Monte Carlo Simulation with Machine Learning for Pricing American Options and Convertible Bonds

Bella Dubrov

November 1, 2015

Abstract

Li, Szepesvari and Schuurmans (2009) show that reinforcement learning (RL) algorithms are superior to the classical methods (such as Longstaff and Schwartz (2001)) in pricing American options using Monte Carlo simulation. We extend their techniques to the problem of pricing convertible bonds and show that RL outperforms LS on this task. Additionally, we propose a new method, based on the random forest algorithm from machine learning [Breiman (2001)], that can be used for pricing both American options and convertible bonds with Monte Carlo simulation. We show that this algorithm outperforms LS and is also superior to RL in most cases.

We demonstrate how to use Monte Carlo simulation with the methods described above for pricing a complex convertible bond trading at the Tel Aviv stock exchange. Like many Israeli convertibles, this bond exhibits the "gradually diminishing principal" feature, meaning that instead of one payment of the principal at maturity, there are multiple principal payments during the lifetime of the bond. This feature presents a challenge to existing models. We also model other exotic features of this bond, such as path-dependent conversion ratio and exchange rate indexation. The prices that we obtain using this model are close to the market prices of the bond.

Electronic copy available at: https://ssrn.com/abstract=2684523
Electronic copy available at: http://ssrn.com/abstract=2684523

1 Introduction

The modern theory of option pricing began with the seminal paper of Black and Scholes (1973), that gave a closed-form solution for pricing European vanilla options. In practice we often encounter non-vanilla derivatives that have complex features, such as American, Asian and Bermudan options, dividend payout schemes of the underlying asset, convertible bonds and so on. When pricing these derivatives, we have to resort to numerical methods, such as lattice and Monte Carlo models, since no analytical solutions exist. One exception to this is the Roll-Geske-Whaley model [Roll (1977); Geske (1979, 1981); Whaley (1981)], which is a closed-form model used to price an American call that pays one dividend. This model, however, contains errors [Haug, Haug and Lewis (2003)].

The most basic numerical method for derivative pricing is the binomial model introduced by Cox, Ross and Rubinstein (1979). This is a simple dynamic programming approach and it can be employed successfully whenever it is possible to build the entire binomial tree. For example, this works well for one-factor models for vanilla options with no dividends on the underlying asset. The resulting tree recombines and therefore does not grow exponentially with the number of time points.

However, it is impractical to build the full tree for models with multiple state-variables, for complex derivatives or underlying assets with discrete dividends. This phenomenon is known as the "curse of dimensionality". In these cases Monte Carlo simulation is often applied. The method works as follows. First, paths for the price process of the underlying asset are chosen randomly (according to the model for the price process, for example, Brownian motion). Then the derivative is priced along every one of these paths. The option price is the average among all the Monte Carlo paths.

Monte Carlo approach to derivative pricing was first used by Boyle (1977) for pricing European options. When applying this method to American options, we face the issue of the optimal exercise strategy: when is it optimal for the investor to exercise the option along each path? Without the exercise strategy we cannot compute the derivative price. Longstaff and Schwartz (2001) show a way to extend the Monte Carlo framework for pricing

Electronic copy available at: https://ssrn.com/abstract=2684523
Electronic copy available at: http://ssrn.com/abstract=2684523

American Options. Their method is currently widely used by practitioners for pricing various instruments with early exercise features. They try to find a close-to-optimal exercise strategy by computing an approximate continuation value of the derivative at each relevant state.

Actually, the problem of finding an optimal exercise policy for an American-style derivative can be presented as the problem of finding an optimal action policy in a *Markov Decision Process* (MDP) [Tsitsiklis and Van Roy (2001); Li, Szepesvari and Schuurmans (2009)]. An MDP is a discrete-time stochastic control process. There is a stochastic system and an agent that acts within the system. At each time point the process is in some state s. The agent chooses some action a and the system then stochastically moves to the next state s' that depends on s and a. Additionally, the agent gets a reward at each time point, depending on the state s and action a. The goal of the agent is to maximize the total expected sum of rewards.

MDPs first became popular in the late fifties and sixties due to research by Bellman (1957) and Howard (1960). Currently MDPs are used in a variety of areas including artificial intelligence, robotics, operations research and more. Because of the "curse of dimensionality" the focus in recent years has been on finding approximate solutions to the optimal policy problem. Many algorithms have been developed for this purpose in the reinforcement learning and approximate dynamic programming communities [Sutton and Barto (1998); Powell (2007); Wiering (2012)].

These algorithms seem naturally suited for the early exercise policy problem and recently a new line of research started applying them in this context [Tsitsiklis and Van Roy (2001); Li, Szepesvari and Schuurmans (2009)]. We implement the algorithms from these papers and run some empirical experiments comparing them to the LS method. Our experiments confirm previously reported results: reinforcement learning algorithms discover better exercise policies, which leads to more accurate option prices.

Additionally, we apply reinforcement learning methods to the problem of pricing convertible bonds. *Convertible bonds* (or simply convertibles) are corporate bonds that the investor can, under certain terms, convert to the stock of the issuing company. Convertibles

Electronic copy available at: https://ssrn.com/abstract=2684523

are complex investment instruments that can have many elaborate features, making them an interesting challenge for pricing models. The price of a convertible has both debt and derivative components. Many of the advances in derivative pricing can be applied to pricing convertibles [De Spiegeleer and Schoutens (2011)] and various aspects of debt pricing, such as default intensity and recovery rates, are also used in the models.

Several common features of convertibles are of special interest. First, convertibles often have a *call feature*, i.e. the issuer can call the bond under certain circumstances. In this case the investor can either convert the bond to stock or accept the call and get the redemption amount. Another interesting feature is the *put feature*, where the investor has the right to put the bond back to the issuer and receive a put price. These features make the convertible a two-party zero-sum game.

Therefore, the framework suitable for pricing convertibles is the theory of *Markov Games* (MG). A Markov Game is an extension of an MDP to environments with multiple agents. This model was first used in the context of reinforcement learning by Littman (1994). We apply the LSPI algorithm for multiple agents [Lagoudakis and Parr (2002)] in the context of pricing convertible bonds. We find that it is superior to the LS method.

We then present a new algorithm for finding optimal exercise policies: the Random Forest (RF). Similar methods are widely used in the machine learning community. Random forests have been first introduced by Breiman (2001). The idea is that instead of using one tree for some task, an *ensemble* of trees is learned, and then a majority vote between the trees is used for making the decision. This is just one of many ensemble methods that have proven useful in machine learning [Opitz and Wang (1999), Polikar (2006), Rokach (2010)]. For our application in Monte-Carlo simulation we use random forests as follows. Instead of building one tree for learning the exercise policy, we build several trees and let each one of them learn an exercise policy with the LS method. Then, when a decision needs to be made on a new tree, we use a majority vote between all the learned policies. Note that the trees are independent of each other and can therefore be built in parallel, which is a great advantage for modern computers. Hence, we can obtain better exercise policies without paying in runtime.

Electronic copy available at: https://ssrn.com/abstract=2684523

We achieve good results with random forests for both American options and convertible bonds. We see that RF outperforms LS in all cases. Sometimes LSPI outperforms RF for convertible bonds, but the difference is insignificant.

We then proceed to apply our methods for pricing a real convertible bond trading at the Tel-Aviv Stock Exchange. The Tower Vav bond is one of the most traded convertible bonds at the exchange. It has the "gradually diminishing principal" feature, meaning that instead of one payment of the principal at maturity, there are two principal payments during the lifetime of the bond. This feature presents a challenge to classical models, but can be incorporated seamlessly into a Monte-Carlo model. In addition, the bond has path-dependent conversion ratio and is indexed to the USD/ILS exchange rate. The prices that we obtain using our model are close to the market prices of the bond.

1.1 Acknowledgments

I am deeply grateful to Prof. Simon Benninga for his collaboration, support and guidance.

1.2 Organization

The rest of this paper is organized as follows. We start with some prerequisites on derivatives, including options and convertible bonds in section 2. We then proceed to introduce the required reinforcement learning background in section 3, including the reinforcement problem formulation, basic algorithms and the more advanced algorithms that we will use in our research. We present the single agent setting in section 3.1 and the multiple agents setting in section 3.2. Section 4 shows how the reinforcement learning algorithms can be applied to derivative pricing, discussing American options in section 4.1 and convertible bonds in section 4.2. We introduce random forests and their application to derivative pricing in section 5. We proceed to present our empirical results for pricing American options and convertible bonds in section 6. We show how to use our techniques for pricing a real convertible bond in section 7. We discuss our conclusions in section 8 and suggest directions for future research in section 9.

Electronic copy available at: https://ssrn.com/abstract=2684523

2 Prerequisites: Derivatives

2.1 Options

The classical option pricing models (in particular the Black and Scholes model and the binomial model) are the basis for the algorithms that we develop here, therefore it is important to understand them before proceeding to the more elaborate Monte Carlo methods. A good source for this purpose is Hull (2011).

In practice options appear in many varieties. *Asian options* are options where the payoff depends on the average price of the underlying asset during some period in the life of the option. This is an example of a path-dependent option, where in order to price the option it is not enough to know the price at the time of option exercise, but the price path of the underlying asset needs to be considered. Therefore, we cannot price these options using a full recombining tree and usually have to resort to Monte Carlo pricing. A *Hawaiian option* is an Asian option with an early exercise feature. It might be interesting to try our methods at pricing these options.

Another challenge for option pricing models is the presence of dividends of the underlying asset. It is convenient to model the dividends as a time-continuous dividend yield. In this case we can use standard modifications of the binomial and Black and Scholes models. This approach is plausible when modeling dividends of a large portfolio of stocks. A more realistic approach for a single stock, however, is to treat dividends as discrete. If the dividends are proportional to the underlying price, a recombining binomial tree can be used. It is reasonable to use this model for long term options, where future dividends are correlated with the stock price. For short-term options the absolute dividend amount can usually be predicted. If the dividends are discrete and absolute, we encounter a problem using the binomial model: the tree does not recombine. If the number of dividends during the option lifetime is not too large, it might be still feasible to build the entire binomial tree. (The tree size grows as follows: the number of nodes at the next level is multiplied by two after each dividend date.) For longer-term options, however, the tree becomes prohibitively large and

Electronic copy available at: https://ssrn.com/abstract=2684523

we again have to turn to Monte Carlo methods.

Another situation where we might want to employ Monte Carlo methods is when there are several stochastic factors driving the model, such as stochastic volatility (as in Heston (1993)), stochastic interest rates an so on. In these cases the tree grows much faster and can become prohibitively large.

2.1.1 Monte Carlo Methods for Options with Early Exercise Features

The method of Monte Carlo simulation is the following. First, paths for the price process of the underlying asset are chosen randomly (according to the model for the price process, for example, Brownian motion). Then the derivative is priced along every one of these paths. The average of the paths prices is the estimated option price.

Monte Carlo approach to derivative pricing was first used by Boyle (1977) for pricing European options. Broadie and Glasserman (1996) use Monte Carlo simulation for Asian options. Longstaff and Schwartz (2001) were the first to extend the Monte Carlo framework for pricing American options.

The problem with using Monte Carlo methods for options that can be exercised early is that of deciding when to exercise the option along each path. The option price is defined as the expected price when using the *optimal* exercise policy. There is a subtle point to notice here. The optimal policy is the best policy among all policies that are based only upon the information that is known at the time of the decision; no policy can predict anything about future price movements. Therefore, when trying to make the early exercise decision along a particular path, it is a mistake to choose the exercise point that leads to the best outcome. The problem here is that the choice is made considering future price movements.

Since future price moves are generally unknown, it is still possible that an optimal exercise policy is used, but the investor "regrets" an exercise decision that she made earlier. This is illustrated by the following simple example. We have an American put option with strike price 100 for a stock with current price 100, up = 1.2, down = 0.8, risk free rate = 5%. The two-period binomial tree is presented in Figure 1. The risk neutral probabilities are:

Electronic copy available at: https://ssrn.com/abstract=2684523

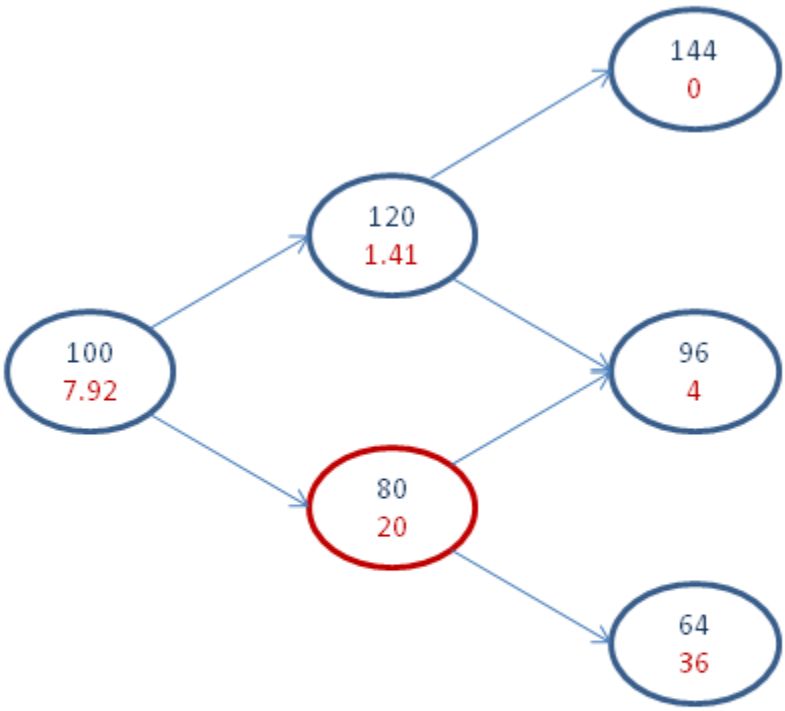

Figure 1: A simple binomial tree for an American option with strike price 100. Option prices are shown in red.

Electronic copy available at: https://ssrn.com/abstract=2684523

$$p_{up} = \frac{e^r - d}{u - d} = 0.6282$$

$$p_{down} = 0.3718$$

At the red node the continuation price of the option is $(4p_{up} + 36p_{down})e^{-r} = 15.12$. The exercise price is 20, therefore the investor chooses to exercise. But in the next period, if the price of the stock goes down to 64, the value of the option is 36 and we have $36e^{-r} = 34.24 > 20$, so the investor regrets exercising the option.

Our research deals with situations where we cannot find the optimal exercise strategy and therefore settle for an approximate solution. This means that our results are always biased low (since an optimal exercise strategy would give a higher option price). This point was observed by Broadie and Glasserman (1996). These authors propose using an algorithm that is biased high in conjunction with the low-biased one in order to obtain a range for the option price.

As we mentioned earlier, we compare our results with the popular Longstaff & Schwartz algorithm. The details of this algorithm can be found in Longstaff and Schwartz (2001).

2.2 Convertible Bonds

For an excellent overview of convertibles and state-of-the-art pricing models for them, please see De Spiegeleer and Schoutens (2011). We give a short summary here.

A *convertible bond* is a corporate bond that can be converted to the stock of the issuing company by the bond holder. This conversion is an option for the investor.

Let N be the face value of the convertible. Let C_r be the conversion ratio, namely the number of stocks that the investor can get for one bond. Then the payoff of the convertible at maturity is $max(N, C_r S) = N + max(0, C_r S - N)$. Some simple valuation models use this formula and price the convertible by viewing it as an option and a corporate bond. However, this model is very simplistic and unrealistic, because of the additional features that are usually present in convertible bonds. We outline some of these features below.

Electronic copy available at: https://ssrn.com/abstract=2684523

Most convertibles are of American nature: they can be converted prior to maturity, usually during a predefined period of time. Some convertibles allow only *contingent conversion (CoCo)*, which means that the bond can be converted only when the stock price is above a trigger level. Next to the conversion feature there is also the *callability* feature: during a specified period of time the issuer has the right to buy back the bond, paying the investor a *redemption amount* for it. After the bond is called, there is a certain period of time when the investor can convert it to stock. This is called *forced conversion*. It addition, some convertibles are *puttable*, i.e. the investor can put the bond back to the issuer for a predetermined put value.

Since a realistic model for convertible pricing needs to consider all these elaborate features, some form of a tree-based model seems most suitable. A tree model can describe all the different scenarios and conditions. In a tree model we price going backwards from the last time point to the first, at each node deciding what is the best action for both parties (investor and issuer). In order to make this decision, both parties need to know the *continuation value* of the bond. In a binomial tree the continuation value is computed using the children of the node. A heuristic for computing the continuation value is needed if a Monte Carlo approach is used. Given the continuation value and the other relevant values (these are the put, redemption and conversion values, which are always known at each node), the best action is decided according to Table 1.

Situation	**Decision**
put is largest	investor puts
redemption > put > others	investor puts
(redemption > conversion > continuation) and (conversion > put)	voluntary convert
(redemption > continuation > conversion) and (continuation > put)	no action
(conversion is largest) and (redemption > continuation)	voluntary convert
(conversion is largest) and (redemption < continuation)	call + convert
continuation is largest, put second largest	call + put
continuation is largest, redemption second largest	call + get redemption
continuation is largest, conversion second largest	forced conversion

Table 1: Decisions for a Node in a Convertible Bond Model

Some important factors to consider in a practical model are credit default risk, stochastic

Electronic copy available at: https://ssrn.com/abstract=2684523

volatility and stochastic interest rates. Because of the "curse of dimensionality" it is rarely practical to build the entire tree. Therefore Monte Carlo methods are gaining popularity in the convertible bonds community [Lvov, Yigitbasioglu, El Bachir (2004); Mishchenko, Mishchenko and Malyarenko (2007); Ammann, Kind and Wilde (2008); Pang, Wang and Li (2011); Beveridge and Joshi (2011)]. This is also the approach that we are going to take in this research.

3 Prerequisites: Reinforcement Learning

3.1 Single Agent Setting

This section mostly follows Sutton and Barto (1998).

3.1.1 The Reinforcement Learning Problem

The reinforcement learning problem is the problem of learning an optimal behavior strategy in a stochastic environment. In this problem the time is assumed to be discrete, i.e. there are discrete time points $t_0, t_1, t_2, ...$, where at time point t $(t = 0, 1, 2, ...)$ the environment is assumed to be at some defined *state* $s_t \in S$. At time point t the agent takes some *action* $a_t \in A$ and the environment moves to the next state $s_{t+1} \in S$, depending on the current state and the chosen action. Additionally, the agent receives a *reward* $r_t \in \mathbb{R}$ at time point t.

The goal of the agent is to maximize the total amount of the rewards it receives *in the long run*. Usually, the goal is formally defined as maximizing the total expected *discounted* sum of rewards $r_0 + \gamma r_1 + \gamma^2 r_2 + ...$, for some $0 < \gamma \leq 1$.

The agent implements a *policy*, i.e. a mapping from states to probability distributions over the action space. The policy $\pi(s, a)$ defines the probability to choose the action a at state s.

We will be looking at tasks that satisfy the *Markov property*, i.e. the next state is dependent only on the current state and current action (and is independent of the rest of

Electronic copy available at: https://ssrn.com/abstract=2684523

the state/action history given the current state and action).

The processes that we described above are called *Markov Decision Processes (MDPs)* and are formally defined as follows.

Definition 1. *A* Markov Decision Process *is a five-tuple* $(S, A, P(\cdot,\cdot,\cdot), R(\cdot,\cdot), \gamma)$*, where* S *is the set of possible states,* A *is the set of possible actions,* $P(s', s, a) = P(s'|s, a)$ *is the probability that the next state will be* s'*, given that the current state is* s *and the current action is* a*,* $R(s, a)$ *is the reward at state* s *for taking action* a *and* γ *is a discount factor.*

Another notion that is frequently encountered in the context of reinforcement learning algorithms is that of *value functions*. A value function estimates "how good" a state is, if a particular policy is followed by the agent from this state on. This can be expressed as the expected (discounted) total reward if a policy π is followed starting from the state s.

Definition 2. *The* state-value function V *for a policy* π *is defined as* $V^\pi(s) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s]$*, where the expectation is according to the transition probabilities function* P *and the policy* π*.*

Definition 3. *The* action-value function Q *for a policy* π *is defined as* $Q^\pi(s, a) = E[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a]$.

The value functions must satisfy the following recursive properties.

$$\begin{aligned} V^\pi(s) &= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s\right] \\ &= \sum_a \pi(s, a) \sum_{s'} P(s'|s, a)[R(s, a) + \gamma V^\pi(s')] \end{aligned} \tag{3.1}$$

Equation 3.1 is called the *Bellman equation for* V^π and the value function V^π is the unique solution to its Bellman equation. An analogous equation can be formulated for Q^π.

For MDPs with finite state and action spaces, we can define the *optimal action policy* π^* as follows. For two policies π and π' we say that $\pi \geq \pi'$ if $V^\pi(s) \geq V^{\pi'}(s)$ for all states s. An

Electronic copy available at: https://ssrn.com/abstract=2684523

optimal policy always exists (but is not necessarily unique). Moreover, it can be shown that a deterministic optimal policy always exists. All optimal policies share the same state-value function, denoted V^* and defined as $V^*(s) = \max_\pi V^\pi(s)$. Similarly, they share the same action-value function, denoted Q^*.

The optimal value function must satisfy the *Bellman optimality equation*, defined as follows.

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a) \left[R(s,a) + \gamma V^*(s')\right] \tag{3.2}$$

For finite MDPs the Bellman optimality equation has a unique solution. Actually, if we have N states, equation 3.2 gives us a system of N equations, which we can solve to find V^*. Once we have V^*, we can find an optimal policy by choosing an action that gives the maximum in equation 3.2 at every state. Solving equation 3.2 directly is rarely feasible, because of large or infinite state spaces. Therefore, approximate solutions are usually used.

3.1.2 Basic Dynamic Programming Algorithms

We first consider the problem of computing the value function V^π for some given policy π. V^π is the solution to its Bellman equation (equation 3.1) and its existence and uniqueness are guaranteed as long as either $\gamma < 1$ or eventual termination of the process is guaranteed. First, note that equation 3.1 is a system of $|S|$ equations with $|S|$ unknowns, so this system can be solved to compute V^π. An iterative method is, however, more practical. The *iterative policy evaluation* algorithm is described in Algorithm 1.

Convergence is defined as having $\max_s |V_k(s) - V_{k-1}(s)|$ smaller than some threshold.

Now that we know how to compute the value function, we can introduce the *policy iteration algorithm*: an iterative algorithm for computing optimal policies. The algorithm is described in Algorithm 2.

In order to compute the value functions faster (line 3), it is best to initialize the com-

Electronic copy available at: https://ssrn.com/abstract=2684523

Algorithm 1 Iterative Policy Evaluation

for all s: initialize $V_0(s)$ arbitrarily
$k \leftarrow 1$
repeat
  for all s: $V_k(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V_{k-1}(s')]$
  $k \leftarrow k + 1$
until convergence
return V_k

Algorithm 2 Policy Iteration

1: for all s: initialize $V(s)$ and $\pi(s)$ arbitrarily
2: **repeat**
3: compute V^π
4: for all s: $\pi(s) \leftarrow argmax_a \sum_{s'} P(s'|s,a)[R(s,a) + \gamma V^\pi(s')]$
5: **until** convergence

putation to the values computed in the preceding iteration (for the preceding policy), since subsequent policies are similar and therefore have close value functions.

Note that in each iteration of the above algorithm, we go over the entire state space, which can be prohibitively large. To avoid doing this, we might use *asynchronous policy iteration*, an algorithm that updates state values and policies as it encounters them. Such an algorithm is guaranteed to converge, as long as it continues to update all states. The advantage of this approach is that it updates the states that are actually encountered by the agent more frequently.

The policy iteration algorithm consists of two processes: computation of the value function for the current policy and greedy improvement of the current policy according to the current available value function. These two processes can be interleaved in many ways: we can perform several steps of value function computation and then a few steps of policy improvement. The term *general policy iteration* refers to any algorithm performing these steps. The algorithm converges, as long as all the states continue to be updated by both processes.

3.1.3 Parametric Value Functions and Algorithms that Use Them

This section mostly follows Lagoudakis and Parr (2003).

Electronic copy available at: https://ssrn.com/abstract=2684523

We frequently encounter state spaces that are very large or even infinite, and in these cases we cannot store the value function for every state. Consequently, we have to work with a generalization of the value function and represent it in parametric form.

A *linear architecture* is a popular form of parametrization. Define $Q(s,a) = \sum_{i=1}^{k} \phi_i(s,a) w_i$, where ϕ_i's are some basis functions and w_i's are weights. The ϕ_i's can be arbitrary linearly independent functions of s and a. The value function evaluation algorithm tweaks the weights w_i in order to better approximate the true value function.

Let Q^π be the (unknown) value function of a policy π given as a column vector of size $|S||A|$. Let also $\hat{Q}^\pi$ be the vector of the approximate state-action values as computed by a linear approximation architecture with parameters w_j and basis functions ϕ_j, $j = 1, 2, ..., k$. Define $\phi(s,a)$ as the following column vector of size k:

$$\phi(s,a) = \begin{pmatrix} \phi_1(s,a) \\ \phi_2(s,a) \\ ... \\ \phi_k(s,a) \end{pmatrix}.$$

Define the matrix Φ as follows:

$$\begin{pmatrix} \phi(s_1,a_1)^T \\ \phi(s_2,a_2)^T \\ ... \\ \phi(s_{|S|},a_{|A|})^T \end{pmatrix},$$

that is, each row of Φ contains all the values of the basis functions for some state-action pair (s,a) and each column contains the values of a specific basis function for all state-action pairs. If the basis functions are linearly independent, then the columns of Φ are linearly independent as well.

Now, $\hat{Q}^\pi$ can be expressed as $\hat{Q}^\pi = \Phi w^\pi$.

We need to find a way to compute $\hat{Q}^\pi$ for a given policy π, that is to compute the

Electronic copy available at: https://ssrn.com/abstract=2684523

weights w^π. What we require from $\hat{Q}^\pi$ is that it approximates Q^π. We now present two ways of approximation: Bellman Residual Minimizing Approximation and Least-Squares Fixed-Point Approximation.

We use the following notation.

- Q^π is the $|S||A| \times 1$ column vector such that $Q^\pi_{a+(s-1)|A|} = Q^\pi(s,a)$.
- R is the $|S||A| \times 1$ column vector that assigns a reward to each state-action pair: $R_{a+(s-1)|A|} = R(s,a)$.
- P is the $|S||A| \times |S|$ transition probability matrix: $P_{a+(s-1)|A|,s'} = P(s'|s,a)$.
- Π_π is a matrix of size $|S| \times |S||A|$ that describes the policy π, i.e. $\Pi_\pi(s,(s',a)) = \pi(a|s)$ if $s = s'$ and 0 otherwise.
- Φ is a $|S||A| \times k$ matrix as defined above.
- w is a $k \times 1$ column vector as defined above.

Bellman Residual Minimizing Approximation We know that Q^π satisfies the Bellman equation:

$$Q^\pi = R + \gamma P \Pi_\pi Q^\pi.$$

Therefore we demand that

$$\begin{aligned} \hat{Q}^\pi &\approx R + \gamma P \Pi_\pi \hat{Q}^\pi \\ \Phi w^\pi &\approx R + \gamma P \Pi_\pi \Phi w^\pi \\ (\Phi - \gamma P \Pi_\pi \Phi) w^\pi &\approx R \end{aligned}$$

Denote $\Psi \triangleq (\Phi - \gamma P \Pi_\pi \Phi)$, so we get

$$\Psi w^\pi \approx R \tag{3.3}$$

Electronic copy available at: https://ssrn.com/abstract=2684523

Equation 3.3 is an overconstrained system of equations: it has $|S||A|$ equations and k unknowns. We would like to solve it in the least-square sense, namely, we would like to minimize $(\Psi w^\pi - R)^2$, which is achieved by choosing

$$w^\pi = (\Psi^T \Psi)^{-1} \Psi^T R \tag{3.4}$$

Equation 3.4 is called the *Bellman residual minimizing approximation* to the true value function.

Least-Squares Fixed-Point Approximation

Definition 4. *Given a policy π we define the* Bellman operator *T_π (operating on action-value functions) as follows:*

$$(T_\pi Q)(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q(s',a').$$

In matrix form we get:

$$T_\pi Q = R + \gamma P \Pi_\pi Q. \tag{3.5}$$

Therefore, we get that the action-value function is the fixed point of the Bellman operator:

$$T_\pi Q^\pi = Q^\pi.$$

Hence, we would like to have the approximation function to be a fixed point of the Bellman operator, namely:

$$T_\pi \hat{Q}^\pi \approx \hat{Q}^\pi. \tag{3.6}$$

By definition $\hat{Q}^\pi$ lies in the space of approximate value functions, namely the space spanned by the basis functions. However, $T_\pi \hat{Q}^\pi$ may not lie in that space and therefore has

Electronic copy available at: https://ssrn.com/abstract=2684523

to be projected there, if we want equation 3.6 to hold. Therefore, we replace $T_\pi \hat{Q}^\pi$ by its orthogonal projection to the approximate value functions space, namely $\Phi(\Phi^T\Phi)^{-1}\Phi^T T_\pi \hat{Q}^\pi$. Note that the orthogonal projection to the column space of Φ is well-defined because the columns of Φ (the basis functions) are linearly independent by definition.

Equation 3.6 therefore becomes:

$$\begin{aligned}
\Phi(\Phi^T\Phi)^{-1}\Phi^T T_\pi \hat{Q}^\pi &= \hat{Q}^\pi \\
\Phi(\Phi^T\Phi)^{-1}\Phi^T (R + \gamma P \Pi_\pi \hat{Q}^\pi) &= \hat{Q}^\pi \\
\Phi(\Phi^T\Phi)^{-1}\Phi^T (R + \gamma P \Pi_\pi \Phi w^\pi) &= \Phi w^\pi \\
\Phi((\Phi^T\Phi)^{-1}\Phi^T (R + \gamma P \Pi_\pi \Phi w^\pi) - w^\pi) &= 0 \\
(\Phi^T\Phi)^{-1}\Phi^T (R + \gamma P \Pi_\pi \Phi w^\pi) - w^\pi &= 0 \\
(\Phi^T\Phi)^{-1}\Phi^T (R + \gamma P \Pi_\pi \Phi w^\pi) &= w^\pi \\
\Phi^T (R + \gamma P \Pi_\pi \Phi w^\pi) &= \Phi^T \Phi w^\pi \\
\Phi^T (\Phi - \gamma P \Pi_\pi \Phi) w^\pi &= \Phi^T R
\end{aligned}$$

This is a system of k equations with k unknowns. The solution of this system is:

$$w^\pi = (\Phi^T(\Phi - \gamma P \Pi_\pi \Phi))^{-1}\Phi^T R.$$

The solution is guaranteed to exist for all, but finitely many, values of γ (Koller and Parr (2000)). Since the orthogonal projection minimizes the L_2 norm, the solution w^π yields a value function $\hat{Q}^\pi$ which can be called the *least-squares fixed-point approximation* to the true value function.

Empirical evidence shows that the least-squares fixed-point approximation performs better than the Bellman residual minimizing approximation (Lagoudakis and Parr (2003)). Therefore they use the least-squares fixed-point approximation in their algorithm.

Electronic copy available at: https://ssrn.com/abstract=2684523

The Least Squares Policy Iteration (LSPI) Algorithm We have seen that the approximate state-action value function can be found by solving the equation

$$\Phi^T(\Phi - \gamma P \Pi_\pi \Phi) w^\pi = \Phi^T R. \tag{3.7}$$

Define $A \triangleq \Phi^T(\Phi - \gamma P \Pi_\pi \Phi)$ and $b \triangleq \Phi^T R$. Equation 3.7 cannot be solved analytically because P and R are either not known or too large. The solution is to learn A and b from samples and then solve equation 3.7 using these learned values.

Let us now take a closer look at A and b. We use the following formula in the derivations: for two matrices C and D of sizes $m \times n$ and $n \times m$ respectively

$$CD = \sum_{i=1}^{n} C_{col_i} D_{row_i},$$

where C_{col_i} is the ith column of C and D_{row_i} is the ith row of D.

We also use the fact that for all s, a

$$\sum_{s'} P(s'|s,a) = 1.$$

Electronic copy available at: https://ssrn.com/abstract=2684523

$$
\begin{aligned}
A &= \Phi^T(\Phi - \gamma P \Pi_\pi \Phi) \\
&= \sum_s \sum_a \phi(s,a)(\phi(s,a) - \gamma \sum_{s'} P(s'|s,a)\phi(s',\pi(s')))^T \\
&= \sum_s \sum_a \sum_{s'} P(s'|s,a)[\phi(s,a)(\phi(s,a) - \gamma\phi(s',\pi(s')))^T]
\end{aligned}
$$

$$
\begin{aligned}
b &= \Phi^T R \\
&= \sum_s \sum_a \phi(s,a) \sum_{s'} P(s'|s,a)R(s,a) \\
&= \sum_s \sum_a \sum_{s'} P(s'|s,a)[\phi(s,a)R(s,a)]
\end{aligned}
$$

We see that the matrix A is the sum of many matrices of the form $\phi(s,a)(\phi(s,a) - \gamma\phi(s',\pi(s')))^T$, where the summation is over all triplets (s,a,s') and it is weighted by the probability $P(s'|s,a)$. The vector b is the sum of many vectors of the form $\phi(s,a)R(s,a)$ and the summation is again over all triplets (s,a,s') and weighted by $P(s'|s,a)$. For large state and/or action spaces it is impractical to go over all triplets (s,a,s'). Since the triplets are weighted by $P(s'|s,a)$, a simulation approach to computing A and b would be to sample pairs (s,a) uniformly and to sample s' given (s,a) according to $P(s'|s,a)$. Therefore, assuming that we have a sample $D = \{(s_i, a_i, r_i, s'_i)|i = 1,2,...,L\}$, where (s_i, a_i) are sampled uniformly and s'_i is chosen according to P, we can estimate A and b as follows:

$$
\begin{aligned}
\tilde{A} &= \frac{1}{L}\sum_{i=1}^{L}[\phi(s_i,a_i)(\phi(s_i,a_i) - \gamma\phi(s'_i,\pi(s'_i)))^T] \\
\tilde{b} &= \frac{1}{L}\sum_{i=1}^{L}[\phi(s_i,a_i)r_i].
\end{aligned}
$$

Since we learn A and b in order to use them for solving the system $Aw^\pi = b$, we can drop

Electronic copy available at: https://ssrn.com/abstract=2684523

the $1/L$ term from both $\tilde{A}$ and $\tilde{b}$. Therefore, we can use the following update rules for $\tilde{A}$ and $\tilde{b}$ as new samples arrive. Initialize $\tilde{A}^{(0)} = 0$, $\tilde{b}^{(0)} = 0$. Given a new sample (s_i, a_i, r_i, s_i'), update $\tilde{A}$ and $\tilde{b}$ as follows:

$$\tilde{A}^{(i+1)} = \tilde{A}^{(i)} + \phi(s_i, a_i)(\phi(s_i, a_i) - \gamma\phi(s_i', \pi(s_i')))^T$$
$$\tilde{b}^{(i+1)} = \tilde{b}^{(i)} + \phi(s_i, a_i)r_i.$$

We now present the *Least Squares Policy Iteration* (LSPI) algorithm developed by Lagoudakis and Parr (2003) and used for option pricing by Li, Szepesvari and Schuurmans (2009). Note that a weight vector w implicitly defines a policy π: this is the policy that chooses at each state s the action a that maximizes $\hat{Q}(s, a) = \phi(s, a)w^T$. The algorithm is described in Algorithm 3.

Algorithm 3 Least Squares Policy Iteration

input: data $s_1, a_1, r_1, s_2, a_2, r_2, ..., s_L, a_L, r_L, s_{L+1}$
initialize w arbitrarily
repeat
  $\tilde{A} = \sum_{i=1}^{L}[\phi(s_i, a_i)(\phi(s_i, a_i) - \gamma\phi(s_{i+1}, \pi(s_{i+1})))^T]$
  $\tilde{b} = \sum_{i=1}^{L}[\phi(s_i, a_i)r_i]$ ($\tilde{b}$ can be computed at the beginning of iterations)
  solve $\tilde{A}w = \tilde{b}$ to obtain the next w (w also defines the next policy)
until convergence

3.2 Multiple Agents Setting

A survey of multi-agent reinforcement learning can be found in Busoniu, Babuska and De Schutter (2008). This section is based on [Littman (1994); Lagoudakis and Parr (2002)]. We will focus on two-player zero-sum Markov games, since this is the setting that we need for pricing convertible bonds. The players are called "agent" and "opponent".

Definition 5. *A* two-player zero-sum Markov game *is a six-tuple* (S, A, O, P, R, γ)*, where* S *is the set of possible states,* A *and* O *are finite sets of actions of the agent and opponent respectively and* P *is a Markovian state transition model.* $P(s, a, o, s')$ *is the probability that*

Electronic copy available at: https://ssrn.com/abstract=2684523

s' will be the next state when the current state is s, the agent takes action a and the opponent takes action o. R is a reward (or cost) function. $R(s, a, o)$ is the reward of the agent for taking action a when the opponent takes action o. γ is a discount factor.

A few notes are in order.

- If the opponent is permitted only a single action, the Markov game becomes an MDP.
- The agent's action in each state is chosen so that to maximize the expected total reward assuming the opponent chooses the worst possible action. (This means that for every possible action of the agent, she assumes the reward to be the lowest possible for that action).
- Unlike MDPs, the optimal policy for a Markov game may be stochastic, but only deterministic policies are possible in our case of convertible bonds.

3.2.1 LSPI for Markov Games

The sample consists of tuples of the form $(s_i, a_i, o_i, r_i, s'_i)$, where a_i is the action of the agent, o_i is the action of the opponent, r_i is the reward and s'_i is the next state. We assume that the optimal policy π is deterministic, since we are interested in policies for convertible bonds. The equations for $\tilde{A}$ and $\tilde{b}$ are modified as follows.

$$\tilde{A}^{(i+1)} = \tilde{A}^{(i)} + \phi(s_i, a_i, o_i)(\phi(s_i, a_i, o_i) - \gamma\phi(s'_i, \pi(s'_i), o'_i))^T$$
$$\tilde{b}^{(i+1)} = \tilde{b}^{(i)} + \phi(s_i, a_i, o_i)r_i.$$

The action o'_i used above is the minimizing opponent's action in computing $\pi(s'_i)$.

Electronic copy available at: https://ssrn.com/abstract=2684523

4 Reinforcement Learning for Derivative Pricing

4.1 Reinforcement Learning for American Options Pricing

We currently implemented a discrete-time and continuous time stochastic processes for the stock price. Actually, any stochastic process for the stock price can be used. Given enough real market data, the algorithms can run without a model for the underlying process, learning from the real data.

4.1.1 Least Squares Policy Iteration (LSPI)

This section mostly follows Li, Szepesvari and Schuurmans (2009).

In the general case the data consists of a set of trajectories of the form

$$(s_{0,0}, a_{0,0}, r_{0,0}, s_{0,1}, a_{0,1}, r_{0,1}, ..., s_{0,t_0}, a_{0,t_0}, r_{0,t_0})$$

$$(s_{1,0}, a_{1,0}, r_{1,0}, s_{1,1}, a_{1,1}, r_{1,1}, ..., s_{1,t_1}, a_{1,t_1}, r_{1,t_1})$$

...

$$(s_{n,0}, a_{n,0}, r_{n,0}, s_{n,1}, a_{n,1}, r_{n,1}, ..., s_{n,t_n}, a_{n,t_n}, r_{n,t_n}).$$

In our case each state is a pair (s_t, t), where s_t is the stock price and t is the time. All actions except for the last action in each trajectory are the "hold" actions (which we denote by 0) and the last action is "exercise" (which we denote by 1). All rewards except for the last one are 0 and the last reward is the intrinsic value of the option, which we denote by $g(s)$.

We now derive the equation for the weights w. We denote by $Q_0(s_t, t)$ the continuation value for a state (s_t, t). The state-action value function is, therefore,

$$Q(s,t,a) = \begin{cases} Q_0(s,t) & \text{if } a = 0 \\ g(s) & \text{if } a = 1. \end{cases}$$

Electronic copy available at: https://ssrn.com/abstract=2684523

We would like to represent Q_0 using a linear architecture, i.e. $Q_0(s,t) = \sum_{i=1}^{k} \phi_i(s,t)w_i$. The following recursive rule applies to Q_0:

$$Q_0(s,t) = \gamma \sum_{s'} P(s'|s)(I_{\pi(s',t+1)=0}Q_0(s',t+1) + I_{\pi(s',t+1)=1}g(s')),$$

where I is the indicator function, i.e. it equals 1 if the underlying condition holds and 0 otherwise. In matrix form we get:

$$Q_0 = \gamma P(I_0 Q_0 + I_1 g), \tag{4.1}$$

where we use the following matrices (N denotes the state space size, considering the number of possible prices and time steps):

- Q_0 is a $N \times 1$ column vector
- P is a $N \times N$ matrix that defines the probability of moving from state (s,t) to a state (s',t') (the probability is 0 when $t' \neq t+1$)
- I_0 is an $N \times N$ diagonal matrix such that $I_0((s,t),(s,t)) = 1$ iff $\pi(s,t) = 0$ and 0 otherwise
- I_1 is an $N \times N$ diagonal matrix such that $I_1((s,t),(s,t)) = 1$ iff $\pi(s,t) = 1$ and 0 otherwise
- g is an $N \times 1$ column vector of intrinsic values for each state

We assume $Q_0 = \Phi w$. We would like to use least-squares fixed-point approximation, so we project the right hand-side of equation 4.1 to the space spanned by the basis functions

Electronic copy available at: https://ssrn.com/abstract=2684523

and get:

$$\Phi(\Phi^T\Phi)^{-1}\Phi^T\gamma P(I_0Q_0 + I_1g) = \Phi w$$
$$(\Phi^T\Phi)^{-1}\Phi^T\gamma P(I_0\Phi w + I_1g) = w$$
$$\Phi^T\gamma P(I_0\Phi w + I_1g) = \Phi^T\Phi w$$
$$\Phi^T\gamma PI_0\Phi w + \Phi^T\gamma PI_1g = \Phi^T\Phi w$$
$$(\Phi^T\Phi - \Phi^T\gamma PI_0\Phi)w = \Phi^T\gamma PI_1g$$

We therefore get an equation of the form $Aw = b$, for $A = \Phi^T(\Phi - \gamma PI_0\Phi)$ (A of size $k \times k$) and $b = \Phi^T\gamma PI_1g$ (b of size $k \times 1$). As before, we would like to estimate A and b from the data, so we rewrite them as follows. We use the following formula for multiplication of two matrices C and D:

$$(CD)_{row_i} = \sum_j C_{i,j}D_{row_j}.$$

$$\begin{aligned}
A &= \Phi^T\Phi - \gamma\Phi^T PI_0\Phi \\
&= \sum_{(s,t)} \Phi^T_{col_{(s,t)}}\Phi_{row_{(s,t)}} - \gamma\sum_{(s,t)} \Phi^T_{col_{(s,t)}}(PI_0\Phi)_{row_{(s,t)}} \\
&= \sum_{(s,t)} \phi(s,t)\phi(s,t)^T - \gamma\sum_{(s,t)} \phi(s,t)\sum_{(s',t')} P((s',t')|(s,t))I_0(s',t')\phi(s',t')^T \\
&= \sum_{(s,t)}\sum_{(s',t')} P((s',t')|(s,t))\phi(s,t)\phi(s,t)^T - \gamma\sum_{(s,t)} \phi(s,t)\sum_{(s',t')} P((s',t')|(s,t))I_0(s',t')\phi(s',t')^T \\
&= \sum_{(s,t)}\sum_{(s',t')} P((s',t')|(s,t))\phi(s,t)[\phi(s,t) - \gamma I_0(s',t')\phi(s',t')]^T
\end{aligned}$$

Electronic copy available at: https://ssrn.com/abstract=2684523

$$
\begin{aligned}
b &= \Phi^T \gamma P I_1 g \\
&= \gamma \sum_{(s,t)} \Phi^T_{col_{(s,t)}} (P I_1 g)_{row_{(s,t)}} \\
&= \gamma \sum_{(s,t)} \phi(s,t) \sum_{(s',t')} P((s',t')|(s,t)) I_1(s',t') g(s') \\
&= \gamma \sum_{(s,t)} \sum_{(s',t')} P((s',t')|(s,t)) \phi(s,t) I_1(s',t') g(s')
\end{aligned}
$$

Therefore, the LSPI algorithm for learning exercise policies is as described in Algorithm 4.

Algorithm 4 LSPI for Exercise Policies

input: data $(s_{0,0}, s_{0,1}, ..., s_{0,t_0}), (s_{1,0}, s_{1,1}, ..., s_{1,t_1}), ..., (s_{n,0}, s_{n,1}, ..., s_{n,t_n})$
initialize w arbitrarily
repeat
  $\tilde{A} = \sum_{(i,t)} \phi(s_{i,t}, t)[\phi(s_{i,t}, t) - \gamma I_0(s_{i,t+1}, t+1)\phi(s_{i,t+1}, t+1)]^T$
  $\tilde{b} = \sum_{(i,t)} \phi(s_{i,t}, t)\gamma I_1(s_{i,t+1}, t+1) g(s_{i,t+1})$
  solve $\tilde{A}w = \tilde{b}$ to obtain the next w (w also defines the next policy)
until convergence

4.1.2 Fitted Q-Iteration (FQI)

This algorithm is from Tsitsiklis and Van Roy (2001) and it was tested empirically by Li, Szepesvari and Schuurmans (2009).

Similarly to the previous section, we define $Q((S_t, t), 0) = w^T \phi(S_t, t)$ to be the estimate of the continuation value. We would like the continuation value to be "close" to $\gamma \max_a Q((S_{t+1}, t+1), a)$, i.e. we would like to solve the following minimization problem

$$
w \leftarrow argmin_w \sum_{t,j} [w^T \phi(S_t^j, t) - \gamma \max_a Q((S_{t+1}, t+1), a)]^2.
$$

Define $A = \sum_{t,j} \phi(S_t^j, t)\phi(S_t^j, t)^T$ and $b = \gamma \sum_{t,j} \phi(S_t^j, t) \max\{g(S_{t+1}^j), Q((S_{t+1}^j, t+1), 0)\}$. Then in each iteration we set $w \leftarrow A^{-1}b$.

Electronic copy available at: https://ssrn.com/abstract=2684523

4.2 Reinforcement Learning for Convertible Bonds

We adjust LSPI for two-player zero-sum Markov games of Lagoudakis and Parr (2002) for convertible bonds.

4.2.1 LSPI for Convertible Bonds

The agent's actions are: continue = 0, put = 1, convert = 2, getFaceValue = 3. The opponent's actions are: continue = 0, call = 1. In this case a state is a pair (s_t, t), where s_t is the price at time t. All actions of the agent and the opponent except for the last action in each path are the "continue" actions. We denote by $Q_{00}(s_t, t)$ the continuation value of the bond for the state (s_t, t). Denote by R_{put} the put value, by R_{call} the redemption value in case of call and by R_{face} the face value of the bond that is received at maturity. The state-action value function is, therefore,

$$Q(s,t,a,o) = \begin{cases} Q_{00}(s,t) & \text{if } a = 0 \\ R_{put} & \text{if } a = 1 \\ s_t & \text{if } a = 2 \\ R_{face} & \text{if } a = 3 \\ R_{call} & \text{if } a = 0 \text{ and } o = 1. \end{cases}$$

We would like to represent Q_{00} using a linear architecture, i.e. $Q_{00}(s,t) = \sum_{i=1}^{k} \phi_i(s,t) w_i$. The following recursive rule applies to Q_{00}:

$$Q_{00}(s,t) = \gamma \sum_{s'} P(s'|s)(I_{\pi(s',t+1)=00} Q_{00}(s', t+1) + I_{\pi(s',t+1)=1} R_{put} +$$
$$I_{\pi(s',t+1)=2} s' + I_{\pi(s',t+1)=3} R_{face} + I_{\pi(s',t+1)=01} R_{call})$$

Electronic copy available at: https://ssrn.com/abstract=2684523

In matrix form we get:

$$Q_{00} = \gamma P(I_{00}Q_{00} + I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call})$$

$$\begin{aligned}
\Phi(\Phi^T\Phi)^{-1}\Phi^T\gamma P(I_{00}Q_{00} + I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}) &= \Phi w \\
(\Phi^T\Phi)^{-1}\Phi^T\gamma P(I_{00}\Phi w + I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}) &= w \\
\Phi^T\gamma P(I_{00}\Phi w + I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}) &= \Phi^T\Phi w \\
\Phi^T\gamma P I_{00}\Phi w + \Phi^T\gamma P(I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}) &= \Phi^T\Phi w \\
(\Phi^T\Phi - \Phi^T\gamma P I_{00}\Phi)w &= \Phi^T\gamma P(I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}).
\end{aligned}$$

Define $A \triangleq \Phi^T(\Phi - \gamma P I_{00}\Phi)$ and $b \triangleq \Phi^T\gamma P(I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call})$.

Using the formula

$$CD = \sum_{i=1}^{n} C_{col_i} D_{row_i},$$

we get

$$A = \sum_{(s,t)} \sum_{(s',t')} P((s',t')|(s,t))\phi(s,t)[\phi(s,t) - \gamma I_{00}(s',t')\phi(s',t')]^T.$$

Let us now compute b:

$$\begin{aligned}
b &= \Phi^T\gamma P(I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call}) \\
&= \gamma \sum_{(s,t)} \Phi^T_{col_{(s,t)}} P(I_1 R_{put} + I_2 S + I_3 R_{face} + I_{01} R_{call})_{row_{(s,t)}} \\
&= \gamma \sum_{(s,t)} \phi(s,t) \sum_{(s',t')} P((s',t')|(s,t))[I_1(s',t')R_{put} + I_2(s',t')s' + I_3 R_{face} + I_{01}(s',t')R_{call}] \\
&= \gamma \sum_{(s,t)} \sum_{(s',t')} P((s',t')|(s,t))\phi(s,t)[I_1(s',t')R_{put} + I_2(s',t')s' + I_3 R_{face} + I_{01}(s',t')R_{call}].
\end{aligned}$$

Therefore, the LSPI algorithm for learning exercise policies for convertibles is as described in Algorithm 5.

Electronic copy available at: https://ssrn.com/abstract=2684523

Algorithm 5 LSPI for Convertibles Exercise Policies

input: data $(s_{0,0}, s_{0,1}, ..., s_{0,t_0}), (s_{1,0}, s_{1,1}, ..., s_{1,t_1}), ..., (s_{n,0}, s_{n,1}, ..., s_{n,t_n})$
initialize w arbitrarily
repeat
$\tilde{A} = \sum_{(i,t)} \phi(s_{i,t}, t)[\phi(s_{i,t}, t) - \gamma I_{00}(s_{i,t+1}, t+1)\phi(s_{i,t+1}, t+1)]^T$
$\tilde{b} = \sum_{(i,t)} \phi(s_{i,t}, t)\gamma[I_1(s_{i,t+1}, t+1)R_{put} + I_2(s_{i,t+1}, t+1)s_{i,t+1} + I_3(s_{i,t+1}, t+1)R_{face} + I_{01}(s_{i,t+1}, t+1)R_{call}]$
solve $\tilde{A}w = \tilde{b}$ to obtain the next w (w also defines the next policy)
until convergence

5 Random Forests Application to Derivative Pricing

Random forests were first introduced in [Breiman (2001)]. This is an intuitive technique that has many advantages. Random forests have since been used in numerous applications in a broad spectrum of domains, see for example [Burea et al. (2005); Cutler et al. (2007); Statnikov et al. (2008)].

A simple random forest builds a large set of decision trees and then acts according to the decision chosen by most trees. The application to Monte Carlo simulation is straightforward: instead of building one tree, build many trees (the number of trees depends on the computational power available, we found that 15 trees are often enough in our models). For each tree run the LS method in order to learn the early exercise policy. Now build a new tree for pricing the derivative (similarly to the second stage of the LS method). At each node where early exercise is possible, ask each of the trees whether it recommends to exercise or hold (or any other action according to the derivative at hand). Choose the action recommended by most trees.

The advantages of random forests are numerous. First, note that the 15 trees do not need to be stored in memory simultaneously (as opposed to building one large tree that contains all the paths of the 15 trees). We can build the trees sequentially or in parallel (if parallel computing resources are available). The trees are independent of each other, so when using parallel computing, each tree can be computed and its regression models extracted independently of the other trees. We only need to store the exercise policies learned by the trees. In our case, we need to store 15 regression models for each time point.

Electronic copy available at: https://ssrn.com/abstract=2684523

When we build the second stage tree, we decide whether to exercise the derivative based on a majority vote between the 15 regression models (for example, if in 10 of the 15 models the estimated continuation value is smaller than the exercise value, we choose to exercise the option). Another advantage of the random forest method is that its model is richer than simple linear regression, which allows to build more precise exercise policies.

6 Results

All our algorithms were implemented in Java and run on a standard laptop. Similar results (for smaller models) were obtained using Excel with Visual Basic.

6.1 American Put Options

In order to test the various algorithms on a simple derivative, we first price a plain vanilla American put option using Monte Carlo simulation. We compare different techniques for learning exercise policies. We use the Brownian motion model for the stock price process. Similar results can be obtained with other models of the stock price process, such as GARCH. We use the following option parameters.

Parameter	**Value**
option type	PUT
current stock price	100
strike price	100
μ	0
σ	0.2
risk free rate	0.03
compounding	continuous
Δt	1/252
T (expiration time in years)	1

Table 2: Option Parameters

The exact option price (using the binomial model) is 6.7389. We run all our Monte Carlo algorithms the following way. For each algorithm we first build 5,000 random tree paths for *training*: these paths are used for learning the exercise policy. The exercise rules consider

Electronic copy available at: https://ssrn.com/abstract=2684523

only the underlying stock price and time at the decision node. We then use 5,000 new paths for *testing*: we employ the exercise rule in order to price the option on these paths. The higher the price of the algorithm along the paths, the better is the discovered exercise rule.

We test the following algorithms.

LS This is the Longstaff and Schwartz method using the following basis functions (S is the stock price):

$$\phi_0(S) = 1$$
$$\phi_1(S) = S$$
$$\phi_2(S) = S^2$$
$$\phi_3(S) = S^3$$

The algorithm learns a different set of weights for each time stamp, i.e. the continuation value of the option is represented by $\sum_i w_i \phi_i$ with different w_i's for each time stamp.

LSPI This is the LSPI algorithm using simple polynomials and three time functions. The algorithm learns one set of weights across many time stamps. The basis functions are the following (S is the stock price and t is the distance from tree root, T is the expiration time):

$$\phi_0(S) = 1$$
$$\phi_1(S) = S$$
$$\phi_2(S) = S^2$$
$$\phi_3(S) = S^3$$
$$\phi_4(t) = \sin(-t\pi/2T + \pi/2)$$
$$\phi_5(t) = \ln(T - t)$$
$$\phi_6(t) = \left(\frac{t}{T}\right)^2$$

Electronic copy available at: https://ssrn.com/abstract=2684523

Similar results can be obtained using Laguerre polynomials (with the same time functions) as in Li, Szepesvari and Schuurmans (2009). These are the following functions:

$$\begin{aligned}
\phi_0(S) &= 1 \\
\phi_1(S) &= e^{-S'/2} \\
\phi_2(S) &= e^{-S'/2}(1 - S') \\
\phi_3(S) &= e^{-S'/2}(1 - 2S' + S'^2/2) \\
\phi_4(t) &= \sin(-t\pi/2T + \pi/2) \\
\phi_5(t) &= \ln(T - t) \\
\phi_6(t) &= \left(\frac{t}{T}\right)^2
\end{aligned}$$

We discovered that simple polynomials work just fine, so there is no need to use more complex functions.

FQI This is the FQI algorithm of Tsitsiklis and Van Roy (2001). The algorithm learns one set of weights across many time stamps.

Random Forest We run our random forest algorithm described in section 5 with just 15 trees (more trees do not give better results in this case).

We run each algorithm 2000 times and compute the average price for training and testing paths. All algorithms run using the same set of training and testing paths. The achieved prices are summarized in the table 3.

Please note, that the option should be priced using the result in the testing column. The training result is just given for completeness. It will usually be biased up, because the exercise strategy is not independent of future behavior of the stock, as we have explained before.

Electronic copy available at: https://ssrn.com/abstract=2684523

Algorithm	Training Average	Testing Average	Training Stdev	Testing Stdev
LS	6.7977	6.6989	0.11	0.11
LSPI	6.7427	6.7056	0.12	0.11
FQI	6.7482	6.7073	0.11	0.11
Random Forest	6.7321	6.7146	0.11	0.11

Table 3: American Put Option Results

We see that the reinforcement learning algorithms outperform the LS algorithm and random forest outperforms all other algorithms.

When comparing the LS and LSPI algorithms, we get a difference of the means of $6.7056 - 6.6989 = 0.0067$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{LSPI}^2}{2000}} = 0.0035$ (2000 is the number of runs). Therefore, our result is 1.93 standard deviations from 0, so we get a p-value of 0.03.

When comparing the LS and random forest algorithms, we get a difference of the means of $6.7146 - 6.6989 = 0.0157$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{RF}^2}{2000}} = 0.0036$ (2000 is the number of runs). Therefore, our result is 4.43 standard deviations from 0, so we get a p-value of 0.

Please note, that all 15 trees can be built in parallel on modern computers. In this case we get running time similar to the original LS algorithm. To conclude, the random forest algorithm is simple, fast and superior to previous methods.

6.2 Convertible Bonds

We price various convertible bonds using the Monte Carlo method again comparing the different techniques for making put/convert/call decisions.

6.2.1 Example 1

This is a convertible bond without put/call options. The bond parameters are specified in table 4.

The binomial tree with the conversion boundary is shown in Figure 2.

The exact convertible price (using the binomial model) is 109.59. We run all our Monte

Electronic copy available at: https://ssrn.com/abstract=2684523

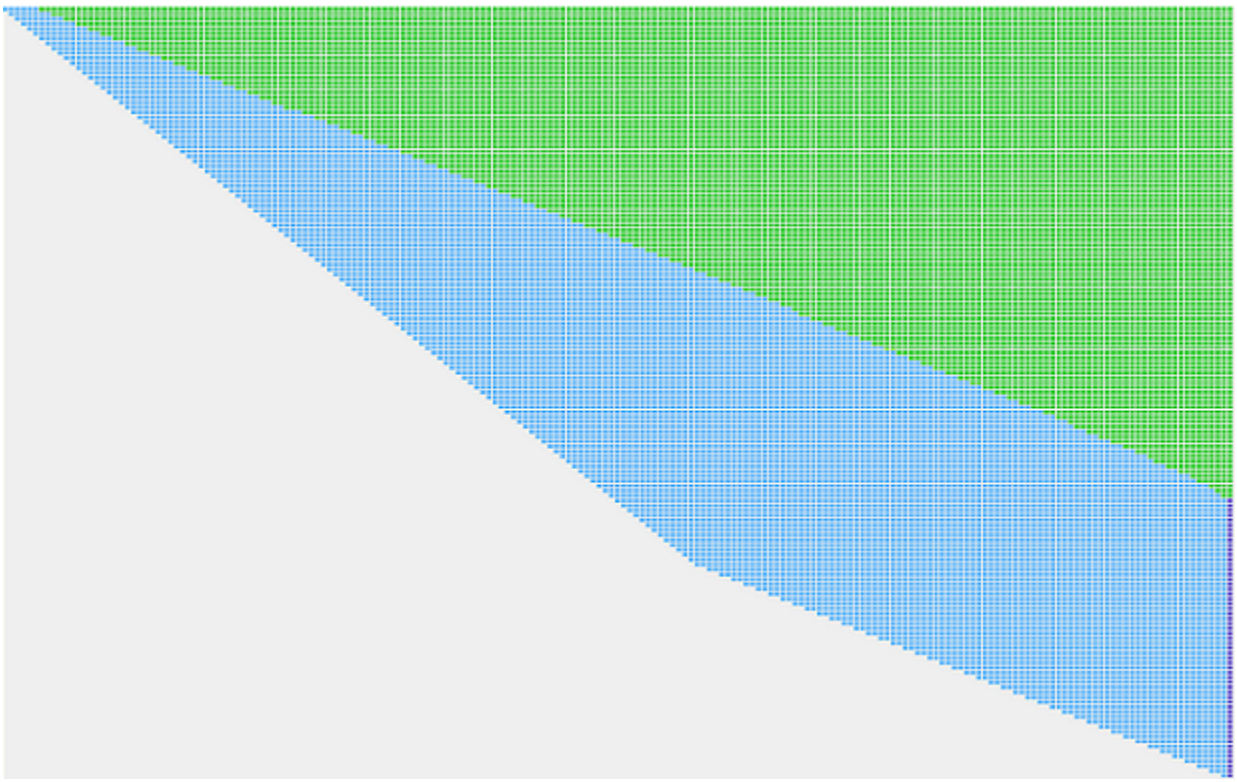

Figure 2: A binomial tree for convertible 1. Green nodes are the nodes where conversion is the optimal action.

Electronic copy available at: https://ssrn.com/abstract=2684523

Parameter	**Value**
current stock price	100
face value	100
coupons	none
μ	0
σ	0.4
dividend yield	0.1
risk free rate	0.03
compounding	continuous
Δt	5/200
T (expiration time in years)	5
conversion ratio	1

Table 4: Convertible 1 Parameters

Carlo algorithms similarly to the put option experiment. We use 2000 paths for training and 2000 paths for testing. The results are an average of 2000 runs. The algorithms we run are LS with simple basis functions, LSPI with simple and time (t and t^2) basis functions with grouping the nodes to buckets of 5 time points each and the RF algorithm with 7 trees. The results are presented in table 5.

Algorithm	**Out of Sample Paths Average**	**Stdev**
LS	108.38	1.36
LSPI	108.46	1.29
Random Forest	109.25	0.65

Table 5: Convertible 1 Results

We again observe that the reinforcement learning algorithm outperforms the LS algorithm and random forest outperforms all other algorithms.

When comparing the LS and LSPI algorithms, we get a difference of the means of $108.46 - 108.38 = 0.08$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{LSPI}^2}{2000}} = 0.042$ (2000 is the number of runs). Therefore, our result is 1.91 standard deviations from 0, so we get a p-value of 0.03.

When comparing the LS and random forest algorithms, we get a difference of the means of $109.25 - 108.38 = 0.87$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{RF}^2}{2000}} = 0.034$. Therefore, our result is 25.8 standard deviations from 0, so we get a

Electronic copy available at: https://ssrn.com/abstract=2684523

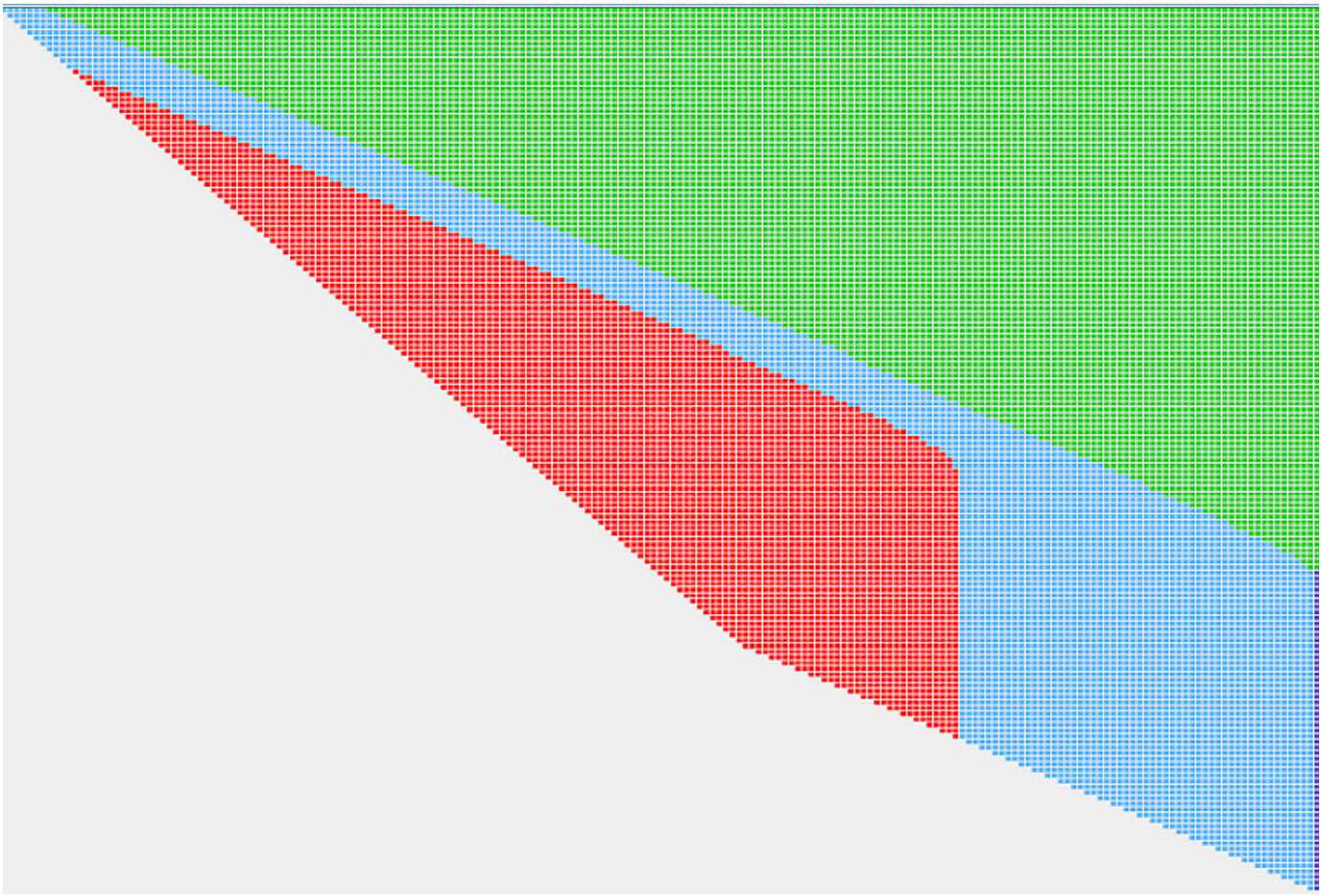

Figure 3: A binomial tree for convertible 2. Green nodes are the nodes where conversion is the optimal action. Red nodes are the nodes where put is the optimal action.

p-value of 0.

6.3 Example 2

This convertible bond is similar to the previous one, but it has a put option with a put price of 96. The binomial model gives a price of 111.32. The binomial tree with the conversion and put boundaries is shown in Figure 3.

The performance of the algorithms is described in 6.

When comparing the LS and LSPI algorithms, we get a difference of the means of $110.95 - 110.68 = 0.27$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{LSPI}^2}{2000}} = 0.021$. Therefore, our result is 12.7 standard deviations from 0, so we get a p-value of 0.

Electronic copy available at: https://ssrn.com/abstract=2684523

Algorithm	Out of Sample Paths Average	Stdev
LS	110.68	0.68
LSPI	110.95	0.66
Random Forest	110.91	0.55

Table 6: Convertible 2 Results

When comparing the LS and RF algorithms, we get a difference of the means of $110.91 - 110.68 = 0.23$. The standard deviation of the difference of the means is $\sqrt{\frac{std_{LS}^2 + std_{LSPI}^2}{2000}} = 0.019$. Therefore, our result is 11.8 standard deviations from 0, so we get a p-value of 0.

6.4 Example 3

This convertible bond is similar to the previous one, but it also has a call feature: during the last year of its life the bond can be called with a redemption amount of 120. The binomial model gives a price of 111.28. The binomial tree with the conversion, put and call boundaries is shown in Figure 4.

The performance of the algorithms is described in 7.

Algorithm	Out of Sample Paths Average	Stdev	Distance from Binomial
LS	110.91	0.68	0.37
LSPI	111.31	0.77	0.03
Random Forest	111.11	0.56	0.17

Table 7: Convertible 3 Results

7 Pricing a Real Convertible Bond

We now demonstrate how to use our methods in order to price a real convertible bond trading at the Tel Aviv Stock exchange. We chose to price one of the most traded convertible bonds at the exchange: Tower Bond Vav (security id: 1121193). On 2014/11/20 this bond had a market value of $993,589,000$ NIS and a trading volume of $5,291,841$ NIS per day.

This convertible bond has many interesting features. First, the conversion ratio is path dependent. This calls for Monte Carlo simulation, since a recombining tree will not be useful:

Electronic copy available at: https://ssrn.com/abstract=2684523

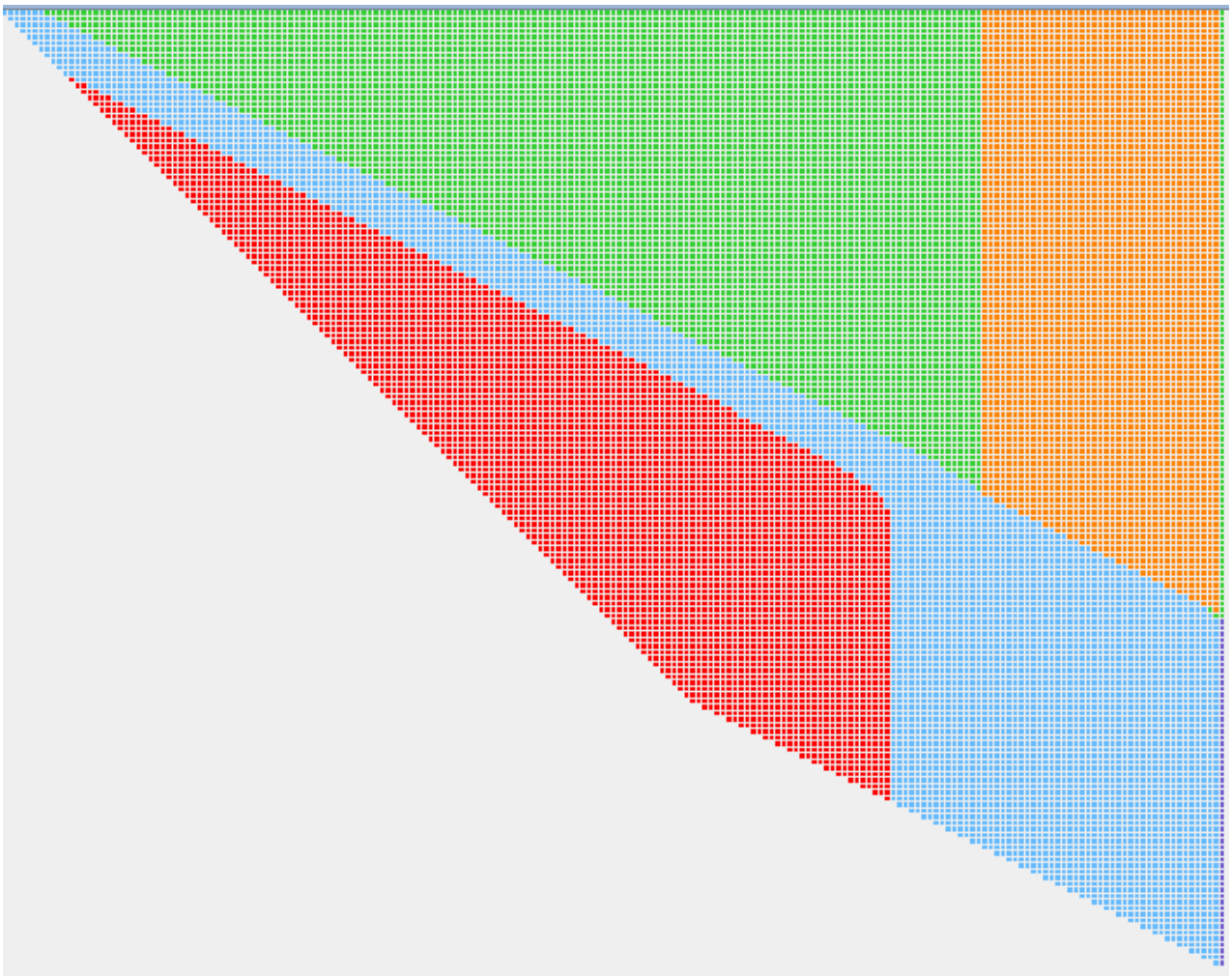

Figure 4: A binomial tree for convertible 3. Green nodes are the nodes where conversion is the optimal action. Red nodes are the nodes where put is the optimal action. Orange nodes are nodes where call is the best action for the issuer and the investor converts (forced conversion).

Electronic copy available at: https://ssrn.com/abstract=2684523

we need to know the path in order to be able to compute the conversion ratio. Second, the bond is indexed to the USD/ILS exchange rate. This means that the principal and interest are stated and payed in ILS, but they vary according to the USD/ILS exchange rate. Third, this bond has multiple payments of the principal: instead of returning the principal at the maturity date, the principal is returned in two payments. This feature is a challenge for non-tree models, but can be incorporated into a tree or a Monte Carlo model in a straightforward fashion.

7.1 Tower Securities

In order to build the model we need to look at three securities of Tower: stock (security id: 1082379), straight bond (Tower Bond Daled, security id: 1106608) and the Tower Vav convertible bond.

The stock initially had a face value of 1 NIS, but on 2012/08/01 stocks have been merged to have a face value of 15 NIS. In our model we will look at the merge-adjusted stock price for all dates (equivalently, we could have looked at the unadjusted price for all dates).

The straight bond is indexed to the consumer price index. It has an annual interest of 8% paid once a year and six principal payments. Its maturity date is 2016/12/31. The cash flows of this bond are described in Table 8.

date	**Principal Payments**	**Remaining Principal**	**Interest**	**Total Cash Flow**
2007/08/01		100		
2007/12/31		100	4.15	
2008/12/31		100	8	
2009/12/31		100	8	
2010/12/31		100	8	
2011/12/31	16.67	83.33	8	24.47
2012/12/31	16.67	66.67	6.67	23.33
2013/12/31	16.67	50	5.33	22
2014/12/31	16.67	33.33	4	20.67
2015/12/31	16.67	16.67	2.67	19.33
2016/12/31	16.67	0	1.33	18

Table 8: Tower Bond Daled Cash Flows

Electronic copy available at: https://ssrn.com/abstract=2684523

The convertible bond Tower Vav started trading on 2010/11/01. It has a face value of 100 NIS and is indexed (principal and coupons) to the USD/ILS exchange rate. There are two principal payments of 50 NIS each at the dates 2015/12/31 and 2016/12/31. A coupon of 7.8% annual interest rate is paid twice a year (3.9% each). The first coupon is proportional to the time the bond has been trading until the first payment (its value is 1.37%). This bond has no put or call options. There are two conversion periods: 2012/09/27 till 2015/12/15 and 2016/01/01 till 2016/12/15. Define the conversion ratio as the face value of the bond that you need in order to get one stock. For this bond the ratio equals to 1.2 multiplied by the average stock price in the 15 trading days prior to the first conversion period, with the bounds of 1 as the minimum conversion ratio and 97.5 as the maximum conversion ratio.

7.2 Default Model

Any realistic model for pricing convertible bonds should handle the possibility of default. We implement the default model described in De Spiegeleer and Schoutens (2011).

At each tree node we first decide whether there is a default or not and then, if there is no default, we decide whether the stock price goes up or down. This is described in Figure 5. A similar model can be used with a continuous model for the stock price, such as Brownian motion.

Suppose that credit default is a Poisson process with rate of arrival λ. Therefore, the probability of surviving the time period t without default is

$$p_s = e^{-\lambda \Delta t} \approx 1 - \lambda \Delta t$$

To estimate λ, we need a straight (non-convertible) bond of the same firm. We estimate λ from the *credit spread* of the straight bond. The credit spread is defined as the IRR of the bond minus the risk free rate. We use the notation specified in Table 9.

We can now write the following two equations for B

Electronic copy available at: https://ssrn.com/abstract=2684523

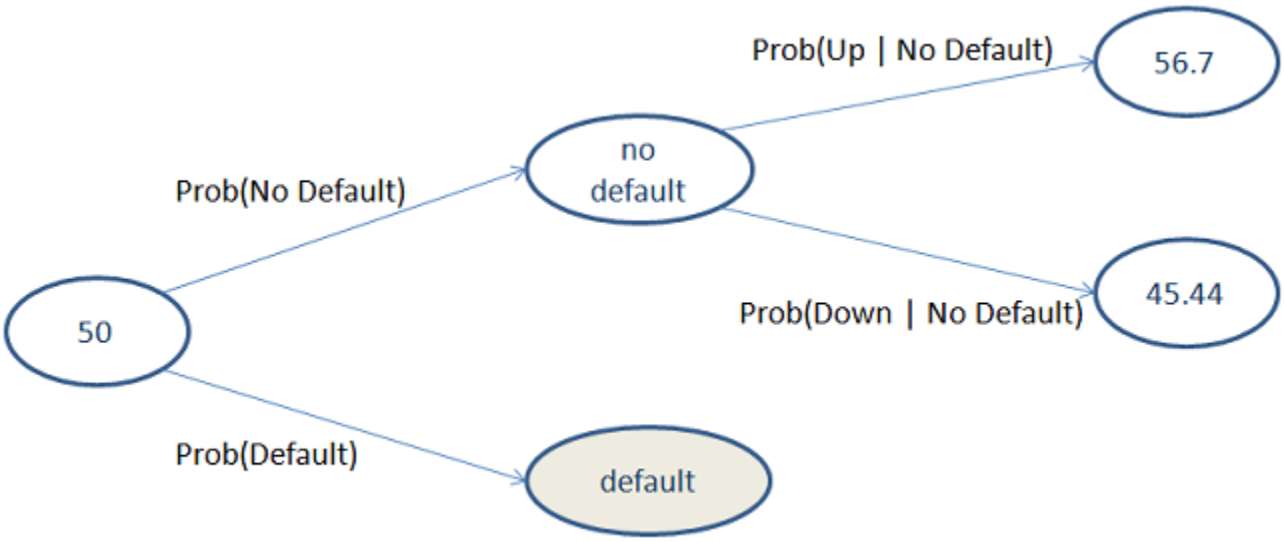

Figure 5: The Default Model.

Symbol	**Meaning**
F	face value
T	time of maturity
R	recovery rate
r	risk free rate
C	credit spread
B	price of the bond at time $T - \Delta t$
p_s	risk neutral probability of survival (no default) for time period Δt

Table 9: Default Model Notation

$$B = Fe^{-(r+C)\Delta t}$$

$$B = e^{-r\Delta t}(p_s F + (1 - p_s)RF)$$

Solving for λ we get

$$\lambda = \frac{C}{1 - R}$$

Therefore we can compute λ by extracting the credit spread from the straight bond.

Electronic copy available at: https://ssrn.com/abstract=2684523

7.2.1 Computing the Credit Spread

In order to extract the credit spread for some date, we first compute the IRR of Tower Bond Daled for that day. Since the bond is indexed to the CPI, we also need to consider inflation: we get inflation prediction for that date from the Bank of Israel. The cash flows of the bond are adjusted according to the already known CPI at the date of the pricing and the expected inflation rate. Denote the stated cash flow at year k by P_k and the inflation-adjusted cash flow by M_k. Suppose that the inflation prediction per year is given for each relevant year as r_1, r_2, ..., r_k. We first compute the inflation per day for each year as $d_i = (1 + r_i)^{1/252} - 1$. Denote by n_i the number of days for year i that are included between the pricing date and the date of the future cash flow. The inflation-adjusted cash flow is therefore

$$M_k = P_k * \frac{CPI}{CPI_{base}} * \prod_{i=1}^{k} (1 + d_i)^{n_i}.$$

These cash flows and the market price of the bond are then used for computing the IRR. The IRR is computed using a java implementation of the Newton method. The credit spread is then computed as IRR minus Makam (simple government bond) rate for that day.

7.3 Other Model Parameters

The volatility of the Tower stock is estimated as the historical volatility for one year before pricing date. The volatility of the USD/ILS exchange rate is estimated as the historical volatility for one year before pricing date.

The Tower Daled bond is subordinated to more senior Tower debt. According to Moody's report the recovery rate for subordinated bonds is 27%. Tower Vav is a subordinated convertible. According to Moody, the recovery rate in this case is smaller than for a subordinated straight bond. Therefore, we assume a 10% recovery rate.

Electronic copy available at: https://ssrn.com/abstract=2684523

7.4 Model Summary

To summarize, the model looks as follows. We use Monte Carlo with a discrete model for the price process and a default model as described above. Until the date when computation of the conversion ratio starts (2012/08/26) the tree can be recombining, since there is no need to know the path that got us to each node. From there the tree does not recombine. There are two factors in the model, these are the two variables that influence the bond value at each node: the stock price and the USD/ILS rate. Actually, each node in the complete tree can have 5 children as shown in Figure 6. The δt (the time between nodes) is one day. Therefore we have a total of 1498 time points from first trading day until maturity.

We run a Monte Carlo simulation with 1000 paths. For computing the early conversion policy we compare the LS and the Random Forest algorithms. It is known that for a convertible bond with no put and call options it only makes sense to convert early at the end of each conversion period. Therefore, we have two time points at which conversion should be considered: at the end of each conversion period. Since these are distinct time points, using the LSPI algorithm does not make sense here, because it requires multiple consecutive time points. We use 7 trees for the Random Forest. We use the following basis functions for both algorithms, where v denotes the conversion ratio

$$\phi_0 = 1$$

$$\phi_1 = S$$

$$\phi_2 = S^2$$

$$\phi_3 = S^3$$

$$\phi_4 = q$$

$$\phi_5 = v$$

Electronic copy available at: https://ssrn.com/abstract=2684523

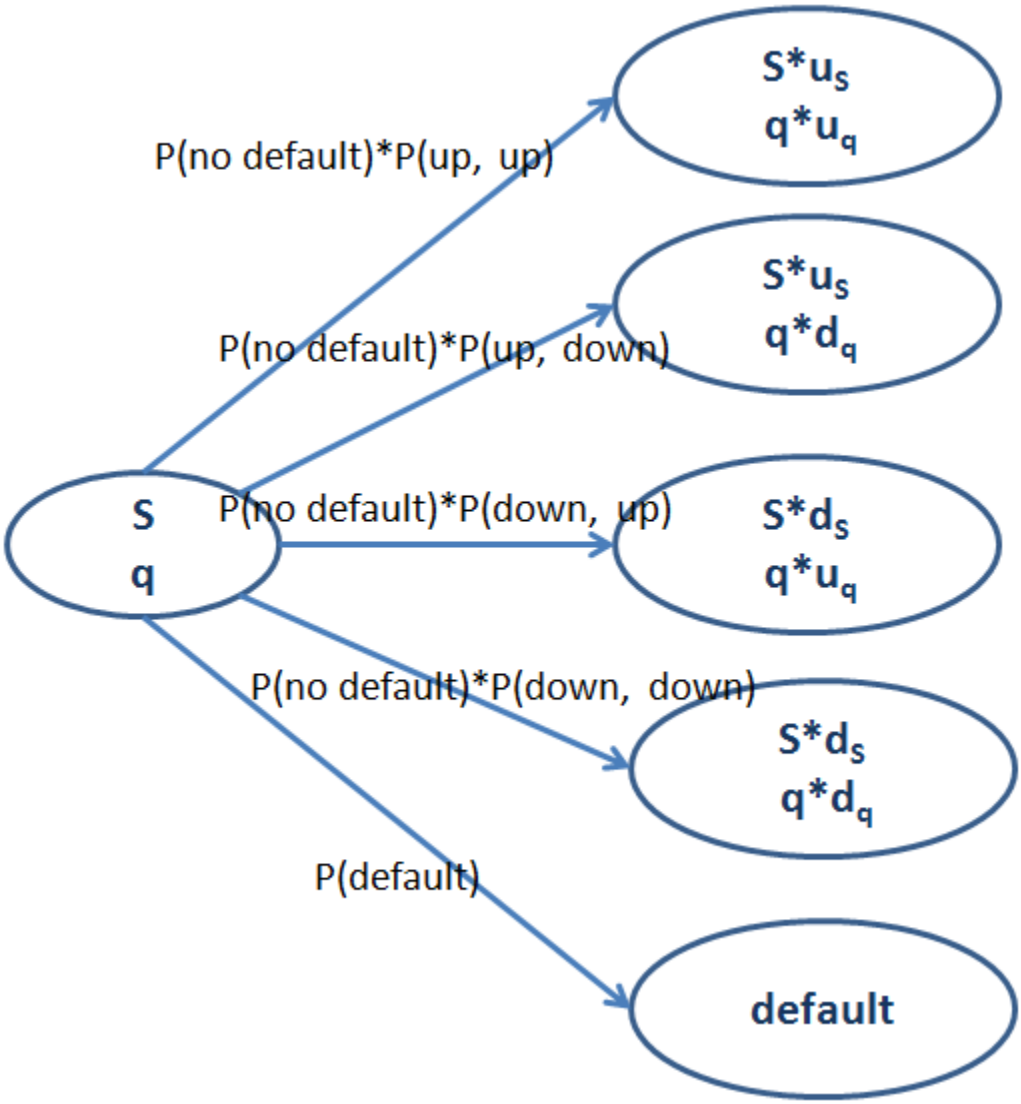

Figure 6: The Model for Pricing Tower Vav. The stock price is denoted by S and the USD/ILS exchange rate is denoted by q. The stock price can either go up by u_S or down by d_S and the exchange rate can either go up by u_q or down by d_q.

Electronic copy available at: https://ssrn.com/abstract=2684523

7.5 Tower Vav Results

The results for pricing the bond on its first trading date are summarized in Table 10. The results are an average of 100 runs.

Algorithm	**Out of Sample Price**	**Standard Deviation**
LS	106.32	4.13
Random Forest	106.71	4.2

Table 10: Tower Vav Results for First Trading Day (2010/11/01).

We see that the random forest gives a higher price than the LS, meaning that it found better conversion strategies. The in-sample price of the LS method can give us an upper bound for the bond price: the price is 107.16. The real bond price for that day is 97.9. It seems that the market undervalues the bond. Indeed the price goes up immediately and does not return to this level for a few months.

We now run the pricing model for the first four months of the bond, 1000 paths and 50 runs with Random Forest for each date. Our results along with the market bond price are presented in Figure 7. The difference between the prices ($|model - market|/market$) is presented in Figure 8. We see that the difference in model and market price is decreasing with the passage of time and in the last month that we looked at the average difference is 2.1%.

Electronic copy available at: https://ssrn.com/abstract=2684523

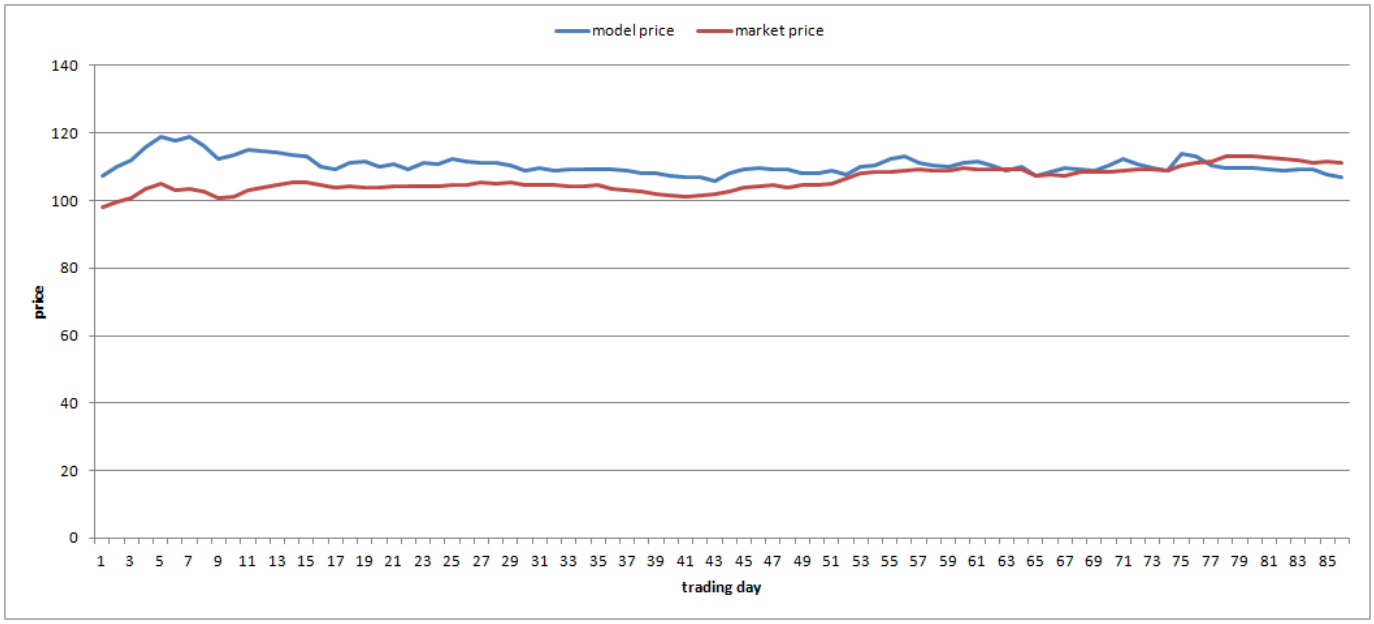

Figure 7: Model price and market price for Tower Vav for the first four trading months.

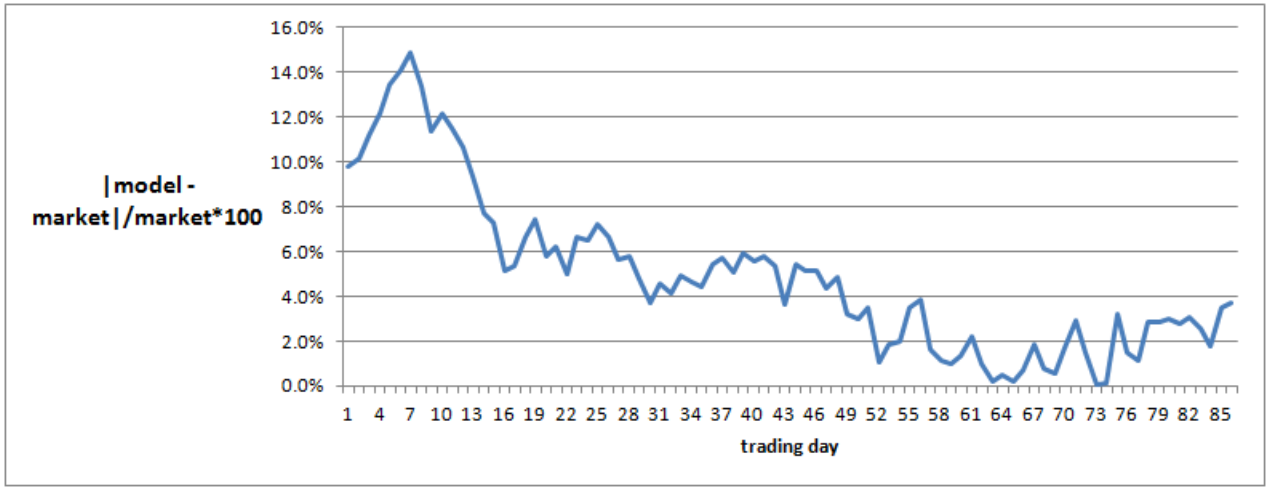

Figure 8: The percentage difference of model price and market price for Tower Vav for the first four trading months.

Electronic copy available at: https://ssrn.com/abstract=2684523

8 Conclusions

Monte Carlo simulation is a powerful method for pricing complex derivatives. The resulting models are flexible and tractable. When using Monte Carlo simulation in the context of derivatives with early exercise features, an algorithm for discovering exercise policies should be applied. We compared several such algorithms: the classical Longstaff and Schwartz method, reinforcement learning methods (which we extended to the case of convertible bonds) and the random forest method (which we applied for derivative pricing for the first time). Our results indicate that reinforcement learning methods outperform the LS method for both American put options and convertible bonds, the random forest algorithm also always outperforms LS and it outperforms reinforcement learning in most cases. Random forest is a simple and fast algorithm with excellent performance and it can be run in parallel on modern computers.

9 Future Research

Reinforcement learning and random forest methods show great promise in pricing derivatives with early exercise features. It will be interesting to apply these methods for other asset classes as well as run a large empirical study on real market data.

References

M. Ammann, A. Kind and C. Wilde. Simulation-Based Pricing of Convertible Bonds. Journal of Empirical Finance 15, 2008.

R. Bellman. A Markovian Decision Process. Journal of Mathematics and Mechanics 6, 1957.

S. Benninga. Financial Modeling. MIT, 2008.

C. Beveridge and M. Joshi. Monte Carlo Bounds for Game Options Including Convertible Bonds. Management Science vol. 57 no. 5, 2011.

Electronic copy available at: https://ssrn.com/abstract=2684523

C. Bishop. Pattern Recognition and Machine Learning. Springer, 2007.

F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. Journal of Political Economy, Vol. 81, No. 3, 1973.

P. P. Boyle. Options: A Monte Carlo Approach. Journal of Financial Economics, Volume 4, Issue 3, 1977.

L. Breiman. Random Forests. Machine Learning 45, Issue 1, 2001.

M. Broadie and P. Glasserman. Estimating Security Price Derivatives Using Simulation. Management Science, 42, 1996.

Bureau, A., Dupuis, J., Falls, K., Lunetta, K. L., Hayward, B., Keith, T. P., Van Eerdewegh, P. (2005). Identifying SNPs predictive of phenotype using random forests. Genetic epidemiology, 28(2), 171-182.

L. Busoniu, R. Babuska and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 38(2), 2008.

J. C. Cox, S. A. Ross and M. Rubinstein. Option Pricing: A Simplified Approach. Journal of Financial Economics 7, 1979.

D. R. Cutler, T. C. Edwards, K. H. Beard, A. Cutler, K.T. Hess, J. Gibson, J.J. Lawler. Random forests for classification in ecology. Ecology, 88(11), 2783-2792, 2007.

J. De Spiegeleer and W. Schoutens. The Handbook of Convertible Bonds. Wiley, 2011.

R. Geske. A Note on an Analytic Valuation Formula for Unprotected American Call Options on Stocks with Known Dividends. Journal of Financial Economics 7, 1979.

R. Geske. Comments on Whaley's Note. Journal of Financial Economics 9, 1981.

T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics, 2011.

Electronic copy available at: https://ssrn.com/abstract=2684523

E.G. Haug, J. Haug, A. Lewis. Back to Basics: a New Approach to the Discrete Dividend Problem. Wilmott Magazine, 2003.

S. L. Heston. A Closed-Form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The Review of Financial Studies 6(2), 1993.

R. A. Howard. Dynamic Programming and Markov Processes, The M.I.T. Press, 1960.

John C. Hull. Options, Futures, and Other Derivatives. Prentice Hall, 8th edition, 2011.

D. and R. Parr. Policy Iteration for Factored MDPs. In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence, 2000.

M. G. Lagoudakis and R. Parr. Least-Squares Policy Iteration. Journal of Machine Learning Research 4, 2003.

M. G. Lagoudakis and R. Parr. Value Function Approximation in Zero-Sum Markov Games. Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence, pages 283-292, 2002.

Y. Li, C. Szepesvari and D. Schuurmans, Learning Exercise Policies for American Options. International Conference on Artificial Intelligence and Statistics, 2009.

M. L. Littman. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In Proceedings of the Eleventh International Conference on Machine Learning, 1994.

F. A. Longstaff and E. S. Schwartz. Valuing American Options by Simulation: a Simple Least-Square Approach. Review of Financial Studies 14, 2001.

D. Lvov, A. B. Yigitbasioglu, N. El Bachir. Pricing Convertible Bonds by Simulation. Technical Report, ICMA Centre, University of Reading, December 2004.

K. Mishchenko, V. Mishchenko and A. Malyarenko. Adapted Downhill Simplex Method for Pricing Convertible Bonds. Research Reports MDH/IMA, Malardalen University, Department of Mathematics and Physics, 2007.

Electronic copy available at: https://ssrn.com/abstract=2684523

T. Mitchell. Machine Learning. McGraw Hill, 1997.

D. Opitz, R. Maclin. Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research 11: 169198, 1999.

H. Pang, A. Wang, S. Li. Pricing the Convertible Bonds Under Complex Call Trigger Condition with Longstaff and Schwartz Model. International Conference on Computer Science and Network Technology, 2011.

R. Polikar. Ensemble based systems in decision making. IEEE Circuits and Systems Magazine 6 (3): 2145, 2006.

W. B. Powell. Approximate Dynamic Programming: Solving the Curses of Dimensionality. Wiley Series in Probability and Statistics, 2007.

L. Rokach. Ensemble-based classifiers. Artificial Intelligence Review 33 (1-2): 139, 2010.

R. Roll. An Analytical Formula for Unprotected American Call Options on Stocks with Known Dividends. Journal of Financial Economics 5, 1977.

A. Statnikov, L. Wang, C. F. Aliferis. A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC bioinformatics 9(1), (2008).

R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.

J. N. Tsitsiklis and B. Van Roy. Regression Methods for Pricing Complex American-style Options. IEEE Transactions on Neural Networks 12(4), 2001.

R. E. Whaley. On the Valuation of American Call Options on Stocks with Known Dividends. Journal of Financial Economics 9, 1981.

M. Wiering (Editor), M. van Otterlo (Editor). Reinforcement Learning: State-of-the-Art (Adaptation, Learning, and Optimization). Springer, 2012.

Electronic copy available at: https://ssrn.com/abstract=2684523