Longstaff-Schwartz Least-Squares Monte Carlo American Option Pricing Software Technical Documentation

John W. Melkumov

Contents

1 Introduction

This document briefly describes the background, structure, and usage of `LongstaffSchwartz`, a C++ based software that prices American call and put options by combining Monte Carlo simulation of the underlying asset with the Longstaff-Schwartz least squares regression algorithm.

1.1 High-level Workflow

1. **Configuration** is read from `params.json`. It sets market data, contract details, and numerical parameters.
2. **Asset paths** are generated by `AssetSimulator` under the risk neutral measure, assuming geometric Brownian motion (GBM).
3. **Payoff values** are provided by `OptionPricer`.
4. The **Longstaff-Schwartz engine** works backward in time, estimating continuation values with a polynomial regression and deciding when to exercise.
5. A CSV file of simulated paths and a summary result are returned to `main.cpp`, where results are printed.

2 Mathematical Background

2.1 Geometric Brownian Motion

Under the risk neutral measure $\mathbb{Q}$ the asset price S_t follows

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t \tag{1}$$

where r is the continuously compounded risk free rate, $\sigma > 0$ the volatility, and W_t a standard Brownian motion. The discrete time solution used in simulation is

$$S_{t+\Delta t} = S_t \exp\left[\left(r - \frac{1}{2}\sigma^2\right)\Delta t + \sigma\sqrt{\Delta t}\,Z\right], \quad Z \sim \mathcal{N}(0,1). \tag{2}$$

2.2 American Options

For an American option with payoff $g(S)$ and maturity T the fair value at $t = 0$ is

$$V_0 = \sup_{\tau \le T} \mathbb{E}^{\mathbb{Q}}\left[e^{-r\tau} g\left(S_\tau\right)\right] \tag{3}$$

where the supremum is over stopping times τ. Closed form solutions are not known in general, so numerical techniques like Longstaff-Schwartz are needed.

2.3 Longstaff-Schwartz Least Squares Monte Carlo

Simulate N paths on a grid of M time steps $t_0, \dots, t_M$. Work backward from $m = M - 1$ to 1:

1. Keep only paths that are in the money: $g(S_{t_m}^{(j)}) > 0$.

2. Regress the discounted payoff $Y^{(j)}_{t_{m+1}}$ on basis functions of $S^{(j)}_{t_m}$. With a polynomial basis of degree p this is

$$Y^{(j)}_{t_{m+1}} \approx \widehat{C}^{(j)}_{t_m} = \sum_{k=0}^{p} \beta_k {S^{(j)}_{t_m}}^k. \tag{4}$$

3. If immediate exercise $g(S^{(j)}_{t_m})$ exceeds the estimated continuation $\widehat{C}^{(j)}_{t_m}$, record an exercise at t_m; otherwise continue.

The mean of all discounted payoffs gives the price estimate $\widehat{V}_0$.

3 Code Architecture

3.1 Directory Layout

```
LongstaffSchwartz/
 include/
    AssetSimulator.h
    ConfigLoader.h
    LongstaffSchwartzEngine.h
    OptionPricer.h
 src/
    AssetSimulator.cpp
    ConfigLoader.cpp
    LongstaffSchwartzEngine.cpp
    OptionPricer.cpp
    main.cpp
 scripts/
    price_fetcher.py
    plot_paths.py
 tests/
    test_engine.cpp
 params.json
 CMakeLists.txt
```

3.2 ConfigLoader

Purpose Reads `params.json`, converts ISO dates to a time to maturity T in years, and validates inputs.

3.3 AssetSimulator

Purpose Generates an $(M + 1) \times N$ matrix of GBM paths using Eq. (2). A `std::mt19937_64` engine with a normal distribution provides randomness.

3.4 OptionPricer

Defines

$$g(S) = \begin{cases} \max(S - K, 0), & \text{call}, \\ \max(K - S, 0), & \text{put}. \end{cases} \tag{5}$$

3.5 LongstaffSchwartzEngine

Core routine The member `price()` performs the backward induction algorithm, solves the normal equations with LAPACKE, and returns an `LsmResult` holding $\widehat{V}_0$, the expected exercise step, and the payoff distribution.

4 Input and Output

4.1 JSON Configuration

Important keys in `params.json`:

- `spot` — initial price S_0
- `vol` — volatility σ
- `rate` — risk free rate r
- `strike` — strike price K
- `optionType` — `"CALL"` or `"PUT"`
- `startDate`, `expiryDate` — ISO strings, used to compute T
- `steps` — number of time steps M
- `paths` — number of Monte Carlo paths N
- `polyOrder` — regression polynomial degree p

4.2 CSV Path Dump

Each row is a time step, each column a path. This file can be visualized with the helper script `plot_paths.py`.

5 Auxiliary Python Scripts

price_fetcher.py Downloads historical daily close prices and outputs them for realized volatility estimation.

plot_paths.py Reads the CSV dump and plots all paths on a single figure for diagnostics.

6 Compiling and Running

1. Ensure a compiler with C++17, CMake 3.16 or newer, and BLAS or LAPACK is installed.
2. Compiling:
```
mkdir build; cd build
cmake ..
cmake --build .
```
3. Running:
```
./build/LongstaffSchwartz params.json
```

7 Possible Future Extensions

- Replace the GBM process with a stochastic volatility model.

Appendix: Least Squares Details

Given the Vandermonde matrix $\mathbf{X}$ and response vector $\mathbf{y}$, the regression coefficients solve

$$(\mathbf{X}^T\mathbf{X})\boldsymbol{\beta} = \mathbf{X}^T\mathbf{y}. \tag{6}$$

`LongstaffSchwartzEngine.cpp` calls `LAPACKE_dgesv` to solve this system.