Static No-Arbitrage Conditions for Option Chains

Quantitative Analysis Team

December 17, 2025

1 Introduction

In the theory of option pricing, market efficiency implies that option prices must satisfy certain structural constraints. If these constraints are violated, a ”static arbitrage” opportunity exists, allowing for a risk-free profit regardless of the future movement of the underlying asset. This report outlines the three primary conditions and provides an implementation for cleaning option dataframes.

2 Theoretical Constraints

2.1 Monotonicity

For any two strikes K_1 and K_2 such that $K_1 < K_2$, the price of a European call option $C(K)$ must be non-increasing with respect to the strike price:

$$C(K_1) \geq C(K_2) \quad \text{for} \quad K_1 < K_2 \tag{1}$$

Conversely, the price of a European put option $P(K)$ must be non-decreasing:

$$P(K_1) \leq P(K_2) \quad \text{for} \quad K_1 < K_2 \tag{2}$$

2.2 Vertical Spread (Slope)

The price difference between two options cannot exceed the present value of the difference in their strikes. For calls:

$$0 \leq C(K_1) - C(K_2) \leq e^{-rT}(K_2 - K_1) \tag{3}$$

A violation of the upper bound allows for a ”bull spread” arbitrage.

2.3 Convexity (Butterfly Spread)

Option prices must be a convex function of the strike price. For any three strikes $K_1 < K_2 < K_3$, the following must hold:

$$C(K_2) \leq \frac{K_3 - K_2}{K_3 - K_1} C(K_1) + \frac{K_2 - K_1}{K_3 - K_1} C(K_3) \tag{4}$$

This ensures that the cost of a long butterfly spread is non-negative.

3 Python Implementation

The following Python code demonstrates how to detect these violations within a pandas DataFrame representing an options chain.

```python
import pandas as pd
import numpy as np

def check_arbitrage(df):
    """
    Checks for Monotonicity and Convexity violations.
    Assumes df columns: ['strike', 'call_price']
    """
    df = df.sort_values('strike').reset_index(drop=True)
    violations = []

    # 1. Monotonicity Check
    df['diff'] = df['call_price'].diff()
    mono_violations = df[df['diff'] > 0]
    if not mono_violations.empty:
        violations.append("Monotonicity violated.")

    # 2. Convexity Check (Butterfly)
    # C''(K) >= 0 approx: (C1 + C3) / 2 >= C2 for equidistant strikes
    # For non-equidistant, we use the butterfly formula:
    for i in range(1, len(df) - 1):
        k1, k2, k3 = df.loc[i-1:i+1, 'strike']
        c1, c2, c3 = df.loc[i-1:i+1, 'call_price']

        # Linear interpolation bound
        w = (k2 - k1) / (k3 - k1)
        convex_bound = (1 - w) * c1 + w * c3

        if c2 > convex_bound + 1e-5:
            violations.append(f"Convexity violation at strike {k2}")

    return violations

# Example Usage
data = {
    'strike': [90, 95, 100, 105, 110],
    'call_price': [12.5, 8.0, 9.5, 4.0, 2.0] # 100 is a violation
}
df_options = pd.DataFrame(data)
results = check_arbitrage(df_options)
print(results)
```

Listing 1: Arbitrage Detection in Pandas

4 Conclusion

By enforcing these constraints, practitioners can ensure that the implied volatility surfaces generated from market data are internally consistent and reflect an arbitrage-free environment.