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

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 investigate the hedging of a European call in a modified Black-Scholes environment. The stock being modelled has time-varying volatility, rather than the constant volatility that classical Black-Scholes assumes.

I analyse how well a trader can replicate the call's payoff using delta and gamma hedging. Delta hedging succeeds if the correct volatility function was used for the hedging, but not if the trader assumed a constant volatility. Gamma hedging, however, can replicate a call payoff in this scenario.

Computations are based on material taught in FM06, [Arm23]. I make particular use of the sections on stock simulation and delta hedging. Numerical results above are to 5 significant figures.

2 Specifying The Model

The stock being modelled has 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 use delta and gamma hedging to replicate the payoff of a European call option on the stock. The option has time to maturity T and strike $K = 188.00$. The continuously compounded risk-free rate is $r = 0.01$. At time $0 \le t < T$, the price of the call is:

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

with

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

and $BS^{call}(S_t, K, T-t, s_t, r)$ the Black-Scholes formula. The time-0 price is $c_0 = 3.1573$; I will verify this numerically when simulating delta hedging.

I will also need to calculate some Greeks when implementing hedging strategies. Applying [Hul17] chapter 19 shows that delta and gamma are:

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

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

3 Simulating The Stock Price

Before simulating hedging, I had to simulate the stock itself. I used an Euler-Maruyama scheme on the log of the stock, as described in [Arm23]. By Itō's lemma if S_t 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, I divided the interval $[0, T]$ into N sub-intervals of equal length $\delta t = T/N$. This gives a set of points $\{0 = t_0, t_1, \dots, t_N = T\}$, $t_i = i\delta t$. Generally I have set $N = 1,000$, as the assignment asks for some results calculated using this value.

The SDE is then simulated as:

$$\begin{aligned}
X_{t_{i+1}} &= X_{t_i} + \left(\mu - \frac{\sigma(t_i)^2}{2}\right) \delta t + \sigma(t_i)(dW_{t_{i+1}} - dW_t) \\
&= X_{t_i} + \left(\mu - \frac{\sigma(t_i)^2}{2}\right) \delta t + \sigma(t_i)\sqrt{\delta t}\epsilon_t
\end{aligned} \tag{6}$$

where ϵ_t are independent and identically distributed standard Normal variables that I generated using the built-in function `numpy.random.normal` ([HMvdW+20]). Taking $X_0 = \log(S_0)$ and applying (6) iteratively for $i = 0, 1, \dots, N-1$ gives a simulation of the log-stock process up to time T.

With M simulations and N timesteps, the output is a $M \times (N+1)$ matrix $\boldsymbol{X}$. The jth row contains simulation j of the log-stock. The ith column vector, $\boldsymbol{X^{(i)}}$, contains all simulations of the log-stock at time $(i-1)\delta t$.

Finally, I generated a matrix of simulations of the stock, $\boldsymbol{S} = \mathrm{e}^{\boldsymbol{X}}$, applying exponentiation element-wise. I used $\boldsymbol{S}$ with equations (3) and (4) to calculate the delta of the call for all times in each simulation. The time parameter required for these calculations, t, is found by noting that column $\boldsymbol{S}^{(i)}$ corresponds to the time $(i-1)\delta t$. The matrix of deltas is $\boldsymbol{\Delta}_{\text{call}}$.

4 Simulating Delta Hedging

4.1 Hedging with the Correct Volatility

I now numerically test the call price c_t in equation (2). I simulated delta hedging of a trader who knows the stock follows the SDE (1). If the payoff of the call is accurately replicated, then c_t is the time-t price of the call.

I set an empty $M \times (N+1)$ matrix $\boldsymbol{W}$ as the wealth of the trader at all times in each simulation. I assumed they charged c_0 for the option, which is their initial wealth and is placed in a risk-free account i.e. $\boldsymbol{W}^{(1)} = c_0$. The trader then holds $\Delta_t(t, S_t)$ of the stock and $w_t - \Delta_t(t, S_t)$ in a risk-free account. Following [Arm23], I simulated the rebalancing of this portfolio at the times t_i. The trader's wealth across all simulations 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 accumulation of the risk-free account, and the second the change in the stock price. Note that in this notation, multiplications of column vectors is applied element-wise and not as a dot product.

I applied (10) iteratively for $i = 0, 1, ...N$ to get the final wealth in each simulation, $\boldsymbol{W}^{(N+1)}$. The hedging error is the difference between final wealth and the call's payoff at time T i.e. $\boldsymbol{W}^{(N+1)} - \max(\boldsymbol{S}^{(N+1)} - K, 0)$. Figure 1 shows hedging error histograms for $1,000$ simulations in two cases: $N = 1,000$ timesteps and $N = 100,000$ timesteps. The histograms are centered about 0 and the error is much smaller when N larger.

This suggests convergence to 0. To be certain, I made a log-log plot in figure 2 of the root mean square error (RMSE) against N. On this plot the error is a straight line with gradient $-1/2$ (which can be seen by comparison

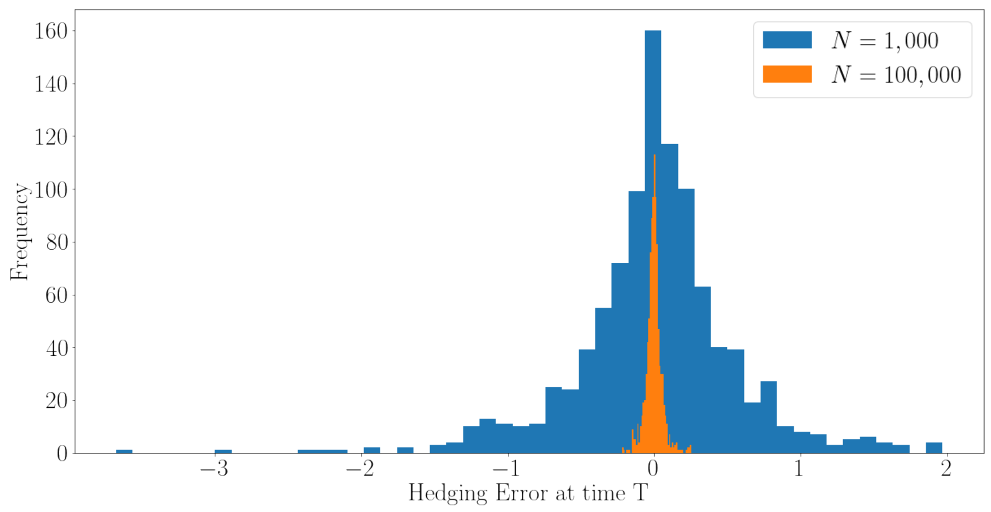


Figure 1: Histograms of delta hedging errors for $1,000$ simulations and two different timestep amounts, $N = 1,000$ and $N = 100,000$

to a line with this gradient). This implies $O(N^{-1/2})$ convergence, which is consistent with [Arm23] Chapter 7. Overall this implies that delta hedging is converging, and therefore that it replicates the call payoff. In particular this means that (2) is correct and so I have verified the time-0 price is $c_0 = 3.1573$.

4.2 Hedging with A Constant Volatility Assumption

In the previous section the trader knew the volatility function $\sigma(t)$ when delta hedging. However, they may assume the stock has a constant volatility $\tilde{\sigma}(t) = \sigma(T)$. In that case, the stock's SDE is:

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

This is a special case of (1) where (3) is constant and equal to $\sigma(T)\ \forall\ t$. Equations (2) and (4) show the trader sets the option's price and Greeks as:

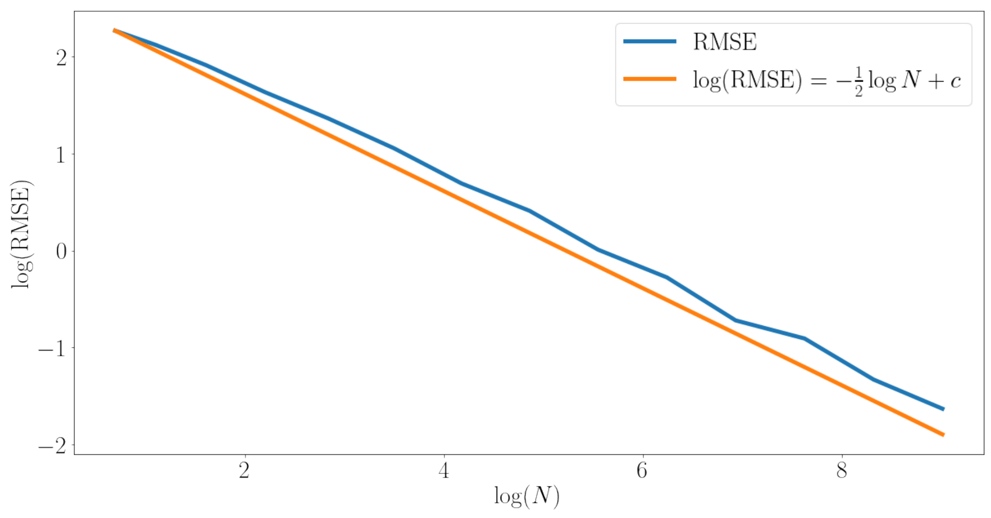


Figure 2: Log-log plot of delta hedging Root Mean Square Error (RMSE) vs. number of timesteps N for 1,000 simulations. Results are compared to a line with gradient $-1/2$.

$$\begin{aligned}
\tilde{c}_t &= BS^{\text{call}}(S_t, K, T-t, \sigma(T), r) \\
\tilde{\Delta}_{\text{call}}(t, S_t) &= \frac{\partial c_t}{\partial S_t} = \Phi(\tilde{d}_1) \\
\tilde{\Gamma}_{\text{call}}(t, S_t) &= \frac{\partial^2 c_t}{\partial S_t^2} = \frac{\Phi'(d_1)}{S_t \sigma(T)\sqrt{T-t}} \\
\tilde{d}_1 &= \frac{\log\left(\frac{S_t}{K}\right) + (r + \frac{\sigma(T)^2}{2})(T-t)}{\sigma(T)\sqrt{T-t}}
\end{aligned} \tag{9}$$

To find the profit/loss of a trader delta hedging with this assumption, I simulated the stock with the original SDE (1). I used (9) for the trader's time-0 price of the call, $\tilde{c}_0$, and a matrix of deltas, $\tilde{\boldsymbol{\Delta}}_{\text{call}}$. I made a new wealth matrix $\tilde{\boldsymbol{W}}$ with initial wealth $\tilde{\boldsymbol{W}}^{(1)} = \tilde{c}_0$. Then I simulated wealth as:

$$\tilde{\boldsymbol{W}}^{(i+1)} = \mathrm{e}^{r\delta t}(\tilde{\boldsymbol{W}}^{(i)} - \tilde{\boldsymbol{\Delta}}^{(i)}_{\text{call}}\boldsymbol{S}^{(i)}) + \tilde{\boldsymbol{\Delta}}^{(i)}_{\text{call}}(\boldsymbol{S}^{(i+1)} - \boldsymbol{S}^{(i)}). \tag{10}$$

I simulated delta hedging up to time $T/2$ with $1,000$ simulations and $1,000$ timesteps, and calculated the hedging error, the difference between time-$T/2$ wealth and the expected payoff, $\tilde{c}_{T/2}$. The hedging errors in Figure 3 show the replication no longer works; the errors all positive with no convergence to 0. This is expected, because $\tilde{\sigma}(t) = \sigma(T) \geq \sigma(t)\,\forall\, t \leq T$. As [KJPS98] shows, in this situation the hedging error over time will be non-decreasing i.e. all hedging errors will be positive. This is what is seen in the chart.

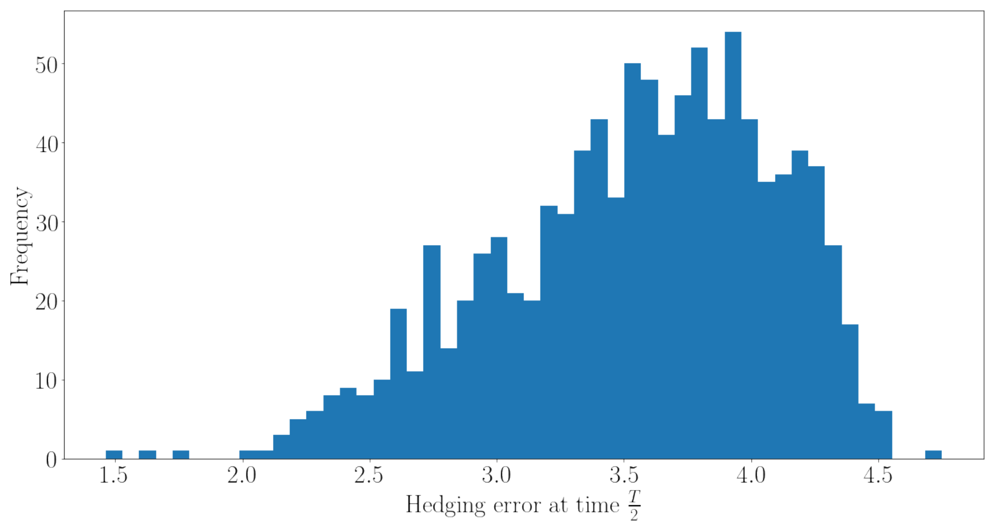


Figure 3: Histogram of delta hedging errors when the trader assumes constant volatility, but the stock has time-varying volatility. Results for 1,000 simulations and 1,000 timesteps.

5 Simulating Gamma Hedging

5.1 Deriving The Gamma Portfolio

Gamma hedging is an extension of delta hedging. The trader again assumes constant volatility equal to $\sigma(T)$. This means $\tilde{c}_t$, $\tilde{\Delta}(t, S_t)$, $\tilde{\Gamma}(t, S_t)$ in equation (9) still describe the trader's opinions on the call price and its Greeks.

There is also a new tradable option - a European put with strike $K^H = S_0$ and maturity T. Its price is $BS^{\text{put}}(S_t, K^H, T-t, \sigma(T), r)$, so its Greeks are:

$$\begin{aligned}
\Delta_{\text{put}}(t, S_t) &= \frac{\partial p_t}{\partial S_t} = -\Phi(-d_1^H) \\
\Gamma_{\text{put}}(t, S_t) &= \frac{\partial^2 p_t}{\partial S_t^2} = \frac{\Phi'(d_1^H)}{S_t \sigma(T)\sqrt{T-t}} \\
d_1^H &= \frac{\log\left(\frac{S_t}{K^H}\right) + (r + \frac{\sigma(T)^2}{2})(T-t)}{\sigma(T)\sqrt{T-t}}.
\end{aligned} \tag{11}$$

In gamma hedging, the trader charges the time-zero price $\tilde{c}_0$. Over time, they hold q_S of the stock and q_H of the put option. The parameters are chosen so the delta and gamma of their portfolio equals those of the option being replicated. The remainder, $w_t - q_S S_t - q_H p_t$, is held in a risk-free account.

The delta of the trader's portfolio is $q_S + q_H \Delta_{\text{put}}(t, S_t)$ and its gamma is

$q_H \Gamma_{\text{put}}(t, S_t)$. Because the trader assumes constant volatility, these should be equal to $\tilde{\Delta}_{\text{call}}(t, S_t)$ and $\tilde{\Gamma}_{\text{call}}(t, S_t)$. I solved these equations to get:

$$\begin{aligned} q_H(t, S_t) &= \frac{\tilde{\Gamma}_{\text{call}}(t, S_t)}{\Gamma_{\text{put}}(t, S_t)} \\ q_S(t, S_t) &= \tilde{\Delta}_{\text{call}}(t, S_t) - q_H(t)\Delta_{\text{put}}(t, S_t). \end{aligned} \tag{12}$$

I simulated gamma hedging in the same way as delta hedging: I first simulated stock prices according to an SDE. I used this to calculate the price, delta and gamma of the put across all simulations - given by the $M \times (N+1)$ matrices $\boldsymbol{P}$, $\boldsymbol{\Delta}_{\text{put}}$ and $\boldsymbol{\Gamma}_{\text{put}}$. Similarly, the trader's valuation of the call's key information is given by $\tilde{\boldsymbol{\Delta}}_{\text{call}}$ and $\tilde{\boldsymbol{\Gamma}}_{\text{call}}$. Using these matrices, and 12, I derived $\boldsymbol{Q}_H, \boldsymbol{Q}_S$, matrices containing $q_H(t, S_t)$ and $q_S(t, S_t)$.

I then created an empty wealth matrix, $\bar{\boldsymbol{W}}$. The initial wealth is $\bar{\boldsymbol{W}}^{(1)} = \tilde{c}_0$. I simulated the trader's wealth over time as:

$$\begin{aligned} \boldsymbol{W}^{(i+1)} &= \mathrm{e}^{r\delta t}(\boldsymbol{W}^{(i)} - \boldsymbol{Q}_S^{(i)}\boldsymbol{S}^{(i)} - \boldsymbol{Q}_H^{(i)}\boldsymbol{P}^{(i)}) \\ &\quad + \boldsymbol{Q}_S^{(i)}(\boldsymbol{S}^{(i+1)} - \boldsymbol{S}^{(i)}) \\ &\quad + \boldsymbol{Q}_H^{(i)}(\boldsymbol{P}^{(i+1)} - \boldsymbol{P}^{(i)}) \end{aligned} \tag{13}$$

where the first term is cash accumulation, the second changes in stock price, and the third changes in the put price. This simulates gamma hedging, and I now compare it to delta hedging.

5.2 Delta vs. Gamma Hedging: Constant Volatility

I first looked at the case of constant volatility, where the stock SDE is:

$$dS_t = \mu S_t dt + \sigma(T) S_t dW_t. \tag{14}$$

I performed $1,000$ simulations of delta and gamma hedging as described in sections 4.2 and 5.1. The simulation ends at time $T/2$. The hedging error is the difference between the wealth of the trader at this time and the payoff $BS^{\text{call}}(S_{T/2}, K, T/2, \sigma(T), r)$.

Figure 4 has a histogram of the errors, and figure 5 the RMSE vs. number of simulations. They show that gamma hedging is more effective than delta hedging - the errors in figure 4 are more tightly grouped around 0. Moreover,

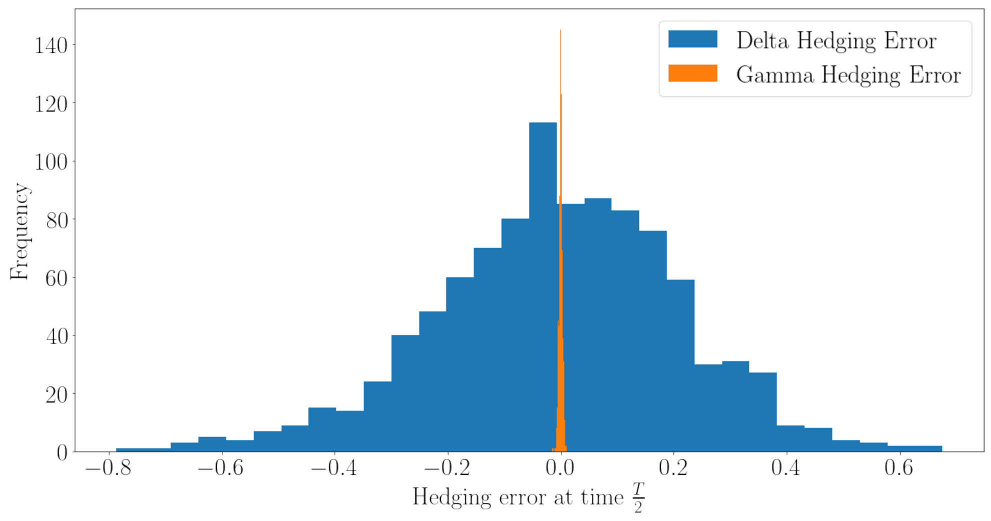


Figure 4: Histograms of delta and gamma hedging errors when the stock has constant volatility. Results at time T/2 with $1,000$ simulations and $1,000$ timesteps

in figure 5 the gamma RMSE is always smaller than delta RMSE for a given N, and the RMSE converges to 0 faster - $O(N^{-1})$ vs $O(N^{-1/2})$.

I can conclude that gamma hedging replicates a call payoff far more accurately in this context. The figures show it is not necessary to rehedge so often to replicate the price of a call option to a reasonable level of accuracy.

5.3 Gamma Hedging with The Wrong Volatility

I now analyse another advantage of gamma hedging. Section 4.2 showed delta-hedging with a constant volatility does not replicate the expected payoff if the stock follows equation (1). Gamma hedging, however, does this.

To show this, I followed the same approach described in section 4.2: I generated 1,000 simulations of the stock using the SDE in (1) and 1,000 timesteps. I simulated gamma hedging as described in section 5.1. The time-$T/2$ hedging error is then the difference between the final wealth of the trader and their target payoff, $BS^{\text{call}}(S_{1/2}, K, 1/2, \sigma(T), r)$.

The hedging error results at time 1/2 are in figure 6. The chart shows successful replication, as the errors are very close to 0.

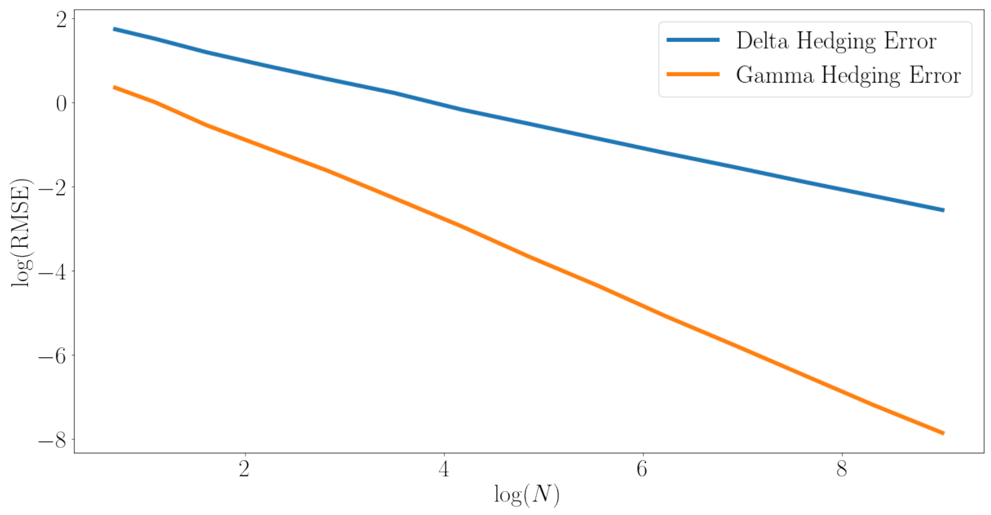


Figure 5: Log-log plot of RMSE of delta and gamma hedging strategies vs. number of timesteps N when the stock has constant volatility. Results based on $1,000$ simulations.

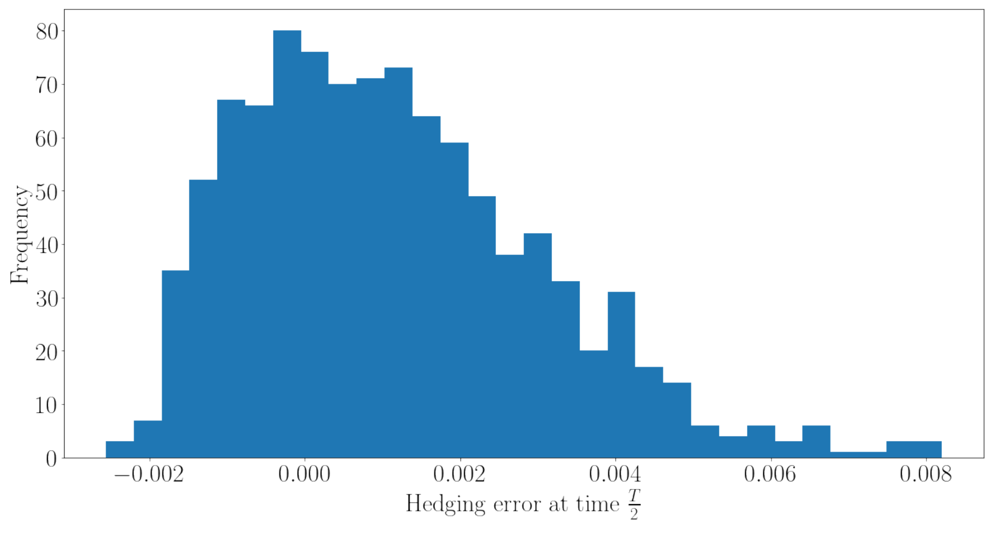


Figure 6: Histogram of gamma hedging errors at time $T/2$ when the stock follows the equation (1). Results shown for $1,000$ simulations and $1,000$ timesteps.

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 #for consistency with assignment
n_steps = 1_000 #for consistency with assignment
```

[ ]:
```
### Helper Functions

### Black-Scholes

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

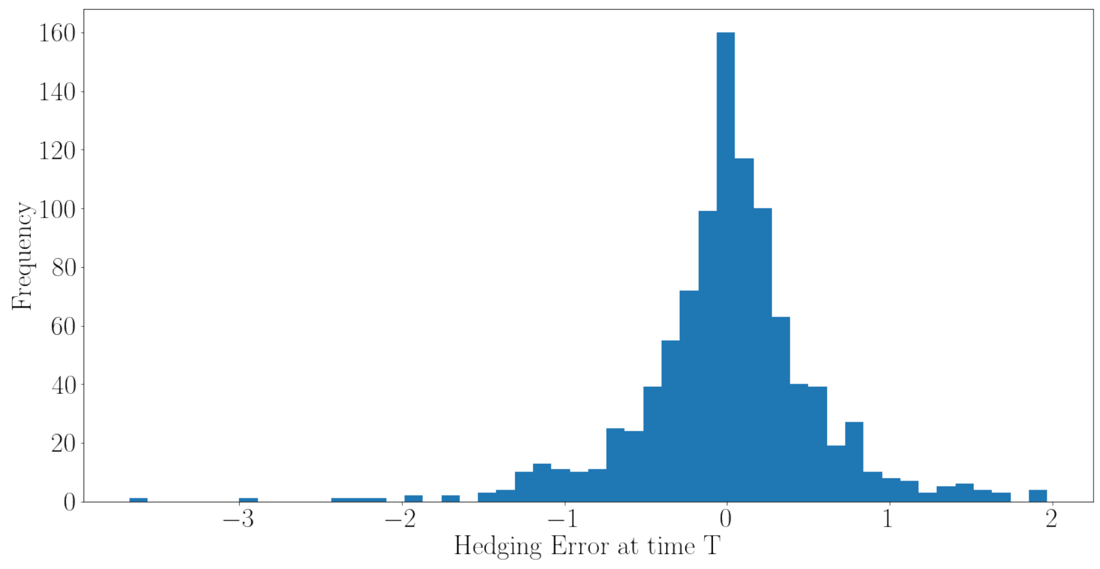
```
        if self.constant_vol_assumption:
            return self.vol_model.time_varying_vol(self.T)
        else:
            return np.sqrt(1 / (end_time - start_time) * self.vol_model.
↪integrate_vol(start_time = start_time, end_time = end_time))

    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

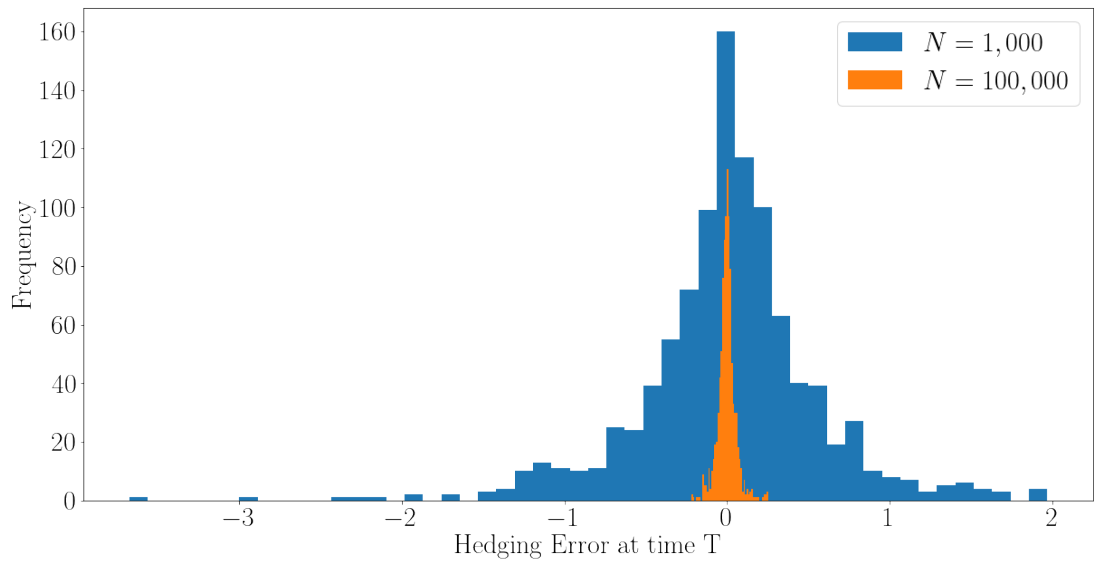
[ ]:
```
vol_model = time_varying_vol()

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'$N = 1,000$')
     plt.hist(error_Q1_high, bins = 50, label = r'$N = 100,000$')
     plt.xlabel('Hedging Error at time T')
     plt.ylabel('Frequency')
     plt.legend()
```

```
[ ]: <matplotlib.legend.Legend at 0x1c82bf9d8a0>
```



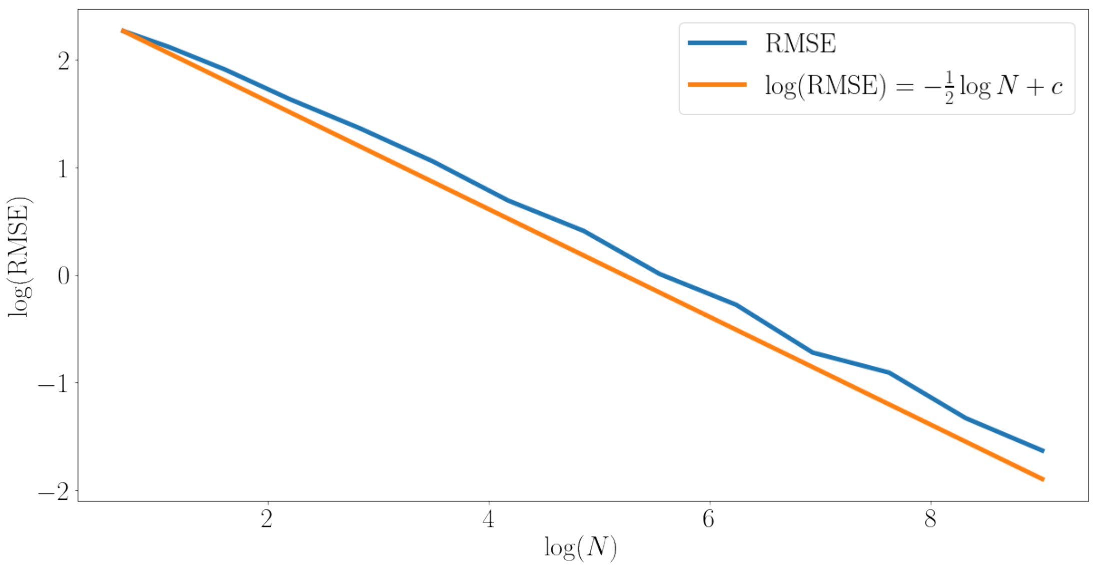
```
[ ]: print(price_Q1)
```

```
3.1573080857906497
```

```
[ ]: #Check
     assert_almost_equal(np.percentile(error_Q1, q = [2.5, 97.5]), np.array([-1.
      ↪1999189,  1.168614]))
```

```
[ ]: 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 0x1c82b74fbe0>
```

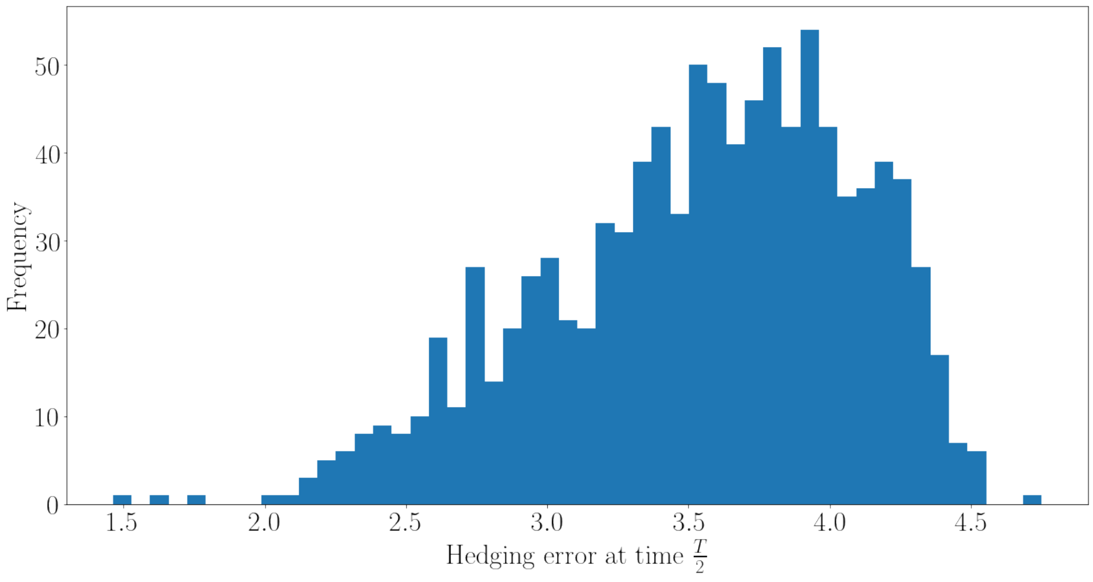


0.3 Q2

```
[ ]: 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')
```



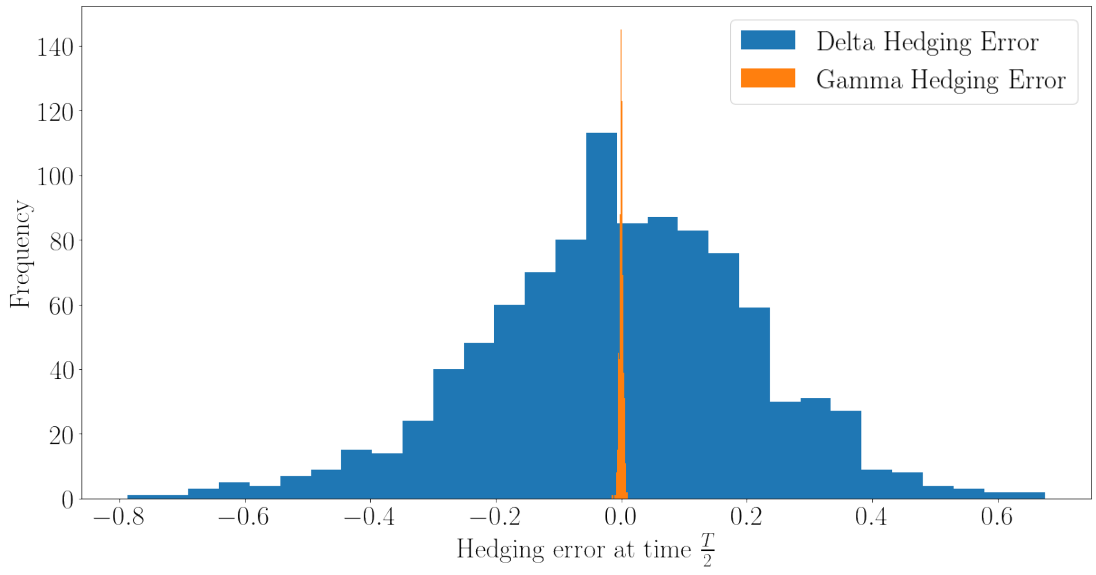
```
[ ]: assert_almost_equal(np.percentile(error_Q2, q = [2.5, 97.5]), np.array([2.
       ↪3824194, 4.3781215]))
```

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()
```

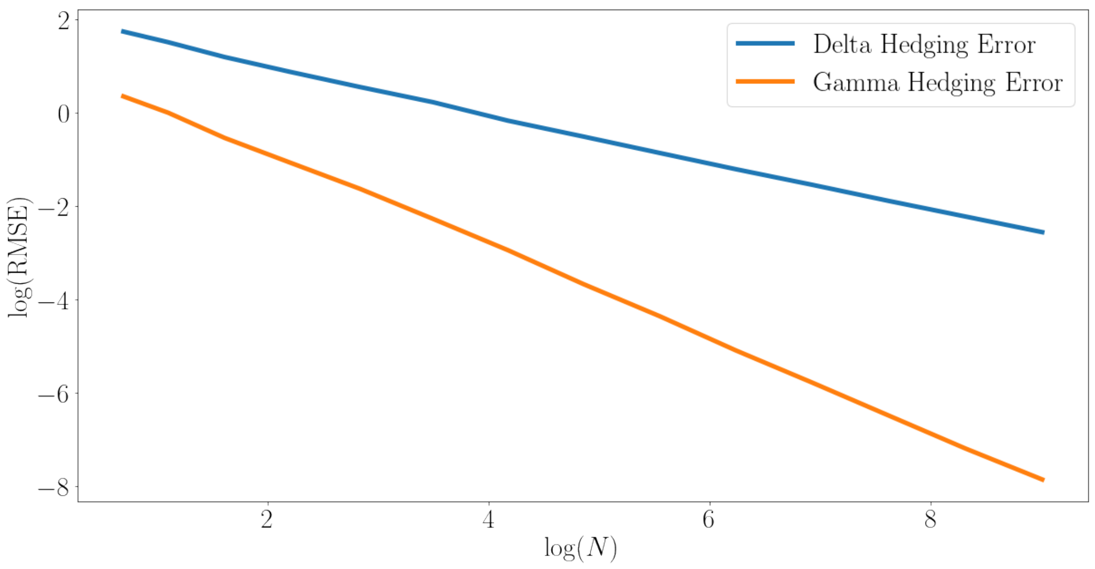
```
[ ]: <matplotlib.legend.Legend at 0x1c82cb11180>
```



```
[ ]: 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()
```

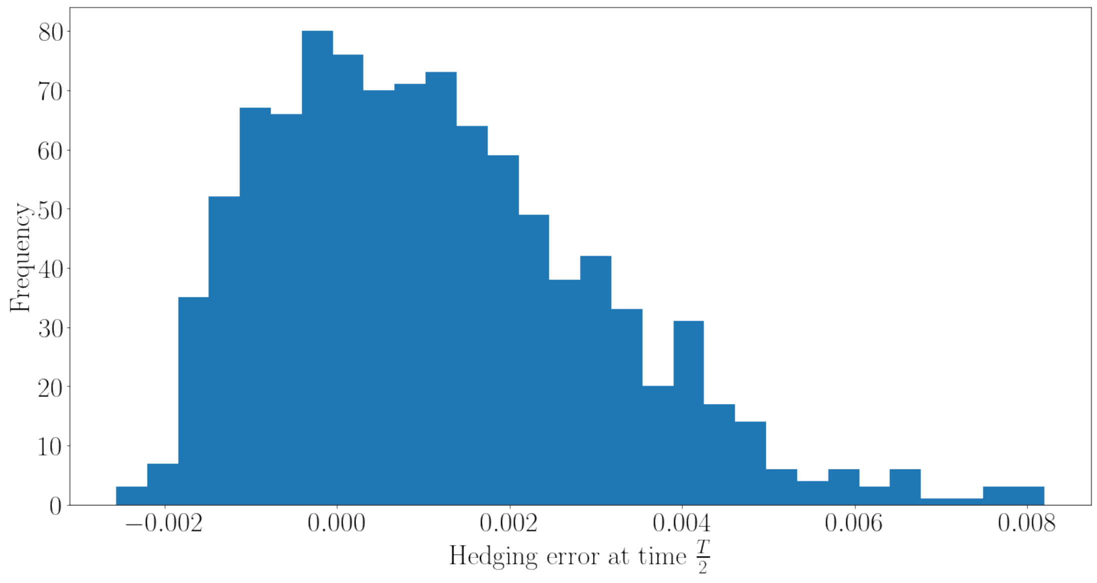
```
[ ]: <matplotlib.legend.Legend at 0x1c82afff700>
```



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 $\text{BS}^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ 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 $\text{BS}^{\text{put}}(S_t, K^H, T-t, \sigma(T), r)$. 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 $\text{BS}^{\text{call}}(S_0, K, T, \sigma(T), r)$ at time 0, and can then guarantee to have a portfolio with market value $\text{BS}^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ 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 $\text{BS}^{\text{call}}(S_{\frac{T}{2}}, K, \frac{T}{2}, \sigma(T), r)$ 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 $\text{BS}^{\text{put}}(S_t, K^H, T-t, \sigma(T), r)$ it is possible to replicate the payoff $\text{BS}^{\text{call}}(S_{T/2}, K, T/2, \sigma(T), r)$ at time $\frac{T}{2}$ for the price $\text{BS}^{\text{call}}(S_0, K, T, \sigma(T), r)$ 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}	