Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Analytical Finance I
Course Code: MMA707

HEDGING OF A VANILLA CALL OPTION USING THE UNDERLYING ASSET

Romann IVANOFF
roman.ivanoff@edu.devinci.fr

Rémi LIEBIG
remi.liebig@edu.devinci.fr

Vincent NAZZARENO
vincent.nazzareno@edu.devinci.fr

Examiner: Jan Römann
Mälardalen University, Västerås

17th December 2024

Introduction

Delta hedging is a key strategy in options trading that helps to minimize the risks caused by price changes in the underlying asset. In this report, we examine how to apply delta hedging using the Black-Scholes model. We improve the model by including volatility estimates from GARCH models and VIX proxies to make the strategy more reliable. Our main objective is to simulate the performance of this hedging strategy with different rebalancing and rolling frequencies, while also taking into account transaction costs and other real-world factors. The code can be viewed on the GitHub/MMA707 and the cloud dashboard here.

Methodology: Theory of Delta Hedging

Black-Scholes Model

The Black-Scholes model is a popular way to price European-style options. It works under the assumption that the price of the underlying asset follows a geometric Brownian motion with constant volatility and interest rates. The formula for a European call option is:

$$C(S,t) = S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2)$$

where:

- S is the current price of the underlying asset.
- K is the strike price of the option.
- r is the risk-free interest rate.
- T is the time to maturity.
- $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution.

With:

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

Delta and Its Role in Hedging

Delta (Δ) indicates how the option price changes with small movements in the underlying asset's price. It is the first derivative of the option price with respect to the asset price:

$$\Delta = \frac{\partial C}{\partial S} = \Phi(d_1)$$

In delta hedging, we aim to create a delta-neutral portfolio, meaning the portfolio's value remains relatively stable against small price changes in the underlying. For a long call option, delta is positive, so we hedge by shorting Δ units of the underlying asset.

Gamma, Theta, and Other Greeks

Other Greeks also play important roles in options trading and hedging:

- **Gamma (Γ)**: Delta's rate of change with the underlying asset price. High gamma increases the need for frequent rebalancing.
- **Theta (Θ)**: Time decay. Long options lose value as time passes.
- **Vega (ν)**: Sensitivity to changes in volatility.

Transaction Costs and Practical Considerations

Continuous delta hedging is an idealized concept, but real-world challenges make its implementation complex. One significant issue is transaction costs, which stems from factors such as bid-ask spreads, commissions, and market slippage. Frequent rebalancing, especially in volatile markets, can exacerbate these costs. Additionally, rebalancing occurs at discrete intervals rather than continuously, leading to potential mismatches between the portfolio's delta and the underlying asset's movements. Another challenge is the accurate estimation of implied volatility, as any miscalculation can result in incorrect hedge adjustments and suboptimal performance. Lastly, the underlying assumptions of the Black-Scholes model—such as constant volatility and interest rates—are very rarely valid in practice, further complicating the possibility of delta hedging in real markets.

These factors often make a delta-hedged long option position lose value over time.

Implementation

Data Collection and Preprocessing

We use historical data (via `yfinance`) for the underlying asset, risk-free rates, and volatility proxies:

- **Stock Data**: Historical closing prices to compute log returns.
- **Risk-Free Rate**: From a financial instrument (5-Year Treasury for example) and annualized.
- **Volatility Proxy**: GARCH or VIX or similar implied volatility measures.

Volatility Estimation

We consider several methods:

1. **GARCH Model**: Models conditional variance for a smooth annualized volatility estimate.
2. **VIX Proxy**: Implied volatility from indices (VIX for SPY, VXN for NASDAQ 100 etc.).
3. **Realized Volatility**: Computed via rolling standard deviations (`pandas.rolling(window).std()`)

Delta Hedging Strategy

The simulation involves:

- **Initial Positioning**: Buy a call option and short Δ shares of the underlying. The strike is ideally chosen to be K_multiplier * S.
- **Rebalancing**: At set intervals (daily, weekly,), adjusting the hedge, considering transaction costs and time decay.

Results

Using the Black-Scholes model with GARCH-estimated volatility, we analyze the portfolio's performance, transaction costs, and Greek behaviors over 15 years. We chose the following parameters:

- $K_{multiplier} = 1.1$
- $Toptionmaturity = 6$ months
- $Volatility$: GARCH
- $r_{risk-free}$: FVX
- $day_{rebalancing,stock} = 1$
- $day_{rolling,option} = -1$ ie no rolling, we let expire everytime the option.
- $fee_{rate} = 1\%$

Portfolio Performance

We start with an initial investment of $\Delta S - N_c C = \$31$ on SPY. After `15` years, the final portfolio value is `-$552`, mostly due to transaction costs, volatility, options losing value due to theta decay.

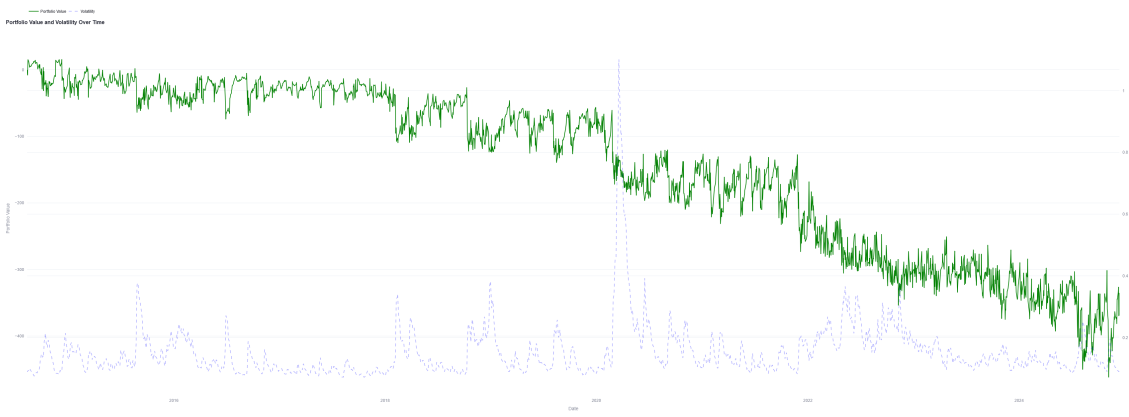

Figure 1: Portfolio Performance Over Time with GARCH Volatility

Figure 1 shows how the portfolio's value evolves over time. The volatility is plotted to show the impact of gamma risk during periods of high volatility and how the portfolio is not gamma neutral.

Stock and Strike Price Over Time

Comparing the stock and strike prices helps us understand volatility-induced slippage, particularly for $K = 1.1 \times S$.

Figure 2 highlights the movement of stock prices relative to the strike price over the simulation period. It is difficult to preserve the portfolio value in times of high volatility.

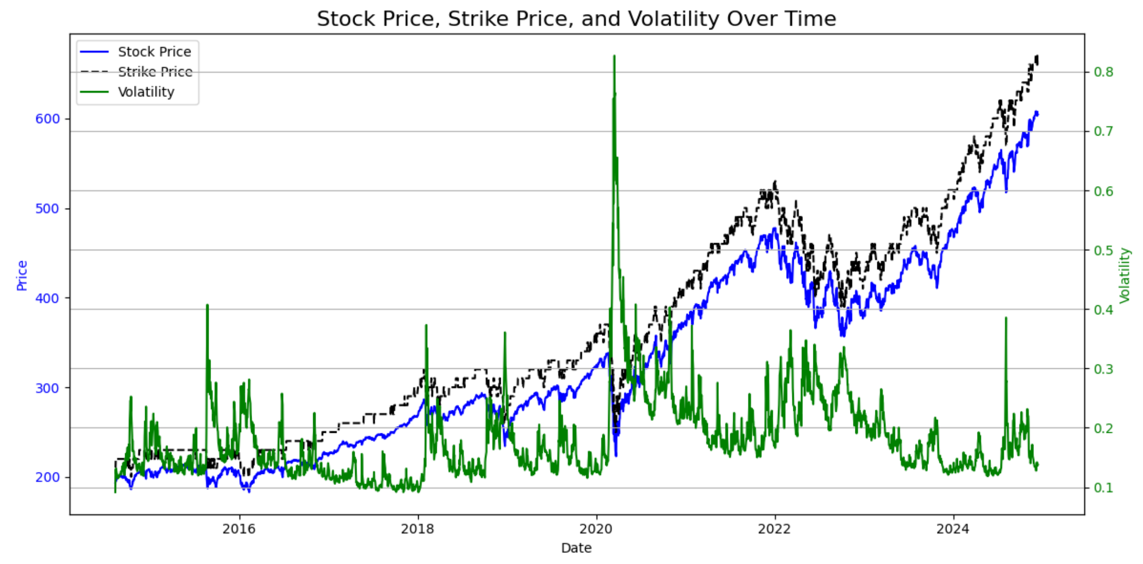

Figure 2: Stock Price and Strike Price Over Time

Cumulative Fees and Delta Drift

Transaction costs and delta drift are the main driver of portfolio performance.

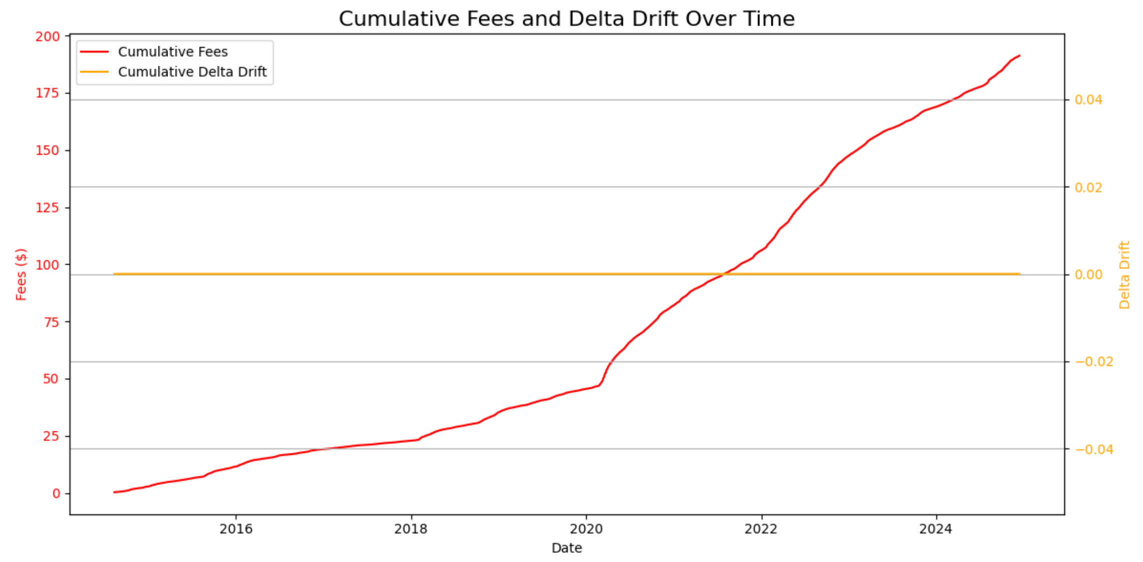

Figure 3: Cumulative Fees and Delta Drift

Figure 3 shows how transaction costs accumulate over time, totaling $191. We see that during high volatility, it is more expensive to hedge the delta of the portfolio. However, delta drift was negligible (but this is due to no slippage in execution, one of the shortcomings of simple simulations), indicating that the portfolio remained delta-neutral.

Delta Over Time of the Option

The delta of the option changes with the stock price and volatility, needing regular rebalancing to be delta hedged. It is also more expensive to hedge a portfolio during high volatility, due to the added costs of the options.

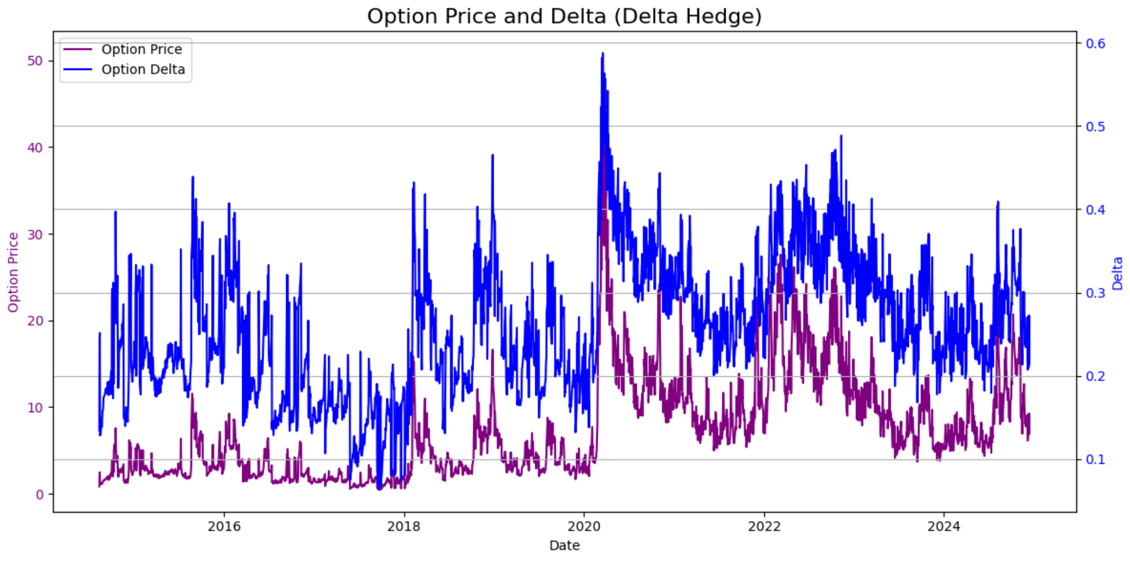

Figure 4: Delta of the Option Over Time

Figure 4 shows the evolution of the option's price and option's delta, which respond dynamically to market changes and market state.

Parameter Sweep Analysis: GARCH vs. VIX Volatility

To compare performance under different rebalancing frequencies, strike multipliers (K), and option maturities, we analyze the results for GARCH- and VIX-based volatility models. The heatmaps in Figures 5 and 6 summarize these interactions.

Table 1 summarizes the results:

Table 1: Summary of Portfolio Performance Metrics

Metric	**Observation**
Initial Portfolio Value	$\Delta S - N_C C = \$31$
Final Portfolio Value	-$543, primarily due to transaction costs, volatility-induced losses, and theta decay.
Cumulative Transaction Costs	$191
Delta Drift	Negligible; the portfolio remained effectively delta-neutral.
Stock and Strike Price Dynamics	Significant vol slippage observed for $K = 1.1 \times S$, leading to hedging challenges in high volatility market states.
Impact of Rebalancing Frequency	Short rebalancing intervals increased losses due to transaction costs.
Impact of K$_{multiplier}$	Far Strike lead to better results due to the decreasing effect of Γ on option price.

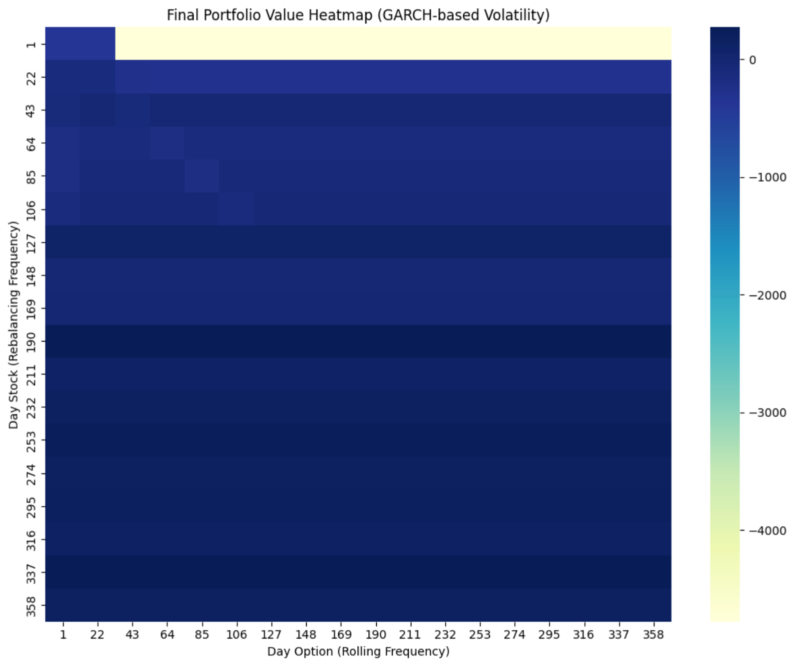

Figure 5: Final Portfolio Value Heatmap (GARCH-based Volatility)

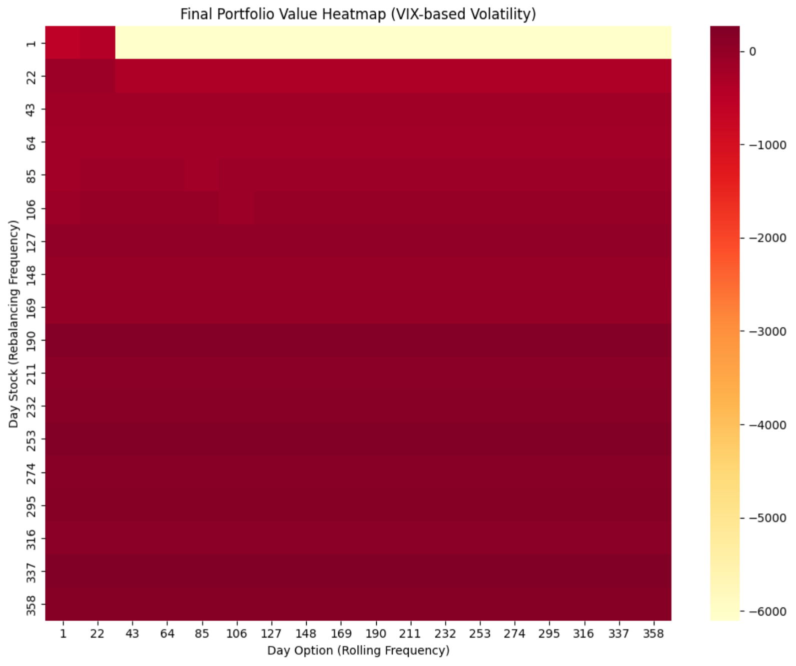

Figure 6: Final Portfolio Value Heatmap (VIX-based Volatility)

Parameter Sweep Analysis: Impact of Rebalancing, Strike, and Maturity

To further analyze the portfolio performance, we explored the interactions between rebalancing frequency (day_stock), strike multiplier ($K\_multiplier$), and option maturity (T). The following 3D surface plots provide insight into these parameter relationships and their effect on the final portfolio value.

Figure 7 shows that shorter rebalancing intervals (day_stock) combined with far out-of-the-money strikes ($K > 1.1$) lead to significantly lower losses due to the decreasing gamma sensitivity.

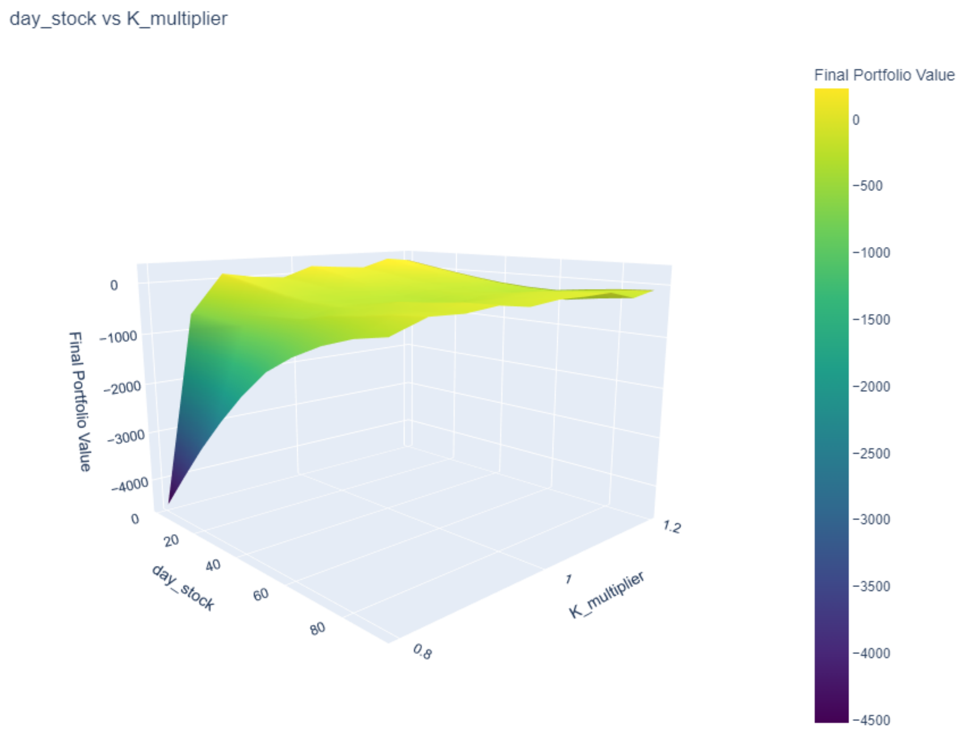

Figure 7: Impact of *day_stock* and *K_multiplier* on Final Portfolio Value.

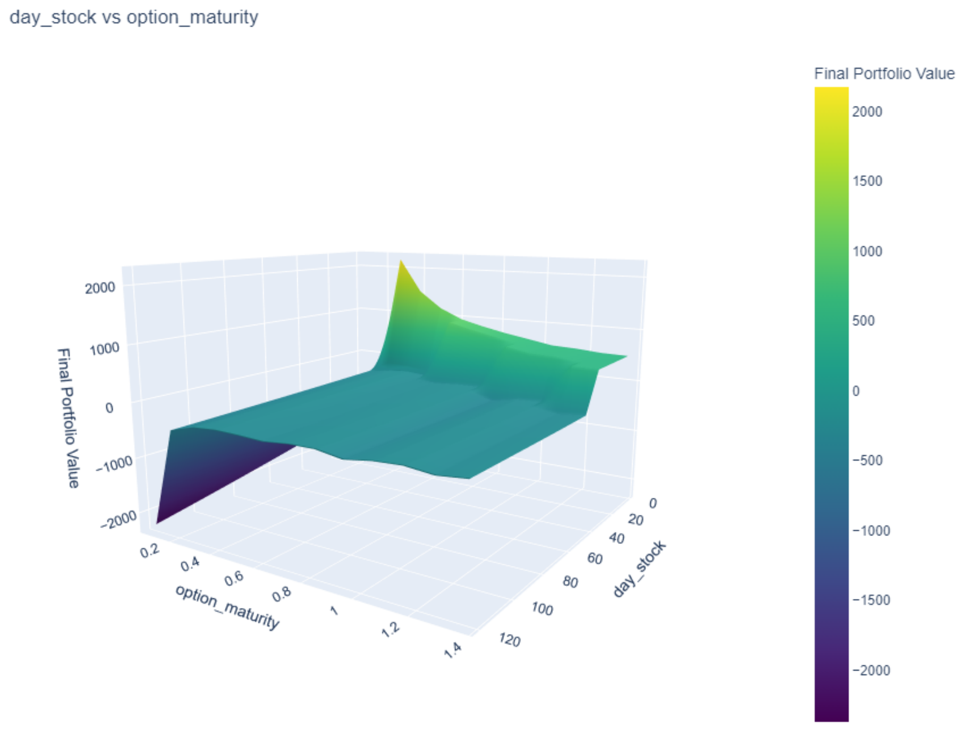

Figure 8: Impact of *day_stock* and Option Maturity on Final Portfolio Value.

Figure 8 illustrates that longer option maturities ($T > 0.5$) and higher *day_stock* values stabilize the portfolio performance, reducing the impact of transaction costs and frequent gamma adjustments.

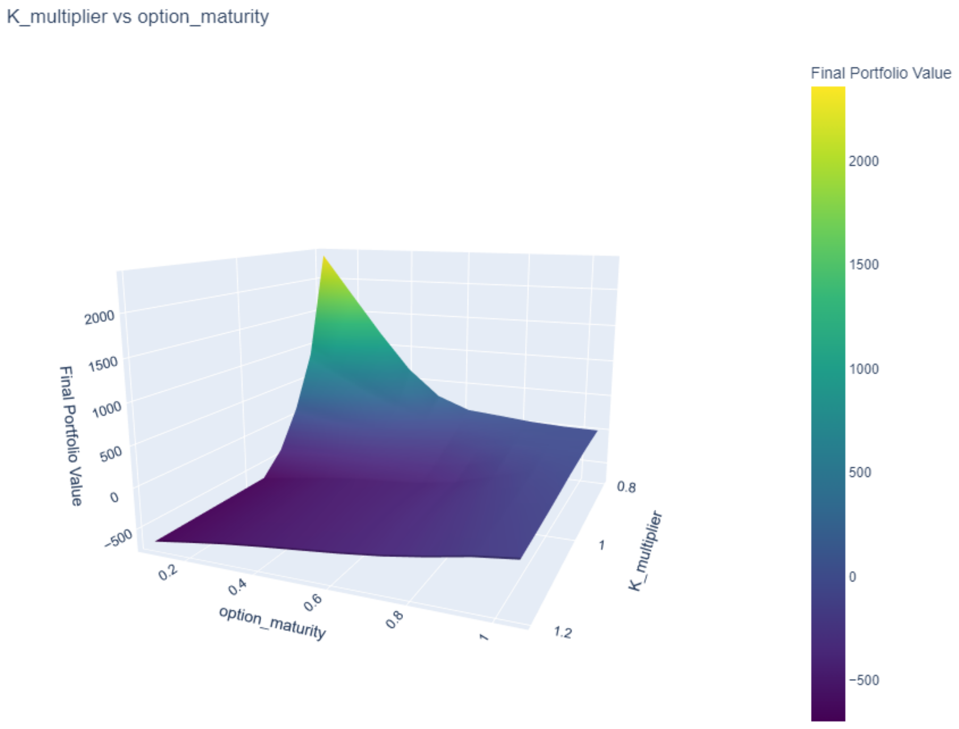

Figure 9: Impact of $K\_multiplier$ and Option Maturity on Final Portfolio Value.

Figure 9 reveals that far out-of-the-money strikes ($K\_multiplier > 1.1$) combined with longer option maturities (T) produce the least portfolio losses, as the reduced gamma sensitivity outweighs the increased theta decay.

Table 2 summarizes different the parameter sensibility on portfolio value:

Table 2: Impact of Key Parameters on Portfolio Performance

Parameter Combination	**Observation**
day_stock **vs** *K_multiplier*	Short rebalancing intervals combined with $K > 1.1$ reduce losses due to lower gamma sensitivity.
day_stock **vs Maturity** (T)	Longer option maturities stabilize performance, reducing gamma-induced losses despite theta decay.
K_multiplier **vs Maturity** (T)	Far strikes and longer maturities yield better results as gamma's impact diminishes over time.

Conclusion

Our simulation shows a gradual decline in portfolio value over the backtest period (15 years) for a six-month call option, primarily due to transaction costs, time decay, and gamma risk. These results align with theoretical expectations when continuously rolling long call positions.

We assume a fixed 1% transaction cost and constant liquidity for options at $K = 1.1 \times S$. Real-world conditions vary, and liquidity issues may arise. The Black-Scholes model's assumptions (constant volatility and interest rates, no dividends, and frictionless markets) may not hold. Our volatility estimation (GARCH and VIX) also has limitations. Additionally, focusing mainly on delta ignores other Greeks, and most specifically the gamma risk as we have seen throughtout the results, and using a specific historical period can lead to bias of the backtest (only very high volatility state).

Future improvements include more realistic transaction cost models, advanced option pricing models (Heston, SABR), adaptive rebalancing strategies sensitive to real-time volatility and market conditions, a wider set of Greeks, and tests across multiple time periods and underlying assets. Incorporating dividends, taxes, and other real-world frictions will further refine the hedging strategy's practical utility at a trading desk in a bank or asset management firm.

Diagram of Functions Interaction of the Streamlit Dashboard App

The Function Interaction Diagram in Figure 10 shows how the main functions of the Streamlit app work together. It starts with the user interacting with the *User Interface* (the sidebar) to choose inputs like the stock ticker, volatility model, and rebalancing frequency.

The `get_data()` function collects and prepares market data, risk-free rates, and volatility measures using Yahoo Finance. Then, the `build_initial_positions()` function sets up the portfolio, calculates option prices with the Black-Scholes model, and determines the initial hedging positions. The `black_scholes()` function relies on the `compute_d1_d2()` function to compute the option prices and deltas needed for hedging.

Next, the `simulate_trades()` function runs the portfolio simulation, adjusting positions over time based on the user's inputs and parameters. To measure the risk, the `calculate_beta()` function estimates the portfolio's sensitivity compared to a benchmark. All the processed data is temporarily stored in the *Data Storage* for fast loading. Finally, the results, such as the portfolio value and hedging performance, are displayed through tables and interactive `plotly`charts in the *Results Visualization* section.

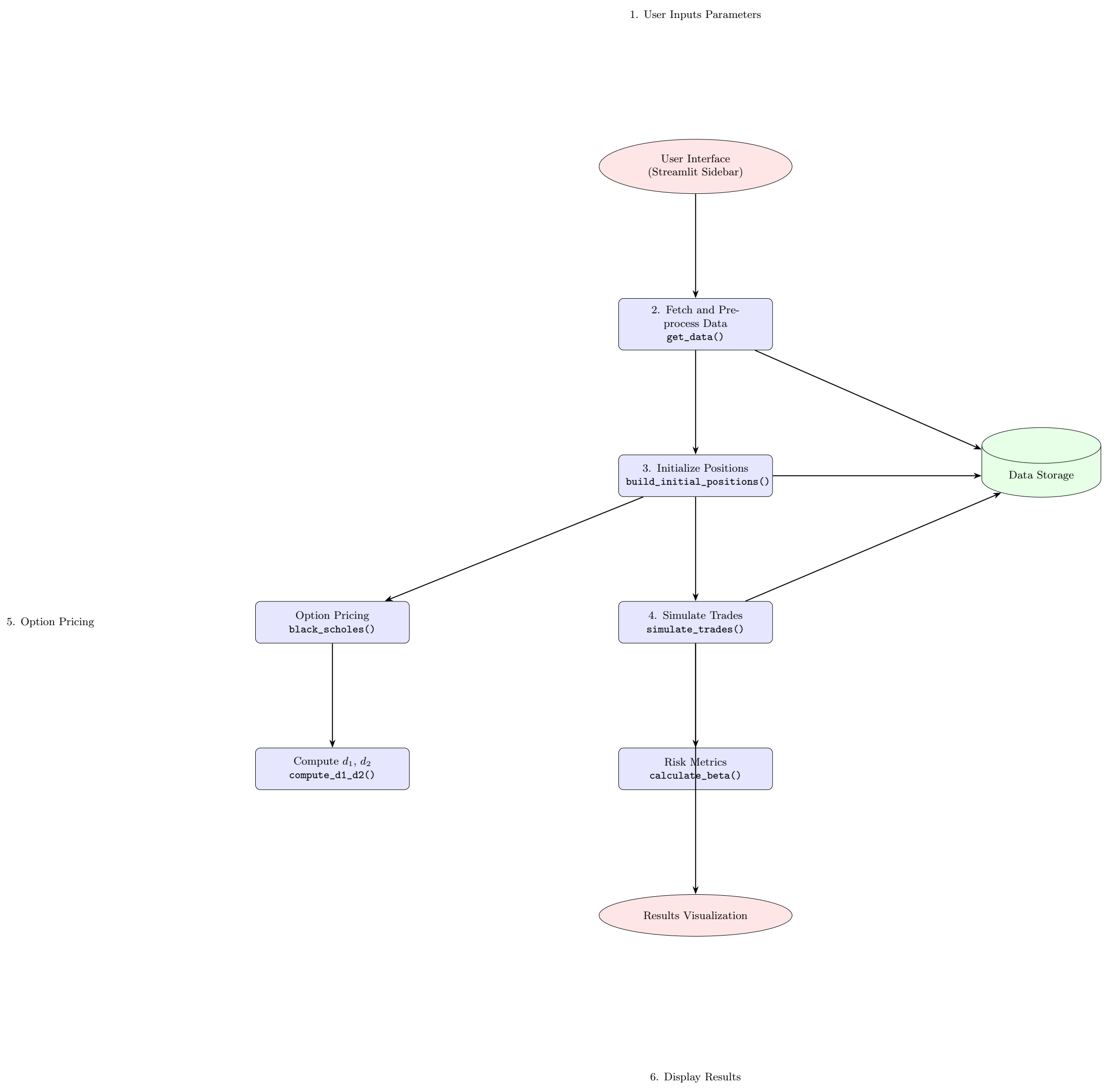
1. User Inputs Parameters
User Interface (Streamlit Sidebar)
2. Fetch and Pre-process Data get_data()
Data Storage
3. Initialize Positions build_initial_positions()
5. Option Pricing
Option Pricing black_scholes()
4. Simulate Trades simulate_trades()
Compute d_1, d_2 compute_d1_d2()
Risk Metrics calculate_beta()
Results Visualization
6. Display Results

Figure 10: Function Interaction Diagram for Streamlit Web App

Python Code for Delta Hedging Strategy

The Python code for the essential functions used in our delta hedging strategy is provided below. These functions handle data acquisition, volatility estimation, option pricing, portfolio initialization, and trade simulation.

```python
def get_data(symbol, risk_free, volatility_proxy, time_period):
    end_date = datetime.date.today()
    start_date = end_date - datetime.timedelta(days=time_period * 252)

    stock_data = yf.download(symbol, start=start_date, end=end_date)
    stock_data['log_returns'] = np.log(stock_data['Close'] /
        stock_data['Close'].shift(1))
    stock_data.dropna(subset=['log_returns'], inplace=True)

    risk_free_data = yf.download(risk_free, start=start_date,
        end=end_date)[['Close']] / 100
    risk_free_data.columns = ["Rate"]

    volatility_proxy_data = yf.download(volatility_proxy, start=start_date,
        end=end_date)[['Close']]
    volatility_proxy_data.columns = ['Vol_Proxy']

    volatility_proxy_data.loc[:, 'implied_volatility'] =
        volatility_proxy_data['Vol_Proxy'] / 100.0

    data = pd.merge(stock_data, risk_free_data, how='left', on='Date')
    data = pd.merge(data, volatility_proxy_data[['implied_volatility']],
        how='left', on='Date')

    data.ffill(inplace=True)

    ##### ESTIMATE VOLATILITY OF STOCK WITH GARCH MODEL #####
    garch_model = arch_model(data['log_returns'] * 100, vol='Garch', p=1, q=1,
        mean='Zero', dist='normal')
    garch_fit = garch_model.fit(disp="off")
    data['volatility'] = garch_fit.conditional_volatility
    data['annualized_volatility'] = data['volatility'] * np.sqrt(252) / 100
    data['smoothed_annualized_volatility'] =
        data['annualized_volatility'].rolling(window=3, min_periods=1).mean()
    data.dropna(subset='smoothed_annualized_volatility', inplace=True)

    plt.figure(figsize=(10, 12))
    plt.subplot(3, 1, 1)
    plt.plot(data.index, data['Close'], label=f"{symbol}", color='blue')
    plt.title(f"{symbol} Price")

    plt.subplot(3, 1, 2)
    plt.plot(data.index, data['implied_volatility'], label=f"Implied Vol
        ({volatility_proxy}-based)", color='red')
    plt.plot(data.index, data['annualized_volatility'], label="GARCH Estimated
        Annual Vol", ls="--", alpha=0.5)
    plt.title(f"Volatilities ({volatility_proxy} and GARCH)")
    plt.legend()

    plt.subplot(3, 1, 3)
    plt.plot(data.index, data['Rate'], label="Risk Free Rate", color='black')
    plt.title("Risk Free Rate")
    plt.legend()

    plt.tight_layout()
    plt.show()

    return data


def compute_d1_d2(S, K, r, sigma, T):
    d1 = (np.log(S / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return d1, d2


def black_scholes(S, K, T, r=0.02, sigma=0.2, option_type='call'):
    d1, d2 = compute_d1_d2(S, K, r=r, sigma=sigma, T=T)
    if option_type == 'call':
        c = S * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    elif option_type == 'put':
        c = K * np.exp(-r * T) * norm.cdf(-d2) - S * norm.cdf(-d1)
    else:
        raise ValueError("Error, option must be of type 'call' or 'put'")
    return c, norm.cdf(d1)


def value_bank(S, N_s, C, N_c):
    bank = -N_c * C + (-N_s) * S
    return bank


def build_initial_positions(data, fees=1/100, K_multiplier=1.1,
    use_vix_proxy=False, use_constant_rate=False, risk_free=0.02,
    use_realized_vol=False, use_ewma=False, use_atr=False, option_maturity=6/12):
    N_call = 1
    S0 = data['Close'].iloc[0]
    K0 = int(round(K_multiplier * S0, -1))
    if use_vix_proxy:
        g0 = data['implied_volatility'].iloc[0]
    elif use_realized_vol:
        g0 = data['realized_volatility'].iloc[0]
    elif use_ewma:
        g0 = data['ewma_volatility'].iloc[0]
    elif use_atr:
        g0 = data['atr_volatility'].iloc[0]
    else:
        g0 = data['smoothed_annualized_volatility'].iloc[0]


    if use_constant_rate:
        r0 = risk_free
    else:
        r0 = data['Rate'].iloc[0]

    T0 = option_maturity
    call_price, call_delta = black_scholes(S0, K0, T0, r0, g0, 'call')
    shares = N_call * call_delta
    bank = (-N_call * call_price) + ((shares) * S0)
    initial_fees = (N_call * call_price + abs(shares * S0)) * fees
    bank -= initial_fees
    value_portfolio = bank + (- shares * S0) + (N_call * call_price)

    return {
        "S": S0,
        "K": K0,
        "T": T0,
        "Volatility": g0,
        "Risk Free": r0,
        "Shares": -shares,
        "Share_Price": S0 * shares,
        "Option_Type": 'call',
        "Option_Price": call_price,
        "Option_Delta": call_delta,
        "Bank": bank,
        "Value_portfolio": value_portfolio,
        "fees_cumsum": initial_fees,
        "fees_transaction": initial_fees
    }


def simulate_trades(data, initial_positions, fees=1/100, day_rebalancing=1,
    day_rolling=21,
```

```python
                    use_vix_proxy=False, use_constant_rate=False,
                        use_realized_vol=False, use_ewma=False, use_atr=False,
                    risk_free=0.02, option_maturity=6/12, K_multiplier=1.1):

    portfolio = {}
    position = initial_positions.copy()
    cumulative_drift = 0
    portfolio[data.index[0]] = position
    N_call = 1


    for i in range(day_rebalancing, len(data), day_rebalancing):
        prev_position = portfolio[data.index[i - day_rebalancing]].copy()
        S = data['Close'].iloc[i]
        K = prev_position['K']
        if use_vix_proxy:
            g = data['implied_volatility'].iloc[i]
        elif use_realized_vol:
            g = data['realized_volatility'].iloc[i]
        elif use_ewma:
            g = data['ewma_volatility'].iloc[i]
        elif use_atr:
            g = data['atr_volatility'].iloc[i]
        else:
            g = data['smoothed_annualized_volatility'].iloc[i]

        if use_constant_rate:
            r = risk_free
        else:
            r = data['Rate'].iloc[i]

        T = prev_position['T'] - (day_rebalancing / 252)
        if T <= 0:
            # Option expired
            payoff = np.maximum(S-K,0)
            K = int(round(K_multiplier * S, -1))
            call_price, call_delta = black_scholes(S, K, option_maturity, r, g,
                'call')

            shares_old = prev_position['Shares']
            shares_new = N_call * call_delta
            shares_change = - shares_new - abs(shares_old)

            transaction_fee = call_price * fees
            bank = prev_position['Bank'] * np.exp(r * (day_rebalancing / 252))
            bank -= transaction_fee
            bank += payoff

            transaction_fee = abs(shares_change * S) * fees
            bank -= transaction_fee

            fees_cumsum = prev_position['fees_cumsum'] + transaction_fee
            value_portfolio = bank + (-shares_new * S) + (N_call * call_price)

            total_delta = call_delta - shares_new
            cumulative_drift += abs(total_delta)

            new_position = {
                "S": S,
                "K": K,
                "T": T,
                "Volatility": g,
                "Risk Free": r,
                "Shares": -shares_new,
                "Share_Price": S * shares_new,
                "Option_Type": 'call',
                "Option_Price": call_price,
                "Option_Delta": call_delta,
                "Bank": bank,
                "Value_portfolio": value_portfolio,
                "fees_transaction": transaction_fee,
                "fees_cumsum": fees_cumsum,
                "Cumulative_Delta_Drift": cumulative_drift,
                "day_rolling": day_rolling
            }

            portfolio[data.index[i]] = new_position

        else:
            if (i % day_rolling == 0):
                # Roll option
                call_price, call_delta = black_scholes(S, K, T, r, g, 'call')
                shares_old = prev_position['Shares']
                shares_new = N_call * call_delta
                shares_change = - shares_new - shares_old

                bank = prev_position['Bank'] * np.exp(r * (day_rebalancing / 252))
                bank += - call_price * fees
                bank += call_price

                T = option_maturity
                S = data['Close'].iloc[i]
                K = int(round(K_multiplier * S, -1))

                transaction_fee = call_price * fees
                bank -= transaction_fee
                bank -= call_price

                fees_cumsum = prev_position['fees_cumsum'] + transaction_fee
                value_portfolio = bank + (-shares_new * S) + (N_call * call_price)

                total_delta = call_delta - shares_new
                cumulative_drift += abs(total_delta)

                new_position = {
                    "S": S,
                    "K": K,
                    "T": T,
                    "Volatility": g,
                    "Risk Free": r,
                    "Shares": -shares_new,
                    "Share_Price": S * shares_new,
                    "Option_Type": 'call',
                    "Option_Price": call_price,
                    "Option_Delta": call_delta,
                    "Bank": bank,
                    "Value_portfolio": value_portfolio,
                    "fees_transaction": transaction_fee,
                    "fees_cumsum": fees_cumsum,
                    "Cumulative_Delta_Drift": cumulative_drift,
                    "day_rolling": day_rolling
                }
                portfolio[data.index[i]] = new_position

            else:
                # Dynamic adjustment of rolling frequency based on vol regime
                if day_rolling > 0:
                    window_length = 21
                    if i >= window_length:
                        if use_vix_proxy:
                            rolling_g = data['implied_volatility'].iloc[i -
                                window_length:i].mean()
                        else:
                            rolling_g =
                                data['smoothed_annualized_volatility'].iloc[i -
                                window_length:i].mean()

                        if g > 2 * rolling_g:
                            day_rolling = max(1, int(day_rolling // 1.1))
                        else:
```

```
                        day_rolling = int(min(252, day_rolling * 1.04))

                bank = prev_position['Bank'] * np.exp(r * (day_rebalancing / 252))
                call_price, call_delta = black_scholes(S, K, T, r, g, 'call')

                shares_old = prev_position['Shares']
                shares_new = N_call * call_delta
                shares_change = - shares_new - shares_old

                transaction_fee = abs(shares_change * S) * fees
                bank -= transaction_fee
                fees_cumsum = prev_position['fees_cumsum'] + transaction_fee
                value_portfolio = bank + (-shares_new * S) + (N_call * call_price)

                total_delta = call_delta - shares_new
                cumulative_drift += abs(total_delta)

                new_position = {
                    "S": S,
                    "K": K,
                    "T": T,
                    "Volatility": g,
                    "Risk Free": r,
                    "Shares": -shares_new,
                    "Share_Price": S * shares_new,
                    "Option_Type": 'call',
                    "Option_Price": call_price,
                    "Option_Delta": call_delta,
                    "Bank": bank,
                    "Value_portfolio": value_portfolio,
                    "fees_transaction": transaction_fee,
                    "fees_cumsum": fees_cumsum,
                    "Cumulative_Delta_Drift": cumulative_drift,
                    "day_rolling": day_rolling
                }

                portfolio[data.index[i]] = new_position

    results = pd.DataFrame(portfolio).T
    results['Date'] = pd.to_datetime(results.index)
    return results
```

Listing 1: Python code for Delta Hedging Strategy Functions

Simulation Execution Code

The Python code to run the delta hedging simulation is provided below. This script sets up the parameters, initializes the portfolio, and simulates the trading strategy using both GARCH-based and VIX-based volatility estimates.

```
symbol = "SPY"
risk_free = '^FVX'
vix_symbol = '^VIX'
backtest_period = 10 # Years
day_stock = 1 # Days
day_option = -1 # Days, if -1 No Rolling
option_maturity = 0.5 # 6-Month Maturity Option
fee_rate = 1 # %
k_multiplier = 1.1

data = get_data(symbol=symbol,
                risk_free=risk_free,
                volatility_proxy=vix_symbol,
                time_period=backtest_period
                )

initial_positions_delta_hedged_garch = build_initial_positions(data=data,
                        fees=fee_rate/100,
                        K_multiplier=k_multiplier,
                        option_maturity=option_maturity)

portfolio_data_delta_hedged_garch = simulate_trades(data,
                            initial_positions_delta_hedged_garch,
                            fees=fee_rate/100,
                            day_rebalancing=day_stock,
                            day_rolling=day_option,
                            K_multiplier=k_multiplier,
                            use_vix_proxy=True,
                            option_maturity=option_maturity
                            )
```

Listing 2: Python code to Run Delta Hedging Simulation