COMPUTATIONAL FINANCE / SESSION 04

SECTION A-A

Python for Monte Carlo: Improving Accuracy with Variance Reduction

Advanced Optimization Techniques

REF: MC-04

VARIANCE REDUCTION METHODS | PYTHON IMPLEMENTATION

SECTION 04.01

Goal

OPTIMIZATION TARGET

PRIMARY OBJECTIVE

Explore **variance reduction techniques** in Python to improve the accuracy and efficiency of **Monte Carlo simulations** for option pricing.

SPEC: VARIANCE ↓ | ACCURACY ↑

SECTION 04.02

Main Topics

MODULE 01

REF: 04.03

Monte Carlo Review

Risk-neutral pricing and simulation fundamentals

MODULE 02

REF: 04.07

Variance Reduction Techniques

Antithetic Variates and Control Variates implementation

MODULE 03

REF: 04.20

Method Comparison

Monte Carlo vs. Black-Scholes trade-offs

Monte Carlo Pricing: Intuition

STEP 01

Simulate Stock Price Paths

Generate N random paths for the stock price S_T under risk-neutral dynamics

STEP 02

Compute Payoffs

For each path i, calculate the option payoff: $P_i = \max(S_{T,i} - K, 0)$

STEP 03

Average and Discount

Take the mean of all payoffs and discount to present value: $V = e^{-rT} \times (1/N) \Sigma P_i$

Risk-Neutral Pricing Formula

FUNDAMENTAL EQUATION

$$V = e^{-rT} \times E\mathbb{Q}[\text{Payoff}]$$

Under the **risk-neutral measure $\mathbb{Q}$**, the option value is the **discounted expected payoff**. Monte Carlo approximates this expectation by averaging simulated payoffs.

Monte Carlo Simulation Steps

01 `Z = np.random.standard_normal(N)`

02 `S_T = S0 × exp((r - 0.5σ²)T + σ√T × Z)`

03 `Payoff = max(S_T - K, 0)`

04 `V = exp(-rT) × mean(Payoff)`

Variance Reduction Techniques

SYSTEM LIMITATION

Standard Monte Carlo has **slow convergence**: error decreases as **$O(1/\sqrt{N})$**. Achieving high accuracy requires millions of simulations.

OPTIMIZATION APPROACH

Variance reduction techniques exploit correlation and known quantities to reduce noise, achieving the same accuracy with **fewer simulations**.

Monte Carlo Error: Signal

PROPERTY 01: UNBIASED ESTIMATOR

Expected Value

$$E[\hat{V}_{MC}] = V_{true}$$

The Monte Carlo estimator is **unbiased**: on average, it equals the true option value.

INTERPRETATION

This means the **signal** (mean) is correct. The challenge is the **noise** (variance) around this mean.

Monte Carlo Error: Noise

VARIANCE FORMULA

$$\mathrm{Var}(\hat{V}_{MC}) = \sigma^2 / N$$

CONVERGENCE RATE

Standard error decreases as $O(1/\sqrt{N})$. To halve the error, you need **4× more simulations**. This is the fundamental limitation we aim to overcome.

Antithetic Variates

- If $Z \sim N(0,1)$ drives a path, also use $-Z$
- These two paths move in **opposite directions**
- Averaging the payoffs **cancels noise**

ADJUSTMENT FORMULA

$$V_{anti} = (1/2) \times (P(Z) + P(-Z))$$

BENEFIT: Works exceptionally well for monotonic payoffs like European calls

Why Antithetic Variates Work Well

Stock price randomness comes from a **single normal variable Z**

STOCK PRICE FORMULA

$$S_T = S_0 \times \exp((r - 0.5\sigma^2)T + \sigma\sqrt{T} \times Z)$$

SECTION 04.11 | MECHANISM ANALYSIS

Antithetic Pair: Opposite Shocks

PATH 01

Using +Z

$Z > 0$

Stock price moves UPWARD

Payoff tends to be HIGH

PATH 02

Using -Z

$-Z < 0$

Stock price moves DOWNWARD

Payoff tends to be LOW

Negative Correlation Reduces Variance

VARIANCE FORMULA

$$\mathrm{Var}((X + Y) / 2) = (\mathrm{Var}(X) + \mathrm{Var}(Y) + 2\mathrm{Cov}(X,Y)) / 4$$

When $\mathrm{Cov}(X, Y) < 0$ (negative correlation), the variance of the average is **smaller** than if X and Y were independent. Antithetic variates exploit this by creating **strong negative correlation**.

Antithetic Variates in Python

CODE SPECIFICATION

```python
def mc_call_antithetic(S0, K, r, sigma, T, N):
    # Generate N/2 standard normal random variables
    Z = np.random.standard_normal(N // 2)

    # Compute stock prices for both Z and -Z
    S_T_pos = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)
    S_T_neg = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * (-Z))

    # Compute payoffs for both paths
    payoff_pos = np.maximum(S_T_pos - K, 0)
    payoff_neg = np.maximum(S_T_neg - K, 0)

    # Average the antithetic pairs
    payoff_avg = (payoff_pos + payoff_neg) / 2

    # Discount and return
    return np.exp(-r * T) * np.mean(payoff_avg)
```

Control Variates

Use a **known quantity** (control variate) that is correlated with the unknown quantity to **correct simulation errors**.

ADJUSTMENT FORMULA

$$\hat{V}_{CV} = \hat{V}_{MC} - \beta \times (\bar{C} - E[C])$$

BENEFIT: We know $E[S_T]$ exactly, so we can use stock price as a control to correct option price estimates

The "Pull" Mechanism

CASE 01

$\bar{C} > E[C]$

Simulated stock prices are too high on average

Option estimate $\hat{V}_{MC}$ is likely too high

SUBTRACT to correct downward

CASE 02

$\bar{C} < E[C]$

Simulated stock prices are too low on average

Option estimate $\hat{V}_{MC}$ is likely too low

ADD to correct upward

Deriving Optimal β (Step 1)

Find the value of β that **minimizes** the variance of the control variate estimator

VARIANCE FORMULA

$$\mathrm{Var}(\hat{V}_{CV}) = \mathrm{Var}(\hat{V}_{MC}) + \beta^2 \mathrm{Var}(C) - 2\beta\, \mathrm{Cov}(\hat{V}_{MC}, C)$$

Deriving Optimal β (Step 2)

DERIVATIVE

$$d/d\beta \ \mathrm{Var}(\hat{V}_{CV}) = 2\beta \mathrm{Var}(C) - 2\mathrm{Cov}(\hat{V}_{MC}, C) = 0$$

OPTIMAL COEFFICIENT

$$\beta^* = \mathrm{Cov}(\hat{V}_{MC}, C) / \mathrm{Var}(C)$$

β* is Regression Slope

INTERPRETATION

$\beta^* = \mathrm{Cov}(\hat{V}_{MC}, C) / \mathrm{Var}(C)$ is the **slope** in the regression of $\hat{V}_{MC}$ on C

PRACTICAL INSIGHT: In Python, we can estimate β* using np.cov() to compute the covariance matrix of simulated option prices and stock prices

Control Variates in Python

CODE SPECIFICATION

```
def mc_call_control_variates(S0, K, r, sigma, T, N):
    # Generate random shocks
    Z = np.random.standard_normal(N)
    S_T = S0 * np.exp((r - 0.5 * sigma**2) * T + sigma * np.sqrt(T) * Z)

    # Compute payoffs and control variate
    payoff = np.maximum(S_T - K, 0)
    control = S_T  # Stock price is the control variate

    # Expected value of control under risk-neutral measure
    E_control = S0 * np.exp(r * T)

    # Estimate optimal beta using covariance
    cov_matrix = np.cov(payoff, control)
    beta = cov_matrix[0, 1] / cov_matrix[1, 1]

    # Apply control variate adjustment
    payoff_cv = payoff - beta * (control - E_control)

    # Discount and return
    return np.exp(-r * T) * np.mean(payoff_cv)
```

Analysis: Monte Carlo Method

STRENGTHS

- **Flexible:** Works for any payoff structure
- **Path-dependent:** Can handle exotic options
- **Multi-dimensional:** Handles multiple assets

LIMITATIONS

- **Slow convergence:** $O(1/\sqrt{N})$ error rate
- **Computational cost:** Needs many simulations for accuracy
- **Random error:** Results vary between runs

System: Black-Scholes Formula

STRENGTHS

- **Exact:** Provides analytical solution
- **Fast:** Instant computation
- **Deterministic:** No random error

LIMITATIONS

- **Limited scope:** Only European options
- **Requires formula:** Needs closed-form solution
- **Assumptions:** Constant volatility, no dividends

Session 4: Summary

[PROBLEM] Standard Monte Carlo convergence is slow ($1/\sqrt{N}$)

[SOLVED] **Antithetic Variates:** Use pairs (Z, -Z) to induce negative correlation

[SOLVED] **Control Variates:** Use known expectations to correct simulation errors

[RESULT] Achieved higher accuracy with fewer simulations

>> OPTIMIZATION COMPLETE. SYSTEM READY.