial options, we fit it to our proposed solar option framework.

Let us first modify the exercise value by changing its discount rates. Then the value becomes

$$V_m^e(\bar{R}_m)=\sum_{n=1}^{m}\left(\gamma^{m-n}C_n\middle|F_m\right)+W_m(\bar{R}_m)+E\left[\sum_{n=m+1}^{T}\left(\gamma^{n-m}B_n(R_n)\middle|F_m\right)\right].$$

In this form, we consider the value to be the NPV in month m instead of at time zero. By our definition, the artificial strike price

$$K_m^j(\bar{R}_m^j)=W_m^j(\bar{R}_m^j)+\sum_{n=1}^{m}\left(\gamma^{m-n}C_n^j\middle|F_m\right), \tag{1.6}$$

and the artificial stock price

$$S_m^j(\bar{R}_m^j)=E\left[\sum_{n=m+1}^{T}\left(\gamma^{n-m}B_n^j(R_n^j)\middle|F_m\right)\right], \tag{1.7}$$

where superscript represents the j^{th} realization of REC prices. To be specific, each realization uses a distinct REC price sample path. We also assume artificial strike prices are positive for ease of computation.

As opposed to the simplicity of obtaining the exercise values in each month, it is relatively difficult to calibrate the holding value $V_m^h\left(\overrightarrow{R}_m\right)$ without knowing when to exercise the option. One remedy is to approximate the holding values. When implementing the LSM method proposed by Longstaff and Schwartz (2001), we use the following approximation formula,

$$\varepsilon^j + c + a_1 R_m^j + a_2 \left(R_m^j\right)^2 = \begin{cases} e^{-r\left(j_{e^*} - m\right)\Delta m} V_{j_{e^*}}^{j,e}\left(\overrightarrow{R}_{j_{e^*}}^j\right), & \text{if } j_{e^*} \le T; \\ 0, & \text{if } j_{e^*} = T+1. \end{cases} \tag{1.8}$$

where j_{e^*} is the optimal exercise time of realization j after current month m, ε^j is the corresponding regression error, and Δm is the time increment. In (1.8), the right-hand side is first computed and then approximated by $\varepsilon^j + c + a_1 R_m^j + a_2 \left(R_m^j\right)^2$ by means of regression. For each j, there is one regression observation, which yields coefficients a_1, a_2 and constant c obtained based on the current optimal exercise values in all realizations.

The method follows the classic idea of a backward algorithm with the starting month to be the last time period T. Next by substituting coefficients a_1, a_2 and constant c in the left hand side of (1.8) without the regression error ε^j, we have the approximated holding values

$$V_m^h\left(R_m\right) = c + a_1 R_m + a_2 \left(R_m\right)^2.$$

If the holding value is less than the exercise value, in other words, the cost is less if the option is held, then we go backward to the previous month and keep the current optimal exercise time j_{e^*} and value $V_{j_{e^*}}^{j,e}$. Otherwise, we update the optimal exercise time to the current month *m* and change the corresponding exercise value to V_m^e. We continue this procedure until we reach the first month. Essentially, in each month, the method compares the value of exercising now with the approximated optimal value of posterior exercising, i.e. the holding value. The algorithm is presented in Algorithm 1.1.

Algorithm 1.1 – The modified least square ADP algorithm

Initialization:

a. Generate N sets of REC price sample paths indexed by j.

b. Calculate artificial strike price $K_m^j\left(\overline{R}_m^j\right)$ and artificial stock price $S_m^j\left(\overline{R}_m^j\right)$ for all j, m.

c. Calculate the cash flows $V_m^{j,e}\left(\overline{R}_m^j\right) = K_m^j\left(\overline{R}_m^j\right) + S_m^j\left(\overline{R}_m^j\right)$ for all j, m.

d. Set $V_T^{j,h} = V_T^{j,e}$, $j_{e^*} = T$ for all j.

For *m*=*T*-1 to 1

e. Apply the least square regression based on N realizations and find values a_1, a_2 and c by solving

$$\varepsilon^i + c + a_1 R_m^i + a_2\left(R_m^i\right)^2 = e^{-r\left(i_{e^*}-m\right)} V_{i_{e^*}}^{i,e} \text{ for all } i=1,\ldots,N.$$

For *j*=1 to *N*

f. Set

$$V_m^h\left(R_m\right) = c + a_1 R_m + a_2\left(R_m\right)^2 \text{ and thus } V_m^{j,h} = c + a_1 R_m^j + a_2\left(R_m^j\right)^2.$$

g. Compare $V_m^{j,h}$ and $V_m^{j,e}$:

If $V_m^{j,h} \le V_m^{j,e}$, update $j_{e^*} = m$ and $V_{j_{e^*}}^{j,e} = V_m^{j,e}$.

End

End

The algorithm returns the optimal exercise time j_{e^*} for each realization. The average optimal exercise month and other statistics can be obtained from these.

1.5. Case Studies

In this section, we present three case studies. We first study the optimal exercise timing and the savings by buying back the panels at the optimal timing by considering only the REC prices as uncertainty. This strictly mimics the setting of our industrial partner. The second case study adds additional uncertainties to the model and the corresponding results are discussed. The third case study presents the sensitivity analysis using different parameter values. These case studies are based on data from a large US company where a project recently conducted is based on a fixed contract. The company assumed a fixed REC price given in advance and stationary during the planning horizon and optimally derived year 21 out of 30 years to buy back its solar panels.

By using stochastic REC prices and computing the difference of the total cost with exercise year 21, or the 252^{th} month, and the total cost with the optimal exercise month, we obtain the savings for incorporating the optimal timing. Note that we are using the same simulated REC prices to calculate the total costs of these two cases. We define this saving to be the value of our buyback option, which differs from the previously defined artificial option value, the total cost.

Next we introduce the setting of our case studies. The project is in California. Because its REC market just started without adequate data for statistical inference, we use historical REC prices from the New Jersey market instead to calibrate the parameters of the Jacobi process and forecast

REC prices. Note that we should first convert the historical REC prices into artificial prices by dividing the REC price by the corresponding ACP value in that year since our inference is based on the artificial prices ranging from 0 to 1. Then by approximated maximum likelihood estimation, we obtain the fitted parameters $b = 0.11,\ \beta = 0.9085, c = 0.0808$ for equation (1.3). To simulate the REC prices in California, we set the ACP value over the 30-year horizon to be constant at $25 since the state does not have a clear plan for ACP. Then by using (1.3) and (1.4), 5000 sample paths of REC prices are simulated.

The solar project input consists of solar panel generation output, electricity energy and demand and utility electricity rates based on 3-tier periods: on-peak, mid-peak and off-peak periods. The utility rate has three main parts: delivery, demand and energy. A monthly peak-demand cost from the utility is also included and is determined by the largest daily peak demand during mid- and on-peak periods. Besides, there are over 20 other parameters in the model. For example, the annual degradation rate of the solar panel generation output, the annual escalation rate of the solar electricity price, the starting price of solar electricity and many other tax exemption parameters. These parameters were given by the partner company. Invertor replacement cost is not considered in the case study since almost all sample paths have exercise time after year 15, which implies that inventor cost is occurred by the third party. By omitting it we also mimic our industrial partner's contractual settings.

Our methodology has been implemented in MATLAB on a Mac computer with 3.4 GHz Intel Core i7-2600 processor and 8 GB 1333 MHz DDR3 RAM.

We first assume that REC prices are the only source of uncertainty. Under a single source of uncertainty, i.e. the REC price, the total cost expected to incur after current month, i.e. the artificial stock price (1.7) first increases and then decreases (Figure 1.7), while the total cost that has been incurred, i.e. the artificial strike price (1.6) increases exponentially (Figure 1.6). The panel purchase price (1.8), decreases over time and reaches zero at year 27.5, or the 330^{th} month, out of the 30-year lifespan (Figure 1.8), which suggests the optimal timing should definitely be prior to this time as the panels can be bought for free and thus utilizing the potential panel benefits over the remaining periods. These costs comprise the basic values in determining the optimal timing. With the ACP value being equal to $25 over the entire time horizon, the average optimal exercise time is year 22.2, or the 266^{th} month. Figure 1.9 shows the histogram of the optimal exercise year.

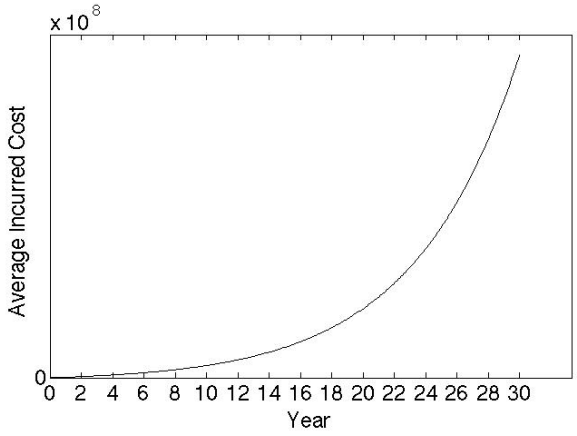


Figure 1.6: The artificial strike prices

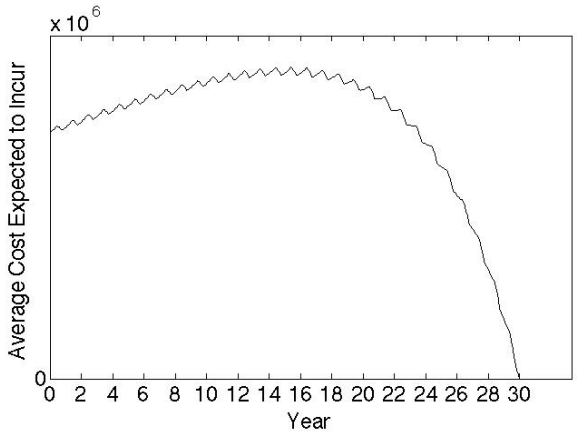


Figure 1.7: The artificial stock prices

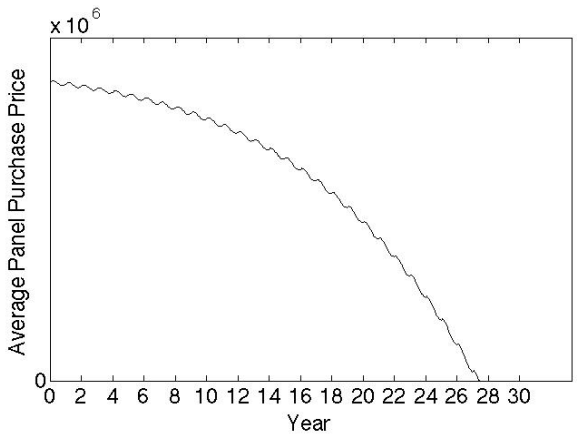


Figure 1.8: The panel purchase prices

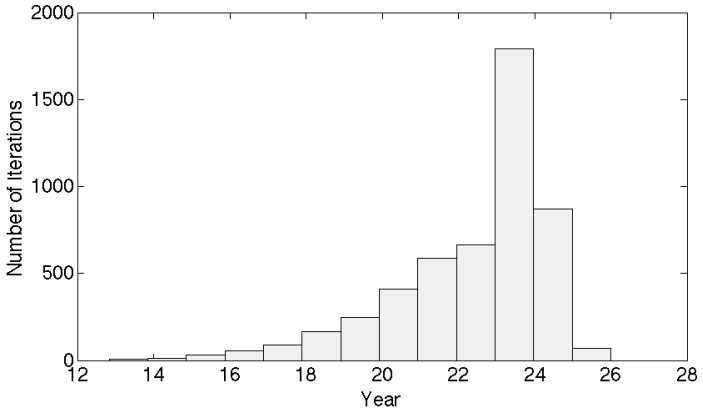


Figure 1.9: The histogram of optimal exercise year

We next discuss the impact of different ACP choices. We let the ACP values increase from a baseline of $20 to a maximum of $500. It turns out that the optimal year increases with the ACP but tends to stabilize when the ACP is close to or over $300 (Figure 1.10). Figure 1.11 shows that the value of the buyback option increases linearly. It should also be noted that the rate of savings, calculated by dividing the savings with the total cost in the fixed contract, increase from 2.55% when ACP value is $20 to a level of 10.76% when the value reaches $500 (Figure 1.12). Since the New Jersey market has a long history of high REC prices (over $500), the savings rate would be over 10% in the NJ case.

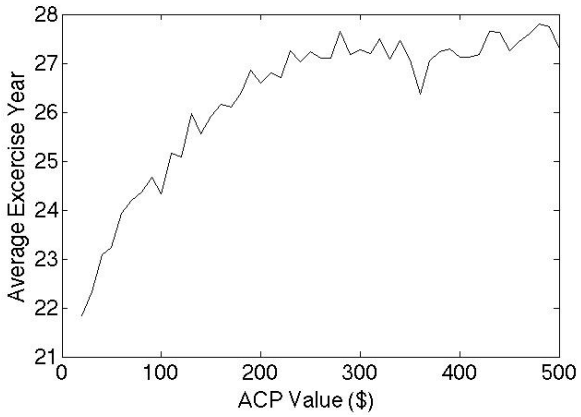


Figure 1.10: The optimal exercise years

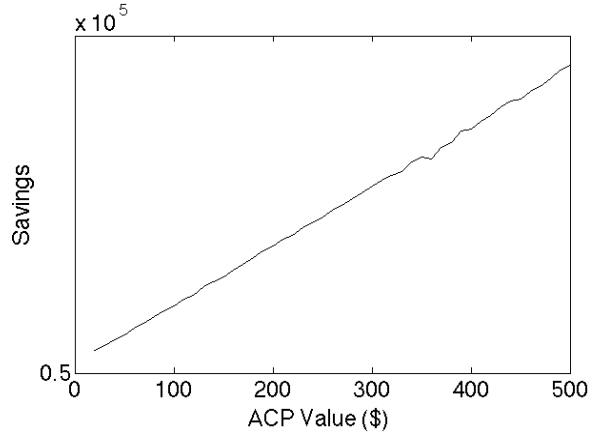


Figure 1.11: The values of the solar options

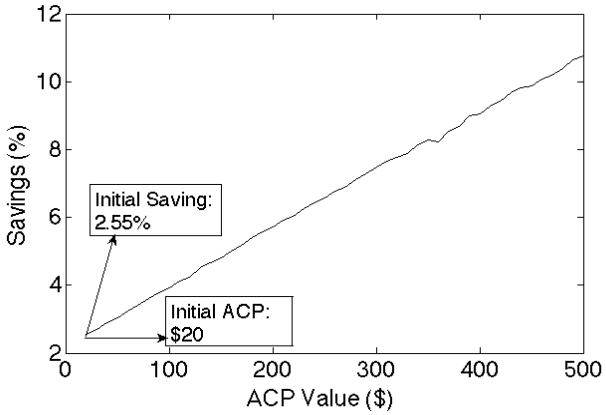


Figure 1.12: The percentages of money saved

So far we have only considered one uncertainty. Since electricity demand, utility electricity price and maintenance cost are not deterministic in reality, we next further assume that the electricity demand and maintenance cost in every time period follows a normal distribution centered at the fixed value used thus far with 10% volatility. We also assume that the escalation rate of the utility electricity price follows a uniform distribution between 1% and 2% every year.

To handle this situation, we have to augment the state space. Hence (1.6) changes to

$$K_m^j\left(\vec{R}_m^j,\vec{D}_m^j,\vec{M}_m^j,\vec{U}_m^j\right)=W_m\left(\vec{R}_m^j,\vec{D}_m^j,\vec{M}_m^j,\vec{U}_m^j\right)+\sum_{n=1}^{m}\left(\gamma^{m-n}C_n^j\middle|F_m\right),$$

and (1.7) changes to

$$S_m^j\left(\vec{R}_m^j,\vec{D}_m^j,\vec{M}_m^j,\vec{U}_m^j\right)=E\left[\sum_{n=m+1}^{T}\left(\gamma^{n-m}B_n^j\left(R_n^j,D_n^j,M_n^j,U_n^j\right)\middle|F_m\right)\right],$$

where $\vec{D}_m=\left(D_{m+1},D_{m+2},...,D_T\right)$, $\vec{M}_m=\left(M_{m+1},M_{m+2},...,M_T\right)$, $\vec{U}_m=\left(U_{m+1},U_{m+2},...,U_T\right)$.

Meanwhile, the regression equation in our algorithm also has to be modified. In particular, step e changes to

$$\varepsilon^i+c+a_1R_m^i+a_2\left(R_m^i\right)^2+x_1D_m^i+x_2\left(D_m^i\right)^2+y_1M_m^i+y_2\left(M_m^i\right)^2+z_1U_m^i+z_2\left(U_m^i\right)^2=e^{-r\left(i_e^*-m\right)}V_{i_e^*}^{i,e}$$

for all i=1,…,N, with six additional regression coefficients x_1,x_2,y_1,y_2,z_1,z_2. Step f also has to be changed in the same way.

After introducing these uncertainties, the basic trends of the optimal year and option value do not change substantially but display more variance (Figures 1.13, 1.14 and 1.15). It is still the case that the higher the REC prices, the higher the savings rate, from 3.3% when ACP is $20 to 10.69% when it reaches $500.

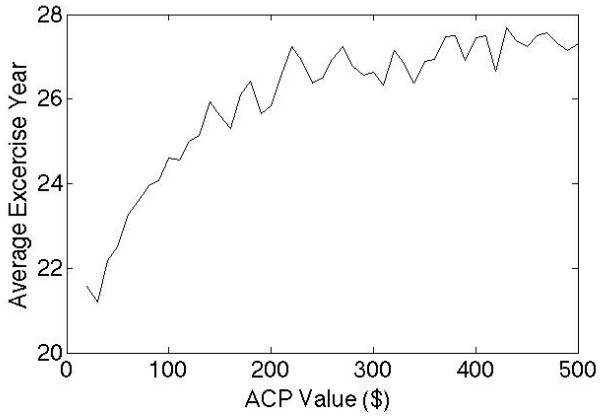


Figure 1.13: The optimal exercise years under the general scenario

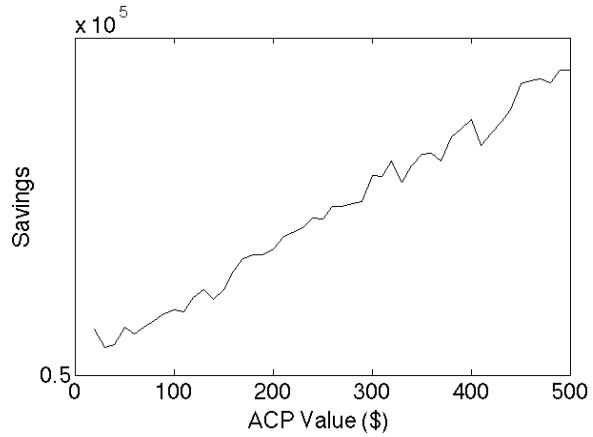


Figure 1.14: The values of the solar option

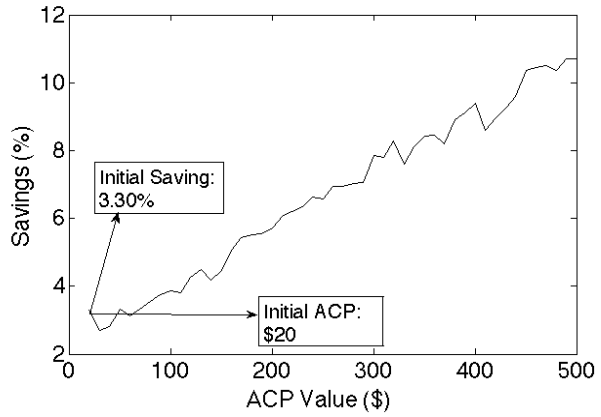


Figure 1.15: The percentages of money saved

The third case study explores the sensitivity of our results with regard to different parameter values. The following tables present the optimal year and savings rates with different input values of company and developer discount rates, annual degradation rate of the solar panel generation output and annual escalation rate of the solar electricity price. The savings rate is the percentage change in cost with respect to the baseline case. The results are obtained from the model with REC uncertainty.

Table 1.2: Optimal exercise years with different discount rates

Optimal Exercise Year						
	Company Discount Rate					
Developer Discount Rate		-2%	-1%	±0%	+1%	+2%
	-2%	23.8	24.3	24.5	24.8	25.2
	-1%	22.3	23.1	23.6	24.2	24.4
	±0%	19.9	21.3	22.2	23.0	23.4
	+1%	14.2	17.7	19.9	21.2	22.1
	+2%	1.2	9.2	14.7	17.7	19.9

Table 1.3: Savings rates with different discount rates

Savings Rate						
	Company Discount Rate					
Developer Discount Rate		-2%	-1%	±0%	+1%	+2%
	-2%	3.45%	3.18%	2.96%	2.77%	2.59%
	-1%	3.23%	2.98%	2.78%	2.59%	2.44%
	±0%	3.11%	2.83%	2.62%	2.46%	2.30%
	+1%	3.13%	2.78%	2.54%	2.34%	2.20%
	+2%	5.71%	3.13%	2.57%	2.31%	2.14%

Table 1.4: Optimal exercise years and savings rates with different solar electricity price escalation rates

Solar Electricity Price Escalation Rate	Optimal Exercise Year	Savings Rate
1%	19.3	2.10%
2%	22.2	2.62%
3%	24.4	3.30%
4%	25.7	4.11%
5%	26.9	5.05%

Table 1.5: Optimal exercise years and savings rates with different solar panel output degradation rates

Solar Panel Output Degradation Rate	Optimal Exercise Year	Savings Rate
0.5%	23.6	2.90%
1.0%	22.2	2.62%
1.5%	20.6	2.41%
2.0%	18.5	2.25%
2.5%	16.0	2.18%

Tables 1.2 and 1.3 exhibit the results when discount rates are perturbed by 1% or 2%. The optimal exercise year increases with increasing company discount rates while it decreases with increasing developer discount rates. The savings rate decreases with increasing company discount rates. With an exception of the +1% case (with a company discount rate perturbed by -2%) and the +2% case (with a company discount rate perturbed by less than or equal to 0%), it decreases with increasing developer discount rates. This exception is reasonable since +2% developer discount rate leads to more volatile exercise years and thus a larger range of savings rates. Generally, the results are more sensitive with low company discount rates and high developer discount rates.

Table 1.4 demonstrates the positive relationships between the solar electricity price escalation rate and our two target metrics. Table 1.5 reveals that the optimal exercise year and savings rate are negatively correlated with the solar panel output degradation rate.

1.6. Conclusion and Future Work

In this chapter we present a developer-host contract that specifies the panel buyback year in which the host buys back the solar panels installed on its property. We model the cash flows of the host as an option framework and solve the minimum cost problem using Monte Carlo simulation based on ADP to obtain a good buyback timing. The modified framework can manage the inconsistent cash flows before and after the option exercise. By considering the panel purchase price to be the expected profit of the developer, the exercise year from the algorithm can balance the benefits of both the developer and host. In addition, we propose a new REC forecasting model, which can be applied to a market with specific RPS and ACP values.

The structure we introduced can fit a broader set of projects, such as wind generation. Such projects should have two common characteristics: only one exercise decision is made during the entire time horizon and cash flows differ before and after the exercise. Future work should extend the assumptions of our model. For example, we assume that the utility price increases year by year based on a fixed contract. However, a forward contract between the host and the utility can be employed. For the REC forecasting method, how market and policy factors influence the REC price should also be examined.

CHAPTER 2

Portfolio Optimization for American Options

2.1. Introduction

Portfolio optimization is a classic topic in financial engineering since the inception of the Modern Portfolio Theory by Markowitz (1952). With different objectives and constraints, a large body of literature has discussed the optimal capital allocation to financial assets such as stocks and bonds in a portfolio (see Brandt (2010) for a survey). Despite the wide recognition that options can help complete the market, only a handful of papers discuss a specific portfolio consisting mainly or only of options. Due to the high leverage nature of options market, option portfolios may yield unexpected returns that can potentially outperform a benchmark index. One may argue that we may apply broad researched portfolio optimization methods to obtain the weight allocation among options. However, it is already hard to find a stochastic process that captures the behavior of option returns exactly. Furthermore, the early exercise feature of American options makes it more difficult to apply methods of classic portfolio optimization to option portfolios. Option portfolios thus attract less attention than portfolio of stocks, futures or other assets.

When adding options into a portfolio, existing literature only considers European options, which are more convenient to incorporate thanks to its structural simplicity. However, with flexible exercise timing, American options are more complicated and thus modeling American option portfolios is much more challenging. Investors solve an American option portfolio in two steps:

first determine when to exercise the options in absence of all other considerations and then find the weights. However, in a simple example given next, we show that this is not necessarily a good strategy.

Suppose there are two independent American options in a portfolio, denoted by option 1 and option 2. By the pricing algorithm for American options in Longstaff and Schwartz (2001), their optimal exercise time can be found. Given option returns, we then implement a portfolio optimization with an objective to minimize the variance of the portfolio return based on the Markowitz mean-variance model. We impose three constraints. The first constraint demands the completeness of weights, i.e. the sum of weights equals to 1. The next constraint requires that the portfolio returns at least 95% of the average of the maximum of the two options returns. The last one is the no-short-selling constraint, which translates into nonnegativity of the two weights. The initial values of the underlying asset for option 1 and 2 are set to be $20 and $30 with volatility 0.2 and 0.3, respectively. We simulate the underlying asset returns using a geometric Brownian motion (GBM) model. In a discrete time setting, options can only be exercised at the end of month 1, 2, …, 6, where month 6 is the maturity time. Their exercise prices are $19.8 and $29, respectively, and the risk free rate is 6%. From the pricing algorithm, the option prices are $0.8 and $1.7, and the optimal exercise time for both options is month 5. Table 2.1 presents the variances of the portfolio return under different combinations of exercise time. It is apparent that if we exercise both options in month 5, we would not obtain the minimum variance. Instead, option 1 should be exercised in month 5 and option 2 in month 2, meaning the optimal exercise

time can be different if we consider the weight and timing decisions together. In fact, this is the motivation and goal of our Q-learning algorithm for American option portfolios.

Table 2.1: Variance of portfolio returns with different exercise times

Month	1	2	3	4	5	6
1	2.16	3.74	3.34	2.81	2.66	2.57
2	2.59	5.18	3.08	2.25	2.08	2.06
3	4.93	2.16	2.47	1.88	2.03	2.19
4	3.09	2.30	1.92	2.24	2.51	2.80
5	2.17	**1.67**	2.17	2.60	**2.93**	3.33
6	1.87	1.79	2.42	2.92	3.35	3.82

In our Q-learning algorithm, we consider two stages: the optimization and evaluation stage. In the optimization stage, we employ an iterative progressive hedging algorithm to find the weights and exercise time of all options at each time period, where the Q-values are approximated by regression. Note that here the term "hedging" differs from the meaning of hedging in finance – it is the name of an algorithm. Particularly, we include a penalty term in the myopic problem with approximate Q-values to drive the weights into convergence since we need only one set of weights (and not weights per period). In the evaluation stage, we mimic real-time trading and refine when to exercise the options for each simulated sample path (trajectory) given the weights from the optimization stage. The algorithm at this stage is similar to the preceding one except that the progressive hedging part is omitted while regression-based Q-value function approximation remains. For this stage, we have designed two algorithms. In one we modify an existing algorithm for pricing American options (our modification of the existing algorithm is needed because it cannot handle a portfolio of American options). The second algorithm uses a variant of our Q-learning algorithm. These two algorithms are compared for a small number of American options and time periods against a quasi-optimal benchmark, which enumerates all

possible weight and exercise time combinations with perfect information, i.e. flawless knowledge of when the maximal returns are achieved. We conclude by means of a simulation study that our algorithms perform well, with a small gap around 10% from quasi optimal in a relatively long time horizon. With empirical experiments, we discuss the scenarios where our Q-learning algorithms beat the underlying index from 2006 to 2015.

The main contributions of this chapter are as follows.

1. It is the first work that adds American options into option portfolio and explicitly takes the optimal exercise time into account concurrently with weights.
2. We develop a non-standard progressive hedging algorithm combined with Q-learning for solving the underlying option portfolio problem.
3. Along the way, we also exhibit a new algorithm for finding exercise times of a portfolio of American options. It significantly outperforms an adaption to the portfolio setting of an existing algorithm for finding the time to exercise an American option.

The structure of this chapter is as follows. Section 2.2 provides a literature review. Section 2.3 introduces the models and algorithms. Section 2.4 discusses the simulation and empirical experiments. Section 2.5 draws conclusions and presents future work.

2.2. Literature Review

In portfolio theory, Markowitz proposed the mean-variance model, an intuitive method that can handle single-period models well. However, investment is not simply a one-period decision.

Arrival of new information or changes in the overall objective can prompt adjustments in trading strategies. Thus, a multi-period model is more appropriate to cope with current complex portfolio optimization problems. Usually, researchers treat portfolio optimization in a continuous or discrete time. Merton (1969, 1971, 1975) first introduces portfolio choice problems in continuous time using stochastic calculus. The continuous setting enables to find a closed-form solution in some simple cases, for example, Merton (1990) gives an analytical solution to a portfolio optimization problem with a Brownian motion model using logarithm or power utility functions. In this chapter we discuss the discrete-time setting, which can be formulated as a Markov decision problem (MDP). There are a number of MDP methods used in the portfolio optimization literature, see Birge (2007) and Haugh and Kogan (2007) for a survey, but none when options are present. Q-learning, a branch of MDP and a model-free reinforcement learning technique, has been widely applied in the fields of machine learning and artificial intelligence. However, very limited literature applies Q-learning to solve portfolio optimization problems (sometimes under the name of approximate dynamic programming, see Denault and Simonato (2017)).

Current works on American option pay most attention to its pricing theory, with dedicated treatment to early exercise timing (e.g. Longstaff and Schwartz (2001), Tsitsiklis and Van Roy (2001), Stentoft (2014)). Yet, with a simulation and regression scheme, their focus is always on a single option and they do not consider the benefits of diversification by adding other American options and pricing the corresponding option portfolio. The value functions in their works

emphasize the approximation to expected option payoffs, while our Q-learning algorithm deals with the utility of option returns.

Since no existing literature adds American options into portfolio optimization, we only summarize papers that build portfolios with European options. Liu and Pan (2003) introduces derivatives into portfolios comprised with only primitive assets such as bonds and stocks. In a continuous time model, they obtain analytical results and conclude that options can improve the portfolio performance because they can complete the markets by adding risk factors such as stochastic volatility and price jumps. Ilhan et al. (2004) builds a portfolio model consisting of only one option and one stock based on stochastic volatility, while our model does not limit the number of derivatives and thus no close form expressions exist. By utility-indifference pricing mechanism, they further attain the optimal static composition. Constantinides et al. (2012) discusses an option portfolio constituted to maintain targeted maturity, moneyness and market beta. Their focus is to explain the cross-sectional variation of index option returns rather than to improve the portfolio performance. Other relevant papers are Jones (2006), Driessen and Maenhout (2013), Eraker (2013) and Hu and Jacobs (2016), who also, to some extent, discuss the role of European options in a portfolio. Their perspectives of optimizing portfolios vary, such as put mispricing, portfolio insurance, and option trading. These papers consider options only as European options. We are the first to add American options into portfolios and particularly deal with the optimal exercise time.

In this chapter, we apply a least-square recursive regression in the Q-learning algorithm for American option portfolios. The general idea of the approximation method can be found in Powell (2011). As the name suggests, regression-based approximations need to update regression coefficients. In the American option algorithm, besides regression, we introduce progressive hedging (PH) to find weights in the optimization stage. PH is proposed by Rockafellar and Wets (1991), which uses a penalty term to lead optimization into convergence. See Bianchi et al. (2009) for a survey of applications using PH.

2.3. Models and Algorithms

In this section we discuss the model for an American option portfolio, which is then solved by a Q-learning (QL) algorithm. In contrast to existing literature, we do not follow a traditional buy or sell option trading scheme and explicitly consider the early exercise opportunity of American options as a potential profit-generating source. In our work, the focus is not to achieve a market neutral portfolio (whether delta or gamma neutral) since our investment horizon is in the order of years, but an optimally weighted portfolio that can be exercised according to pre-calibrated value functions of utilities, aiming at maximizing the utility function or certainty equivalent of returns. There is no portfolio rebalance in our model; once the weights are determined, the only decision left is exercise timing. Moreover, we do not allow short selling and borrowing; therefore, weight values are strictly nonnegative and sum up to 1.

We assume that the time horizon is finite and investors can only trade options at discrete times t=1,…T. Suppose the number of options is N, and they are based on the underlying asset whose

price $A_t = (A_{1,t}, A_{2,t}, ..., A_{N,t})$ evolves based on a stochastic process. This is a general setup, but one could assume that the underlying asset is the same for all options.

The price of option *i* is $p_i = \left(p_{i,1}, p_{i,2}, ..., p_{i,T}\right)$, and the strike price is $K_i = \left(K_{i,1}, K_{i,2}, ..., K_{i,T}\right)$. Then the return of option *i* during time *t* is simply

$$r_{i,t} = \begin{cases} \left(\frac{A_{i,t} - K_{i,t}}{p_{i,t}}\right)^+ & \text{if call option;} \\ \left(\frac{K_{i,t} - A_{i,t}}{p_{i,t}}\right)^+ & \text{if put option,} \end{cases}$$

where $(\bullet)^+ = \max(\bullet, 0)$. Note that this schema can be extended to include other hedging instruments into the portfolio. For example, if an investor would like to add the underlying asset, we can simply include the return process of the underlying as a new $r_{i,t}$, where *i* now represents the underlying asset. We consider the underlying asset is "exercised" when it is sold and the transaction time as the "exercise time." In this way, the underlying asset can be effectively treated as an option in the following discussions. The hedging scenario will be revisited during empirical experiments in Section 2.4.

The portfolio strategy over the entire horizon is represented by

$$w = \left(w_1, w_2, ..., w_T\right) \in X = \left\{ w \in \mathbb{R}_+^{N \times T} : \sum_{i=0}^{N} w_{i,t} = 1, \text{for every } t \right\}, \text{ where } w_t = \left(w_{1,t}, w_{2,t}, ..., w_{N,t}\right)^T.$$

Weight $w_{i,t}$, for every $i \in \{1, .., N\}$ is the weight of option *i* during time *t* in the portfolio.

In solving the portfolio optimization problem comprising American options, we face challenges not only to come up with the optimal weights, but also the optimal exercise timing. To tackle the challenge, we create a Q-value function that is a product of option weights, exercise time and the underlying asset price (i.e. product of the actions and states, to be defined in Section 2.3.1), the most crucial features for an American option portfolio to approximate the value function of utilities. Q-learning is more appropriate in this context since exercise times are discrete values, which are present in both the actions and state space. For the same reason, regression is more appropriate to approximate the Q-value function; it does not involve derivatives and updates the slopes dynamically from trajectory to trajectory.

Given a set of weights, the optimal exercise times have to be determined. Following such an approach it is not clear how to change or adjust the weights. The idea is to relax the restriction that we have a single weight vector that comprises the weights of each option. Instead we assume that there is a weight vector per time period which are then adjusted in each iteration. In other words, the weights are for each option and for every time period. To drive the weight optimization into convergence, we introduce a progressive hedging component into the optimality equation. PH adds a factor to the myopic optimization problem that penalizes the weights to differ across time periods. At the end of the PH algorithm, we fix the portfolio weights by taking the average across all time periods, which becomes an optimal weight vector that can be further used when we try to find the optimal exercise timing in the *evaluation stage*, the only decision variable that remains. The evaluation stage provides a more accurate assessment of the average performance than the optimization stage only.

2.3.1. Optimization stage

In this stage, our main task is to find optimal option weights that are used later.

Let the exercise time of each option

$$S_t = \left(s_{1,t}, s_{2,t}, ..., s_{N,t}\right)$$

and underlying asset prices A_t be the *state variables*. Here $s_{i,t}$ is the exercise status indicator which equals to t if option i is exercised in time period t, and 0 otherwise. Particularly, $s_{i,0} = 0,$ for every $i = 1,...,N$. Asset prices are part of the state space for algorithmic purposes (the Q-value approximation is also a function of asset prices). This also allows the opportunity to stochastically generate them based on a time dependent process without a need to change the algorithm. One of the *action variables* is the set of options that should be exercised in time period t. Note that the exercise times depend on the realization of asset prices and option returns but for simplicity we omit this dependency in our notation. We represent the option set by a vector of index variables, denoted by y_t. For example, if options 1 and 2 are to be exercised, then $y_t = \{1,2\}$. Hence, the corresponding *post decision state variable* is

$$S_t^y\left(S_t, y_t\right) = S_t + t \cdot 1^{y_t}, \tag{2.1}$$

where 1^{y_t} is an N-element vector of indicator variables with element values equal to 1 if corresponding options are to be exercised, and 0 otherwise. In the previous example, $1^{y_t} = (1,1,0,0,...,0)$.

Once the state variable $s_{i,t}$ of option *i* changes from 0 to a positive integer of time, it is fixed to this integer in later time periods. Another *action variable* is the option weight in each time period

$$w_t = \left(w_{1,t}, w_{2,t}, ..., w_{N,t}\right).$$

Note that we have different weights for each option and each time period, which purposely deviates from the practice that only one set of weights is required before an investment. To obtain the optimal set of weights w_i for every *i*, we simply take the average of $w_{i,t}$ for every *t*. This set of weights is fixed and then used in the evaluation stage. Here we follow the strategy outlined in the introduction of the section. In other words, we have side constraints $w_1 = w_2 = ... = w_T = \overline{w}$. These constraints are relaxed in the PH spirit.

The *objective function* for our problem is $\max_{w} E\left[U\left(W_T\left(w\right)\right)\right]$, where $w = \left\{w_t\right\}_{t=1}^{T}$. Here W_T is the terminal portfolio wealth. The *optimality equation* reads

$$V_t\left(S_t, A_t\right) = \max_{w_t} E\left[\max\left(a, b\right) \middle| S_t, A_t\right],$$

where

$$a = \max_{y_t \subset \left\{i:\, s_{i,t}=0\right\}} V_{t+1}\left(S_t^y(S_t, y_t), A_t\right),$$

$$b = U\left(\sum_{i:\, s_{i,t}\neq 0} w_{i,t} r_{i,s_{i,t}} + \sum_{i:\, s_{i,t}=0} w_{i,t} r_{i,t}\right) + V_{t+1}\left(S_t^y\left(S_t, \left\{i : s_{i,t} = 0\right\}\right), A_t\right).$$

Contribution *a* is the value function if less than (*R* – 1) options are exercised, where *R* is the number of unexercised options up till now. It corresponds to the case that not all options are exercised, i.e. y_t is a proper subset of $\left\{i : s_{i,t} = 0\right\}$. Term *b* is the value function if all the

remaining options are exercised, thereby no optimization is involved in this equation. Moreover, the utility of portfolio wealth is added only after all options are exercised . Without loss of generality, we assume $W_1^{total} = 1$, and hence $W_T^{total} = \sum_{i:\, s_{i,t} \neq 0} w_{i,t} r_{i,s_{i,t}} + \sum_{i:\, s_{i,t}=0} w_{i,t} r_{i,t}$ is the argument of the utility function. The first sum of W_T is the weighted return of exercised options before time period *t*. The second sum is the weighted return of the remaining options exercised in time period *t.* The maximum of *a* and *b* is then considered as the objective function that is being optimized to find the best weights and exercised times.

Instead of approximating V_t, Q-value function is derived to approximate V_{t+1} in *a* and *b* as a function of S_t, A_t and w_t. In essence, we rewrite

$$a = \max_{y_t \subset \{i:\, s_{i,t}=0\}} Q_{t+1}\left(S_t^y\left(S_t, y_t\right), A_t, w_t\right),$$

$$b = U\left(\sum_{i:\, s_{i,t} \neq 0} w_{i,t} r_{i,s_{i,t}} + \sum_{i:\, s_{i,t}=0} w_{i,t} r_{i,t}\right) + Q_{t+1}\left(S_t^y\left(S_t, \{i : s_{i,t} = 0\}\right), A_t, w_t\right).$$

This is not quite the standard Q-value approximation but a minor variation. To approximate Q_{t+1}, we use recursive least square regression for nonstationary data. In the regression,

$$\overline{Q}_{t+1}\left(S_t^y, A_t, w_t\right) = \sum_{i=1}^{N} \theta_{i,t+1} \cdot \left(w_{i,t} s_{i,t}^y A_t\right),$$

where θ are the regression slopes, and inside the bracket is the product of three features – weights, exercise times and asset prices. The slope updates keep track of the temporal differences of old and new estimates of the value functions from iteration to iteration. Note that our method is

model-free and does not involve transition functions since asset prices are based on Monte Carlo simulation.

We also include a progressive hedging mechanism to accelerate convergence by imposing that $w_1 = w_2 = ... = w_T = \bar{w}$, which means all time periods should have the same set of weights. To achieve this, a penalty term is introduced in the optimality equation to drive the optimal weights to converge to $\bar{w}$. The adjusted *optimality equation* now becomes

$$\overline{V}_t\left(S_t, A_t\right) = \max_{w_t} E\left\{\left[\max(a,b) - \left(z_t\right)^T \cdot w_t - \frac{\rho}{2}\left\|w_t - \overline{w}\right\|_2^2\right] \middle| S_t, A_t\right\},$$

where parameter z_t represents the cumulative difference between w_t and $\bar{w}$.

The algorithm called IPH (iterative progressive hedging) is presented in Algorithm 2.1. In Step 2, we find the updated value of V_t based on the current approximation to Q_{t+1}. Step 3 exhibits standard formulas for updating regression coefficients when a single new observation is added. After each iteration of progressive hedging, the new average weight $\bar{w}^{NEW}$ and cumulative deviation z_t are updated in Steps 5 and 6. We terminate the algorithm if the norm between $\bar{w}^{NEW}$ and $\bar{w}$ is less than a given threshold g_{term} (Step 7); otherwise, let $\bar{w}$ take the new value (Step 1). The obtained single set of weights $\bar{w}$ is further used in the evaluation stage as an input.

Algorithm 2.1

---

a. Initialize $\overline{\theta}_t^0, \lambda$, $B_t^0 = \varepsilon I$, simulate samples $A_t^n, r_{i,t}^n$.

b. Set $g_k = 1$, $\overline{w}^{NEW} = \{1/N\}_{N\times 1}$, $z = \{0\}_{N\times T}$.

Loop

1. Let $\overline{w} = \overline{w}^{NEW}$.

For *n*=1,…,N_1

For *t*=1,…,*T*

2. Solve $\widetilde{V}_t^n = \max_{w_t^n}\left[\max(a,b) - \left(z_t^n\right)^T \cdot w_t^n + \frac{\rho}{2}\left\|w_t^n - \overline{w}\right\|_2^2\right]$ where

$$a = \max_{y_t^n \subset \{i:\, s_{i,t}^n = 0\}} \overline{Q}_{t+1}^{n-1}\left(S_t^y(S_t^n, y_t^n), A_t^n, w_t^n\right),$$

$$b = U\left(\sum_{i:\, s_{i,t}^n \neq 0} w_{i,t}^n r_{i,s_{i,t}^n}^n + \sum_{i:\, s_{i,t}^n = 0} w_{i,t}^n r_{i,t}^n\right) + \overline{Q}_{t+1}^{n-1}\left(S_t^y\left(S_t^n, \{i : s_{i,t}^n = 0\}\right), A_t^n, w_t^n\right),$$

$$\overline{Q}_{t+1}^{n-1}\left(S_t^{y,n}, A_t^n, w_t^n\right) = \sum_{i=1}^{N} \overline{\theta}_{i,t+1}^{n-1} \cdot \left(w_{i,t}^n s_{i,t}^{y,n} A_t^n\right).$$

Let $w_t^{n,*}$ be an optimal solution and $y_t^{n,*}$ be an optimal solution to the maximization problem for computing a or $\{i : s_{i,t}^n = 0\}$, depending on which term attains the maximum in $\max(a,b)$. By using $y_t^{n,*}$ and (2.1), we update the exercise time of each option i to $s_{i,t}^{y^*,n}$. We then define $\phi_{i,t}^n = w_{i,t}^{n,*} s_{i,t}^{y^*,n} A_t^n$ for every option i.

3. Update

$$\bar{\theta}_t^n = \bar{\theta}_t^{n-1} - H_t^n \phi_t^n \hat{\varepsilon}_t^n,$$

where

$$\hat{\varepsilon}_t^n = \left(\bar{\theta}_t^{n-1}\right)^T \phi_t^n - \tilde{V}_t^n,\ H_t^n = \frac{1}{\gamma_t^n} B_t^{n-1},\ \gamma_t^n = \lambda + \left(\phi_t^n\right)^T B_t^{n-1} \phi_t^n,$$

$$B_t^n = \frac{1}{\lambda}\left(B_t^{n-1} - \frac{1}{\gamma_t^n} B_t^{n-1} \phi_t^n \left(\phi_t^n\right)^T B_t^{n-1} \right).$$

4. Find the next pre-decision state

$$S_{t+1}^n = S_t^y \left(S_t^n, y_t^{n,*}\right).$$

End

End

5. Update $\bar{w}^{NEW} = \frac{1}{NT} \sum_{t,n} w_t^n$.

6. Update $z_t^n = z_t^n + \rho\left(w_t^n - \bar{w}^{NEW}\right)$ for all *n*, *t*.

7. If $\left\|\bar{w}^{NEW} - \bar{w}\right\| < g_{term}$, exit.

End

The most important output of IPH are weights $\bar{w}$ (although the approximate Q-value function is also an output).

2.3.2 Evaluation stage

With weights from the optimization stage, we now move on to evaluation. By fixing option weights, we mimic real-world practice that only allows one single set of weights that does not

vary over time. In essence, this stage evaluates the weights more precisely by using two different algorithms tailored specifically for exercising options.

The first algorithm is a stripped-down version of IPH, which omits the outer loop of progressive hedging. With these simplifications, the new algorithm called IPH-QL only finds the exercise time of each option in every trajectory (given weights from the optimization stage).

The second evaluation algorithm is a modification of Longstaff and Schwartz (2001). They proposed a Least Square Monte Carlo (LSMC) algorithm, which prices an American option by regression and returns exercise time for each sample path. We modify their method, apply it in the portfolio setting and evaluate their performance in discovering the exercise time. To capture risk aversion, cash flows in the original algorithm are replaced by the utility of option returns. The algorithm assumes an additive utility function so that the portfolio utility is the sum of individual utilities. The modified LSMC algorithms are exhibited as Algorithms A.2 and A.3 in Appendix A. The resulting algorithm is labeled as IPH-LSMC (weights obtained by IPH and evaluation done by our version of LSMC).

Note that the two evaluation algorithms only provide two approaches to evaluate the weights by determining exercise times in two distinct ways. One is based on our own QL (stripped-down version of IPH where weights are fixed but subsets of options to exercise are explicitly captured in states and actions) and the other one is a modification of a known algorithm based on LSMC.

2.3.3. Quasi-optimal benchmark algorithm

To benchmark our IPH algorithm at the optimization stage, we also introduce a 'quasi' optimal algorithm that sweeps all possible weight vectors in fine-granular discrete steps. Since there are many weight value combinations, the algorithm works for only a small number of options. For each weight vector, we then search the best exercise time of each option in every sample path, which is found by enumerating all possible sets of time periods and taking the one with the largest utility. This enumeration step implies that the number of time periods also needs to be reasonably low. Finally, we select the weight vector with the largest portfolio utility. The weights obtained from this enumeration process are further used at the evaluation stage to assess the quality of IPH. The resulting versions based on the two evaluation methods are denoted as QO-QL and QO-LSMC.

To add on top of this the quality of the evaluation, we assume perfect information at the evaluation stage, i.e. simply enumerates all possible sets of exercise times and picks the one that yields the largest portfolio utility, given the quasi optimal weights. We call the resulting algorithm QO-PERFECT.

It should be stressed that to obtain quasi optimal weights at the optimization stage and optimal exercise times at the evaluation stage requires perfect information, which is impossible in reality. Except for enumerating weights, thus, QO-PERFECT provides an upper bound on an optimal solution. In what follows we use QO-PERFECT as the baseline and all other solutions in the simulation study are measured against it.

2.4. Numerical Results

In this section, we show numerical results by comparing the proposed algorithms. Both simulation study and empirical experiment are presented. We include a simulation study because it helps to assess the average or expected performance of our algorithms, while the empirical experiment only evaluates an actual sample path.

We use the CRRA utility function

$$U(W) = \frac{W^{1-\gamma}}{1-\gamma}.$$

To avoid extremely negativity, the utility is set to a fixed negative value given a W less than a negative threshold. The threshold is determined upon the choice of γ.

All algorithms have been implemented in MATLAB on an Apple Mac computer with 4.0 GHz Intel Core i7 processor and 16 GB of RAM.

2.4.1. Simulation study

We simulate 20 months of log returns of the underlying asset using GEV distribution with parameter k = -0.149 , σ=0.0153 , μ=-0.00545 . These parameters are fitted based on 15 years of historical S&P500 index values. After simulating the monthly returns, the cumulative index returns for each trajectory are calculated, which is positively skewed as shown in Figure 2.1. The CRRA parameter used in this simulation study is set to 9.

The portfolio contains four American options: an ATM put option, a 5% OTM put option, an ATM call option and a 5% OTM call option, all of which depend on the same underlying index. Their prices are determined using the algorithm proposed by Longstaff and Schwartz (2001), with LIBOR rates and historical volatilities as the pricing inputs. At the optimization stage, 50 iterations are used (sample paths) to find weights with $g_{term} = 0.1$ and $\rho = 1$ for $T = 5,10$ and $g_{term} = 0.15$ and $\rho = 2$ for $T = 15,20$. These parameters are chosen to trade off run time and utility performance. At the evaluation stage, we create 1,000 iterations (resampled trajectories) to generate a histogram of exercise times (Figure 2.4) with a reasonable number of bins. In the benchmark algorithm, the weights are discretized by 0.05.

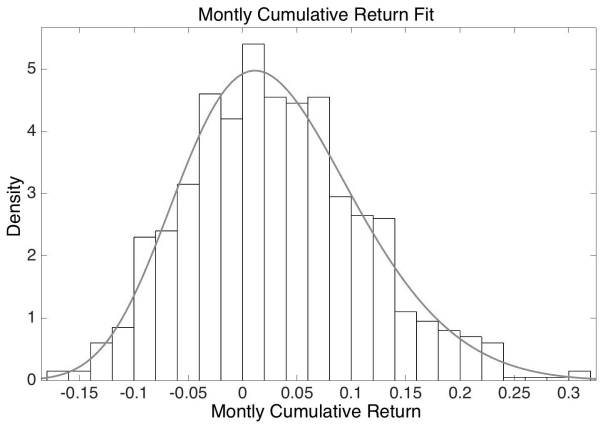


Figure 2.1: Monthly cumulative return of simulated index fitted by GEV distribution

To see how well the algorithm performs, the utility gap is calculated between the benchmark algorithm and IPH-QL or IPH-LSMC. Since the QO-PERFECT algorithm injects perfect information for both weights and exercise times, penalty is needed in order to get a fair gap. We therefore add a gradient-based penalty to the QO-PERFECT (baseline) portfolio utility, proposed by Brown and Smith (2011). It takes 80 seconds to run IPH, 2 seconds to run QL-based evaluation and 1 second to run LSMC.

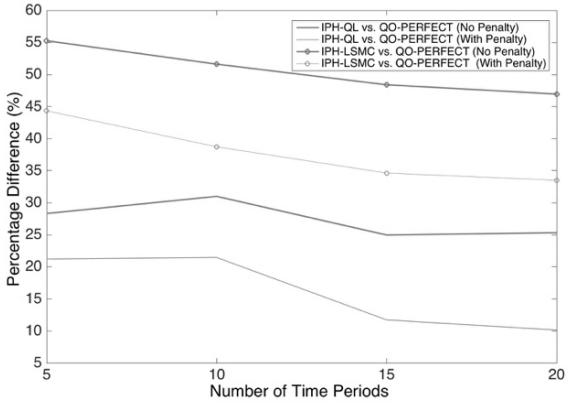


Figure 2.2: Utility gap from baseline

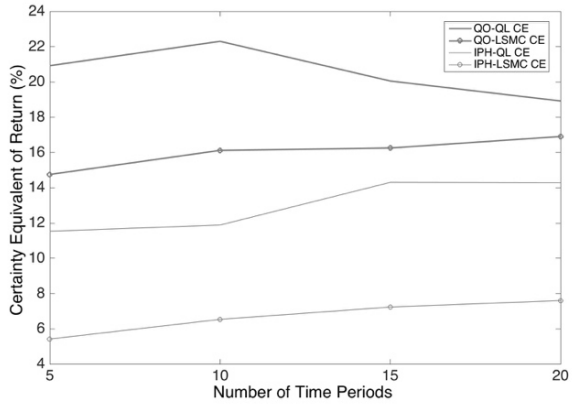


Figure 2.3: Certainty equivalent of returns

Note that both IPH-QL and IPH-LSMC use the weights from the IPH algorithm. Observed from the IPH-QL algorithm, the utility gap without penalty tends to stabilize around 25 – 31%. With penalty, the gap is reduced by 7 – 15% (Figure 2.2). We see that as the number of time periods increases, the utility gap with penalty tends to decrease. IPH-LSMC leads to a higher utility gap, with penalized gap even higher than the unpenalized one from the IPH-QL algorithm. The next simulation experiment tries to find how far the IPH weights are from quasi optimal. As we have discussed in 3.2.3, despite different weight choices, we still use the QL and LSMC algorithms to

measure performances. In other words, we compare IPH-QL against QO-QL, and IPH-LSMC against QO-LSMC. From Figure 2.3, the difference of certainty equivalent (CE) between the IPH-QL algorithm and QO-QL is narrowing and decreases from around 10% to 4.6%. However, although rising slowly as the number of time periods increases, CE of the IPH-LSMC algorithm is very low, less than 8%.

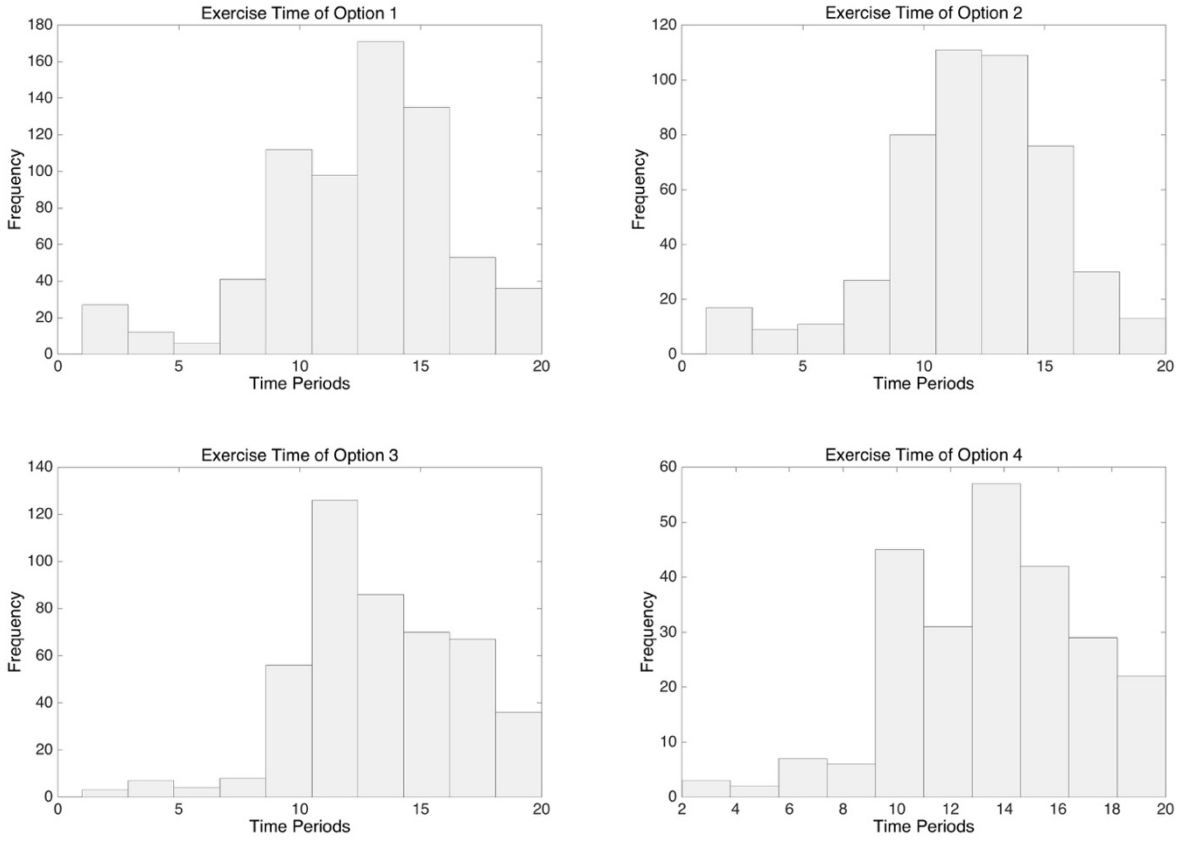


Figure 2.4: Exercise time of each option

In Figure 2.4, we only plot the exercise times for options that are exercised in each sample path using the IPH-QL algorithm. Options tend to be exercised in late periods before maturity.

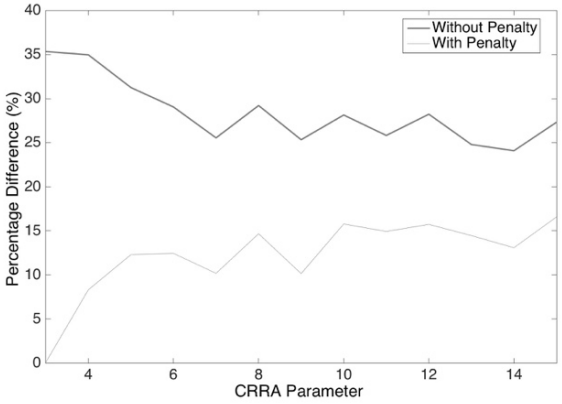


Figure 2.5: Utility gap from baseline

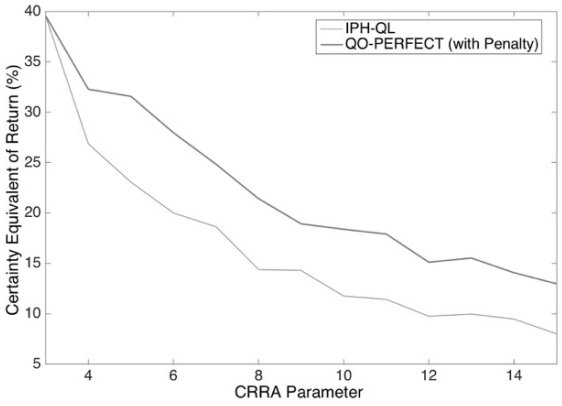


Figure 2.6: Certainty equivalent of return

In the 3rd experiment, we vary the CRRA parameter (Figures 2.5 and 2.6). The utility gap between IPH-QL and QO-PERFECT is less than 16% with penalty, while the CE of IPH-QL is around 5% less than the penalized QO-PERFECT value.

In the next simulation experiment, we increase the number of options to 10 and evaluate the performance of the IPH-QL algorithm. The added 6 options are: a 5% ITM call option, a 5% ITM put option, a 2.5% OTM call option, a 2.5% OTM put option, a 2.5% ITM call option and a

2.5% ITM put option. The portfolio is now symmetric; there are both put and call, ITM and OTM for all values of moneyness. Due to a larger number of options, we set $g_{term} = 0.25$, $\rho = 4$ and $T = 20$. Because of the exponentially increased running time of QO (enumerating all options), we do not implement the QO benchmark since it would take days to finish a single run. CE of IPH-QL is 14.1%, similar to the 4-option case. The algorithm takes 8 minutes to terminate, which shows its scalability.

2.4.2. Empirical experiment

In this section, we design empirical experiments with historical market data from the OptionMetrics Ivy DB database. Options are no longer priced via simulation, but trajectories of asset prices are still simulated to find optimal weights and exercise timing. The underlying asset of options is SPDR S&P500 ETF (Symbol: SPY), which closely tracks the S&P500 index but is traded at $1/10^{th}$ of the index value.

The experiment time horizon ranges from 2006 to 2015. In each year, we construct portfolios twice, one in January and another one in July due to the availability of tradable SPY options. In January, the length of time periods (expiration) takes value of 1 year (12 months), while July takes value of 1.5 years (18 months). This setting helps to answer whether a shorter or a longer time horizon benefit most from our algorithm. Specifically, the PH hyper parameters are $g_{term} = 0.1$, $\rho = 1$ for 12-month maturity and $g_{term} = 0.125$, $\rho = 1.5$ for 18-month maturity.

In the following experiments, two scenarios are discussed: portfolio with options only (non-hedge case) and portfolio with options and underlying as the hedging instrument (hedge case). For both cases, we include 4 portfolio settings:

1. an ATM call, an ATM put, a 5% OTM call and a 5% OTM put,
2. an ATM call, an ATM put, a 5% ITM call and a 5% ITM put,
3. an ATM call, an ATM put, a 5% ITM call and a 5% OTM put,
4. an ATM call, an ATM put, a 5% OTM call and a 5% ITM put.

Since strike prices increment by 5 points across order book levels, the closest integer strike prices are used to approximate 95% or 105% of moneyness. Option prices on each date are determined at the average closing ask (we always long options due to the no-short-selling constraint), which we consider a way to embed market friction and transaction cost. Options can be exercised in the beginning of any months before expiration (one opportunity a month). This limitation essentially reduces the size of the action space.

To compare the performance of investors with different risk preference, we vary the risk aversion CRRA parameter from -0.5 to 20. Special cases are -0.5, 0 and 1: value -0.5 represents a risk-seeking investor (convex utility function), value 0 means risk-neutral (linear) and value 1 corresponds to a log utility function. The other four tested values are 2, 5, 10 and 20. We also discuss if hedging with the underlying ETF SPY improves the overall performance, whose weight is determined by the same IPH algorithm, since it can be effectively modeled as an option (see Section 2.3). Additionally, we investigate how the choice of distribution influences

performance, given that the value functions are approximated by simulated log returns. Two distributions are thus evaluated: GEV and GBM. Their parameters are calibrated by the historical prices in the past 10 years of each portfolio construction date.

Our IPH-QL algorithm is tested against equal weights (EW-QL) and perfect timing (IPH-PERFECT). Quasi-optimal weights are not applied here because it is more suitable to evaluate average performances as in the simulation study, while the actual trajectory only represents one realization.

In summary, in our empirical study we try to answer the following questions:

1. Does our algorithm perform better in a shorter or longer time horizon?
2. Which portfolio setting outperforms the others?
3. What type of investors are appropriate to invest in American option portfolios?
4. Should we hedge our position using the underlying asset?
5. Which distribution models the underlying asset returns better, GEV or GBM?

Figure 2.7 summarizes the expected return and Sharpe Ratio under different CRRA parameters under the horizon of 1 year. It can be observed that the weights from the IPH algorithm performs much better than equal weights by comparing IPH-QL with EW-QL in all portfolio settings. IPH assigns larger weights to call options and results in a greater delta, which is considered a good strategy given the strong upward pattern of the S&P500 index in recent years. Yet the QL evaluation algorithm still has room for improvement to close the gap between IPH-QL and IPH-

PERFECT, despite the fact that perfect information can never be attained in reality. The same observation applies to the time horizon of 1.5 years.

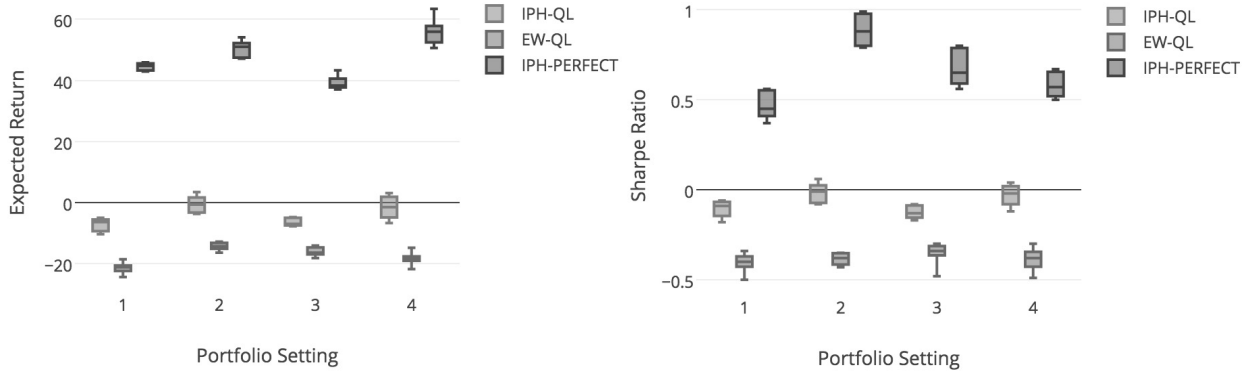


Figure 2.7: Expected return and Sharpe ratio of each algorithm (1-year, GEV distribution, non-hedge)

Figure 2.8 indicates that IPH-QL presents better profitability in a longer term that allows more exercise opportunities. Over the same period, holding the underlying asset SPY achieves an annualized return of 5.1% and a Sharpe ratio of 0.31. Only portfolio setting 2 and 3 have the potential to beat the index based on our experiments. Both settings include an ITM call option that enjoys large positive returns. In what follows, we use the time horizon of 1.5 years.

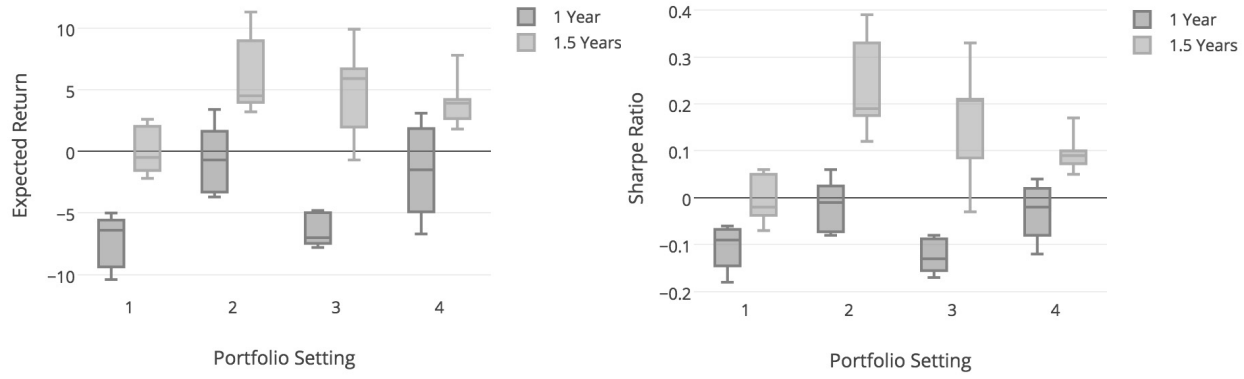


Figure 2.8: Expected return and Sharpe ratio of different maturities

(GEV distribution, non-hedge, IPH-QL)

The risk aversion parameter affects the performance. In general, the greater the CRRA parameter, the worse the performance (Figure 2.9). Rephrasing, investors with strong risk aversion are advised not to invest in American option portfolios. The performance of each portfolio setting also depends on risk preference; settings 2 and 3 outperform the other two under small risk aversion (less than 5), while settings 2 and 4 perform better with CRRA parameter greater than or equal to 5.

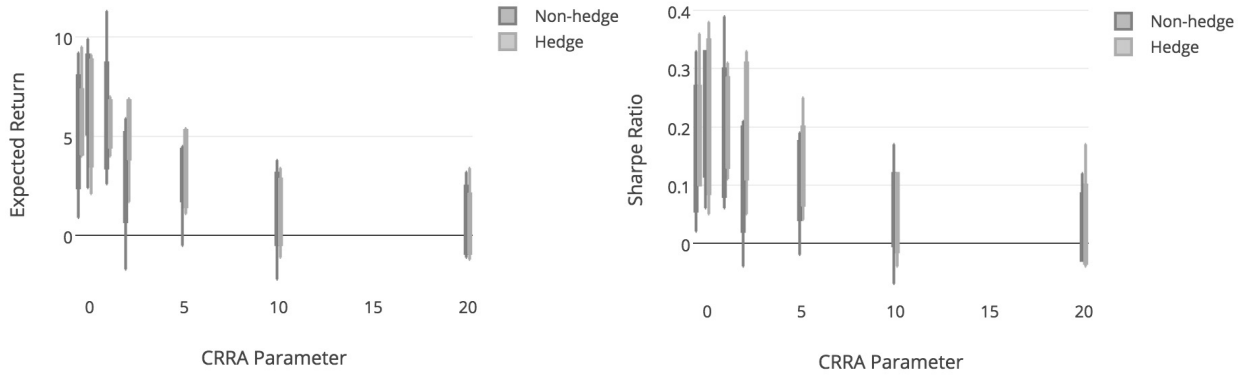


Figure 2.9: Expected return and Sharpe ratio of different CRRA parameters

(GEV distribution, non-hedge vs. hedge, IPH-QL)

By further examining Figure 2.9, we conclude that hedging with the underlying asset positively impact the returns and Sharpe ratio. Under the GEV distribution, the average expected return and Sharpe ratio across all risk preferences are 4% and 0.15, compared with the non-hedge case of 3.8% and -0.07. The GBM distribution also benefits from the hedge scenario with an average expected return of 4.3% and Sharpe ratio of 0.16, compared with the non-hedge case of 3.6% and 0.12.

Conducting a similar analysis, we notice that the GEV distribution leads to a better performance when small risk aversion is present (less than 5), while GBM yields better results when the parameter is greater than or equal to 5 (Figure 2.10).

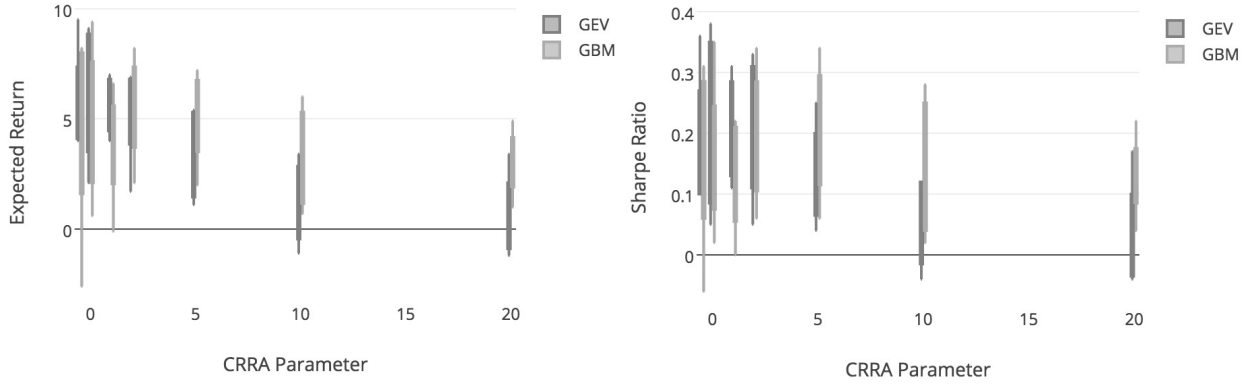
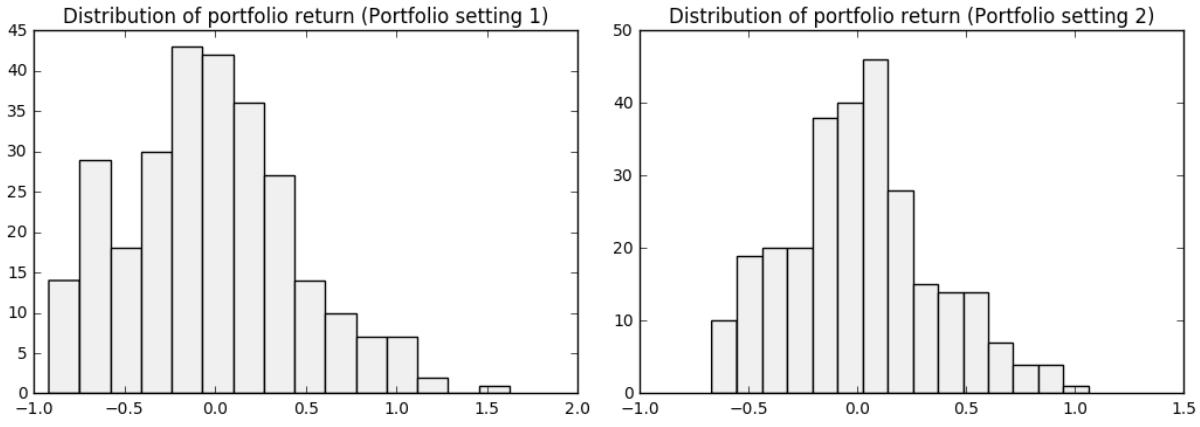


Figure 2.10: Expected return and Sharpe ratio of different CRRA parameters (GEV vs. GBM, hedge, IPH-QL)

Figure 2.11 displays the distribution of portfolio returns. All of them show positive skewness (around 0.3 to 0.4). Portfolio setting 3 has the largest excess kurtosis (0.7) and setting 4 presents a negative kurtosis (-0.14). The kurtosis for portfolio setting 1 and 2 are 0.08 and 0.01.



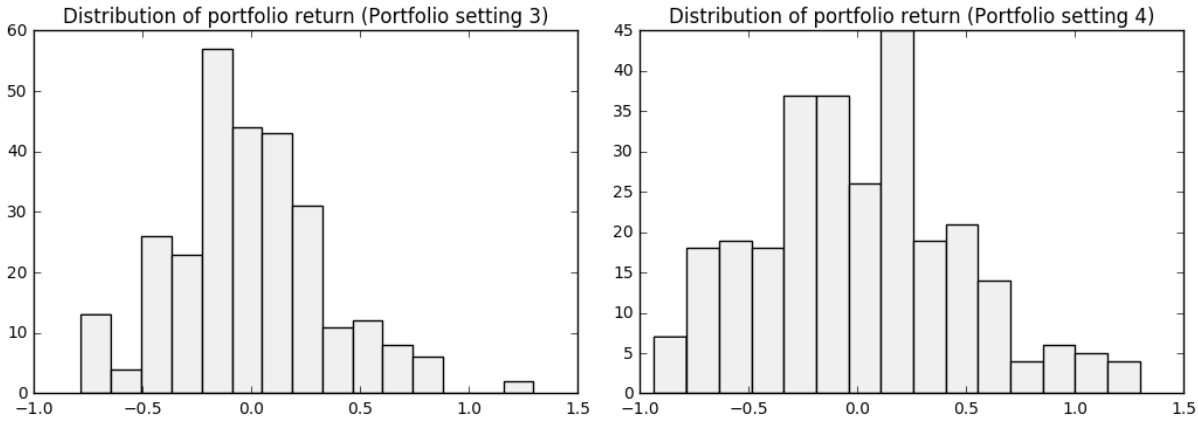
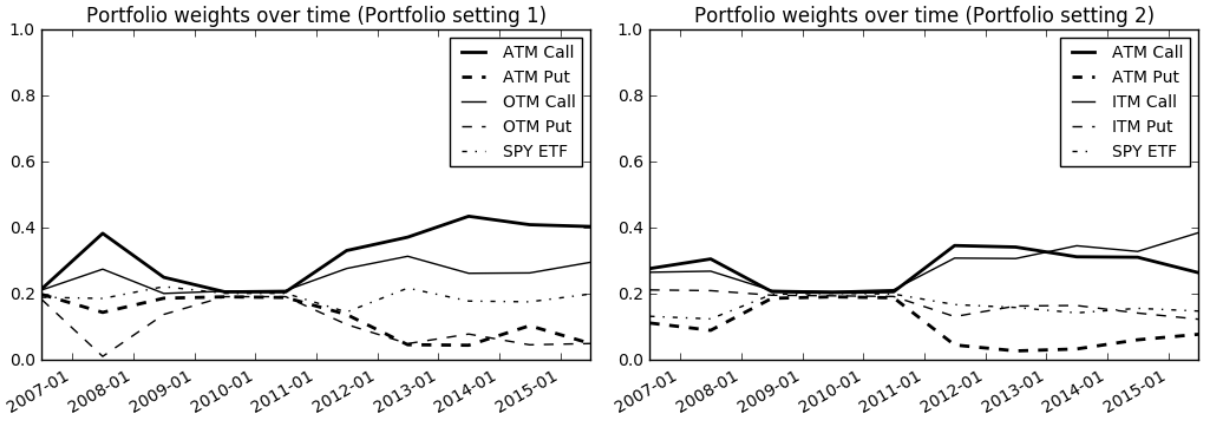


Figure 2.11: Distribution of portfolio returns

(GEV, hedge, IPH-QL)

Figure 2.12 exhibits the average portfolio weights over time. Within 2 years after the financial crisis in 2008, our IPH algorithm prefers equal weights for all options that essentially form a straddle strategy. Given an uncertainty about market directions, long straddle strategies make profit if the market moves either up or down considerably. During other times, call options dominate the portfolios and result in a positive delta. The underlying asset, however, tends to share a stable weight of 20% at all times.



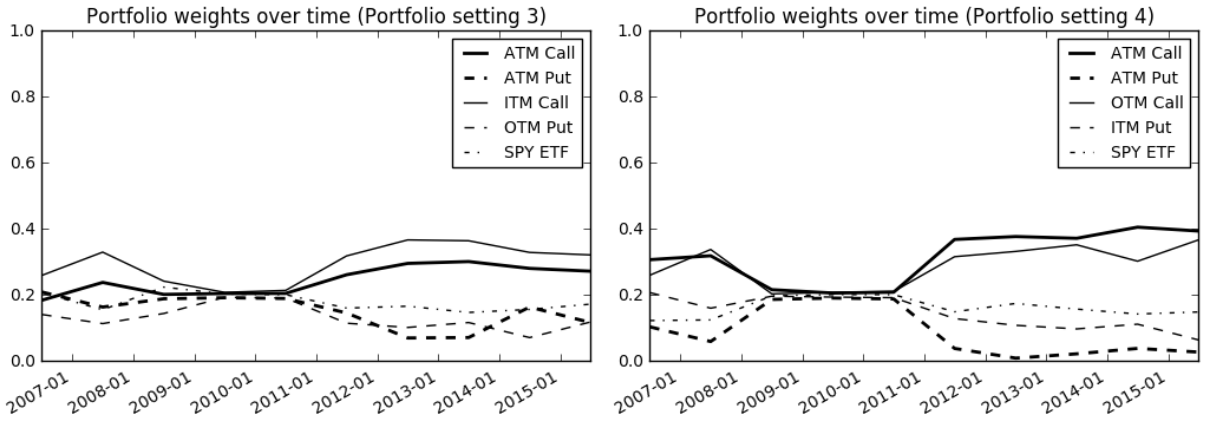


Figure 2.12: Portfolio weights over time

(GEV, hedge, IPH-QL)

Further details of algorithmic performances can be found in Section A.4, Appendix A.

2.5. Conclusion and Future Work

In this chapter, we propose a model of American option portfolios and use regression-based Q-learning algorithms to find excellent portfolio compositions. Our algorithms outperform LSMC with regard to the utility gap from optimal and CE of return. The gap and CE are better in a longer time horizon, while with an increasing CRRA parameter, the gap is relatively stable and the CE decreases. The empirical experiments show that our weights perform well, yet the evaluation algorithm can still be improved to achieve a higher Sharpe ratio. In addition, the underlying asset as a hedging instrument improves the overall portfolio performance. Finally, we advise that risk averse investors avoid constructing American option portfolios due to extreme high-leverage risk. In Sections A.1 and A.2, Appendix A, a study of European option portfolios is presented as a complement to this chapter.

CHAPTER 3

Online Adaptive Machine Learning Based Algorithm for Implied Volatility Surface Modeling

3.1. Introduction

Machine learning is gaining interest in the finance industry. In the last two decades, support vector machine, neural networks, decision trees, reinforcement learning, genetic programming and other machine learning models have been widely applied to tackle complex problems in finance, such as market direction forecasting, sentiment analysis, portfolio optimization, bankruptcy prediction, credit risk modeling, etc. For these topics, an important aspect is the challenge of the non-stationarity of noisy data, due to parameter regimes varying from time to time. Inability to react to a pattern drift can lead to damaging predictive performance and unprofitability in real-time trading. This fact motivates us to go beyond off-line training and to propose a novel online adaptive machine learning algorithm that is applied, for example, to tick data from the S&P500 options market.

Since the inception of the Black-Scholes-Merton model, implied volatility surface (IVS) modeling has been a popular topic in options pricing theory. IVS is a mapping from the strike prices and time to maturity of options to a nonnegative value, implied volatility, whose value depends on strike prices, time to maturities, interest rates, dividends and so forth. Despite the recognition that their assumptions do not hold in a realistic trading environment, the Black-Scholes-Merton formula is widely used due to its simplification from an option price to a

nonnegative value called implied volatility, which enables a fair comparison of options with different strikes, maturity and the underlying assets. As Poon and Granger (2003) point out, option implied volatility is shown to have the most information on future market volatility and outperforms classical time series based models. It also performs well across different asset classes and over a long forecasting horizon. Various methods can be used to model the IVS, such as stochastic volatility models, Levy processes, GARCH, spline interpolation, etc. (see Homescu, 2011 for a survey). Nonetheless, machine learning algorithms are seldom applied. Recent works include Audrino and Colangelo (2010) (regression trees) and Wang, Lin et al. (2012) (artificial neural nets). Yet, all the above works view IVS modeling from a static perspective, not allowing the model to update adaptively when new market information arrives.

In this work, we propose a novel adaptive machine learning method based on support vector regression (SVR), which is further employed to update IVS. The SVR method designed is an adaptation and enhancement of Shalev-Shwartz et al. (2007), who develop an effective support vector machine (SVM) method using stochastic sub-gradient descent that solely optimizes the primal objective function. Compared with the dual formulation, their primal SVM has an advantage of simplicity and can be easily adapted to the stochastic gradient descent method. Aiming at classification, they briefly mention the modification of the Pegasos algorithm suitable for regression with ϵ-intensive loss but they do not derive the full regression algorithm, which we discuss in details. As an enhancement, we introduce the concept of feature vector selection (FVS) into the primal SVR algorithm. Instead of training with all data, the online algorithm updates the model using selective data points (as support vectors) that are orthogonal in the reproduced

kernel Hilbert space. The idea of combining FVS and SVR is first proposed by Liu and Zio (2016), but their online SVR is based on the dual formulation of the optimization problem rather than the primal. In addition, to adaptively modify the model upon pattern drift, their solution attributes to incremental and decremental learning (Cauwenberghs and Poggio, 2000), while our algorithm updates support vectors by budget maintenance through removal (Wang, Crammer et al., 2012), which maintains the support vector size defined in FVS. To further speed up the algorithm, we implement the most computationally intensive parts in a Field Programmable Gate Arrays (FPGA) hardware developed by Maxeler Technologies, and contrast its runtime performance against a pure CPU implementation.

To summarize, our contributions focus on the following four aspects.

1. This work presents the first derivation and implementation of online primal kernel SVR. Pegasos provides an algorithm for primal kernel SVM and a quick mentioning of the extension to primal kernel SVR with no implementation and computational study in the SVR setting.
2. We provide an algorithmic enhancement to online primal SVR by means of FVS and adaptive support vector updates through budget maintenance.
3. We propose a new IVS modeling algorithm using our online primal SVR.
4. A new application of the FPGA technology is provided to accelerate the most computationally intensive parts in our algorithm.

The rest of this chapter is structured as follows. Section 3.2 reviews existing literature. Section 3.3 gives background of primal SVR algorithms and IVS modeling. Section 3.4 presents our

SVR algorithm and its application to model IVS. Section 3.5 exhibits an empirical study using tick data from the S&P500 options market. Section 3.6 draws conclusions and presents future work.

3.2. Literature Review

A handful of financial applications using adaptive machine learning models have been recently developed. Chen et al. (2011) propose a bankruptcy prediction model based on an adaptive fuzzy k-nearest neighbor. The neighborhood size and the fuzzy strength parameter are updated over time by continuous particle swarm optimization. Li et al. (2012) apply an evolution strategy based support vector machine (SVM) to perform credit risk classification, which adapts the penalty term in the objective function according to time-varying data structures. Sun et al. (2013) put forward a method named adaptive and dynamic ensemble of SVM to predict corporate financial risk with focus on the concept drift of financial distress hidden in a corporate data flow. Booth (2016) explores the use of artificial neural nets, SVM, random forests and other machine learning methods in adaptive stock price return prediction and limit order book modeling. Similar to these applications, we emphasize on the ability to update the model upon occurrence of pattern drift, but our focus is on dynamic IVS modeling.

In financial market volatility forecasting, a few papers focus on SVR. Chang and Tsai (2008) introduce the combination of SVR, grey model and GARCH using artificial neural nets and show that the composite models perform better in volatility prediction than a time series method. Chen et al. (2010) apply SVR under the GARCH framework to forecast market volatility. They

conclude that SVM-GARCH models are better than all competing methods in most situations of one-period-ahead forecasting. Wang (2011) combines SVR and a stochastic volatility model with jump to form an efficient currency option pricing model. He claims that the new model reduces forecasting errors and outperforms artificial neural nets. While machine learning has been widely recognized in forecasting market volatility (refer to Hahn, 2013 for a detailed survey), IVS from the Black-Scholes-Merton model has not yet been extensively studied by machine learning approaches. A few examples are as follows. Malliaris and Salchenberger (1996) apply artificial neural nets to forecast S&P100 implied volatility with past volatilities and other options market factors. Fengler et al. (2007) model IVS dynamics using a semiparametric factor model by means of a principal component analysis, with empirical experiments using the DAX index options data. Lee et al. (2007) propose a particle swarm optimization method. Based on an analysis of the KOSPI 200 index options market, they find that their prediction yields option prices closer to theoretical values than generic algorithms. Audrino and Colangelo (2010) present a semi-parametric model by means of regression trees to forecast implied volatility and conduct an empirical study for S&P500 index options. All four papers assert promising results in implied volatility prediction, which further motivates us to explore an SVR application. To the best of our knowledge, SVR has not been tailored to model IVS, not to mention adaptive SVR.

SVR is the regression form of SVM. Usually, SVR is formulated as a dual optimization problem. For online training of the dual, Cauwenberghs and Poggio (2000) propose incremental and decremental support vector machine that can be used to bound the number of support vectors in a model and updates the model by one support vector at a time. The increments using matrix

manipulation are adiabatic, allowing the retention of Karush-Kuhn-Tucker conditions on all previous training data. In turn, the decrement step is a reversal of the increment by means of a leave-one-out procedure. Throughout the updating process, they require a book-keeping routine that migrates data points among different sets of support vectors: margin support vectors, error support vectors and (ignored) vectors within the margin. Ma et al. (2003) apply incremental and decremental SVM in a dual ϵ-SVR setting (Vapnik, 1998) (named accurate online SVR). Similar to accurate online SVR, our online primal ϵ-SVR algorithm entails a support vector adding and removal process, an online budget maintenance idea that is first proposed by Crammer et al. (2003) and thoroughly discussed in Wang, Crammer et al. (2012). Due to the primal setting, the model update rule requires much lower computational resources than incremental and decremental SVM. Budget maintenance of support vectors plays an important role in keeping the sparsity of an online model regardless of the primal or dual formulation; without it, the number of support vectors typically grows linearly with the number of training examples (Steinwart, 2003). In this work, we introduce the budget maintenance idea into the primal ϵ-SVR algorithm called Pegasos (adapted by us from its original SVM version, proposed by Shalev-Shwartz et al., 2007). Compared with a well-established dual formulation, the primal problem is much easier and faster to solve using stochastic sub-gradient descent (Shalev-Shwartz and Srebro, 2008). Similar stochastic gradient descent based methods are applied to SVM classification problems by Kivinen et al. (2004) and Zhang (2004), who use different learning rates than Pegasos. For a detailed comparison of large scale and online SVM methods, we refer the reader to Wang, Crammer et al. (2012).

An additional challenge is how to decide the upper limit of support vectors during budget maintenance. Liu and Zio (2016) embed the idea of FVS, first proposed by Baudat and Anouar (2003), into dual ϵ-SVR. Inspired by them, we include FVS in primal ϵ-SVR with their notions of new pattern and changed pattern. FVS is designed for kernel implementations targeting at complexity control of the size of feature basis, in our case, the number of support vectors. To insert a new feature (or support) vector, the rule of thumb is to determine if the mapping of a new data point is nonlinearly independent from existing support vectors in the reproduced kernel Hilbert space. If so, it is viewed as a new pattern that cannot be expressed as a linear combination of the mapping of existing support vectors and is immediately added into the support vector set. Unlike a new pattern, a changed pattern indicates that the mapping of the new data point is not linearly independent in the reproduced kernel Hilbert space, but the bias of its predicted value exceeds a predetermined threshold. In this case, an existing support vector is replaced by the changed pattern while the nonlinear independence of all support vectors in reproduced kernel Hilbert space is still preserved. Continuously adding support vectors by detecting new patterns and replacing support vectors by identifying changed patterns are critical steps in our algorithm that are essential for adaptive model update, sparsity preservation and computational cost/complexity/overfitting reduction. New patterns determine the number of support vectors needed while changed patterns tell us when and where budget maintenance thought support vector removal is to be performed. A similar method that involves adaptive quantity control of support vectors is ν-SVR (Schölkopf et al., 2000) that employs a different loss function than ϵ-SVR. Recently, Gu et al. (2015) combine ν-SVR with incremental and

decremental SVM and design a new online algorithm: incremental *v*-SVR (INSVR). However, the decremental (support vector removal) step is missing in their work.

During the training phase of our SVR algorithm, the inverse of the kernel matrix has to be constantly updated upon support vector insertion and replacement. To accelerate such computation, we implement the matrix inverse calculation in the FPGA hardware developed by Maxeler Technologies. Besides this, the prediction part of our algorithm also has its implementation in FPGA. In existing literature, many forms of SVM have been designed specifically for a parallel FPGA implementation, with recent examples such as CORDIC based SVM and SVR by Ruiz-Llata et al. (2010), a novel Cascade SVM by Papadonikolakis and Bouganis (2012), and an adapted Cascade SVM by Kyrkou et al. (2013). All of these papers are for inference only, due to the iterative nature of the training phase that is difficult to parallelize. We not only implement the inference in FPGA, but also parts of the training procedure.

3.3. Background

In this section, we review basics of ϵ-SVR, kernel Pegasos SVR and IVS modeling.

3.3.1. ϵ-SVR

Given a training set $S = \{(x_i, y_i)\}_{i=1}^{m}$, where $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}$, ϵ-SVR solves the following quadratic optimization problem

$$\min_{w,b} \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m} l\big(w; (x_i, y_i)\big), \tag{3.1}$$

where the ϵ loss function is

$$l\big(w;(x_i,y_i)\big)=\begin{cases}|y_i-f(x_i)|-\epsilon, & |y_i-f(x_i)|\ \geq\ \epsilon,\\ 0, & \text{otherwise,}\end{cases}$$

and the estimate function

$$f(x)=\langle w,\phi(x)\rangle+b. \tag{3.2}$$

The L2 norm in the objective function represents the regularization term, where λ is referred as a regularizing parameter that serves to shrink the overall model complexity. The second term is the average empirical error measured by loss function $l\big(w;(x_i,y_i)\big)$. Optimization in (3.1) penalizes data points whose *y* values differ from $f(x)$ by more than ϵ. In (3.2), $\phi(x)$ is a nonlinear mapping from input x to reproduced kernel Hilbert space; $\langle u,v\rangle$ denotes the standard inner product between vectors u and v; term b is the regression intercept.

Estimates of *w* and *b* can be obtained by solving the following equivalent model to (3.1):

$$\min_{w,b}\frac{\lambda}{2}\|w\|^2+\frac{1}{m}\sum_{i=1}^{m}(\xi_i+\xi_i^*)$$

subject to

$$y_i-f(x_i)\leq\epsilon+\xi_i,$$

$$f(x_i)-y_i\leq\epsilon+\xi_i^*,$$

$$\xi_i^*,\xi_i\geq 0,\qquad i=1,\dots,m.$$

Slack variables ξ_i and ξ_i^* measure the excess deviation of positive and negative errors. They are added to cope with the scenarios where no function $f(x)$ exists to satisfy the ϵ constraints by allowing regression error up to ξ_i^* or ξ_i.

3.3.2. Kernel Pegasos SVR algorithm

The dual formulation of SVR attracted more attention than the primal, with various versions of online dual SVR proposed based on incremental and decremental SVM (refer to Section 3.2). In contrast, we dedicate our effort to devising a primal online SVR, enhanced from a stochastic sub-gradient descent based SVM algorithm called Pegasos (Shalev-Shwartz et al., 2007), originally for classification. Compared with the dual, the primal formulation of SVR has an advantage of simplicity and can be easily adapted to the stochastic gradient descent method. Next, we derive the regression version of the Pegasos algorithm.

The convex optimization problem (3.1) can be rewritten as follows by substituting the loss function into the objective:

$$\min_{w,b} g(w,b;x_i,y_i) = \left[\frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m}(\max\{0, y_i - f(x_i) - \epsilon\} + \max\{0, f(x_i) - y_i - \epsilon\})\right]. \quad (3.3)$$

To solve (3.3), the stochastic sub-gradient descent method takes one random data point (x_i, y_i) at a time to estimate the sub-gradient of g, which reads

$$\nabla g_w = \lambda w + \begin{cases} \phi(x_i), & \text{if } f(x_i) - y_i - \epsilon > 0, \\ -\phi(x_i), & \text{if } y_i - f(x_i) - \epsilon > 0, \\ 0, & \text{otherwise.} \end{cases}$$

$$\nabla g_b = \begin{cases} 1, & \text{if } f(x_i) - y_i - \epsilon > 0, \\ -1, & \text{if } y_i - f(x_i) - \epsilon > 0, \\ 0, & \text{otherwise.} \end{cases}$$

With these sub-gradients, it is clear that the rules to update w and b are

$$w \leftarrow w - \frac{1}{\lambda t}\nabla g_w, \qquad b \leftarrow b - \frac{1}{\lambda t}\nabla g_b, \quad (3.4)$$

where t represents the current iterate index, and $\frac{1}{\lambda t}$ the learning rate. Substituting ∇g_w and ∇g_b into (3.4), we obtain

$$w \leftarrow \left(1 - \frac{1}{t}\right) w \pm \frac{\phi(x_i)}{\lambda t}, \qquad b \leftarrow b \pm \frac{\phi(x_i)}{\lambda t}, \tag{3.5}$$

if the sample falls outside the ϵ bound; otherwise,

$$w \leftarrow \left(1 - \frac{1}{t}\right) w. \tag{3.6}$$

One of the major benefits of SVR is the kernel trick that avoids direct access to the high-dimension mapping ϕ and only uses the inner products of samples specified through a kernel function. We next discuss how to embed the kernel trick with a support vector dictionary S. Every time a sample x falls out of the ϵ bound, it becomes a support vector if it is not a current support vector; coefficient w is updated by a discounted mapping $\pm \frac{\phi(x)}{\lambda t}$. This leads to creating a dictionary to keep track the cumulative sum of the discount factors $\pm \frac{1}{\lambda t}$ for each support vector. To be specific, the keys of S are comprised of current support vectors and their corresponding values are the cumulative sums of the discount factors. The regression coefficient w can be represented as

$$w = \sum\nolimits_{s \in S} S[s] \cdot \phi(s)$$

and we have

$$f(x) = \langle w, \phi(x) \rangle + b = \sum\nolimits_{s \in S} S[s] \cdot \phi(s)^T \phi(x) + b = \sum\nolimits_{s \in S} S[s] \cdot K(s, x) + b, \tag{3.7}$$

where K is a nonlinear kernel function with $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. The kernel trick allows a feature space of arbitrary dimensionality without explicit computation of the map $\phi(x)$. As long as a function satisfies the Mercer conditions (Vapnik, 1998), it can be used as a kernel function.

The kernel Pegasos SVR (KPSVR) algorithm is exhibited in Algorithm 3.1. Parameter T denotes the maximum number of iterations. Step 2.b uses the primal form of the estimate function (3.7). Step 2.c replaces w by S in (3.5) and (3.6). Step 2.d updates the support vector dictionary if the new sample lies outside the ϵ bound.

Algorithm 3.1 – Kernel PSVR (KPSVR)

1. Initialize $S = \emptyset$
2. For $t = 1, \ldots, T$
    a. Randomly sample (x_t, y_t)
    b. Predict $f(x_t)$ by iterating all keys in S and using the kernel trick

$$f(x_t) \leftarrow \sum_{s \in S} S[s] \cdot K(s, x_t) + b$$

    c. $S[s] \leftarrow \left(1 - \frac{1}{t}\right) S[s]$ for all $s \in S$
    d. If $|y_t - f(x_t)| > \epsilon$, then x_t is a support vector

        If key x_t is in S, $S[x_t] \leftarrow S[x_t] \pm \frac{1}{\lambda t}$; else insert a key value pair, $S[x_t] \leftarrow \pm \frac{1}{\lambda t}$

        Additionally, $b \leftarrow b \pm \frac{1}{\lambda t}$

3.3.3. IVS modeling

The implied volatility surface (IVS) is a mapping from the strike prices κ and time to maturity τ of options to a nonnegative value – implied volatility, i.e. a mapping

$$\tilde{\sigma}_t^{IV}: (\kappa, \tau) \mapsto \mathbb{R}.$$

Implied volatility at a given point and measured as the standard deviation of the rate of return of the underlying asset is obtained by plugging the option price, the price of the underlying asset, the risk-free rate (estimated by Treasury yield in this chapter), κ and τ into the Black-Scholes-Merton formula and back-solving for implied volatility. Since there is no closed-form solution for computing implied volatility, typical methods are by bisection or Newton-Raphson. Implied volatility is valuable for comparison of options with dissimilar characteristics such as different underlying, strike, time to maturity, etc. Although the Black-Scholes-Merton model assumes constant volatility across all options, empirical evidence shows the existence of the volatility smile and skew among a cross-section of options. Moreover, the IVS is not static; it changes over time and thus requires adaptive updates.

To model the IVS, we turn to a parametric quadratic volatility function introduced by Dumas et al. (1998). The following ad hoc model has been proven to be a simple yet robust method (usually the best among all competing functional forms) to approximate the IVS:

$$\tilde{\sigma}_t^{IV}(\kappa, \tau) = \alpha_0 + \alpha_1\kappa + \alpha_2\kappa^2 + \alpha_3\tau + \alpha_4\kappa\tau.$$

It explores the variation in volatility to asset price and time. The quadratic form is chosen due to the parabolic shape of the IVS and an attempt to avoid over-parametrization. In our kernel SVR setting, this function translates into a 4-dimension representation of each data point $(\kappa, \kappa^2, \tau, \kappa\tau)$,

which can be further substituted into a kernel function to calculate the dot products between two samples.

3.4. Method

In this section, we describe FVS, budget maintenance, our enhanced kernel Pegasos SVR algorithm, its adaptation to IVS modeling and how FPGA technology is applied to accelerate the computationally intensive parts of our algorithm.

3.4.1. FVS

The idea of FVS is first proposed by Baudat and Anouar (2003) to select "feature" vectors from a data set and form a basis in the reproduced kernel Hilbert space that can express other data points by projection, i.e. a linear combination of the mapping of selected vectors. In our SVR setting, FVS is treated as a natural way to add support vectors and control the size of the support vector set. Furthermore, FVS is designed specifically for the kernel trick, enabling a seamless integration with SVR.

To determine if a new data point can be spanned by existing support vectors, the following statistic, named *local fitness*, is calculated:

$$J_{S,x} = \frac{k_{S,x}^T K_{S,S}^{-1} k_{S,x}}{k_{x,x}}, \tag{3.8}$$

where S denotes the current support vector set, x is a new data point, $K_{S,S}$ represents the kernel matrix, $k_{S,x}$ denotes the kernel vector of dot products between x and the support vectors, $k_{x,x}$ is the dot product of x mapping itself.

Local fitness functions as an approach to measure the maximum possible collinearity between the original data mapping and the approximation using a linear combination of the mapping of support vectors (Baudat and Anouar, 2003). In their original work, FVS is an iterative process of forward selection that repeatedly samples the entire data set to find the next support vector with smallest local fitness. This searching process is terminated when the maximum number of support vectors is reached, or the average local fitness of all data points (called *global fitness*) exceeds a certain threshold, or a complete basis is found. Since our SVR is an online algorithm, their framework does not fit our need. Instead we enforce a threshold criterion to enlarge the support vector set, i.e. add a new data point as a new support vector if its local fitness is smaller than a given threshold ρ, in which case the new data point cannot be sufficiently approximated by any linear combination of the mapping of existing support vectors (thus the invertibility of $K_{S,S}$ and its nonlinear independence from existing support vectors are guaranteed). The new data point is aliased as a *new pattern*. Note that a smaller ρ leads to a lower number of support vectors, and vice versa. Choosing a good ρ is hence important to help noise reduction while keeping a sufficient number of support vectors for satisfactory model performance.

3.4.2. Budget maintenance

When a new data point x arrives that is not a new pattern (i.e. a large local fitness is present), we ought to further check if it represents a *changed pattern*. A changed pattern occurs if its prediction error by the current model surpasses a certain limit ϵ, indicating that the support vector set needs an adaptive update: a removal of an existing support vector (old pattern) and an insertion of the new data point (changed pattern). The number of support vectors, however, remains unchanged since it is controlled by FVS.

To determine which support vector to remove, Liu and Zio (2016) put forward a contribution based method by deleting the least contributing support vector. Wang, Crammer et al. (2012) discuss three budget maintenance ideas that fix the number of support vectors to a pre-specified value B (in our case, a value controlled by FVS): support vector removal, projection and merging. Support vector projection projects a support vector onto remaining support vectors while support vector merging merges two support vectors and creates a new one. In our primal setting, removal is much easier to accomplish for budget maintenance purpose, which also results in less kernel matrix manipulations than projection and merging. Since we already have the support vector dictionary S, we could simply remove the key with the smallest absolute value in S, known as a process that leads to the least gradient error or, equivalently, the least weight degradation (Wang, Crammer et al., 2012). In the following analysis, budget maintenance refers to support vector removal.

Yet this works only for the Gaussian kernel. For a general kernel function K, we remove the support vector key with least $S[s] \cdot \phi(s)$ or based on the kernel trick $(S[s])^2 \cdot K(s,s)$. It is easy to see that this rule in the case of Gaussian kernel, which has $K(x,x) = 1$ for any x, is the same as least weight degradation. The support vector dictionary we create serves two purposes: a regression coefficients container as in (3.7) and a reference for budget maintenance. After the old pattern has been removed, the changed pattern x is added to S.

Every time the keys in dictionary S are modified, the kernel matrix $K_{S,S}$ demands an update. More challenging, the inverse of $K_{S,S}$ in (3.8) needs to be recalculated. Since only one support vector is added or removed at a time in our algorithm, we must be able to efficiently manage these matrix inverse computations. Baudat and Anouar (2003) propose a method that only deals with support vector addition, which does not work directly on the inverse matrix. Nonetheless, their method cannot be extended to the case of support vector deletion. In the following, we derive the formulas for updating the kernel inverse upon support vector addition and removal.

Suppose we have a working set of n support vectors forming kernel matrix K_n (we leave out subscript S,S for notation simplicity) and its inverse K_n^{-1}. Let us assume we want to add a new support vector x, and update the kernel matrix by appending a new column and a new row:

$$K_{n+1} = \begin{pmatrix} K_n & k_{S,x} \\ k_{S,x}^T & k_{x,x} \end{pmatrix},$$

where $k_{S,x}$ denotes an $n \times 1$ vector of dot products between x and the previous n support vectors.

Let

$$K_{n+1}^{-1} = \begin{pmatrix} X & Y \\ Y^T & z \end{pmatrix} \tag{3.9}$$

be the updated inverse matrix, where X is an $n \times n$ matrix, Y is an $n \times 1$ vector, and z is a scalar. The inverse matrix is symmetric because the kernel matrix is always symmetric.

The solutions for X, Y and z are

$$\begin{aligned} z &= 1/\left(k_{x,x} - k_{S,x}^T K_n^{-1} k_{S,x}\right), \\ Y &= -z K_n^{-1} k_{S,x}, \\ X &= K_n^{-1} - K_n^{-1} k_{S,x} Y^T. \end{aligned} \tag{3.10}$$

The denominator of z is never zero, because otherwise, the local fitness of the new data point x is one, meaning it is an existing support vector and cannot be inserted into the support vector dictionary once again (assuming $\rho < 1$, which is reasonable).

Upon support vector deletion, we are given K_{n+1}^{-1} as in (3.9) and after deleting Y we obtain:

$$K_n^{-1} = X - \frac{YY^T}{z}. \tag{3.11}$$

Updating the kernel matrix is straightforward by removing the row and column corresponding to the deleted support vector.

3.4.3. Enhanced KPSVR algorithm

We first incorporate budget maintenance and FVS to KPSVR. Wang, Crammer et al. (2012) discuss how budgeted SVM can improve computational efficiency in both time and space, but

with no mentioning of its potential extension to SVR. Algorithm 3.2 exhibits the budgeted KPSVR algorithm. Step 2.e removes the support vector with the least absolute value if the maximum number is exceeded.

Algorithm 3.2 – Budgeted KPSVR (BKPSVR)

---

1. Initialize $S = \emptyset$
2. For $t = 1, \ldots, T$

   Step 2.a to 2.d from KPSVR

   e. If $|S| > B$, select the key s in S with the smallest $(S[s])^2 \cdot K(s, s)$, remove its key value pair

---

A fixed number of support vectors may not be optimal when a new pattern emerges or when data patterns are continuously changing, in which cases the number of support vectors should adapt responsively. This is addressed by incorporating FVS into BKPSVR. Once a sample x cannot be sufficiently approximated by any linear combinations of the mapping of existing support vectors (i.e. a small local fitness $J_{S,x}$ that is less than the preset threshold ρ), it is added into the support vector dictionary S as a new pattern without checking if it is a changed pattern. Otherwise, if its prediction is not within the ϵ bound, we call it a changed pattern that further activates budget maintenance. Algorithm 3.3 presents the enhanced KPSVR algorithm with FVS and adaptive updates of support vectors through budget maintenance. In particular, Step 2.d is modified to detect new patterns and changed patterns, where support vector addition and budget maintenance are conducted. Upon support vector insertion into or deletion from S, formulas (3.9) and (3.11)

are used to efficiently update the matrix inverse $K_{S,S}^{-1}$ so that it is ready to calculate local fitness $J_{S,x}$ in the next iteration.

Algorithm 3.3 – Enhanced KPSVR (EKPSVR)

---

1. Initialize $S = \emptyset$
2. For $t = 1, \dots, T$

   Step 2.a to 2.c from KPSVR

   a. If local fitness is violated, i.e. $J_{S,x_t} < \rho$, then x_t is a new support vector (new pattern)

      Add key x_t into S, $S[x_t] \leftarrow \pm\frac{1}{\lambda t}$, $b \leftarrow b \pm \frac{1}{\lambda t}$

   Else if $|y_t - f(x_t)| > \epsilon$, then x_t is a support vector (changed pattern)

      If key x_t is in S, $S[x_t] \leftarrow S[x_t] \pm \frac{1}{\lambda t}$; else select the key s in S with the smallest $(S[s])^2 \cdot K(s,s)$, remove its key value pair, then insert key x_t, $S[x_t] \leftarrow \pm\frac{1}{\lambda t}$

      Additionally, $b \leftarrow b \pm \frac{1}{\lambda t}$

3.4.4. IVS modeling by EKPSVR

To model the constantly fluctuating IVS, the stochastic gradient descent based EKPSVR algorithm has to be tailored to reflect the online nature of the training and predicting process using market data.

If EKPSVR is directly applied to model IVS without further modification, then later in time when t gets large enough, newly received data barely influences the model (with small step size $\pm\frac{1}{\lambda t}$ close to 0), which is then almost unchanged and fails to capture regime changes. As a result, reopening is necessary by reinitiating $t = 1$ at the end of a certain interval for adjustments to latest market conditions. For instance, the interval might be market opening or based on empirical evidence that uses minute level frequency in the context of intraday tick data. We name such an interval a *reopening interval*. To inherit models from previous intervals upon reopening, we adjust the learning rate from $\frac{1}{\lambda t}$ to $\frac{1}{\lambda(t+\omega)}$ by introducing a positive warm-start hyper-parameter ω into the denominator (otherwise the model would be completely retrained since Step 2.c would have $1 - \frac{1}{t} = 0$).

Algorithm 3.4 finalizes the online IV-EKPSVR algorithm. Upon arrival of a new tick, regression errors are recorded and the model is updated according to local fitness and prediction bias, after which a new IVS prediction is made. Once reaching the end of a reopening interval, t is reset to 1.

Algorithm 3.4 – Online IVS-EKPSVR

1. Initialize $t = 1, S = \emptyset$
2. Loop
    a. Receive a new observation (x_t, y_t) at time t (where x_t is the feature vector and y_t is the computed IV value based on x_t)
    b. Obtain $f(x_t)$ from the predicted IVS
    c. $S[s] \leftarrow \left(1 - \frac{1}{t+\omega}\right) S[s]$ for all $s \in S$
    d. If local fitness is violated, i.e. $J_{S,x_t} < \rho$, then x_t is a new support vector (new pattern)

        Add key x_t into S, $S[x_t] \leftarrow \pm\frac{1}{\lambda(t+\omega)}$, $b \leftarrow b \pm \frac{1}{\lambda(t+\omega)}$

        Else if $|y_t - f(x_t)| > \epsilon$, then x_t is a support vector (changed pattern)

        If key x_t is in S, $S[x_t] \leftarrow S[x_t] \pm \frac{1}{\lambda(t+\omega)}$; else select the key s in S with the smallest $(S[s])^2 \cdot K(s,s)$, remove its key value pair, then insert key x_t, $S[x_t] \leftarrow \pm\frac{1}{\lambda(t+\omega)}$

        Additionally, $b \leftarrow b \pm \frac{1}{\lambda(t+\omega)}$
    e. Predict IVS for each strike price and maturity of interest by (7) with the updated support vector dictionary
    f. $t \leftarrow t + 1$
    g. If the end of current reopening interval is reached, reset $t = 1$

3.4.5. FPGA implementation

Field Programmable Gate Arrays (FPGA) hardware has been widely used in the high frequency trading sector to accelerate and reduce the latency of packet capture (e.g. FIX/FAST messages), order book modeling, theoretical price calculations and other finance statistic evaluations (e.g. option greeks). In this work, an FPGA embedded server MaxWorkstation10G (developed by Maxeler Technologies) is adopted for parallel computing. This powerful server is equipped with one Vectis dataflow engine (DFE) and Intel Core i7 quad-core CPU with 16GB RAM. The Vectis board includes a Xilinx Virtex-6 SX475T FPGA, where highly parallelizable computations are performed. MaxWorkstation10G is a connectivity development platform with CPU and DFE connected via PCI Express gen2 x8, guaranteeing its ultra-low latency (Figure 3.1). Compared with Graphical Processing Unit (GPU) based accelerators, reconfigurable FPGA implementations enjoy lower power consumption alongside its strong capability in high-performance computing, but at the expense of ease of programming. Unlike conventional FPGAs, Maxeler FPGA solutions offer extended flexibility and significantly improve the programming experience (with MaxIDE in MaxelerOS). They developed a customized Java-based language to program the DFE kernels, which specifies computational logic, and the DFE managers, which connect the data flows among CPU, kernels, and memories (fast on-chip memory FMem or large off-chip memory LMem). Note that data flows are streamed into and out of the DFE, meaning that additional data handling is necessary, for example, matrix serialization. By Simple Live CPU interface (SLiC), the FPGA application can be embedded into a number of major programming languages such as C/C++, Java, python, MATLAB, R etc.

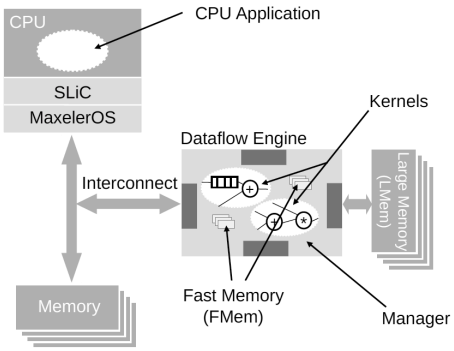


Figure 3.1: Maxeler dataflow engine architecture (Courtesy of Maxeler tutorial)

As presented in Section 3.4.2, the support vector insertion and deletion in Step 2.d of IVS-EKPSVR require kernel matrix inverse updates by formulas (3.9) and (3.11), which are computationally intensive and thus become a good candidate for FPGA acceleration. Essentially, these matrix updates are a number of nested-for loops that are highly parallelizable. We also notice that the local fitness calculation (at the beginning of Step 2.d) and the predictions for each sample (Step 2.e) consist only of nested-for loops. Based on these observations, four parts of the IVS-EKPSVR algorithm are prime candidates for DFE: support vector addition, support vector removal, local fitness, and sample prediction.

Consider inference (Step 2.e). To make these predictions, two for loops are required – one sweeps all data points and another scans all support vectors. Its pseudo code is as follows (denote the prediction output vector as p, the number of samples as M, the number of support vectors as N).

Initialize $p[i] = 0$ for all $i = 1, \dots, M$
For i from 1 to M
    For j from 1 to N
        $p[i] = p[i] + S[j] \cdot K_{S,S}[i, j]$
Output p

Our DFE implementation is shown in Figure 3.2 (assuming there are 3 support vectors). The outer loop in the CPU code is the target of parallelization since the predictions for each sample are independent. The support vector related data is stored in the read-only memory (ROM) in FPGA. Each time a new sample is streamed into DFE, it is then distributed into every ROM and multiplied with the support vector to calculate $S[j] \cdot K_{S,S}[i, j]$. Then, we sum all these contributions to obtain the prediction output for this particular data point.

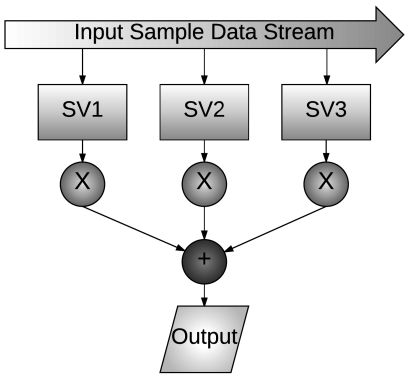


Figure 3.2: DFE design for sample prediction

Next we discuss local fitness. The formulas to calculate $J_{S,x}$ in (3.8) and z in (3.10) are very similar. In (3.8), the numerator is matrix multiplication $k_{S,x}^T K_{S,S}^{-1} k_{S,x}$ and the denominator is a scalar. In (3.10), the solution to z also involves a matrix multiplication, similar to $k_{S,x}^T K_{S,S}^{-1} k_{S,x}$. They can indeed share the same DFE design. We develop a two-step data flow for this matrix

multiplication: multiply $k_{S,x}^T K_{S,S}^{-1}$ first, resulting in an intermediate row vector denoted I; then multiply $I \cdot k_{S,x}$ and output the desired scalar c. The pseudo code is given below.

```
Initialize I[i] = 0 for all i = 1, …, N
Initialize c = 0
For i from 1 to N
        For j from 1 to N
                I[i] = I[i] + k_{S,x}[j] · K_{S,S}^{-1}[i, j]
        c = c + I[i] · k_{S,x}[i]
Output c
```

The corresponding DFE implementation is presented in Figure 3.3. In both the left and right DFE kernels, we map the vector $k_{S,x}$ into on-chip ROM, meaning each box in the figure represents an element of $k_{S,x}$ (suppose the kernel inverse matrix is 3-by-3). The inverse matrix columns of $K_{S,S}^{-1}$ are then streamed into the left DFE kernel and multiplied with each element of $k_{S,x}$, which yields $k_{S,x}[j] \cdot K_{S,S}^{-1}[i,j]$. By summing up these multiplications, the intermediate row vector I is attained. Vector I is then streamed into the next DFE kernel on the right and multiplied with each vector element of $k_{S,x}$ to obtain $I[i] \cdot k_{S,x}[i]$. Finally, summing these multiplications yields the desired scalar c.

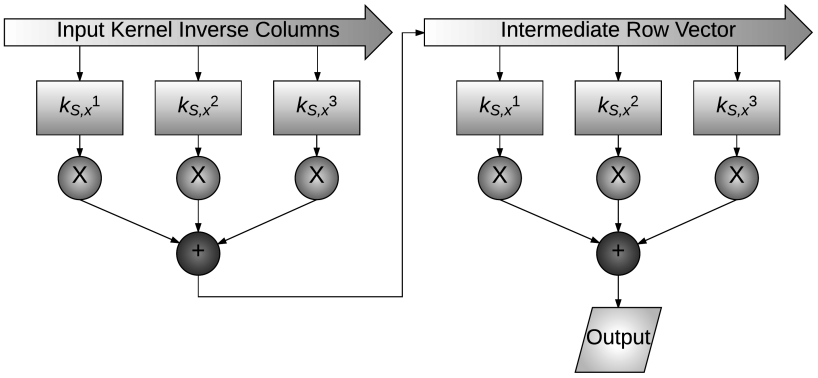


Figure 3.3: DFE design for two-step matrix multiplication

Once the intermediate row vector I and scalar z are obtained, vector Y in (3.9) immediately follows by multiplying I^T with $-z$.

Finally we discuss support vector deletion and addition. This amounts to computing X in (3.10) and the inverse matrix update upon support vector deletion using (3.11). Note that the solution to X in (3.10) requires the intermediate row vector as well, therefore we rewrite

$$X = K_n^{-1} - I^T Y^T,$$

a form identical to (3.11). This means that these two parts can share the same data flow design again. The pseudo code and DFE implementation are omitted due to its resemblance to the previous two cases. The basic idea is that for each element in matrix X or K_n^{-1}, depending on the formulas, we subtract its value by the product of corresponding elements in vectors I^T and Y^T or the product of scalar $1/z$ and elements in vectors Y and Y^T. Matrix X or K_n^{-1} is streamed into the DFE, while two vectors and a scalar (in the X case, the scalar is 1) are mapped onto the on-chip ROM.

3.5. Computational Study

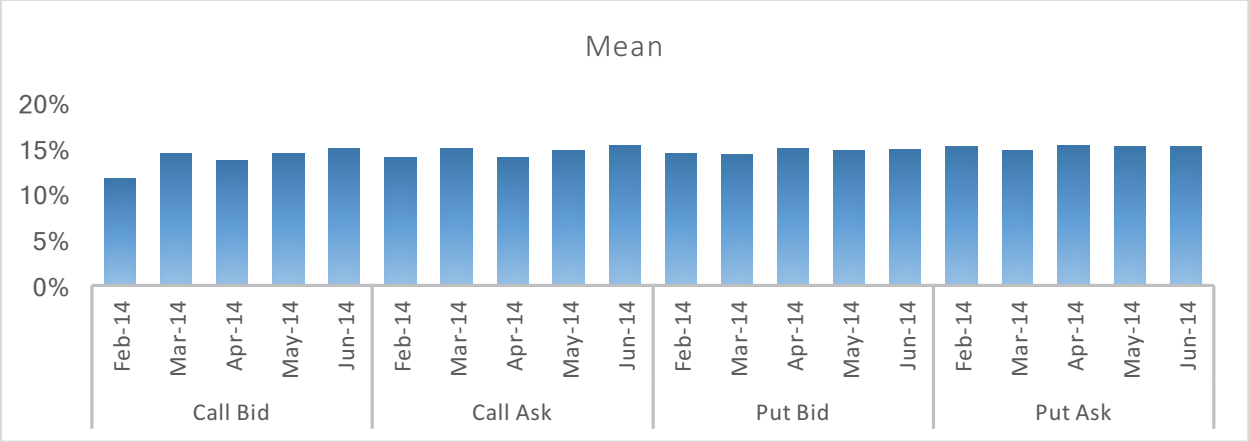
Using empirical data from the E-mini S&P 500 futures and options market, in this section we present a computational study that compares our IVS-EKPSVR algorithm against competing methods.

3.5.1. Data

The E-mini S&P 500 option has the E-mini S&P 500 future as the underlying asset, both traded in the Chicago Mercantile Exchange (CME). The options tick data used for this study is based on dates 01/27/2014 to 01/31/2014 and contains 5 maturities: February to June 2014. Each tick represents the latest top level of a limit order book. The trading hours are Sunday to Friday 5 pm to 4 pm Central Time with a halt from 3:15 pm to 3:30 pm, and a 60-minute break beginning at 4 pm. These non-trading time periods are excluded from our experiments. Although trading activity can occur almost any time in a trading day, the busiest hours are from 9 am to 4 pm. For example, over 54.4 million ticks are recorded during this time period out of 79.9 million on 01/27/2014, i.e. approximately 70% ticks in 30% time of a day. For this reason we built our models only for these hours. The moneyness of options is defined as the ratio of strike price divided by the underlying asset price. Because out-the-money and in-the-money options are less traded in a high-frequency intra-day setup, we limit the moneyness of options to 0.95 to 1.05 (i.e. at-the-money or ATM). Their matching strike prices are determined by the settlement price of the underlying futures in the previous trading day of 01/27/2014, which is 01/24/2014. The total number of data points on each modeled IVS (Call Bid, Call Ask, Put Bid and Put Ask) is 200, i.e. 40 strike prices of the ATM options for each of the 5 maturities. These samples on the strike-

maturity grid display varying values of implied volatility over time and thus become the targets of our online prediction models.

Given the price data, implied volatilities are computed using the Black-Scholes formula with interest rates linearly interpolated from the daily Treasury yield curve. Figure 3.4 summarizes the statistics of the average implied volatility from 9 am to 4 pm on 01/27/2014 (details can be found in Appendix B). Generally, Feb 2014 maturity shows the most volatile properties with the largest standard deviations. The longer the maturity, the smaller the standard deviation. Another observation is that put options are on average priced higher than call options by examining the mean of implied volatility.



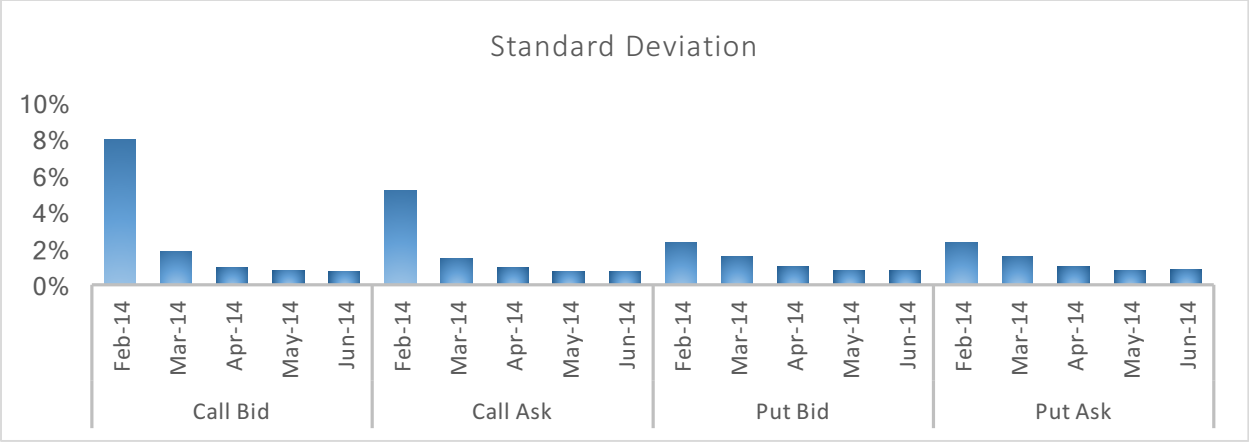
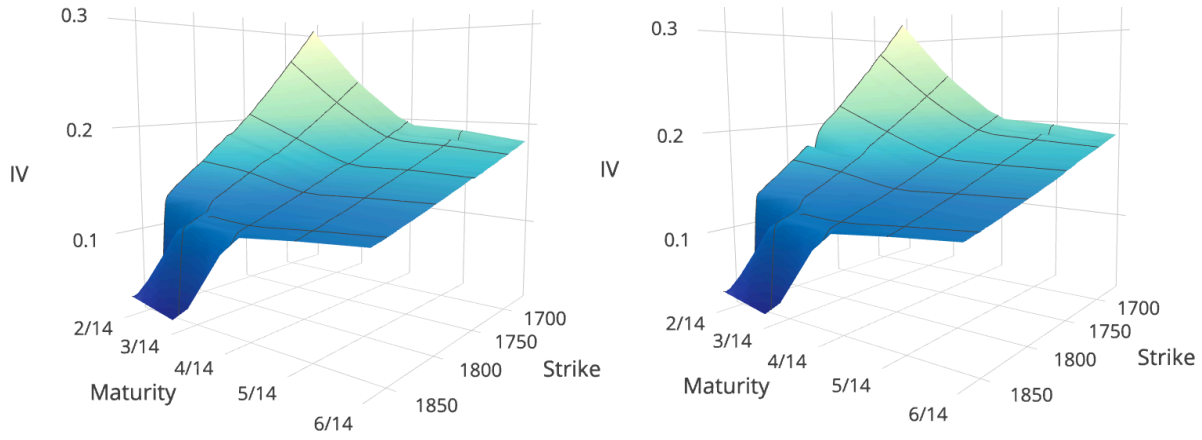


Figure 3.4: Summary statistics of IV

Figure 3.5 presents the Call Bid, Call Ask, Put Bid and Put Ask IVS models for ATM options at 9 am on 01/27/2014 (strike prices range from 1,670 to 1,865 with a discrete increment of 5). Volatility smile can hardly be identified but volatility skew exists in all four surfaces. It can also be observed that dramatic changes of implied volatility appear on the higher end of strike prices for shorter maturities. The above observations not only apply to 01/27/2014, but all other four days.



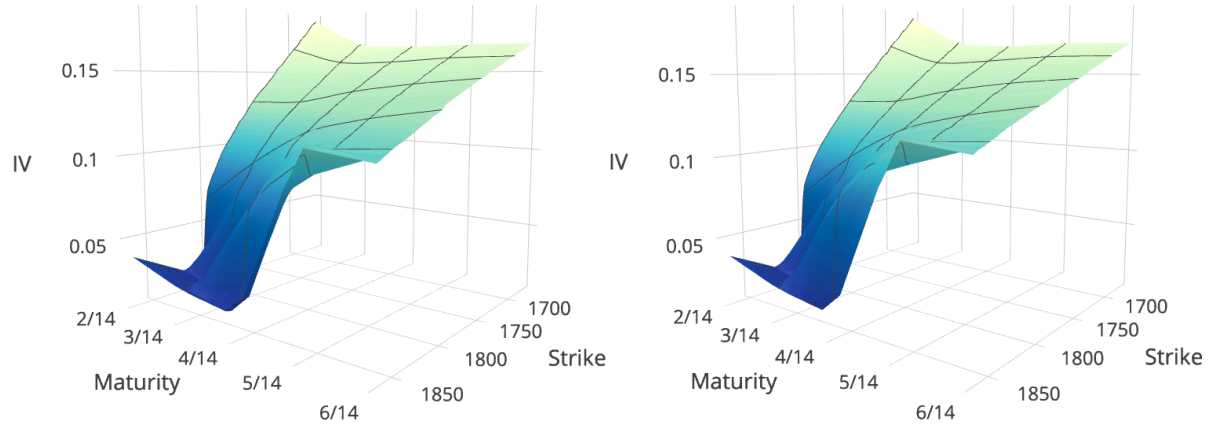


Figure 3.5: IVS

(From left to right, top to bottom: Call Bid, Call Ask, Put Bid, Put Ask)

3.5.2. Results

Our algorithms are developed in C++ and a Java-based FPGA programming language on a load-free MaxWorkstation10G with 4.0 GHz Intel Core i7 processor and 16 GB of RAM.

In the implementation, once the top level of a limit order book is updated by a new tick in C++ (on the CPU), that tick is then streamed into the FPGA hardware (or DFE) to update the SVR model by examining its local fitness, updating the kernel matrix inverse upon support vector addition or removal and finally to predict the entire IVS (see Section 3.4.5 for details). In between these operations, C++ functions as a data transfer medium that serializes, recovers and stores vectors and matrices, and connects input and output data flows with the PCI Express portal of the FPGA hardware. A counterpart implementation that accomplishes the same computations exclusively in C++ has also been developed for a comparison purpose (we call it a pure CPU implementation).

Experiments are implemented on a slightly modified version of the online IVS-EKPSVR. In Algorithm 3.4, models are continuously updated at each tick and so does the IVS prediction. Empirically, tick-by-tick prediction and evaluation are not necessary since the surface does not vary drastically within a short period. This fact motivates us to delay prediction and error evaluation until receipt of a certain number of ticks or elapse of a certain amount of time. For simplicity, we set such time point to be the end of a reopening interval, where regression errors are recorded and prediction for the next interval is performed. Due to our intraday setting, the length of reopening intervals is set to be 1 minute, a lower limit of common choices between 1 to 5 minutes (Hansen and Lunde, 2005) due to higher liquidity of ATM options. Similar minute-level intervals are used by Bollerslev et al. (2009) and Sévi (2014) for estimation of volatilities using tick data. Since the IVS prediction is delayed, Step 2.b of Algorithm 3.4 then follows Step 2.b of KPSVR to obtain the predicted value of a new observation using the latest support vector dictionary instead of extracting it from previously predicted IVS.

The hyper-parameters chosen for the IVS-EKPSVR algorithm are as follows: $\rho = 0.3, \lambda = 0.75, \omega = 7, \epsilon = 0.01$. SVR is equipped with the Gaussian kernel[1] using $\gamma = 0.25$. They were selected based on calibration with data from 01/27/2014. After fixing these hyper-parameters, the other four days essentially form a hold-out set used to assess all algorithms. We do not carry a model from the previous day to the next, i.e. the training process is restarted every morning, but the order book is constantly updated upon receipt of new ticks. Model fitting starts at 8 am, and

[1] Gaussian kernel function: $K(x, y) = exp(-\gamma|x - y|^2)$.

inference begins at 9 am. This one-hour lag provides a warm start for inference tasks each day. Both model update and prediction end at 4 pm daily with a halt from 3:15 to 3:30 pm.

We first present select behavior on the validation date of 01/27/2014. Figure 3.6 exhibits the implied volatility prediction time series of IVS-EKPSVR for options with a strike price of 1,770 (right ATM) on this date. The sequence-axis represents the time sequence discretized by 1 minute from 9 am to 4 pm (3:15 to 3:30 pm excluded) with 0 representing 9 am. Models from our algorithm adaptively adjust themselves and lead to predictions varying with the shifting market conditions. Specifically, prediction errors behave as if they are white noises (corresponding plots are omitted here), with absolute prediction error averaging 0.87%, 0.86%, 0.68%, 0.68% for Call Bid, Call Ask, Put Bid and Put Ask IVS models, respectively. Prediction error is relatively large on the edge of the strike-maturity grid for options with the shortest and longest maturities, because these options have less reference points to infer their implied volatilities during online learning. The aforementioned observations not only apply to strike 1,770, but also to other strikes and days.

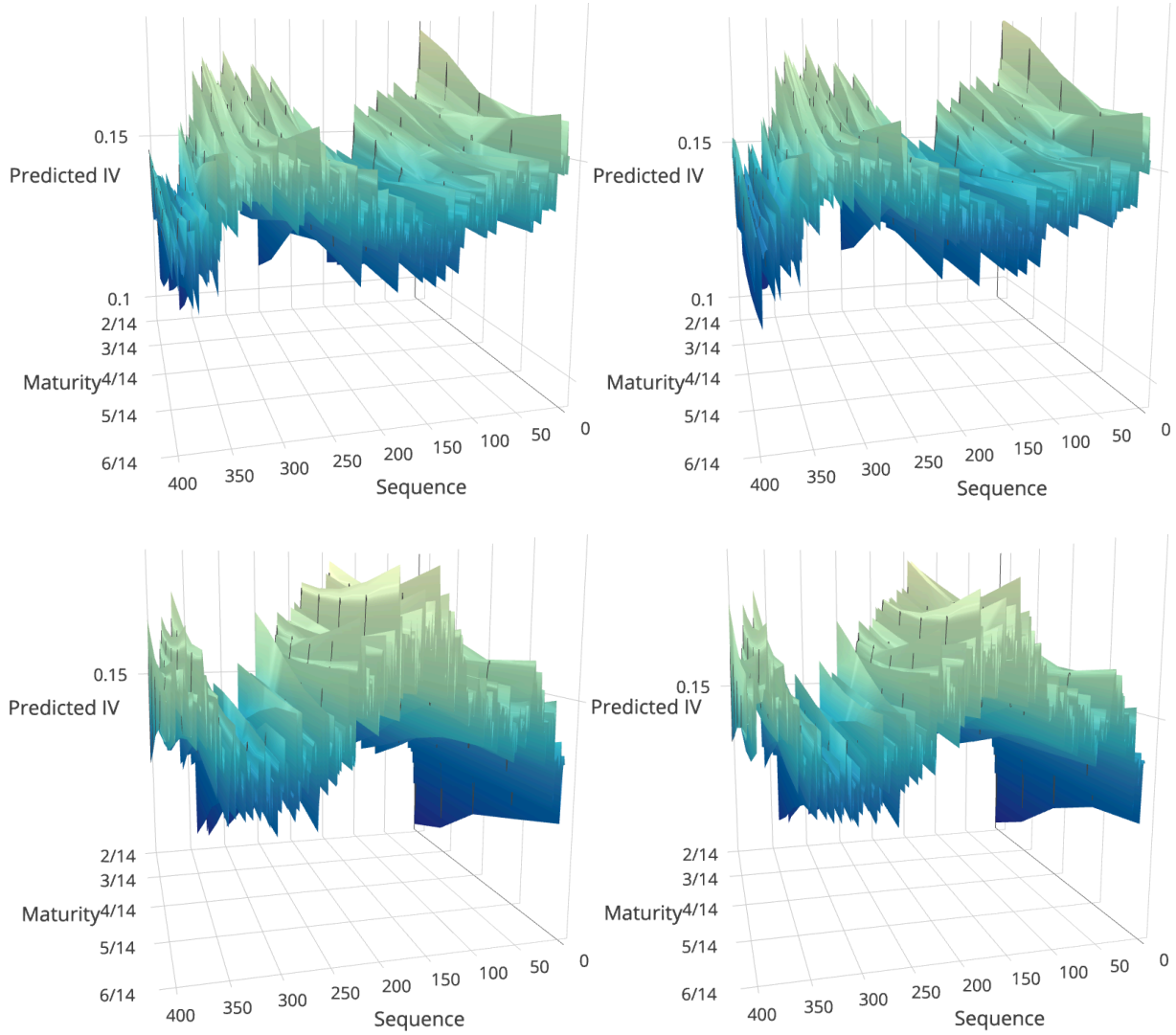


Figure 3.6: IV prediction

(From left to right, top to bottom: Call Bid, Call Ask, Put Bid, Put Ask)

We now explore a few competing models and contrast their average performance against IVS-EKPSVR with data from 01/28/2014 to 01/31/2014. For consistency, the same strike-maturity grid is used for these four days as on 01/27/2014. As a simplification, KPSVR provides a baseline that excludes any enhancements, and BKPSVR embeds budget maintenance. Similar to Algorithm 3.4, identical IVS prediction schemas are attached to these benchmarks, naming them

to IVS-KPSVR and IVS-BKPSVR correspondingly. Average performance statistics across the four days are shown in Table 3.1 and Figure 3.7 (details can be found in the Appendix B). Table 3.1 is about the average support vector sizes over the four days. Note that for budget maintenance purpose, the support vector size in IVS-BKPSVR is set to 50. We choose this value as a complement to the other two algorithms to reveal the performance when a quarter of available data points are considered support vectors while the other two discuss scenarios where all or half samples are used as support vectors. Figure 3.7 presents the differences of performance measured by average minute-by-minute MAPE (mean average percentage error) and RMSE (root mean square error) over the four days between IVS-EKPSVR and IVS-KPSVR or IVS-BKPSVR.

Figure 3.7 particularly asserts that IVS-EKPSVR and IVS-KPSVR are comparable in performance metrics with the latter slightly outperforming the former. Both clearly substantially beat IVS-BKPSVR. To verify that IVS-EKPSVR and IVS-KPSVR perform equivalently in a statistical sense, we perform a two-sample *t* test with the null hypothesis that these two algorithms have the same mean MAPE. Using the Gaussian kernel, the *p* values are 82.51%, 49.89%, 52.90% and 74.79% for Call Bid, Call Ask, Put Bid and Put Ask IVS models respectively, meaning that the null hypothesis cannot be rejected. The same test between IVS-EKPSVR and IVS-BKPSVR yields *p* values of 0.91%, 0.09%, 3.37% and 1.86% implying that the mean of IVS-EKPSVR is lower than the mean of IVS-BKPSVR with 95% confidence. Analogous conclusions can be drawn for the linear kernel. However, peeking at Table 3.1 we observe that IVS-EKPSVR uses substantially fewer support vectors in the Gaussian kernel case

than IVS-KPSVR, which does not bound the total number of support vectors and keeps adding support vectors provided that the ϵ condition is violated. Thus, in the call bid and ask models, the number of support vectors reaches the maximum 200 (the total number of data points on the strike-maturity grid, i.e. all samples) while the put bid and ask models arrive at 199. FVS in IVS-EKPSVR functions as a support vector size controller. It only adds a support vector when local fitness $J_{S,x}$ of a new sample x is smaller than the preset threshold ρ. In the Gaussian kernel case, IVS-EKPSVR uses around 50 ~ 60% of support vectors as in IVS-KPSVR, but results in a similar performance that can be seen from Figure 3.7. Nevertheless, it distinguishes itself from IVS-BKPSVR by dynamic support vector size tuning and further reduction in error rates. In essence, it finds a balance point where it stops increasing model complexity once performance reaches its limit. In the linear kernel case the difference is not that pronounced. We now conclude that IVS-EKPSVR achieves the same performance numbers as the best of the two competing algorithms but with fewer support vectors and is thus the preferred choice of the algorithm.

The actual MAPE values in all settings range from 12% to 15% while RMSE is from 1.5% to 2.5%.

Taking a closer look at Table 3.1 and Figure 3.7, we obtain the following observation that under IVS-EKPSVR, the Gaussian kernel behaves better than the linear kernel, whereas the linear kernel acts analogous to the baseline IVS-KPSVR, encompassing almost entire samples into the support vector space and achieving similar performance to IVS-KPSVR. Table 3.2 exhibits the

two-sample *t* statistics and *p* values of MAPE, with a null hypothesis that the linear and Gaussian kernel share the same mean MAPE. By examining Table 3.2, we observe that when support vector size is the same, the linear kernel performs comparable to the Gaussian kernel in IVS-KPSVR and IVS-BKPSVR since a large *p* value does not reject the null hypothesis. With much more support vectors, the linear kernel does not lead to significant improvement over the Gaussian kernel in IVS-EKPSVR. This being said, the Gaussian kernel works better for our online algorithm.

As a final note based on Table 3.1 and Figure 3.7, we find that a larger number of support vectors leads to smaller MAPE and RMSE but at the expense of more computational resources to perform kernel matrix manipulations, a conclusion that can be further justified by the sensitivity analysis presented in the latter part of this section.

Table 3.1: Support vector size

	Kernel	Call Bid	Call Ask	Put Bid	Put Ask
IVS-KPSVR	Gaussian	200	200	199	199
	Linear	200	200	199	199
IVS-BKPSVR	Gaussian	50	50	50	50
	Linear	50	50	50	50
IVS-EKPSVR	Gaussian	110	104	116	120
	Linear	191	194	196	196

Table 3.2: Two-sample t-test of MAPE for Gaussian vs. Linear Kernels

	Call Bid		Call Ask		Put Bid		Put Ask	
	T Stat.	P Val. (%)	T Stat.	P Val. (%)	T Stat.	P Val. (%)	T Stat.	P Val. (%)
IVS-KPSVR	0.39	69.62	-0.43	66.74	-0.63	52.87	-0.71	47.81
IVS-BKPSVR	0.55	58.45	-0.56	57.46	-1.29	19.64	-1.31	18.93
IVS-EKPSVR	0.43	66.67	-0.82	41.23	-1.08	28.15	-1.09	27.43

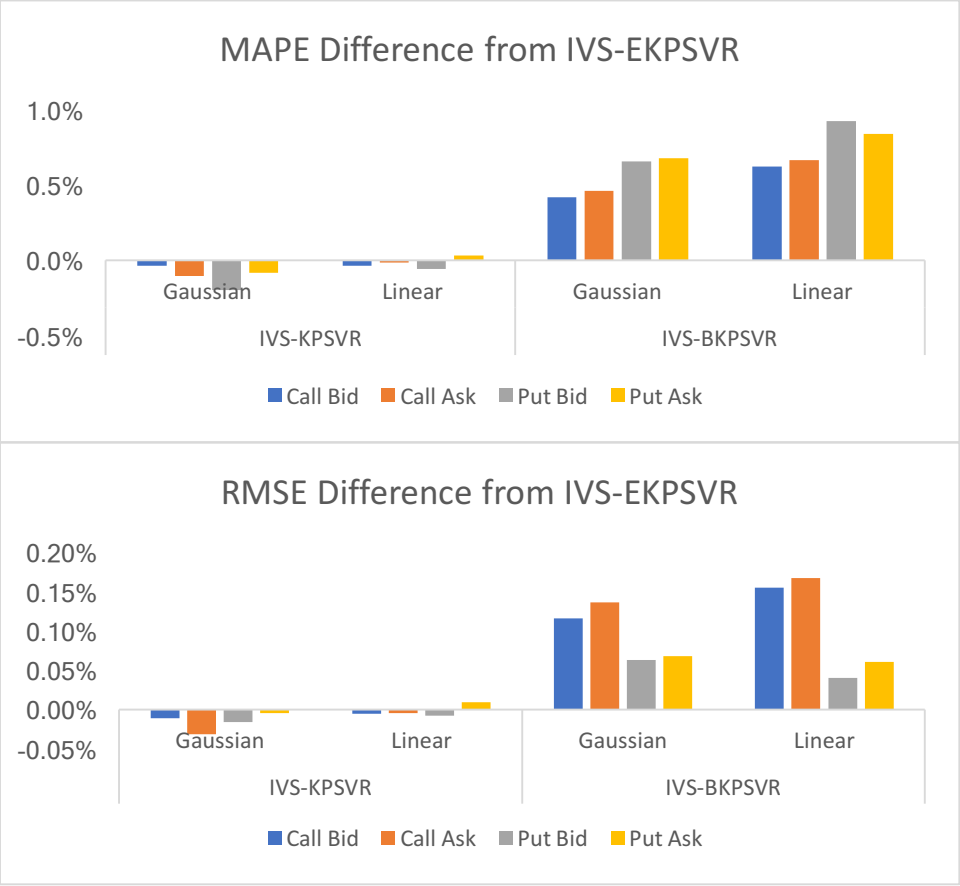


Figure 3.7: Average performance difference from IVS-EKPSVR

Edge points on the strike-maturity grid have less reference points to infer their values, in which case accuracy on the edges is negatively impacted. In all IVS models, the edge points that lie outside of the 20 strike prices in the center part of the grid can increase the regression error by

about 30 ~ 60% (detailed numbers can be found in the Appendix B). In Figure 3.8, we present the performance differences of IVS-KPSVR and IVS-BKPSVR from IVS-EKPSVR if edge strike prices are ruled out. In the center part of the grid, IVS-EKPSVR again outperforms IVS-BKPSVR but shows identical results to IVS-KPSVR with much less support vectors.

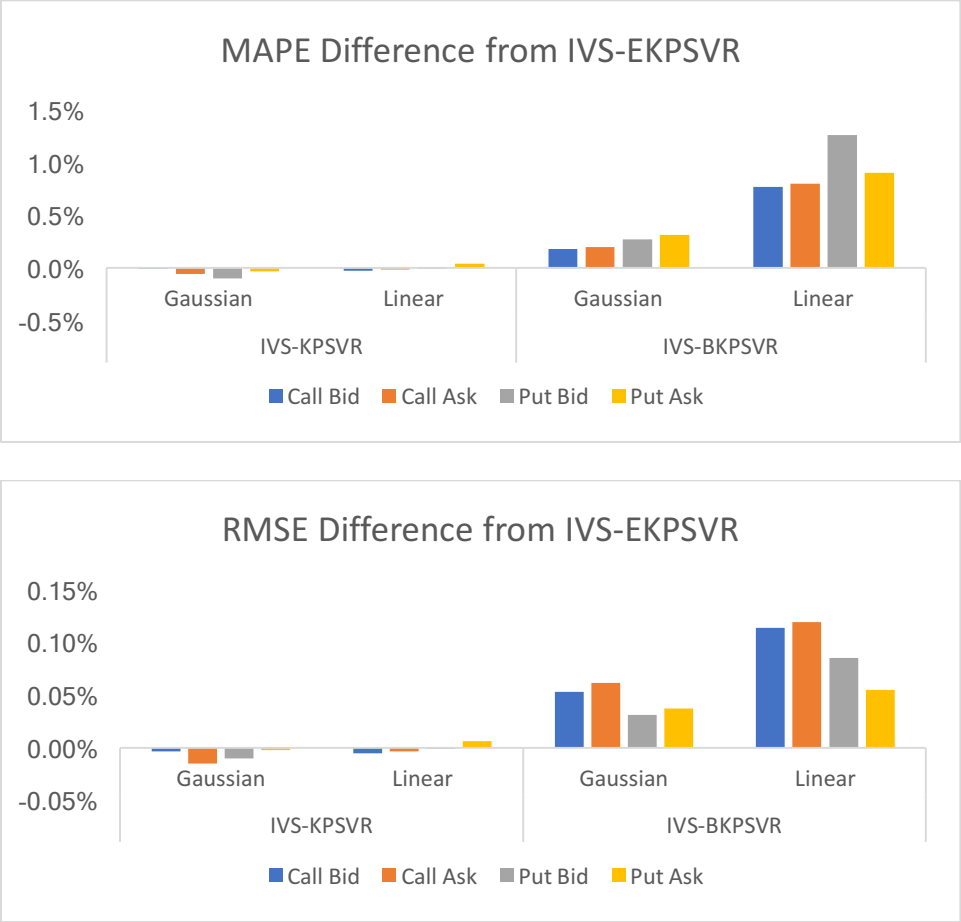


Figure 3.8: Average performance difference from IVS-EKPSVR

Before Pegasos, two stochastic gradient descent based methods were introduced to solve SVM classification problems – Kivinen et al. (2004) and Zhang (2004). Kivinen suggests a learning rate of $\frac{p}{\lambda\sqrt{t}}$ in their algorithm called NORMA, while Zhang simply let it be a constant η,

regardless of iterations (we name it BSGD). To adapt these methods in an online adaptive regression setting, we update Step 2.c in Algorithm 3.4 to $S[s] \leftarrow \left(1 - \frac{p}{\sqrt{t}}\right) S[s]$ (for NORMA) and $S[s] \leftarrow (1 - \eta\lambda) S[s]$ (for BSGD). Step sizes in Step 2.d are changed to $\pm \frac{p}{\lambda\sqrt{t}}$ and η respectively. We name these enhanced algorithms IVS-NORMA[2] and IVS-BSGD. In IVS-BSGD, constant η is set to 0.01 and λ is set to 10 to attain a relatively good result that balances the support vector size and prediction error (tuned using data from 01/27/2014). Since the shrinkage multiplier for $S[s]$ are nonnegative in both cases, the warmup parameter is not needed for these two algorithms. All other parameters remain the same as in IVS-EKPSVR. Table 3.3 shows that, under Gaussian kernel, IVS-EKPSVR uses the least number of support vectors but also achieves the smallest error rates. This further verifies the superiority of the learning rate $\frac{1}{\lambda t}$ and all other enhancements behind IVS-EKPSVR.

Table 3.3: Performance summary of competing algorithms

	Call Bid			Call Ask			Put Bid			Put Ask		
	MAPE (%)	RMSE (%)	SV	MAPE (%)	RMSE (%)	SV	MAPE (%)	RMSE (%)	SV	MAPE (%)	RMSE (%)	SV
IVS-EKPSVR	**12.09**	**2.27**	110	**11.86**	**2.48**	**104**	**14.58**	**1.63**	**116**	**12.45**	**1.66**	**120**
IVS-NORMA	18.52	3.29	170	17.89	3.34	169	24.66	2.72	164	21.13	2.63	160
IVS-BSGD	18.63	3.38	**105**	17.20	3.39	106	27.35	3.01	120	23.10	2.87	125

Hyper parameter tuning directly impacts the number of support vectors and prediction accuracy. Parameter γ in the Gaussian kernel controls how far the influence of a support vector reaches;

[2] Optimal p is determined by $0.5\sqrt{\left(2 + \frac{0.5}{\sqrt{T}}\right)}$, where T is the maximum number of iterations. We let T go to infinity due to the large size of option tick data and hence $p = 0.71$.

constant ρ is related to local fitness; parameter λ regularizes the loss function; warmup coefficient ω defines the magnitude of model updates at each step. In the following analysis, we discuss the sensitivity of IVS-EKPSVR to these parameters using a Gaussian kernel and data from 01/27/2014. All the parameters chosen previously are based on the subsequent grid search process that trades off the model complexity (the number of support vectors) and error rates.

Figure 3.9 demonstrates the changes of average support vector size and MAPE of the four models (Call Bid, Call Ask, Put Bid and Put Ask) with regard to $1/\gamma$ and ρ[3]. A larger $1/\gamma$ exerts a greater influence of support vectors to more distant samples, while a smaller value constrains support vector's influence on nearby data points and hence calls for more support vectors in a model. The MAPE, on the other hand, reaches its lowest value when $1/\gamma = 4$. Local fitness threshold ρ mainly controls the number of support vectors. The larger the threshold ρ is, the more support vectors will be selected. In the MAPE plot, given a fixed $1/\gamma$, parameter ρ yields fairly stable error rates across different ρ values. The spike in the MAPE plot occurs as a resultant of a large $1/\gamma$ and a small ρ, which produces very few support vectors in the model (a sudden drop at the bottom in the left figure), followed by a large MAPE.

[3] For a better viewing angle, the axis directions are different in the two plots.

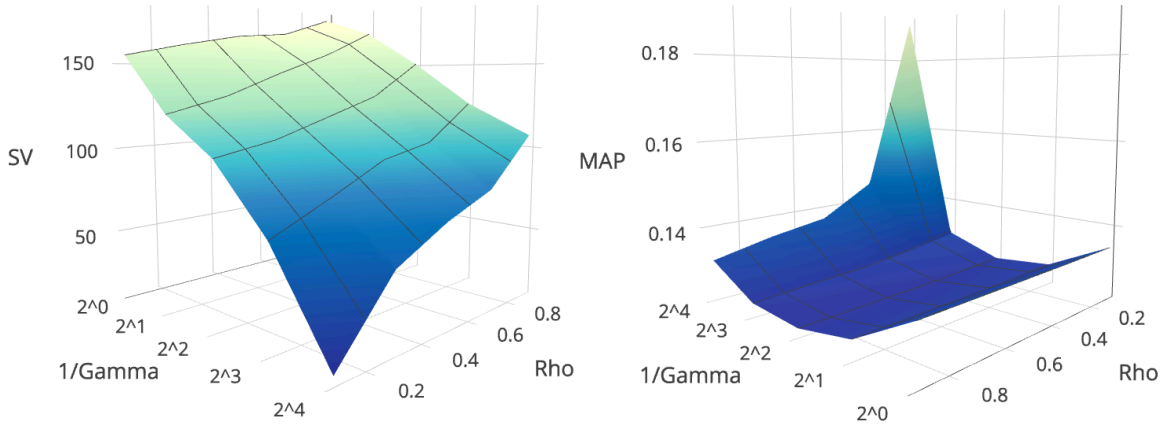


Figure 3.9: Sensitivity of support vector size and MAPE to gamma (γ) and rho (ρ)

Figure 3.10 presents the sensitivity analysis of ω and λ. The warmup factor ω does not influence the MAPE to a great extent, but impacts the support vector sizes: a larger ω leads to a smaller support vector set. A smaller regularization parameter λ puts more emphasis on the regression error by introducing more freedom or support vectors into the model, which further yields a reduced MAPE.

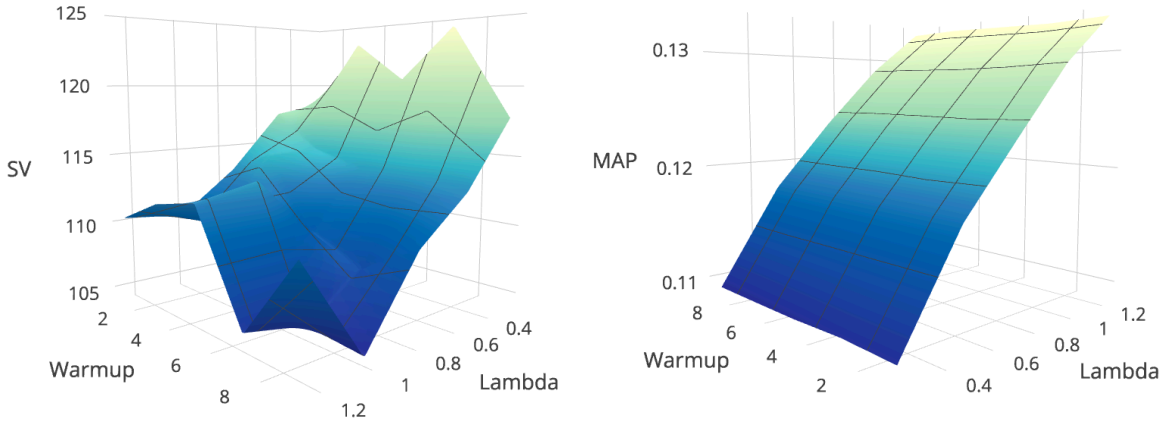


Figure 3.10: Sensitivity of support vector size and MAPE to warmup (ω) and lambda (λ)

In Figure 3.11, we contrast the runtime performance of FPGA against CPU implementations for the four parts of the algorithm as explained in Section 3.4.5. Given the support vector size limit of 200 (the original grid size), FPGA cannot fully utilize its computational power and does not

show much speed improvement due to the communication overhead between CPU and FPGA. For this reason, we enlarge the support vector size 100 times to 20,000, which represents a much finer grid. In the following analysis, it is assumed that all data points on the grid are used as support vectors. During the model training phase, FPGA shows a 16.7 speedup for calculating a local fitness, 7.2 faster for completing the matrix inverse calculations upon a support vector addition and 5.4 speedup for a support vector removal. Prediction phase (Step 2.e in Algorithm 3.4) enjoys the most speedup due to its highly parallel nature. It obtains a 131.8 acceleration for predicting 20,000 samples. For a large-scale online implementation of machine learning algorithms, not only ours, FPGA technology is thus an excellent alternative to reduce latency and to enable agile responses to the unremitting changes in the real world.

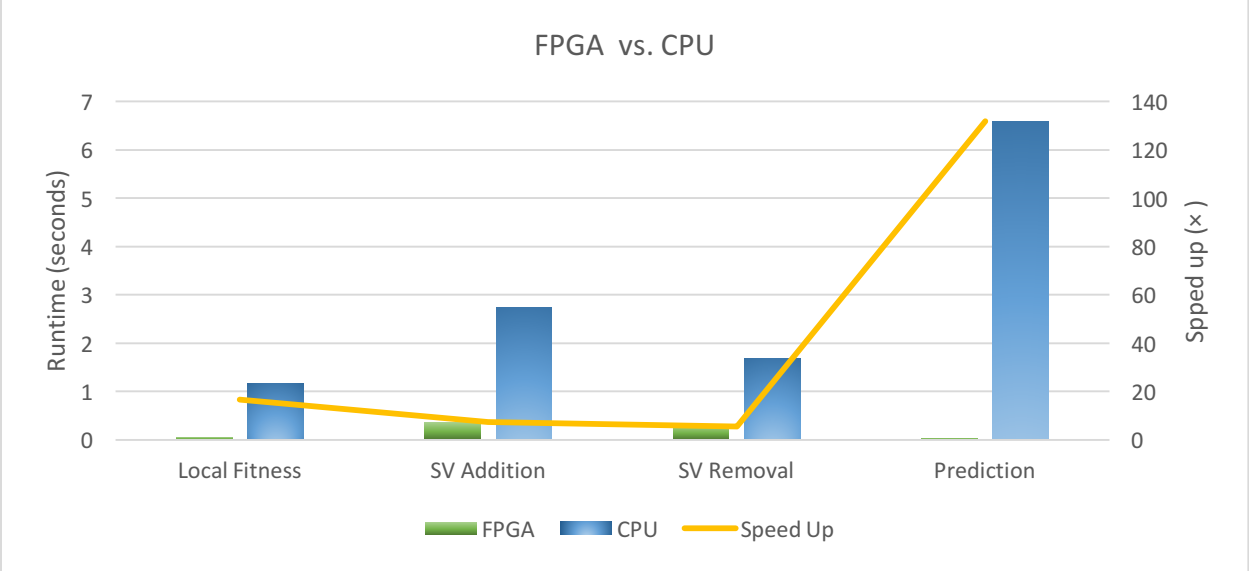


Figure 3.11: FPGA vs. CPU speed comparison

3.6. Conclusion and Future Work

This chapter presents the first implementation of an online adaptive primal SVR algorithm with an application to model the implied volatility surface in the E-mini S&P 500 options market. We introduce feature vector selection and budget maintenance to control the number of support vectors and dynamically update the model once a new pattern or changed pattern emerges, which then evolves into the IVS-EKPSVR algorithm that outperforms either IVS-KPSVR or IVS-BKPSVR. We find that the linear kernel does not work well with FVS in regulating the support vector size, but performs similarly to Gaussian kernel in terms of error rates if an identical number of support vectors are used such as in IVS-KPSVR and IVS-BKPSVR. Due to less reference points, edges of the maturity-strike grid possess larger regression errors that may boost overall MAPE and RMSE by around 30 ~ 60%. Compared with competing methods, IVS-EKPSVR outperforms IVS-NORMA and IVS-BSGD to a great extent, with a MAPE gap up to 12.7% and RMSE gap up to 1.4%. Finally, FPGA hardware has been proved to significantly accelerate the training and prediction phase of our algorithm. Future work can focus on improving the prediction accuracy on the edges of the IVS grid, for example, by introducing more predictor variables from the markets.

References

Amram, M. and Kulatilaka, N. (1999). Real options: Managing strategic investment in an uncertain world. Harvard Business School Press, Boston, Mass.

Ashuri, B., Kashani, H. and Lu, J. (2011). A real options approach to evaluating investment in solar ready buildings, Management and Innovation for a Sustainable Built Environment, Amsterdam, The Netherlands.

Audrino, F., and Colangelo, D. (2010). Semi-parametric forecasts of the implied volatility surface using regression trees. Statistics and Computing, 20(4), 421-434.

Baudat, G., and Anouar, F. (2003). Feature vector selection and projection using kernels. Neurocomputing, 55(1), 21-38.

Belien, J., De Boeck, L., Colpaert, J. and Cooman, G. (2013). The best time to invest in photovoltaic panels in Flanders. Renewable Energy 50, 348-358.

Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. Natural Computing: an international journal, 8(2), 239-287.

Birge, J. R. (2007). Optimization methods in dynamic portfolio management. Handbooks in Operations Research and Management Science, 15, 845-865.

Bollerslev, T., Kretschmer, U., Pigorsch, C., & Tauchen, G. (2009). A discrete-time model for daily S & P500 returns and realized variations: Jumps and leverage effects. Journal of Econometrics, 150(2), 151-166.

Booth, A. (2016). Automated algorithmic trading: machine learning and agent-based modelling in complex adaptive financial markets (Doctoral dissertation, University of Southampton).

Brandt, M. W. (2009). Portfolio choice problems. Handbook of Financial Econometrics, 1, 269-336.

Brown, D. B., and Smith, J. E. (2011). Dynamic portfolio optimization with transaction costs: Heuristics and dual bounds. Management Science, 57(10), 1752-1770.

Cauwenberghs, G., and Poggio, T. (2000, December). Incremental and decremental support vector machine learning. In NIPS (Vol. 13).

Chang, B. R., and Tsai, H. F. (2008). Forecast approach using neural network adaptation to support vector regression grey model and generalized auto-regressive conditional heteroscedasticity. Expert systems with applications, 34(2), 925-934.

Chen, H. L., Yang, B., Wang, G., Liu, J., Xu, X., Wang, S. J., and Liu, D. Y. (2011). A novel bankruptcy prediction model based on an adaptive fuzzy k-nearest neighbor method. Knowledge-Based Systems, 24(8), 1348-1359.

Chen, S., Härdle, W. K., and Jeong, K. (2010). Forecasting volatility with support vector machine-based GARCH model. Journal of Forecasting, 29(4), 406-433.

Constantinides, G. M., Jackwerth, J. C., and Savov, A. (2013). The puzzle of index option returns. Review of Asset Pricing Studies.

Crammer, K., Kandola, J. S., and Singer, Y. (2003, December). Online Classification on a Budget. In NIPS (Vol. 2, p. 5).

Denault, M., and Simonato, J. G. (2017). Dynamic portfolio choices by simulation-and-regression: Revisiting the issue of value function vs portfolio weight recursions. Computers and Operations Research, 79, 174-189.

Dixit, A.K. and Pindyck, R.S. (1994). Investment under uncertainty. Princeton University Press, Princeton, N.J.

Driessen, J., and Maenhout, P. (2013). The world price of jump and volatility risk. Journal of Banking and Finance, 37(2), 518-536.

Dumas, B., Fleming, J., and Whaley, R. E. (1998). Implied volatility functions: Empirical tests. The Journal of Finance, 53(6), 2059-2106.

Energy Information Administration (2012). Annual Energy Review 2011, www.eia.gov/totalenergy/data/annual/pdf/aer.pdf.

Eraker, B. (2013). The performance of model based option trading strategies. Review of Derivatives Research, 16(1), 1-23.

Faias, J., and Santa-Clara, P. (2011). Optimal option portfolio strategies. In AFA 2011 Denver Meetings Paper. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1569380

Fengler, M. R., Härdle, W. K., and Mammen, E. (2007). A semiparametric factor model for implied volatility surface dynamics. Journal of Financial Econometrics, 5(2), 189-218.

Fernandes, B., Cunha, J. and Ferreira, P. (2011). The use of real options approach in energy sector investments. Renewable Sustainable Energy Review 15, 4491-4497.

Gouriéroux, C. and Valéry, P. (2004). Estimation of a Jacobi process. Preprint, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.196andrep=rep1andtype=pdf.

Gu, B., Sheng, V. S., Wang, Z., Ho, D., Osman, S., and Li, S. (2015). Incremental learning for ν-support vector regression. Neural Networks, 67, 140-150.

Hahn, T. (2013). Option pricing using artificial neural networks: an Australian perspective.

Hansen, P. R., & Lunde, A. (2005). A realized variance for the whole day based on intermittent high-frequency data. Journal of Financial Econometrics, 3(4), 525-554.

Haugh, M. B., and Kogan, L. (2007). Duality theory and approximate dynamic programming for pricing American options and portfolio optimization. Handbooks in operations research and management science, 15, 925-948.

Hoff, T.E., Margolis, R. and Herig, C. (2003). A simple method for consumers to address uncertainty when purchasing photovoltaics, http://www.cleanpower.com/Research.

Homescu, C. (2011). Implied volatility surface: Construction methodologies and characteristics.

Hu, G., and Jacobs, K. (2016). Volatility and Expected Option Returns.

Ilhan, A., Jonsson, M., and Sircar, R. (2004). Portfolio optimization with derivatives and indifference pricing. Indifference Pricing (ed. Carmona), 181-210.

Jones, C. S. (2006). A nonlinear factor analysis of S&P 500 index option returns. The Journal of Finance, 61(5), 2325-2363.

Kivinen, J., Smola, A. J., and Williamson, R. C. (2004). Online learning with kernels. IEEE transactions on signal processing, 52(8), 2165-2176.

Kyrkou, C., Theocharides, T., and Bouganis, C. S. (2013, July). An embedded hardware-efficient architecture for real-time cascade support vector machine classification. In Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on (pp. 129-136). IEEE.

Lee, S., Lee, J., Shim, D., and Jeon, M. (2007, September). Binary particle swarm optimization for black-scholes option pricing. In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (pp. 85-92). Springer Berlin Heidelberg.

Li, J., Li, G., Sun, D., and Lee, C. F. (2012). Evolution strategy based adaptive L q penalty support vector machines with Gauss kernel for credit risk analysis. Applied Soft Computing, 12(8), 2675-2682.

Liu, J., and Pan, J. (2003). Dynamic derivative strategies. Journal of Financial Economics, 69(3), 401-430.

Liu, J., and Zio, E. (2016). An adaptive online learning approach for Support Vector Regression: Online-SVR-FID. Mechanical Systems and Signal Processing, 76, 796-809.

Longstaff, F. A., and Schwartz, E. S. (2001). Valuing American options by simulation: a simple least-squares approach. Review of Financial studies, 14(1), 113-147.

Ma, J., Theiler, J., and Perkins, S. (2003). Accurate online support vector regression. Neural computation, 15(11), 2683-2703.

Malliaris, M., and Salchenberger, L. (1996). Using neural networks to forecast the S&P 100 implied volatility. Neurocomputing, 10(2), 183-195.

Markowitz, H. (1952). Portfolio selection. The Journal of Finance, 7(1), 77-91.

Martinez-Cesena, E.A., Azzopardi, B. and Mutale, J. (2013). Assessment of domestic photovoltaic systems based on real options theory. Progress in Photovoltaics 21, 250-262.

Martinez-Cesena, E.A. and Mutale, J. (2011). Application of an advanced real options approach for renewable energy generation projects planning. Renewable Sustainable Energy Review 15, 2087-2094.

Merton, R. C. (1969). Lifetime portfolio selection under uncertainty: The continuous-time case. The Review of Economics and Statistics, 51(3), 247-257.

Merton, R. C. (1971). Optimum consumption and portfolio rules in a continuous-time model. Journal of Economic Theory, 3(4), 373-413.

Merton, R. C. (1975). Theory of finance from the perspective of continuous time. Journal of Financial and Quantitative Analysis, 10(04), 659-674.

Merton, R. C. (1990). Continuous-time finance. Cambridge, Mass., B. Blackwell.

Mun, J. (2002). Real options analysis: Tools and techniques for valuing strategic investments and decisions. John Wiley and Sons, Hoboken, N.J.

Munoz, J.I., Contreras, J., Caamano, J. and Correia, P.F. (2009). Risk assessment of wind power generation project investments based on real options. 2009 IEEE Bucharest Powertech, Vols 1-5, 2346-2353.

National Renewable Energy Laboratory (2009). Power purchase agreement checklist for state and local governments, www.nrel.gov/docs/fy10osti/46668.pdf.

National Renewable Energy Laboratory (2011). Solar renewable energy certificate (SREC) markets: Status and trends, http://apps3.eere.energy.gov/greenpower/pdfs/52868.pdf.

Papadonikolakis, M., and Bouganis, C. S. (2012). Novel cascade FPGA accelerator for support vector machines classification. IEEE Transactions on Neural Networks and Learning Systems, 23(7), 1040-1052.

Poon, S. H., and Granger, C. W. (2003). Forecasting volatility in financial markets: A review. Journal of economic literature, 41(2), 478-539.

Powell, W. B. (2011). Approximate Dynamic Programming: Solving the curses of dimensionality. John Wiley and Sons.

Powell, W., Ruszczyński, A., and Topaloglu, H. (2004). Learning algorithms for separable approximations of discrete stochastic optimization problems. Mathematics of Operations Research, 29(4), 814-836.

Rabieah, M. B., and Bouganis, C. S. (2015, September). FPGA based nonlinear Support Vector Machine training using an ensemble learning. In Field Programmable Logic and Applications (FPL), 2015 25th International Conference on (pp. 1-4). IEEE.

Rockafellar, R. T., and Wets, R. J. B. (1991). Scenarios and policy aggregation in optimization under uncertainty. Mathematics of operations research, 16(1), 119-147.

Rodrigues, A. and Armada, M.J.R. (2006). The valuation of real options with the least squares Monte Carlo simulation method, http://ssrn.com/abstract=887953 or http://dx.doi.org/10.2139/ssrn.887953.

Ruiz-Llata, M., Guarnizo, G., and Yébenes-Calvino, M. (2010, July). FPGA implementation of a support vector machine for classification and regression. In Neural Networks (IJCNN), The 2010 International Joint Conference on (pp. 1-5). IEEE.

Sarkis, J. and Tamarkin, M. (2008). Real options analysis for renewable energy technologies in a GHG emissions trading environment, Emissions Trading. Springer, New York, pp. 103-119.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. Neural computation, 12(5), 1207-1245.

SEIA/GTM Research (2013). U.S. Solar Market Insight 2012 Year in Review.

Sévi, B. (2014). Forecasting the volatility of crude oil futures using intraday data. European Journal of Operational Research, 235(3), 643-659.

Shalev-Shwartz, S., and Srebro, N. (2008, July). SVM optimization: inverse dependence on training set size. In Proceedings of the 25th international conference on Machine learning (pp. 928-935). ACM.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007, June). Pegasos: Primal estimated sub-gradient solver for svm. In Proceedings of the 24th international conference on Machine learning (pp. 807-814). ACM.

Steinwart, I. (2003). Sparseness of support vector machines. Journal of Machine Learning Research, 4(Nov), 1071-1105.

Stentoft, L. (2014). Value function approximation or stopping time approximation: a comparison of two recent numerical methods for American option pricing using simulation and regression.

Sun, J., Li, H., and Adeli, H. (2013). Concept drift-oriented adaptive and dynamic support vector machine ensemble with time window in corporate financial risk prediction. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 43(4), 801-813.

Tang, A., Chiara, N. and Taylor, J.E. (2012). Financing renewable energy infrastructure: Formulation, pricing and impact of a carbon revenue bond. Energy Policy 45, 691-703.

Trigeorgis, L. (1996). Real options: Managerial flexibility and strategy in resource allocation. MIT Press, Cambridge, Mass.

Tsitsiklis, J. N., and Van Roy, B. (2001). Regression methods for pricing complex American-style options. IEEE Transactions on Neural Networks, 12(4), 694-703.

U.S. Department of Energy (2009). President Obama sets a target for cutting U.S. greenhouse gas emissions, http://apps1.eere.energy.gov/news/news_detail.cfm/news_id=15650.

United Nations Framework Convention on Climate Change (2008). Kyoto Protocol Reference Manual, http://unfccc.int/resource/docs/publications/08_unfccc_kp_ref_manual.pdf.

Vapnik, V. N., and Vapnik, V. (1998). Statistical learning theory (Vol. 1). New York: Wiley.

Venetsanos, K., Angelopoulou, P. and Tsoutsos, T. (2002). Renewable energy sources project appraisal under uncertainty: The case of wind energy exploitation within a changing energy market environment. Energy Policy 30, 293-307.

Wang, C. P., Lin, S. H., Huang, H. H., and Wu, P. C. (2012). Using neural network for forecasting TXO price under different volatility models. Expert Systems with Applications, 39(5), 5025-5032.

Wang, P. (2011). Pricing currency options with support vector regression and stochastic volatility model with jumps. Expert Systems with Applications, 38(1), 1-7.

Wang, Z., Crammer, K., and Vucetic, S. (2012). Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. Journal of Machine Learning Research, 13(Oct), 3103-3131.

Zhang, T. (2004, July). Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning (p. 116). ACM.

APPENDIX A

Appendix for Chapter 2

A.1. Model for European Option Portfolio

A European option portfolio consists of a number of European options that depend on one or more underlying assets. It also contains a risk-free account that stores part of the total wealth while the remainder is allocated to the options. In general, we want to maximize the utility of the portfolio terminal wealth.

We assume that the time horizon is finite and investors can only trade options at discrete times t=1,…T. The maturity time of each option is one time period. Suppose the number of options is N, and they are based on the underlying asset whose price $A_t = (A_{1,t}, A_{2,t}, ..., A_{N,t})$ evolves based on a stochastic process. The risk-free asset follows the return process $r_{0,t}$. The portfolio is self-financing.

The price of option i is $p_i = \left(p_{i,1}, p_{i,2}, ..., p_{i,T}\right)$, and the strike price is $K_i = \left(K_{i,1}, K_{i,2}, ..., K_{i,T}\right)$. Then the return of option i during time t is simply

$$r_{i,t} = \begin{cases} \left(\frac{A_{i,t} - K_{i,t}}{p_{i,t}}\right)^+ & \text{if call option;} \\ \left(\frac{K_{i,t} - A_{i,t}}{p_{i,t}}\right)^+ & \text{if put option,} \end{cases} \tag{A.1}$$

where $(\bullet)^+ = \max(\bullet, 0)$.

The portfolio strategy over the entire horizon is represented by

$$w = (w_1, w_2, ..., w_T) \in X = \left\{ w \in \mathbb{R}_+^{N \times T} : \sum_{i=0}^{N} w_{i,t} = 1, \text{for every } t \right\}, \text{ where } w_t = \left(w_{0,t}, w_{1,t}, ..., w_{N,t}\right)^T.$$

Weight $w_{0,t}$ is the weight of the risk-free asset during time t and $w_{i,t}$, for every $i \in \{1,..,N\}$ is the weight of option i during time t in the portfolio. Particularly, we do not allow short selling and borrowing.

The total wealth of the portfolio thus follows $W_{t+1}^{total} = W_t^{total} r_t^{portfolio}$, where $r_t^{portfolio} = \sum_{i=0}^{N} r_{i,t} w_{i,t}$ is the rate of return of the portfolio during time period t.

The objective function then reads

$$\max_{w \in X} E\left[\overline{U}\left(W_1^{total}, W_2^{total}, ..., W_T^{total}\right)\right], \text{ where } W_t^{total} = W_t^{total}\left(w_1, ..., w_{t-1}\right), \tag{A.2}$$

and $\overline{U}(\bullet)$ is the utility function. Clearly, the terminal wealth is a function of the weights throughout the horizon. We assume an additive utility function in the following form:

$$\overline{U}\left(W_1^{total}, W_2^{total}, ..., W_T^{total}\right) = \sum_{t=1}^{T} U\left(W_t^{total}\right).$$

Without loss of generality, we suppose there are N European options based on a single underlying asset in the following analysis. In the beginning of each time period, only one decision is made: the weights of wealth allocated to each option and the risk-free account.

A.1.1. Optimal Option Portfolio Strategy (OOPS, Faias and Santa-Clara, 2011)

As we have introduced, OOPS introduces a simple but intuitive method that solves a portfolio problem explicitly for European options. The resulting solution is not optimal but we use this term in order to be consistent with the terms used by Faias and Santa-Clara. We next summarize their key ideas.

By simulating N_1 series of underlying asset values and substituting them into (A.1), they first get N_1 series of option returns $r_{i,t}^n$ for each option *i* during each time period *t*, and sample $n \in \{1,...,N_1\}$. Then from time 1 to the last period *T*, they perform the following unconstrained optimization to obtain optimal weights for the next time period

$$\max_{w_t} E\left[U\left(W_t^{total} r_t^{portfolio}\right)\right] \approx \max_{w_t}\left[\frac{1}{N_1}\sum_{n=1}^{N_1} U\left(W_t^{total} r_t^{portfolio,n}\right)\right],$$

where $r_t^{portfolio,n} = \sum_{i=0}^{N} r_{i,t}^n w_{i,t}$.

After the optimization, they then update the portfolio wealth and step forward into the next time period.

Note that they do not have any constraints in the optimization, which means the weights can be negative. If the weight of an option is negative, it corresponds to short selling. However, if the weight of the risk-free account turns negative, it requires net borrowing money. We do not allow such features in our model, even though then can be easily incorporated.

Another point is that their method is myopic. By treating each time period independently, they only optimize the utility for next time period instead of the entire time horizon. Essentially, their method is a repeating process that solves multiple single-period models.

A.1.2. ADP Algorithm

We now propose an ADP method that treats problem (A.2) as a multi-period model. This method is an extension to the portfolio optimization method in Powell (2011).

We first define relevant variables. Let the wealth of risk free account and each option during every time period be denoted by $W_{i,t}$ for every $i \in \{0,1,...,N\}, t \in \{1,...,T\}$. The *decision variable*, action, is the wealth transfer $x_{m,n,t}$ between assets *m* and *n* in time period *t*. We denote $x_t = \left(x_{m,n,t}\right)_{m,n}$.

Note that instead of the weights in the general model and OOPS, we define wealth transfers to be the decision variables. The weights can be implied from $W_{i,t}$ as $w_{i,t} = \frac{W_{i,t}}{W_t^{total}}$.

We also define the following *post decision state variables*

$$W_{i,t}^x = W_{i,t} + \sum_{m=0}^{N} x_{m,i,t} .$$

After a decision is made during some time period *t*, exogenous information $\omega_{t+1} = \omega_{t+1}\left(A_{i,t}, K_{i,t}, p_{i,t}\right)$ realizes. With $A_{i,t}$, $K_{i,t}$ and $p_{i,t}$, we calculate the return $r_{i,t}$ of option *i* by (A.1). Then we update the state variable using the following *transition function*

$$W_{i,t+1} = r_{i,t} W_{i,t}^{x}.$$

The objective is to maximize the total utility over time, i.e.

$$\max_{x_1, x_2, \ldots, x_t} E\left[\sum_{t=0}^{T} U\left(\sum_{i=0}^{N} W_{i,t+1}\right)\right].$$

Based on the objective, the *optimality equation* reads

$$V_{t-1}^{x}\left(W_{0,t-1}^{x}, \ldots, W_{N,t-1}^{x}\right) = E\left\{\max_{x_t}\left[U\left(W_t^{total}\right) + V_t^{x}\left(W_{0,t}^{x}, \ldots, W_{N,t}^{x}\right)\right] \middle| W_{0,t-1}^{x}, \ldots, W_{N,t-1}^{x}\right\}.$$

Post decision state variables eliminate the need to find the one-step transition matrix by pulling the expectation operator out of the max operator, which makes the algorithm computationally tractable.

To solve this ADP, we apply the piecewise linear approximation method to estimate the value function. We let the estimation

$$\overline{V}_{i,t}^{n}\left(W_{i,t}^{x,n}\right) = \sum_{m=1}^{\left\lfloor W_{i,t}^{x,n} \right\rfloor} \overline{v}_{i,t}^{n-1}(m-1) + \left(W_{i,t}^{x,n} - \left\lfloor W_{i,t}^{x,n} \right\rfloor\right) \overline{v}_{i,t}^{n-1}\left(\left\lfloor W_{i,t}^{x,n} \right\rfloor\right)$$

approximate the value function of $W_{i,t}^{x,n}$ in the n^{th} iteration. Notation $\lfloor \bullet \rfloor$ is the floor operator. Here $\overline{v}_{i,t}^{n-1}$ is the slope of the piecewise linear function of asset *i* during time *t* in the n^{th} iteration.

Algorithm A.1 lists the entire algorithm for computing the weights. We add up the value functions of each asset to get the value function approximation of the portfolio wealth (Step 2.1). Across iterations, we want to improve the performance of the piecewise linear approximation in estimating the value functions, which is done by updating slopes $\bar{v}_{i,t}^{n-1}$ based on dual variables in the mathematical program. The mathematical program also gives the optimal decision of wealth transfer (Step 2.2). Again, in every iteration *n* we update the slope values and use them in the next iteration (Step 2.4). We follow the trick provided by Powell (2011) to calculate the slopes (Step 2.3). It is straightforward calculus to verify Step 2.3. To make the slopes form a concave function, we apply the separable, projective approximation routine (SPAR) algorithm provided in Powell et al. (2004). Concavity implies that maximization in Step 2.2 is a linear program.

Thus $\tilde{v}_{i,t}^{n}$ is the derivative of $\frac{\delta \tilde{V}_t^n}{\delta W_{i,t-1}^x}$.

Algorithm A.1

Initialize $\bar{v}_t^0$, W_1^0, n=1.

Step 1 Choose a sample path of the underlying asset price A_t^n and determine the option and strike prices. Based on these data, calculate returns $r_{i,t}^n$.

Step 2 For t=1,…,T

1. Let

$$\bar{V}_t^{n-1}\left(W_{i,t}^{x,n}\right)=\sum_{i=0}^{N}\bar{V}_{i,t}^{n-1}\left(W_{i,t}^{x,n}\right).$$

2. Solve

$$\widetilde{V}_t^n = \max_{x_t}\left(U(W_t^{total,n}) + \overline{V}_t^{n-1}\left(W_{i,t}^{x,n}\right)\right) = U(W_t^{total,n}) + \max_{x_t}\left(\overline{V}_t^{n-1}\left(W_{i,t}^{x,n}\right)\right)$$

$$s.t. \quad \sum_{j=0}^{N} x_{i,j,t} = W_{i,t}^n, \quad \text{for every } i; \qquad \text{(A.3)}$$

$$x_{i,j,t} \geq 0, \qquad \text{for every } i, j.$$

Let x_t^n be an optimal solution to the maximization problem, and $\hat{v}_{i,t}^n$ be the dual variable of (A.3).

3. Compute

$$\tilde{v}_{i,t}^n = \frac{\delta \widetilde{V}_t^n}{\delta W_{i,t-1}^x} = \left(\frac{\delta U\left(W_t^{total}\right)}{\delta W_{i,t}}\Big|_{W_t^{total}=W_t^{total,n}} + \hat{v}_{i,t}^n\right) r_{i,t}^n .$$

4. Update $\bar{v}_{t-1}^n$ using

$$\bar{v}_{i,t-1}^n(m) = \begin{cases} (1-\alpha_{n-1})\bar{v}_{i,t-1}^{n-1}(m) + \alpha_{n-1}\tilde{v}_{i,t}^n, & \text{if } m = \lfloor W_{i,t-1}^{x,n} \rfloor. \\ \bar{v}_{i,t-1}^n(m), & \text{otherwise.} \end{cases}$$

5. Find the post-decision state $W_{i,t}^{x,n}$, the next pre-decision state $W_{i,t+1}^n$ based on an optimal solution in Step 2.2 and update portfolio wealth $W_{t+1}^{total,n} = \sum_{i=0}^{N} W_{i,t+1}^n$

Step 3 $n = n + 1$. If $n \leq N_1$, go back to Step 1.

Step 4 Return value functions $\left(\widetilde{V}_t^{N_1}\right)_{t=1}^T$.

A.2. Results for European Option Portfolio

We assume there are four options in the portfolio: an ATM put option, a 5% OTM put option, an ATM call option and a 5% OTM call option, which have higher liquidity. They are based on a single underlying asset – S&P500 index. Besides, there is also a risk-free account. We perform a 30-month test for both algorithms, where one month is a time period. In the beginning, we fit the parameters by using historical data using the generalized extreme value (GEV) distribution and then sample the initial index returns 500 times. As we move forward in time from a month to the next one, we re-fit the parameters of GEV based on the historical values available up to the current time. For example, we use 15-years of historical S&P500 data to train the initial GEV parameters. Options expire at the end of the 1st month and we enter a new time period. Then we add the known index value during the 1st month to fit a new set of GEV parameters. The new parameters are used to sample asset returns for the 2nd month. The 30 sets of GEV parameters are not listed here. The simulated index (asset) values can be obtained by $A_t^n = r_t^n A_{t-1}^n$. Without the data of historical option prices, the option prices are computed based on Black Scholes formula by inputting the historical index volatilities. The risk-free rate is defined to be the one-month London Interbank Offered Rate (LIBOR). We use the same sample paths and data to evaluate both the ADP and OOPS algorithms. For simplicity, the same set of samples is used for optimization and evaluation. It takes 75 seconds to run ADP and 6 seconds to run OOPS.

Figure A.1 shows the comparison of portfolio returns during each time period. Mean values of portfolio returns are used. We find that our ADP method outperforms the OOPS method. Out of the 30 months, the ADP algorithm surpasses OOPS in 28 months. The across-time mean

portfolio return of OOPS is -6.18%, while ADP returns 72.51%. From Figure A.2 we see that the ADP algorithm also outperforms OOPS in terms of the cumulative wealth. The mean cumulative return for OOPS is -92.63%, while 66.75% for ADP.

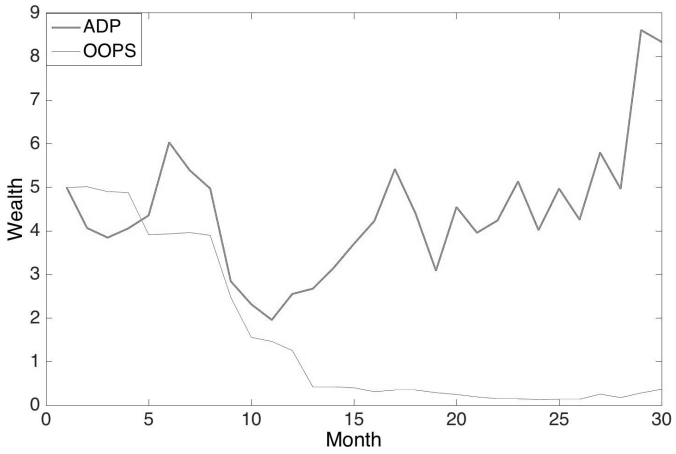


Figure A.1: Comparison of portfolio returns

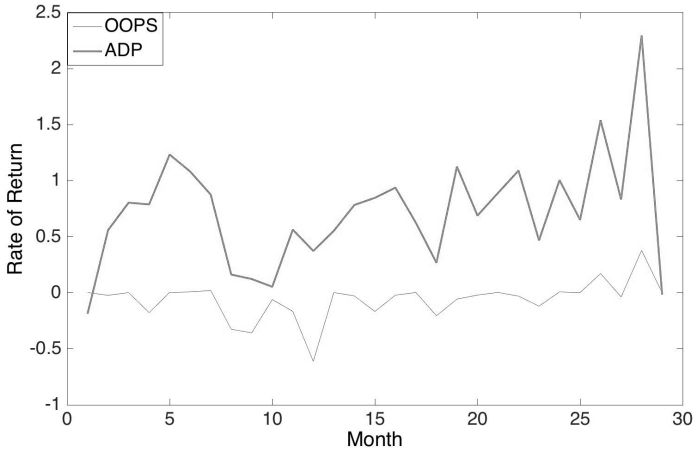


Figure A.2: Portfolio wealth overtime

Figure A.3 shows the convergence of the ADP algorithm. We pick the mean rate of return in month 4 and plot it against the increasing number of iterations. We find the mean values to

stabilize after 300 iterations. This is also the case for the cumulative portfolio return, see Figure A.4.

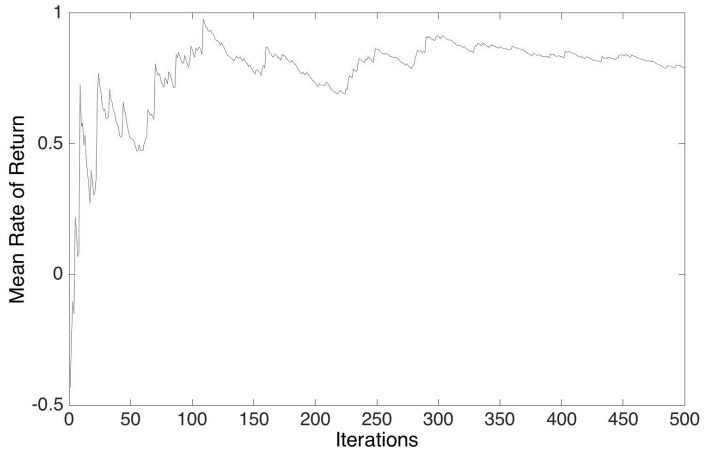


Figure A.3: Mean rate of return during month 4

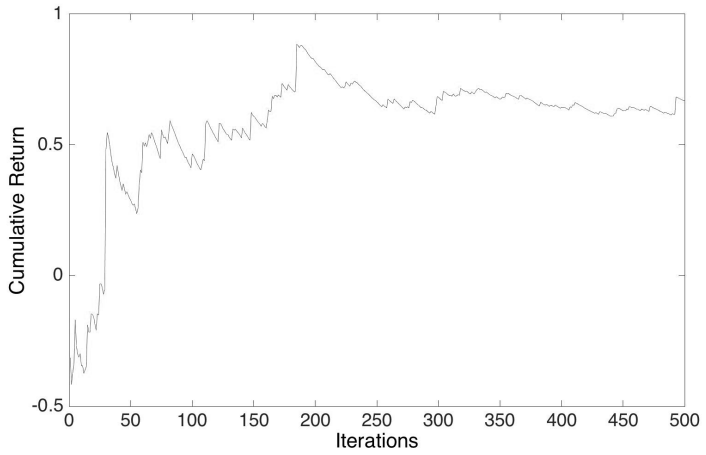


Figure A.4: Cumulative portfolio return

A.3. Benchmark Algorithms for American Option Portfolio

Here we present the modified algorithm to find exercise times of a portfolio of American options assessed by a utility function. The algorithm is a modification of LSMC from Longstaff and Schwartz (2001) where a single option with no utility is dealt with.

Algorithm A.2 finds a set of linear regression parameters for each option in every time period. By comparing the holding value against the exercise value, Algorithm A.2 determines what options we should exercise in each time period. After Algorithm A.2 computes the regression coefficients based on a set of sample paths, Algorithm A.3 resamples the paths and computes exercise times by using the regression coefficients from Algorithm A.2.

Note that Algorithm A.2 starts from the last time period *T*, while Algorithm A.3 goes forward from the first time period. The weights from IPH are used only in Algorithm A.3.

Algorithm A.2

---

a. Sample $A_t^n, r_{i,t}^n$ for $n = 1,...,N_1$.

b. Calculate the exercise value function $V_{i,t}^{n,e} = U\left(r_{i,t}^n\right)$ for all *i*, *n*, *t*.

c. Set $V_{i,T}^{n,h} = V_{i,T}^{n,e}$, $n_{e^*} = T$ for all *n*, *i* (superscript *e* stands for "exercise"; *h* stands for "hold"; e^* stands for "optimal exercise time").

For *t*=*T*-1 to 1

1. Apply the least square regression based on N_1 realizations and find the values for $c,\ b_1,\ b_2,$ by solving

$$\xi^m + c_{i,t} + b_{i,1,t}A_t^m + b_{i,2,t}\left(A_t^m\right)^2 = V_{i,m_{e^*}^*}^{m,e} \text{ for all } i,\ m \in \{1,...,N_1\}.$$

For *n*=1 to N_1

2. Set $V_{i,t}^{n,h} = c_{i,t} + b_{i,1,t}A_t^n + b_{i,2,t}\left(A_t^n\right)^2$ for all i.

3. Compare $V_{i,t}^{n,h}$ and $V_{i,t}^{n,e}$:

$$\text{If } V_{i,t}^{n,h} \le V_{i,t}^{n,e}, \text{ update } n_{e^*} = t,\ V_{i,n_{e^*}}^{n,e} = V_{i,t}^{n,e} \text{ for all } i.$$

End

End

4. Return $c,\ b_1,\ b_2$.

Algorithm A.3

---

a. Sample $A_t^n, r_{i,t}^n$ for $n = 1,...,N_1$.

b. Apply Algorithm 2 to obtain vectors

$$c_t = \left(c_{i,t}\right)_{i=1}^N, b_{1,t} = \left(b_{i,1,t}\right)_{i=1}^N, b_{2,t} = \left(b_{i,2,t}\right)_{i=1}^N.$$

For *n*=1,…,N_1

For *t*=0,1,…,*T*

1. Set $I = \{1,...,N\}$.

2. Solve

$$\max_{y_t^n \in \{0,1\}} \sum_{i \in I} w_i \left[\left(1 - y_{i,t}^n\right) V_{i,t}^{n,h}\left(A_t^n\right) + y_{i,t}^n V_{i,t}^{n,e}\left(A_t^n\right) \right],$$

where

$$V_{i,t}^{n,h}\left(A_t^n\right) = c_{i,t} + b_{i,1,t} A_t^n + b_{i,2,t}\left(A_t^n\right)^2, \quad V_{i,t}^{n,e}\left(A_t^n\right) = U\left(r_{i,t}^n\right).$$

Let $y_t^{n,*}$ be an optimal decision of the optimization problem.

3. Set

$$I = I \setminus \left\{ i : y_{i,t}^{n,*} = 1 \right\}, \quad t_i^{n,*} = t \text{ for } i \text{ with } y_{i,t}^{n,*} = 1.$$

End

End

4. Return exercise times $t_i^{n,*}$ for each sample path *n* and option *i*.

A.4. Summary Statistics of Empirical Experiments

In Tables A.1 – A.3, notation E(R) represents the annualized expected return, SR the annualized Sharpe Ratio, and CE the annualized certainty equivalent of return. The Skew and Kurt columns measure the skewness and excess kurtosis of annualized portfolio returns. The Delta column reports the portfolio delta given the weights from IPH. The Delta column under IPH-PERFECT is omitted because they share the same delta as IPH-QL due to identical weights.

Table A.1: Performance summary statistics (1-year maturity, GEV distribution, non-hedge)

CRRA	PS	IPH-QL						EW-QL						IPH-PERFECT				
		E(R)	SR	CE	Skew	Kurt	Delta	E(R)	SR	CE	Skew	Kurt	Delta	E(R)	SR	CE	Skew	Kurt
-0.5	1	-5.5%	-0.06	53.8%	1.00	0.32	0.34	-21.1%	-0.40	51.7%	-0.15	-1.38	0.08	45.5%	0.37	95.0%	1.72	3.21
	2	3.4%	0.06	54.5%	0.55	-0.60	0.30	-14.7%	-0.42	49.3%	-0.30	-1.24	0.09	54.1%	0.79	87.2%	0.98	0.62
	3	-4.8%	-0.08	51.3%	0.71	-0.53	0.33	-14.1%	-0.31	54.0%	0.50	0.44	0.14	39.0%	0.56	80.8%	1.34	1.75
	4	1.4%	0.02	55.1%	0.33	-1.43	0.30	-19.3%	-0.44	49.9%	-0.21	-1.28	0.03	63.4%	0.51	104.7%	1.21	1.38
0	1	-7.5%	-0.10	-7.5%	0.65	-0.75	0.32	-21.2%	-0.37	-21.2%	0.61	0.91	0.08	45.9%	0.41	45.9%	1.61	2.64
	2	2.0%	0.03	2.0%	0.36	-1.14	0.32	-14.3%	-0.36	-14.3%	0.08	-0.99	0.09	52.5%	0.79	52.5%	0.97	0.63
	3	-4.9%	-0.08	-4.9%	0.61	-0.59	0.35	-17.0%	-0.34	-17.0%	0.97	2.47	0.14	37.9%	0.58	37.9%	1.27	1.41
	4	2.0%	0.02	2.0%	0.56	-0.96	0.30	-14.8%	-0.30	-14.8%	0.02	-1.14	0.03	56.7%	0.50	56.7%	1.33	1.57
1	1	-5.0%	-0.06	-3.1%	1.13	1.02	0.32	-22.1%	-0.42	-27.1%	-0.03	-1.21	0.08	42.9%	0.41	26.4%	1.54	2.35
	2	0.5%	0.01	-7.8%	0.23	-1.27	0.30	-15.3%	-0.40	-24.3%	0.08	-0.73	0.09	51.0%	0.83	28.0%	0.82	0.17
	3	-7.4%	-0.14	-12.7%	0.55	-0.70	0.32	-18.2%	-0.48	-25.0%	-0.39	-0.87	0.14	37.0%	0.62	15.2%	1.14	1.24
	4	3.1%	0.04	-1.0%	0.27	-1.46	0.27	-18.5%	-0.38	-26.1%	0.11	-1.06	0.03	58.2%	0.55	38.3%	1.13	0.96
2	1	-6.4%	-0.09	-11.2%	0.53	-0.86	0.30	-20.5%	-0.37	-28.9%	0.31	-0.24	0.08	43.4%	0.45	8.8%	1.37	1.85
	2	-0.7%	-0.01	-14.9%	0.24	-1.41	0.28	-16.4%	-0.43	-27.1%	0.26	-0.21	0.09	51.4%	0.88	8.9%	0.68	-0.40
	3	-7.0%	-0.13	-18.0%	0.25	-1.04	0.31	-16.4%	-0.35	-27.2%	0.43	0.70	0.14	37.6%	0.65	0.3%	0.89	0.40
	4	-1.6%	-0.02	-9.2%	0.46	-1.17	0.25	-21.8%	-0.49	-28.7%	-0.15	-1.41	0.03	55.9%	0.57	14.8%	1.02	0.54
5	1	-5.8%	-0.09	0.5%	0.39	-0.61	0.25	-18.6%	-0.34	-4.0%	0.33	-0.17	0.08	43.2%	0.53	4.5%	1.17	1.70
	2	-2.4%	-0.05	-0.1%	0.25	-1.26	0.28	-13.0%	-0.35	-3.4%	0.01	-0.52	0.09	47.8%	0.94	7.0%	0.80	0.50
	3	-7.8%	-0.16	-0.7%	0.33	-0.68	0.31	-16.9%	-0.37	-3.6%	0.29	0.24	0.14	38.3%	0.75	5.6%	0.82	0.84
	4	-1.5%	-0.02	0.8%	0.30	-1.37	0.22	-17.6%	-0.34	-4.2%	0.39	-0.23	0.03	53.9%	0.64	5.8%	0.97	0.60
10	1	-10.0%	-0.16	0.1%	0.26	-1.06	0.24	-24.4%	-0.50	-2.0%	-0.20	-1.35	0.08	44.8%	0.56	3.1%	0.92	0.95
	2	-3.6%	-0.08	-0.1%	0.36	-1.24	0.27	-12.8%	-0.35	-1.4%	-0.06	-0.38	0.09	47.1%	0.99	5.1%	0.65	0.18
	3	-7.5%	-0.17	-0.3%	0.07	-1.04	0.31	-15.2%	-0.32	-1.7%	0.38	0.00	0.14	41.1%	0.80	4.8%	0.61	0.01
	4	-6.0%	-0.10	0.0%	0.20	-1.44	0.20	-18.2%	-0.36	-2.0%	0.41	-0.11	0.03	51.9%	0.67	3.3%	0.84	0.38
20	1	-10.4%	-0.18	0.0%	-0.04	-1.51	0.24	-22.5%	-0.43	-1.0%	0.02	-1.18	0.08	45.6%	0.56	2.1%	0.79	0.39
	2	-3.7%	-0.08	-0.1%	0.28	-1.18	0.28	-13.8%	-0.38	-0.8%	0.04	-0.37	0.09	47.3%	0.99	4.1%	0.73	0.34
	3	-5.2%	-0.11	-0.1%	0.03	-1.12	0.32	-14.5%	-0.30	-0.9%	0.34	-0.14	0.14	43.3%	0.80	3.9%	0.69	0.13
	4	-6.7%	-0.12	0.0%	0.28	-1.28	0.20	-17.7%	-0.39	-1.0%	-0.22	-1.33	0.03	50.6%	0.66	2.3%	0.87	0.47

Table A.2: Performance summary statistics (1.5-year maturity, GEV distribution)

CRRA	PS	IPH-QL Non-hedge						IPH-QL Hedge					
		E(R)	SR	CE	Skew	Kurt	Delta	E(R)	SR	CE	Skew	Kurt	Delta
-0.5	1	0.9%	0.02	36.8%	0.47	-0.49	0.33	4.2%	0.10	36.1%	0.22	-0.79	0.45
	2	9.2%	0.33	36.4%	0.98	0.85	0.31	9.5%	0.36	35.3%	0.51	-0.92	0.44
	3	6.9%	0.21	37.6%	0.60	-0.11	0.36	5.2%	0.18	31.4%	0.37	0.16	0.49
	4	3.9%	0.09	39.1%	0.04	-1.39	0.31	4.0%	0.10	35.5%	0.16	-1.19	0.42
0	1	2.4%	0.06	2.4%	0.04	-1.20	0.32	2.1%	0.05	2.1%	0.45	-0.61	0.45
	2	8.3%	0.33	8.3%	0.91	0.15	0.33	9.1%	0.38	9.1%	0.65	-0.48	0.43
	3	9.9%	0.33	9.9%	0.60	-1.21	0.34	8.6%	0.32	8.6%	1.04	0.19	0.45
	4	7.8%	0.17	7.8%	0.11	-1.31	0.32	4.9%	0.12	4.9%	0.43	-0.58	0.41
1	1	2.6%	0.06	-3.8%	-0.06	-1.28	0.30	4.0%	0.11	-1.6%	-0.11	-1.27	0.44
	2	11.3%	0.39	0.6%	0.50	-0.99	0.31	7.0%	0.31	-0.3%	0.78	-0.16	0.42
	3	6.1%	0.21	-1.9%	0.87	0.39	0.32	6.6%	0.26	-0.3%	1.00	0.29	0.44
	4	4.2%	0.10	-2.7%	-0.08	-1.32	0.29	4.9%	0.15	-1.4%	-0.11	-1.27	0.41
2	1	-1.7%	-0.04	-9.7%	-0.01	-0.90	0.30	1.7%	0.05	-6.0%	0.20	-0.86	0.43
	2	4.5%	0.19	-8.4%	0.59	-0.02	0.29	6.9%	0.33	-5.7%	0.47	-0.45	0.43
	3	5.9%	0.21	-7.8%	0.35	-0.51	0.31	6.7%	0.29	-5.8%	0.26	-0.85	0.44
	4	3.1%	0.08	-7.5%	0.28	-0.52	0.28	6.0%	0.17	-5.4%	0.00	-1.13	0.41
5	1	-0.5%	-0.02	0.0%	-0.20	-0.73	0.26	1.1%	0.04	1.1%	-0.05	-0.65	0.40
	2	4.5%	0.19	0.5%	0.19	-1.04	0.29	5.2%	0.25	0.9%	-0.02	-1.72	0.39
	3	4.0%	0.16	0.3%	0.00	-0.80	0.28	1.8%	0.09	0.7%	0.12	-0.86	0.39
	4	4.2%	0.10	0.6%	-0.02	-1.36	0.26	5.4%	0.15	1.2%	-0.03	-1.29	0.38
10	1	-2.2%	-0.07	-0.4%	-0.46	-0.69	0.24	-1.1%	-0.04	0.3%	-0.24	-1.02	0.41
	2	3.8%	0.17	0.2%	0.07	-1.28	0.28	2.3%	0.12	0.8%	-0.03	-1.49	0.40
	3	1.3%	0.06	0.0%	-0.10	-0.82	0.28	0.2%	0.01	0.5%	0.25	-1.06	0.42
	4	2.5%	0.07	0.1%	-0.34	-1.51	0.25	3.4%	0.12	0.7%	-0.38	-1.40	0.38
20	1	-1.1%	-0.03	-0.3%	-0.43	-0.65	0.23	-1.2%	-0.04	0.3%	0.00	-0.94	0.43
	2	3.2%	0.12	0.2%	0.12	-1.24	0.28	3.4%	0.17	0.6%	-0.23	-1.82	0.42
	3	-0.7%	-0.03	-0.1%	-0.08	-1.18	0.29	-0.6%	-0.03	0.4%	0.19	-1.29	0.45
	4	1.8%	0.05	0.0%	-0.37	-1.50	0.24	0.8%	0.03	0.5%	-0.29	-1.82	0.40

Table A.3: Performance summary statistics (1.5-year maturity, GBM distribution)

CRRA	PS	IPH-QL Non-hedge						IPH-QL Hedge					
		E(R)	SR	CE	Skew	Kurt	Delta	E(R)	SR	CE	Skew	Kurt	Delta
-0.5	1	2.7%	0.06	37.4%	0.01	-1.24	0.36	8.2%	0.18	39.1%	-0.22	-1.32	0.48
	2	5.8%	0.23	39.3%	1.40	3.55	0.30	5.8%	0.31	33.7%	0.59	1.04	0.45
	3	7.6%	0.25	37.4%	0.25	-0.35	0.37	7.8%	0.26	33.4%	-0.17	-0.02	0.52
	4	1.0%	0.02	37.1%	0.41	-0.95	0.33	-2.6%	-0.06	30.2%	0.48	-0.85	0.44
0	1	2.7%	0.06	2.7%	-0.20	-1.30	0.36	5.8%	0.14	5.8%	0.08	-0.92	0.5
	2	2.1%	0.09	2.1%	0.92	0.73	0.34	3.6%	0.13	3.6%	0.59	0.03	0.48
	3	8.6%	0.28	8.6%	0.17	0.44	0.34	9.4%	0.35	9.4%	0.22	-0.41	0.52
	4	0.1%	0.00	0.1%	0.57	-0.94	0.34	0.6%	0.02	0.6%	0.7	-0.31	0.46
1	1	4.8%	0.10	-2.8%	-0.03	-1.22	0.30	4.6%	0.11	0.7%	0.08	-0.9	0.47
	2	8.0%	0.32	-2.5%	0.59	0.64	0.29	4.2%	0.2	-2.7%	0.8	-0.08	0.45
	3	9.6%	0.35	-0.8%	-0.09	1.57	0.34	6.6%	0.22	1.6%	0.52	0.02	0.5
	4	4.7%	0.11	-2.6%	0.62	-0.32	0.28	-0.1%	0	-4.9%	0.56	-0.7	0.44
2	1	1.6%	0.04	-8.2%	0.13	-0.39	0.29	2.1%	0.06	-5.0%	0.04	-0.89	0.45
	2	7.0%	0.25	-9.1%	0.02	0.45	0.26	8.2%	0.34	-5.4%	0.22	0.14	0.42
	3	6.8%	0.24	-6.8%	0.08	0.59	0.33	6.5%	0.23	-3.1%	0.21	-0.22	0.49
	4	-1.5%	-0.04	-10.5%	0.10	-0.18	0.23	5.3%	0.15	-7.2%	0.2	-1.08	0.41
5	1	6.2%	0.16	1.0%	-0.35	-0.70	0.29	2.0%	0.06	2.9%	-0.04	-1.18	0.46
	2	5.8%	0.22	0.4%	0.24	0.61	0.28	6.3%	0.25	1.8%	0.83	0.15	0.43
	3	2.3%	0.09	1.0%	0.19	1.78	0.32	7.2%	0.34	2.1%	0.05	-0.28	0.5
	4	-1.2%	-0.03	-0.3%	-0.47	-0.28	0.21	5.0%	0.17	1.0%	-0.37	-0.91	0.37
10	1	-0.2%	-0.01	0.1%	-0.22	-0.46	0.26	0.7%	0.02	1.1%	-0.17	-1.44	0.45
	2	5.2%	0.20	0.4%	0.15	0.47	0.27	4.6%	0.22	1.1%	0.22	-0.41	0.42
	3	2.5%	0.10	0.4%	0.05	1.52	0.32	6.0%	0.28	1.5%	0.11	-0.75	0.51
	4	1.7%	0.05	-0.3%	-0.39	-0.35	0.19	1.6%	0.06	0.6%	-0.47	-1.08	0.37
20	1	-0.8%	-0.03	-0.1%	-0.62	0.15	0.25	1.0%	0.04	0.6%	-0.13	-1.47	0.46
	2	3.2%	0.13	0.2%	-0.32	-0.40	0.28	4.9%	0.22	0.7%	0.47	0.38	0.44
	3	4.0%	0.16	0.3%	-0.10	-0.54	0.34	2.8%	0.13	1.0%	0.09	-0.77	0.53
	4	-0.9%	-0.03	-0.1%	-0.63	-0.67	0.20	3.4%	0.13	0.4%	-0.68	-0.99	0.38

APPENDIX B

Appendix for Chapter 3

Table B.1 shows that Feb 2014 maturity presents the most volatile properties with the largest standard deviation (std.), skewness (skew.) and kurtosis (kurt.). The longer the maturity is, the smaller the std. and the absolute values of skew. and kurt. are. Table B.2 summarizes the performances of our algorithms while Table B.3 shows the same except without edge strike prices.

Table B.1: Summary statistics of implied volatility on 01/27/2014

	Maturity	Mean (%)	Std. (%)	Skew.	Kurt.
Call Bid	Feb 2014	11.74	7.98	-2.51	6.11
	Mar 2014	14.42	1.86	-2.37	14.73
	April 2014	13.66	0.98	-0.06	-0.56
	May 2014	14.47	0.81	-0.03	-0.64
	June 2014	15.05	0.78	-0.07	-0.43
Call Ask	Feb 2014	14.10	5.20	-3.57	15.40
	Mar 2014	15.03	1.48	-0.86	4.72
	April 2014	14.02	0.96	-0.03	-0.55
	May 2014	14.80	0.79	0.02	-0.59
	June 2014	15.34	0.77	-0.04	-0.42
Put Bid	Feb 2014	14.53	2.30	0.28	-0.95
	Mar 2014	14.40	1.54	-0.21	-0.67
	April 2014	15.06	1.02	-0.02	-0.66
	May 2014	14.88	0.80	-0.01	-0.67
	June 2014	14.97	0.82	0.03	-0.50
Put Ask	Feb 2014	15.30	2.35	0.10	-1.01
	Mar 2014	14.83	1.53	-0.25	-0.58
	April 2014	15.38	1.03	-0.01	-0.72
	May 2014	15.19	0.82	0.00	-0.74
	June 2014	15.24	0.84	0.04	-0.57

Table B.2: Performance summary of our algorithms (in %)

		Call Bid		Call Ask		Put Bid		Put Ask	
	Kernel	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE
IVS-KPSVR	Gaussian	12.05	2.26	11.76	2.45	14.38	1.61	12.36	1.66
	Linear	12.20	2.08	11.70	2.14	14.42	1.63	12.37	1.67
IVS-BKPSVR	Gaussian	12.51	2.39	12.33	2.61	15.24	1.69	13.12	1.73
	Linear	12.86	2.24	12.38	2.31	15.40	1.68	13.18	1.72
IVS-EKPSVR	Gaussian	12.09	2.27	11.86	2.48	14.58	1.63	12.45	1.66
	Linear	12.23	2.08	11.71	2.14	14.48	1.63	12.34	1.66

Table B.3: Performance summary of our algorithms without edge strikes (in %)

		Call Bid		Call Ask		Put Bid		Put Ask	
	Kernel	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE	MAPE	RMSE
IVS-KPSVR	Gaussian	8.13	1.47	7.80	1.50	9.90	1.23	8.41	1.21
	Linear	8.50	1.50	8.13	1.51	10.16	1.27	8.68	1.25
IVS-BKPSVR	Gaussian	8.32	1.53	8.06	1.58	10.27	1.27	8.76	1.25
	Linear	9.30	1.62	8.95	1.63	11.44	1.36	9.55	1.30
IVS-EKPSVR	Gaussian	8.14	1.48	7.85	1.51	9.99	1.24	8.45	1.21
	Linear	8.53	1.51	8.15	1.51	10.17	1.28	8.64	1.25