Black-Scholes Option Pricing Implementation

1 Introduction

The Black-Scholes model remains one of the most widely used frameworks for pricing European-style options. This report presents a Python implementation of the model, including pricing functions for both call and put options, computation of the Greeks, and an implied volatility calculator.

2 Model Framework

Under the Black-Scholes assumptions, the underlying asset price follows geometric Brownian motion:

$$dS_t = rS_t\,dt + \sigma S_t\,dW_t$$

where r is the risk-free rate, σ is the volatility, and W_t is a standard Brownian motion.

The Black-Scholes formulas for European call and put options are given by:

$$C(S, K, T, r, \sigma) = S\Phi(d_1) - Ke^{-rT}\Phi(d_2) \tag{1}$$

$$P(S, K, T, r, \sigma) = Ke^{-rT}\Phi(-d_2) - S\Phi(-d_1) \tag{2}$$

where

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}} \tag{3}$$

$$d_2 = d_1 - \sigma\sqrt{T} \tag{4}$$

and $\Phi(\cdot)$ denotes the cumulative distribution function of the standard normal distribution.

3 Implementation

The implementation provides core functionality through several Python functions:

3.1 Pricing Functions

The `black_scholes_call` and `black_scholes_put` functions compute option prices directly from the closed-form solutions. These implementations use NumPy for numerical operations and SciPy's `norm.cdf` for evaluating the normal CDF.

3.2 The Greeks

The Greeks measure the sensitivity of option prices to various parameters. Our implementation computes:

- **Delta** (Δ): Sensitivity to underlying price
- **Gamma** (Γ): Second derivative with respect to underlying price
- **Vega** ($\mathcal{V}$): Sensitivity to volatility
- **Theta** (Θ): Time decay
- **Rho** (ρ): Sensitivity to interest rate

For a call option, delta is given by $\Delta_{\text{call}} = \Phi(d_1)$, while for a put option, $\Delta_{\text{put}} = \Phi(d_1) - 1$.

3.3 Implied Volatility

The implied volatility calculator uses a simple iterative search to find the volatility that matches an observed market price. Starting from $\sigma = 0.001$, the algorithm increments volatility in steps of 0.001 until the model price converges to within 0.001 of the target price.

4 Conclusion

I found this implementation pretty insightful. The math behind the equation is not fully clear to me. At least a small part of it (which I plan on solving by taking relevant math courses) but I think I now have an intuitive feel for it. It uses the log-normal distribution of stock prices to determine two key probabilities: the likelihood of exercise (via d_2) and the expected stock value at expiration (via d_1), discounted to present value.