King's College London
Department of Mathematics
Submission Cover Sheet for Coursework

The following cover sheet must be completed and submitted with any dissertation, project or essay submitted as a part of formal assessment for degree within the Mathematics Department.

You are not required to write your name on your work

Candidate Number:	AE06879
Module Code:	FM06
Title of Project:	FM06 - Numerical and Computational Methods in Finance - Coursework

Declaration

By submitting this assignment I agree to the following statements:

I have read and understand the King's College London Academic Honesty and Integrity Statement that I signed upon entry to this programme of study.

I declare that the content of this submission is my own work.

I understand that plagiarism is a serious examination offence, an allegation of which can result in action being taken under the College's Misconduct regulations.

Your work may be used as an example of good practice for future students to refer to. If chosen, your work will be made available either via KEATS or by paper copy. Your work will remain anonymous; neither the specific mark nor any individual feedback will be shared. Participation is entirely optional and will not affect your mark. If you consent to your submission being used in this way please tick the box.	Yes

Office Use Only: Date/Time Stamp	Agreed Mark

k-Number	k2257667
Price P	3.1573
Confidence interval for p^{Δ}	$(2.3824, 4.3781)$
Confidence interval for p^{Γ}	$(-0.0017121, 0.0053743)$

1 Introduction

In this essay, I will investigate the hedging of a European call option payoff in a modified Black-Scholes environment. The stock being modelled has a time-varying volatility, rather than the constant volatility that classical Black-Scholes would assume.

I will analyse the ability of delta and gamma hedging strategies to replicate the call payoff in this environment. I will first show that delta hedging is successful only if the correct volatility function has been used for the hedging. I will then show gamma hedging has two advantages over delta hedging. First, it is more accurate when the volatility function is constant. But more importantly, it can replicate call payoffs even when the wrong volatility function has been used for hedging.

Computations in this essay are based on techniques taught in the FM06 course, [Arm23]. I will in particular make use of the sections on simulating a stock price and delta hedging. All results are given to 5 significant figures.

2 Specifying The Model

Throughout this essay, I will be simulating a stock governed by the stochastic differential equation (SDE):

$$dS_t = \mu S_t dt + \sigma(t) S_t dW_t \tag{1}$$

where $S_0 = 149, \mu = 0.13$ and W_t is a Brownian motion. $\sigma(t)$ is given by:

$$\sigma(t) = \begin{cases} 0.11 + 0.13 \times \frac{t-0.30}{T-0.30} & \text{if } t > 0.30 \\ 0.11 & \text{otherwise} \end{cases}$$

with $T = 1.70$.

I will be using delta and gamma hedging to replicate the payoff of a European call option on the stock. The target option has time to maturity T and strike $K = 188.00$, and the continuously compounded risk-free rate is $r = 0.01$. At time $0 \leq t < T$, the price of the call is given as:

$$c_t = BS^{call}(S_t, K, T-t, s_t, r) \tag{2}$$

where

$$s_t^2 = \frac{1}{T-t}\int_t^T \sigma(u)^2\, du \tag{3}$$

and $BS^{call}(S_t, K, T-t, s_t, r)$ is the standard Black-Scholes formula. From the formula above, $c_0 = 3.1573$. I will verify this numerically when implementing delta hedging.

Additionally, I will need to calculate some of the option's Greeks when implementing hedging strategies below. Applying the results of [Hul17] chapter 19 shows that this option has delta and gamma given by:

$$\begin{aligned}
\Delta_{\text{call}}(t, S_t) &= \frac{\partial c_t}{\partial S_t} = \Phi(d_1) \\
\Gamma_{\text{call}}(t, S_t) &= \frac{\partial^2 c_t}{\partial S_t^2} = \frac{\Phi'(d_1)}{S_t s_t \sqrt{T-t}} \\
d_1 &= \frac{\log\left(\frac{S_t}{K}\right) + (r + \frac{s_t^2}{2})(T-t)}{s_t\sqrt{T-t}}
\end{aligned} \tag{4}$$

where $\Phi(x)$ is the cumulative distribution function of the standard Normal distribution at x.

3 Simulating The Stock Price over Time

In order to implement delta or gamma hedging, I need to simulate the stock up to time T. To do this, I used an Euler-Maruyama scheme on the log of the stock, as described in [Arm23]. By Itō's lemma, if S follows (1) then $X_t = \log S_t$ follows the SDE:

$$dX_t = \left(\mu - \frac{\sigma(t)^2}{2}\right) dt + \sigma(t) dW_t. \tag{5}$$

To simulate this SDE, I divided the interval $[0, T]$ into N sub-intervals of equal length $\delta t = T/N$. This gives a set of points $\{0 = t_0, t_1, \ldots, t_N = T\}$, with $t_i = i \times \delta t$. My choice of N assumes $T = 1.7$ is given in years and sets N to represent the number of days in this period rounded down, $\lfloor 1.7 * 365 \rfloor = 620$. This is the default value of N, though I will modify it for certain analyses.

The SDE can then be simulated on the set of points as:

$$X_{t_{i+1}} = X_{t_i} + \left(\mu - \frac{\sigma(t_i)^2}{2}\right) dt + \sigma(t_i)\delta W_{t_i} \tag{6}$$

where $\delta W_{t_i} = W_{t_{i+1}} - W_{t_i} \sim N(0, t_{i+1} - t_i) = N(0, \delta t)\ \forall i$. This equation applies for $i \in \{0, 1, 2, ...N-1\}$. The full simulation algorithm I used is therefore:

1. Choose a number of simulations, M.
2. Create an empty $M \times (N+1)$ array $\boldsymbol{X}$. The jth row represents the time evolution of the log-stock price in simulation j. The ith column, $\boldsymbol{X^{(i)}}$, represents the log-stock price over all simulations at the time $(i-1)\delta t$.
3. Set $\boldsymbol{X^{(1)}} = \log S_0$. This is the starting value of the log-stock process, the same in all simulations.
4. Generate a $M \times N$ array $\boldsymbol{Z}$ of independent and identically distributed random values from a Normal distribution with mean 0 and variance δt. As above, $\boldsymbol{Z^{(i)}}$ is the ith column of $\boldsymbol{Z}$.
5. Iteratively apply equation (6): $\boldsymbol{X^{(i+1)}} = \boldsymbol{X^{(i)}} + (\mu - \frac{\sigma(t_{i-1})^2}{2})\delta t + \sigma(t_{i-1})\boldsymbol{Z^{(i)}})$ for $i = 0, 1, ...N$.
6. When $\boldsymbol{X}$ is populated with non-zero values, generate an array of stock prices as $\boldsymbol{S} = \mathrm{e}^{\boldsymbol{X}}$. Exponentiation is applied component-wise.

I have used this approach for all simulations of stock prices in this coursework. The final output is $\boldsymbol{S}$, an array containing values of the stock at all time points in all simulations. The array $\boldsymbol{Z}$ was generated using the built-in `numpy` function `random.normal`. Details of the `numpy` package can be found in [HMvdW$^+$20].

I used $\boldsymbol{S}$ to calculate the price of the call, its delta, and its gamma in all simulations and at all points in the simulation. This is done by applying equations (2) and (4). I will denote these quantities by the $M \times (N+1)$ arrays $\boldsymbol{C}$, $\boldsymbol{\Delta}_{\text{call}}$ and $\boldsymbol{\Gamma}_{\text{call}}$ respectively.

4 Delta Hedging A Call Option

4.1 Delta Hedging with The Right Volatility

Having generated $\boldsymbol{S}$ and $\boldsymbol{\Delta}_{\text{call}}$, I can now implement delta hedging. I created another array $\boldsymbol{W}$, which represents the wealth of the trader at all times in each simulation. Then, as explained in [Arm23], delta hedging is implemented by starting with an initial wealth of c_0 in a risk-free cash account i.e. $\boldsymbol{W}^{(1)} = c_0$. The trader then holds $\Delta_t(S_t, t)$ of the stock, and $w_t - \Delta_t$ in a risk-free cash account. In theory this should be continuously rebalanced, but in practice I approximate this by rebalancing only on the series of points $0 = t_0, t_1..., t_N = T$. Note that $\Delta(t, S_t)$ varies in each simulation, and so the trader's wealth across all simulations then evolves as:

$$\boldsymbol{W}^{(i+1)} = \mathrm{e}^{r\delta t}(\boldsymbol{W}^{(i)} - \boldsymbol{\Delta}_{\text{call}}^{(i)}\boldsymbol{S}^{(i)}) + \boldsymbol{\Delta}_{\text{call}}^{(i)}(\boldsymbol{S}^{(i+1)} - \boldsymbol{S}^{(i)}) \tag{7}$$

where the first term represents the interest from the risk-free account, and the second the change in the stock price. Note that multiplications of column vectors are applied component-wise rather than as a dot product.

I applied this formula for $i = 0, 1, ...N$ recursively until I had a final wealth value $\boldsymbol{W}^{(N+1)}$ for all simulations. The hedging error is defined as the difference between the final wealth of the trader and the payoff of the option at time T i.e. $\boldsymbol{W}^{(N+1)} - \max(\boldsymbol{S}^{(N+1)} - K, 0)$. Figure 1 shows a plot of hedging errors in two cases: $M = 1000$, $N = 620$ ($\delta t = \frac{T}{N} = \frac{1.7}{620}$) and $M = 1000$, $N = 62,000$ ($\delta t = \frac{1.7}{62000}$). In both cases, the histograms are centered about 0 and the error is much smaller when the number of steps is larger.

This suggests that the error is converging to 0. To confirm the rate of convergence, I re-ran the simulation using many different values of N. I then made a log-log plot in figure 2 of the root mean square hedging error across all simulations against N. On a log-log plot, the error is a straight line with gradient $-\frac{1}{2}$ (shown by comparison to a line with this gradient). This implies $O(N^{-\frac{1}{2}})$ convergence, consistent with the results of [Arm23] Chapter 7. Overall, this suggests that the delta hedging algorithm works, and that the time-t price of the option is indeed given by (2).

4.2 Delta Hedging with The Wrong Volatility

Thus far I have assumed the trader knows the form of the volatility function $\sigma(t)$. However, it is plausible that the trader does not. This is discussed

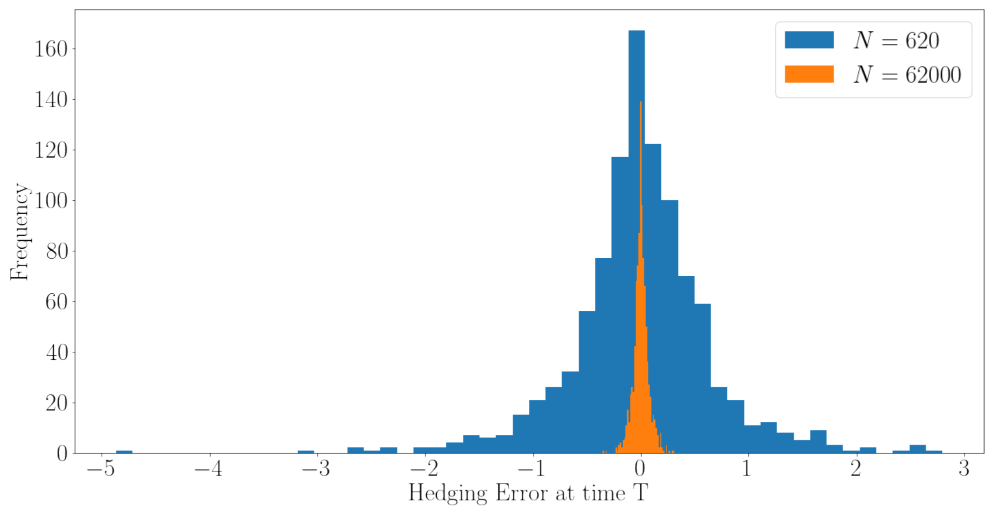


Figure 1: Histograms of the delta hedging error for 1000 simulations and two different timestep amounts, $N = 620$ and $N = 62000$

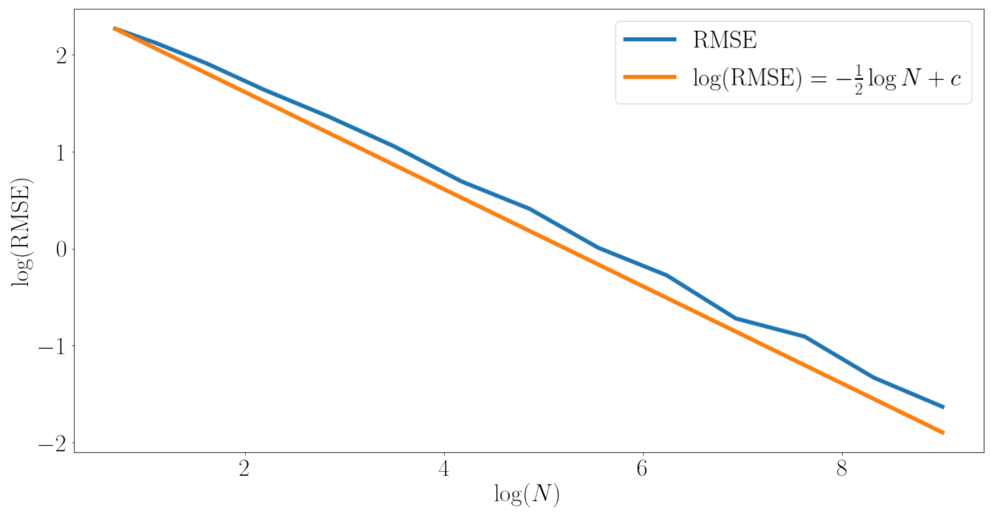


Figure 2: delta hedging Root Mean Square Error (RMSE) as the number of timesteps N varies. This is a log-log plot and the result is compared to a reference line with gradient $-\frac{1}{2}$

in Question 2 of the assignment, where the trader delta hedges on the basis that the stock follows the SDE (1) but with a constant volatility function is $\tilde{\sigma}(t) = \sigma(T)$:

$$dS_t = \mu S_t dt + \tilde{\sigma}(t) S_t dW_t \tag{8}$$

To determine the outcome of this situation, I simulated the stock according to the original SDE. However, I then recalculated the time-0 price of the call and the array of deltas assuming that the volatility function is $\tilde{\sigma}(t)$. I did this by noting that $\tilde{\sigma}(t)$ is constant and so the integral in (3) gives us $s_t = \sigma(T)\ \forall t$. Then updated values, $\tilde{c}_0$ and $\tilde{\mathbf{\Delta}}_{\text{call}}$, can be calculated as a special case of (2) and (4).

I then reapplied delta hedging iteratively as before. I simulated the stock up to time $\frac{T}{2}$. The hedging error is then the difference between the trader's final wealth and the payoff they were expecting - which is $BS^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}.\sigma(T), r)$.

The hedging error results at time $\frac{T}{2}$ are in figure 3. The chart shows that the replication was not successful, as the hedging errors are clearly non-zero and all positive. This is expected, because $\tilde{\sigma}(t) = \sigma(T) \geq \sigma(t)\ \forall t \leq T$. In this situation, as discussed in [KJPS98], the hedging error over time will be non-decreasing, so I expect all hedging errors to be positive. This finding is reflected in the chart.

5 Gamma Hedging A Call Option

Gamma hedging is an extension of delta hedging. In this approach, we assume there are two purchasable options - the call we are trying to replicate, and a put option with strike $K^H = S_0$ and maturity T. Analogously to the call option, its price, delta and gamma are given by:

$$\begin{gathered}
p_t = (S_t, K^H, T-t, s_t, r) = K^H \mathrm{e}^{-r(T-t)}\Phi(-d_2) - S_t\Phi(-d_1) \\
\Delta_{\text{put}}(t, S_t) = \frac{\partial p_t}{\partial S_t} = -\Phi(-d_1) \\
\Gamma_{\text{put}}(t, S_t) = \frac{\partial^2 p_t}{\partial S_t^2} = \frac{\Phi'(d_1)}{S_t s_t \sqrt{T-t}}
\end{gathered} \tag{9}$$

where d_1 and d_2 reflect the new strike price. Similarly to the setup described in section 3, the simulation of the stock price allows me to construct arrays of this information across all simulations and at all times: $\boldsymbol{P}$, $\mathbf{\Delta}_{\text{put}}$ and $\mathbf{\Gamma}_{\text{put}}$.

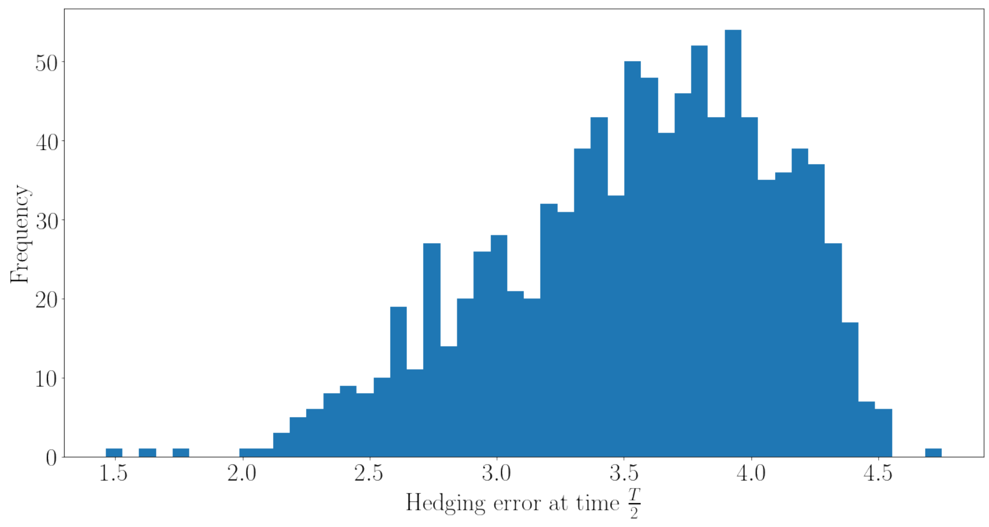


Figure 3: Histogram of Hedging Error at time $\frac{T}{2}$ when a trader is using the wrong replication strategy for $M = 1000$ simulations and $N = 1000$ timesteps. In this scenario, the trader is assuming the stock follows the SDE (1) except with a new volatility function $\gamma(t) = \sigma(T)\ \forall t$

Given this information, I implemented gamma hedging as follows: start with an initial wealth of w_0 in a risk-free cash account, where w_0 is equal to the time-0 price of the option being hedged. Over time, we hold q_S of the stock and q_H of the put option. We choose these two parameters such that the delta and gamma of the portfolio are equal to those of the call being replicated. The remainder, $w_t - q_S S_t - q_H p_t$, is held in a risk-free cash account.

At any time t, we can determine the values of q_S, q_H: the delta of the portfolio is $q_S + q_H \Delta_{\text{put}}(t, S_t)$ and its gamma is $q_H \Gamma_{\text{put}}(t, S_t)$. Setting these equal to $\Delta_{\text{call}}(t, S_t)$ and $\Gamma_{\text{call}}(t, S_t)$ respectively gives a set of simultaneous equations that is solved by:

$$\begin{aligned} q_H(t, S_t) &= \frac{\Gamma_{\text{call}}(t, S_t)}{\Gamma_{\text{put}}(t, S_t)} \\ q_S(t, S_t) &= \Delta_{\text{call}}(t, S_t) - q_H(t)\Delta_{\text{put}}(t, S_t) \end{aligned} \tag{10}$$

in the same style as above, we can construct arrays of these values for all simulations and times: $\boldsymbol{Q_H}, \boldsymbol{Q_S}$.

I applied gamma hedging in the same way as described in section 4, iterating a trader's wealth over time in all simulations. The trader's wealth now evolves as:

$$\begin{aligned}\boldsymbol{W}^{(i+1)} &= \mathrm{e}^{r\delta t}(\boldsymbol{W}^{(i)} - \boldsymbol{Q_S}^{(i)}\boldsymbol{S}^{(i)} - \boldsymbol{Q_H}^{(i)}\boldsymbol{P}^{(i)}) \\ &+ \boldsymbol{Q_S}^{(i)}(\boldsymbol{S}^{(i+1)} - \boldsymbol{S}^{(i)}) \\ &+ \boldsymbol{Q_H}^{(i)}(\boldsymbol{P}^{(i+1)} - \boldsymbol{P}^{(i)})\end{aligned} \tag{11}$$

where the first term represents interest earned on the cash account, the second the change in stock price, and the third the change in the put option price.

To demonstrate the efficacy of gamma hedging, I first consider it applied in a simple example. Here, we assume that the stock follows a modified version of 1:

$$dS_t = \mu S_t dt + \sigma(T) S_t dW_t \tag{12}$$

I simulated the stock price according to this model and then applied delta and gamma hedging up until time $\frac{T}{2}$. I then calculated the hedging error as the difference between the wealth of the investor at time $\frac{T}{2}$ and the payoff $BS^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$. Figure 4 plots a histogram of the errors, and figure 5 the convergence of hedging errors.

The two figures show that gamma hedging is much more effective than delta hedging - in figure 4 its errors are far more tightly grouped around 0 than those of delta hedging. Moreover, the convergence in figure **??**55s that the error of gamma hedging is not only always smaller than that of delta hedging, but also converges to 0 faster - $O(N^{-1})$ compared to $O(N^{-\frac{1}{2}})$ for delta hedging.

Overall then, gamma hedging is far more effective at replicating a call payoff than delta hedging is. Indeed it is not necessary to rehedge so often to replicate the price of a call option to a reasonable level of accuracy.

5.1 Gamma Hedging with The Wrong Volatility

I now show that gamma hedging has another feature that delta hedging does not. As discussed in section 4.2, if the trader incorrectly assumes a constant volatility function, delta hedging does not replicate the expected payoff. Gamma hedging, however, is able to do this.
To show this, I followed the same approach described in section 4.2: I generated simulations of the stock using the SDE in (1). However, I applied gamma hedging iteratively assuming that the volatility function was $\tilde{\sigma}(t) = \sigma(T)$.

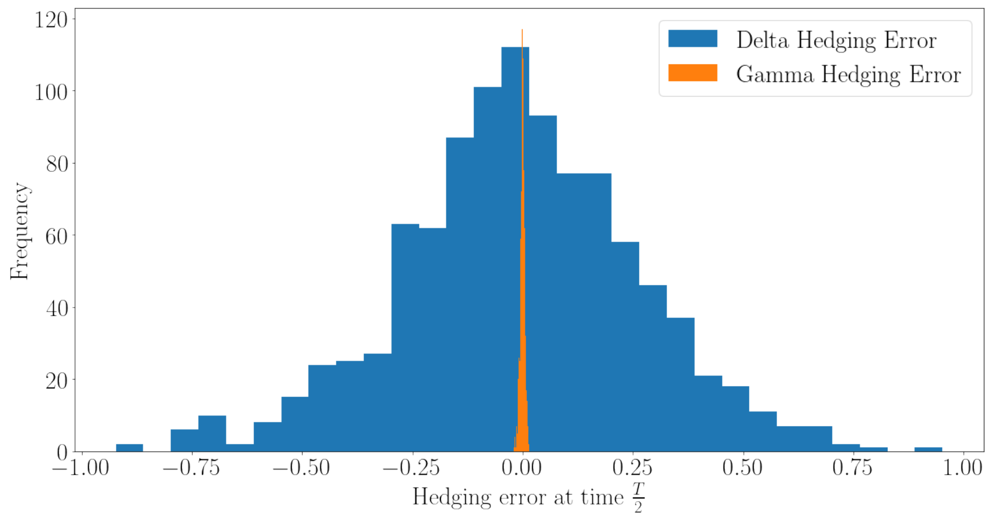


Figure 4: Hedging errors of delta and gamma Hedging strategies in a model with constant volatility. Results are shown as at time $\frac{T}{2}$ for 1000 simulations and $N = 620$ timesteps

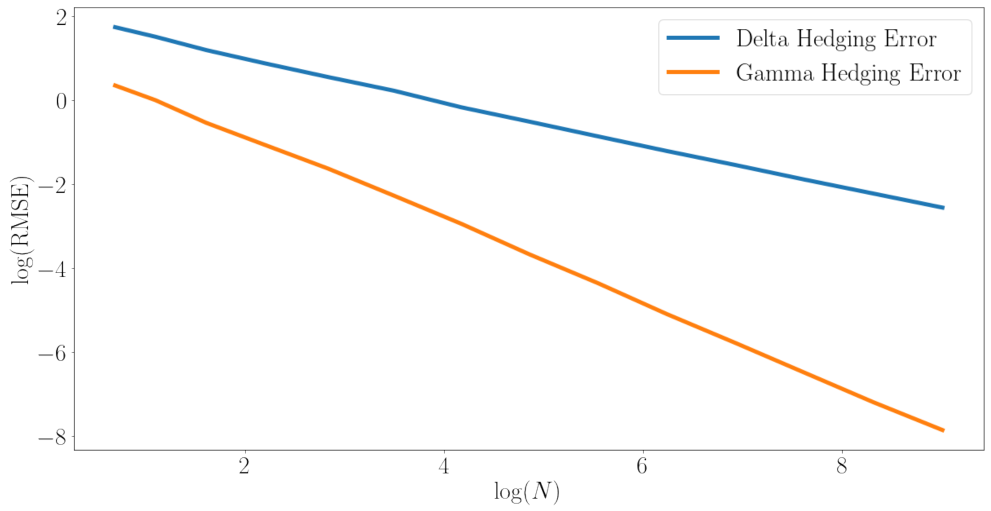


Figure 5: RMSE of delta and gamma hedging strategies as the number of timesteps N varies. This is shown on a log-log plot

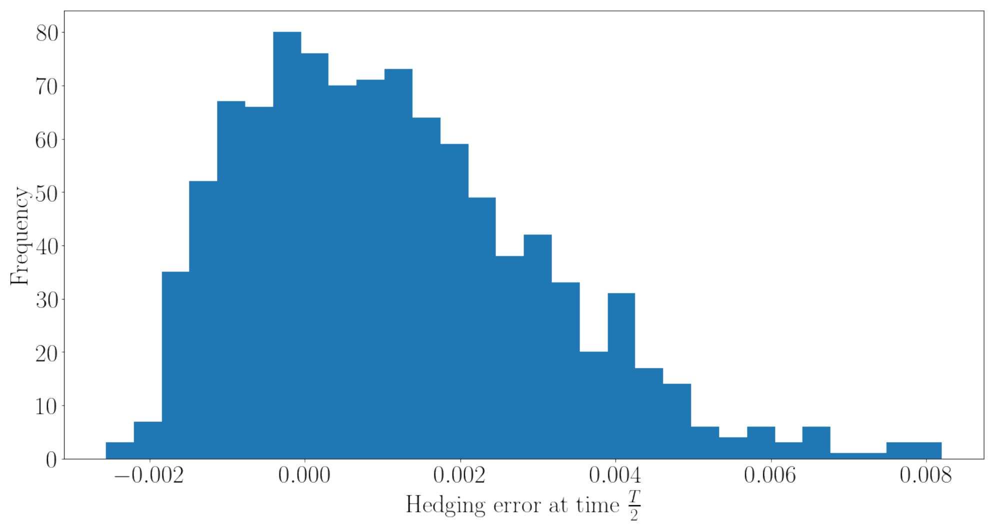


Figure 6: Histogram of Hedging Error at time $\frac{T}{2}$ when a trader is gamma hedging but has the wrong volatility assumption in their replication strategy. Here, the trader is gamma hedging a call option assuming that the underlying stock follows a Black Scholes model with constant volatility $\sigma(T)$. The results are for $M = 1000$ simulations and $N = 1000$ timesteps.

The hedging error is the difference between the final wealth of the trader and their target payoff, $BS^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$.
The hedging error results at time $\frac{T}{2}$ are in figure **??**. The chart shows that, unlike with delta hedging, the replication is successful, as the errors are closely centered around 0.

References

[Arm23] John Armstrong. Lecture notes for 7CCMFM06, Computational and Numerical Methods in Mathematical Finance. `https://keats.kcl.ac.uk/course/view.php?id=114334`, 2023.

[HMvdW$^+$20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Hul17] John C Hull. *Options, futures, and other derivatives.* Pearson Education, 9th edition. edition, 2017.

[KJPS98] Nicole El Karoui, Monique Jeanblanc-Picquè, and Steven E. Shreve. Robustness of the black and scholes formula. *Mathematical finance*, 8(2):93–126, 1998.

A Jupyter notebook

notebook

February 19, 2024

[ ]:
```
#Setup
import numpy as np
from numpy.testing import assert_almost_equal
import scipy.integrate as integrate
from scipy.stats import norm
import matplotlib.pyplot as plt
from matplotlib import rc
rc('text',usetex=True)
rc('lines', linewidth = 5)
plt.rcParams['figure.figsize'] = (20, 10)

font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 30}
rc('font', **font)
```

0.1 Part 0 - Setup

[ ]:
```
### Constants
sigma1 = 0.11
sigma2 = 0.13
T = 1.7
mu = 0.13
r = 0.01
S0 = 149
time_break = 0.3
K = 188
KH = S0
n_sims = 1_000
n_steps = int(T * 365) #Assume time in years, this is daily time-steps
```

[ ]:
```
### Helper Functions

### Black-Scholes

def get_d1_and_d2(S, t, K, T, r, sigma):
    tau = T - t
```

```python
    d1 = 1/(sigma * np.sqrt(tau)) * (np.log(S/K) + (r + sigma ** 2 / 2) * tau)
    d2 = d1 - sigma * np.sqrt(tau)
    return d1, d2

def black_scholes_call_price(S, t, K, T, r,sigma):
    d1, d2 = get_d1_and_d2(S, t, K, T, r, sigma)
    return S * norm.cdf(d1) - K * np.exp(-r * (T- t)) * norm.cdf(d2)

def black_scholes_put_price(S,t, K,T,r,sigma):
    d1, d2 = get_d1_and_d2(S, t, K, T, r, sigma)
    return  K * np.exp(-r * (T - t)) * norm.cdf( -d2) - S * norm.cdf(-d1)

def black_scholes_call_delta(S, t, K, T, r,sigma):
    d1, d2 = get_d1_and_d2(S, t, K, T, r, sigma)
    return norm.cdf(d1)

def black_scholes_put_delta(S, t, K, T, r,sigma):
    d1, d2 = get_d1_and_d2(S, t, K, T, r, sigma)
    return -1 * norm.cdf(-1 * d1)

def black_scholes_gamma(S, t, K, T, r,sigma):
    #same for a call and put
    d1, d2 = get_d1_and_d2(S, t, K, T, r, sigma)
    return norm.pdf(d1)/(S * sigma * np.sqrt(T - t))

### Test helper functions. Numerical values evaluated on a calculator

assert_almost_equal(black_scholes_call_price(100,0,  100, 1, 0.01, 0.2), 8.
  ↪43332, decimal = 5)
assert_almost_equal(black_scholes_call_price(100,0.99,  95, 1, 0.01, 0.2), 5.
  ↪01264, decimal = 5)
assert_almost_equal(black_scholes_put_price(100,0, 100, 1, 0.01, 0.2), 7.43831,␣
  ↪decimal = 5)
assert_almost_equal(black_scholes_put_price(100,0.99,  95, 1, 0.01, 0.2), 0.
  ↪00314, decimal = 5)

assert_almost_equal(black_scholes_call_delta(100,0, 100, 1, 0.01, 0.2), 0.
  ↪55962, decimal = 5)
assert_almost_equal(black_scholes_call_delta(100,0.99,  95, 1, 0.01, 0.2), 0.
  ↪99506, decimal = 5)
assert_almost_equal(black_scholes_put_delta(100,0, 100, 1, 0.01, 0.2), -0.
  ↪44038, decimal = 5)
assert_almost_equal(black_scholes_put_delta(100,0.99, 95, 1, 0.01, 0.2), -0.
  ↪00494, decimal = 5)
```

```
assert_almost_equal(black_scholes_gamma(100,0, 100, 1, 0.01, 0.2), 0.01972,␣
↪decimal = 5)
assert_almost_equal(black_scholes_gamma(100,0.99, 95, 1, 0.01, 0.2), 0.00716,␣
↪decimal = 5)
```

[ ]:
```
class time_varying_vol:
    def __init__(self):
        pass

    def time_varying_vol(self, t):
        if t > time_break:
            return sigma1 + sigma2 * (t - time_break)/(T - time_break) #Note␣
↪reference to global constant T, this is intentional
        return sigma1

    def integrate_vol(self, start_time, end_time):
        squared_vol = lambda x: self.time_varying_vol(x) ** 2
        result = integrate.quad(squared_vol, start_time, end_time)
        return result[0]

###Testing time varying vol. Numerical values evaluated manually

Test_vol = time_varying_vol()
assert_almost_equal(np.array([Test_vol.time_varying_vol(0.2), Test_vol.
↪time_varying_vol(0.4), Test_vol.time_varying_vol(0.5), Test_vol.
↪time_varying_vol(T)]) ,
                    np.array([sigma1, 0.11928571428, 0.128571429, sigma1 +␣
↪sigma2]))

assert_almost_equal(np.array([Test_vol.integrate_vol(start_time = 0, end_time =␣
↪time_break), Test_vol.integrate_vol(0, 1), Test_vol.integrate_vol(start_time␣
↪= 0, end_time = T)]),
                    np.array([time_break * sigma1 ** 2, 0.018090833333, 0.
↪0484766666]))
```

[ ]:
```
def simulate_gbm(S0, mu, vol_model, n_steps, n_sims, simulation_end_time):
    rng = np.random.default_rng(seed = 42) #Seed for result consistency
    dt = simulation_end_time / n_steps
    times = np.linspace(0, simulation_end_time, n_steps + 1)
    epsilon = rng.normal(size = [n_sims, n_steps])

    paths = np.zeros([n_sims, n_steps + 1])
    paths[:, 0] = np.log(S0)

    for i in range(0, n_steps):
        #Euler-Maruyama Scheme, as in notes
        drift = (mu - 0.5 * vol_model.time_varying_vol(times[i]) ** 2) * dt
        stochastic = vol_model.time_varying_vol(times[i]) * np.sqrt(dt) *␣
↪epsilon[:, i]
        paths[:, i + 1] = paths[:, i] + drift + stochastic

    paths = np.exp(paths)
    return times, paths
```

[ ]:
```
class Hedger:
    def __init__(self, S0, K, T, r, mu, n_sims, n_steps, vol_model,␣
↪simulation_end_time = None, constant_vol_assumption = False,␣
↪do_gamma_hedging = False):

        #Inputs
        self.S0 = S0
        self.K = K
        self.T = T
        self.r = r
        self.mu = mu
        self.vol_model = vol_model
        self.constant_vol_assumption = constant_vol_assumption
        self.n_sims = n_sims
        self.n_steps = n_steps
        self.do_gamma_hedging = do_gamma_hedging

        if self.do_gamma_hedging:
            self.constant_vol_assumption = True #override this - in this␣
↪coursework if gamma hedging is happening i.e. Q3/Q4, we are assuming␣
↪constant vol

        if simulation_end_time is not None:
            self.simulation_end_time = simulation_end_time
        else:
            self.simulation_end_time = T #Assume full period unless specified

        #Derived
        self.dt = self.simulation_end_time / self.n_steps
        full_period_vol = self.get_integrated_vol(0, self.T)
        self.initial_price = black_scholes_call_price(self.S0, 0, self.K, self.
↪T, self.r, full_period_vol)

        #Always need paths and payoffs, so do this on initialisation
        self.simulation_setup()
        self.get_final_call_prices()

    def get_integrated_vol(self, start_time, end_time):
        #calculates integrated vol over a remaining time period. Designed to␣
↪return vol depending on the assumption of the investor
```

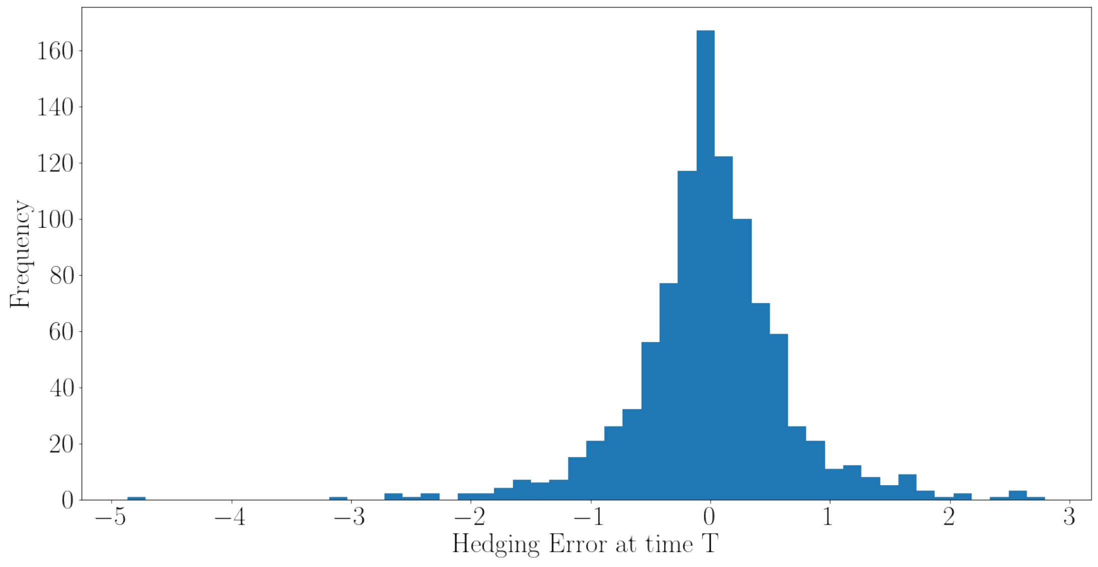
```
        if self.constant_vol_assumption:
            return self.vol_model.time_varying_vol(self.T)
        else:
            return np.sqrt(1 / (end_time - start_time) * self.vol_model.
↪integrate_vol(start_time = start_time, end_time = end_time))

    def simulation_setup(self):
        self.times, self.paths = simulate_gbm(self.S0, self.mu, self.vol_model,␣
↪self.n_steps, self.n_sims, self.simulation_end_time)

    def get_final_call_prices(self):
        S_T = self.paths[:, -1]
        if self.simulation_end_time == self.T:
            #then call is maturing now so can use the payoff formula
            self.final_call_prices = np.maximum(S_T - self.K, 0)
        else:
            #call not yet matured, so needs to be priced using BS
            integrated_vol = self.get_integrated_vol(self.simulation_end_time,␣
↪self.T)
            self.final_call_prices = black_scholes_call_price(S_T, self.
↪simulation_end_time, self.K, self.T, self.r, integrated_vol)

    def simulate_delta_varying_vol(self):

        portfolio = np.zeros(self.paths.shape)
        portfolio[:, 0] = self.initial_price

        for i in range(0, len(self.times) - 1):
            t = self.times[i]

            S_at_t = self.paths[:, i]
            S_at_t_plus_dt = self.paths[:, i+1]

            vol_t = self.get_integrated_vol(t, self.T)
            delta_t = black_scholes_call_delta(S_at_t, t, self.K, self.T, self.
↪r, vol_t)

            if self.do_gamma_hedging:
                #we are holding an option. We derive the option holding based␣
↪on gamma, then the stock holding based on delta
                t_plus_dt = self.times[i+1]

                #can use same vol everywhere because it is assumed constant
                hedge_option_at_t = black_scholes_put_price(S_at_t, t, KH, self.
↪T,self.r, vol_t)
```

```
                hedge_option_at_t_plus_dt =␣
↪black_scholes_put_price(S_at_t_plus_dt, t_plus_dt, KH, self.T,self.r, vol_t)

                hedging_gamma = black_scholes_gamma(S_at_t, t, KH, self.T,self.
↪r, vol_t)
                hedging_delta = black_scholes_put_delta(S_at_t, t, KH, self.
↪T,self.r, vol_t)

                target_gamma = black_scholes_gamma(S_at_t, t, self.K, self.
↪T,self.r, vol_t)

                option_holding = target_gamma / hedging_gamma
            else:
                #we aren't holding any option, so all the below need to be 0
                hedge_option_at_t = 0
                hedge_option_at_t_plus_dt = 0
                option_holding = 0
                hedging_delta = 0

            stock_holding = (delta_t - option_holding * hedging_delta)

            bank_at_t = portfolio[:, i] - stock_holding * S_at_t -␣
↪option_holding * hedge_option_at_t
            bank_at_t_plus_dt = bank_at_t * np.exp(self.dt * self.r)

            new_stock_value = stock_holding * S_at_t_plus_dt
            new_option_value = option_holding * hedge_option_at_t_plus_dt

            portfolio[:, i + 1] = bank_at_t_plus_dt + new_stock_value +␣
↪new_option_value

        final_portfolio = portfolio[:, -1]
        error = final_portfolio - self.final_call_prices
        return error, self.initial_price
```

0.2 Q1 - Delta Hedging

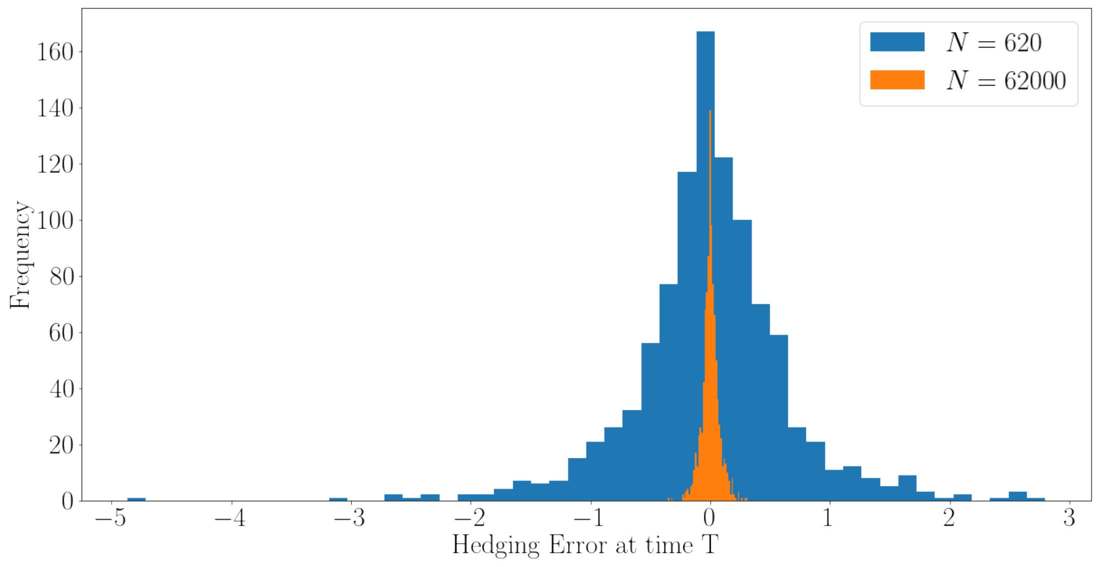
[ ]:
```
vol_model = time_varying_vol()

Q1 = Hedger(S0, K, T, r, mu, n_sims, n_steps, vol_model = vol_model)
error_Q1, price_Q1 = Q1.simulate_delta_varying_vol()
plt.hist(error_Q1, bins = 50)
plt.xlabel('Hedging Error at time T')
plt.ylabel('Frequency')
```

[ ]:
```
Text(0, 0.5, 'Frequency')
```

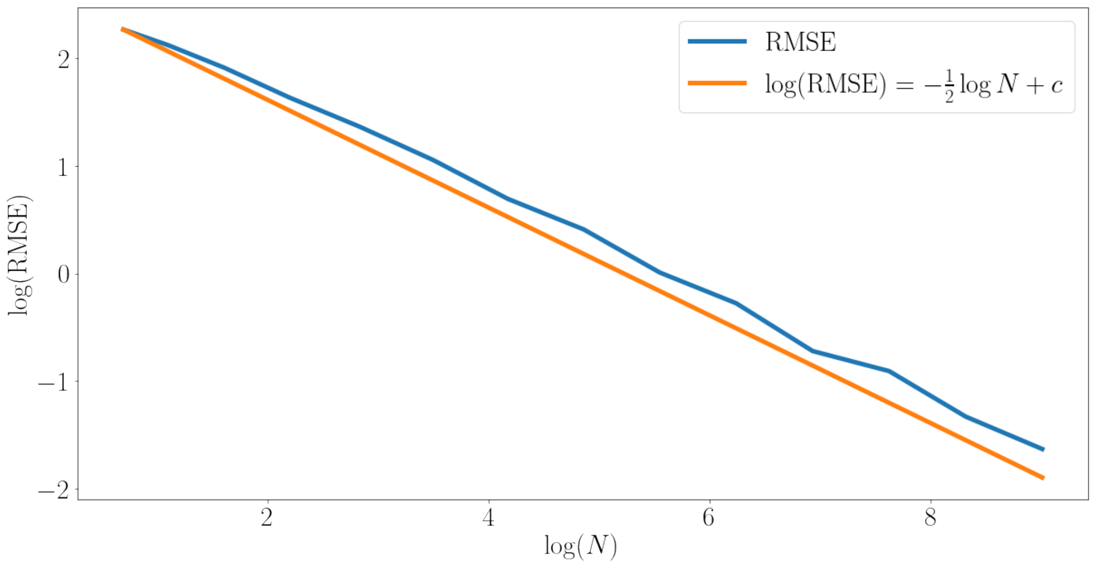


```
[ ]: High_steps_Q1 = Hedger(S0, K, T, r, mu, n_sims, n_steps * 100, vol_model =␣
      ↪vol_model)
     error_Q1_high, price_Q1_high2 = High_steps_Q1.simulate_delta_varying_vol()

     plt.hist(error_Q1, bins = 50, label = r'$N = 620$')
     plt.hist(error_Q1_high, bins = 50, label = r'$N = 62000$')
     plt.xlabel('Hedging Error at time T')
     plt.ylabel('Frequency')
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x234253449d0>
```



```
[ ]: print(price_Q1)

     3.1573080857906497

[ ]: assert_almost_equal(np.percentile(error_Q1, q = [2.5, 97.5]), np.array([-1.
      ↪4437569,  1.4015494]))

[ ]: n_points = 14
     n_steps_rme_testing = np.zeros(n_points)
     rms_error = np.zeros(n_points)
     for i in range(0, n_points):
         n_steps_rme_testing[i] = 2 ** i + 1
         dummy_hedger = Hedger(S0, K, T, r, mu, n_sims, int(n_steps_rme_testing[i]),␣
      ↪vol_model = vol_model)
         error, price = dummy_hedger.simulate_delta_varying_vol()
         rms_error[i] = np.sqrt(np.mean(error ** 2))

     plt.plot(np.log(n_steps_rme_testing), np.log(rms_error), label = 'RMSE')
     plt.plot(np.log(n_steps_rme_testing), -0.5 * np.log(n_steps_rme_testing) + (np.
      ↪log(rms_error[0]) + 0.5 * np.log(n_steps_rme_testing[0])), label =␣
      ↪r'$\log($RMSE$) = -\frac{1}{2}\log{N} + c$')


     plt.xlabel(r'$\log(N)$')
     plt.ylabel(r'$\log($RMSE$)$')
     plt.legend()

[ ]: <matplotlib.legend.Legend at 0x23423570d60>
```

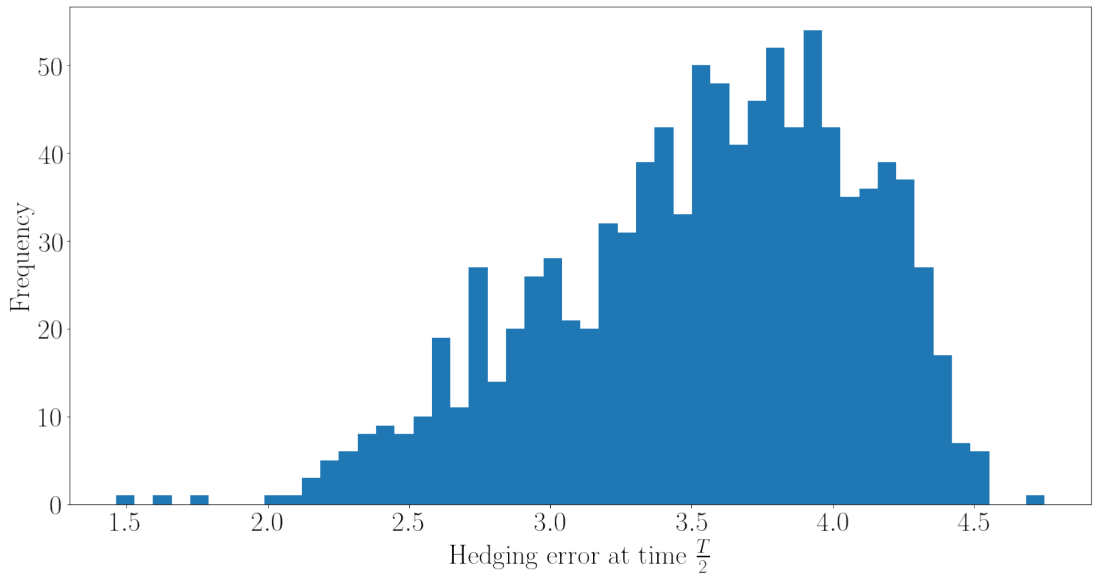


0.3 Q2

```
[ ]: Dummy = Hedger(S0, K, T, r, mu, n_sims, n_steps, vol_model = vol_model,␣
      ↪simulation_end_time = T/2)

     e, p = Dummy.simulate_delta_varying_vol()
```

```
[ ]: Q2 = Hedger(S0, K, T, r, mu, 1_000, 1_000, vol_model = vol_model,␣
      ↪simulation_end_time = T/2, constant_vol_assumption = True)
     error_Q2, price_Q2 = Q2.simulate_delta_varying_vol()


     plt.hist(error_Q2, bins = 50)

     plt.xlabel(r'Hedging error at time $\frac{T}{2}$')
     plt.ylabel(r'Frequency')
```

```
[ ]: Text(0, 0.5, 'Frequency')
```



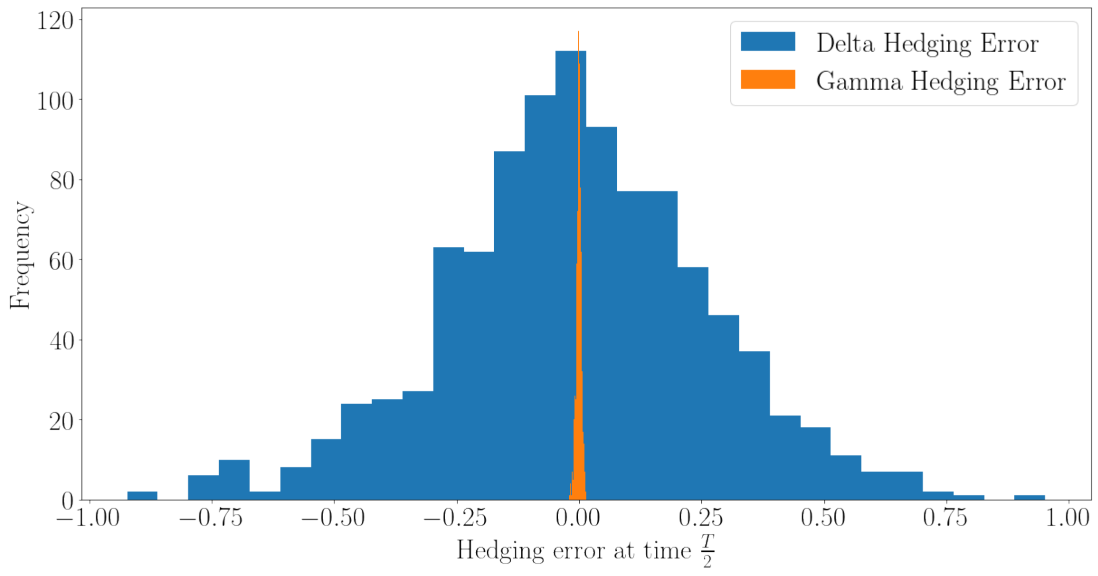
```
[ ]: assert_almost_equal(np.percentile(error_Q2, q = [2.5, 97.5]), np.array([2.
      ↪3824194, 4.3781215]))
```

0.4 Q3

```
[ ]: Q3_vol_model = time_varying_vol()

     def flat_vol(t):
         return sigma1 + sigma2

     Q3_vol_model.time_varying_vol = flat_vol

     assert(Q3_vol_model.time_varying_vol(1) == sigma1 + sigma2)
```

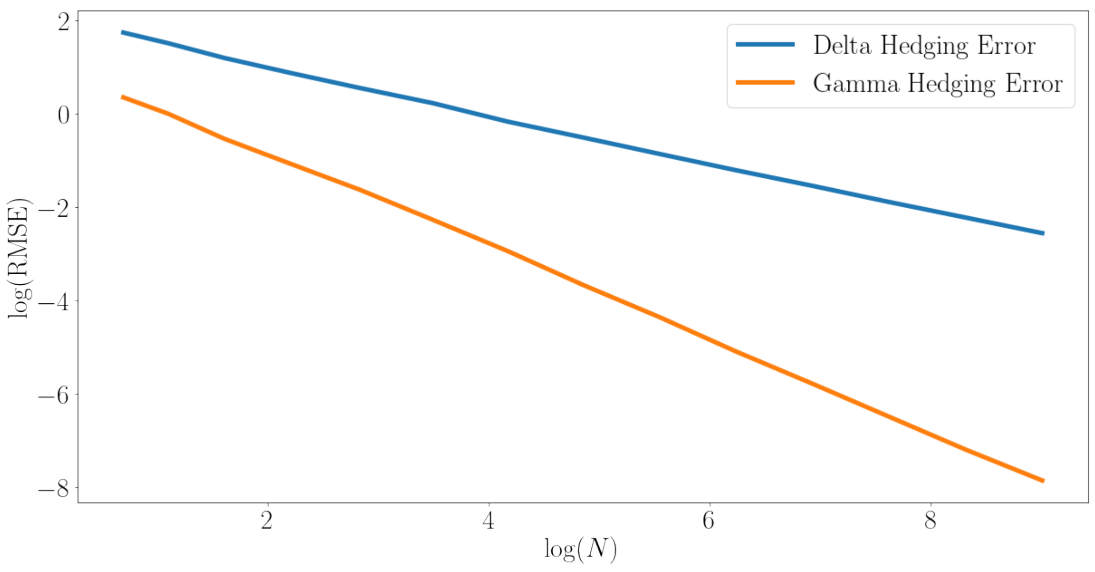
```
[ ]: Q3_gamma = Hedger(S0, K, T, r, mu, n_sims, n_steps, vol_model = Q3_vol_model,␣
       ↪simulation_end_time = T/2, constant_vol_assumption = True, do_gamma_hedging␣
       ↪= True)
     error_Q3_gamma, price_Q3_gamma = Q3_gamma.simulate_delta_varying_vol()

     Q3_delta = Hedger(S0, K, T, r, mu, n_sims, n_steps, vol_model = Q3_vol_model,␣
       ↪simulation_end_time = T/2, constant_vol_assumption = True, do_gamma_hedging␣
       ↪= False)
     error_Q3_delta, price_Q3_delta = Q3_delta.simulate_delta_varying_vol()
     plt.hist(error_Q3_delta, bins = 30, label = 'Delta Hedging Error')
     plt.hist(error_Q3_gamma, bins = 30, label = 'Gamma Hedging Error')

     plt.xlabel(r'Hedging error at time $\frac{T}{2}$')
     plt.ylabel(r'Frequency')
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x23426755300>
```



```
[ ]: n_points = 14
     n_steps_rme_testing = np.zeros(n_points)
     rms_error_delta = np.zeros(n_points)
     rms_error_gamma = np.zeros(n_points)

     for i in range(0, n_points):
         n_steps_rme_testing[i] = 2 ** i + 1
         dummy_hedger_delta = Hedger(S0, K, T, r, mu, n_sims,␣
       ↪int(n_steps_rme_testing[i]), vol_model = Q3_vol_model,  simulation_end_time␣
       ↪= T/2, constant_vol_assumption = True, do_gamma_hedging = False)
         dummy_hedger_gamma = Hedger(S0, K, T, r, mu, n_sims,␣
       ↪int(n_steps_rme_testing[i]), vol_model = Q3_vol_model,  simulation_end_time␣
       ↪= T/2, constant_vol_assumption = True, do_gamma_hedging = True)
         delta_error, delta_price = dummy_hedger_delta.simulate_delta_varying_vol()
         rms_error_delta[i] = np.sqrt(np.mean(delta_error ** 2))

         gamma_error, gamma_price = dummy_hedger_gamma.simulate_delta_varying_vol()
         rms_error_gamma[i] = np.sqrt(np.mean(gamma_error ** 2))

     plt.plot(np.log(n_steps_rme_testing), np.log(rms_error_delta), label = 'Delta␣
       ↪Hedging Error')
     plt.plot(np.log(n_steps_rme_testing), np.log(rms_error_gamma), label = 'Gamma␣
       ↪Hedging Error')


     plt.xlabel(r'$\log(N)$')
     plt.ylabel(r'$\log($RMSE$)$')
     plt.legend()
```

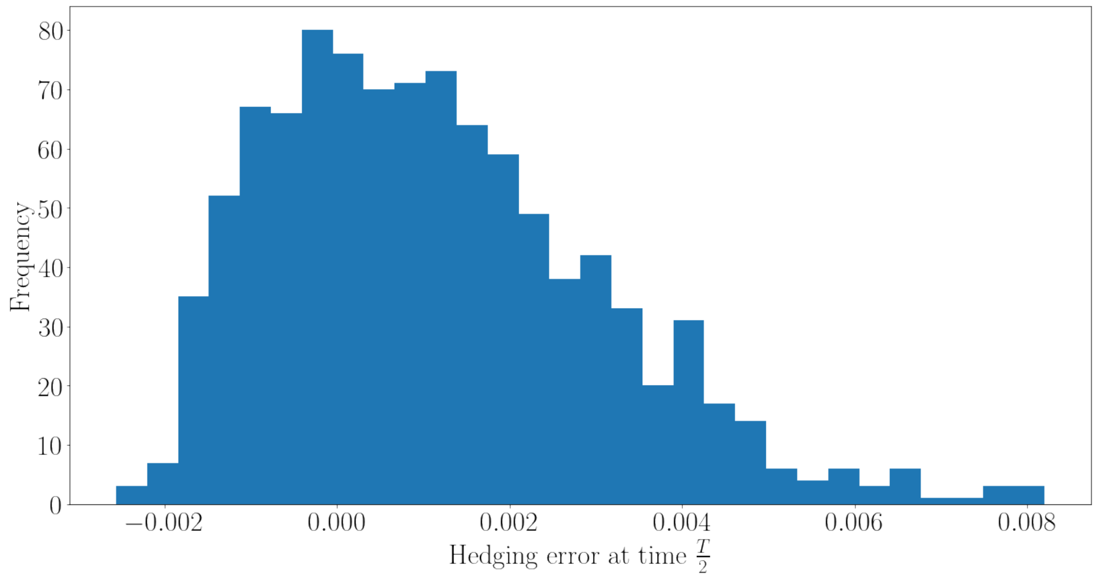
```
[ ]: <matplotlib.legend.Legend at 0x23424f43d30>
```



0.5 Q4

```
[ ]: Q4_vol_model = time_varying_vol()
     Q4 = Hedger(S0, K, T, r, mu, 1_000, 1_000, vol_model = Q4_vol_model,␣
      ↪simulation_end_time = T/2, constant_vol_assumption = True, do_gamma_hedging␣
      ↪= True)
     error_Q4, price_Q4 = Q4.simulate_delta_varying_vol()


     plt.xlabel(r'Hedging error at time $\frac{T}{2}$')
     plt.ylabel(r'Frequency')


     plt.hist(error_Q4, bins = 30)
```

```
[ ]: (array([ 3.,  7., 35., 52., 67., 66., 80., 76., 70., 71., 73., 64., 59.,
             49., 38., 42., 33., 20., 31., 17., 14.,  6.,  4.,  6.,  3.,  6.,
              1.,  1.,  3.,  3.]),
      array([-2.56403046e-03, -2.20503963e-03, -1.84604879e-03, -1.48705796e-03,
             -1.12806712e-03, -7.69076289e-04, -4.10085454e-04, -5.10946190e-05,
              3.07896216e-04,  6.66887051e-04,  1.02587789e-03,  1.38486872e-03,
              1.74385956e-03,  2.10285039e-03,  2.46184123e-03,  2.82083206e-03,
              3.17982290e-03,  3.53881373e-03,  3.89780457e-03,  4.25679540e-03,
              4.61578624e-03,  4.97477707e-03,  5.33376791e-03,  5.69275874e-03,
              6.05174958e-03,  6.41074041e-03,  6.76973125e-03,  7.12872208e-03,
              7.48771292e-03,  7.84670375e-03,  8.20569459e-03]),
      <BarContainer object of 30 artists>)
```



```
[ ]: assert_almost_equal(np.percentile(error_Q4, q = [2.5, 97.5]), [-0.0017121,  0.
       ↪0053743 ])
```

B Individual Question

Individual Question for Student with k-number k2257667

Suppose that a stock price S_t evolves according to the stochastic differential equation

$$dS_t = \mu S_t \, dt + \sigma(t) S_t \, dW_t. \tag{1}$$

Here $S_0 = 149.00$, $\mu = 0.13$ is a constant, $\sigma(t)$ is given by the deterministic function

$$\sigma(t) = \begin{cases} 0.11 + 0.13 \times \frac{t-0.30}{T-0.30} & t > 0.30 \\ 0.11 & \text{otherwise} \end{cases},$$

and W_t is a Brownian motion. Suppose that there is also a risk-free asset which grows at the continuously compounded rate $r = 0.01$.

If σ was constant, a call option with strike K and maturity T at a time $0 < t < T$ would have its price given by the Black–Scholes formula

$$\text{price} = \text{BS}^{\text{call}}(S_t, K, T-t, \sigma, r).$$

We will similarly write $\text{BS}^{\text{put}}(S_t, K, T-t, \sigma, r)$ for the Black–Scholes formula for put options.

1. The price of an option at time t in this time-dependent volatility model is given by

   $$\text{BS}^{\text{call}}(S_t, K, T-t, s_t, r)$$

   where

   $$(s_t)^2 = \frac{1}{T-t} \int_t^T \sigma(t)^2 \, \mathrm{d}t$$

   Show numerically using a simulation that it is possible to replicate a call option with strike $K = 188.00$ and maturity $T = 1.70$ by charging this price and then following the delta-hedging strategy.

   Make sure that you describe how you simulated the stock prices, and give a clear mathematical description of how you performed the simulation of delta-hedging. Give the price, P_0 of the option at time 0 in your table of results.

2. Show that the delta-hedging strategy does not work if the trader is mistaken about the volatility. To do this, suppose a trader believes the stock has constant volatility $\sigma(T)$, where T is the maturity, and decides to follow the delta-hedging strategy with all prices and deltas computed on this basis. However, suppose that in reality the stock price process is still generated according to the equation (1). Calculate numerically the expected profit (which may be negative), p^Δ, of the trader if:

   (i) they sell the option at time 0 for the price $\text{BS}^{\text{call}}(S_0, K, T, \sigma(T), r)$;

   (ii) they then follow the delta-hedging strategy assuming constant volatility up to time $\frac{T}{2}$;

(iii) they then buy back the option for the price $\mathrm{BS}^{\mathrm{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ and calculate their profit.

Use 1000 simulations and 1000 time-steps to compute this value. Give the 95% confidence interval for p^{Δ} in your table of results.

3. Suppose it is possible at all times t, $0 \leq t < T$, to buy or sell a put option with strike $K^H = S_0$ for a price $\mathrm{BS}^{\mathrm{put}}(S_t, K^H, T-t, \sigma(T), r)$. In the gamma-hedging strategy the trader assumes that the volatility is constant and equal to $\sigma(T)$. At all times they hold q_S units of stock and q_H units of the hedging option with strike K^H in such a way as to ensure that the total delta of their portfolio is equal to the delta of the option with strike K and the total gamma of their portfolio is equal to the gamma of the option with strike K. Any remaining funds (or debts) grow at the risk-free rate.

   Show through a simulation that, if the stock price is generated using a model where volatility is constant and equal to $\sigma(T)$, then a trader can charge $\mathrm{BS}^{\mathrm{call}}(S_0, K, T, \sigma(T), r)$ at time 0, and can then guarantee to have a portfolio with market value $\mathrm{BS}^{\mathrm{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ at time $\frac{T}{2}$ by following the gamma-hedging strategy.

   Since continuous time trading is not possible, perfect replication using the delta-hedging strategy cannot be achieved in practice. Show using appropriate charts that one advantage of the gamma-hedging strategy is that with gamma-hedging you do not need to rehedge so often to replicate the price $\mathrm{BS}^{\mathrm{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ at time $\frac{T}{2}$ with reasonable accuracy.

   (*The reason you are only being asked to consider hedging up to time $T/2$ is to ensure that the gamma remains bounded throughout your simulation. One difficulty with gamma hedging is that the gamma can become very large if the stock price is near the strike price close to maturity and the question is designed to avoid this issue.)*

4. Another feature of the gamma-hedging strategy is that so long as the price of the hedging option with strike K^H is given by $\mathrm{BS}^{\mathrm{put}}(S_t, K^H, T-t, \sigma(T), r)$ it is possible to replicate the payoff $\mathrm{BS}^{\mathrm{call}}(S_{T/2}, K, T/2, \sigma(T), r)$ at time $\frac{T}{2}$ for the price $\mathrm{BS}^{\mathrm{call}}(S_0, K, T, \sigma(T), r)$ even if the underlying stock price is generated using equation (1). Calculate the expected profit p^{Γ} of following the gamma hedging strategy in this case using 1000 scenarios and 1000 time steps. Give a 95% confidence interval for your result.

You must BEGIN your write up by displaying the following table of results.

k-Number	k2257667
Price P Confidence interval for p^{Δ} Confidence interval for p^{Γ}	