CentraleSupélec | université PARIS-SACLAY

Pricing of Amercian Options

CHAFIK Hala
CHIBA Nessrine
DEMRI Lina
EL OMARI Chaimae
NAGAZ Sarra

Supervised by GUO Gaoyue

P2025

March 2024

Contents

Chapter 1

Least square method

1.1 Introduction

An American option grants holders the ability to exercise at any point up until expiration, unlike European options, which are only exercisable on the expiration day. This added flexibility allows investors to swiftly capitalize on favorable market movements and dividend distributions. In this project, we aim to implement the algorithm described in the paper[1]

1.2 Framework

Let $S = (S_t)_{0\leq t\leq T}$ be the price process of some risky asset where $T \in \mathbb{N}$. We consider a riskless asset of price $S^0 = (S^0_t)_{0\leq t\leq T}$ s.t $S^0_0 = 1$

Define $F = F_t := \sigma(S_0, \ldots, S_t)_{0\leq t\leq T}$. An American option of payoff process $Z = Z_t := g(t, S_t)_{0\leq t\leq T}$ is an option that may be exercised at any trading day $t \in \{0, \ldots, T\}$ and its holder may receive Z_t. For $t \leq T$, denote by $\mathcal{T}_t$ the set of stopping times τ taking values in $\{t, \ldots, T\}$.

$$U_0 := \text{Price}(Z) = \sup_{\tau\in\mathcal{T}_0} \mathbb{E}[\tfrac{Z_\tau}{S^0_\tau}].$$

1.2.1 Optimal stopping

Following the classical optimal stopping theory, we introduce the Snell envelope $U = (U_t)_{0\leq t\leq T}$ of $Z = (Z_t)_{0\leq t\leq T}$:

$$U_t := \sup_{\tau\in\mathcal{T}_t} \mathbb{E}[\frac{Z_\tau}{S^0_\tau}|F_t].$$

The dynamic programming principle can be written as follows:

$$\begin{cases} U_T = Z_T, \\ \tilde{U}_t = \max\{\tilde{Z}_t, \mathbb{E}[\frac{U_{t+1}}{S^0_{t+1}}|F_t]\}, \quad 0 \leq t \leq T-1. \end{cases}$$

1.2.2 An alternative formulation

We consider the stopping time τ :

$$\tau_j = \min\{k \geq j \,|\, U_k = Z_k\}$$

The dynamic programming principle can be rewritten in terms of the optimal stopping times τ_j , as follows:

$$\begin{cases} \tau_T := T, \\ \tau_t := \begin{cases} t, & \text{if } \tilde{Z}_t \geq \mathbb{E}[\frac{Z_{\tau_{t+1}}}{S^0_{t+1}}|F_t], \\ \tau_{t+1}, & \text{otherwise,} \end{cases} \end{cases} \quad 0 \leq t \leq T-1.$$

Then $U_0 = \mathbb{E}[Z_{\tau_0}]$.

1.2.3 First approximation

Under the Markovian assumption, one has $\mathbb{E}[\frac{Z_{t+1}}{S^0_{t+1}}|F_t] = \mathbb{E}[\frac{Z_{t+1}}{S^0_{t+1}}|S_t] = f_t(S_t)$ for some measurable function f_t. In this first approximation following the steps of [**?**], we will project the conditional expectation in a polynomial base of dimension m. Here the choice of the base is important and it depends on the model used for the asset price.
The idea is to write:

$$\mathbb{E}[\frac{Z_{t+1}}{S^0_{t+1}}|S_t] \approx \langle \alpha | e^m(S_t) \rangle$$

Where α is a vector of dimension m determined using least squares to minimize the residue which is the difference between $\mathbb{E}[\frac{Z_{t+1}}{S^0_{t+1}}|S_t]$ and $\langle \alpha | e^m(S_t) \rangle$.
For the choice of the base we look at polynomial bases such as the basic polynomial base: $(1, X, ..., X^m)$ and we eventually look at orthogonal bases like the Laguerre base: $L_n(x) = e^x \frac{d^n}{dx^n}(x^n e^{-x})$

1.2.4 Second approximation

The second approximation is then to evaluate numerically $\mathbb{E}[Z_{\tau_1^m}]$ by Monte-Carlo methods.

- Simulate N independent paths $S^{(1)}, \ldots, S^{(N)}$ of the Markov process S and denote $Z_t^{(n)} := g(t, S_t^{(n)})$.
- For each path $S^{(n)}$,

$$\begin{cases} \tau_T^{m,n,N} := T, \\ \tau_t^{m,n,N} := t\mathbf{1}\{\tilde{Z_t^{(n)}} \geq \alpha_t^{(m,N)} \cdot e_m(S_t^{(n)})\} + \tau_{t+1}^{m,n,N}\mathbf{1}\{\mathbf{1.2.5} \quad \textit{Goal} \end{cases}$$

Under suitable conditions, show the convergence:

$$U_0^m \to U_0 \text{ as } m \to \infty,$$
$$U_0^{m,N} \to \text{a.s. } U_0^m \text{ as } N \to \infty.$$

Details regarding the proofs are available in the referenced paper.

1.2.6 Algorithm steps

– **Step1:** Monte Carlo simulation for the asset prices (First approximation). we consider that the asset price follows a geometric Brownian motion:

$$dS_t = S_t(rdt + \sigma dW_t)$$

where:

* S_t is the price of the asset at time t
* r is the risk-free interest rate
* σ is the volatility of the asset
* dt is a small time increment
* dW_t is the increment of the standard Brownian motion.

Therefore we have that the price follow this : $S_t = S_{t-1} \times \exp\left(\left(r - \frac{\sigma^2}{2}\right) \times \frac{1}{L} + \sigma\sqrt{\frac{1}{L}} \times Z_t\right)$
Where Z_t follows a normal distribution with mean 0 and standard deviation 1.

Algorithm 1 Monte Carlo Price Simulator

Function $monte - carlo - price - simulator(self)$:
$Price\_simulation \leftarrow$ array of size $(L+1) \times n$, initialized with zeros
for each path from 1 to n **do**
  Set $Price\_simulation[0, path] = S0$ {Set initial price for each path}
  for each time step i from 1 to L **do**
    Generate a random number z_i from a normal distribution with mean 0 and standard deviation 1
    Calculate $Price\_simulation[i, path]$ as follows: $Price\_simulation[i, path] = Price\_simulation[i-1, path] * exp((r - \sigma^2/2) * (1/L) + \sigma * \sqrt{(1/L)} * z_i)$
  end for
end for
return $Price\_simulation$ =0

– **Step2:** Least squares function to approximate the conditional expectation.
For this, we need to define the polynomial base used to project as explained before.
However, to write the code we suppose that we already have it. The purpose of the least square is to determine for each time step the alpha that minimizes the residue for all Montecarlo paths.
We tested two algorithms for the least square method to find the alpha:

* Using Scipy to minimize defining the loss fuction

Algorithm 2 Least Square Minimizer

function LEAST_SQUARE_MINIMIZER
  Input:
    payoff_simulation: The payoff simulation matrix
    Tau_i_1: Array containing time indices
    Price_simulation_i: Array containing price simulations
    projection_base: Function to compute projection base
    m: Parameter for projection base function
    n: Number of paths in simulation
  Output:
    α: The optimal coefficients
  function LOSS_FUNCTION(α)
    $total\_loss \leftarrow 0$ **for** $n = 1$ ***to*** $N - 1$ **do**
      $Z_n \leftarrow \text{payoff\_simulation}[\text{int}(\text{Tau}_{i+1}[n-1]), n-1]$
      $em\_X_n \leftarrow \text{Projection\_base}(m, \text{Price\_simulation}_i[n-1])$
      $loss \leftarrow (Z_n - \alpha \cdot em\_X_n^T)^2$ {Transpose em_X_n to make dot product compatible}
      $total\_loss \leftarrow total\_loss + loss$

      return $total\_loss$
  end function
  $result \leftarrow \text{minimize}(\text{LOSS\_FUNCTION}, \text{np.zeros}(m+1))$
  $\alpha \leftarrow \text{result.x}$
  return α
end function=0

* Using linear regression of sklearn package. We needed to define for this case.

Let Y and X be matrices defined as follows:

$$Y = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

Where Y_k and X_k are computed as:

$$Y_k = \text{payoff\_simulation}\left[\text{int}(Tau\_i\_1[k] - 1), k\right] \cdot \exp(-r \cdot (T - \text{int}(Tau_{i+1}[k])))$$
$$X_k = \text{projection\_base}(m, \text{Price\_simulation\_i}[k])$$

Algorithm 3 Least Square Minimizer

0: **function** LEASTSQUAREMINIMIZER
0: **Input**:
0: payoff_simulation: Matrix of simulated payoffs
0: Tau_i+1: Array of time indices i+1
0: Price_simulation_i: Array of simulated prices
0: projection_base: Function for projecting
0: m: Parameter for projection
0: n: Number of simulation paths
0: **Output**:
0: α: Optimal coefficients
0: $Y \leftarrow$ array of zeros with length Tau_i_1
0: **for** $k \leftarrow 1$ **to** n
0: $Y[k] \leftarrow$ payoff_simulation[int(Tau_i_1$[k] - 1), k$]
0: **endfor**
0:
0: $X \leftarrow$ array of projections computed by projection_base(m, Price_simulation_i[$path$]) for $path$ in $[0, n-1]$
0: *Instantiate a linear regression model*
0: *Fit the model with X and Y*
0: $\alpha \leftarrow$ coefficients of the fitted model
0: **return** α
0: **end function**=0

The two least squares functions give similar results; however, the one where we use linear regression with the sklearn model is faster. Therefore, we use it to price our American options.

- **Step3:** Dynamic programming to create find τ_1

Algorithm 4 Dynamic Programming for Price Calculation

0: **function** DynamicProgPrice
0: **Input**:
0: S_0: Initial asset price
0: Payoff_0 : initial payoff
0: n: Number of paths
0: m: Parameter for projection
0: L: Number of time steps
0: $payoff\_function$: Function to compute payoff
0: projection_base: Function to compute projection
0: discounted Price Simulation: Montecarlo price simulation
0: discounted Payoff Simulation: The payoff of the simulated prices
0: **Output**:
0: Initialize Tau as an array of zeros with shape (L, n)
0: $Tau[L-1, :] \leftarrow L \times \mathbf{1}_n$ {Set last time step}
0: **For** $i \leftarrow L-2$ **to** 0 **step** -1
0: Compute α_i using LeastSquareMinimizer
0: **For** $path$ **in** $[0, 1, ..., n-1]$
0: $approx \leftarrow \alpha_i^T \times \text{projection\_base}(m, discounted\_price\_simulation[i, path])$
0: **If** $discounted\_payoff\_simulation[i, path] \geq approx$
0: $Tau[i, path] \leftarrow i + 1$
0: **Else**
0: $Tau[i, path] \leftarrow [i + 1, path]$
0: **EndIf**
0: **EndFor**
0: **EndFor**
0:
0: $monte\_carlo\_approx \leftarrow \frac{1}{n} \sum_{i=1}^{n} discounted\_payoff\_simulation[\text{int}(Tau[0, i]) - 1, 0]$
0: $U_0 \leftarrow \max(payoff_0, monte\_carlo\_approx)$
0: **return** U_0
0: **end function**=0

Remark: we need to discount the prices and the payoffs. This doesn't change the purpose of least square since the paper works with discounted prices or assumes that $r = 0$.

1.2.7 Implementation

In the implementation of the algorithm, we defined different classes that interact with each other:

- Option class, which takes the following attributes: S0 (price at time 0), K (strike price of the option), T (maturity of the option), and a boolean to indicate if it is a call or a put, and has a method "payoff" that returns the payoff of the option.

- Montecarlo class, which takes the following attributes: modeltype (it needs to choose whether we will consider the asset price a binomial or that it follows a Brownian geometric model), r (the risk-free rate), σ (the volatility of the asset price), n (number of Monte Carlo simulations), and payofffunction to eventually simulate the payoff. It has three methods: one to simulate the prices depending on the chosen model, one to visualize the prices, and the other one to get the payoff.

- The Dynamicpricing class is the one we use to price the option. It inherits from montecarlo its attributes and methods. We define other attributes that are specific to the dynamic programming pricing, such as 'm' (the dimension of the projection base) and the type of the projection, which takes the string 'poly' if we want to use a polynomial base, and 'Laguerre' if we want to use the Laguerre polynomials. It has two static methods that define the bases, and three methods: one to set the projection base, one to perform the least square minimization, and the one that returns the price.

1.3 Results

1.3.1 Comparing with black scholes model

Since we work with a non-dividend paying asset, the price of the American option needs to be the same as the European option. We consider therefore this model and we calculate each time to compare the two.

For $\sigma = 0.2$, $r = 0.01$, $L = 10$,$n = 10000$, $m = 50$, $S_0 = 100$ and $k = 95$ for a call option, we obtain :

- **Algorithm price=** 5
- **Black Scholes price =** 5.02

We also test the algorithm for put option and for $\sigma = 0.2$, $r = 0.01$, $L = 10$,$n = 10000$, $m = 50$, $S_0 = 95$ and $k = 100$ we get :

- **Algorithm price=** 5
- **Black Scholes price =** 5.02

Remark1: For put call and put options for $n = 10000$ and $m = 50$ takes about 57s to execute when the black sholes pricer takes about 0.4s.

Remark2: In here when we use $r \neq 0$ we make sure that we discount the payoffs in the conditional expectations. Since we work with a geometric Brownian motion for the asset price we use continuous discounting.
To consider it we discount the payoffs and the prices before entering the Dynamic programming algorithm. Where $\tilde{S}_t = S_t.exp(-r.(L-t))$

Remark3: The choice of the base changes according the parameters we have. For example sometimes the canonical Polynomial bases gives better result. For $\sigma = 0.5$, $r = 0.01$, $L = 20$,$n = 10000$, $m = 50$, $S_0 = 95$ and $k = 100$

We get :

- **Algorithm price Laguerre=** 6.13
- **Algorithm price cannocal polynomial base =** 7.55
- **Black Scholes price =** 7.51

Remark4: For $r = 0$ we always get the same price as the black scholes model using Laguerre polynomials.

1.3.2 Convergence analysis

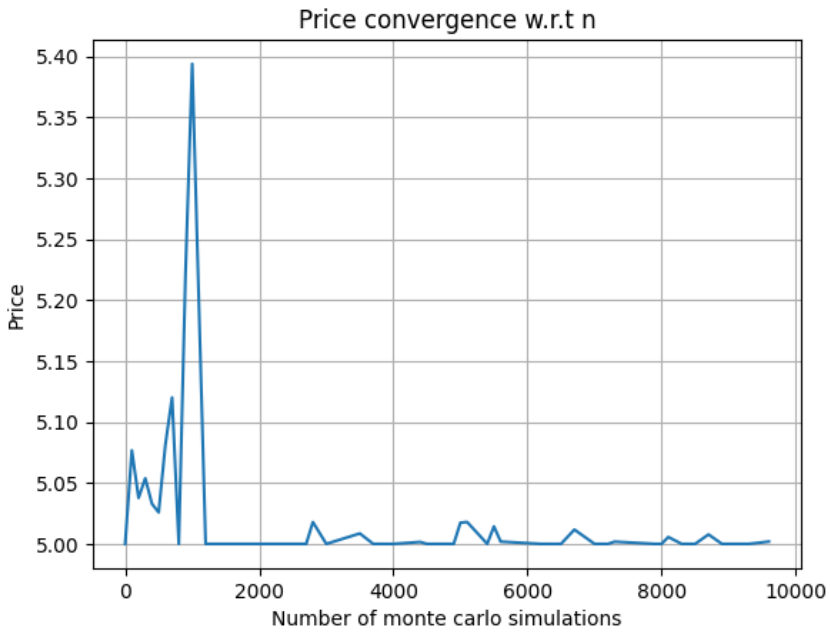

Figure 1.1: Price convergence w.r.t n $\sigma = 0.1$, $r = 0.01$, $L = 10$,$m = 50$, $S_0 = 100$ and $k = 95$

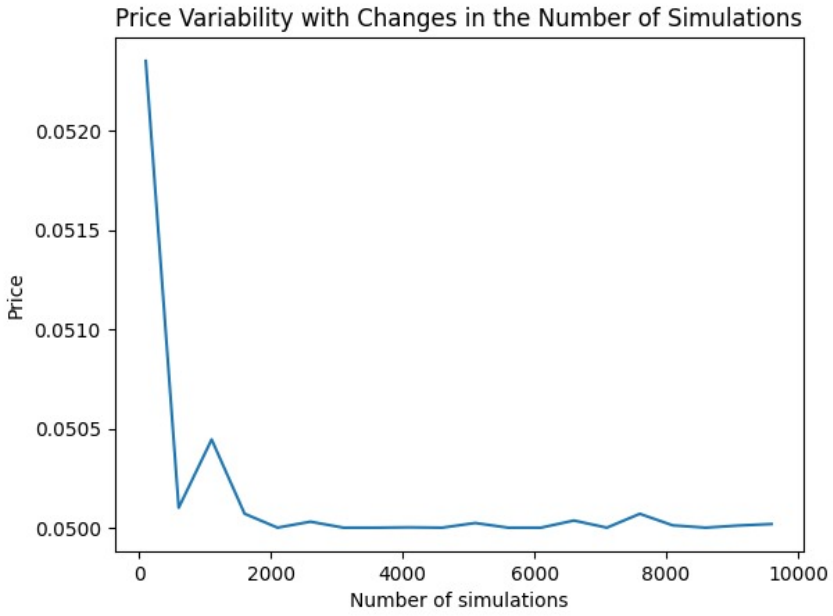

Figure 1.2: Price convergence w.r.t n $\sigma = 0.02$, $r = 0.0$, $L = 10$,$m = 50$, $S_0 = 1$ and $k = 0.95$

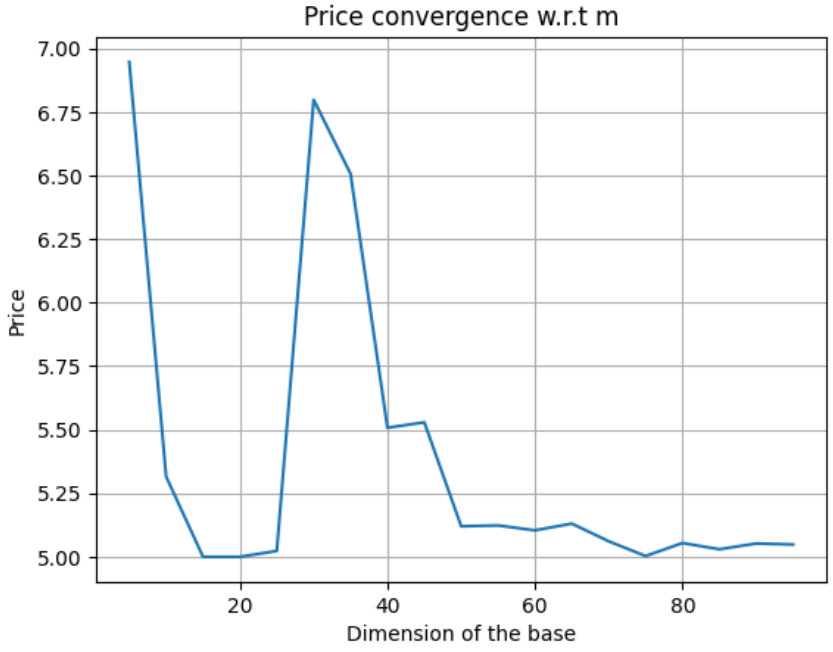

Figure 1.3: Price convergence w.r.t m for $\sigma = 0.1$, $r = 0.01$, $L = 10$,$n = 10000$, $S_0 = 95$ and $k = 100$

1.3.3 Price variation with respect to maturity

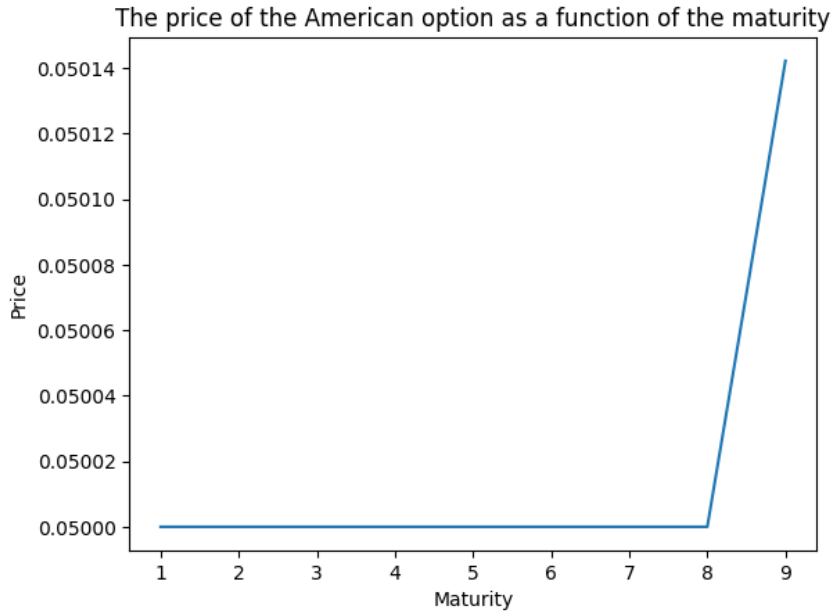

Figure 1.4: Price variation w.r.t Maturity L using Laguerre basis for $\sigma = 0.02$, $r = 0$, $m = 50$,$n = 2000$, $S_0 = 0.95$ and $k = 1$

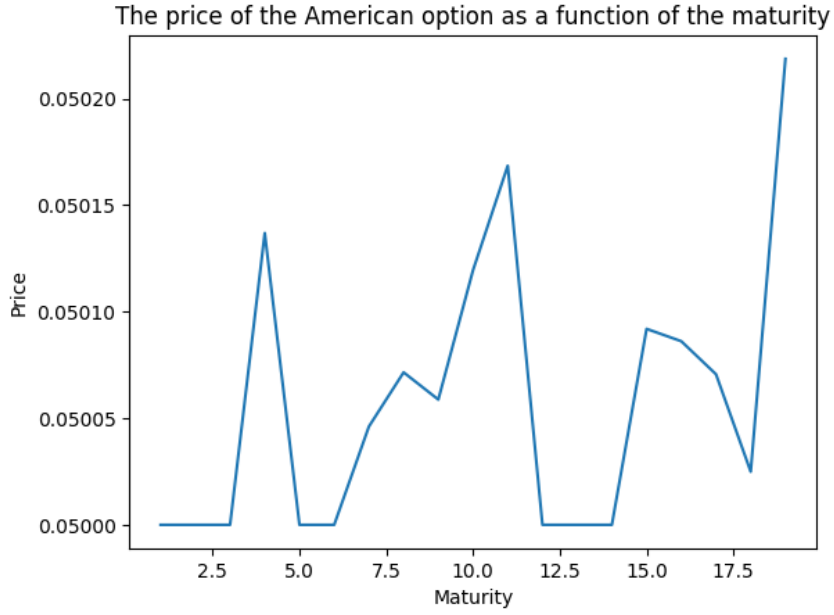

Figure 1.5: Price variation w.r.t Maturity L using Polynomial basis for $\sigma = 0.02$, $r = 0$, $m = 50$,$n = 5000$, $S_0 = 0.95$ and $k = 1$

Chapter 2

The Quantization Method

2.1 The Idea

The underlying idea is to employ a multinomial model for the prices by discretizing the values they can assume. We define:

$$\hat{X} := f(X) \approx X, \quad \text{such that } \operatorname{supp}(\hat{X}) = n, \quad \text{where } n \text{ is a finite integer.} \tag{2.1}$$

Given that $\forall \epsilon > 0, \exists M > 0$ such that $\mathbb{P}(|X| > M) \leq \epsilon$, we define the function f as follows:

$$f(x) = \begin{cases} 0 & \text{if } x \notin [-M, M], \\ \frac{kM}{n} & \text{if } x \in \left[\frac{kM}{n}, \frac{(k+1)M}{n}\right). \end{cases} \tag{2.2}$$

For this method to be effective, one must first select an ϵ, then choose M such that :

$$\frac{\max_t \mathbb{E}[S_t^2]}{M} < \frac{\epsilon}{2} \tag{2.3}$$

And subsequently pick n large enough so that :

$$\frac{M}{n} < \frac{\epsilon}{2}. \tag{2.4}$$

Following this, we compute the corresponding prices $(\hat{V}_t)_{0 \leq t \leq T}$ of the American option with payoff H using the formula:

$$\begin{cases} \hat{V_T} & = h(T, \hat{S_T}), \\ \hat{V_t} & = \max\left(h(t, \hat{S}_t), \mathbb{E}[\hat{V}_{t+1} | \hat{S}_t]\right), \end{cases} \tag{2.5}$$

Where the conditional expectation can be written as the following sum:

$$\mathbb{E}[\hat{V}_{t+1} | \hat{S}_t = x_i^t] = \frac{\sum_{j=1}^n \hat{V}_{t+1}(t+1, x_j^{t+1}) \mathbb{P}(\hat{S}_t = x_i^t, \hat{S}_{t+1} = x_j^{t+1})}{\sum_{j=1}^n \mathbb{P}(\hat{S}_t = x_i^t, \hat{S}_{t+1} = x_j^{t+1})}, \quad \forall i \leq n,$$

And each transition probability can be written as the following and computed using Monté-Carlo simulations :

$$\mathbb{P}(\hat{S}_t = x_i^t, \hat{S}_{t+1} = x_j^{t+1}) = \mathbb{P}\left(S_t \in [x_i^t, x_{i+1}^t), S_{t+1} \in [x_j^{t+1}, x_{j+1}^{t+1})\right)$$

$$= \frac{1}{m} \sum_{k=1}^{m} \mathbf{1}_{\{x_i^t \leq S_t^{(k)} < x_i^{t+1}, x_j^{t+1} \leq S_{t+1}^{(k)} < x_{j+1}^{t+1}\}}.$$

2.2 Implementation and Results

The implementation begins with the simulation of asset prices using the Monte-Carlo method, as previously defined.

2.2.1 Defining the Discretizing Function

The discretizing function is defined as follows:

Algorithm 5 Quantization Function

Data: x, M, n
Result: Quantified value of x

```
if |x| > M then
    return 0
else
    k ← int(x · n/M)  return k · M/n
```

2.2.2 Parameter Selection

Parameters are chosen to satisfy conditions (2.3) and (2.4):

$$\epsilon = 0.1, \quad M = \frac{4 \max_t \mathbb{E}[S_t^2]}{\epsilon}, \quad n = \text{int}\left(\frac{4M}{\epsilon}\right), \quad K = 0.95.$$

Illustrations of our quantization are provided below:

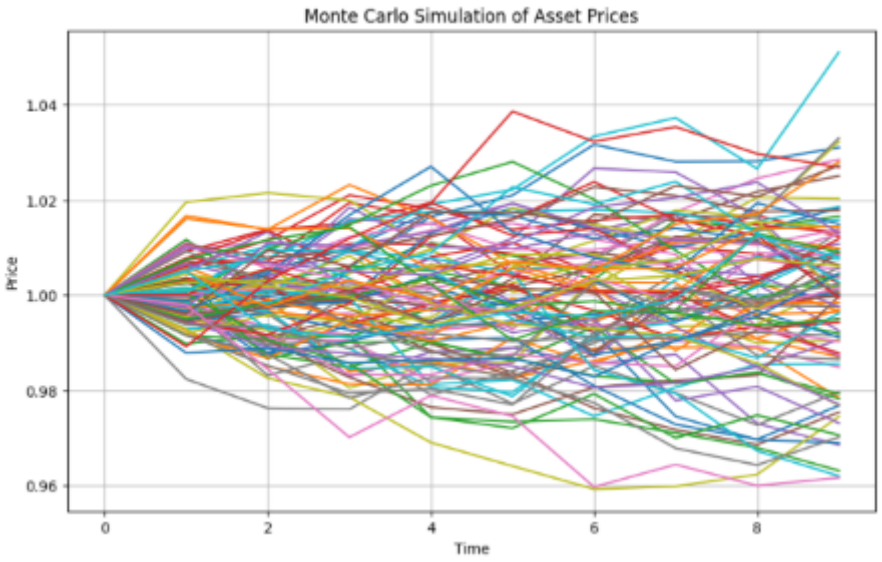

Figure 2.1: Monte-Carlo simulations for m=100, r=0, $\sigma = 0.02$, T=10, $S_0 = 1$

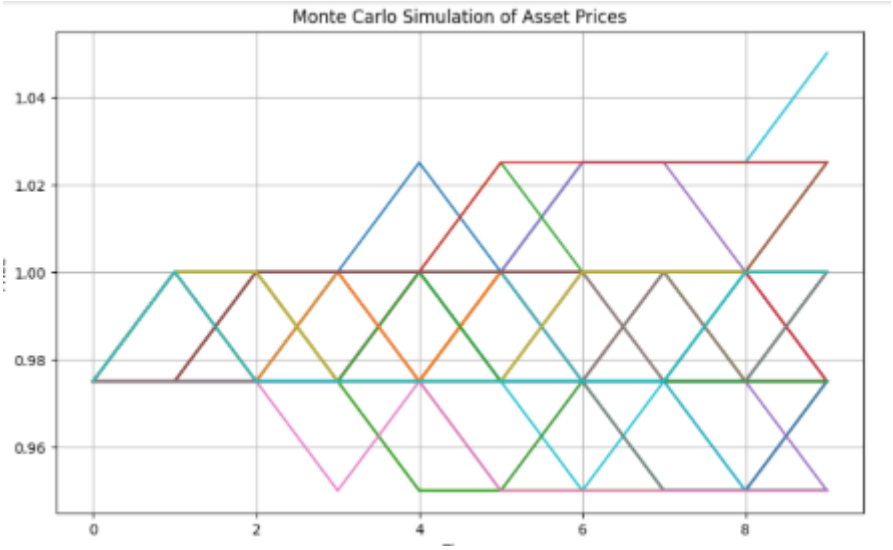

Figure 2.2: Quantization of the previous simulation

2.2.3 Dynamic Programming and Probability Computation

Dynamic programming is employed to compute option prices for different maturity values with $K = 0.95$. This process requires defining a function to compute $\mathbb{P}(\hat{S}_t = x_i^t, \hat{S}_{t+1} = x_j^{t+1})$ using the Monte-Carlo simulations and the payoff function for a call option of strike K.

Algorithm 6 Probability Calculation ProbXItXJt1

Result: Probability calculation *prob*

Function `ProbXItXJt1`$(t, x_{it}, x_{i+1t}, x_{jt+1}, x_{j+1t+1}, sim\_asset\_prices)$**:**

  $prob \leftarrow 0$ **for** $l \leftarrow 0$ **to** $m-1$ **do**

    if $x_{it} \leq sim\_asset\_prices[t,l] < x_{i+1t}$ ***and*** $x_{jt+1} \leq sim\_asset\_prices[t+1,l] < x_{j+1t+1}$ **then**

      $prob \leftarrow prob + 1$

    end

  end

  $prob \leftarrow prob/m$ **return** *prob*

Algorithm 7 Payoff Function PayoffFunction

Result: Payoff calculation

Function `PayoffFunction`(x, K)**:**

  return $\max(x-K, 0)$

Algorithm 8 Dynamic Programming Algorithm for Option Pricing

Data: $K, sim\_asset\_prices, quant\_sim\_asset\_prices$

Result: Calculation of V_hat for barrier options

$V\_hat \leftarrow \text{zeros}(L, n)$ $V\_hat[L-1, :] \leftarrow$ payoff_function(linspace$(-M, M, n), K)$ **for** $t \leftarrow L-2$ **to** 0 **do**

  for $j \leftarrow 0$ **to** $n-1$ **do**

    $cond\_exp \leftarrow 0$ $prob \leftarrow 0$ **for** $i \leftarrow 0$ **to** $n-1$ **do**

      $p \leftarrow$ prob_x_it_x_jt1$(t, j \cdot M/n, (j+1) \cdot M/n, i \cdot M/n, (i+1) \cdot M/n, sim_asset_prices)$ $cond\_exp \leftarrow cond\_exp + V\_hat[t+1, i] \cdot p$ $prob \leftarrow prob + p$

    $cond\_exp \leftarrow cond\_exp/prob$ $V\_hat[t, j] \leftarrow \max(\text{payoff\_function}(j \cdot M/n, K), cond\_exp)$

Algorithm 9 Calculation of the Option Price at $t = 0$

Data: $S0, K, sim\_asset\_prices, quant\_sim\_asset\_prices$

Result: Option price calculation $V\_hat[0, k]$

$k \leftarrow \text{int}(S0 \cdot n/M)$ $V\_hat \leftarrow$ dynamic_prog$(K, sim_asset_prices, quant_sim_asset_prices)$ **return** $V\_hat[0, k]$

2.2.4 Final Results

The culmination of these steps is visualized in the plot below :

Figure 2.3: Price(T)

2.3 Limitations of this approach

Selecting an ϵ that is excessively small results in a large n. Given the algorithm's complexity of $O(n^2Tm)$, this can significantly slow down code execution, rendering the approach suboptimal. Conversely, an ϵ that is too large compromises the method's accuracy. Therefore, it is crucial to find a balanced ϵ value that ensures both satisfactory results and manageable computational efficiency

2.4 Conclusion

Both approaches allow us to determine a price for American options, which are challenging to price accurately due to their optionality feature, making replication and precise pricing difficult.

Our findings indicate that the first approach, employing least squares, offers faster calculation times. However, it's worth noting that it is less stable to some extent.

There is room for improvement in both methods to refine the determination of the correct price for American options.

You can find the code at:
https://github.com/NESRINE202/Pricing_American_options.git

Bibliography

[1] Emmanuelle Clément, Damien Lamberton, and Philip Protter. An analysis of a least squares regression method for american option pricing. *Finance and stochastics*, 6:449–471, 2002.