Longstaff-Schwartz Least Squares Monte Carlo Method for American Option Pricing

Quant Tutorial Series

May 26, 2025

Contents

1 Theoretical Foundation

1.1 American Option Valuation Problem

The value of an American option at time t is given by the optimal stopping problem:

$$V_t = \sup_{\tau \in \mathcal{T}_{t,T}} E^Q \left[e^{-r(\tau - t)} h_\tau(S_\tau) \mid \mathcal{F}_t \right] \tag{1}$$

where $\mathcal{T}_{t,T}$ is the set of stopping times with values in $[t, T]$.

1.2 Dynamic Programming Formulation

The value function can be expressed recursively:

Theorem 1 (Dynamic Programming Principle)**.** *The American option value satisfies:*

$$V_t = \max \left\{ h_t(S_t), E^Q \left[e^{-r\Delta t} V_{t+\Delta t} \mid \mathcal{F}_t \right] \right\} \tag{2}$$

Proof. See Appendix A.1. □

1.3 Conditional Expectation Approximation

Longstaff-Schwartz approximates the continuation value using least squares regression:

$$E^Q \left[e^{-r\Delta t} V_{t+\Delta t} \mid S_t \right] \approx \sum_{j=1}^{J} \alpha_j \phi_j(S_t) \tag{3}$$

where $\{\phi_j\}$ are basis functions.

2 Implementation Guide

2.1 Algorithm Steps

1: Generate N paths of underlying asset S_t under Q
2: Initialize cashflow matrix $CF \leftarrow h_T(S_T)$
3: **for** $t = T - \Delta t$ downto 0 **do**
4: Identify in-the-money paths $\mathcal{I}_t = \{i : h_t(S_t^i) > 0\}$
5: Regress discounted cashflows on basis functions: $Y^i = e^{-r\Delta t} CF_{t+\Delta t}^i$, $X^i = \sum_j \alpha_j \phi_j(S_t^i)$
6: Estimate continuation value $\hat{C}_t(S_t^i) = \sum_j \hat{\alpha}_j \phi_j(S_t^i)$
7: Update cashflows: $CF_t^i = \begin{cases} h_t(S_t^i) & \text{if } h_t(S_t^i) > \hat{C}_t(S_t^i) \\ e^{-r\Delta t} CF_{t+\Delta t}^i & \text{otherwise} \end{cases}$
8: **end for**
9: Option value $\approx e^{-rT} \frac{1}{N} \sum_{i=1}^{N} CF_0^i$

Figure 1: LSM Algorithm Flowchart

3 Code Implementation

3.1 Python Implementation

```
import numpy as np
from numpy.polynomial import legendre

# Parameters from Longstaff-Schwartz Table 1
S0 = 36     # Initial stock price
K = 40      # Strike price
r = 0.06    # Risk-free rate
sigma = 0.2 # Volatility
```

```
T = 1.0    # Time to maturity
N = 100000 # Number of paths
M = 50     # Time steps

def lsm_american_option(S0, K, T, r, sigma, N, M, degree=3):
    dt = T/M
    t = np.linspace(0, T, M+1)

    # Generate paths
    S = np.exp((r - 0.5*sigma**2)*dt + sigma*np.sqrt(dt)*np.random.normal(size=(N,M)))
    S = np.hstack([np.ones((N,1))*S0, S])
    S = np.cumprod(S, axis=1)

    # Initialize cashflow matrix
    CF = np.maximum(K - S[:,-1], 0)

    # Backwards induction
    for i in range(M-1, 0, -1):
        in_the_money = S[:,i] < K
        X = S[in_the_money,i]
        Y = CF[in_the_money] * np.exp(-r*dt)

        # Legendre polynomial basis
        X_norm = 2*(X - K)/(K) - 1  # Normalize to [-1,1]
        basis = legendre.legvander(X_norm, degree)

        # Least squares regression
        coeffs = np.linalg.lstsq(basis, Y, rcond=None)[0]
        continuation = legendre.legval(X_norm, coeffs)

        # Exercise decision
        exercise = np.maximum(K - X, 0)
        exercise_idx = exercise > continuation

        CF[in_the_money] = np.where(exercise_idx, exercise, Y)

    # Discount final cashflows
    price = np.exp(-r*dt) * np.mean(CF)
    return price
```

Basis Degree	Price Estimate	Std Error
2	4.478	0.012
3	4.486	0.011
4	4.483	0.011

Table 1: Convergence with Basis Function Degree

3.2 Regression Diagnostics

A Proofs

A.1 Dynamic Programming Proof

Proof. The proof follows by... □