Derivatives Pricing and Options Theory

Complete Mathematical Framework and Implementation

Chapter 1: Introduction to Derivatives

1.1 What Are Derivatives?

A **derivative** is a financial contract whose value is derived from the performance of an underlying asset, index, or rate. The underlying can be stocks, bonds, commodities, currencies, interest rates, or market indexes.

Types of Derivatives:

1. **Forwards:** Customized contracts to buy/sell asset at future date
2. **Futures:** Standardized exchange-traded forwards
3. **Options:** Right (not obligation) to buy/sell at specified price
4. **Swaps:** Exchange of cash flows between parties

1.2 Options Fundamentals

Call Option: Right to **buy** underlying asset at strike price K by expiration T

Put Option: Right to **sell** underlying asset at strike price K by expiration T

Key Terms:

- **Strike Price (K):** Price at which option can be exercised
- **Expiration/Maturity (T):** Last date option can be exercised
- **Premium:** Price paid to purchase the option
- **Intrinsic Value:** Immediate exercise value
- **Time Value:** Premium - Intrinsic Value

Option Styles:

- **European:** Can only exercise at expiration
- **American:** Can exercise any time before expiration
- **Bermudan:** Can exercise on specific dates

1.3 Payoff Diagrams

Call Option Payoff at Expiration:

$$\text{Payoff} = \max(S_T - K, 0)$$

where S_T is the stock price at expiration.

Put Option Payoff at Expiration:

$$\text{Payoff} = \max(K - S_T, 0)$$

Profit:

$$\text{Profit} = \text{Payoff} - \text{Premium}$$

Chapter 2: No-Arbitrage Pricing and Risk-Neutral Valuation

2.1 The Law of One Price

Fundamental Principle: If two portfolios have identical payoffs in all states, they must have the same price today.

Arbitrage: Risk-free profit opportunity with no initial investment.

2.2 Risk-Neutral Pricing

Key Insight: Option prices can be computed as expected payoffs under a **risk-neutral measure** $\mathbb{Q}$, discounted at the risk-free rate.

$$C(S_0, K, T) = e^{-rT}\mathbb{E}^{\mathbb{Q}}[\max(S_T - K, 0)]$$

where:

- r: Risk-free interest rate
- $\mathbb{Q}$: Risk-neutral probability measure
- $\mathbb{E}^{\mathbb{Q}}$: Expectation under $\mathbb{Q}$

Risk-Neutral Drift:
Under $\mathbb{Q}$, the stock price follows:

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

Note: Expected return is r (not the actual stock return μ).

2.3 Put-Call Parity

European Options:

$$C - P = S_0 - Ke^{-rT}$$

where:

- C: Call option price
- P: Put option price
- S_0: Current stock price
- K: Strike price
- r: Risk-free rate
- T: Time to maturity

Proof: Construct two portfolios with identical payoffs at T:

- Portfolio A: Long call + Ke^{-rT} cash
- Portfolio B: Long put + Long stock

Both worth $\max(S_T, K)$ at expiration, so must have same price today.

Chapter 3: Binomial Tree Model

3.1 Single-Period Binomial Model

Stock Price Dynamics:

$$S_0 \to \begin{cases} S_0 u & amp; \text{with probability } p \\ S_0 d & amp; \text{with probability } 1-p \end{cases}$$

where (up factor), (down factor).

Option Payoffs:

- Up state: $C_u = \max(S_0 u - K, 0)$
- Down state: $C_d = \max(S_0 d - K, 0)$

Replicating Portfolio:
Construct portfolio of Δ shares and B bonds that replicates option payoff.

$$\Delta = \frac{C_u - C_d}{S_0(u-d)}$$

$$B = e^{-rT} \frac{uC_d - dC_u}{u-d}$$

Option Price:

$$C_0 = \Delta S_0 + B$$

Risk-Neutral Probability:

$$q = \frac{e^{rT} - d}{u - d}$$

Alternative Formula:

$$C_0 = e^{-rT}[qC_u + (1-q)C_d]$$

3.2 Multi-Period Binomial Tree

Backward Induction:

1. Calculate payoffs at terminal nodes (expiration)
2. Work backward, computing option value at each node:

$$C_t = e^{-r\Delta t}[qC_{t+1,u} + (1-q)C_{t+1,d}]$$

American Options:
At each node, compare continuation value vs immediate exercise:

$$C_t = \max\left(S_t - K,\, e^{-r\Delta t}[qC_{t+1,u} + (1-q)C_{t+1,d}]\right)$$

3.3 Python Implementation

```
import numpy as np

def binomial_tree_option(S0, K, T, r, sigma, N, option_type='call', exercise='european'):
    """
    Price options using binomial tree model

    Parameters:
    -----------
    S0 : float - Initial stock price
    K : float - Strike price
    T : float - Time to maturity (years)
    r : float - Risk-free rate
    sigma : float - Volatility
    N : int - Number of time steps
    option_type : str - 'call' or 'put'
    exercise : str - 'european' or 'american'

    Returns:
    --------
    float - Option price
    """

    dt = T / N
    u = np.exp(sigma * np.sqrt(dt))
    d = 1 / u
    q = (np.exp(r * dt) - d) / (u - d)

    # Initialize asset prices at maturity
```

```python
    stock_prices = np.zeros(N + 1)
    for i in range(N + 1):
        stock_prices[i] = S0 * (u ** (N - i)) * (d ** i)

    # Initialize option values at maturity
    if option_type == 'call':
        option_values = np.maximum(stock_prices - K, 0)
    else:  # put
        option_values = np.maximum(K - stock_prices, 0)

    # Backward induction
    for t in range(N - 1, -1, -1):
        for i in range(t + 1):
            # Risk-neutral valuation
            continuation = np.exp(-r * dt) * (q * option_values[i] + (1 - q) * option_val

            if exercise == 'american':
                # Early exercise value
                stock_price = S0 * (u ** (t - i)) * (d ** i)
                if option_type == 'call':
                    exercise_value = max(stock_price - K, 0)
                else:
                    exercise_value = max(K - stock_price, 0)

                option_values[i] = max(continuation, exercise_value)
            else:
                option_values[i] = continuation

    return option_values[0]


# Example usage
S0 = 100      # Current stock price
K = 100       # Strike price
T = 1.0       # 1 year to expiration
r = 0.05      # 5% risk-free rate
sigma = 0.2   # 20% volatility
N = 100       # 100 time steps

call_price = binomial_tree_option(S0, K, T, r, sigma, N, 'call', 'european')
put_price = binomial_tree_option(S0, K, T, r, sigma, N, 'put', 'european')

print(f"European Call Price: ${call_price:.4f}")
print(f"European Put Price: ${put_price:.4f}")

# Verify put-call parity
parity_check = call_price - put_price - (S0 - K * np.exp(-r * T))
print(f"Put-Call Parity Error: {parity_check:.6f}")
```

Chapter 4: Black-Scholes-Merton Model

4.1 Assumptions

1. Stock price follows geometric Brownian motion:
$$dS_t = \mu S_t dt + \sigma S_t dW_t$$
2. No dividends, transaction costs, or taxes
3. Continuous trading possible
4. Constant risk-free rate r and volatility σ
5. No arbitrage opportunities

4.2 Black-Scholes PDE

The option price $V(S, t)$ satisfies:

$$\frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} = rV$$

with boundary condition $V(S, T) = \max(S - K, 0)$ for call.

4.3 Black-Scholes Formula

European Call:

$$C(S_0, K, T, r, \sigma) = S_0 N(d_1) - Ke^{-rT}N(d_2)$$

European Put:

$$P(S_0, K, T, r, \sigma) = Ke^{-rT}N(-d_2) - S_0 N(-d_1)$$

where:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T} = \frac{\ln(S_0/K) + (r - \sigma^2/2)T}{\sigma\sqrt{T}}$$

and $N(\cdot)$ is the cumulative standard normal distribution.

Interpretation:

- $N(d_1)$: Delta (hedge ratio)
- $N(d_2)$: Risk-neutral probability of exercise
- $S_0 N(d_1)$: Expected value of stock if exercised
- $Ke^{-rT}N(d_2)$: Expected cost if exercised

4.4 Python Implementation

```
from scipy.stats import norm
import numpy as np

def black_scholes(S0, K, T, r, sigma, option_type='call'):
    """
    Black-Scholes option pricing formula

    Parameters:
    -----------
    S0 : float - Current stock price
    K : float - Strike price
    T : float - Time to maturity (years)
    r : float - Risk-free rate
    sigma : float - Volatility (annualized)
    option_type : str - 'call' or 'put'

    Returns:
    --------
    float - Option price
    """

    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    if option_type == 'call':
        price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'put':
        price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
    else:
        raise ValueError("option_type must be 'call' or 'put'")

    return price


# Example
S0 = 100
K = 100
T = 1.0
r = 0.05
sigma = 0.2

call = black_scholes(S0, K, T, r, sigma, 'call')
put = black_scholes(S0, K, T, r, sigma, 'put')

print(f"Black-Scholes Call: ${call:.4f}")
print(f"Black-Scholes Put: ${put:.4f}")

# Verify put-call parity
parity = call - put - (S0 - K * np.exp(-r * T))
print(f"Put-Call Parity Check: {parity:.10f}")
```

Chapter 5: The Greeks

5.1 Definition and Intuition

The Greeks measure sensitivities of option prices to various parameters.

Delta (Δ): Rate of change of option price with respect to stock price

$$\Delta = \frac{\partial V}{\partial S}$$

Call Delta:

$$\Delta_C = N(d_1)$$

Put Delta:

$$\Delta_P = N(d_1) - 1$$

Interpretation:

- $\Delta = 0.5$: Option price changes by $0.50 for $1 stock price change
- Used for **delta hedging:** Hold $-\Delta$ shares to neutralize price risk

Gamma (Γ): Rate of change of delta with respect to stock price

$$\Gamma = \frac{\partial^2 V}{\partial S^2} = \frac{\partial \Delta}{\partial S}$$

Formula (same for call and put):

$$\Gamma = \frac{N'(d_1)}{S_0 \sigma \sqrt{T}}$$

where $N'(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ is the standard normal PDF.

Interpretation:

- High gamma: Delta changes rapidly (more frequent rehedging needed)
- Gamma highest for ATM options near expiration

Vega (ν or $\mathcal{V}$): Sensitivity to volatility

$$\nu = \frac{\partial V}{\partial \sigma}$$

Formula:

$$\nu = S_0 \sqrt{T} N'(d_1)$$

Interpretation:

- Vega positive for long options (benefit from volatility increase)

- Vega highest for ATM options with longer time to expiration

Theta (Θ): Rate of change with respect to time (time decay)

$$\Theta = \frac{\partial V}{\partial t} = -\frac{\partial V}{\partial T}$$

Call Theta:

$$\Theta_C = -\frac{S_0 N'(d_1)\sigma}{2\sqrt{T}} - rKe^{-rT}N(d_2)$$

Put Theta:

$$\Theta_P = -\frac{S_0 N'(d_1)\sigma}{2\sqrt{T}} + rKe^{-rT}N(-d_2)$$

Interpretation:

- Usually negative (options lose value as time passes)
- Accelerates near expiration

Rho (ρ): Sensitivity to interest rate

$$\rho = \frac{\partial V}{\partial r}$$

Call Rho:

$$\rho_C = KTe^{-rT}N(d_2)$$

Put Rho:

$$\rho_P = -KTe^{-rT}N(-d_2)$$

5.2 Python Implementation

```
def calculate_greeks(S0, K, T, r, sigma, option_type='call'):
    """
    Calculate option Greeks using Black-Scholes

    Returns:
    --------
    dict - Dictionary containing price and all Greeks
    """

    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)

    # Price
    if option_type == 'call':
        price = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
        delta = norm.cdf(d1)
        theta = (- (S0 * norm.pdf(d1) * sigma) / (2 * np.sqrt(T))
```

```python
                   - r * K * np.exp(-r * T) * norm.cdf(d2))
        rho = K * T * np.exp(-r * T) * norm.cdf(d2)
    else:  # put
        price = K * np.exp(-r * T) * norm.cdf(-d2) - S0 * norm.cdf(-d1)
        delta = norm.cdf(d1) - 1
        theta = (- (S0 * norm.pdf(d1) * sigma) / (2 * np.sqrt(T))
                 + r * K * np.exp(-r * T) * norm.cdf(-d2))
        rho = -K * T * np.exp(-r * T) * norm.cdf(-d2)

    # Gamma and Vega (same for call and put)
    gamma = norm.pdf(d1) / (S0 * sigma * np.sqrt(T))
    vega = S0 * np.sqrt(T) * norm.pdf(d1)

    return {
        'Price': price,
        'Delta': delta,
        'Gamma': gamma,
        'Vega': vega,
        'Theta': theta,
        'Rho': rho
    }


# Example
greeks_call = calculate_greeks(100, 100, 1.0, 0.05, 0.2, 'call')
print("Call Option Greeks:")
for greek, value in greeks_call.items():
    print(f"  {greek}: {value:.6f}")

greeks_put = calculate_greeks(100, 100, 1.0, 0.05, 0.2, 'put')
print("\nPut Option Greeks:")
for greek, value in greeks_put.items():
    print(f"  {greek}: {value:.6f}")
```

Chapter 6: Implied Volatility

6.1 Definition

Implied Volatility (σ_{imp}): The volatility that, when input into Black-Scholes, produces the observed market price.

Problem: Given market price V_{market}, solve for σ:

$$V_{market} = BS(S_0, K, T, r, \sigma_{imp})$$

No closed-form solution → use numerical methods.

6.2 Newton-Raphson Method

Iteration:

$$\sigma_{n+1} = \sigma_n - \frac{BS(\sigma_n) - V_{market}}{\text{Vega}(\sigma_n)}$$

Algorithm:

1. Initial guess: $\sigma_0 = 0.2$ (20%)
2. Iterate until convergence:

6.3 Python Implementation

```
def implied_volatility(market_price, S0, K, T, r, option_type='call',
                       max_iter=100, tol=1e-6):
    """
    Calculate implied volatility using Newton-Raphson

    Parameters:
    -----------
    market_price : float - Observed option price
    max_iter : int - Maximum iterations
    tol : float - Convergence tolerance

    Returns:
    --------
    float - Implied volatility
    """

    # Initial guess
    sigma = 0.2

    for i in range(max_iter):
        # Calculate price and vega at current sigma
        price = black_scholes(S0, K, T, r, sigma, option_type)

        d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
        vega = S0 * np.sqrt(T) * norm.pdf(d1)

        # Newton-Raphson update
        diff = price - market_price

        if abs(diff) &lt; tol:
            return sigma

        sigma = sigma - diff / vega

        # Ensure positive volatility
        if sigma &lt;= 0:
            sigma = 0.01

    raise ValueError(f"Implied volatility did not converge after {max_iter} iterations")
```

```python
# Example
market_call_price = 10.45
S0, K, T, r = 100, 100, 1.0, 0.05

implied_vol = implied_volatility(market_call_price, S0, K, T, r, 'call')
print(f"Implied Volatility: {implied_vol:.4f} ({implied_vol*100:.2f}%)")

# Verify
reconstructed_price = black_scholes(S0, K, T, r, implied_vol, 'call')
print(f"Market Price: ${market_call_price:.4f}")
print(f"Reconstructed Price: ${reconstructed_price:.4f}")
```

6.4 Volatility Smile and Skew

Observed Pattern: Implied volatility varies with strike price:

- **Volatility Smile:** Higher IV for OTM and ITM options
- **Volatility Skew:** Higher IV for OTM puts (downside protection)

Causes:

- Fat tails in return distributions (Black-Scholes assumes normality)
- Jump risk and crashes
- Supply/demand dynamics

Chapter 7: Monte Carlo Simulation

7.1 Methodology

Idea: Simulate many price paths, calculate payoff on each, average and discount.

Algorithm:

1. Generate N random price paths using risk-neutral dynamics
2. Calculate option payoff for each path
3. Average payoffs and discount at risk-free rate

Stock Price Simulation:

$$S_{t+\Delta t} = S_t \exp\left[\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}\, Z\right]$$

where $Z \sim N(0,1)$.

7.2 Python Implementation

```
def monte_carlo_option(S0, K, T, r, sigma, n_simulations=100000,
                      option_type='call', n_steps=252):
    """
    Price European option using Monte Carlo simulation

    Parameters:
    -----------
    n_simulations : int - Number of price paths
    n_steps : int - Time steps per path

    Returns:
    --------
    tuple - (option_price, standard_error)
    """

    dt = T / n_steps

    # Generate random paths
    Z = np.random.standard_normal((n_simulations, n_steps))

    # Initialize price paths
    S = np.zeros((n_simulations, n_steps + 1))
    S[:, 0] = S0

    # Simulate paths
    for t in range(1, n_steps + 1):
        S[:, t] = S[:, t-1] * np.exp((r - 0.5 * sigma**2) * dt +
                                      sigma * np.sqrt(dt) * Z[:, t-1])

    # Calculate payoffs at maturity
    if option_type == 'call':
        payoffs = np.maximum(S[:, -1] - K, 0)
    else:  # put
        payoffs = np.maximum(K - S[:, -1], 0)

    # Discount to present value
    option_price = np.exp(-r * T) * np.mean(payoffs)

    # Standard error
    standard_error = np.exp(-r * T) * np.std(payoffs) / np.sqrt(n_simulations)

    return option_price, standard_error


# Example
mc_call, mc_se = monte_carlo_option(100, 100, 1.0, 0.05, 0.2,
                                    n_simulations=100000, option_type='call')
bs_call = black_scholes(100, 100, 1.0, 0.05, 0.2, 'call')

print(f"Monte Carlo Call Price: ${mc_call:.4f} ± ${mc_se:.4f}")
print(f"Black-Scholes Call Price: ${bs_call:.4f}")
print(f"Difference: ${abs(mc_call - bs_call):.4f}")
```

7.3 Variance Reduction Techniques

1. Antithetic Variates:
For each random path Z, also simulate $-Z$.

2. Control Variates:
Use known analytical price of similar security to reduce variance.

3. Importance Sampling:
Sample more from regions that contribute most to payoff.

Chapter 8: Exotic Options

8.1 Asian Options

Payoff depends on average price:

$$\text{Call Payoff} = \max\left(\frac{1}{n}\sum_{i=1}^{n} S_{t_i} - K, 0\right)$$

No closed-form solution → Monte Carlo or approximations.

8.2 Barrier Options

Knock-Out: Option becomes worthless if barrier crossed
Knock-In: Option activates if barrier crossed

Example - Down-and-Out Call:

$$\text{Payoff} = \begin{cases} \max(S_T - K, 0) & amp; \text{if } \min_{0 \leq t \leq T} S_t \quad gt; B \\ 0 & amp; \text{otherwise} \end{cases}$$

8.3 Lookback Options

Payoff depends on maximum/minimum price:

$$\text{Call Payoff} = \max_{0 \leq t \leq T} S_t - K$$

Chapter 9: Practical Applications

9.1 Delta Hedging Strategy

```
def delta_hedge_simulation(S0, K, T, r, sigma, n_days=252):
    """
    Simulate delta hedging strategy
    """

    dt = T / n_days
```

```
    hedging_errors = []

    # Simulate stock price path
    np.random.seed(42)
    S = [S0]
    for _ in range(n_days):
        dS = S[-1] * (r * dt + sigma * np.sqrt(dt) * np.random.randn())
        S.append(S[-1] + dS)

    # Initial option position
    portfolio_value = []

    for i, St in enumerate(S[:-1]):
        t = i * dt
        time_to_expiry = T - t

        # Calculate delta
        d1 = (np.log(St / K) + (r + 0.5 * sigma**2) * time_to_expiry) / \
             (sigma * np.sqrt(time_to_expiry))
        delta = norm.cdf(d1)

        # Option price
        option_price = black_scholes(St, K, time_to_expiry, r, sigma, 'call')

        # Hedge: Short delta shares
        hedge_value = -delta * St

        # Total portfolio
        pv = option_price + hedge_value
        portfolio_value.append(pv)

    return S, portfolio_value
```

9.2 Trading Strategies

Covered Call: Long stock + Short call (income generation)
Protective Put: Long stock + Long put (downside protection)
Bull Spread: Long call at K_1 + Short call at
Straddle: Long call + Long put (volatility play)

Chapter 10: Further Topics

10.1 Dividends

Continuous dividend yield q:
Modify Black-Scholes:

$$C = S_0 e^{-qT} N(d_1) - K e^{-rT} N(d_2)$$

where:

$$d_1 = \frac{\ln(S_0/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$$

10.2 American Options

No closed-form solution. Methods:

- Binomial trees with early exercise check
- Least Squares Monte Carlo (LSM)
- Finite difference PDE solvers

10.3 Interest Rate Derivatives

- **Caps/Floors:** Options on interest rates
- **Swaptions:** Options on interest rate swaps
- Models: Vasicek, Cox-Ingersoll-Ross (CIR), Hull-White

Further Reading

- Hull: *Options, Futures, and Other Derivatives*
- Wilmott: *Paul Wilmott on Quantitative Finance*
- Shreve: *Stochastic Calculus for Finance II*
- Joshi: *The Concepts and Practice of Mathematical Finance*

This textbook provides rigorous mathematical foundations and complete Python implementations for derivatives pricing, from fundamentals through advanced topics.