An Analysis and Extension of the Longstaff–Schwartz Method for American Option Pricing

Jeppe Vanderhaegen & Asger Damgaard-Sørensen

Københavns Universitet

Økonomisk Institut

August 2025

Notes

It should be noted that ChatGPT has been used as a writing assistant throughout this assignment to improve coherence and consistency in the language.

Supervisor: Bertel Schjerning

Number of characters : ≈ 38000

1 Introduction

How can American put options be priced most effectively under stochastic asset dynamics? Valuing American-style derivatives is notoriously difficult due to the option of early exercise under stochastic asset dynamics. Traditional finite-difference and tree methods succumb to the curse of dimensionality as the number of underlying state variables grows. To address this Longstaff and Schwartz (2001) introduced the Least-Squares Monte Carlo (LSM) algorithm, which estimates the continuation value by regressing simulated payoffs onto a finite set of basis functions. This approach unites Monte Carlo's flexibility with the stability of orthogonal-polynomial expansions, enabling efficient pricing in moderate dimensions.

In this project, we start by examining the Longstaff-Schwartz algorithm in the single-asset setting by looking at how the choice of basis functions, the degree of polynomial expansion, and the simulation framework affect pricing accuracy. Specifically, we compare Laguerre, Hermite, and Chebyshev polynomials and number of degrees used. We further investigate the performance of the method under jump diffusion dynamics and explore the use of quasi-Monte Carlo sampling as an alternative to standard Monte Carlo. Through these experiments, we aim to better understand the strengths and limitations of the LSM approach in one dimension.

We then extend our analysis to the multi-asset setting by considering American put options on up to five underlying assets. While Monte Carlo methods mitigate the curse of dimensionality in the simulation stage, the regression step in LSM still suffers from exponential growth in the number of basis terms when using tensorized polynomial expansions. This makes it increasingly difficult to accurately approximate the continuation value in higher dimensions. To address this issue, we introduce a neural network–based approach that replaces fixed polynomial bases with a compact multilayer perceptron.

Moving on, Section 2 formalizes the American-option stopping problem. Section 3 reviews the one-asset LSM algorithm, while Section 4 extends it to different simulation and multiple assets. Section 5 introduces our neural-network approach. Section 6 describes our experimental setup, and Section 7 presents the results leading into a discussion in Section 8.

We conclude that the Longstaff-Schwartz method yields accurate prices when using Laguerre polynomials of degree 3. Incorporating Merton's jump diffusion improves realism and reduces computation time, with only a slight increase in option value. In higher dimensions, polynomial regression becomes infeasible and convergence breaks down. A neural network approach failed to deliver reliable prices, indicating a need for further investigation.

2 The American option problem

In order to answer the problem at hand, we will follow the approach outlined in Glasserman (2003) for American options. We will formulate the continuous time American option pricing problem by letting $U(t)$, $0 \leq t \leq T$, represent the discounted payoff from exercising at time

t, and with $\mathcal{T}$ being some stopping times a long the way on the interval $[0, 1]$. Based on this, the problem is to find the optimal discounted payoff:

$$\sup_{\tau \in \mathcal{T}} E^Q[U(\tau)] \tag{1}$$

Here Q defines the risk-neutral measure, under which we will price our options following Black and Scholes (1973). Furthermore, we assume that the underlying stock will not pay out any dividends, and that the underlying stock follows a Geometric Brownian Motion, given as:

$$dS_t = rS_t dt + \sigma S_t dW_t^Q \tag{2}$$

Where r is the risk-free rate and dW_t is a Brownian Motion. Therefore, we can formulate our problem as:

$$\sup_{\tau \in \mathcal{T}} E^Q[e^{-r\tau}(K - S(\tau))^+] \tag{3}$$

We can solve our problem by finding the optimal stopping in the form of:

$$\tau^\star = \inf\{t \geq 0 : S(t) \leq b^\star(t)\} \tag{4}$$

Here $b^\star(t)$ is the optimal exercise boundary. This means, that the option holder should exercise the option at the lowest t, conditioned on the stock price being below the exercise boundary. Thus, the early-exercise boundary defines the decision policy that optimally solves the American-option stopping problem. Moreover, we have written the payoff in equation [3] as $(K - S(\tau))^+$. This means that exercising an out-of-the-money option will give a payoff of zero instead of a negative payoff. This will allow us to include the possibility of the option to expire worthless at maturity. Therefore, we may take payoff throughout this paper as non-negative.

Backward Induction

We consider an American put option which expires at T and look at it as a finite horizon optimal stopping problem as described in Adda and Cooper (2002). We discretize the life of the option into $n + 1$ dates with $0 = t_0 < t_1 < \cdots < t_n = T$. Technically, this turns the American into a Bermudan option, but taking $n \nearrow \infty$, we can make the best approximation for the American option, which can be exercised continuously. Continuing on, we denote the discounted asset price under the risk-neutral measure, Q, and define the payoff function as:

$$H(s) = (K - s)^+ \tag{5}$$

The value function $V_i(s)$ at date t_i satisfies the Snell-envelope recursion from

$$V_n(s) = H(s) \tag{6}$$

$$V_i(s) = \max\Big\{ H(s),\ e^{-r\,\Delta t} E^Q\big[V_{t_{i+1}}(S_{t_{i+1}}) \mid S_{t_i} = s\big]\Big\}, \quad i = n-1, \ldots, 0 \tag{7}$$

This yields the following function:

$$V_i = \max\Big\{H(s), C_i(s)\Big\} \tag{8}$$

Where $H(s)$ is the immediate payoff of exercising, and $C_i(s)$ is continuing holding on to the option, which is given by:

$$C_i(s) = e^{-r\Delta t} E^Q \left[V_{i+1}(S_{t_{i+1}}) \mid S_{t_i} = s \right] \tag{9}$$

Whenever $H(s) \geq C_i(s)$ the exercise of the option will be optimal. In practice, we will estimate the continuation value $C_i(s)$ by regressing discounted payoffs.

3 The LSM

In order to approximate the Snell-envelope recursion from Section 2, Longstaff and Schwartz (2001) developed a regression-based approach, which estimates the continuation value on Monte Carlo simulated paths. The main idea is to treat the conditional expectation

$$C_i(s) = e^{-r\Delta t} E^Q \left[V_{i+1}(S_{t_{i+1}}) | S_t = s \right] \tag{10}$$

as the projection of the discounted future payoffs onto a finite set of basis functions of the current state.

3.1 The algorithm

We start by simulating M independent paths of the underlying asset price under the risk-neutral measure. This will yield $\left\{ S^m_{t_0}, S^m_{t_1}, ..., S^m_{t_n} \right\}$ for $m = 1, 2, ..M$. Then for each path m, at the last exercise date $t_n = T$, the holder of the option knows exactly, how much the option would receive if they waited until maturity, this is given as:

$$V^m_n = H(S^m_{t_n}) = (K - S^m_{t_n})^+ \tag{11}$$

We will then start at $i = n - 1$, and compute the discounted continuation value for each path m:

$$Y^m_{i+1} = e^{-r\Delta t} V^m_{i+1} \tag{12}$$

We restricted us to paths where the immediate exercise yields a positive payoff, i.e wherever $H(S^m_{t_i}) > 0$. This follows from equation [3], that we only look at positive payoffs, and it will lower the amount of paths, we look at. For every path, which satisfies $H(S^m_{t_i}) > 0$, we denote them as the subset $\mathcal{M}$.

We then fit a linear model in order to predict the discounted future cash-flow Y^m_{i+1} based on the current state $S^m_{t_i}$. Here we choose a set of basis functions φ_k, on which we want to capture as much as possible. The linear model will be given as:

$$Y^m_{t_i} \approx \sum_{k=0}^{K-1} \beta_{i,k} \phi_k(S^m_{t_i}) \quad for\ m \in \mathcal{M} \tag{13}$$

The choice of basis functions, will be discussed in Section 3.2. For each $m \in \mathcal{M}$, we then estimate the continuation value:

$$\hat{C}^m_i = \sum_{k=0}^{K-1} \beta_{i,k} \varphi_k(S^m_{t_i}) \tag{14}$$

When the continuation value has been determined, we compare it with the immediate payoff $H(S_{t_i}^m)$. If we have $H(S_{t_i}^m) > \hat{C}_i^m$, then we set $V_i^m = H$ and record that path m has been exercised at t_i. If this is not the case, we set $V_i^m = Y_{i+1}^m$ and continue with the backward induction.

When reaching $i = 0$, each simulated path m will yield a best time to stop, τ_m. At path m, at τ_m we receive a single discounted cash-flow for that path. Because the risk-neutral option price is the expected value of this cash-flow, we approximate it by the average over the M paths simulated. This is given as:

$$\hat{V}_0 = \frac{1}{M} \sum_{m=1}^{M} V_{V_{\tau_m}^m} \tag{15}$$

Which is the price of the option at $t = 0$. The Algorithm used given en the following pseudo-code:

Algorithm 1 Longstaff-Schwartz

Input: Simulated paths $S_{t_i}^m$ for $i = 0, \ldots, n$, $m = 1, \ldots, M$
$V_n^m \leftarrow H(S_{t_n}^m)$ ▷ Terminal payoff
for $i = n - 1$ **downto** 0 **do**
    $\mathcal{M}_i \leftarrow \{m : H(S_{t_i}^m) > 0\}$ ▷ Set of in-the-money paths at time t_i
    if $|\mathcal{M}_i| > 0$ **then**
        $X_m \leftarrow [\phi_0(S_{t_i}^m), \ldots, \phi_{K-1}(S_{t_i}^m)]$ for $m \in \mathcal{M}_i$
        $Y_{i+1}^m \leftarrow e^{-r\Delta t} \cdot V_{i+1}^m$ for $m \in \mathcal{M}_i$
        Estimate $\beta_{i,k}$ by regressing Y_{i+1}^m on X_m
        for $m = 1$ to M **do**
            $\widehat{C}_i^m \leftarrow \sum_{k=0}^{K-1} \beta_{i,k} \cdot \phi_k(S_{t_i}^m)$
            if $H(S_{t_i}^m) \geq \widehat{C}_i^m$ **then**
                $V_i^m \leftarrow H(S_{t_i}^m)$
                $\tau^m \leftarrow t_i$
            else
                $V_i^m \leftarrow e^{-r\Delta t} \cdot V_{i+1}^m$
$\widehat{V}_0 \leftarrow \frac{1}{M} \sum_{m=1}^{M} V_{\tau^m}^m$
return $\widehat{V}_0$

3.2 Basis functions

As noted above, LSM estimates the conditional expectation of future continuation values by regressing discounted cash–flows on a set of basis functions. We have chosen to work with the Laguerre, Hermite and Chebyshev polynomials. These are all the a special case of the orthogonal polynomials. These polynomials, $[P_n(x)]_{n=0}^{\infty}$, are orthogonal with respect to a weight function $w(x)$, such that:

$$\int_a^b P_n(x)P_m(x)w(x)dx = 0, \quad \text{whenever} \quad n \neq m \tag{16}$$

The orthogonality ensures that when we project a function, fx. the discounted future pay-off, onto the span of the first d polynomials, we arrive at coefficients, that are unique and numerically stable.

Laguerre Polynomials
The generalized Laguerre polynomials $L_n^{(\alpha)}(x)$ are orthogonal on $[0,\infty)$ under the weight $w(x) = x^\alpha e^{-x}$:

$$\int_0^\infty L_n^{(\alpha)}(x)\, L_m^{(\alpha)}(x)\, x^\alpha e^{-x}\, dx = \frac{\Gamma(n+\alpha+1)}{n!}\, \delta_{nm}.$$

They satisfy

$$(n+1)L_{n+1}^{(\alpha)}(x) = (2n+\alpha+1-x)\, L_n^{(\alpha)}(x) - (n+\alpha)\, L_{n-1}^{(\alpha)}(x),$$

Laguerre polynomials naturally handle non-negative prices and right-skewed tails, which makes them very handy when it comes to asset prices. If we set $\alpha = 0$, then we will arrive at the first five polynomials, we will be using.

$$L_0(x) = 1, \quad L_1(x) = 1 - x, \quad L_2(x) = \frac{1}{2}(x^2 - 4x + 2), \quad L_3(x) = \frac{1}{6}(-x^3 + 9x^2 - 18x + 6)$$

$$L_4(x) = \frac{1}{24}(x^4 - 16x^3 + 72x^2 - 96x + 24), \quad L_5(x) = \frac{1}{120}(-x^5 + 25x^4 - 200x^3 + 600x^2 - 600x + 120)$$

Hermite Polynomials
The probabilists' Hermite polynomials $H_n(x)$ obey

$$\int_{-\infty}^{\infty} H_n(x)\, H_m(x)\, \frac{e^{-x^2/2}}{\sqrt{2\pi}}\, dx = n!\, \delta_{nm},$$

with recurrence

$$H_{n+1}(x) = x\, H_n(x) - n\, H_{n-1}(x).$$

They perform well when the underlying is approximately Gaussian. The ones we will be using are given as:

$$H_0(x) = 1, \quad H_1(x) = x, \quad H_2(x) = x^2 - 1, \quad H_3(x) = x^3 - 3x, \quad H_4(x) = x^4 - 6x^2 + 3,$$

$$H_5(x) = x^5 - 10x^3 + 15x$$

Chebyshev Polynomials
The first-kind Chebyshev polynomials $T_n(x)$ are orthogonal on $[-1,1]$ with weight $(1-x^2)^{-1/2}$:

$$\int_{-1}^{1} T_n(x)\,T_m(x)\,\frac{dx}{\sqrt{1-x^2}} = \begin{cases} 0, & n \neq m, \\ \pi, & n = m = 0, \\ \pi/2, & n = m \neq 0, \end{cases}$$

and satisfy

$$T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x).$$

Since the Chebyshev polynomials are only stable on the $[-1,1]$ interval, we must first map the stock price onto it. We rescale with the help of the following function:

$$\tilde{S} = \frac{2S_t - (S_{min} + S_{max})}{S_{max} - S_{min}} \tag{17}$$

The polynomials, we will be using, where we set $u = 0$, are given as the following:

$$T_0(u) = 1, \quad T_1(u) = u, \quad T_2(u) = 2u^2 - 1, \quad T_3(u) = 4u^3 - 3u, \quad T_4(u) = 8u^4 - 8u^2 + 1,$$

$$T_5(u) = 16u^5 - 20u^3 + 5u$$

Once a family and degree d are chosen, the computed $\hat{C}_i(s)$ enters directly into the backward induction introduced above.

3.3 Simulation: Monte Carlo

Accurate estimation of continuation values depends on the quality of the simulated asset price paths. We apply both Monte Carlo simulation methods to generate paths under the risk-neutral measure, following the numerical integration methods in Judd (1998). Monte Carlo simulation uses pseudo-random normal draws to discretize geometric Brownian motion. Although it converges slowly at rate $O(M^{-1/2})$, it remains the standard approach due to its generality and simplicity. In the LSM context, it allows for unbiased estimation of expected payoffs conditional on simulated in-the-money states. The generated paths form the basis for backward induction. The simulation method interacts with the choice of basis functions by determining the quality of the regression input, which ultimately affects the accuracy of the estimated American option price.

Simulated Monte Carlo paths with geometric brownian motion and antithetic variance are shown in Figure 1

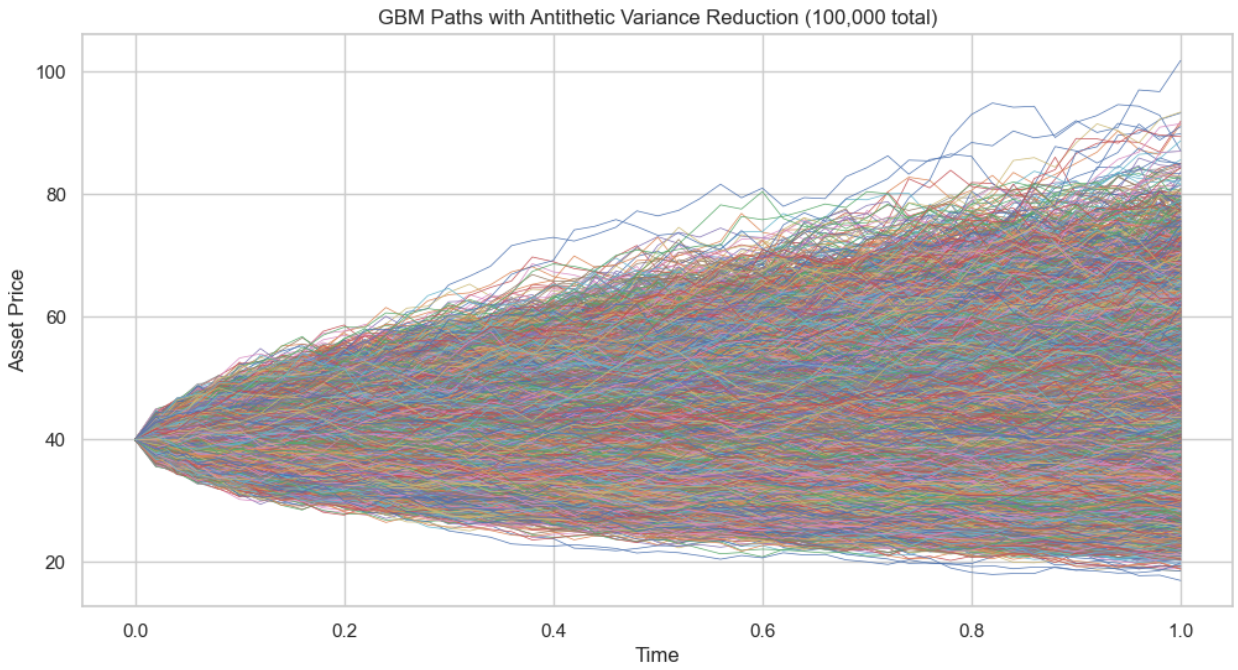
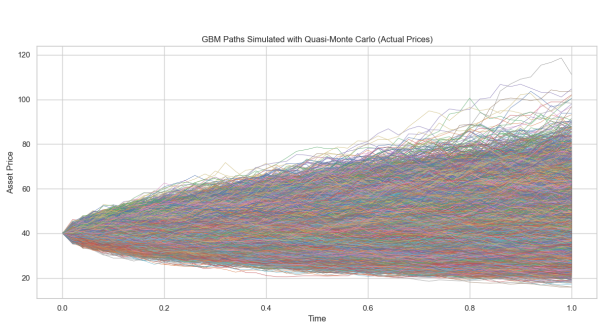
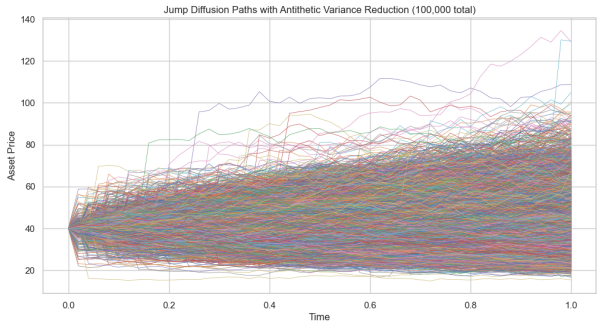


Figure 1: Simulated Monte Carlo paths with geometric brownian motion and antithetic variance



(a) Quasi-Monte Carlo



(b) MC with Merton jump diffusion

Figure 2: Simulations with QMC and MC with Jump Diffusion

4 Extending the model

4.1 Adding Quasi-Monte Carlo and Jump Difussion

Quasi-Monte Carlo simulation replaces pseudo-random draws with low-discrepancy Sobol sequences to provide more uniform coverage of the sample space. When the continuation value is sufficiently smooth, QMC can achieve faster convergence than standard Monte Carlo. This improved sampling uniformity reduces variance in the regression inputs, which can lead to more stable estimation of the early exercise strategy within the LSM algorithm.

A natural extension of the Longstaff and Schwartz simulation-based approach to American option pricing is to incorporate the Merton (1976) jump diffusion model directly into the simulation of the underlying asset dynamics. Although Adda and Cooper (2002) primarily focus on empirical applications, their theoretical formulation of stochastic dynamic programming provides a suitable foundation for modeling state transitions driven by discontinuous shocks. In this setting, the asset price follows a Merton jump diffusion process, combining continuous Brownian motion with rare Poisson-distributed jumps.

Although the theoretical state space of the underlying asset is unbounded, the simulation-based approach in Longstaff and Schwartz operates on a discretized domain with bounded support. As a result, the value function remains bounded in practice. Furthermore, the Snell-envelope recursion used to define the option value corresponds to a Bellman recursion, and under standard assumptions including discounting, the Bellman operator satisfies the contraction property. These conditions ensure existence and uniqueness of the value function, as guaranteed by Theorem 3 in Adda and Cooper (2002). Both types of simulations are shown in Figure 2

4.2 Multiple Assets

We now want to extend the model to be able to handle n assets. Since we follow real market practices, we will have that these underlying assets are correlated. Yet for some simplicity, we will weigh the basket of assets equally, thus giving us $\frac{1}{n}$-weights for all assets. Now instead of having a singular underlying asset, we introduce a vector of assets given as:

$$\mathbf{S}_t = (S_t^{(1)}, \ldots, S_t^{(n)}) \tag{18}$$

where each S_t^i follows the underlying stock price introduced in equation [2]. Moreover, we introduce correlation in order to better model the real financial markets. The correlation is given as:

$$\langle dW_t^{(i),Q}, dW_t^{(j),Q} \rangle = \rho_{ij}\, dt, \quad j = 1, \ldots, n \tag{19}$$

Here ρ_{ij} encodes pairwise correlations between the underlying assets. If $\mathbf{w}$ denotes the fixed asset weights and K the strike, the instantaneous payoff at exercise becomes

$$H(\mathbf{S}_t) = \left(\mathbf{w}^\top \mathbf{S}_t - K\right)^+ \tag{20}$$

Following the steps taken in equation [5] - [9], the multi-asset Snell-envelope then becomes

$$V_n(\mathbf{S}_t) = H(\mathbf{S}_t) \tag{21}$$

$$V_i(\mathbf{S}_t) = \max\Big\{H(\mathbf{S}_t), e^{-r\Delta t}E^Q\big[V_{i+1}(\mathbf{S}_{t_{i+1}}) \;\big|\; \mathbf{S}_{t_i} = \mathbf{S}\big]\Big\}, \quad i = n-1, \ldots, 0 \tag{22}$$

This yields a similar function to the one asset case:

$$V_i(\mathbf{S}_t) = \max\big\{H(\mathbf{S}_t), C_i(\mathbf{S}_t)\big\}. \tag{23}$$

Where again the $H(\mathbf{S}_t)$ is the immediate payoff of exercising, and $C_i(\mathbf{S}_t)$ is continuing holding on to the option, which is given as:

$$C_i(\mathbf{S}) = e^{-r\,\Delta t}\,E^Q\big[V_{i+1}(\mathbf{S}_{t_{i+1}}) \;\big|\; \mathbf{S}_{t_i} = \mathbf{S}\big] \tag{24}$$

4.3 Cholesky correlation matrix

In a multi-asset simulation, we must generate Brownian increments which preserve the correlations between the assets, as only few asset in the financial sphere are uncorrelated. A standard way to do this is with the help of a Cholesky decomposition of the correlation matrix. In order to generate M correlated asset trajectories, we begin by assembling the $n \times n$ correlation matrix

$$\mathbf{R} = \big(\rho_{ij}\big)_{i,j=1}^{n} \tag{25}$$

where each entry ρ_{ij} specifies the instantaneous correlation between the Brownian drivers of assets i and j. While $\rho_{ii} = 1$. We then perform a Cholesky decomposition

$$\mathbf{R} = \mathbf{L}\mathbf{L}^\top \tag{26}$$

where $\mathbf{L}$ is lower triangular. At each time step Δt, we draw an independent standard-normal vector $Z \sim N(0, I_n)$ and set

$$\Delta\mathbf{W} = \sqrt{\Delta t}\mathbf{L}Z \tag{27}$$

to obtain the lower-triangular matrix $\mathbf{L}$. During simulation, independent standard-normal draws $Z \sim N(0, I_n)$ are transformed into correlated increments via $\mathbf{L}Z$, ensuring that the resulting asset paths exhibit the prescribed pairwise correlations.

4.4 Tensor-product basis functions

When our price depends on multiple assets, simple one-dimensional polynomials will no longer suffice. Instead, we build basis functions by taking products of single-asset polynomials, which should capture both individuals and joint price effects. In order to do this, we first choose a maximum polynomial degree d and consider all n-tuples of nonnegative integers $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ with each α_j between 0 and d. For each such multi-index $\boldsymbol{\alpha}$, we define a basis function

$$\varphi_{\boldsymbol{\alpha}}(\mathbf{S}) = P_{\alpha_1}(\tilde{S}^{(1)}) \times \cdots \times P_{\alpha_n}(\tilde{S}^{(n)}) = \prod_{j=1}^{n} P_{\alpha_j}\big(\tilde{S}^{(j)}\big) \tag{28}$$

where P_{α_j} is the one-dimensional polynomial of degree α_j (for example, a Chebyshev, Laguerre, or Hermite polynomial), and $\tilde{S}^{(j)}$ denotes the linearly rescaled asset price $S^{(j)}$. By taking all combinations of degrees up to d, we obtain $(d+1)^n$ distinct basis functions, which capture both individual asset effects and cross-asset interactions in the continuation-value regression. It shall be noted, that increasing the amount of degrees and of course assets, the computation will be more intensive and will potentially take longer times to compute.

5 Introducing Neural Network

Instead of regressing the continuation value onto a large set of polynomials, we can train a small neural network to learn this mapping directly from the simulated data. A neural network should in theory be better to discover non-linear patterns in our simulated paths. Where our polynomial regression may miss some interactions because of the potential high degree of polynomials needed to extract some interactions. A network with activation functions can adapt to our true continuation-value.

We will use a multilayer neural network as Becker et al. (2020) and Pommelgård Lind (2020), where at each exercise date t_i, we can replace the linear regression step

$$C_i(\mathbf{S}) = e^{-r\Delta t} E^Q\big[V_{i+1}(\mathbf{S}) \mid \mathbf{S}_{t_i} = \mathbf{S}\big] \approx \sum_{\boldsymbol{\alpha}} \beta_{i,\boldsymbol{\alpha}}\, \varphi_{\boldsymbol{\alpha}}(\mathbf{S}) \tag{29}$$

with a feedforward network $f_{\theta_i}(\mathbf{S})$:

$$\hat{C}_i(\mathbf{S}) = f^{\theta i}(\mathbf{S}) \tag{30}$$

where $f^{\theta_i} : \mathbb{R}^n \to \mathbb{R}$. Our network consists of one input layer, which is our vector of n asset prices S_t. Two hidden layers both consisting of a number of neurons. These contain the parameters which determine how the input is mapped from one layer to the next. We will denote the number of neurons in the ith layer by m^i. Therefore, for our input layer, we have:

$$m^1 = k, \quad \text{for } x = (x_1, x_2, ..., x_k) \tag{31}$$

i.e the number of neurons in the first layer will equal the dimensions of our input vector. The entire network is build as follows:

$$F(x) = f_1 \circ f_2 \circ \cdots \circ f_{n+1} \tag{32}$$

$$f_{\theta_i} : \mathbb{R}^{m^{i-1}} \longrightarrow \mathbb{R}^{m^i} \tag{33}$$

where n is the number of hidden layers. Each mapping is given by

$$f_{\theta_i}(x) = g\big(W^T x + w_0\big) \tag{34}$$

where

$$x \in \mathbb{R}^{m^{i-1}}, \quad W \in \mathbb{R}^{m^{i-1} \times m^i}, \quad w_0 \in \mathbb{R}^{m^i}.$$

Here, the vectors $W_j \in \mathbb{R}^{m^{i-1}}$ and scalars $w_{0,j} \in \mathbb{R}$ are the parameters of neuron j. The matrix W contains the weights, and w_0 is the bias vector. Moreover, g is an *activation function* which introduces nonlinearity into our mapping, which enables the network to capture complex relationships. In this project, we will use the Rectified Linear Unit (ReLU), defined by

$$g(x) = \max(0, x). \tag{35}$$

This is one of the most common activation functions. Figure 3 illustrates an MLP with two hidden layers.

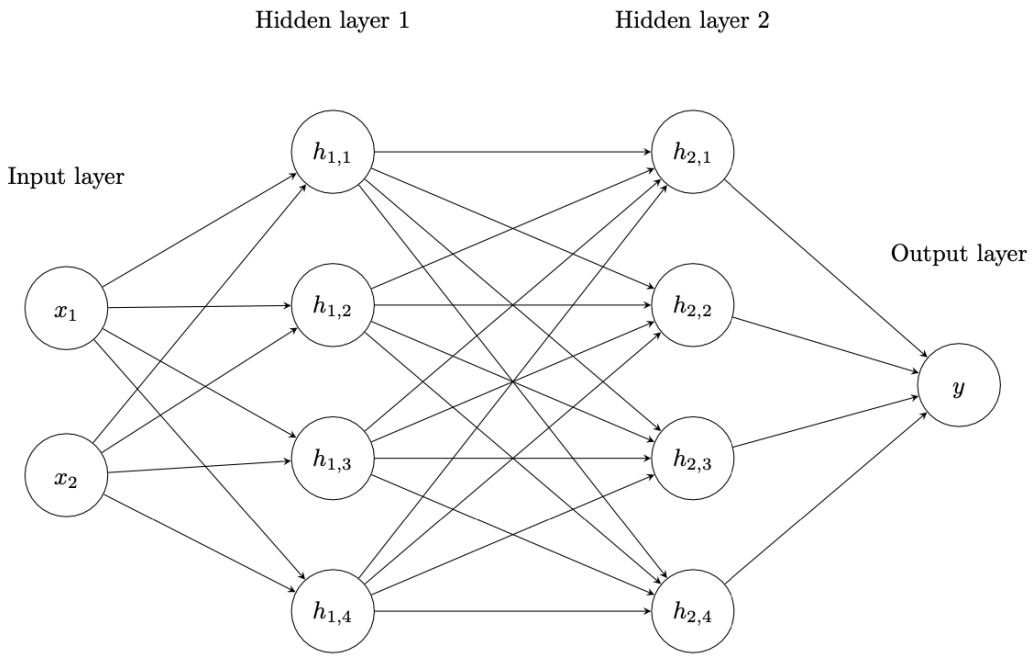


Figure 3: Neural network with two hidden layers

The network is trained solely on the paths where the option is in-the-money at time t_i. We denote the set of paths as:

$$\mathcal{M}_i = \big\{m : H(\mathbf{S}_{t_i}^m) > 0\big\}, \quad H(\mathbf{S}_{t_i}^m) = \big(\mathbf{w}^\top \mathbf{S}_{t_i}^m - K\big)^+$$

For each $m \in \mathcal{M}_i$, the target value is the discounted next-step payoff

$$Y^m = e^{-r\,\Delta t} V_{i+1}^m \tag{36}$$

The network parameters θ_i are found by minimizing the mean squared error:

$$\theta_i^* = \arg\min_{\theta_i} \sum_{m \in \mathcal{M}_i} \big[f_{\theta_i}(\mathbf{S}_{t_i}^m) - Y^m\big]^2 \tag{37}$$

Here $f_{\theta_i}(\mathbf{S}_{t_i}^m)$ denotes the true value and Y^m is the networks estimate. In order to minimize θ_i, we will make use of the Adam algorithm. Adam is well suited for large datasets, because it estimates gradients using small batches of data, which is far less heavy computationally than computing the gradient over the entire dataset at once. Since our dataset consists of a few hundred thousands data points, the mini batch optimization becomes essential. Moreover, the Adam algorithm features an adaptive learning rate, η, which adjust according to the curvature of the cost function. The parameter will update at each iteration with:

$$w_{new} = w_{old} - \eta \Delta \theta_i(w_{old})$$

Once trained, the network's prediction $f_{\theta_i}(\mathbf{S}_{t_i})$ serves as our continuation-value estimate. We then compare it to the immediate exercise payoff $\big(\mathbf{w}^\top \mathbf{S}_{t_i} - K\big)^+$ to decide whether to exercise or continue just as with the standard LSM-method.

6 Our experiment

The goal of this paper is to efficiently extend the LSM model and price options under different circumstances. In order to do this, we must have something to test our models against, to see if they work. We will start with the baseline.

Baseline
All our models will be run on the same coefficients. For our one-dimensional LSM we will be using the same parameter values as in the original paper, Longstaff and Schwartz (2001). We will be using a stock, S and strike price, K at 40, the risk-free rate, r at 0.06, volatility, σ at 0.2, time to maturity, T at 1. Lastly, we will use $N = 50$ timestep and $m = 100,000$ paths. Since this is exactly the same metrics as LSM, we will use their calculated option price as our baseline and we will use the European option price as a further check.

When extending the model, we price a basket of options, just like a portfolio of options. Therefore, we need to weigh them, which we have chosen to do equally, also known as a naive portfolio. Therefore, our weights are given as $w = \frac{1}{n}$, where n is the number of assets. Furthermore, we have chosen to incorporate correlations between the assets in our multi-dimensional setting in order to come somewhat closer to reality. Therefore, we have introduced pairwise correlations between the assets, which we have chosen as $\rho = 0.2$. In our multi-dimensional setting, we do not have a reference price for our basket of options. Therefore, we will rely on European price for using the same parameters. Lastly, for the Neural Network, we will use batches at the size of 512, learning rate, $\eta = 0.001$ and a two hidden layer network, built as:

$$m^1 \times m^2 \times m^3 \times m^4 = 1 \times 40 \times 40 \times 1$$

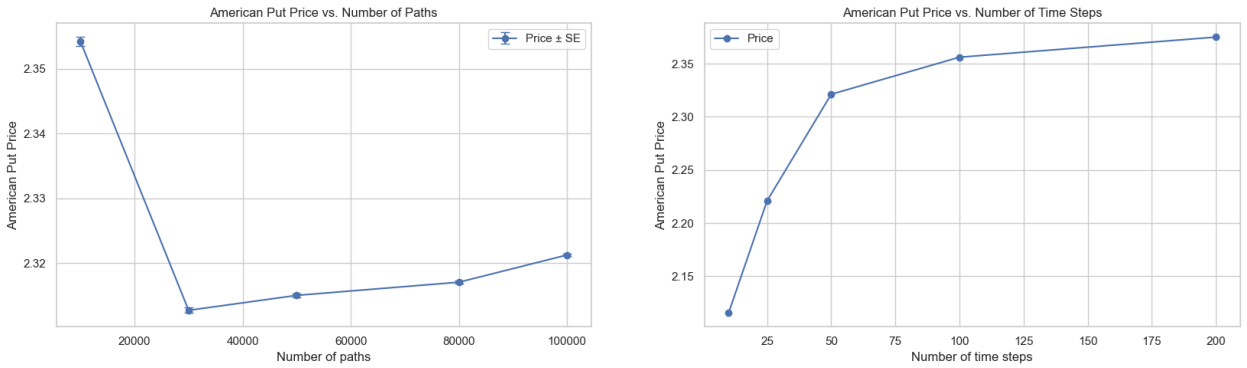
The metrics
In order to quantify our work, we must have some testable metrics. First and foremost, the price of the option must be above the European counterpart. Should this not be the case, then we have priced it wrong. Secondly, we want to test the time. How long does it take for the model yield a price. We want to test for time, as we want to still have some efficiency in our pricing of options, and we do not want our model to having to run for hours in order to come up with a singular number. It shall be noted, that small discrepancies in the time count may occur since other programs may interfere with the CPU/memory while running our scripts.

Lastly, we must test for convergence. We will test for the number of polynomials degrees which are needed for a consistent price. Here we will test from $d = [1, 2, 3, 4, 5]$, the number of paths from $m = [10,000, 25,000, 50,000, 100,000, 200,000]$ and the number of time steps from $N = [10, 25, 50, 100, 200]$. All in order to find a numerically stable price. We will use the change in price to determine, when we have reached a numerically stable price, i.e. if the change in price is negligible, then we consider the price as good.

7 Results

7.1 Singular dimensions

7.1.1 Convergence of the Longstaff–Schwartz Method



(a) Number of paths vs price

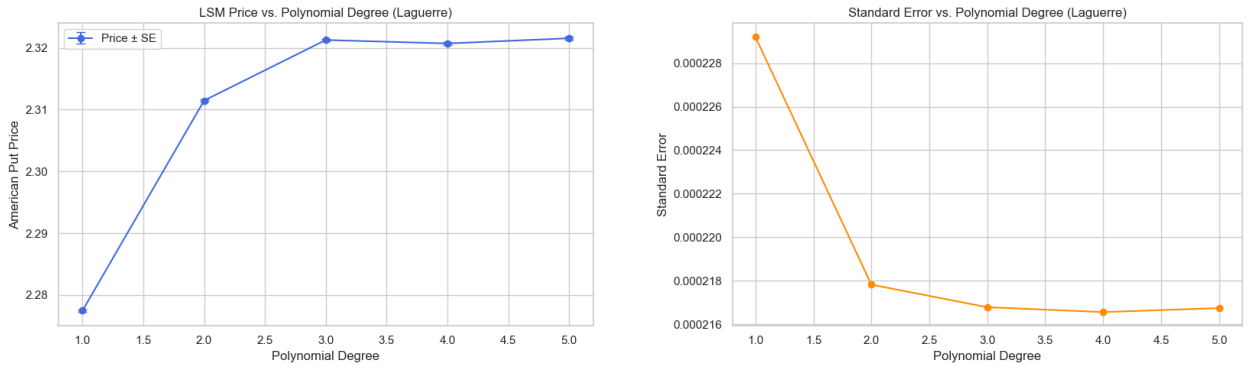
(b) Number of time steps for 1 periode vs price

Figure 4: Convergency tests of the Longstaff–Schwartz Method Using Laguerre(3)

Before analyzing results different ways to price an american put option, it is important to evaluate the numerical choices made in the original Longstaff–Schwartz method. In particular, we examine whether using 100,000 Monte Carlo paths and 50 time steps provides stable and accurate estimates. Figure 4 (a) shows the option price as a function of the number of simulation paths. We observe that the price stabilizes around 30,000 paths and fluctuates only marginally beyond this point. It is especially worth noting that the y-axis is very narrow, centered around 2.32, indicating that the estimates are already precise at relatively low path counts. Hence, 100,000 paths clearly satisfy the convergence criterion. In Figure 4(b), we plot the option price against the number of time steps used to discretize the exercise period. The price increases with more steps but converges visibly around 50. This confirms that 50 time steps are sufficient to capture the early exercise feature of the American option accurately. Together, these tests confirm that the standard LSM setup with 100,000 paths and 50 time steps yields stable and converged price estimates.

7.1.2 Testing of the Polynomial Degrees

To begin with it is worth noting that these are some very small standard errors. The figures clearly show that both the estimated option price and the standard error converge as the degree of the Laguerre polynomial basis increases. The largest improvements occur when moving from a low to a moderate number of basis functions, while the marginal gains diminish rapidly beyond that point. This pattern reflects a well-known property in function approximation using orthogonal polynomials. Although Judd (1998) presents this result in the context of Chebyshev polynomials, the same principle applies to Laguerre polynomials, which are used in the LSM approach. In Theorem 6.4.2, Judd shows that: If $f \in C^k[-1, 1]$



(a) LSM vs. Polynomial Degrees

(b) Std. error vs. Polynomial Degrees

Figure 5: Effect of polynomial degree on LSM pricing accuracy using Laguerre basis

and has a Chebyshev expansion:

$$f(x) = \frac{1}{2}c_0 + \sum_{j=1}^{\infty} c_j T_j(x),$$

then there exists a constant c such that

$$|c_j| \leq c \cdot j^{-k}.$$

This means that the smoother the function, the faster the coefficients c_j decay as the degree j increases. The same behavior holds for Laguerre polynomials, since they also form an orthogonal basis, specifically on the interval $[0, \infty)$ with respect to the weight function $w(x) = e^{-x}$. For smooth functions defined on this domain, the Laguerre coefficients decay rapidly as well, meaning that high-degree terms contribute very little to the overall approximation. In the context of Longstaff and Schwartz (2001), this theoretical result helps explain why only the first 3 Laguerre polynomials are needed to achieve accurate pricing of American options.

7.1.3 Basis Function Type in One Dimension

Table 1: LSM price and computational time for degree 3 basis functions

Basis function	**Price (s.e.)**	**Computation Time**
Laguerre	2.321277 (0.000217)	33.32 sec
Hermite	2.317713 (0.000214)	32.58 sec
Chebyshev	2.320905 (0.000217)	33.06 sec

Building on the result in Figure 4 where we saw that most of the gain in pricing accuracy is achieved already at polynomial degree 3, we now turn to the question of which type of orthogonal polynomial to use at 3-degrees. Table 1 reports the estimated prices, standard errors, and computation times for Laguerre, Hermite, and Chebyshev polynomials of degree 3 in one dimension. The differences across the three bases are minimal in terms of price, variance, and runtime. They all show a very small s.e. making them suitable for pricing 1 asset which is also reflected in their very similar price.

This similarity across polynomial types can be attributed to the low dimensionality of the problem. In one dimension, the continuation value is a smooth function that is relatively easy to approximate. Although Laguerre, Hermite, and Chebyshev polynomials are orthogonal with respect to different weight functions and defined over different domains, they all span sufficiently rich function spaces to capture the relevant features of the continuation value in this setting. As noted by Judd (1998), in low-dimensional problems many standard polynomial bases perform equally well because the approximation task is not very demanding. The differences between the bases become more apparent only as dimensionality increases and the functional structure becomes more complex.

7.1.4 Different simulation methods

Table 2: American Put Prices Using Different Simulation Methods (Laguerre Degree 3)

Method	**Price**	**Standard Error**	**Computation Time**
QMC (GBM)	2.5369	0.0002	43.92 s
Jump Diffusion	2.4867	0.0003	20.88 s
MC (GBM)	2.3213	0.0002	42.97 s

We will now look at the simulation. Here, we'll explore what happens when we introduce jump diffusion, as mentioned earlier. We will also compare Quasi-Monte Carlo due to this quote by Longstaff and Schwartz, *"Quasi-Monte-Carlo techniques in conjunction with the LSM algorithm may lead to significant improvements in computational speed and efficiency."* (Longstaff and Schwartz, 2001, p. 144) In Table 2 we see a higher price observed with Quasi-Monte Carlo (QMC) which is due to its more uniform sampling of the in-the-money region, enabled by low-discrepancy Sobol sequences. This leads to more stable continuation value regressions in the LSM algorithm, particularly in marginal exercise cases. While QMC does not change the exercise logic, it reduces the noise in the regression, which can result in slightly higher and more consistent price estimates compared to standard Monte Carlo.
Jump diffusion also produces higher prices, as the presence of downward jumps increases the likelihood of early exercise. These jumps introduce left-tail risk, which enhances the value of American put options by creating more scenarios where the option ends up deep in the money before expiry.

Jump diffusion reduces computation time a reason for this might be because its large and

sudden price movements push more paths into clearly exercisable or clearly out-of-the-money states earlier in the backward induction process. As a result, fewer paths remain active at later time steps, leading to smaller regression problems and faster evaluation. Although simulating jumps adds complexity per step, this is outweighed by the reduced number of continuation value estimations overall. Therefore one could argue that it will be a good idea to add jump diffusions as they might be more empirical correct and give a better computation time.

7.2 Multiple dimensions

Firstly, it shall be noted, that the following part has been run on a GPU, compared with the above CPU run. Therefore, the computational times will be somewhat faster. With that being said, in order to have timely convergence plots for the multiple dimensions, we had to stop testing at $d = 4$.

7.2.1 2 assets

From Table 3, we see that all three polynomial basis functions yield the same price and standard error at 1.6609 and 0.0064 respectively. The only deviations between the three are the computational time, which is very low at 3, 2 and 4 seconds. This does not yield any major difference in computation and could simply be down to smaller GPU interference. Since the pricing for all three is equal, the value of time of the American put will also be equal across all three polynomial basis functions.

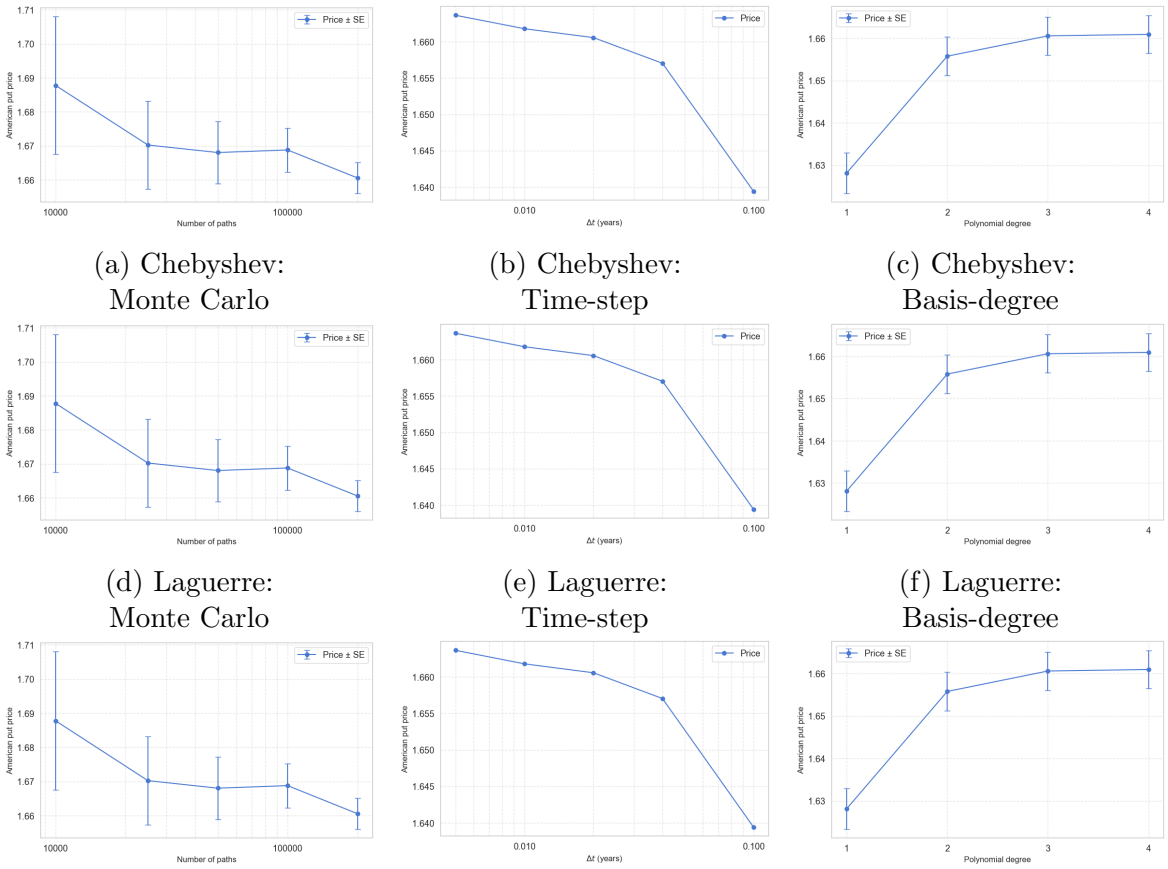
Polynomial	Price	Std error	Computational time	Euro price	Time value
Laguerre	1.6609	0.0064	3 seconds	1.4103	0.2506
Chebyshev	1.6609	0.0064	2 seconds	1.4103	0.2506
Hermite	1.6609	0.0064	4 seconds	1.4103	0.2506

Table 3: LSM with two assets

Moving on to our convergence plots for the different basis functions in Figure 6. Here we see that, they all follow somewhat along the same lines. We see that increasing the number of simulated paths for all yield a consistent price, yet we see a small downward jump for all at 200,000 paths. Suggesting, we might should have chosen a higher set of paths. Moreover, the error bands becomes smaller the more simulated paths we have, which confirms the $\sqrt{N}$-convergence in simulation.

Then looking at the amount of time-steps, we see that increasing the number of time steps i.e smaller time increments, will make the price converge. Once again, it does not seem that we have found optimal number of time steps, and we may should have been using more steps, yet one may argue that the change in price is negligible when going above our current $N = 50 \approx \Delta t = 0.02$.

Lastly, looking at the number of polynomial degrees, we see that the curve almost flattens after the third degree, which suggest, we have found a stable and consistent price. One may argue, that a polynomial degree of 2 would be enough.



(a) Chebyshev: Monte Carlo

(b) Chebyshev: Time-step

(c) Chebyshev: Basis-degree

(d) Laguerre: Monte Carlo

(e) Laguerre: Time-step

(f) Laguerre: Basis-degree

(g) Hermite: Monte Carlo

(h) Hermite: Time-step

(i) Hermite: Basis-degree

Figure 6: Convergence plots for two assets. Rows correspond to the polynomial family (Chebyshev, Laguerre, Hermite), while columns show Monte Carlo convergence, time-step convergence, and basis-function degree.

7.2.2 5 assets

Table 4 summarizes the LSM estimates for the assets for the three polynomial bases. Firstly, we see that the price estimates for the three differ. Laguerre gives the price 1.2003, while Chebyshev and Hermite yield 1.2106 and 1.2021, respectively. Furthermore, the standard errors are smaller than these discrepancies which means it is not standard sampling noise. Once again, they are all above the European price, which gives differences in time values for the three bases.

Turning to the computational we see some slower times for all three polynomials. For the Laguerre it takes 146 seconds, which is approximately 2,5 minutes. For the Chebyshev and Hermite, the times are approximately the same. This is a little time investment in each price, by introducing 3 more asset to the basket, yet it shows the non-linear increase in computational time by adding more assets.

Polynomial	**Price**	**Std error**	**Computational time**	**Euro price**	**Time value**
Laguerre	1.2003	0.0046	146 seconds	0.9205	0.2798
Chebyshev	1.2106	0.0046	160 seconds	0.9205	0.2901
Hermite	1.2021	0.0046	140 seconds	0.9205	0.2816
MLP	0.9148	0.0029	147 seconds	0.9205	-0.0057

Table 4: LSM with five assets

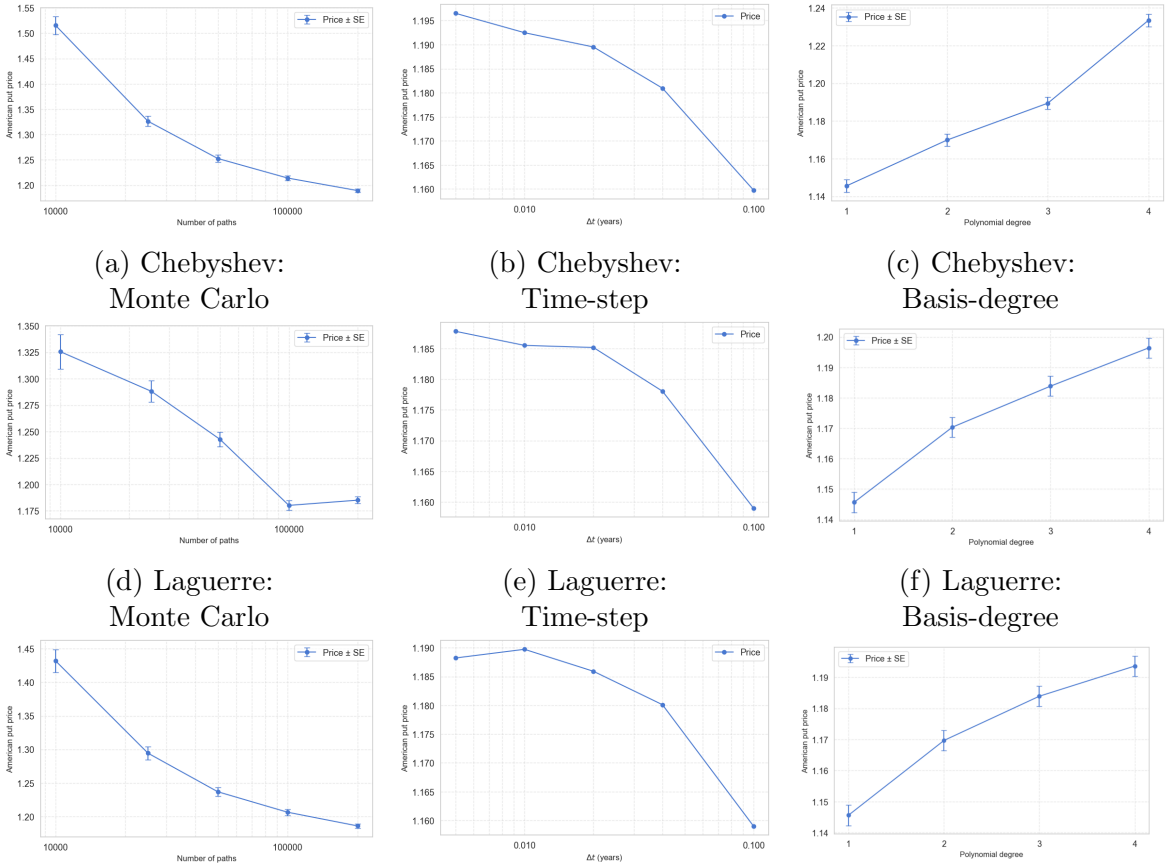
Moving on to the convergence plots for five assets in Figure 7. Here we see some differences between the polynomials bases. Looking at the Monte Carlo convergence for all three, we see a downward pattern for each polynomial when increasing the number of paths. Although, the change in prices becomes smaller for each path added, the changes are still relatively large for all. For the Laguerre polynomial, there is even a slight upwards jump when going from 100,000 to 200,000 paths. It shall be noted, that the standard error does become smaller by increasing the number of paths.

Turning to the number of time steps. We see that refining the grid, makes the change in price relatively negligible. This is best seen for the Laguerre polynomial in Figure 7e. Here the curve almost flattens, making the change in price neglible. The same can not be said for the Hermite and Chebyshev polynomials, as these both move even when going from 100 to 200 timesteps.

Moving on to our number of degrees for the polynomial bases. Looking at the number of degrees shown in Figure 7c, 7f and 7i, we see that we find no convergence at all, since we do not stabilize towards a singular price within the chosen level of degrees and more should be added. This holds for all polynomial bases, as we can not be sure of convergence with just $d = 3$ and even more than $d = 5$, should be tested. This means, that we may not be able to capture the higher order interactions, which we introduce by adding more stocks into our basket, yielding different pricing across the three polynomial bases.

Lastly, we tried to implement a Neural Network to alleviate the higher computational cost and yield somewhat better accuracy. Unfortunately, the American basket option is priced

below the European, which makes the pricing infeasible as the American time value should always be equal to zero or more, which means the American should be valued equal or higher than the European at all times. Moreover, the computation takes about the same time, which was the most prevalent thing we were searching for, the speed of computation.



(a) Chebyshev: Monte Carlo

(b) Chebyshev: Time-step

(c) Chebyshev: Basis-degree

(d) Laguerre: Monte Carlo

(e) Laguerre: Time-step

(f) Laguerre: Basis-degree

(g) Hermite: Monte Carlo

(h) Hermite: Time-step

(i) Hermite: Basis-degree

Figure 7: Convergence plots for five assets. Rows correspond to the polynomial family (Chebyshev, Laguerre, Hermite), while columns show Monte Carlo convergence, time-step convergence, and basis-function degree.

8 Discussion and concluding remarks

8.1 Results

The results obtained across the two and five assets experiments are somewhat consistent with expectations from dynamic programming and the Longstaff–Schwartz (LSM) framework. In both the one- and two-asset case, we see observed stability across different basis functions, which confirms, that for low-dimensional problems(few assets) and relatively low degree of polynomial bases is good enough to capture the continuation value. This is further

supported by corresponding convergence plots, where by increasing the number of simulated paths and time steps, we observe diminishing effects on the estimated option price and the lowering of the standard error.

In contrast, the five-asset results show some limitations of the polynomial regression approach. Firstly, there is some difference between the pricing of the option. Secondly, we struggle to find some stabilization in our convergence plots and especially the basis functions convergence-plot. This indicate that our chosen level of degree for these basis functions might not be sufficient in order to approximate the continuation value accurately in our higher asset-case. This matches the theory, as when the number of dimensions(assets) grow, the function space required in order to approximate the true continuation value expands, which leads the low-polynomials to be potentially underfitting. Therefore, a higher degree should most likely have been chosen to combat this problem. But as the number of polynomial degrees increased the computational burden rose non-linearly, and even after running our multiple asset case on a GPU, as a CPU was infeasible, we have might have demonstrated Bellman's curse of dimensionality in just these few dimension cases. This is because the regression, must now span over a larger set of observations increasing the computational time.

Our attempt to "break" Bellman's curse was by running a neural network. However, our MLP did price the American option below the European, which is not possible. This can be because of insufficient network depth, suboptimal hyperparameter tuning, or overfitting to local regions of the state space without capturing the global exercise boundary. Here more testing should be done, in order to make eligible to help us price basket options.

8.2 Alternative pricing methods

An alternative way of pricing could be numerical quadrature which offers an efficient and accurate alternative to simulation-based pricing in one dimension. By replacing Monte Carlo with deterministic integration rules such as Gauss–Hermite Judd (1998), continuation values can be computed precisely at each time step. For smooth payoff functions, convergence is fast and requires only a small number of nodes. However, quadrature becomes increasingly costly with finer time discretization, since each point in the price grid requires a separate integral. As the number of timesteps increases, the total number of integrals grows rapidly. This limits the method's practicality, even in one dimension, and highlights its sensitivity to time resolution. Additionally, quadrature does not extend well to higher dimensions: the number of integration points grows exponentially with the number of assets meaning it's subject to the of the curse of dimensionality. We do also know about reinforcement methods such as Least Sqaures Policy Iteration(LSPI) and Fixed Q iteration (FQI) but our understanding is that these methods does not give a more accurate price with lower standard error and are computational costly. We have thus not investigated them further.

For multi-asset American options, recent work by Yang and Li (2025) proposes using sparse grid interpolation to efficiently approximate continuation values. Their method transforms the asset space into a bounded domain and applies high-order interpolation on static sparse

grids to handle the Bellman recursion. This significantly reduces the number of grid points compared to full tensor grids, allowing them to price basket options with up to 16 assets which far beyond what is feasible with traditional grid or regression-based methods. Compared to our results, which relied on regression with polynomials, sparse grids like and neural networks offer better scalability in high dimensions, though they still require smoothness assumptions and are more complex to implement. The reason we did not investigate this further was simply cause we choose to down the neural network route.

8.3 Final remarks

We find that the Longstaff-Schwartz method performs optimally when they are using Laguerre polynomials with degree 3, yielding accurate and stable prices as in their study. Incorporating Merton's jump diffusion model into the simulation improves empirical realism and reduces computational time, while maintaining low standard error. However, it results in slightly higher option prices compared to the original MC simulation.

Moving to higher dimensions, we find that lower degrees of polynomials are still valid for lower dimensions, yet moving onto higher dimensions our chosen low polynomials become infeasible as the function state expands. In higher dimensions, both the convergence and computational time exhibited problems, showing Bellman's curse. Finally, our attempt to alleviate the problem with Neural Network, gave us a incorrect price. Thus more testing will be needed.

References

Adda, J. and Cooper, R. W. (2002). *Dynamic Economics: Quantitative Methods and Applications*. The MIT Press, Cambridge, MA.

Becker, S., Cheridito, P., and Jentzen, A. (2020). Pricing and hedging american-style options with deep learning. *Journal of Risk and Financial Management*, 13(7):158.

Black, F. and Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654.

Glasserman, P. (2003). *Monte Carlo Methods in Financial Engineering*, volume 53 of *Stochastic Modelling and Applied Probability*. Springer Science+Business Media, New York, NY.

Judd, K. L. (1998). *Numerical Methods in Economics*. MIT Press.

Longstaff, F. A. and Schwartz, E. S. (2001). Valuing american options by simulation: A simple least squares approach. *The Review of Financial Studies*, 14(1):113–147.

Merton, R. C. (1976). Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1–2):125–144.

Pommelgård Lind, P. (2020). Classical option pricing theory and extensions to deep learning. *Københavns Universitet, Matematisk Institut*.

Yang, J. and Li, G. (2025). On sparse grid interpolation for american option pricing with multiple underlying assets. *Journal of Computational and Applied Mathematics*, 464:116544.