Reinforcement Learning for American Put Options

Author: Meshal AL Marzuqi
Supervisor: Prof. John Shawe-Taylor

Computer Science

University College London

September 2016

Abstract

Pricing derivatives with American features remains a challenging and an interesting problem for the financial engineering community, where standard numerical methods such as finite differences become intractable in a multi-factor environment. In this thesis we compare the performance of three state-of-the-art simulation based continuous Reinforcement Learning algorithms Longstaff-Schwartz (LSM), Fitted-Q Iteration (FQI), and Least Squares Policy Iteration (LSPI) as applied to the pricing and trading of American put options. Initially we closely follow the paper of Schuurmans et. al. [35] and attempt to re-create their results by fully mathematically deriving each algorithm in order to gain in-depth understanding of they work and by designing the necessary coding architecture, which can be built upon. The authors report the largest payoffs given by the FQI algorithm, followed by LSPI, which gives similar but slightly lower payoffs, followed by LSM, which is reported to give poor payoffs (sometimes lower than its European counterpart) and very choppy exercise boundaries. Upon the initial re-creation of the results, our payoffs agree with those of the original paper, where FQI finds the highest payoffs, followed by LSPI and LSM. Next we seek to improve the reported results and investigate how additional basis functions (which depend on the option's volatility) affect learning of the exercise policies. We find that the weights learnt for volatility basis are close to zero, indicating that these basis were not found to be important for the continuation value function approximation. Lastly we attempted to improve the policies learnt using the LSM and their associated exercise boundaries, since this algorithm is the default method of choice within the Financial Engineering community for pricing derivatives with American features. We discover that our version of the Longstaff-Schwartz algorithm is able to gain considerably larger payoffs and produce boundaries which are the best out of all the tested algorithms (smooth and monotonically increasing) and we argue that this is an original contribution to the Schuurmans et. al. paper.

Nomenclature

$C^A(S,t)$	American call price at a given underlying price and a given time $\in \mathbb{R}^{>0}$
$P^A(S,t)$	American call price at a given underlying price and a given time $\in \mathbb{R}^{>0}$
Δ_t	Amount of stock at time t$\in \mathbb{R}$
$f(t,S,\sigma)$	Basis Function
$\zeta_{1i}(t,S,\sigma)$	Complex Volatility Basis Function $i=1,2$
$N(\cdot)$	Cumulative distribution function of the standard normal distribution
γ	Discount Factor
$\widehat{Q}^{\pi}(S(t),t)$	Estimated option continuation value $\in \mathbb{R}$
$C^E(S,t)$	European call price at a given underlying price and a given time $\in \mathbb{R}^{>0}$
$P^E(S,t)$	European put price at a given underlying price and a given time $\in \mathbb{R}^{>0}$
$V(S_t,t)$	Function of the value of the underlying at t
$g(S(t))$	Intrinsic value of the option $\in \mathbb{R}^{\geq 0}$
$\max_{FQI}$	Maximum number of iterations for FQI algorithm $\in \mathbb{R}^{Z^+}$
$\max_{LSPI}$	Maximum number of iterations for LSPI algorithm $\in \mathbb{R}^{Z^+}$
$\min_{FQI}$	Minimum number of iterations for FQI algorithm $\in \mathbb{R}^{Z^+}$
$\min_{LSPI}$	Minimum number of iterations for LSPI algorithm $\in \mathbb{R}^{Z^+}$
$\psi(t)$	No Volatility Basis Function Dependent on Time

$\varphi(S)$	No Volatility Basis Function Independent of Time
n_ϕ	Number of basis functions $\in \mathbb{R}^{Z^+}$
N_{real}^{train}	Number of real data generated paths used for training $\in \mathbb{R}^{Z^+}$
N_{real}^{test}	Number of real generated paths used for testing $\in \mathbb{R}^{Z^+}$
N_{synth}^{test}	Number of synthetically generated paths used for testing $\in \mathbb{R}^{Z^+}$
N_{synth}^{train}	Number of synthetically generated paths used for training $\in \mathbb{R}^{Z^+}$
N_{real}^{test}	Number of testing trajectories obtained from real data $\in \mathbb{R}^{Z^+}$
N_{synth}^{test}	Number of testing trajectories obtained from synthetic data $\in \mathbb{R}^{Z^+}$
N_{real}^{train}	Number of training trajectories obtained from real data $\in \mathbb{R}^{Z^+}$
N_{synth}^{train}	Number of training trajectories obtained from synthetic data $\in \mathbb{R}^{Z^+}$
$\boldsymbol{\pi}$	Optimal policy / exercise strategy $\in \mathbb{R}^{>0}$
$Q(S(t),t)$	Option continuation value $\in \mathbb{R}$
K	Option strike price $\in \mathbb{R}^{>0}$
T	Option tenor (maturity) $\in \mathbb{R}^{Z^+}$
π	Policy
Π_t	Portfolio of one unit of derivative plus a number of Δ_t of underlying stock
$\Pr$	Probability $\in \mathbb{R}^{>0}$
$\mathcal{T}(s,a,s')$	Probability of transitioning to next state s' from state s under action a, $\in \mathbb{R}^{>0}$
$\mathcal{R}$	Reward function - amount of reward (positive/negative) received for the state transition $\in \mathbb{R}$
μ	Risk neutral expected stock return $\in \mathbb{R}$
r	Risk neutral interest rate $\in \mathbb{R}$

a	RL action $\in \mathbb{R}^{Z^+}$
a'	RL next action $\in \mathbb{R}^{Z^+}$
s'	RL next state $\in \mathbb{R}^{Z^+}$
s	RL state $\in \mathbb{R}^{Z^+}$
$\mathcal{A}(s)$	Set of all actions an agent can encounter $\in \mathbb{R}^{Z^+}$
$\mathcal{S}$	Set of all states an agent can encounter $\in \mathbb{R}^{Z^+}$
$\zeta_i(t, S, \sigma)$	Simple Volatility Basis Function $i = 1, 2$
σ	Stationary stock volatility $\in \mathbb{R}^{>0}$
$d\mathbb{W}_t$	Stochastic differential into the future $\sim \mathcal{N}(0, dt)$
dS_t	The change in the stock price at time t
dt	The change in time t$\in \mathbb{R}^{>0}$
$\mathbf{w}(t)$	Time dependent weight vector $\in \mathbb{R}^{n_\phi \times T}$
$w(t)$	Time dependent weight vector at time step $t \in \mathbb{R}^{n_\phi}$
w	Time independent weight vector at time step $\in \mathbb{R}^{n_\phi}$
t	Time step $\mathbb{R}^{Z^+}$
$S(t)$	Underlying price at time t, $\in \mathbb{R}^{>0}$
$\sigma(t)$	Volatility of the stochastic process $S(t)$, $\in \mathbb{R}^{>0}$
$\mathbb{W}_t$	Wiener process / Brownian motion $\sim \mathcal{N}(0, t)$

Table of contents

List of figures

List of tables

Chapter 1

Introduction

"Prediction is very difficult, especially about the future."

— *N. Bohr*

The Financial Engineering discipline is concerned with finding the fair value of derivative products (contracts traded between two parties on an exchange or over the counter, and which are based on underlying assets such as cash and other derivative products). The modern theory of derivative pricing began with the discovery of the Black-Scholes partial differential equation [8], the analytical solution of which provides a fair price of European vanilla options [28]. In practice non-vanilla complex derivatives, such as those with American-style features, are encountered across all asset classes. A derivative is said to have an American style if it can be exercised before the expiration date (*early exercise*). Examples of such derivatives are equities: American options; foreign exchange and commodities: instalment swaps, American options (not very popular) [36]; interest rates: callable debt, exotic swaps [14]; credit and mortgages: callable debt, convertible debt, mortgages; real estate, emerging markets, convertible and energy, which all have an early exercise optionality.

An American call (put) option gives holder the right to buy (sell) an underlying asset at time ($S(t)$) for a pre-specified price (*strike*), on an expiration date (*maturity*) or earlier. There are many variables (*factors*) that affect the option price (*premium*), which among many are: underlying stock price and its volatility $\sigma(t)$, maturity T, interest rate r, strike price K and dividends. The option premium is made of two components: the *intrinsic value* - the amount option holder receives at exercise/expiry (i.e. for call option this is a maximum of the difference between underlying price and strike or zero, $\max(S_t - K, 0)$ or equivalently $(S_t - K)^+$; for the put option this is the maximum of the difference between strike and underlying price or zero, $\max(K - S_t, 0)$ or equivalently $(K - S_t)^+)$, and the *time value* - the time to option's maturity, where longer time to expiration leads to higher option price [7, 24].

The task of pricing an American option encompasses finding an optimal stopping (i.e. exercise) time. Pricing American options remains a fundamental problem in financial engineering, particularly when multiple factors are accounted for and in which case traditional pricing methods become impractical. This lead to the application of simulation techniques for valuing American options, capable of dealing with multiple factors, both path-dependent and American exercise features, and allowing the underlying asset to follow varying stochastic processes (e.g. jump diffusion, non-Markovian and semi-Martingale processes) [37]. Finding the fair value of an American option [24] forms a platform for pricing more complex derivatives with similar early exercise optionality. At any time point, the holder of an American option optimally compares the expected future payoff from continuing to hold the option (*continuation value*, $Q(S(t), t)$) vs. the payoff from its immediate exercise (the intrinsic value of the contract) and determines the most profitable action. Thus the optimal exercise strategy is fundamentally determined by the conditional expectation of the payoff from contiuing to keep the option alive [37]. An American call option written on an underlying with no dividends/carry is equivalent in price to a European call option and should always be held until expiration (see Chapter 2 for further explanation) [24]. However, for American call options on an underlying with dividends and for American put options an *exercise boundary* needs to be determined, such that if the underlying price falls above (below) the boundary the corresponding call (put) option is exercised to gain maximum payoff. The exercise boundary is a function of time, defined all the way until expiration date.

1.1 Existing Pricing Methods

The Black-Scholes model provides a closed form analytical solution for the linear problem of valuing European style options (see Section 2.1.1 for a full derivation). The difficulty in finding a closed form solution for American options lies in the fact that they can be exercised early, thus changing the mathematical problem into a *free boundary problem.* The optimal/early exercise boundary in the American option formulation is now time dependent (vs. European which is not) and forms a part of the solution, thus making the problem a highly non-linear one.

1.1.1 Analytical Methods

Several analytical approximation methods for pricing American options exist, however these are typically applicable for pricing American options with either short term (up to 2 years) or very long term (10 years) maturities and are often not convergent [33]. These are based on one of the following techniques: **(a)** approximate schemes with exact representation of the free boundary problem, where the accuracy of the solution is improved with an addition of more terms (however these rapidly become inefficient) [32]; **(b)** regression techniques that fit an analytical approximation relying on upper/lower bounds of an American option (these require computing large number of options accurately in order to learn the regression coefficients and may not be convergent) [15]; **(c)** analytical approximations [4, 38], such as those which consider the free boundary problem as an option on an option (i.e. compound option) problem and find the boundary at discrete time points, which are then extrapolated [25].

1.1.2 Numerical Methods

Majority of American options are priced by non-analytic solution techniques and the most well-known numerical methods are Binomial Trees and Finite Differences. However these are computationally intensive and are unable to deal with jump diffusion processes or multi-factor models. The *Binomial model* is an approximation to the assumed behaviour of an underlying asset price. It matches the risk-neutral mean, μ, and standard deviation, σ, of the Geometric Brownian Motion (GBM) process (described by a stochastic differential equation), which the underlying stock is assumed to follow:

$$dS_t = \mu S_t dt + \sigma S_t d\mathbb{W}_t \tag{1.1.1}$$

where S_t is stock price at time t, μ is risk neutral expected stock return, *drift* (for a non-dividend paying stock $\mu = r$), σ is stationary stock volatility, $\mathbb{W}_t \sim \mathcal{N}(0, t)$ is Wiener process / Brownian motion, and $d\mathbb{W}_t \sim \mathcal{N}(0, dt)$ is stochastic differential into the future.

Binomial Tree The *Binomial Tree* model can be used to price European and American style options. It assumes that at each time step t the underlying asset price either goes up with a fixed amount and probability, p, or goes down with a fixed amount and probability $(1 - p)$.

Since there are three parameters (amount up 'u', down 'd' and probability 'p') but only two are matched (mean and standard deviation), this leads to a free choice in one parameter 'p', resulting in three models: Trigeorgis [45], Cox, Ross & Rubinstein [20], and Jarrow & Rudd [30].

Finite Difference The *Finite Difference* method solves differential equations by converting them to difference equations where finite differences approximate the derivatives and solve them by going backwards in time which is naturally suited for pricing American options. These methods, however, can carry high computational demand, may be slow and may not be tractable in multi-factor models / high dimensional state spaces [31].

Monte Carlo The *Monte Carlo* (MC) method uses a simulation approach and was first applied to European option pricing [11]. Many sample paths for the underlying asset are generated according to a model for the price process (e.g. as given by Equation 1.1.1) using random numbers going forward in time. A European option is then priced along each simulated path and an average across paths is taken as the estimated option price. American options require an optimal exercise boundary as part of the solution and are naturally priced backward in time due to early exercise. More recently MC methods have been applied to derivatives with American features, since they can incorporate jump diffusions for the underlying process and are successful for high dimensional problems (more than one factor e.g. dependency on many underlyings). An approximation to the optimal stopping rule is derived based on comparing the estimated payoff for stopping at a given time step with an estimated payoff for continuing to hold an option [40]. In such situations MC methods converge faster to a solution and require less computational power than numerical methods, thus making them the focus of our research.

1.2 Reinforcement Learning for American Options

The problem of finding an optimal exercise strategy for American-style derivatives falls naturally under the framework of Reinforcement Learning (RL) [35, 48], and can be presented as a Markov Decision Process (MDP)[1] [43]. An MDP process is defined by a 4-tuple $<\mathcal{S}, \mathcal{A}(s), \mathcal{T}, \mathcal{R}>$, where $\mathcal{S}$ is the set of all states an agent (decision maker) can encounter; $\mathcal{A}(s)$ is the set of all available

[1]The MDPs require that the system adheres to the *Markov Property* – meaning that each consecutive state and reward depend only on the previous state and action.

actions; $\mathcal{T} : \mathcal{S} \times \mathcal{A}(s) \times \mathcal{S} \to \mathbb{R}^{\geq 0}$ is the *transition probability*, $\mathcal{T}(s, a, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ i.e. the probability of transitioning to state s' from state s under action a; and $\mathcal{R} : \mathcal{S} \times \mathcal{A}(s) \times \mathcal{S} \to \mathbb{R}$ is the *reward function*, $\mathcal{R}(s, a, s') = E_{\pi}(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$, i.e. the amount of reward (positive/negative) received for the state transition. At each time step the agent: **(1)** senses the state of an environment in which the system is present in, **(2)** selects an action from the set of available actions, while present in that state, and **(3)** obtains a reward for performing an action in a given state and transitioning to a new state (see Figure 1.1). The value function forms the core of RL and is defined as the expected return when starting in state s and following policy $\boldsymbol{\pi}$ thereafter, $V^{\pi}(s) = E_{\pi}(R_t | s_t = s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right]$.

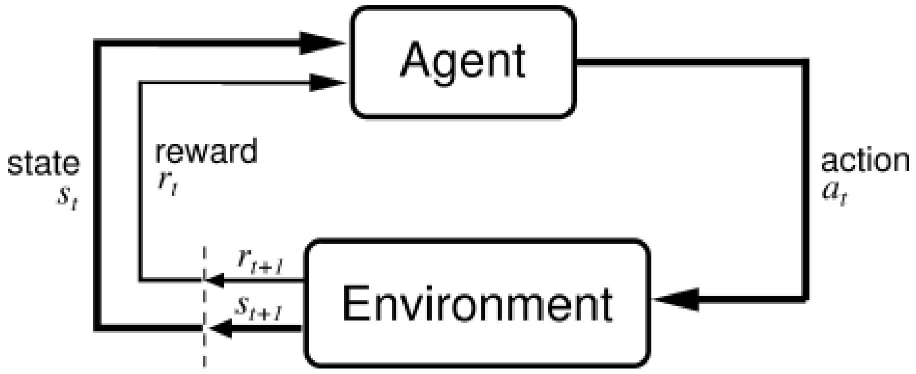


Fig. 1.1 **Agent-environment interaction in reinforcement learning.** At each time step the Agent (decision maker) implements a mapping (policy,$\pi_t(s, a) = p(a_t = a | s_t = s)$) from states to probabilities of selecting each possible action. The environment responds to the action applied to it and transitions to the next state, s_{t+1}, displaying a reward r_{t+1} for choosing action a_t in state s_t and transitioning to the next state s_{t+1}. The agents total goal is to maximise the amount of long term reward it receives. Adapted from Sutton & Barto book [43].

1.2.1 Mapping American Option Pricing into MDP Framework

In cases where the state space is low-dimensional and the transition probability function is known in advance, an MDP problem can be solved by methods of Dynamic Programming, for example the classic value iteration and policy iteration algorithms, which repeatedly apply the Bellman Equation [43] (see Section 2.2.2 for a full explanation). However, real life problems often have multi-dimensional states (which could lead to the curse of dimensionality[2], making the problem intractable) or have transition probability functions which are unknown *a priori*, thus requiring learning without their knowledge (*model-free* algorithms) or inferring them from data (*model-based* algorithms). Continuous RL methods involving Temporal Difference solutions

[2] Curse of dimensionality – an exponential growth in the complexity of a problem as the dimensionality of the task increases (in practice up to 6 dimensions can be discretised successfully).

[43] do not assume a model of the environment and learn by repeatedly interacting with the environment while maximising long term reward. More recently TD methods have been applied in a MC setting for estimating American option prices, without relying on finance theory, and by using historical/implied data instead in order to find estimated continuation value, $\widehat{Q}^{\pi}(S(t), t)$. The problem of American option pricing can be mapped into an MDP as follows:

1. State s: price of the underlying, $S(t)$, time t, and option status (exercised or alive).
2. Actions $\pi(S, t)$: "exercise, 1" if $\Big(K - S_t > \widehat{Q}^{\pi}(S(t), t)\Big)$ and "keep, 0" otherwise.
3. Reward $r(S, t)$: intrinsic value $g(S(t)) = max\Big(K - S(t), 0\Big)$ if exercised, and zero otherwise.
4. Next state s': determined by the time series dynamics of the underlying price.

Traditional RL algorithms lack stability and/or convergence guarantees when combined with value function approximation, thus much work has been done to come up with a suitable solution. Early research in simulation based approaches addresses the problem of pricing American options, focused on maximising the simulated value of the option [9], while later approaches used parametrisation techniques to approximate the early exercise boundary [10, 16, 17, 19, 29, 39]. The three most prominent algorithms which combine RL with Monte Carlo are described below.

1.3 Simulation Based American Option Pricing

The difficulty in using MC for derivatives with early exercise optionality resides in finding an optimal point of exercise of the option along each path. The option's price is defined as the expected price when using the optimal exercise policy (the best known policy based upon the presently available information). The first extension of MC techniques to address the problem of pricing American options was carried out by Longstaff and Schwartz [22]. Originally, this Least Squares Monte-Carlo [37] method was thought of as the bridge between classic option pricing ideas and statistical estimation. However, now the method is classified as part of an RL technique involving value function approximation. The LSM method is widely applicable by practitioners to pricing various derivatives with early exercise features and it's convergence properties have been extensively studied [37, 48].

Longstaff-Schwartz Method (LSM) The LSM algorithm estimates the conditional expectation value function of the derivative at each time step, starting at expiration and working backwards in time [37]. At each step it regresses the ex-post realised payoffs from continuation on functions of values of the state variables. The fitted value from the least squares regressions provides a direct estimate of the conditional expectation function, thus allowing to establish the optimal exercise strategy along each path. The option is then valued by starting at the present time, moving forward along the path until the first stopping time occurs and discounting the resulting cash flow. This is repeated for all in-the-money paths and an average provides the option price estimate. The authors report results which, in the single factor setting, are indistinguishable from the more computationally demanding finite difference method solutions. The continuation value at expiry is given by the intrinsic value $g(S_T)$ and at each time step prior it is estimated by finding the minimiser $w(t)$ of the loss [35] (0 means 'keep option alive')

$$L_t(w) = \sum_{j=1}^{N} \left[\widehat{Q}^{\pi}\Big((S_t, t), 0; w\Big) - max\Big(g(S^j_{t+1}), \gamma \widehat{Q}^{\pi}\Big((S^j_{t+1}, t+1); w_{t+1}\Big)\Big) \right]^2 \tag{1.3.1}$$

Fitted-Q Iteration (FQI) The FQI algorithm is proposed in Section 6 of Tsitsiklis et. al. [48] who study its properties, including convergence. Initially the authors present an overview of value iteration methods, where at each step value function Q is approximated via sampling sub-set of representative states only. The authors show that, if the representative states are sampled according to an arbitrary measure, the estimation error may grow exponentially with time horizon. They develop a special variant of approximate value iteration for finite horizon problems, where rather than approximating the value function at each time period, they simultaneously approximate all parameter vectors. Alternatively, if sequences of states are sampled from the empirical measure (i.e. bootstrapped from the history), the estimation error is bounded. The FQI algorithm uses basis functions that generalise over both state space and time, uses iterative calculations corresponding to taking expectation w.r.t empirical measure provided by simulations, and re-uses simulated trajectories upon each pass over the algorithm (which guarantees convergence of the parameter vector sequence) [48]. The continuation value is estimated by finding the minimiser w of least squares approach to the fitted Q-iteration [35]

$$\sum_{j,t}\left[\widehat{Q}^{\pi}\Big((S_t^j,t),0;w\Big)-\gamma max_{a\in\mathcal{A}(s)}\widehat{Q}^{\pi}\Big((S_{t+1}^j,t+1),a\Big)\right]^2 \tag{1.3.2}$$

Least Squares Policy Iteration (LSPI) LSPI is an off-line, off-policy, model-free algorithm, which learns the state-action value function, $\widehat{Q}^{\pi}(s,a;w)$, through a linear weighted combination of basis functions [34]. It allows for an incremental policy improvement within a policy iteration framework and provides an architecture which is more powerful than black box methods such as neural networks. LSPI has an ability to generalise to unseen experience and its convergence properties have been well studied [1, 35, 48]. The finite time bounds on the performance of the algorithm are more relevant in practice and have been studied in Schuurmans et. al. [35], while based on the mathematical machinery developed in Munos et. al. [1].

The above methods belong to batch-based RL iterative methods with value function approximation [43]. Schuurmans et. al. compared these methods for application to American put option pricing [35] and applied the algorithms to real and simulated data with parameters derived from historical data. This thesis attempts to re-create their results when tested on the DJI stock index option and looks for ways to improve the testing methodology in order to gain a better understanding at how these three algorithms compare. The contribution of this thesis consists of examining a bigger cross-section of stocks, testing a different historical period and suggests an additional set of basis functions which are volatility dependent.

1.4 Thesis Organisation

The rest of the thesis is organized as follows: Chapter 2 develops the mathematical background necessary for carrying out comparison of the Monte Carlo based Reinforcement Learning methods for American option pricing. Chapter 3 compares the three state-of-the-art algorithms LSPI, FQI and LSM for Amercian put option pricing of the Dow Jones Option Index. This research is largely based on Schuurmans et. al. paper [35] and attempts to re-create the reported results, while thoroughly researching each algorithm's performance. Chapter 4 proposes improvements to the suggested basis functions [35] and tests the algorithms with certain ideas for improvement. Chapter 5 provides a general discussion and conclusions.

Chapter 2

Background

"It's more interesting to live not knowing than to have answers which might be wrong."

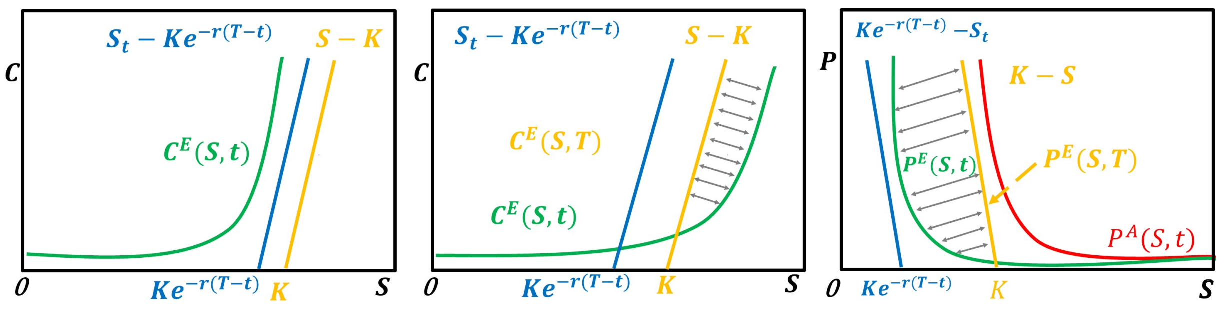
— R. Feynman

This Chapter defines key concepts and develops all the necessary mathematical tools in order to enable the comparison of three most prominent Reinforcement Learning Monte Carlo algorithms used for pricing American Options. First the pricing theory for both European and American options is discussed, followed by a thorough Reinforcement Learning background for discrete and continuous cases. The detailed derivations for each state-of-the-art reinforcement learning algorithm are given in the last section.

2.1 Option Pricing

The theory of option pricing is concerned with finding a fair value of a contract which has an optionality of being executed early (American options) or upon expiry (European options), as well products with more exotic exercise functionality. The typical payoffs received for European and American call/put options with/without dividends is discussed in Figure 2.1. The price of a European option has a closed form analytical solution derived from the famous Black and Scholes partial differential equation, which earned its authors a Nobel Prize.

An American option is either equal to or more expensive than its European option counterpart, because of the extra optionality of early exercising (which may not be optimal and thus the early exercise policy needs to be established). The price of an American call without dividends is



(a) **Euro/American call, no div** (b) **Euro call, with div** (c) **Euro and American put**

Fig. 2.1 **Option Payoffs Diagrams.** Depicts the payoffs of the European and American calls and puts with and without dividends. **(a)** The payoff of a European call without dividends prior to expiry is depicted with a green curve. The blue solid line indicates the difference of the stock price at time t and the discounted strike K. Upon expiry the yellow solid line indicates the value of the option, which is the intrinsic value if in the money $S_T - K$ or zero. The American call option on a non-dividend paying asset is equivalent to the European call option and hence will not be exercised early, and has an equivalent payoff as its European counterpart. **(b)** Shows the payoff of a European call with dividend payments, where the value of the call prior to expiry can go below $max(S_t - K, 0)$, which is sometimes called *Negative Intrinsic Value* (grey shaded area), thus making early exercise profitable. **(c)** Value of a European put prior to expiry can also have negative intrinsic value (grey shaded area; green curve is below the yellow intrinsic value at expiry line), making early exercise profitable. However the price of the American put can never go below the intrinsic value (red curve is above the yellow line). Thus the price of an American put option (red curve) is always strictly higher than its European put counterpart (green curve).

equivalent in price to the European call option and should never be exercised before expiration. The reason for that is three fold:

- Holding an American call is equivalent to holding the underlying stock, with an added benefit of protection against a downward movement below the strike price.
- It costs more to exercise the option at a given strike today vs. at the same strike in the future due to the time value of money.
- The time value of the option would be lost by exercising early.

However the price of an American call on an underlying that pays dividends may not equal the price of the European call, because:

- The option holder is entitled to dividends after exercising the call which they would not receive otherwise.

An American put option on an underlying with or without dividends can diverge from its European counterpart. This is because:

- The option has a limited payoff since the underlying stock cannot be negative.

Table 2.1 **Determinants of Option Value.**

Factor	**Value of a Call**	**Value of a Put**
Increase in Stock Price	Increases	Decreases
Increase in Strike Price	Decreases	Increases
Increase in variance of underlying asset	Increases	Increases
Increase in time to expiration	Increases	Increases
Increase in interest rates	Increases	Decreases
Increase in dividends paid	Decreases	Increases

- It may be optimal to exercise a deep in-the-money put and earn risk free rate on the profit.

Thus the two exceptions for early execution are (see Table 2.1 for factors affecting the option price): **(a)** when the underlying asset pays large dividends, decreasing the value of underlying asset thus in tern devaluing the price of a call. In this case it will be better to execute the option just before the ex-dividend date if the time premium of the call is less than the expected decline in stock price. **(b)** when an investor holds a deep in the money put during high interest rate levels. The time value of the put might be less than the gain from exercising the put early and earning interest on the strike. This thesis is concerned with pricing American put on non-dividend paying stocks, thus more attention is made on learning an optimal early exercise of these options (see Figure 2.2 for an explanation with an example).

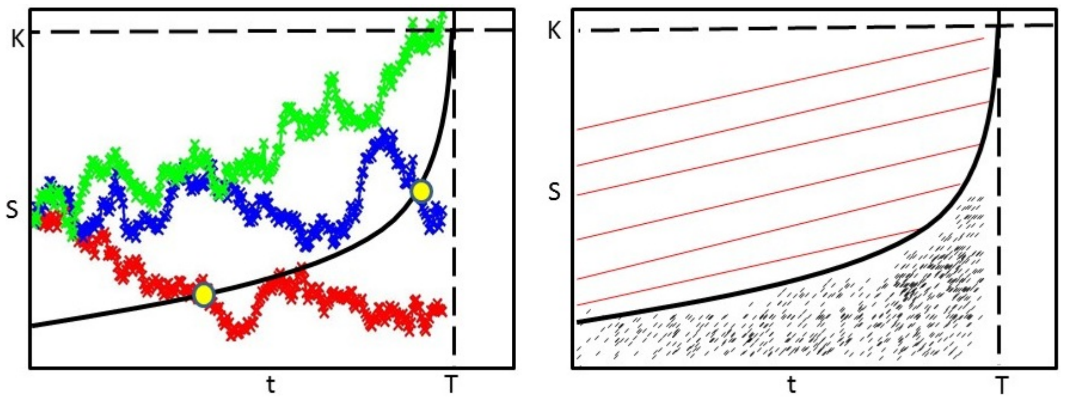


Fig. 2.2 **American Put Option Continuation Boundary.** The figure explains the American option optimal exercise boundary and depicts the continuation region. The strike, K, and the expiry date, T, are marked with black dashed lines. The optimal exercise boundary is marked with a solid black line. **(a)** Depicts the optimal exercise boundary, $C(t, S^*(t))$, with three example stock realisations (red, blue, green lines), where S^* is the stock price lying on the exercise boundary. The early exercise decision is taken when the stock price falls below the optimal exercise boundary (yellow solid circle), i.e. American option is exercised early at the boundary. In our case the stock realisations depicted by the red and blue lines go down in price so that the American put option is exercised prior to expiry, T. The green stock realisation is not exercised early and is out of the money on expiry, therefore the American put option would not be exercised. **(b)** Depicts the continuation region, C, which satisfies the Black and Scholes PDE, (red solid lines), the optimal exercise boundary (black solid line) and the region where the American put option price is in-the-money, $P_t^{(A)} < (K - S_t)$.

2.1.1 European Options

We will now provide the derivation of the Black–Scholes PDE using the hedging argument, where a portfolio Π is created from a derivative, those price at time t is $V(S_t, t)$ put a Δ_t amount of the underlying S sold short. Hence the value of such portfolio at time t is

$$\Pi_t = V(S_t, t) - \Delta_t S_t \tag{2.1.1}$$

$$d\Pi_t = \underbrace{dV(S_t, t)}_{=V(S_t+dS_t, t+dt)} - \underbrace{d(\Delta_t S_t)}_{=\Delta_t dS_t} \tag{2.1.2}$$

The Taylor expansion of an arbitrary function of 2 variables $f(x, y)$ is

$$\begin{aligned} f(x+h, y+k) &= f(x,y) + \frac{\partial f}{\partial x}h + \frac{\partial f}{\partial y}k + \frac{1}{2}\Big(\frac{\partial^2 f}{\partial x^2}h^2 + \frac{\partial^2 f}{\partial y^2}k^2 + 2\frac{\partial f}{\partial x \partial y}hk\Big) \\ \underbrace{f(x+h, y+k) - f(x,y)}_{df(x,y)} &= \frac{\partial f}{\partial x}h + \frac{\partial f}{\partial y}k + \frac{1}{2}\Big(\frac{\partial^2 f}{\partial x^2}h^2 + \frac{\partial^2 f}{\partial y^2}k^2 + 2\frac{\partial f}{\partial x \partial y}hk\Big) \end{aligned} \tag{2.1.3}$$

Thus using Equation 2.1.3 the Taylor expansion of the derivative $V(S_t, t)$ for two variables (i.e $f(x, y)$ is now $V(S_t, t)$) is

$$\begin{aligned} V(S_t + dS_t, t + dt) &= V(S_t, t) + \frac{\partial V}{\partial S}dS_t + \frac{\partial V}{\partial t}dt + \frac{1}{2}\Bigg[\frac{\partial^2 V}{\partial S^2}(dS_t)^2 + \frac{\partial^2 V}{\partial t^2}(dt)^2 \\ &\qquad + 2\frac{\partial V}{\partial S \partial t}\underbrace{dS_t dt}_{(\mu S dt + \sigma S dW)dt = 0}\Bigg] \\ \underbrace{V(S_t + dS_t, t + dt) - V(S_t, t)}_{dV(S_t,t)} &= \frac{\partial V}{\partial S}dS_t + \frac{\partial V}{\partial t}dt + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(\sigma^2 S^2 dt) \\ dV(S_t, t) &= \frac{\partial V}{\partial S}dS_t + \frac{\partial V}{\partial t}dt + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}(\sigma^2 S^2 dt) \\ &= \Big(\frac{\partial V}{\partial t} + \frac{1}{2}\frac{\partial^2 V}{\partial S^2}\sigma^2 S^2\Big)dt + \frac{\partial V}{\partial S}dS_t \end{aligned} \tag{2.1.4}$$

Thus $d\Pi_t$ as given by the Equation 2.1.1 and by the Absence of Arbitrage, becomes

$$\begin{aligned} d\Pi_t &= \Big(\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2}\Big)dt + \frac{\partial V}{\partial S}dS_t - \Delta_t dS_t \\ &= \Big(V_t + \frac{1}{2}\sigma^2 S^2 V_{SS}\Big)dt + (V_S - \Delta_t)dS_t \end{aligned}$$

$$= \Big(V_t + \frac{1}{2}\sigma^2 S^2 V_{SS}\Big)dt = r\Pi_t dt \tag{2.1.5}$$

Therefore from Equation 2.1.5 we obtain the famous Black and Scholes formula

$$\begin{aligned}
\Big(V_t + \frac{1}{2}\sigma^2 S^2 V_{SS}\Big)dt &= r\Pi_t dt \\
V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} - r\Big(V(S,t) - \Delta S\Big) &= 0 \\
V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} - rV + r\frac{\partial V}{\partial S}S &= 0 \\
\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + r\frac{\partial V}{\partial S}S &= rV
\end{aligned} \tag{2.1.6}$$

The solution to this formula for the value of a European call, is

$$C^E(S,t) = N(d_1)S - N(d_2)Ke^{-r(T-t)} \tag{2.1.7}$$

$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\Big[ln\Big(\frac{S}{K}\Big) + \Big(r + \frac{\sigma^2}{2}\Big)(T-t)\Big] \tag{2.1.8}$$

$$d_2 = d_1 - \sigma\sqrt{T-t} \tag{2.1.9}$$

where $N(\cdot)$ is the cumulative distribution function of the standard normal distribution, and $T-t)$ is the time to maturity. Hence, the corresponding European put will be calculated using the call-put parity:

$$\begin{aligned}
P^E(S,t) &= Ke^{-r(T-t)} - S + C^E(S,t) \\
&= N(-d_2)Ke^{-r(T-t)} - N(-d_1)S
\end{aligned} \tag{2.1.10}$$

2.1.2 American Put Options

The value of an American put, $P^A(S,t)$, at time t is given by $P^A(S,t) \geq max(K-S,0)$, i.e. the payoff cannot go below its intrinsic value (see Figure 2.1 **(c)** demonstrating this via an example - i.e. could construct an arbitrage by buying the stock for S_t together with the put, executing the put at K and locking in a risk-free profit $K - S_t - P^A(S_t,t)$.

Intuitively, P^A will only be exercised if $S_t = S$ becomes too small, and there should exist an *optimal exercise boundary*, defined by some function $S^* = S^*(t)$, such that the put will be exercised at t if $S \leq S^*(t)$, and kept alive if $S > S^*(t)$. We should have that

$$S > S^*(t) \Leftrightarrow P^A(S,t) > max(K - S, 0) : \tag{2.1.11}$$

which shows that the exercise boundary is implicitly determined by the solution. In this so-called *continuation region* $(S,t) : S > S^*(t)$, the option is a traded instrument, and its price should satisfy the Black and Scholes PDE, by the hedging argument (which applies to any derivative as long as it is traded). Hence we should have:

$$\partial_t P^A + \frac{1}{2}\sigma^2 S^2 \partial_S^2 P^A + rS\partial_S P^A = rP^A, S > S^*(t), t < T. \tag{2.1.12}$$

If $S \leq S^*(t)$ the option is exercised and its value would is its exercise value, which gives us the following *boundary condition* for the problem (see Equation 2.1.12) since the put will surely be exercised before its exercise value drops to 0

$$P^A(S^*(t), t) = max(K - S^*(t), 0) = K - S^*(t), \tag{2.1.13}$$

The final boundary condition also needs to be satisfied

$$P^A(S,T) = max(K - S, 0) \tag{2.1.14}$$

Equations 2.1.12 and 2.1.13 provide two coupled equations for the two functions $S^*(t)$ and $P^A(S,t)$, however are not enough to provide a unique solution. A third equation is needed and is obtained from the fact that the function $S^* = S^*(t)$ needs to be chosen in such a way that $P^A(S,T)$ is as large as possible, thus leading to the *smooth pasting condition*:

$$\frac{\partial P^A}{\partial S}(S^*(t), t) = -1 \tag{2.1.15}$$

Equations 2.1.14 and 2.1.15 express that the function $\widehat{P}^A(S,t)$ is both continuous and has a continuous derivative w.r.t. S across the exercise boundary $S = S^*(t)$

$$\widehat{P}^A(S,t) = \begin{cases} P^A(S,t), & S > S^*(t) \\ K - S, & S \leq S^*(t) \end{cases}$$

An example *free boundary value problem* is thus presented via the system of Equations 2.1.12 - 2.1.15, where the optimal exercise boundary $S^*(t)$ needs to be identified as function of t as part of the problem, thus making it highly non-linear. This exercise boundary implies a trading strategy, whereby if the underlying stock crosses the continuation boundary the American option is exercised early at the boundary (see Figure 2.2).

2.2 Reinforcement Learning

The field of Reinforcement Learning (RL) was formed through the unification of research from psychologists (trial-and-error animal learning using reward and punishment [49]), computer scientists (artificial neural networks to generate control signals for driving motors [50]), control theorists (discrete stochastic version of optimal control problem [5]), and artificial intelligence (training computers to play a game of checkers [42]). It explicitly considers the whole learning problem by trying available actions in each state and learning a mapping from situations to actions which maximises a long term discounted reward in (possibly) stochastic environments. RL's two prominent features, the trial-and-error search and delayed reward, allow an agent (a decision maker) to learn an optimal policy in stochastic environments. The agent must try a variety of actions *and* progressively favour those that appear to be the best. For stochastic tasks each action must be tried many times in order to have a reliable estimate of its expected reward.

The core elements of RL are: **(1)** *Policy* - a mapping from perceived states of environment to actions, $\boldsymbol{\pi}(s,a)$; **(2)** *Reward Function* - a mapping from each perceived state-action pair $\mathcal{R}$; **(3)** *Value Function* - an expected accumulated reward for: being in state s *state-value function*, $V(s)$, or being in state s and choosing action a *state-action value function*, $Q(s,a)$; and **(4)** *Model* - a description of an environment, which can either be made available a priori (which can be solved with RL method of Dynamic Programming), or which can be stored from experience for updating a policy based on simulated possible future (e.g. DYNA-Q algorithm).

2.2.1 Classical Dynamic Programming

A famous control theorist, R. Bellman, broke a single multi-stage decision problem into a set of sub-problems [5] in the 1960s, to which he applied his *Principle of Optimality*[1] and defined the following concepts: *state* – the system's description at a given time point; *optimal policy* – most advantageous decision chain, *optimal value function* – a return obtained using an optimal policy. Bellman thus reduced the dimensionality of a control problem by choosing a decision optimally at the present time based on present state only and defined the Bellman Equation, which expresses a relationship between the value of a state and the values of its successor states:

$$v(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} \Big(cost(\mathbf{x}, \mathbf{u}) + v\big(next(\mathbf{x}, \mathbf{u})\big)\Big) \tag{2.2.1}$$

where $\mathbf{x}$ is a state of an environment, $\mathbf{u}$ is a control signal (an action), $v(\mathbf{x})$ is the value of the current state, $\boldsymbol{\pi}(\mathbf{x})$ a deterministic policy, $cost(\mathbf{x}, \mathbf{u})$ is the cost for applying control $\mathbf{u}$ in state $\mathbf{x}$, and $next(\mathbf{x}, \mathbf{u})$ is the state resulting from applying control $\mathbf{u}$ in state $\mathbf{x}$.

The method of *Dynamic Programming* can be applied to environments with known dynamics and works by calculating a globally optimal policy off-line. It iterates backwards in time from the final time-step where terminal cost is known, and solves the Bellman equation for all possible states at each time step where the action of the least cost is stored. This technique is known as *Value Iteration* and as *Backward Induction.*

2.2.2 Discrete Reinforcement Learning

2.2.2.1 RL Dynamic Programming

The classical optimal control dynamic programming theory laid foundations for the Reinforcement Learning dynamic programming algorithm, which also requires knowledge of system dynamics, yet works by iterating forward in time. If the system has a low dimensional state (avoiding curse of dimensionality) and a known transition probability function (known system dynamics), a backup for each state can be created at each time step and used to iteratively update the value function being estimated under the current policy $\boldsymbol{\pi}$.

[1]Principle of Optimality - an optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decisions

The Bellman Equation for the state value function of the classical dynamic programming (2.2.1) takes on a new form in the RL framework with an iterative update rule (*Policy Evaluation*)

$$V_{k+1}(s) = \sum_a \left[\boldsymbol{\pi}(s,a) \Big(\sum_{s'} \Big[\mathcal{T}^a_{ss'} \Big(\mathcal{R}^a_{ss'} + \gamma V_k(s') \Big) \Big] \Big) \right] = \sum_a \Big[\boldsymbol{\pi}(s,a) Q_k(s,a) \Big] \tag{2.2.2}$$

where the value of a state is related to the value of its successor state. Once the change in the value function upon two time steps becomes "small enough" (defined by a pre-determined tolerance), the evaluated policy is updated (*Policy Improvement*)

$$\boldsymbol{\pi}'(s,a) = \arg\max_a \sum_{s'} \Big[\mathcal{T}^a_{ss'} \Big(\mathcal{R}^a_{ss'} + \gamma V_k(s') \Big) \Big] = \arg\max_a Q_k(s,a) \tag{2.2.3}$$

The steps can then be repeated until there are no changes in the greedy policy and this iterative process is known as *Policy Iteration*. It is important to note that the DP algorithm uses *Bootstrapping* - it bases estimates on other estimates, which can slow down learning. This problem is avoided with the use of *Monte Carlo RL* methods, which is based on averaging complete returns, however it is only used for episodic tasks.

2.2.2.2 RL Monte Carlo

Monte Carlo (MC) an incremental model-free method which uses complete sample returns on an episode by episode bases since real life problems often have unknown transition probabilities and/or reward functions. MC collects triplets $\{s_t, a_t, r_{t+1}\}$ while moving forward in time until an absorbing state (i.e. terminal state) is reached thus forming an episode. The algorithm runs over many episodes and the policy is only updated at the end of each episode, repeating the process for a pre-determined number of iterations. The primary goal of MC is to estimate the state-action value function $Q^{\pi}(s,a)$ since without a model the state function $V(s)$ is not sufficient. The two assumptions of MC: **(1)** exploring starts, and **(2)** stochastic policies, are ensured via **(1)** using all of the collected returns for each (s,a) pair on an episode by episode basis (evaluation step), and **(2)** using a *soft policy* where $\boldsymbol{\pi}(s,a) > 0$, typically ϵ-greedy policies $\boldsymbol{\pi}(s,a) \geq \frac{\epsilon}{|\mathcal{A}(s)|}$. MC methods do not bootstrap, however one must wait until the end of the episode to update the value of each (s,a) pair which is time consuming. This problem can be alleviated by the use of Temporal Difference methods, which combine the DP idea of updating estimates based on other estimates with the MC idea of direct learning from raw experience.

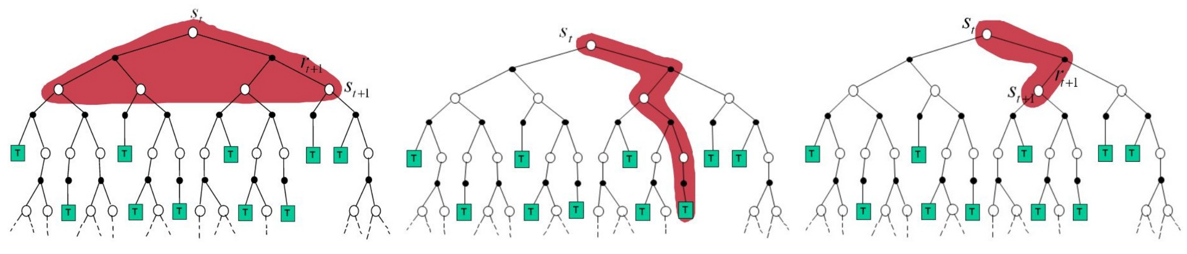
2.2.2.3 RL Temporal Difference

The temporal difference is a model-free method which makes value function updates on a step by step basis during the evolution of each episode

$$V_{k+1}(s) = V_k(s) + \alpha\delta = V_k(s) + \alpha\Big[r + \gamma V_k(s') - V_k(s)\Big] \tag{2.2.4}$$

where the *TD error* δ is used to evaluate whether action a selected in s should be strengthened ($\delta > 0$) or weakened ($\delta < 0$). The most prominent TD algorithms are

1. ϵ-greedy on-policy *SARSA*, $Q_{k+1}(s,a) = Q_k(s,a) + \alpha\Big[r + \gamma Q_k(s',a') - Q_k(s,a)\Big]$.
2. Off-policy[2] *Q-Learning*, $Q_{k+1}(s,a) = Q_k(s,a) + \alpha\Big[r + \gamma \max_a(Q_k(s',a)|s') - Q_k(s,a)\Big]$.
3. On-policy *Q-Learning*, $Q_{k+1}(s,a) = Q_k(s,a) + \alpha\Big[r + \gamma E(Q_k(s',a')|s') - Q_k(s,a)\Big]$.



(a) **Dynamic Programming** (b) **Monte Carlo** (c) **Temporal Difference**

Fig. 2.3 **The three most prominent methods in Reinforcement Learning.** White solid circle denotes a state s, black solid circle denotes a reward observed for transitioning from state s to s', green square denotes a terminal state T. **(a)** Dynamic Programming updates the value of state s_t by inspecting all possible next states at s_{t+1} and the rewards associated with transitioning to them, hence utilising the bootstrapping technique; **(b)** Monte Carlo methods update state s_t after a terminal state has been encountered (while starting from state s_t, collecting $\{s,a,r\}$ triplets, and ending in a terminal state - note actually all (s,a) pairs encountered along a path are updated at the end of the episode); and **(c)** Temporal Difference methods update the state s_t following just one time step, while observing only a partial realisation of the episode i.e. one $\{s,a,r,s'\}$ 4-tuple, thus utilising both bootstrapping and learning directly from experience.

The use of *eligibility traces* allows to build a family of methods which span MC at one end, and TD on the other, known as TD(λ), with two equivalent views *Forward* and *Backward View.*

The forward view TD(λ) consists of all n-step return backups, each weighted by λ^{n-1}

$$R_k^\lambda = \Big[(1-\lambda)\Sigma_{n=1}^{T-k-1}\lambda^{n-1}R_k^{(n)}\Big] + \Big[\lambda^{T-k-1}R_k\Big] \tag{2.2.5}$$

[2] *Off-policy* methods use a behaviour policy $\boldsymbol{\pi}'$ to generate behaviour which is different to the evaluated/improved policy $\boldsymbol{\pi}$.

where for TD(1), $\lambda = 1$, $R_k^\lambda = R_k$ which is constant α Monte Carlo, while for TD(0), $\lambda = 0$, $R_k^\lambda = R_k^{(1)}$ which is one step TD. The value function is then updated using

$$V_{k+1}(s) = V_k(s) + \alpha\Big[R_k^\lambda - V_k(s)\Big] \tag{2.2.6}$$

In the backward view TD(λ), the TD error signal triggers proportional updates to all recently visited states as signalled by non-zero eligibility traces, where the trace accumulates each time a state is visited and decays gradually otherwise.

$$e_k(s) = \begin{cases} \gamma\lambda e_{k-1}(s) & \text{if } s \neq s_k; \\ \gamma\lambda e_{k-1}(s) + 1 & \text{if } s = s_k. \end{cases}$$

The value function is then updated using

$$\begin{aligned} V_{k+1}(s) &= V_k(s) + \alpha e_k(s)\delta_k \ \ \forall\ s \in \mathcal{S} && (2.2.7) \\ &= V_k(s) + \alpha e_k(s)\Big[r_{k+1} + \gamma V_k(s_{k+1}) - V_k(s_k)\Big] && (2.2.8) \end{aligned}$$

where the most prominent algorithms are SARSA(λ), Q(λ), Watkin's Q(λ) and Peng's Q(λ).

The above methods form the core of RL methods which use discretised state, action and time spaces. Real life examples however are far more complicated and may involve high dimensional state and action spaces, necessitating decisions on the type of discretisation to be used. If coarse discretisation is used the control output may be non-smooth and result in a poor performance. When a fine discretisation is used the number of states and iteration steps become very large leading to high computational demands and many learning trials, or all together intractable. In order to keep the number of states manageable, an a priori knowledge of the variables is be used to form an elaborate partitioning. The above problems are addressed by combining RL with supervised machine learning techniques in order to work in continuous spaces thus leading to continuous RL. Other RL methods may be relevant for more complicated financial problems such as trying to avoid putting technical pressure on the market while working a trade (applicable to ϵ-greedy exploration). Alternatively, learning from incomplete episodes may be useful when risk managing highly illiquid products.

2.2.3 Continuous Reinforcement Learning

American style options present a problem of optimal stopping, which is naturally done via backward induction (dynamic programming, value iteration algorithm), however multiple sources of uncertainty (each represented as a state variable) would quickly make this problem intractable thus requiring function approximation. Thus such continuous and high-dimensional real life problems cannot make use of RL algorithms relying on tabular representation (i.e. where value functions/policies are represented as tables). The development of function approximation methods with parametrised representation of value functions allow working in continuous spaces within the RL framework [6, 43, 46]. Such algorithms operate on *continuous state spaces* e.g. extended Q-learning for discrete-time continuous state systems with linear dynamics [12], the least squares TD [13] and the least squares policy iteration TD [34]; *continuous time spaces* e.g. Advantage Updating which is an extended version of Q-learning for continuous time discrete state systems [3], the TD algorithm for semi-Markov decision problems [23] ; and *continuous action spaces* e.g. the TD algorithm with minimisation of TD error and use of eligibility traces for continuous time systems without a priori discretisation of time, state and action [21].

The application of parametrised RL to American Option pricing has seen the work on approximate value iteration for optimal stopping problems over infinite horizon (perpetual options) [47], however this an exponential growth in approximation error was encountered. This work was extended by using approximate value iteration over finite horizon with simulated trajectories [48] in order to make the error uniformly bounded, while similar methods (such as LSM [37]) also benefit from the same property. Other methods of approximate value iteration exist in option pricing literature, such as those based on Rust's method adapted to optimal stopping [41], for example, piece-wise constant approximations [44] and stochastic mesh [2, 18].

Value Function Representation Linear architectures offer the simplest value function approximation and have been widely studied due to their strong generalisation ability and their transparency when compared to black box methods such as neural networks. The true state-action value function $Q^\pi(s,a)$ is approximated with an estimate $\widehat{Q}^\pi(s,a;w)$, represented with adjustable weights w through a linear parametric[3] combination of basis functions, where for

[3] *Parametric approximators* use functions which are defined a priori (i.e. not data dependent) but the parameter of which are adjusted based on data. While *nonparametric approximators* use functions the form of

a state s and action a the value function is not stored but calculated on demand using weights

$$\widehat{Q}^{\pi}(s,a;w) = \sum_{i=1}^{M} \phi_i(s,a)w_i = \phi(s,a)^{\mathrm{T}}w \tag{2.2.9}$$

where $i = 1, \dots, M$ is the number of basis functions and $\phi(s,a) \in \mathbb{R}^M$ is the column vector of all the basis functions for a certain pair (s,a). The number of basis functions required to get a good approximation of Q^{π} is generally much smaller than the tabular representation($M \ll |\mathcal{S}||\mathcal{A}|$). The basis function are typically non-linear functions of the state and action, while their importance corresponds to the magnitude of each parameter. The entire state-action value function is represented by

$$\begin{aligned}\widehat{Q}^{\pi} &= \begin{bmatrix} 1 & \phi_2(s_1,a_1) & \phi_3(s_1,a_1) & \cdots & \phi_M(s_1,a_1) \\ 1 & \phi_2(s_2,a_2) & \phi_3(s_2,a_2) & \cdots & \phi_M(s_2,a_2) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_2(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) & \phi_3(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) & \cdots & \phi_M(s_{|\mathcal{S}|},a_{|\mathcal{A}|}) \end{bmatrix} w^{\mathrm{T}} \\ &= \begin{bmatrix} \phi(s_1,a_1)^{\mathrm{T}} \\ \cdots \\ \phi(s,a)^{\mathrm{T}} \\ \cdots \\ \phi(s_{|\mathcal{S}|},a_{|\mathcal{A}|})^{\mathrm{T}} \end{bmatrix} w^{\mathrm{T}} = \mathbf{\Phi} w^{\mathrm{T}} \end{aligned} \tag{2.2.10}$$

An appropriate choice of basis function for American option valuation is suggested in the Longstaff and Schwartz [37], the selection of which is crucial for as close functional approximation to the true state-action value function as possible.

No Volatility Basis The three prominent algorithms studied in this thesis (LSM, FQI and LSPI) use either partial or full set of basis functions described below. The basis functions approximate the continuation value, Q^{π}, of the American put options and use Laguerre polynomials which generalise over the underlying i.e. the stock price and time (in this thesis they are termed the 'No Volatility Basis'). The choice of these basis is guided by the facts that the function $\exp^{(-S/2)}$ goes to zero fast and that the optimal exercise boundary for an American

which and the parameters of which are are derived from the data (i.e. data dependent; this method usually requires more data).

put option is a monotonically increasing function of time [24].

$$\varphi_1 = 1 \tag{2.2.11}$$

$$\varphi_2(S) = \exp^{(-S/2)} \tag{2.2.12}$$

$$\varphi_3(S) = \exp^{(-S/2)}(1-S) \tag{2.2.13}$$

$$\varphi_4(S) = \exp^{(-S/2)}(1-2S+S^2/2) \tag{2.2.14}$$

$$\varphi_5(t) = \sin\Big((t\pi)/(2T) + \pi/2\Big) \tag{2.2.15}$$

$$\varphi_6(t) = \ln(T-t) \tag{2.2.16}$$

$$\varphi_7(t) = (t/T)^2 \tag{2.2.17}$$

where S is the standardised stock price with the starting stock price $S_1 = 1$ (see Chapter 3 for further details on standardisation of the stock trajectories, t is the current time step, and T is the total number of step up to and including expiration date).

2.3 American Option Pricing Algorithms

This section presents the mathematical derivations of the three state-of-the-art simulation based continuous RL algorithms LSM, FQI and LSPI, and prepares the reader for their application in an American Option pricing framework in Chapters 3 and 4. Each algorithm is applied to American options which can be only exercised at T discrete time points, thus technically making them *Bermudian* options. This is necessary due to computational requirements of discrete steps during iterative approximations, however in real life American options are continuously exercisable and the following algorithms approximate the American options in the limit of $dt \to 0$, where dt is the time step size between which the option is allowed to be exercised. The estimated $\widehat{Q}^{\boldsymbol{\pi}}(t,S)$ function implies the exercise boundary and thus exercise strategy, $\boldsymbol{\pi}(t,S)$

$$\boldsymbol{\pi}(t,S) = \begin{cases} 1, & \text{exercise} \quad\quad\quad g(S) > Q(t,S) \\ 0, & \text{keep the option} \quad g(S) \leq Q(t,S) \end{cases}$$

2.3.1 Least Squares Monte-Carlo (LSM) / Longstaff-Schwartz Method

The LSM algorithm uses a backward recursive dynamic programming approach [37] and is the simulation method of choice for pricing derivatives with American features in the financial community, which is used by Wall Street professionals on a daily basis. The algorithm uses least squares to estimate the conditional expected payoff from continuation, which is fundamentally different to other American option pricing algorithms, which estimate the transitional density function (early exercise boundary) instead of the conditional expectation [19, 29]. This method provides a pathways approximation to the optimal stopping rule that maximises the value of an American option and works in multi-factor path-dependent situations, unlike traditional finite difference techniques, which become intractable. The LSM pseudo-algorithm adapted to the American option pricing framework is given by Algorithm 1.

Algorithm 1 Longstaff-Schwartz Algorithm

```
1. Initialise
   1: $\widehat{\mathbf{w}}(T) \leftarrow \mathbf{0} \in \mathbb{R}^{M \times n}$
   2: $\widehat{Q}(T, S; \widehat{\mathbf{w}}) \leftarrow \mathbf{0}$
   3: $\widehat{C}(\widehat{\mathbf{w}}, S; t, T) \leftarrow \max(\mathbf{0}, K - S(T)) \in \mathbb{R}^n$
   4: $\boldsymbol{\tau}(T) \leftarrow T \in \mathbb{R}^n$
2. Calculate
   1: for each time step $t = T-1, \ldots, 1$ do
   2:     Perform Regression
   3:         $\mathbf{x}(t) \leftarrow S^j(t) \in \left(S^j(t) < K\right), \ \forall\, j = 1, \ldots, n$
   4:         $\mathbf{y}(t) \leftarrow e^{(-rdt(\boldsymbol{\tau}(\mathbf{x})-t))}\widehat{C}(\widehat{\mathbf{w}}, \mathbf{x}; t, T)$
   5:         $\widehat{\mathbf{w}}(t) \leftarrow \left(\left(\phi(t, \mathbf{x}_t)^{\mathrm{T}}\phi(t, \mathbf{x}_t)\right)^{-1}\phi(t, \mathbf{x}_t)^{\mathrm{T}}\right)\mathbf{y}_t$
   6:     Update Policy
   7:         $\widehat{Q}(t, \mathbf{x}; \widehat{\mathbf{w}}) \leftarrow \phi(t, \mathbf{x}_t)\widehat{\mathbf{w}}_t$
   8:         $\mathbf{z}(t) \leftarrow S^j(t) \in (K - \mathbf{x}_t > \widehat{Q}_t) \ \forall\, j = 1, \ldots, n$
   9:         $\widehat{C}(\widehat{\mathbf{w}}, \mathbf{z}; t, T) \leftarrow \mathbf{z}_t$
  10:         $\boldsymbol{\tau}(t) \leftarrow t \ \forall\ \mathbf{z}_t$
  11: end for
3. Estimate
   1: $\widehat{P}(\widehat{\mathbf{w}}) \leftarrow \frac{1}{n}\sum_{j=1}^{n} e^{(-rdt\tau^j)}\widehat{C}^j$
```

where $\widehat{\mathbf{w}}$ are the parameters of the regression calculated at each time step; n number of training stock paths; $\widehat{Q}(t, \mathbf{x}; \widehat{\mathbf{w}})$ estimated value of continuation; $\widehat{C}(\widehat{\mathbf{w}}, \mathbf{x}; t, T)$ option cash flow at time step t; $\boldsymbol{\tau}$ exercise time steps for each stock trajectory; $\mathbf{x}(t)$ stock prices at time t which is in-the-money w.r.t. strike K; $\mathbf{y}(t)$ stock prices at time $t+1$ (which corresponds to the in-the-money stock prices at time t) discounted to time t; $\mathbf{z}(t)$ stock prices at time t for trajectories chosen to be exercised; and $\widehat{P}(\widehat{\mathbf{w}})$ is the estimated price of the American put option.

The Valuation Algorithm The key to optimally exercising an American option is to correctly estimate the conditional expected value of continuation, $Q^{\pi}(s,a)$. Prior to expiration the optimal strategy is to compare the immediate exercise value with the continuation value and exercise if the immediate payoff is positive and greater or equal to the conditional expectation. The continuation value is estimated from data by regressing the subsequent realised cash flows from continuation on a set of basis functions for the paths where the option is in-the-money (i.e. this is where the exercise decision is relevant). The fitted value of this regression is an efficient unbiased estimate of the conditional expectation function and allows to accurately estimate the optimal stopping rule for the option. The process is repeated by going backwards in time from expiration date and identifying exercise decisions for each time step of each path. The option is valued by going forward along each path until the first stopping time occurs, discounting the resulting cash flow to time zero and taking the average across all paths. The LSM algorithm uses four basis functions (see Equations 2.2.11 - 2.2.14), and the authors find these sufficient to obtain effective convergence, above which further addition of basis functions does not have a significant effect on an increase in the estimated option value. The regression converges in mean square and in probability to Q as the number of in-the-money paths goes to infinity.

2.3.2 Fitted Q Iteration (FQI)

The least squares approach to the fitted Q-iteration shown in Equation 1.3.2 leads to an iterative Algorithm which is applied to the same trajectories over the algorithm's iterations. The authors show that rather than approximating the value function at each time period, a simultaneous approximation over all of the weights can be carried out using a special variant of approximate value iteration for finite horizon problems. The FQI algorithm uses basis functions that generalise over both state space and time, uses iterative calculations corresponding to taking expectation w.r.t empirical measure provided by simulations, and re-uses simulated trajectories upon each pass over the algorithm (which guarantees convergence of the parameter vector sequence) [48]. The iterative update rules to the matrices A and b are shown in the pseudocode Algorithm 2.

Algorithm 2 Fitted Q iteration Algorithm

```
1. Initialize
   1: $w^{(1)} = 0$
2. Calculate
   1: for each iteration $i = 1, \dots, n_{FQI}$ or until convergence do
   2:     $A^{(i)} = \Sigma_{j,t}\phi(t, S_t^j)\phi(t, S_t^j)^{\mathrm{T}}$
   3:     $b^{(i)} = \gamma\Sigma_{j,t}\phi(t, S_t^j)\max\Big(g(S_{t+1}^j), Q^{(i)}\Big(S_{t+1}^j, t+1\Big)\Big)$
   4:     $w^{(i+1)} = (A^{(i)})^{-1}b^{(i)}$
   5:     Converged if change in policy is small
   6: end for
3. Report the weights only if there has been convergence within the maximum allowed
   number of iterations.
```

2.3.3 Least Squares Policy Iteration (LSPI)

The *Least Squares Policy Iteration* (LSPI) learns a parametric state-action value function $\widehat{Q}^{\pi}(s, a; w)$ of a fixed policy π (*policy evaluation* step) and performs incremental policy updates (*policy improvement* step) i.e. forming an approximate policy iteration (actor-critic) algorithm suitable for control. It is an off-line (samples can be collected arbitrarily from any reasonable sampling distribution), off-policy (learning is separated from execution), model-free (does not require knowledge of the system dynamics) algorithm, which does not require an approximate policy representation (note a policy is represented by the basis functions ϕ and parameters, w). The approximation architecture is insensitive to relative magnitudes of basis functions, is linear in the parameters, w, and does not require tuning of any learning parameters (e.g. learning rate). Collected samples can be (re)used upon each iteration. The *Least Squares Temporal Difference* (LSTD) leans the state value function of a fixed policy (policy evaluation step) and thus requires knowledge of the system dynamics in order to be applicable to control.

The *Least Squares Temporal Difference Q* (LSTDQ) is the extended version of the LSTD, which learns the state-action value function of a fixed policy. It outputs the learnt parameters,

w and can take in any source of samples, which may be re-used upon each iteration of the LSPI algorithm if the set of samples adequately covers the state-action space.

Only the parameters need to be stored (note the policy is computed on demand and not stored anywhere), thus having much smaller computational requirements than the tabular (exact) representation. The approximation error can be reduced by using adequate choice of the basis functions. These need to appropriately cover the state-action space (the basis functions may be different upon different iterations of the LSPI algorithm). The sampling distribution is allowed great flexibility by the algorithm and samples collected using one policy can be used to evaluate the state-action value function of another policy.

2.3.3.1 Least Squares Fixed Point Solution

The state-action value function $\widehat{Q}^{\pi}(s,a)$ of any fixed policy π can be found exactly by solving a linear system of Bellman Equations

$$\begin{aligned}
Q^{\pi}(s,a) &= \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')R(s,a,s') + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \sum_{a'\in\mathcal{A}} \pi(a';s')Q^{\pi}(s',a') \\
&= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \sum_{a'\in\mathcal{A}} \pi(a';s')Q^{\pi}(s',a') \\
Q^{\pi} &= \mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_{\pi} Q^{\pi}
\end{aligned}$$

The state-action value function Q^{π} is also the fixed point of the Bellman operator, T_{π}, thus

$$\begin{aligned}
T_{\pi}Q^{\pi} &= Q^{\pi} \\
(T_{\pi}Q^{\pi})(s,a) &= \mathcal{R}(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \sum_{a'\in\mathcal{A}} \pi(a';s')Q^{\pi}(s',a') \\
&= \mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_{\pi} Q^{\pi} \qquad (2.3.1)
\end{aligned}$$

The weighted least squares fixed point approximation can be found by forcing the approximate state-action value function to be a fixed point under the Bellman operator

$$T_{\pi}\widehat{Q}^{\pi} \approx \widehat{Q}^{\pi} \qquad (2.3.2)$$

Even though $\widehat{Q}^\pi$ (fixed point) by definition lies in the space of approx. value functions (basis space), the $T_\pi \widehat{Q}^\pi$ may not and thus must be projected using *orthogonal projection* $(\Phi(\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T})$ (where the basis functions $m = 1, \ldots, M$ forming Φ are linearly independent). The approx. value function should be invariant under one application of the Bellman operator, followed by the orthogonal projection. Thus the problem of learning the approx. state-action value function, $\widehat{Q}^\pi$, is equivalent to learning parameters w^π of Equation 2.2.10 by solving $M \times M$ linear system

$$\begin{aligned}
\widehat{Q}^\pi &= \Phi(\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T}(T_\pi \widehat{Q}^\pi) \\
\widehat{Q}^\pi &= \Phi(\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T}(\mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \widehat{Q}^\pi) \\
\Phi w^\pi &= \Phi(\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T}(\mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi w^\pi) \\
\mathbf{0} &= \Phi\Big((\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T}(\mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi w^\pi) - w^\pi\Big) \\
\mathbf{0} &= (\Phi^\mathrm{T}\Phi)^{-1}\Phi^\mathrm{T}(\mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi w^\pi) - w^\pi \\
\Phi^\mathrm{T}\Phi w^\pi &= \Phi^\mathrm{T}(\mathcal{R} + \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi w^\pi) \\
\mathbf{0} &= \Phi^\mathrm{T}\mathcal{R} + \gamma \Phi^\mathrm{T}\boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi w^\pi - \Phi^\mathrm{T}\Phi w^\pi \\
\underbrace{\Phi^\mathrm{T}\mathcal{R}}_{M\times 1} &= \underbrace{\Phi^\mathrm{T}(\Phi - \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi)}_{M\times M} w^\pi \\
w^\pi &= (\Phi^\mathrm{T}(\Phi - \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi))^{-1}\Phi^\mathrm{T}\mathcal{R} \qquad (2.3.3) \\
w^\pi &= \mathbf{A}^{-1} b \qquad (2.3.4)
\end{aligned}$$

where $\mathbf{A} = \Phi^\mathrm{T}(\Phi - \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \Phi), \in \mathbb{R}^{M\times M}$ and $b = \Phi^\mathrm{T}\mathcal{R}, \in \mathbb{R}^{M\times 1}$, thus matrix $\mathbf{A}$ must be invertible ($|A| \neq 0$). The solution to this system cannot be found a priori, but incremental updating is possible using samples (s, a, r, s'), since Equation 2.3.3 is a sum of matrices/vectors:

$$\begin{aligned}
\mathbf{A} &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a)\Big(\phi(s,a) - \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s')\phi(s',\pi(s'))\Big)^\mathrm{T} \qquad (2.3.5) \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \left[\phi(s,a)\Big(\phi(s,a) - \gamma\phi(s',\pi(s'))\Big)^\mathrm{T}\right] \qquad (2.3.6) \\
b &= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \phi(s,a) \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') R(s,a,s') \qquad (2.3.7) \\
&= \sum_{s\in\mathcal{S}} \sum_{a\in\mathcal{A}} \sum_{s'\in\mathcal{S}} \mathcal{P}(s,a,s') \Big[\phi(s,a) R(s,a,s')\Big] \qquad (2.3.8)
\end{aligned}$$

In practice, samples $(s_t, a_t, r_t, s'_t) \in \mathcal{D}, t = 1, \ldots, T$, where $\mathcal{D}$ is a finite set of samples, obtained from the environment provide all the necessary information for individual terms of the summation at each time step, t, thus enabling incremental updates. An algorithm is iterated for $i = 1, \ldots, n_{LSPI}$ episodes, over the entire set of examples, $j = 1, \ldots, J$, or until convergence

$$\begin{aligned} \widetilde{\mathbf{A}}^{(i)}_{t+1} &= \widetilde{\mathbf{A}}^{(i)}_t + \phi(s_t, a_t)\Big(\phi(s_t, a_t) - \gamma\phi(s'_t, \pi(s'_t))\Big)^{\mathrm{T}} \\ \widetilde{b}^{(i)}_{t+1} &= \widetilde{b}^{(i)}_t + \phi(s_t, a_t) R(s_t, a_t, s'_t) \end{aligned} \tag{2.3.9}$$

The LSPI algorithm adapted to the American option pricing framework is given in Algorithm 3, with the necessary adjustments to the representation: the state s of the system is given by the stock price S^j_t of trajectory j, time t, and volatility σ_t (to be used for extended version of LSPI in Chapter 4), the action a is not part of the basis function, but is given by the indicator function $\mathbb{1}$ (0 - keep, 1 - exercise), where the indicator function $\mathbb{1}(a) = \begin{cases} 1, & \text{if a is true} \\ 0, & \text{if a is false} \end{cases}$.

Algorithm 3 Least Squares Policy Iteration (LSPI) Algorithm

```
1. Initialize
   1: $\widetilde{w}^{(0)} \leftarrow \mathbf{0}$
2. Calculate
   1: for $i = 1, \ldots, N_{LSPI}$ iterations or until convergence do
   2:     $\widetilde{\mathbf{A}}^{(0)} \leftarrow \mathbf{0}$
   3:     $\widetilde{b}^{(0)} \leftarrow \mathbf{0}$
      (a)     for each stock trajectory $j = 1, \ldots, J$ do
           i.         for each time step $t = 1, \ldots, T$ do
          ii.             $\widetilde{\mathbf{A}}^{(i)}_{t+1} = \widetilde{\mathbf{A}}^{(i)}_t + \phi(t, S^j_t, \sigma^j_t)\Big[\phi(t, S^j_t, \sigma^j_t) - \gamma\mathbb{1}[\pi^{(i)}(t, S^j_t, \sigma^j_t) = 0]\Big]\phi(t+1, S^j_{t+1}, \sigma^j_{t+1})^{\mathrm{T}}$
         iii.             $\widetilde{b}^{(i)}_{t+1} = \widetilde{b}^{(i)}_t + \phi(t, S^j_t, \sigma^j_t)\mathbb{1}[\pi^{(i)}(t, S^j_t, \sigma^j_t) = 1]g(S^j_{t+1})$
          iv.         end for
      (b)     end for
   4:         $\widetilde{w}^{(i+1)} = (\widetilde{\mathbf{A}}^{(i)})^{-1}b^{(i)}$
   5:         Converged if change in policy is small
   6:     end for
3. Report the weights only if there has been convergence within the maximum allowed
   number of iterations.
```

Chapter 3

Exercise Policies for American Options

"An approximate answer to the right problem is worth a good deal more than an exact answer to an approximate problem."

— J. Tukey

This chapter attempts to re-create the results found in Schuurmans et. al. [35], which applies the state-of-the-art simulation based Reinforcement Learning algorithms (LSM, FQI, LSPI) to the problem of pricing American put options, specifically for pricing the Dow Jones Option Index. We apply the LSM, FQI and LSPI algorithms in the American option framework and develop the required code, which is built upon in (see Chapter 4).

3.1 Introduction

The Monte Carlo method is a powerful, flexible and tractable technique for pricing complex derivatives with American features, which present an extra optionality of early exercise thus making it important to learn optimal exercise policies. The Longstaff and Schwartz show a way of extending Monte Carlo framework to American options (LSM algorithm) [37], while Tsitsiklis and Van Roy (FQI algorithm) [48] and Li, Szepesvari and Schuurmans (LSPI algorithm) [35] show how to solve an American option pricing problem by finding optimal exercise policy in a Reinforcement Learning framework. The three algorithms are empirically compared by Schuurmans et. al. [35] when applied to pricing American put options on the Dow Jones Index. We develop the necessary code and attempt to re-create their results in order to: first confirm their findings and second be build upon the code and suggest improvements.

Table 3.1 **List of Parameters from studied paper by Schuurmans et. al.**

Name	**Symbol**	**Value**	**SI Unit**
Strike	K	S_0	[USD]
Risk Free Rate	r	0.03	[a.u.]
Time Step	Δt	$\frac{1}{252}$	[year]
Discount Factor	γ	$\exp^{(-\frac{r}{252})}$	[a.u.]
Tenor	T	[63,126,252]	[day]
Max FQI Iter.	$\max_{FQI}$	15	[a.u.]
Min FQI Iter.	$\min_{FQI}$	3	[a.u.]
Max LSPI Iter.	$\max_{LSPI}$	15	[a.u.]
Min LSPI Iter.	$\min_{LSPI}$	3	[a.u.]
Train traj. lengths (synthetic)	N_{synth}^{train}	5000	[a.u.]
Test traj. lengths (synthetic)	N_{synth}^{test}	1000	[a.u.]
Train traj. lengths (real)	N_{real}^{train}	[600, 500, 500]	[a.u.]
Test traj. lengths (real)	N_{real}^{test}	[500, 450, 250]	[a.u.]

3.2 Methods

The three state-of-the-art American option pricing algorithms, LSM, FQI, LSPI, studied by Schuurmans et. al. [35], are applied to empirically compare the learnt policies on the Dow Jones Option Index (DJI). The authors obtain 30 underlying stock price data (see Appendix A for the exact list of companies) from Wharton Research Data Services (WRDS, period Jan 2002 - Dec 2006, ordered chronologically). The study assumes that the stocks are non-dividend paying, thus we use adjusted daily closing prices to account for any dividends. In order to re-create the results we tried to obtain the same data set, but could not due to WRDS not making their data open source, therefore we used Google Financial Services database instead [27].

The quarterly, semi-annual and annual maturity terms are studied (i.e. Tenor = (63, 126 and 252 business days respectively)), where a full trading year is assumed to contain 252 business days. The full list of parameters used is given in Table 3.1, where the studied at-the-money strike is chosen to be the initial stock price; the risk free rate is a constant; the time step is equal to one trading day; and the discount factor corresponds to the daily interest rate. It is more interesting to study at-the-money strikes because Gamma is greatest at the money i.e. option price changes quickest here as the underlying price varies ($\Gamma = \frac{\partial^2 V}{\partial S^2}$ rate of change of delta[1] with respect to the underlying price.).

[1]$\Delta = \frac{\partial V}{\partial S}$ rate of change of the option value with respect to the price of the underlying

3.2.1 Algorithms

The FQI and LSPI algorithms typically iterate until the difference in between in the weight vector becomes small. However when these algorithms are applied in the option pricing framework, the iteration is stopped when the difference between two successive policies is sufficiently small, or when it has run for $\max_{FQI}, \max_{LSPI}$ number of iterations (the authors report that the algorithms converge within 4 - 5 iterations). The LSM algorithm used by the authors takes a single pass over all of the stock prices starting at the last time step (expiration) and works backwards calculating estimated optimal exercise policy at each time step for each trajectory. The basis functions used by the LSM algorithm are constant over time and are given by Equations 2.2.11 - 2.2.14, resulting in weights which are time step dependent $\mathbf{w}(t) \in \mathbb{R}^{n_\phi \times T}$. While both the FQI and LSPI algorithms use time dependent and independent basis functions given by Equations 2.2.11 - 2.2.17, resulting in a single vector of weights, $w(t) = w$.

3.2.2 Data

The authors use both real and synthetic data for learning exercise policies by simulation. For real data, two possible approaches exist: **(1)** Learn exercise policies on real data, and test the found policies on real data; **(2)** Learn exercise policies on synthetic data, and test the found policies on real data; For simulated data, the exercise policies are learnt and tested on synthetic independent identically distributed (i.i.d.) data generated using either a Geometric Brownian Motion (GBM) model or Generalized Autoregressive Conditionally Heteroskedastic (GARCH) model, with parameters extracted from real data. The authors estimate the parameters of each model (GBM, GARCH(1,1)) from real data, where for quarterly, semi-annual and annual maturities the first 662, 625 and 751 stock prices of the series are used. The synthetic data is produced so that there are 50000 paths generated for training, and 10000 are generated for testing. All of the testing sample paths do not overlap and are separate from training data.

3.2.2.1 Geometric Brownian Motion Model

The Geometric Brownian Motion assumes that the underlying stock price follows a process described by Equation 1.1.1. In practice it is more accurate to simulate $\ln S_t$, thus using Taylor expansion and Ito's lemma

$$\begin{aligned}
f(S_t) &= ln(S_t) \\
df(S_t) &= dln(S_t) \\
f(S_t + dS_t) - f(S_t) &= \frac{dln(S_t)}{dS_t} dS_t + \frac{1}{2}\frac{d^2 ln(S_t)}{dS_t^2}(dS_t)^2 \\
\frac{dln(S_t)}{dS_t} &= \frac{1}{S_t} \\
\frac{d^2 ln(S_t)}{dS_t^2} &= -\frac{1}{S_t^2} \\
(dS_t)^2 &= (\mu S_t dt + \sigma S_t d\mathbb{W}_t)^2 \\
&= \mu^2 S_t^2 (dt)^2 + 2\mu\sigma S_t^2 dt d\mathbb{W}_t + \sigma^2 S_t^2 (d\mathbb{W}_t)^2 \\
&= \sigma^2 S_t^2 dt \\
dln(S_t) &= \frac{1}{S_t} dS_t + \frac{1}{2}\Big(-\frac{1}{S_t^2}\Big)\sigma^2 S_t^2 dt \\
&= \frac{1}{S_t}(\mu S_t dt + \sigma S_t d\mathbb{W}_t) - \frac{1}{2}\sigma^2 dt \qquad (3.2.1)
\end{aligned}$$

Thus a discretised version of the stock price model can be obtained by integrating both sides

$$\begin{aligned}
\int_0^t 1 \, dln(S_s) &= \int_0^t \Big(\mu - \frac{1}{2}\sigma^2\Big)\, dS_s + \int_0^t \sigma d\mathbb{W}_s \\
\Big[ln(S_s)\Big]_0^t &= \Big(\mu - \frac{1}{2}\sigma^2\Big)\Big[S_s\Big]_0^t + \sigma\Big[d\mathbb{W}_s\Big]_0^t \\
ln(S_t) - ln(S_0) &= \Big(\mu - \frac{1}{2}\sigma^2\Big)t + \sigma\mathbb{W}_t \\
ln(S_t) &= ln(S_0) + \Big(\mu - \frac{1}{2}\sigma^2\Big)t + \sigma\mathbb{W}_t \\
e^{ln(S_t)} &= e^{ln(S_0) + \Big(\mu - \frac{1}{2}\sigma^2\Big)t + \sigma\mathbb{W}_t} \\
S_t &= S_0 e^{\Big[\Big(\mu - \frac{1}{2}\sigma^2\Big)t + \sigma\mathbb{W}_t\Big]} \\
S_{t+1} &= S_t e^{\Big[\Big(\mu - \frac{1}{2}\sigma^2\Big)\Delta t + \sigma\sqrt{\Delta t}\epsilon\Big]} \qquad (3.2.2)
\end{aligned}$$

where Δt is a small time step, and $\epsilon \sim \mathcal{N}(0,1)$. The Geometric Brownian Motion Equation 3.2.2 is used to simulate the underlying stock price movement. The policies found with synthetic paths obtained with the GBM process are termed LSM$_{gbm}$ FQI$_{gbm}$ and LSPI$_{gbm}$. The authors use *antithetic variance reduction* for each simulated trajectory starting from the time step at half way point $t_{T/2}$ up to the end of tenor time point T. Note some discrepancies between the GBM model obtained with our code and the GBM model obtained by the authors [35] exist, because we used log of individual stock returns, while the authors used maximum likelihood estimation (unclear exactly how from the paper / no parameter estimation code available). Our calculations of the GBM volatility, σ, use the following code

```
% calculating log of return log(price_t+1 / price_t)
Real_Log_Return     = log(Underlying(2:T_Train)) - log(Underlying(1:(T_Train-1)));
Mu(stock,1)         = mean(Real_Log_Return) / Settings.dt;      % drift
GBM_Vol(ten,stock)  = std(Real_Log_Return) / sqrt(Settings.dt); % volatility
```

3.2.2.2 Generalized Autoregressive Conditionally Heteroskedastic Model

The Generalized Autoregressive Conditionally Heteroskedastic (GARCH) model does not assume constant volatility of the stock price (unlike GBM), since in reality volatility may be stochastic. The authors use a stochastic volatility GARCH(1,1) model, where the volatility is assumed to evolve according to

$$\sigma_t^2 = w + \alpha u_{t-1}^2 + \beta \sigma_{t-1}^2 \tag{3.2.3}$$

where $u_t = ln\left(\frac{S_t}{S_{t-1}}\right)$; α, β are weights for the $u_{t-1}^2, \sigma_{t-1}^2$ respectively[2]; $w = (1 - \alpha - \beta)\sigma_L$ is a constant related to long term average volatility σ_L. The discretised version used for creating the synthetic data is

$$S_{t+1} = S_t e^{\left[\left(\mu - \frac{\sigma_t^2}{2}\right)\Delta t + \sigma_t \sqrt{\Delta t}\epsilon\right]} \tag{3.2.4}$$

We use Glosten, Jagannathan, and Runkle [26] GARCH model, which includes P lagged conditional variances, Q lagged squared innovations, and Q leverage terms. The policies found with synthetic paths obtained using the GARCH model are termed LSM$_{garch}$ FQI$_{garch}$ and LSPI$_{garch}$. Note our model for obtaining the GARCH parameters is different to the inbuilt function (garchfit) used by the authors [35], since we believe it to be more accurate, therefore some discrepancies in the exact models obtained from the real data exist between our code an the authors code.

[2]The stability is GARCH(1,1) model requires that $\alpha + \beta < 1$.

3.2.2.3 Real Data

Since there is only one realised trajectory per stock price time series for each company, there is a scarcity of real data available for training and testing the algorithms. The authors come up with a *Windowing Technique* for slicing data into required maturity durations, while re-using the same single stock trajectory [48]. For each company, for quarterly, semi-annual, annual maturity terms, 600, 500, 500 training paths are obtained, each with duration 63, 126, 252 price respectively. The first path is the first duration days of stock prices, the second path is the duration starting from one day ahead, and so on moving through the original trajectory until the required number of paths is obtained [35]. The obtained trajectories are used to train the algorithms and after the exercise policies are found, their performances are compared on the test paths, which are obtained as follows: For each company, for the quarterly, semi-annual and annual maturity terms, 500, 450, 250 testing paths are obtained, each with duration = 63, 126, 252 prices. The first path is the last duration days of stock prices, then the window is moved one day back and the second path is obtained, and so on. Figure 3.1 gives an overview of the Windowing Technique and uses an example trajectory of historical Intel stock price in order to explain how both the training and testing paths are obtained. The policies obtained from training the three algorithms on sample paths from real data are called LSM$_{data}$, FQI$_{data}$ and LSPI$_{data}$, respectively. See Table 3.1 for the full list of parameters used in the paper [35].

3.2.2.4 Standardisation

In order to train and test the data on comparable scale, the trajectories are scaled so that each trajectory starts from the initial stock price of the whole series, S_0. For synthetic data, the trajectories are artificially created in such a way so that they start from S_0. For real data however, the $N \times T$ matrix obtained (where N is the number of trajectories and T is the option Tenor) with the windowing technique results in stock prices which do not all start from S_0. Hence a scaling factor is found for each trajectory, j, by dividing the first price of the whole series by the first price of each trajectory $\alpha^j = \frac{S_0}{S_1^j}$, where $j = 1, \ldots, N$ is the trajectory number. Each trajectory is scaled by multiplying all of the time steps by the respective scaling factor. The stock prices are further scaled inside the basis functions by dividing entire trajectories by S_0, in order for each trajectory to start from 1. This standardisation process is described using an example of a historical Apple stock price for the period of Jan 2002 - Dec 2006 and demonstrate how real data would be split using a windowed technique and standardised so that each trajectory starts from $S_1 = 1$. A tenor of 63 days is used as an example, which requires 600 training and 500 testing paths [35]. Please see Appendix Figure A.1 shows the resulting train and test trajectories for real data. Note: in our implementation of the code the standardisation is done in a slightly different way, which is equivalent to the method above. We follow the same

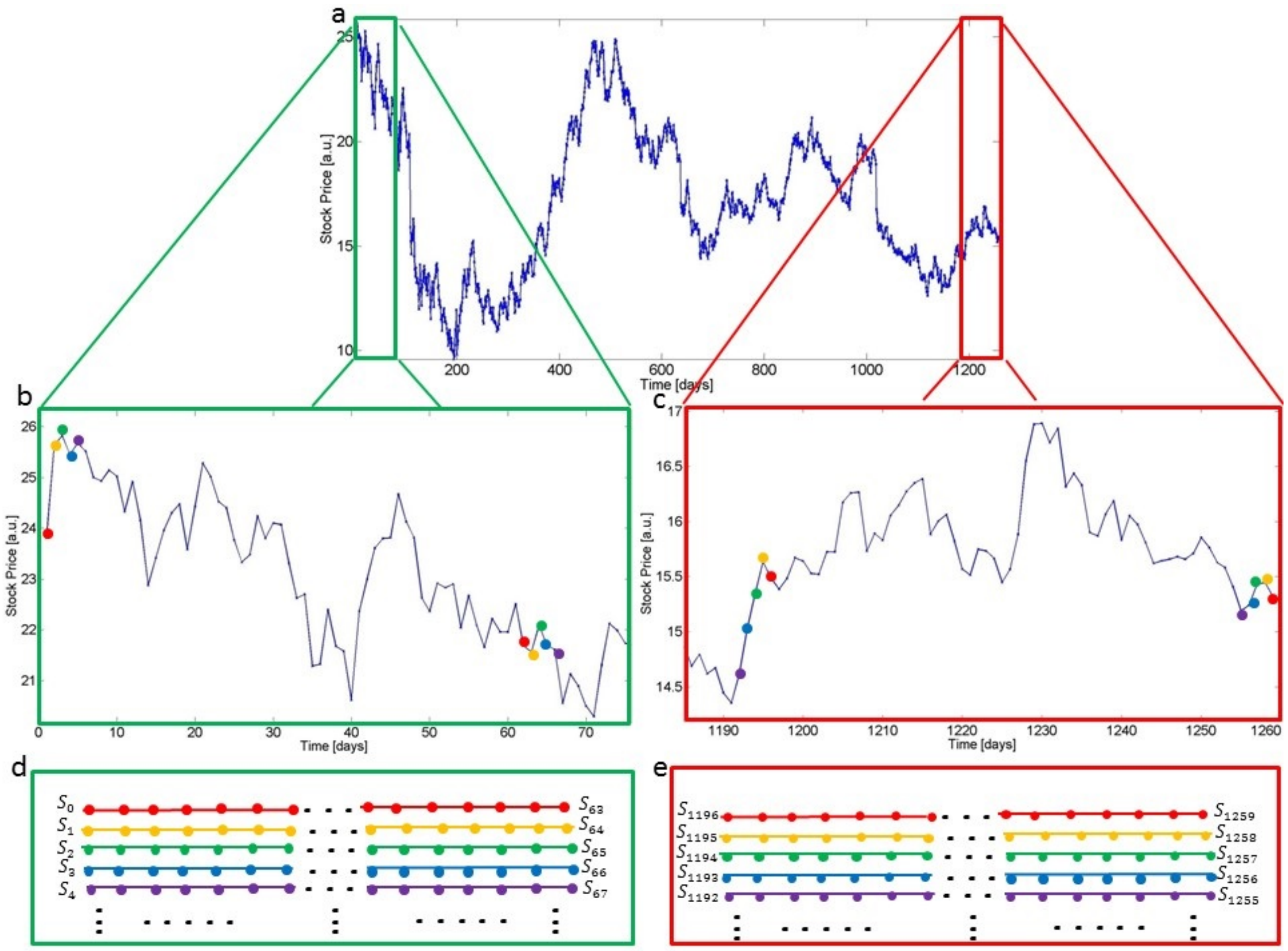


Fig. 3.1 **Moving Window Technique for Real Data.** This figure explains the segmentation of a single stock trajectory into a number of paths each of a given duration. An example 5 paths with tenor 63 days is given for training and testing data. **(a)** Historical adjusted daily prices of Intel stock from Jan 2002 - Dec 2006, where the first, green square, (last, red square) prices are used to create the training (testing) trajectories respectively, see zoomed in boxes below. **(b)** The training trajectories are obtained starting from the first price S_0 of the stock in the series and taking a window of 63 days duration (marked by red solid circles). The window is then moved 1 day ahead to obtain the second trajectory (yellow solid circle), and so on until the required number of trajectories is obtained (in this example, 5). **(c)** The testing trajectories are obtained using the windowing technique but starting from the last stock price of the series, with the first trajectory encompassing the last 63 data points (red solid circles). The window is moved one day back and another trajectory is obtained (yellow solid circles), and so on until 5 trajectories are obtained. **(d),(e)** The training/testing trajectories are stacked into a matrix of the size $n \times 63$. The first training trajectory starts with stock price at S_0 and ends at S_{63}, while the second trajectory starts from S_1 and ends at S_{64} and so on. The first testing trajectory starts with S_{1196} and ends with S_{1259}, while the second trajectory starts with S_{1195} and ends with S_{1258} and so on.

steps as per Figure A.1 in order to obtain a matrix of windowed trajectories. Following this we divide each trajectory by its respective stock price at time $t = 1$ (i.e. the first stock price of the trajectory, S_1^j, for $j = 1, \dots, N$). This results in standardised trajectories starting from $S_1^j = 1$ and we store all of the S_1^j in order to un-standardise back into original space to present results.

3.2.3 Boundaries

We have done work on checking for the best way of identifying the continuation boundaries using learnt policies. The authors Schuurmans et. al. [35] do not state how the optimal exercise boundaries were found, and their code did not contain any syntax relating to this.

3.2.3.1 Root Searching Algorithm - Standard Bisection Method

Our method of choice is the *Bisection* method, due to its simplicity and the convergence guarantees. The algorithm is used to numerically solve $f(x) = 0$ equation for a real variable x, where $f(\cdot)$ is a monotonic function defined on an interval $[a, b]$ (which *brackets the root*) and where $f(a)$ and $f(b)$ have opposite signs. The continuous function f must have at least one root in the interval (a, b), as guaranteed by the *Intermediate Value Theorem.* The algorithm's

Algorithm 4 Bisection Pseudo-code, $f(x) = x^2$

```
$n \leftarrow$ Number to be square-rooted
$n_{low} \leftarrow a; n_{high} \leftarrow b$
$tol \leftarrow 10^{-3}; diff \leftarrow (n_{high} - n_{low})$
while $abs(diff) > 0$ do
    $mid = 0.5(n_{low} + n_{high})$
    $f_{mid} = mid^2$
    $diff = f_{mid} - n$
    if $diff > 0$ then
        $n_{high} = mid$
    else
        $n_{low} = mid$
    end if
end while
```

pseudo-code is given for an example function $f(x) = x^2$, see Algorithm 4. At each iteration the algorithm updates it's upper / lower interval limits depending on the size of the difference between $f(\frac{a+b}{2})$ and the value to be square-rooted, n, i.e. want to find value of x which gives $f(x) = 0$ i.e. find the root of the function.

In order to establish the exercise boundaries of American put options, the function $f(S_t) = S_t - K + \widehat{Q}_t$ needs to be solved for the value of the stock, S^*, which makes $f(S^*) = 0$. An American option is exercised if $K - S_t > \widehat{Q}_t$, thus the continuation region is $-K + S_t + \widehat{Q}_t < 0$, where $\widehat{Q}_t$ is found via linear approximation using Equation 2.2.9. The Bisection root searching algorithm for finding the root of the function $f(S_t)$ is given in Algorithm 5 (note both S and K are standardised, as per Section 3.2.2.4). The Bisection algorithm may not find a correct

Algorithm 5 Bisection Pseudo-code, $f(S_t) = S_t - K + \widehat{Q}_t$

```
Continuation Boundary, $C \leftarrow \mathbf{0} \in \mathbb{R}^T$
$tol \leftarrow 10^{-3}$;
for $t = 1, \ldots, T$ time steps do
    $n_{low} \leftarrow 0; n_{high} \leftarrow K$
    $tol \leftarrow 10^{-12}; diff \leftarrow (n_{high} - n_{low})$
    while $abs(diff) > tol$ do
        $midS = 0.5(n_{low} + n_{high})$
        $f_{midS} = midS - K + \phi(midS)^{\mathrm{T}} w$
        if $f_{midS} > 0$ then
            $n_{high} = midS$
        else
            $n_{low} = midS$
        end if
        $diff = n_{high} - n_{low}$
    end while
    $C_t = midS$
end for
```

approximation to the root if the function is discontinuous or non-smooth / non-monotonic. We have found the continuation boundaries to abide the monotonicity requirement on certain occasions (e.g. see Chapter 4 for improved LSM results), however sometimes this was not the case and the boundaries were found to be discontinuous. In order to address this problem an alternative algorithm to Bisection was used, which we term Exhaustive Bisection.

3.2.3.2 Root Searching Algorithm - Exhaustive Bisection Method

We have developed a method for finding the root of a function, which exhaustively searches for the stock price S^* that results in $f(S^*) = 0$. This alternative was necessary since most exercise boundaries found by the discovered policies were non-monotonically increasing (i.e. showing occasional decrease), thus leading to discontinuous boundaries found by the Bisection algorithm.

The *Exhaustive Bisection* method iterates through all possible stock prices on the interval $[a, b]$, where $a = 0, b = K$ (note that due to standardisation of the stock and strike prices the interval becomes $[0, 1]$, thus making this exhaustive search bounded by a reasonable number of steps, we take the step size of 0.0001). This brackets the root, since $b = K$ is the highest possible and $a = 0$ is the lowest possible early exercise point for an American put option with strike $K = 1$. We have found that the continuation boundaries found by our Exhaustive Bisection algorithm

Algorithm 6 Exhaustive Bisection Pseudo-code, $f(S_t) = S_t - K + \widehat{Q}_t$

```
Continuation Boundary, $C \leftarrow \mathbf{0} \in \mathbb{R}^T$
$tol \leftarrow 10^{-3}$;
for $t = 1, \dots, T$ time steps do
    $midS \leftarrow 0; f_{midS} \leftarrow 0$
    for $midS = 1, \dots, 0$ stock values (step size = 0.0001) do
        $f_{midS} = midS - K + \phi(midS)^{\mathrm{T}} w$
        if $f_{midS} < tol$ then
            $C_t = midS$
            break
        end if
    end for
end for
```

are consistently better than those found by the Standard Bisection method. This is due to the fact that this alternative version can deal with non-monotonic functions.

3.3 Results

The authors report results of separate stock options, where an average over the testing paths of the 30 stocks is taken. Following this, the average payoffs of the DJI stock option prices are averaged again to find the final DJI option estimate. This is performed for quarterly, semi-annual and annual tenors using exercise policies found with LSM, FQI and LSPI algorithms. The authors neither provide the European counterpart option price nor do they provide finite difference equivalent in order to compare the found results. Since this section focuses on reproducing the results from Schuurmans et. al. [35], we will not provide these here either, however will focus on a more detailed examination of the results in Chapter 4 (and propose original contributions).

3.3.1 Original Paper's Results

The results reported in the Schuurmans et. al. [35] favour: 1st FQI, and 2nd LSPI method, which originated in the optimization community, while not only was the LSM estimated DJI option price below its European counterpart, the LSM exercise boundaries where non-monotonic with sharp large fluctuations (see Fig 1. [35]). These results are reportedly (highly) significant when tested by a paired two-tailed t-test (the differences between the performances of LSPI and FQI are not significant). The authors consider full-episode batch versions of each algorithm and report that FQI and LSPI algorithms converge within 5 iterations, while LSM is used as a single pass algorithm over the data via backward recursion (a non-iterative regression method).

3.3.1.1 Test Data: Real

The approximated optimal exercise policies for individual stocks result in estimated av. DJI option prices, see Table 3.2, where the algorithms used synthetic training data (GBM, GARCH results), while and real widowed data (DATA results).

Table 3.2 **Original Paper's Results: Average DJI option payoffs, real data.**

	LSPI			FQI			LSM		
Tenor	GBM	GARCH	DATA	GBM	GARCH	DATA	GBM	GARCH	DATA
63	1.310	1.333	1.339	1.321	1.341	1.331	0.573	0.572	0.719
126	1.681	1.663	1.739	1.718	1.749	1.797	0.693	0.687	0.887
252	1.599	1.496	1.677	1.832	1.797	2.015	0.717	0.685	0.860

3.3.1.2 Test Data: Synthetic

The authors also report the DJI option results for exercise policies learnt and tested on synthetic data (GBM trained, GBM tested and GARCH trained, GARCH tested), see Table 3.3. The use of synthetic data for testing results in consistently higher DJI option price.

Table 3.3 **Original Paper's Results: Average DJI option payoffs, synthetic data.**

	GBM			GARCH		
Tenor	LSPI	FQI	LSM	LSPI	FQI	LSM
63	2.071	2.054	2.044	1.889	1.866	0.785
126	2.771	2.758	2.742	2.546	2.530	0.997
252	3.615	3.645	3.580	3.286	3.311	1.241

3.3.1.3 Key Observations of Original Paper and Code

We have carefully written our code in order to re-create all of the results from Schuurmans et. al. paper [35]. The authors have kindly provided their versions of the LSM, FQI and LSPI algorithms (as tested on GBM synthetic data only, without specifying parameter σ used by the model). Some key discrepancies w.r.t. the write up and the provided code must be discussed:

- The Section 4.1 LSPI derivation of Schuurmans et. al. [35] state that the matrix $b^{(i)} = \gamma \sum_{t,j} \left[\phi(S_t^j, t) \mathbb{1}_{\pi^{(i)}(S_{t+1}^{(j)})=1} g(S^(j)_{t+1}) \right]$. However we believe this to be incorrect as $b^{(i)}$ should not be discounted by γ. The underlying mathematics can be confirmed from the original LSPI algorithm derivations of Lagoudakis et. al. [34] p.1122 Equations (unlabelled) and from our derivations provided by Equations 2.3.2 - 2.3.9. This is further supported by authors own code, which does NOT discount $b^{(i)}$ by γ. Therefore our code does not contain the discount factor when calculating the LSPI algorithm's matrix $b^{(i)}$.

- Similarly the author's FQI code does not contain the discount factor for matrix $b^{(i)}$, however the matrix is discounted in the formulae of Schuurmans et. al. [35] paper (see Section 4.2, p. 354). The discount factor should not be present, as supported by the derivations in the original FQI algorithm (see Equation 7, p. 702 of Tsitsiklis et. al. [48]). Therefore our code does not discount matrix $b^{(i)}$ of the FQI algorithm.

- The authors used *Antithetic Variance Reduction* technique for generating Geometric Brownian Motion simulated trajectories, as discovered by closely examining the provided code. This variance reduction technique is implemented by reducing the second half of the variance for each generated trajectory, i.e. for time steps $(T/2 - T)$ the following GBM model is used $S_{t+1} = S_t e^{\left[\left(\mu - \frac{1}{2}\sigma^2\right)\Delta t - \sigma\sqrt{\Delta t}\epsilon\right]}$ (note that the $\sigma\sqrt{\Delta t}\epsilon$ term is taken away rather than added as per (3.2.2)). This is not reported in the paper of Schuurmans et. al. [35] and therefore was incorporated into our code following the original code's steps.

- The exact real data used in Schuurmans et. al. [35] was obtained from WRDS, however this is not an open source database and we did not have access. Thus the historical adjusted daily closes of the 30 stocks comprising the DJI were obtained from Google Financial Services, which also constitute to discrepancies in the results.

- The authors report poor payoffs and non-smooth continuation boundaries for the policies found with LSM algorithm. This is surprising since LSM is an algorithm widely used in the industry for pricing derivatives with American option features. We have closely investigated the synthax for the LSM provided by the authors [35] and have found key differences, which will be addressed in Chapter 4 of this thesis.

3.4 Re-Created Results

We have carefully constructed our code which compares the three algorithms for American put option pricing on DJI option, period Jan 2002 -Dec 2006. The discrepancies found when comparing our code to the theory/code provided by Schuurmans et. al. [35] (see Section 3.3.1.3) were incorporated in our code, unless otherwise stated. The DJI option payoffs are found by averaging the estimated av. payoffs of individual stock indices (see Tables 3.4, 3.4).

Table 3.4 **Average estimated DJI option payoffs, real data.**

	LSPI			FQI			LSM		
Tenor	GBM	GARCH	DATA	GBM	GARCH	DATA	GBM	GARCH	DATA
63	1.103	1.118	1.123	1.123	1.074	1.082	1.118	1.132	1.096
126	1.344	1.329	1.367	1.396	1.457	1.422	1.371	1.417	1.236
252	1.354	1.332	1.565	1.681	1.796	1.687	1.504	1.571	1.617

Table 3.5 **Average estimated DJI option payoffs, synthetic data.**

	GBM			GARCH		
Tenor	LSPI	FQI	LSM	LSPI	FQI	LSM
63	2.889	2.137	2.290	2.046	1.896	2.055
126	3.149	2.902	3.150	2.769	2.469	2.780
252	4.159	3.776	4.194	3.529	3.214	3.550

Overall we can see that for real data (Table 3.4) the largest payoffs are for the FQI algorithm, followed by the LSM, which is followed by the LSPI. For synthetic data LSM seems to perform the best, followed by LSPI and FQI. We must be careful about these result, since the LSM produces choppy boundaries (see Figure 3.4,3.5), which ultimately determine the trading strategy. The is in agreement with the original authors report and thus we will seek to improve the LSM algorithm's results in the next Chapter. The individual results within each algorithm are reported below.

3.4.1 LSPI Results

The results found by the LSPI Algorithm 3 match closely those reported by the authors [35]:

Real Data The Table 3.4 shows that the policies found for data trained and tested with real windowed trajectories give higher payoffs than policies trained with synthetic data while tested on real data. This is consistent with the original results of Table 3.2, thus supporting the reproducibility of author's findings. The LSPI$_{gbm}$ is consistently higher than the LSPI$_{garch}$ payoffs, also supporting the original results.

Synthetic Data The Table 3.5 shows that the policies found for data trained and tested on synthetic data give higher payoffs than policies tested on real data. This is consistent with the original results of Table 3.3, thus supporting the reproducibility of author's findings. The GBM ang GARCH payoffs found by our LSPI algorithm give higher payoffs than original results of Table 3.3. This difference is most likely due to the way the GBM and GARCH parameters are calculated (this is different and improved way, as demonstrated by the results) - these key differences are outlined in Section 3.3.1.3.

Boundaries The boundaries found by the LSPI algorithm (see Figures 3.2 and 3.3) are smooth, but do display some non-monotonicity (similar effect can be found in the results of the original paper [35], thus again we were able to reproduce the results correctly).

Interestingly we only used 5000 training (vs 50000 training paths, original) and 1000 testing (vs 10000 training paths, original) synthetic trajectories, which concludes that the differences in the parameter simulation of the two processes contributed to the major difference, as well as possibly the difference in the data set used (Google, us vs. WRDS, original).

3.4.2 LSM Results

The LSM algorithm written by us iteratively approximates the continuation value, Q^{π} in order to price American put options. However the results obtained from our version of the code are discussed in Chapter 4, since these relate to our original contribution to the Schuurmans et. al. paper [35]. The results reported in this section are obtained using the LSM code provided by authors, which uses a single pass over the data using backward recursion (starting at expiry

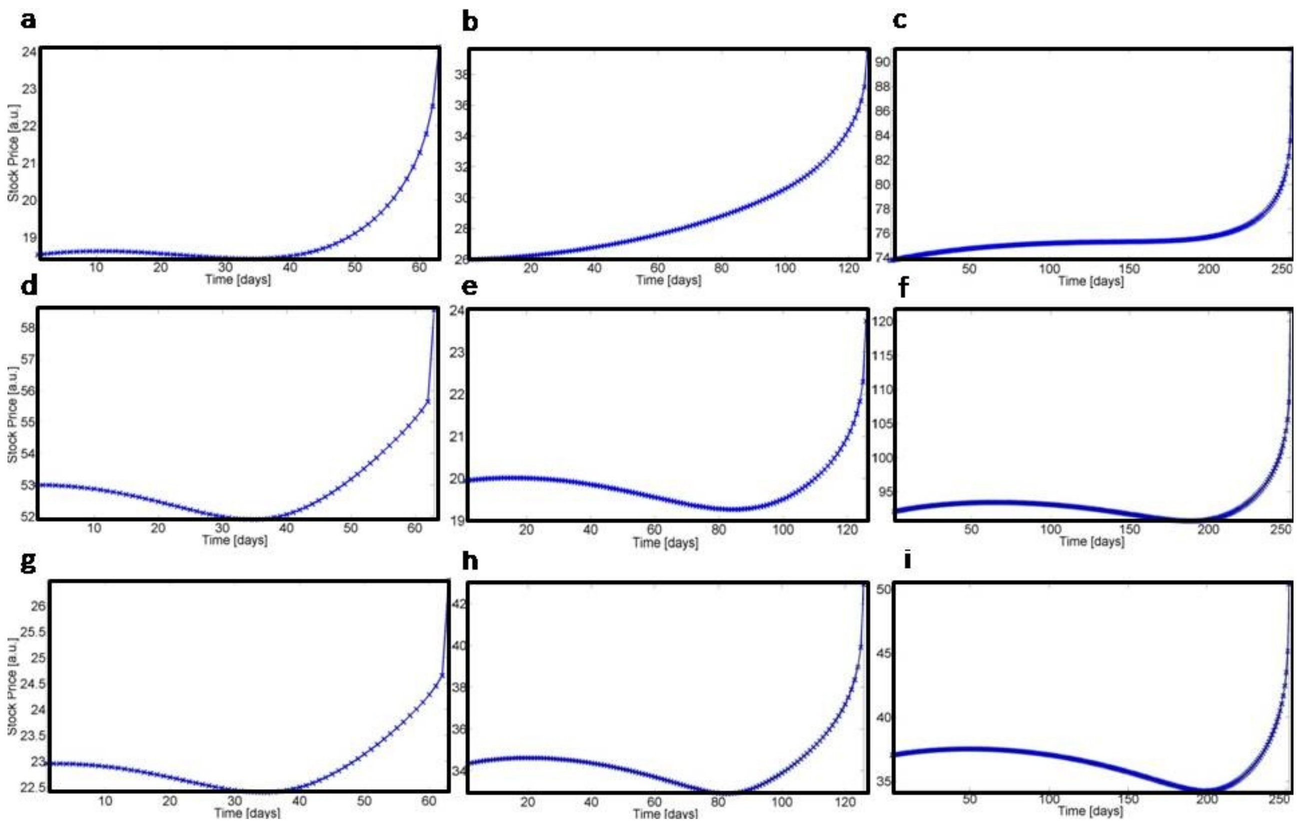


Fig. 3.2 **LSPI boundaries for individual stocks, real data.** The boundaries are shown for a number of different stocks, tenors, and for policies trained on a variety of data (real/synthetic) and tested on real data. This figure demonstrates that the LSPI algorithm has learnt smooth continuation boundaries. **(a)** Company: JPM. Policy was trained with real data, and tested on real data (Tenor = 63 days). **(b)** Company: XOM. Policy was trained with real data, and tested on real data (Tenor = 126 days). **(c)** Company: GS. Policy was trained with real data, and tested on real data (Tenor = 252 days). **(d)** Company: MMM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 63 days). **(e)** Company: KO. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 126 days). **(f)** Company: IBM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 252 days). **(g)** Company: MCD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 63 days). **(h)** Company: DD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 126 days). **(i)** Company: HD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 252 days).

and iterating backwards). We have incorporated the code provided into our framework and generated results shown in Tables 3.4,3.5.

The DJI option payoffs estimated by LSM for the different maturities and data combinations show slightly higher estimates than the LSPI algorithm. This is contrary to what the authors find in their research, which shows (see Tables 3.2 and 3.3) that the LSM algorithm is very poor and in some cases predicts DJI option price which is lower than its European counterpart. We attribute the difference to previously mentioned discrepancies (different dataset; different GBM and GARCH parameter estimation).

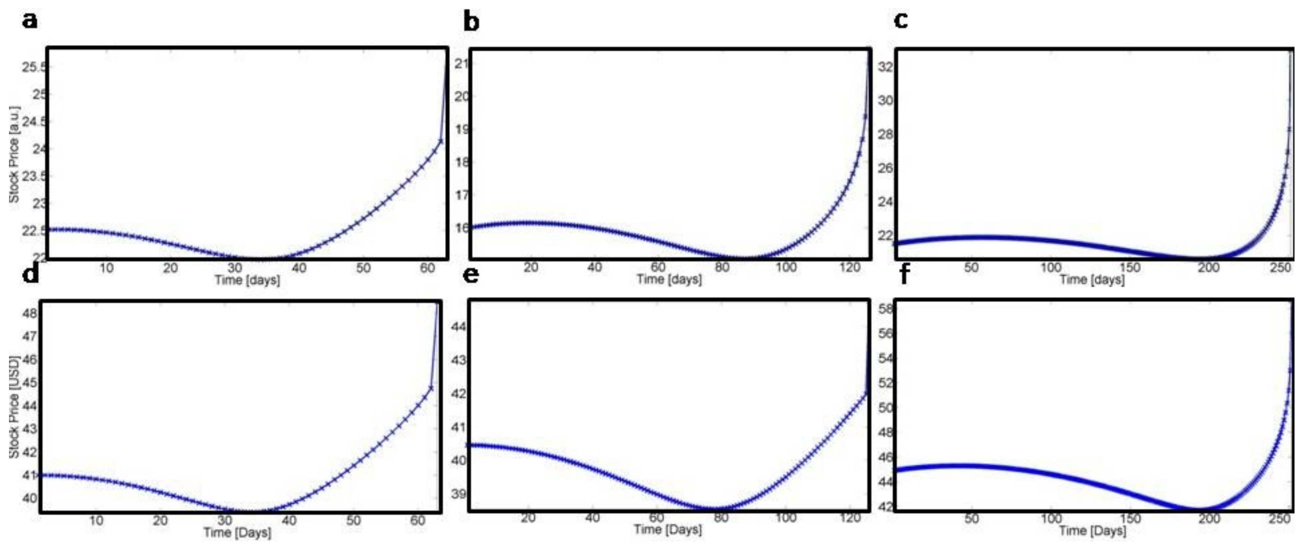


Fig. 3.3 **LSPI boundaries for individual stocks, synthetic data.** This figure gives example boundaries found by the LSM algorithm for individual stocks of different tenors and for policies trained and tested on synthetic data only. **(a)** Company: CAT. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 63 days). **(b)** Company: DIS. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 126 days). **(c)** Company: INTC. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 252 days). **(d)** Company: VZ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 63 days). **(e)** Company: CVX. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 126 days). **(f)** Company: JNJ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 252 days).

However the estimated exercise boundaries found by the algorithm are in agreement with [35] and are non-smooth, choppy (see Figures 3.4 and 3.5) do not appear to be correct, since these are expected to be smooth and monotonically increasing. The results are consistent with those reported by Schuurmans et. al. (see Figure 1, p. 358 [35], which shows a choppy Intel stock exercise boundary). This is unexpected since the LSM algorithm is a standard method of choice in the industry for pricing derivatives with American features and the results should support this default algorithm of choice.

3.4.3 FQI Results

The results reported in this section are obtained using the FQI code written by us, as provided by the pseudo-code of Algorithm 2.

Real Data The DJI payoffs found by FQI policies show that the GARCH model outperforms the GBM and DATA policies for longer maturities, while for tenor of 63 days the results are

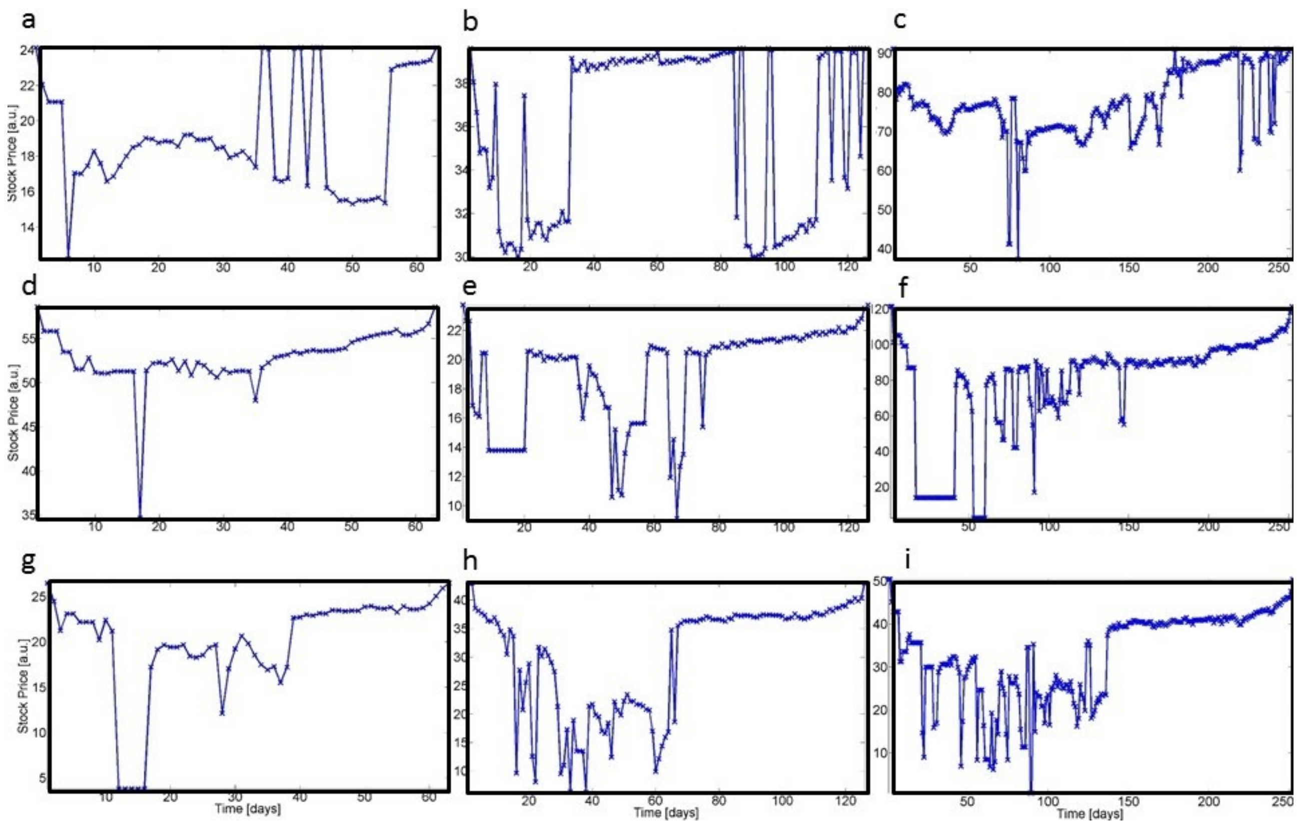


Fig. 3.4 **LSM boundaries for individual stocks, real data.** This figure demonstrates the LSM algorithm's boundaries as learnt using the original LSM code provided by authors Schuurmans et. al. [35]. The boundaries are shown for a number of different stocks, tenors, and for policies trained on a variety of data (real/synthetic) and tested on real data. **(a)** Company: JPM. Policy was trained with real data, and tested on real data (Tenor = 63 days). **(b)** Company: XOM. Policy was trained with real data, and tested on real data (Tenor = 126 days). **(c)** Company: GS. Policy was trained with real data, and tested on real data (Tenor = 252 days). **(d)** Company: MMM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 63 days). **(e)** Company: KO. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 126 days). **(f)** Company: IBM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 252 days). **(g)** Company: MCD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 63 days). **(h)** Company: DD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 126 days). **(i)** Company: HD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 252 days).

very similar payoffs (see Table 3.4). This is in contrast to original findings, see Table 3.2 [35] who report largest payoffs for real data policy.

Synthetic Data The FQI policies trained and tested on synthetic data show that on average the GBM does better than the GARCH trained/tested policies (see Table 3.5). This is consistent with the original findings, see Table 3.3 [35], thus showing reproducibility.

Boundaries From the two Figures (3.6 and 3.7) show that the boundaries are smooth but non-monotonic, this is consistent with the original paper's findings.

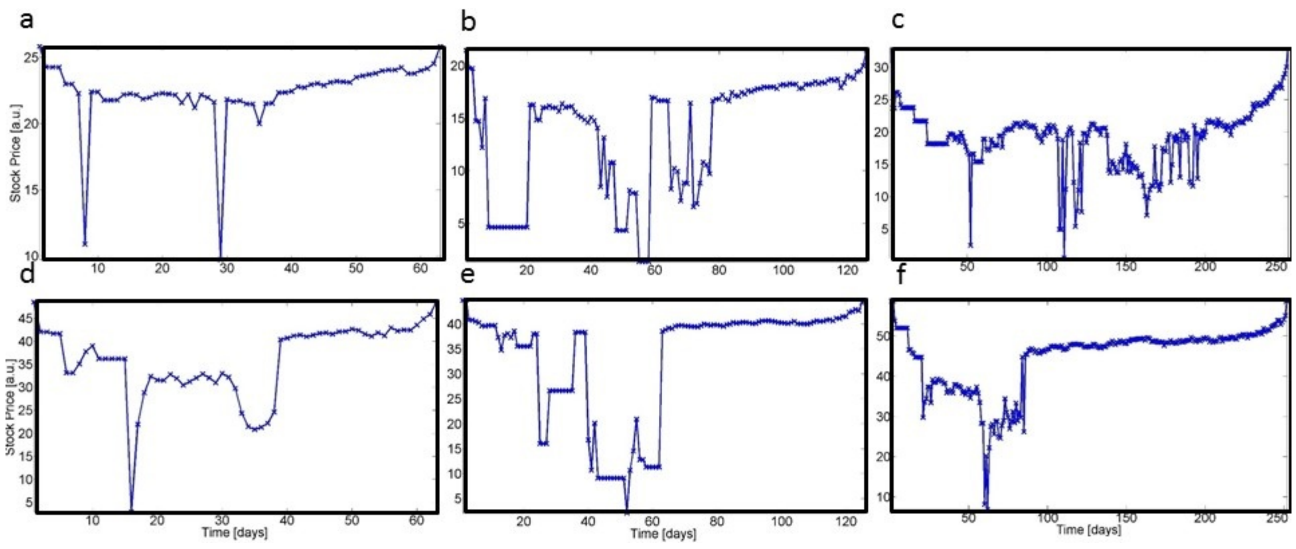


Fig. 3.5 **LSM boundaries for individual stocks, synthetic data.** This figure gives example boundaries found by the LSM algorithm for individual stocks of different tenors and for policies trained and tested on synthetic data only. **(a)** Company: CAT. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 63 days). **(b)** Company: DIS. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 126 days). **(c)** Company: INTC. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 252 days). **(d)** Company: VZ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 63 days). **(e)** Company: CVX. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 126 days). **(f)** Company: JNJ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 252 days).

We re-create the exercise boundaries (see Figure 3.8) obtained by the three algorithms on Intel stock, learnt and tested on the windowed real data (see Figure 1, p. 358, Schuurmans et. al. [35]). These show that the LSM boundary is poor (in agreement with original), and that the FQI boundary comes higher than the LSPI boundary (opposite of originally reported). This difference could be due to the different data sources used. No explanation was given by the authors which boundary was expected to be higher, thus we are not sure if this result is correct.

3.5 Discussion

In this Chapter we replicated the results from Schuurmans et. al. [35] paper for pricing American put options using the three state-of-the-art simulation based Reinforcement Learning algorithms: Longstaff-Schwartz (LSM), Fitted-Q Iteration (FQI), and Least Squares Policy Iteration (LSPI).

The original results of Schuurmans et. al. [35] show choppy LSM boundaries and we are in agreement with, however we disagree on the payoffs where in some cases LSM comes out as the algorithm with highest payoff. This result is suspicious due to the non-smooth boundaries and

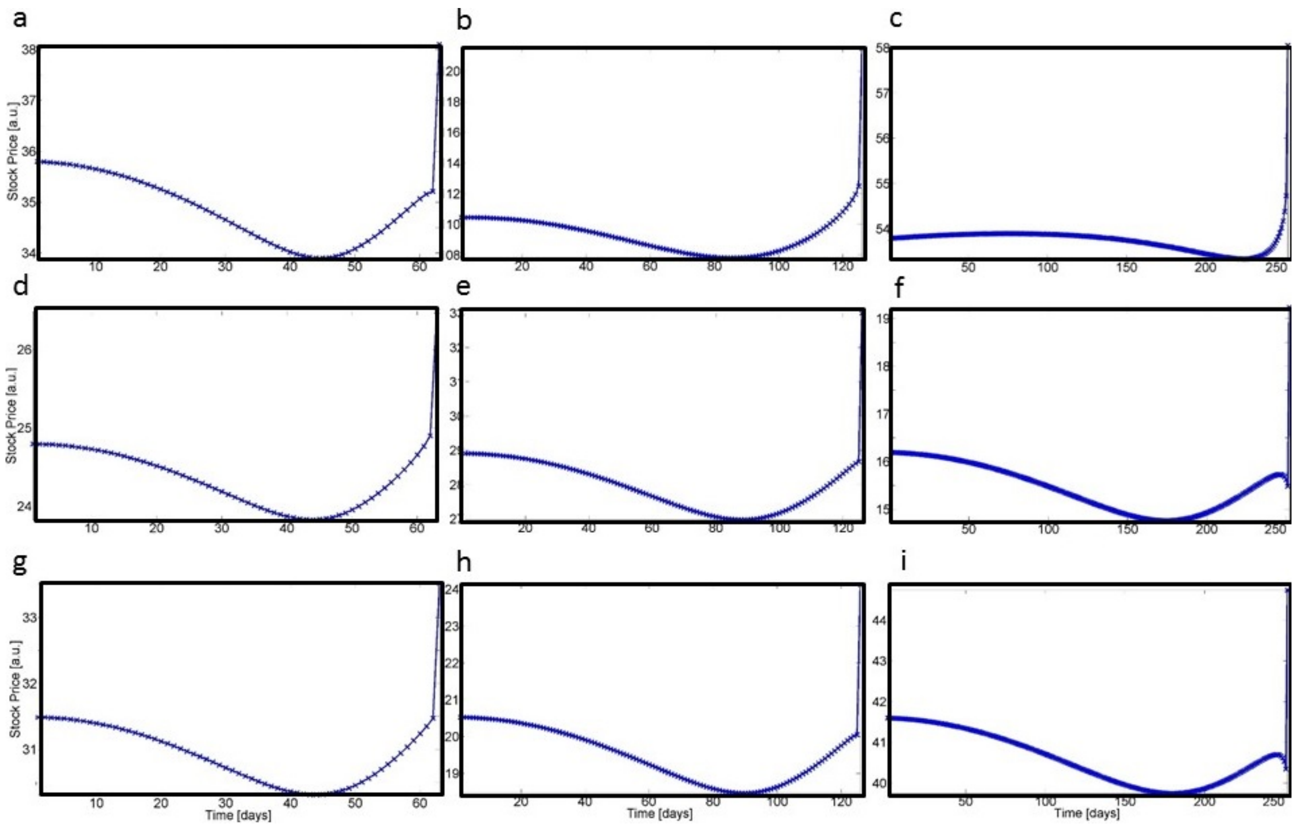


Fig. 3.6 **FQI boundaries for individual stocks, real data.** The boundaries are shown for a number of different stocks, tenors, and for policies trained on a variety of data (real/synthetic) and tested on real data. **(a)** Company: BA. Policy was trained with real data, and tested on real data (Tenor = 63 days). **(b)** Company: IBM. Policy was trained with real data, and tested on real data (Tenor = 126 days). **(c)** Company: WLT. Policy was trained with real data, and tested on real data (Tenor = 252 days). **(d)** Company: MCD. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 63 days). **(e)** Company: INTC. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 126 days). **(f)** Company: CSCO. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 252 days). **(g)** Company: MSFT. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 63 days). **(h)** Company: JPM. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 126 days). **(i)** Company: CVX. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 252 days).

will be investigated further in the next chapter. It must be noted that the higher payoffs were achieved while using the exact same algorithm provided by the authors Schuurmans et. al. [35], thus any discrepancies must be due to the data source used and/or the way GBM/GARCH model's parameters were simulated (our way is different to author's way, see (3.3.1.3)).

The FQI generally showed better results than the LSPI algorithm, which is the opposite to the original results, however both algorithms reported smooth, slightly non-monotonic boundaries, which is consistent with original results. We have tested the author's and our's LSPI algorithms on identical data sets with a fixed random number generator seed and obtained exactly the same results. Thus the algorithms are absolutely equivalent and the main difference in payoffs

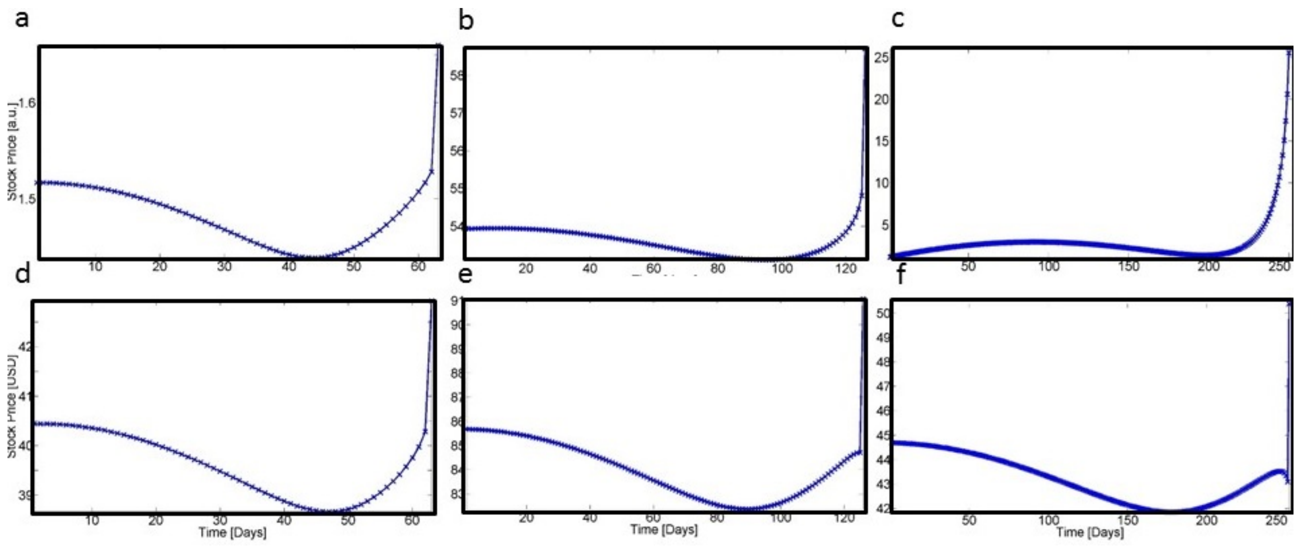


Fig. 3.7 **FQI boundaries for individual stocks, synthetic data.** This figure gives example boundaries found by the FQI algorithm for individual stocks of different tenors and for policies trained and tested on synthetic data only. **(a)** Company: AAPL. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 63 days). **(b)** Company: JNJ. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 126 days). **(c)** Company: UTX. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 252 days). **(d)** Company: DD. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 63 days). **(e)** Company: GS. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 126 days). **(f)** Company: HD. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 252 days).

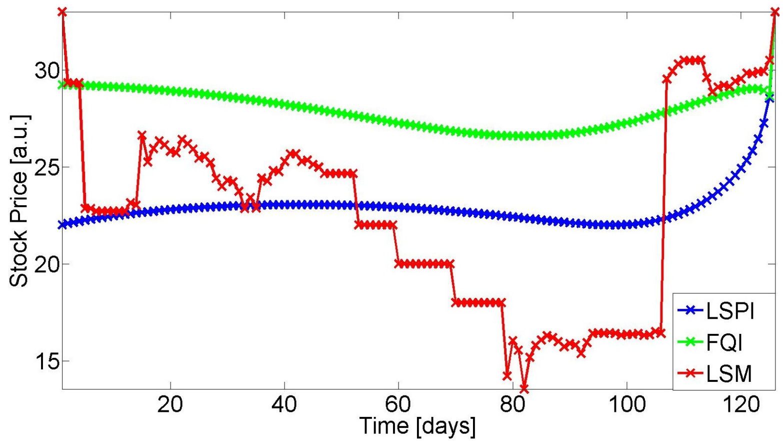


Fig. 3.8 **Exercise Boundaries, Intel stock option - real data.** The LSM boundary (red line) does not represent a good exercise policy because it is non-smooth. However, LSPI and FQI boundaries (blue line and red line respectively) show good exercise policies.

between us and them must lie in the data source and/or GBM/GARCH parameter estimation. Overall we are happy to report that the results are broadly reproducible, that our code manages to find smooth boundaries for LSPI and FQI, while supporting the poor results of the LSM boundaries. Thus we have successfully built the framework for the three algorithms and are highly keen to see if the LSM results can be improved. This is the focus of our next Chapter.

Chapter 4

Extended Research: RL for American Option Pricing

"If you torture the data enough, nature will always confess."

— R. Coase

4.1 Introduction

The previous Chapters have focused strongly on building the framework for testing the three prominent state-of-the-art simulation based continuous Reinforcement Learning algorithms for pricing American put options: the Longstaff-Schwartz (LSM), Fitted-Q Iteration (FQI), and Least Squares Policy Iteration (LSPI) algorithms.

We have found that the three algorithms are able to price the option correctly, producing smooth boundaries with the exception of the LSM algorithm. In this Chapter we will focus on making any possible improvements to the way the three algorithms are tested and/or implemented.

4.2 Methods

We have three key ideas we would like to test in order to better compare the strength of the three American option pricing algorithms. **(1.)** First we would like to add more basis functions, which we suggest should depend on stock's volatility and examine the associated regression weights. **(2.)** We propose a slightly different version of the LSM algorithm to that provided by in the author's code (i.e. and to the one used in Chapter 3 - resulting in poor boundaries). **(3.)** Potential inefficiency in LSPI algorithm changed, so that the option is only exercised if it is in-the-money during training.

4.2.1 Additional Basis Functions - Volatility Dependent

It is well known that the option values are sensitive to volatility in the underlying asset, and that this volatility changes over time. In fact when volatility goes to infinity the current price of the stock becomes irrelevant, in which case the larger the volatility, the more important it becomes than the other factors. We considered this point in our study by introducing several basis functions depending on the interplay of the time remaining and the realized volatility. For each path of S_t, the corresponding values of volatility σ_t are estimated off S_t using the rolling window approach.

4.2.1.1 Simple Volatility Basis

This thesis proposes to use basis functions which depend on exponentially weighted realized historical volatility.

$$\zeta_1(t, S, \sigma) = \sigma(T-t)^{1/2}S \tag{4.2.1}$$

$$\zeta_2(t, S, \sigma) = \sigma^2(T-t)S \tag{4.2.2}$$

where σ is calculated off a rolling window of 21 business days. The exponential decay parameter is chosen such that the weight of the most remote shock in the rolling window is 50% of the weight of the most recent shock.

4.2.1.2 Complex Volatility Basis

This thesis proposes to use complex volatility basis functions which are based on the Taylor expansion of the Black-Scholes formula. These basis are the re-scaled approximations of the first-order and second-order terms in the expansion for the time t price of the European call option with strike 1.

$$\zeta_{11}(t,S,\sigma) = \sigma(T-t)^{1/2} S N^{'}(t,S,\sigma) \tag{4.2.3}$$

$$\zeta_{12}(t,S,\sigma) = \sigma^2(T-t) S N^{'}(t,S,\sigma) \tag{4.2.4}$$

where $N^{'}(t,S,\sigma) = \exp^{(-d_1^2/2)}$ and $d_1 = \frac{\ln(S)+(r+\sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}}$.

The inspiration behind the complex volatility basis comes from the Black-Scholes formula for the time t price of the European call option

$$C(t,S,\sigma) = S\Phi(d_1) - K\exp^{-r(T-t)}\Phi(d_2) \tag{4.2.5}$$

where the stock price has been standardised (see Chapter 3 for further details), $\Phi(\cdot)$ is the cumulative distribution function of standard normal distribution and where $d_2 = \frac{\ln(S)+(r-\sigma^2/2)(T-t)}{\sigma(T-t)^{1/2}}$.

A natural choice would be to decompose $C(t,S,\sigma)$ into the Taylor series with respect to volatility σ and collect the first several terms as basis functions. This Taylor series is $C(t,S,\sigma) = C(t,S,0) + \frac{dC(t,S,\sigma)}{d\sigma}\sigma + \frac{d^2C(t,S,\sigma)}{d\sigma^2}\frac{\sigma^2}{2} + \mathrm{O}\Big(\sigma^2\Big)$ However the evaluation of derivatives $\frac{dC(t,S,\sigma)}{d\sigma}$ and $\frac{d^2C(t,S,\sigma)}{d\sigma^2}$ would increase the computational demand and including these as basis functions would defeat the purpose of value function approximation. Thus contrary to intuition, this thesis proposes to only approximate the most important part of $C(t,S,\sigma)$ with simple expressions. Due to a free choice of the basis functions, we choose a non-exact Taylor decomposition since the Black-Scholes formula serves only as a guide to non-linear terms with “high potential”.

We choose to decompose the part of $C(t,S,\sigma)$ which captures most of the interplay between volatility σ and underlying price S (the term $C_1(t,S,\sigma) = S\Phi(d_1)$). The second order Taylor expansion of the term $C_1(t,S,\sigma)$ becomes

$$C_1(t,S,\sigma) = C_1(t,S,0) + \frac{dC_1(t,S,\sigma)}{d\sigma}\sigma + \frac{d^2C_1(t,S,\sigma)}{d\sigma^2}\frac{\sigma^2}{2} + \mathrm{O}\Big(\sigma^2\Big) \tag{4.2.6}$$

Next we "pull out" the most important parts out of terms $\frac{dC_1(t,S,\sigma)}{d\sigma}\sigma$ and $\frac{d^2C_1(t,S,\sigma)}{d\sigma^2}\frac{\sigma^2}{2}$, where more computationally demanding terms are dropped even if sometimes they are not negligible (thus using the term ’pulling out’ more appropriate than ’approximating’). Informally this can be written in the following logical chain for the first and second order term:

$$\begin{aligned}
\frac{d}{d\sigma}\Big(C_1(t,S,\sigma)\Big)\sigma &= A\ S\exp^{-\left(d_1^2/2\right)}\frac{d}{d\sigma}\Big(d_1\Big)\sigma \\
&\approx A\ S\exp^{-\left(d_1^2/2\right)}\frac{d}{d\sigma}\left(\frac{\frac{\sigma^2}{2}(T-t)}{\sigma(T-t)^{1/2}}\right)\sigma \\
&\approx A\ S\exp^{-\left(d_1^2/2\right)}(T-t)^{1/2}\sigma \\
&= A\ \zeta_{11}(t,S,\sigma) \qquad (4.2.7)\\
\frac{d^2}{d\sigma^2}\Big(C_1(t,S,\sigma)\Big)\frac{\sigma^2}{2} &= \frac{d}{d\sigma}\left(\frac{d}{d\sigma}\Big(C_1(t,S,\sigma)\Big)\right)\frac{\sigma^2}{2} \\
&\approx A\ \frac{d}{d\sigma}\left(S\exp^{-\left(d_1^2/2\right)}(T-t)^{1/2}\right)\frac{\sigma^2}{2} \\
&\approx A\ S\exp^{-\left(d_1^2/2\right)}(T-t)^{1/2}(T-t)^{1/2}\sigma^2 \\
&= A\ \zeta_{12}(t,S,\sigma) \qquad (4.2.8)
\end{aligned}$$

where A is a constant of our "approximation". Thus functions $\zeta_{11}(t,S,\sigma)$ and $\zeta_{12}(t,S,\sigma)$ are part of decomposition (see Equation 4.2.6) and can be tried as basis functions, which are relatively quick to evaluate.

4.2.2 Alternative LSM Iterative Algorithm Implementation

The authors [35] closely follow the code developed in the Longstaff et. al. paper [37], which evaluates a single pass over the entire data starting from expiry (all trajectories at each time step are evaluated) and recursively moving backwards. This results in a single iteration over the LSM algorithm, which we believe could be improved. We propose to stay in the spirit of the iterative regression using the matrices A and b, which are re-calculated based on the updated weight, until two successive policies are found only have small differences. The Algorithm 7 follows the results of the loss minimisation provided by Equation 1.3.1. We still work backwards using backward recursion, but iterate either until convergence or for a maximum of $n_{LSM} = 15$ iterations.

Algorithm 7 Iterative Version of Longstaff-Schwartz Algorithm

```
1. Initialize the Q-function at time t = T
   1: $\widehat{Q}(T, S, \sigma) \leftarrow \mathbf{0}$
   2: $\widehat{w}(T) \leftarrow \mathbf{0}$
2. Calculate
   1: for each iteration $i = 1, \ldots, n_{LSM}$ or until convergence do
   2:     for each time step $t = T - 1, \ldots, 1$ do
   3:         $\widehat{A}(t) = \Sigma_j \phi(t, S_t^j, \sigma_t^j)\phi(t, S_t^j, \sigma_t^j)^{\mathrm{T}}$
   4:         $\widehat{b}(t) = \gamma\Sigma_j \phi(t, S_t^j, \sigma_t^j)\max\Big(g(S_{t+1}^j), Q\big(t+1, S_{t+1}^j, \sigma_{t+1}^j; \widehat{w}(t+1)\big)\Big)$
   5:         $\widehat{w}(t) = (A(t))^{-1}b(t)$
   6:     end for
   7:     Converged if change in policy is small
   8: end for
3. Report $\widehat{\mathbf{w}}$
```

4.2.3 Exercise Only If In-The-Money (LSPI, training)

The authors train the LSPI algorithm by allowing it to exercise the option if immediate payoff is greater than the continuation value, $(K - S_t) > Q_t$ as discovered via close examination of their code, which was kindly provided by the authors. The authors do NOT check if the option is in-the-money, which leads to an exercise while the option is out of-the-money and negative Q, giving a reward of 0 (thus updating the A as $\phi\phi^{\mathrm{T}}$ vs. the actual update $\phi_t(\phi_t - \gamma\phi_{t+1})^{\mathrm{T}}$). The variable 'stopIndex' acts as an action, since it is later used to make $Phi_{next} = 0$ if 'stopIndex' is a number, otherwise (if 'stopIndex' is NaN, the variable Phi_{next} is not set to 0).

```
S                          = samplePaths(path,:)';
Phi_curr(:, :)             = Phi(:,path,:);
currQ                      = Phi_curr'*w;
Phi_next                   = Phi_curr;
Phi_next(:, 1:numSteps-1)  = Phi_curr(:, 2:numSteps);
Phi_next(:, numSteps)      = 0;
reward                     = zeros(numSteps, 1);
stopIndex                  = find( K-S > currQ );
reward(stopIndex, 1)       = max(0,K-S(stopIndex));
Phi_next(:, stopIndex)     = 0;
```

```
A = A + Phi_curr * (Phi_curr- gamma * Phi_next)';
b = b + Phi_curr* reward;
```

We believe this to be suboptimal, since it incorrectly affects the calculation of matrix A, via an incorrect setting of $\phi_{t+1} = 0$. Thus in our code of Chapter 4 we implement our version of LSPI, which only executes the option if it is in-the-money during training.

4.3 Results

Volatility Basis We have tested the volatility basis discussed above, however they did not seem to contribute to the learning of the exercise policies, since the weights associated with each of the additional basis functions was almost zero. The performance of the three algorithms with volatility basis functions is comparable to their performance without volatility basis functions however no large differences were observed, therefore we do not report the results. The Complex Volatility basis seems to perform slightly better than the No Volatility basis, which performs better than the Simple Volatility basis. This idea was not pursued further, however can be crossed off the list as tested and can still be thought of as contribution, because we have proposed volatility basis functions inspired by Black-Scholes.

Iterative LSM The LSM boundaries discovered by the iterative LSM algorithm developed in this thesis, shows smooth, monotonically increasing boundaries as a function of time (see Figures 4.1 for real test data and 4.2, for simulated test data). This is in contrast to the choppy boundaries found by the Schuurmans et. al. [35], as re-produced in Chapter 3 using original LSM code (see Figures 3.4 and 3.5). Equally the payoffs (see Tables 4.1, 4.2) show an improvement when compared to the payoffs of the non-iterative LSM algorithm of Chapter 3 (see Tables 3.4,3.5).

Table 4.1 **Average DJI option payoffs, real data (No Vol).**

	LSM		
Tenor	GBM	GARCH	DATA
63	1.186	1.164	1.127
126	1.546	1.508	1.471
252	1.771	1.851	2.024

Table 4.2 **Average DJI option payoffs, synthetic data (No Vol).**

	LSM	
Tenor	GBM	GARCH
63	2.083	1.887
126	2.866	2.525
252	3.800	3.221

Updating Policy - LSPI We have tested the different execution rule (only in the money options are exercised during training), and this has not significantly improved the results. Therefore this idea was not pursued further, however can be crossed off the list as tested. This

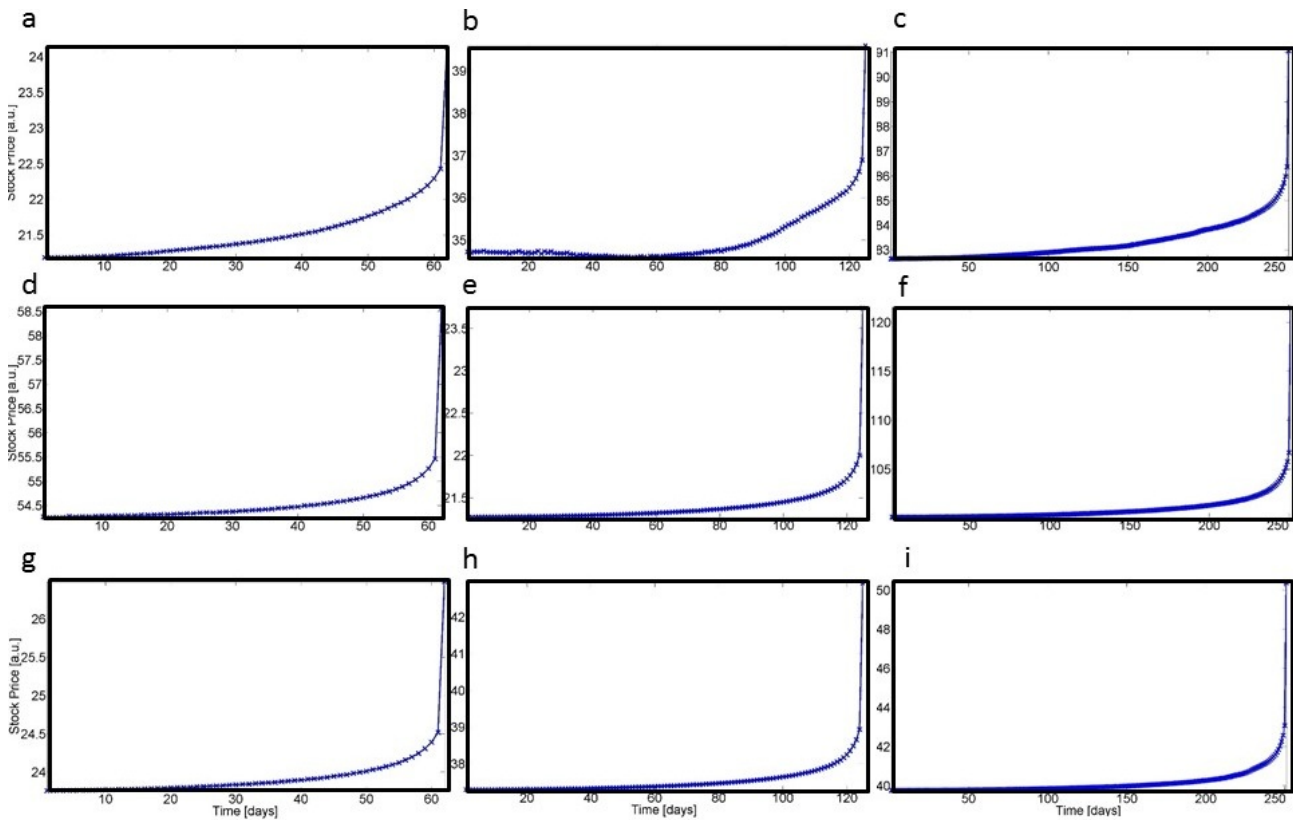


Fig. 4.1 **LSM (improved) - Boundaries for individual stocks, real data.** This figure demonstrates the LSM algorithm's boundaries as learnt using the LSM code developed for this thesis. The boundaries are shown for a number of different stocks, tenors, and for policies trained on a variety of data (real/synthetic) and tested on real data. **(a)** Company: JPM. Policy was trained with real data, and tested on real data (Tenor = 63 days). **(b)** Company: XOM. Policy was trained with real data, and tested on real data (Tenor = 126 days). **(c)** Company: GS. Policy was trained with real data, and tested on real data (Tenor = 252 days). **(d)** Company: MMM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 63 days). **(e)** Company: KO. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 126 days). **(f)** Company: IBM. Policy was trained with GBM synthetic data, and tested on real data (Tenor = 252 days). **(g)** Company: MCD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 63 days). **(h)** Company: DD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 126 days). **(i)** Company: HD. Policy was trained with GARCH synthetic data, and tested on real data (Tenor = 252 days).

must be due to the fact that the awarded reward was zero if the option is out the money and therefore it probably did not affect the learning policy enough to make large payoff improvements.

4.4 Discussion

We have tested some further ideas in this Chapter on how the three developed codes can be improved, and the suggestion for the iterative LSM algorithm seemed to pay off. The estimated av. DJI option payoffs are higher, and most importantly the found boundaries are the smoothest from all of the results we have seen in the previous chapters. We believe that this is our biggest original contribution to of this thesis together with building the framework for the 3 algorithms, which can be investigated further via future work.

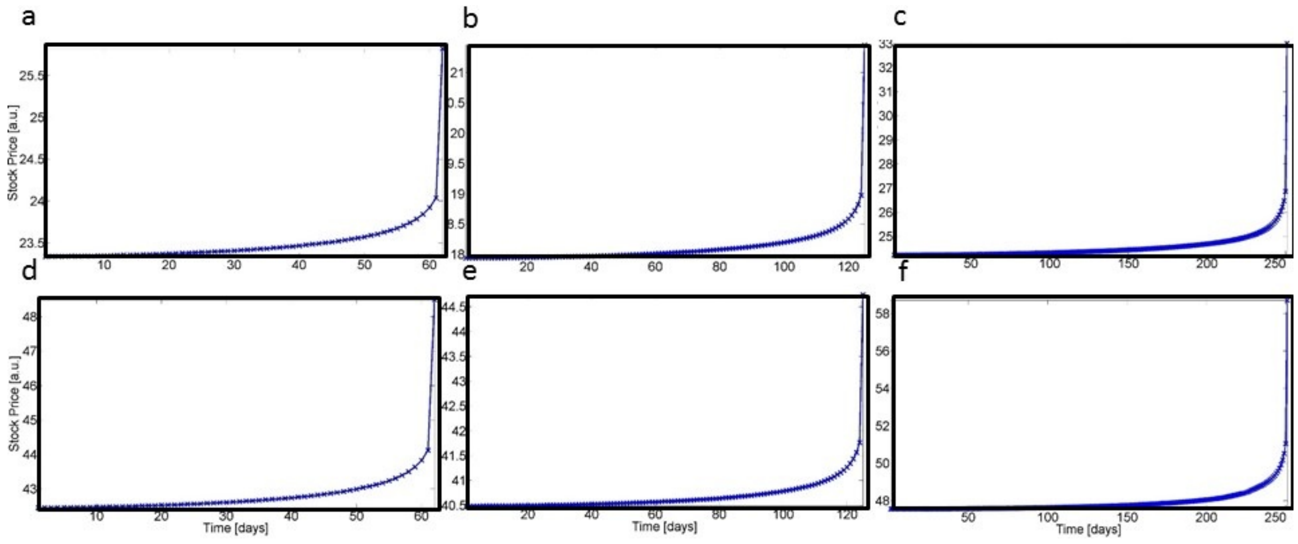


Fig. 4.2 **LSM (improved) - Boundaries for individual stocks, synthetic data.** This figure gives example boundaries found by the LSM algorithm for individual stocks of different tenors and for policies trained and tested on synthetic data only. **(a)** Company: CAT. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 63 days). **(b)** Company: DIS. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 126 days). **(c)** Company: INTC. Policy was trained with GBM synthetic data, and tested on synthetic GBM data (Tenor = 252 days). **(d)** Company: VZ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 63 days). **(e)** Company: CVX. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 126 days). **(f)** Company: JNJ. Policy was trained with GARCH synthetic data, and tested on synthetic GARCH data (Tenor = 252 days).

Chapter 5

General Discussion

We have addressed an important class of problems which remains a challenge to the Financial Engineering community - the pricing of derivatives with American features, via examination the three prominent state-of-the-art simulation based continuous Reinforcement Learning algorithms: the Longstaff-Schwartz (LSM), Fitted-Q Iteration (FQI), and Least Squares Policy Iteration (LSPI) algorithms.

Our thesis introduces the problem in Chapter 1 and provides the necessary Literature review as well as mathematical background, including full derivation of the LSPI algorithm in Chapter 2.

Chapter 3 then build a solid foundation for the three algorithms, establishing the necessary computational framework and re-creates the results of the Schuurmans et. al. [35] paper, which largely agree with our findings.

We then note the specifically poor results of the LSM algorithm, which is surprising since it constitutes the algorithm of choice with the Wall Street practitioners. We seek to improve the non-iterative version of the LSM algorithm presented by the authors [35], by sticking to the iterative nature of the RL algorithms FQI and LSPI, where the weights are improve until successive policies show small enough differences.

The improved LSM results following in Chapter 4, which we believe form our main contribution.

References

[1] Antos, A., Szepesvári, C., and Munos, R. (2008). Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129.

[2] Averbukh, V. Z. (1997). *Pricing American Options Using Monte Carlo Simulation.* Cornell University, Aug.

[3] Baird III, L. C. (1994). Reinforcement learning in continuous time: Advantage updating. *IEEE International Conference on Neural Networks*, 4:2448–2453.

[4] Barone-Adesi, G. and Whaley, R. E. (1987). Efficient analytic approximation of american option values. *The Journal of Finance*, 42(2):301–320.

[5] Bellman, R. (1954). The theory of dynamic programming. Technical report, DTIC Document.

[6] Bellman, R. and Dreyfus, S. (1959). Functional approximations and dynamic programming. *Mathematical Tables and Other Aids to Computation*, 13(68):247–251.

[7] Björk, T. (2004). *Arbitrage theory in continuous time.* Oxford university press.

[8] Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *The journal of political economy*, pages 637–654.

[9] Bossaerts, P. (1989). Simulation estimators of optimal early exercise. *Graduate School of Industrial Administration.*

[10] Boyle, P., Broadie, M., and Glasserman, P. (1997). Monte carlo methods for security pricing. *Journal of economic dynamics and control*, 21(8):1267–1321.

[11] Boyle, P. P. (1977). Options: A monte carlo approach. *Journal of financial economics*, 4(3):323–338.

[12] Bradtke, S. J. (1993). Reinforcement learning applied to linear quadratic regulation. *Advances in neural information processing systems*, pages 295–295.

[13] Bradtke, S. J. and Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57.

[14] Brigo, D. and Mercurio, F. (2007). *Interest rate models-theory and practice: with smile, inflation and credit.* Springer Science & Business Media.

[15] Broadie, M. and Detemple, J. (1996). American option valuation: new bounds, approximations, and a comparison of existing methods. *Review of Financial Studies*, 9(4):1211–1250.

[16] Broadie, M., Detemple, J., Ghysels, E., and Torrés, O. (2000). American options with stochastic dividends and volatility: A nonparametric investigation. *Journal of Econometrics*, 94(1):53–92.

[17] Broadie, M. and Glasserman, P. (1997). Pricing american-style securities using simulation. *Journal of Economic Dynamics and Control*, 21(8):1323–1352.

[18] Broadie, M. and Glasserman, P. (2004). A stochastic mesh method for pricing high-dimensional american options. *Journal of Computational Finance*, 7:35–72.

[19] Broadie, M., Glasserman, P., and Jain, G. (1997). Enhanced monte carlo estimates for american option prices. *The Journal of Derivatives*, 5(1):25–44.

[20] Cox, J. C., Ross, S. A., and Rubinstein, M. (1979). Option pricing: A simplified approach. *Journal of financial Economics*, 7(3):229–263.

[21] Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Comp.*, 12(1):219–245.

[22] Dubrov, B. (2015). Monte carlo simulation with machine learning for pricing american options and convertible bonds. *Available at SSRN 2684523.*

[23] Duff, S. J. and Bradtke Michael, O. (1995). Reinforcement learning methods for continuous-time markov decision problems. *Adv Neural Inf Process Syst*, 7:393.

[24] Duffie, D. (2010). *Dynamic asset pricing theory.* Princeton University Press.

[25] Geske, R. and Johnson, H. E. (1984). The american put option valued analytically. *The Journal of Finance*, 39(5):1511–1524.

[26] Glosten, L. R., Jagannathan, R., and Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. *The journal of finance*, 48(5):1779–1801.

[27] Google (2016). Financial data services.

[28] Hull, J. C. (2006). *Options, futures, and other derivatives.* Pearson Education India.

[29] Ibanez, A. and Zapatero, F. (2004). Monte carlo valuation of american options through computation of the optimal exercise frontier. *Journal of Financial and Quantitative Analysis*, 39(02):253–275.

[30] Jarrow, R. A. and Rudd, A. T. (1983). *Option pricing.* Richard d Irwin.

[31] Jia, Q. (2009). Pricing american options using monte carlo methods. *Department of Mathematics, Uppsala University.*

[32] Ju, N. (1998). Pricing by american option by approximating its early exercise boundary as a multipiece exponential function. *Review of Financial Studies*, 11(3):627–646.

[33] Ju, N. and Zhong, R. (1999). An approximate formula for pricing american options. *The Journal of Derivatives*, 7(2):31–40.

[34] Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149.

[35] Li, Y., Szepesvari, C., and Schuurmans, D. (2009). Learning exercise policies for american options. In *International Conference on Artificial Intelligence and Statistics*, pages 352–359.

[36] Lipton, A. (2001). *Mathematical methods for foreign exchange: A financial engineer's approach.* World Scientific.

[37] 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.

[38] MacMillan, L. W. (1986). Analytic approximation for the american put option. *Advances in futures and options research*, 1(1):119–139.

[39] Raymar, S. B. and Zwecher, M. J. (1997). Monte carlo estimation of american call options on the maximum of several stocks. *The Journal of Derivatives*, 5(1):7–23.

[40] Rogers, L. C. (2002). Monte carlo valuation of american options. *Mathematical Finance*, 12(3):271–286.

[41] Rust, J. (1997). Using randomization to break the curse of dimensionality. *Econometrica: Journal of the Econometric Society*, pages 487–516.

[42] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3:211–229.

[43] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction.* MIT press.

[44] Tilley, J. A. (1993). Valuing american options in a path simulation model. *Transactions of the Society of Actuaries*, 45(83):104.

[45] Trigeorgis, L. (1991). A log-transformed binomial numerical analysis method for valuing complex multi-option investments. *Journal of Financial and Quantitative Analysis*, 26(03):309–326.

[46] Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690.

[47] Tsitsiklis, J. N. and Van Roy, B. (1999). Optimal stopping of markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control*, 44(10):1840–1851.

[48] Tsitsiklis, J. N. and Van Roy, B. (2001). Regression methods for pricing complex american-style options. *Neural Networks, IEEE Transactions on*, 12(4):694–703.

[49] Vrabie, D., Pastravanu, O., Abu-Khalaf, M., and Lewis, F. L. (2009). Adaptive optimal control for continuous-time linear systems based on policy iteration. *Automatica*, 45(2):477–484.

[50] White, D. A. and Sofge, D. A. (1992). *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptative Approaches.* Van Nostrand Reinhold Company.

Appendix A

Additional Information

A.1 Data

The full list of 30 Dow Jones Index stock names and their corresponding tickers is contained in Table A.1.

A.1.1 Processing Data

An example historical Apple stock price is used to demonstrate the windowing and the standardisation techniques described in Chapter 3.

Table A.1 **Dow Jones underlying stocks.**

Ticker	**Stock Name**
MMM	3M
AXP	American Express
AAPL	Apple
BA	Boeing
CAT	Caterpillar
CVX	Chevron
CSCO	Cisco
KO	CocaCola
DIS	Disney
DD	E I du Pont de Nemours and Co
XOM	Exxon Mobil
GE	General Electric
GS	Goldman Sachs
HD	Home Depot
IBM	IBM
INTC	Intel
JNJ	Johnson and Johnson
JPM	JPMorgan Chase
MCD	McDonalds
MRK	Merck
MSFT	Microsoft
NKE	Nike
PFE	Pfizer
PG	Procter and Gamble
TRV	Travellers Companies Inc
UTX	Untied Technologies
UNH	United Health
VZ	Verizon
V	Visa
WMT	Walmart

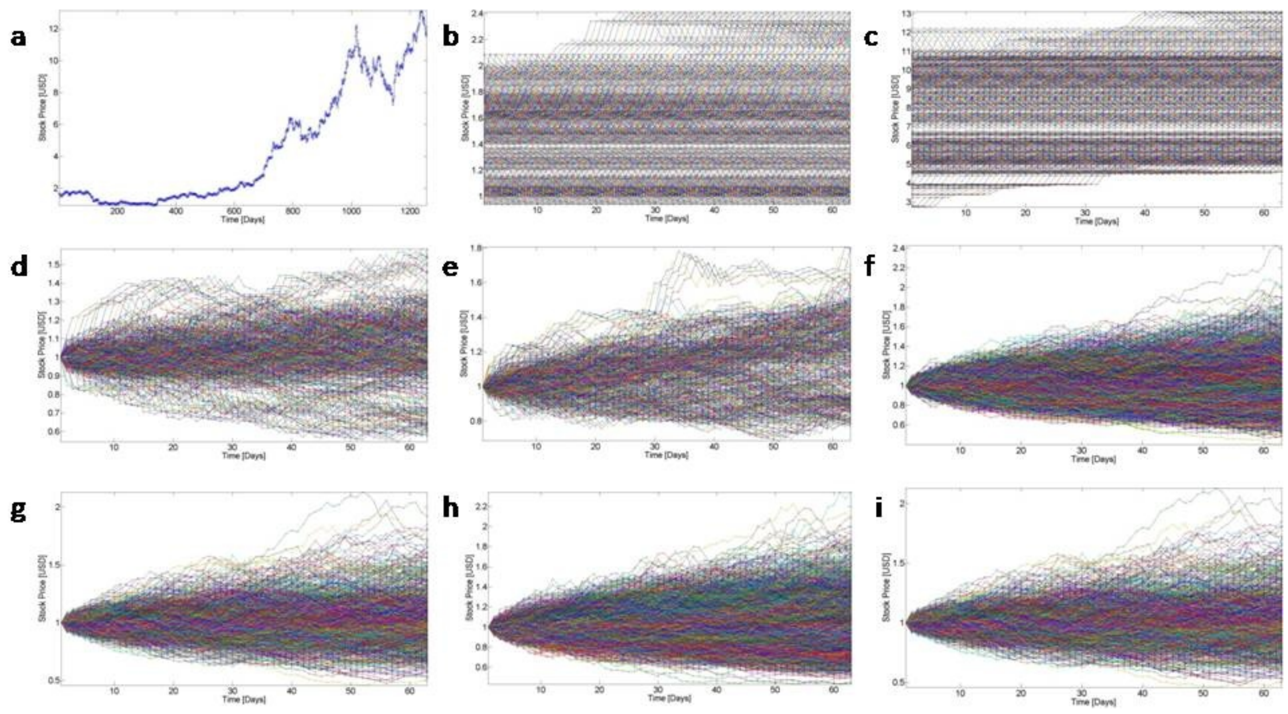


Fig. A.1 **Apple Stock Price Data Processing.** The figure demonstrates the windowing technique for an example tenor of 63 business days using historical adjusted Apple stock prices over the period of Jan 2002 - Dec 2006. **(a)** depicts the 1259 data points (days) of realised adjusted daily closing Apple stock prices. This example is particularly interesting because the stock price has gone up so dramatically thus showing the need for standardisation of the trajectories after the windowing technique is applied. If this is not done, the algorithm would be trained on data in the price range of 1 - 2.4 usd, while it would be tested on price range of 3 - 12 usd. **(b)** depicts the real data training trajectories obtained following the windowing technique, which result in 600 trajectories of 63 days duration in the range of 1 - 2.4 usd. **(c)** depicts the real data testing trajectories obtained following the windowing technique, which result in 500 trajectories of 63 days duration in the range of 3 - 12 usd. **(d)** depicts real training data of figure (b), standardised such that each path begins from $S_1 = 1$ stock price. **(e)** depicts real testing data of figure (c), standardised such that each path begins from $S_1 = 1$ stock price. **(f)** depicts the training trajectories obtained with a GBM model, with parameters $\mu = r = 0.03$ and $\sigma = 0.4262$ extracted from real data. The model uses antithetic variance reduction as per Equation **??**, where each simulated trajectory has a reduction in the variance for the second half of the data. **(g)** depicts the testing trajectories obtained with a GBM model, with parameters $\mu = r = 0.03$ and $\sigma = 0.4262$ extracted from real data. **(h)** depicts the training trajectories obtained with a Gaussian GARCH model, with parameters $const = 0.0001$ and $garch = 0.7947$ extracted from real data, using Matlab inbuilt function *estiamte()*. **(i)** depicts the testing trajectories obtained with a Gaussian GARCH model, with parameters $const = 0.0001$ and $garch = 0.7947$ extracted from real data, using Matlab inbuilt function *estiamte()*.

Appendix B

Code

B.1 Thesis Code

All of the results can be re-created by using the $seed = 112212$ on our code, which is supplied in the Dropbox with the following link:

$www.dropbox.com/sh/29k97aref1hdzvz/AAC9WmU4tjpEvTJBqF0fXmKIa?dl = 0.$

B.1.1 Main Script

The script run.m is the code which runs everything. It requires a choice of the algorithm, specified by variable 'm' (m=1 is LSPI, m=2 is FQI and m=3 is LSM).

```
clearvars; close all;
% use either a random or a given seed as entry point to the random num. generator
useSeeding = 1; % keep track of which seed was used
giveSeed = 1;   % give a particular seed to be used
givenSeed = 112212;
seed = 0;
if useSeeding
seed = randseed();
if giveSeed
seed = givenSeed;
end
display([ 'seeding experiment with: ' num2str(seed)]);
```

```
s = RandStream('mcg16807','Seed',seed);
oldStream = RandStream.setGlobalStream(s); % old stream replaced with seed
end
r                    = 0.03; % 3% interest rate used for all data
Tenors               = [63, 126, 252]; % investigated tenors
N_Training_Paths     = 50000; % Num GBM training paths (must be divisible by 2).
N_Test_Paths         = 10000;
Realized_Vol_Window = 1; % 0, 1 or 21 % if <=1 means we are not using volatility.
% ---------------------- SELECT ALGORITHM FOR TESTING ---------------------
m = 1; % LSPI
% m = 2; % FQI
% m = 3; % LSM
% ---------------------- TEST THE ALGOIRTHM --------------------------------
[Real_Payoffs,Sim_Payoffs,Mu,GBM_Vol,GARCH_Param] = American_Option_Pricing...
(numBasis,Tenors,N_Training_Paths,N_Test_Paths,Realized_Vol_Window,r,m)
```