Simulation of a Delta-Hedging Strategy on Short European Call Position

Author: Sekhar Kanuri

We'll assume the underlying price follows the Geometric Brownian Motion (GBM) process, such that:

$$S_{t+1} = S_t * exp[(\mu - \frac{1}{2}\sigma^2)dt + \sigma dW(t)]$$

where

S_i is the price of the underlying at time i

μ is the historical mean return where return is computed as $ln(P_{t+1}/P_t)$

σ is historical volatility of returns

$dW(t) = z\sqrt{dt} : z \sim \mathcal{N}(0,1)$

$dW(t)$ is the infinitesimal change in the Weiner process from t to $t+1$

In order to delta-hedge, we'll follow a self-financing strategy whose value at maturity is exactly equal to the value of the derivative we're trying to replicate. The Black-Scholes model assumes we trade continuously, but that's not possible in practice. Instead, we hedge periodically, but then we would no longer be exactly replicating the option payoff. Below, we vary the hedging frequency from 1 day to 10 days to observe the effect on the total hedging P&L.

Initial value of option, $V_0(S_0, \sigma_0) = BS(S_0, r, K, \sigma_0, T)$

where

S_0 is initial price of the underlying

$BS(.)$ is the Black-Scholes function to price a European call

r is the annualized risk-free interest rate

K is the strike price

σ_0 is the annualized implied volatility of returns

T is the time to maturity in years

$BS(S_0, r, K, \sigma_0, T) = S_0 N(d_1) - Xe^{-rT} N(d_2)$

$$d_1 = \frac{ln\left(\frac{S_0}{K}\right) + \left(r + 0.5\sigma^2\right)T}{\sigma\sqrt{(T)}}$$

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

$$N(X) = P(Z \leq X) : Z \sim \mathcal{N}(0,1)$$

In order to estimate the premium received for selling the call, we could observe the market price. Here though, we'll use the $BS(.)$ function (only on day one) to estimate the premium received. We'll purchase and hold $N(d_1)$ number of the underlying units (we're short call so must go long underlying). For example with SPY ETF as in this analysis, a single short call position pertains to 100 shares of the ETF. A short at-the-money (ATM) call expiring in 90 days has the delta computed as 0.5914. The number of units of the ETF symbol to purchase would be $0.5914 * 100 = 59.14$. The premium received on the short call position is computed to be $\$1,609.50$. The cash account value is then computed as premium received less money required to purchase $N(d_1)$ units of the underlying. From day two onwards, the self-financing replicating portfolio prices the option.

Subsequently, $V_{i+1} = V_i + \delta_i(S_{i+1} - S_i) + (V_i - \delta_i S_i)e^{r\Delta t}$

where

V_i is the value of the replicating portfolio at time i

$\delta_i = N(d_1) = P(Z \leq d_1) : Z \sim \mathcal{N}(0,1)$

δ_i is the number of units of the underlying held at time i

$\delta_i(S_{i+1} - S_i)$ is the change in portfolio value from adjusting the number of units of the underlying

$(V_i - \delta_i S_i)e^{r\Delta t}$ is contribution to change in portfolio value from cash account

The cash account is depleted (grows) upon purchase (sale) of more units of the underlying. It also earns an overnightly risk-free interest rate. If the short call option position were perfectly and constantly hedged, changes in the option value should be exactly offset by changes in the value of the replicating portfolio. The non-zero total hedging P&L arises from the fact that the delta-hedging we apply is neither continuous nor perfect.

Total Hedging P&L is then computed as:

$$\pi = (V_T - V_0) - max(0, S_T - K)$$

where

$(V_T - V_0)$ is the change in value of the replicating portfolio between T and $t = 0$

$-max(0, S_T - K)$ is the payoff to the option holder (we're short the option)

In [ ]:

```
import warnings
import numpy as np
import scipy
import pandas as pd
import seaborn as sns
import datetime
import matplotlib.pyplot as plt
import yfinance as yf
from collections import defaultdict

warnings.filterwarnings("ignore")

class DeltaHedge():

    # Constructor method
    def __init__(self, symbol='^SPX', maturity=1, risk_free=0.05, \
                 num_periods=100, num_options=1, look_back_days=30, \
                 vol_factor=1, hedging_freq=1):
        self._symbol = symbol
        self._maturity = maturity
        self._risk_free = risk_free
        self._num_periods = num_periods
        self._num_options = num_options
        self._price_path = np.array([])
        self._self_financiang_portfolio_final_value = 0.0
        self._payoff_at_maturity = 0.0
        self._total_hedging_pnl = 0.0
        self._returns = np.array([])
        self._initial_price = 0
        self._look_back_days = look_back_days
        self._vol_factor = vol_factor
        self._hedging_freq = hedging_freq

    # Method to fetch historical price data from Yahoo! Finance
    def fetch_market_data(self):
        df = yf.Ticker(self._symbol).history(period="6mo").reset_index()
        df['Date'] = df['Date'].dt.date
        df = df[['Date','Close']]
        self._returns = np.log(df['Close'].values[-self._look_back_days+1:]) - \
                        np.log(df['Close'].values[-self._look_back_days:-1])
        self._mu, self._sigma = np.mean(self._returns), np.std(self._returns)
        self._sigma = self._sigma * self._vol_factor
        self._price_path = [df['Close'].values[-1]]
        self._initial_price = self._price_path[0]
        self._strike = self._initial_price

    # Method to generate price paths using Geometric Brownian Motion (GBM) process
    def generate_price_path(self):
        for i in range(self._num_periods):
            self._price_path = np.append(self._price_path, \
                    self._price_path[-1] * np.exp((self._mu - 0.5*self._sigma**2) \
                       * (1/self._num_periods) + self._sigma * \
                              np.sqrt(1/self._num_periods) * np.random.randn(1)))
            self._sigma_annualized = self._sigma * np.sqrt(252)
        self._price_path = self._price_path[::self._hedging_freq]
        return self._price_path

    """
    Method to compute B-S price of option in lieu of an observed market price.
```

```
    The B-S computed price is sanity checked to match (close to) the market price.
    'The wrong number in the wrong formula to get the right price!'
    """
    def black_scholes(self, type='call'):
        if type == 'call':
            d1 = ( np.log(self._initial_price/self._strike) + \
            (self._risk_free + 0.5 * self._sigma_annualized**2)*self._maturity ) \
                                / (self._sigma_annualized * np.sqrt(self._maturity))
            d2 = d1 - self._sigma_annualized * np.sqrt(self._maturity)
            Nd1 = scipy.stats.norm.cdf(d1)
            Nd2 = scipy.stats.norm.cdf(d2)
            bs = self._initial_price * Nd1 - self._strike * \
                 np.exp(-1*self._risk_free*self._maturity) * Nd2
        return d1, d2, Nd1, Nd2, bs

    # Method to carry out the delta-hedging mechanics
    def execute_hedge(self):
        d1, d2, Nd1, Nd2, bs = self.black_scholes()
        portfolio = defaultdict(float)
        portfolio['Period'] = [0]
        portfolio['UL Price'] = [self._initial_price]
        portfolio['d1'] = [round(d1, 4)]
        portfolio['N(d1)'] = [round(Nd1, 4)]
        portfolio['UL Units'] = [round(Nd1 * self._num_options, 2)]
        portfolio['Cash'] = [round(-1 * Nd1 * self._initial_price * \
                                   self._num_options + bs * self._num_options, 2)]
        portfolio['Portfolio Value'] = [round(bs * self._num_options, 2)]

        # Loop through each period (anything between 1 to 9 days) passed to function
        for n in range(1, len(self._price_path)):
            T = self._maturity * (len(self._price_path) - n) / len(self._price_path)
            S = self._price_path[n]
            d1 = ( np.log(S/self._strike) + (self._risk_free + \
                            0.5*(self._sigma_annualized**2))*T ) / \
                            (self._sigma_annualized * np.sqrt(T))
            d2 = d1 - self._sigma_annualized * np.sqrt(T)
            Nd1 = scipy.stats.norm.cdf(d1)
            ul_units = round(Nd1 * self._num_options, 2)
            diff = S * (ul_units - portfolio['UL Units'][-1])
            cash = round(portfolio['Cash'][-1] - diff, 2)
            sf_portfolio_value = round(cash + (ul_units * S), 2)
            portfolio['Period'].append(n)
            portfolio['UL Price'].append(S)
            portfolio['d1'].append(d1)
            portfolio['N(d1)'].append(Nd1)
            portfolio['UL Units'].append(ul_units)
            portfolio['Cash'].append(cash)
            portfolio['Portfolio Value'].append(sf_portfolio_value)

        # Compute value of self-financing replacing portfolio as the option value
        self._self_financing_portfolio_final_value = portfolio['UL Units'][-1] * \
                        self._price_path[-1] + portfolio['Cash'][-1]
        self._payoff = max(0, self._price_path[-1] - self._strike) * \
                                                        self._num_options
        self._total_hedging_pnl = self._self_financing_portfolio_final_value - \
                            portfolio['Portfolio Value'][-1] - self._payoff

        df = pd.DataFrame(portfolio)
        return df, self._total_hedging_pnl
```

```
# Initialize results dict to collect results from vol multiple vs. frequency runs
results_dict = {}
results_dict['Volatility Factor'] = []
results_dict['Hedging Frequency'] = []
results_dict['Average Hedging P&L'] = []
vol_factors = list(np.arange(0.1, 2.1, 0.1))
hedge_freqs = list(range(1, 10))

# Loop through volatility multiples and hedging frequencies
for v in [1]:
    for f in [1]:
        print(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S'), \
                                    ": Vol Factor=", v, "Hedge Freq=", f)

        paths_dict = defaultdict(float)
        pnl_list = []
        sf_prtf_dict = defaultdict(float)
        nd1_dict = defaultdict(float)

        # Repeat for 250 separate price paths
        for sim in range(250):
            deltaHedge = DeltaHedge(symbol='SPY', maturity=0.25, risk_free=0.05, \
                                    num_periods=90, num_options=100, look_back_days=30, \
                                    vol_factor=v, hedging_freq=f)
            deltaHedge.fetch_market_data()
            path = deltaHedge.generate_price_path()
            df, pnl = deltaHedge.execute_hedge()
            pnl_list.append(int(pnl))
            paths_dict[sim] = path
            nd1_dict[sim] = df['N(d1)']
            sf_prtf_dict[sim] = df['Portfolio Value']

        results_dict['Volatility Factor'].append(v)
        results_dict['Hedging Frequency'].append(f)
        results_dict['Average Hedging P&L'].append(int(np.min(pnl_list)))

        df_res = pd.DataFrame(results_dict)
        df_res.to_csv('delta_hedging_20240311.csv')


# Graph results showing hedging P&L for various vol mutiples and hedge frequencies
df1 = df_res.pivot(index='Volatility Factor', columns='Hedging Frequency', \
                   values='Average Hedging P&L')
X, Y = np.meshgrid(df1.columns, df1.index)
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(Y, X.astype(float), df1.values, cmap='coolwarm')
ax.view_init(elev=15, azim=65)
ax.set_title('Delta Hedging P&L vs. Returns Volatility vs. Hedging Frequency\n \
Portfolio of Short Calls on SPY, Long Underlying & Cash')
ax.set_zlabel('Average Hedging P&L ($)')
ax.set_xlabel('Volatility Multiple')
ax.set_ylabel('Hedging Frequency (Days)')
plt.show()
```