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:	
Module Code:	
Title of Project:	

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.	

Office Use Only: Date/Time Stamp	Agreed Mark

Delta Vs Gamma Hedging: A comparison under correct and incorrect volatility model assumptions

K23110623

July 8, 2024

Abstract

In this paper, we examine both delta and gamma hedging strategies for a European call stock option priced using the Black-Scholes equation. Employing Monte-Carlo techniques to explore and compare each hedging strategy. We focus on hedging when we assume correctly and incorrectly the true volatility model that describes the Geometric Brownian Motion of the underlying stock. The results of this exploration show the delta hedging strategy is a valid method of hedging options when the volatility model is assumed correctly but invalid when assumed incorrectly. Conversely, we find that gamma hedging can hedge options regardless of the volatility model's correctness. Additionally, it is observed that gamma hedging is a dominant strategy regarding accuracy, needing far fewer re-hedges to replicate the accuracy displayed by delta hedging. We also show that a delta hedging strategy can perform a hedge to maturity, something gamma hedging is unable to do due to instability when the option is At-The-Money close to maturity.

Note: This paper is not a true academic paper and is designed to mimic the structure of one, whilst answering the questions found in individual questions section of the appendix.

Table of Results

k-Number	k23110623
Price P	2.42409
Confidence Interval for p^{Δ}	[2.58884, 2.64835]
Confidence Interval for p^{Γ}	[0.00053, 0.00073]

1 Introduction

This paper aims to explore delta and gamma (also known as delta-gamma) hedging options priced under the Black-Scholes (BS) equation, examining scenarios for which volatility models are assumed correctly and incorrectly. This will be done through the use of Monte-Carlo methods, where we perform simulations for each of the four cases outlined in Table 1. The data obtained from each Monte-Carlo simulation is then used to realise the observations presented in this paper.

Note that we will only cover the hedging of a single European Call Option with strike price $K = 139$ and maturity $T = 1.60$ throughout this investigation. Additionally, the underlying stock of any option described will have initial value $S_0 = 105$ and a price process S_t that evolves according to Geometric Brownian Motion (GBM) as defined by either (1.1) or (1.2).

$$dS_t = \mu S_t dt + \sigma(t) S_t dW_t \tag{1.1}$$

$$dS_t = \mu S_t dt + \sigma(T) S_t dW_t \tag{1.2}$$

We denote t and its subscript to be time and the time indexing of a variable respectively, W_t to be Standard Brownian Motion, $\mu = 0.12$ to be drift and $\sigma(\cdot)$ to be volatility. S_t is either modeled with constant volatility $\sigma(T)$ or time dependent volatility $\sigma(t)$ as defined in (1.4) and (1.3) respectively. This will enable us to simulate hedging strategies across various volatility assumptions.

$$\sigma(t) = \begin{cases} 0.13 + 0.13 \times \frac{t-0.50}{T-0.50} & \text{if } t > 0.50 \\ 0.13 & \text{other} \end{cases} \tag{1.3}$$

$$\sigma(T) = 0.26 \tag{1.4}$$

The risk-free rate is taken to be $r = 0.04$. Therefore any bank account B_t will evolve with dynamics $dB_t = rB_t dt$, implying $B_{t+\delta t} = B_t e^{r\delta t}$.

Case 1:	We let the true stock price process follow (1.1) and perform a delta hedge of the call option assuming correctly that the volatility of S_t is described by (1.3)
Case 2:	We let the true stock price process follow (1.1) and perform a delta hedge of the call option assuming incorrectly that the volatility of S_t is described by (1.4)
Case 3:	We let the true stock price process follow (1.2) and perform a gamma hedge of the call option assuming correctly that the volatility of S_t is described by (1.4)
Case 4:	We let the true stock price process follow (1.1) and perform a gamma hedge of the call option assuming incorrectly that the volatility of S_t is described by (1.4)

Table 1: Investigation Outline

The rest of the paper will cover the theory (Section 2) and methodology (Section 3) of each Monte-Carlo simulation, then present observational findings (Section 4), and conclude with a summary of these observations (Section 5).

2 Theory

2.1 Simulating Geometric Brownian Motion

Although there are several valid approximation methods for simulating GBM from its dynamics equation, we acknowledge that directly simulating GBM from its solution is not an approximation and thus should be the preferred method of simulation when possible [Arm21].

To obtain the explicit solution of the Itô diffusion equation $dS_t = \mu S_t dt + \sigma S_t dW_t$ that defines GBM, we begin by examining the function $f(S_t, t) = \ln(S_t)$. by applying Itô's formula to $f(S_t, t)$ we arrive at the subsequent dynamic equation

$$df(S_t, t) = \Big(\mu - \frac{1}{2}\sigma^2\Big)dt + \sigma dW_t$$

which can then be used to solve $f(S_t, t)$ by integration

$$f(S_t, t) = f(S_0, 0) + \int_0^t \mu - \frac{1}{2}\sigma^2 dt + \int_0^t \sigma dW_t \tag{2.1}$$

Obtaining the explicit solution of our original SDE now becomes a matter of substituting (2.1) into the equality $f(S_t, t) = \ln(S_t)$ resulting in

$$S_t = S_0 e^{\int_0^t \mu - \frac{1}{2}\sigma^2 dt + \int_0^t \sigma dW_t} \tag{2.2}$$

To simulate S_t using (2.2), evaluate $f(S_{t+\delta t}, t) - f(S_t, t)$ by reconsidering (2.1)

$$f(S_{t+\delta t}, t) - f(S_t, t) = \int_t^{t+\delta t} \mu - \frac{1}{2}\sigma^2 dt + \int_t^{t+\delta t} \sigma dW_t \tag{2.3}$$

Notice that the RHS of (2.3), which we denote as $X_{t+\delta t}$, contains a deterministic term and a stochastic integral. Assuming that the stochastic integral in $X_{t+\delta t}$ converges in the $\mathcal{L}^2$-norm [Shr11] we can deduce that $X_{t+\delta t}$ is normally distributed using the properties of Brownian motion [RJ22]. S_t can therefore be simulated by generating samples of the normally distributed random variable $X_{t+\delta t}$ [Arm21]. This can be done by identifying the distribution parameters of $X_{t+\delta t}$, which can be done by utilizing Itô Isometry to compute the conditional expectation and variance given the filtration $\mathcal{F}_t$ as shown below [RJ22].

$$\begin{cases} \mathbb{E}\left[f(S_{t+\delta t}, t) - f(S_t, t) \,|\, \mathcal{F}_t\right] = \mathbb{E}\left[X_{t+\delta t} \,|\, \mathcal{F}_t\right] = \int_t^{t+\delta t} \mu - \frac{1}{2}\sigma^2 dt \\ \mathrm{Var}\left[f(S_{t+\delta t}, t) - f(S_t, t) \,|\, \mathcal{F}_t\right] = \mathrm{Var}\left[X_{t+\delta t} \,|\, \mathcal{F}_t\right] = \int_t^{t+\delta t} \sigma^2 dt \end{cases}$$

Since $X_{t+\delta t}$ represents the change in the log of the stock price and $X_{t+\delta t} \sim N\Big(\int_t^{t+\delta t} \mu - \frac{1}{2}\sigma^2 dt\,, \int_t^{t+\delta t} \sigma^2 dt\Big)$, it follows that we can discreetly simulate S_t by using our explicit solution (2.2) in a recursive manner (2.4), after sampling random values of $X_{t+\delta t}$ [Arm21].

$$S_{t+\delta t} = S_t e^{X_{t+\delta t}} \tag{2.4}$$

2.2 Option Pricing: Black-Scholes-Merton

Pricing options using the Black-Scholes PDE (2.5), where V_t denotes the price of an option, can be done through the use of the Feynman-Kac Theorem [Arm21]. Stating that the risk-neutral solution to (2.5) is the conditional expectation of the discounted payoff of the option [Shr11].

$$\begin{cases} \frac{\partial V_t}{\partial t} + \frac{1}{2}\sigma^2 S_t^2 \frac{\partial^2 V_t}{\partial S_t^2} + r S_t \frac{\partial V_t}{\partial S_t} - r V_t = 0 \\ V_T = G(S_T) \end{cases} \tag{2.5}$$

That is $V_t = \mathbb{E}^{\mathbb{Q}^*}[e^{-r(T-t)} G(S_T) \,|\, \mathcal{F}_t]$ solves (2.5), where $G(\cdot)$ is the payoff function of the option at maturity. Computing the conditional expectation to find V_t can be done through the use of Girsanov's Theorem to describe W_t under the new risk-free probability measure $\mathbb{Q}^*$. As a result, we can derive the following BS equations for call and put options [Shr11].

$$C_t^{\mathrm{BS}}(S_t, K, T, \sigma, r) = N(d_1) S_t - N(d_2) K e^{-r(T-t)} \tag{2.6}$$

$$P_t^{\mathrm{BS}}(S_t, K, T, \sigma, r) = N(-d_2) K e^{-r(T-t)} - N(-d_1) S_t \tag{2.7}$$

where N is the CDF of a standard normal distribution and d_1, d_2 are

$$d_1 = \frac{\ln(\frac{S_t}{K}) + (r + \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}} \qquad d_2 = \frac{\ln(\frac{S_t}{K}) + (r - \frac{1}{2}\sigma^2)(T-t)}{\sigma\sqrt{T-t}}$$

Note the set of BS equations above where the implied volatility σ is taken to be the volatility of the GBM, works only for valuing options whose underlying follows GBM with constant drift and volatility, such as (1.2). We therefore cannot implement (2.6), (2.7) in this form to value options linked to GBM with constant drift and time-dependent volatility, such as (1.1). This however can be resolved by making appropriate adjustments to the implied volatility of the BS equation to ensure accurate pricing in such instances.

Examine again $V_t = \mathbb{E}^{\mathbb{Q}^*}[e^{-r(T-t)} G(S_T) | \mathcal{F}_t]$, where we observe that any deviations from the original BS equation will stem from variations in S_T caused by different GBM's. Consequently, we consider S_T for (1.2).

$$S_T = S_t \frac{S_T}{S_t} = S_t e^{(\mu - \frac{1}{2}\sigma(T)^2)(T-t) + \sigma(T)\sqrt{T-t}Z} \tag{2.8}$$

and then for (1.1)

$$S_T = S_t \frac{S_T}{S_t} = S_t e^{\int_t^T \mu - \frac{1}{2}\sigma^2 dt + \sqrt{\int_t^T \sigma(t)^2 dt} Z}$$
$$= S_t e^{(\mu - \frac{1}{2}s(t)^2)(T-t) + s(t)\sqrt{T-t}Z} \tag{2.9}$$

We see that by choosing $s(t)$ as defined in (2.10), s_t in (2.9) becomes synonyms with $\sigma(T)$ in (2.8). Therefore by evaluating V_t using (2.9), we gain a new set of BS equations for GBM with constant drift and time-dependent volatility.

$$s(t)^2 = \frac{1}{T-t}\int_t^T \sigma(t)^2 dt \tag{2.10}$$

These equations are of the form (2.6) and (2.7) but with the adjustment of taking the implied volatility as $\sigma = s(t)$.

2.3 Option Greeks: Delta & Gamma

The delta of an option is the sensitivity of its value to changes in the underlying asset value [YX13]. Similarly, the gamma of an option is the sensitivity of its delta to changes in the underlying asset value. Mathematically, the delta and gamma of a stock option are defined as the first and second-order derivative with respect to the underlying stock price, as shown in (2.11).

$$\Delta = \frac{\partial}{\partial S} \qquad \Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2}{\partial S^2} \tag{2.11}$$

Since the value of a stock option can be described by the BS equations which are continuous and twice differentiable, we can compute the delta and gamma of a stock option [YX13] to be

$$\Delta(C_t^{BS}) = N(d_1) \qquad \Delta(P_t^{BS}) = -N(-d_1) \tag{2.12}$$

$$\Gamma(C_t^{BS}) = \frac{n(d_1)}{S_t\sigma\sqrt{T-t}} \qquad \Gamma(P_t^{BS}) = \frac{n(d_1)}{S_t\sigma\sqrt{T-t}} \tag{2.13}$$

where n is the PDF of the standard normal distribution.

2.4 Hedging & Replicating Portfolios Strategies

Hedging strategies attempt to minimise exposure to certain risks incurred by holding an asset over time [CFI24]. Delta and gamma hedging, as their names imply, aim to reduce the risk associated with the delta and gamma of a security (or set of securities) when implemented into a portfolio. This can be done by the construction of a dynamic self-financing replicating portfolio as discussed in the following subsections.

2.4.1 Replicating Portfolio: Delta Strategy

The delta hedging strategy utilizes a replicating portfolio constructed with a bank account and the underlying stock of the call option being replicated [Arm21]. Let us denote this portfolio by $(B_t,\, q_t^s)$, where B_t is the amount of cash invested into the bank account and q_t^s is the quantity of stock bought at time t. By construction, the value Π_t of portfolio is therefore

$$\Pi_t = B_t + q_t^S S_t \tag{2.14}$$

The aim of this portfolio is to replicate both the value and delta of the call option at all times i.e.

$$\Delta(\Pi_t) = \Delta(C_t) \qquad\qquad \Pi_t = C_t \tag{2.15}$$

where C_t is the value of the call option. Since continuous trading is not possible, we settle to achieve (2.15) periodically (every δt), which consequently gives this strategy its dynamic structure [Arm21]. This can done by adjusting the quantity of each asset in the portfolio after a period of δt has elapsed such that (2.15) is re-satisfied. In the case of a delta replicating portfolio we can use (2.14), (2.15) to solve for q_t^S, B_t, as shown below

$$\begin{aligned} \Delta(C_t) = \Delta(\Pi_t) = \Delta(B_t + q_t^S S_t) = q_t^S \\ \Rightarrow q_t^S = \Delta(C_t) \end{aligned} \tag{2.16}$$

$$\begin{aligned} \Pi_t = B_t + q_t^S S_t \\ \Rightarrow B_t = \Pi_t - q_t^S S_t \end{aligned} \tag{2.17}$$

where $\Delta(V_t)$ can be approximated by $\Delta(V_t^{\mathrm{BS}})$.

After investing into the portfolio $(q_t^S,\, B_t)$ at time t and waiting a period of δt, the new value of our portfolio $\Pi_{t+\delta t}$ becomes

$$\Pi_{t+\delta t} = B_t e^{r(\delta t)} + q_t^S S_{t+\delta t} \tag{2.18}$$

by standard evolution of a bank account, as described in Section 1. The portfolio $(q_t^S,\, B_t)$ at time $t + \delta t$ can then be liquidated and readjusted by investing the resulting cash into the portfolio $(q_{t+\delta t}^S,\, B_{t+\delta t})$. The process of liquidating and readjusting after each time step δt can be repeated on the time discrete grid $\mathcal{T} = \{0, \delta t, 2\delta t, 3\delta t, \dots\}$ until a desired point in time or until the maturity T of the Call option being replicated [Arm21]. Since we do not need to inject any additional cash into this portfolio, we call this strategy self-financing [Shr05]. Note that the initial cash injection needed to execute this strategy is the price of the Call Option at time 0 which we take to be $\Pi_0 = C_0^{\mathrm{BS}}$ [Arm21].

2.4.2 Replicating Portfolio: Gamma Strategy

The replicating portfolio used in the gamma hedging strategy is constructed in a similar fashion to that of the delta hedging strategy. The difference is the addition of an asset that has a non-zero gamma [Arm22]. We choose this asset to be a continuously tradable European put option whose payoff depends on the same underlying stock as that of the call option we are replicating. Let (B_t, q_t^S, q_t^P) denote the gamma replicating portfolio, where q_t^P is the quantity of put options held in the portfolio. Then the value of our portfolio Π_t is

$$\Pi_t = B_t + q_t^S S_t + q_t^P P_t \tag{2.19}$$

The goal of this portfolio is to replicate both the delta and gamma of the call option as well as its value periodically.

$$\Delta(\Pi_t) = \Delta(V_t) \qquad \Gamma(\Pi_t) = \Gamma(V_t) \qquad \Pi_t = V_t \tag{2.20}$$

Therefore by using (2.19, 2.20) we find B_t, q_t^S, q_t^P to be the following

$$\Gamma(V_t) = \Gamma(\Pi_t) = \Gamma(B_t + q_t^S S_t + q_t^P P_t) = q_t^P \, \Gamma(P_t)$$

$$\Rightarrow q_t^P = \frac{\Gamma(V_t)}{\Gamma(P_t)} \tag{2.21}$$

$$\Delta(V_t) = \Delta(\Pi_t) = \Delta(B_t + q_t^S S_t + q_t^P P_t) = q_t^S + q_t^P \, \Delta(P_t)$$

$$\Rightarrow q_t^S = \Delta(V_t) - q_t^P \, \Delta(P_t) \tag{2.22}$$

$$V_t = \Pi_t = B_t + q_t^S S_t + q_t^P P_t$$

$$\Rightarrow B_t = \Pi_t - q_t^S S_t + q_t^P P_t \tag{2.23}$$

where $\Delta(P_t)$, $\Gamma(V_t)$, $\Gamma(P_t)$ can be approximated by $\Delta(P_t^{\text{BS}})$, $\Gamma(V_t^{\text{BS}})$, $\Gamma(P_t^{\text{BS}})$, respectively.

After investing into the portfolio (B_t, q_t^S, q_t^P) at time t we have that its value at $t + \delta t$ becomes

$$\Pi_{t+\delta t} = B_t e^{r(\delta t)} + q_t^S S_{t+\delta t} + q_t^P P_{t+\delta t} \tag{2.24}$$

which we liquidate and readjust, by investing the resulting cash into the portfolio $(B_{t+\delta t}, q_{t+\delta t}^S, q_{t+\delta t}^P)$. This process is iterated for each time step δt up to or before the maturity T of the replicated call option [Arm22]. The initial cash injection is again the BS value of the call option at time 0.

2.4.3 Hedging strategy

The delta and gamma hedging strategy written as a step-by-step process can be described as: (i) At time 0 long the call option C_0^{BS} and short the replicating portfolio (B_0, q_0^S) if delta hedging and (B_0, q_0^S, q_0^P) if gamma hedging. (ii) Execute the respective replicating portfolio strategy for each time step δt in a hedged manner. That is we take the negative of the replicating portfolio such that we are negatively replicating the value of the option and its delta or delta and Gamma depending on the strategy, this is also called a re-hedging. (iii) At the maturity T of the call option we can realize the payoff of the option and liquidate the hedged replicating portfolio to realise the profit and loss (PnL) of the strategy. Alternatively, we can liquidate the total portfolio (long call option + short replicating portfolio) at any time step before the maturity of the option T and realize the PnL of the strategy this way.

3 Method

Each Monte-Carlo simulation performed as part of this exploration will contain 1000 simulations. The Monte-Carlo simulations will begin by choosing the length of the hedging strategy (hedge to half the maturity $\frac{T}{2}$ or hedge to maturity T) and the number of re-hedges performed in the chosen hedging period. When Gamma hedging we will only perform a hedge to half the maturity to avoid the instability of the hedging strategy when At-The-Money is close to maturity. The time step δt consequently becomes either $\frac{T/2}{n}$ or $\frac{T}{n}$ depending on the hedging length. A case from Table 1 is then chosen to perform our Monte-Carlo simulation.

Where we simulate 1000 stock price paths according to Section 2.1) and perform the respective hedging strategy for each stock price process following Sections 2.2, 2.3 and 2.4). The profit and loss (PnL or error) can then be realised by exiting our position at the end of the hedging length chosen. The PnL data collected for each Monte-Carlo simulation we carry out is collected and analysed to form results covered in Section 4.

4 Results

Following the PnL data produced from case 2 performed to half maturity $\frac{T}{2}$ we find that delta hedging under an incorrect volatility assumption yields profit indefinitely. This can be further supported by the 95% confidence interval (CI_{95}) of the true PnL being [2.5888, 2.6484]. Case 1 performed to maturity T and cases 3, 4 performed to half maturity $\frac{T}{2}$ instead show

promising CI_{95} that are [-0.022, 0.011], [-0.0001, 0.0002] and [0.0005, 0.0007] respectively. One thing to note is that the last CI_{95} does not contain zero although being significantly close suggesting its true PnL may not be zero unlike that of the other two cases. In fact, the PnL distribution for case 4 is heavily skewed (Figure 2), unlike Cases 1 and 3 which are rather symmetrical in distribution (Figure 4 and 5).

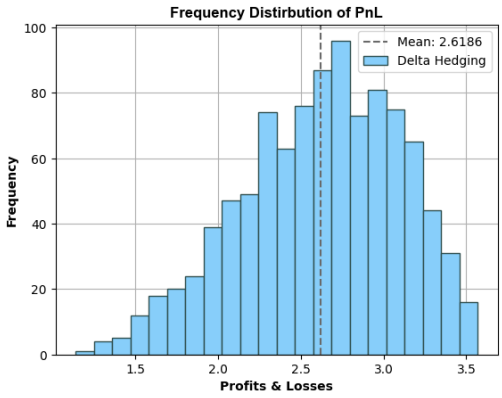


Figure 1: hedge to half maturity, case 2 with 1000 re-hedges

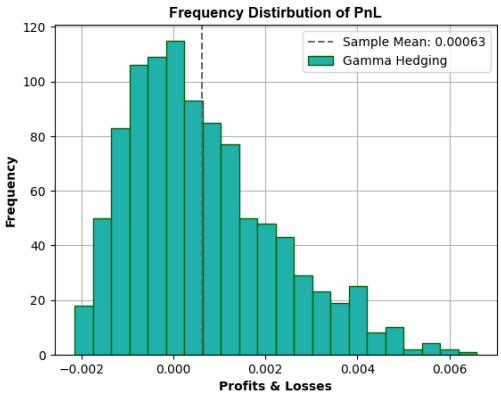


Figure 2: hedge to half maturity, case 4 with 1000 re-hedges

Figure 3: hedge to half maturity, a case comparison of log-log RMS error Vs re-hedges

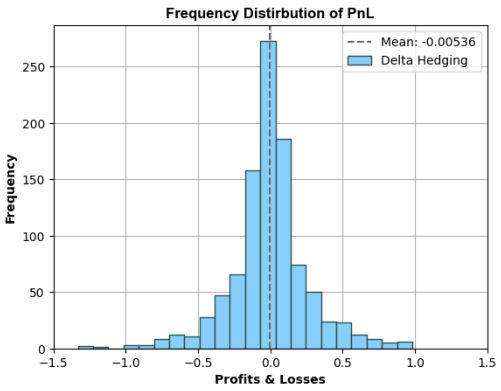


Figure 4: hedge to maturity, case 1 with 2000 re-hedges

We take the analysis of cases 1, 2 and 3 one step further by simulating the convergence of RMS error [Arm21] as we increase the number of re-hedges. Plotting the log-log of RMS error Vs re-hedges (Figure 3) to find the order of convergence we discover an order of convergence approximately $\mathcal{O}(n^{-\frac{1}{2}})$ for case 1 to maturity and half maturity and approximately $\mathcal{O}(n^{-1})$ for cases 3 & 4 to half maturity. This result, suggests that hedging in cases

1, 3 & 4 is possible with sufficient re-hedging. Additionally, we can infer that Gamma hedging has far better accuracy than Delta hedging, which we reiterate graphically in Figure 5 and 6 showing the need for far fewer re-hedges.

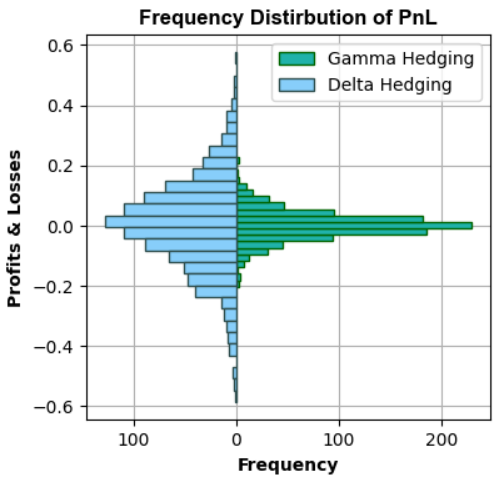


Figure 5: hedge to half maturity, gamma hedging case 3 with 50 re-hedges and delta hedging case 3 with 1000 re-hedges

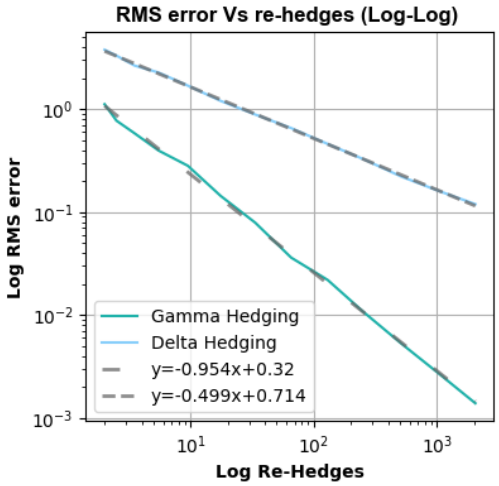


Figure 6: hedge to half maturity, Delta & Gamma case 3 log-log RMS error comparison for 1000 re-hedges

Lastly, we observe unlike gamma hedging it is possible to delta hedge to maturity as shown in Figure 4.

5 Conclusion

In this paper, we have shown that gamma hedging can hedge European call options to half their maturity with a superior accuracy compared to delta hedging. In fact, gamma hedging needs far less re-hedging to obtain equal or greater accuracy than delta hedging. We also observe that gamma hedging can hedge regardless of any assumptions made for the volatility model of the underpinning GBM (i.e. for both cases 3 and 4). This is found to not be the case for delta hedging, which is only valid when volatility is assumed correctly (case 1) and not when assumed incorrectly (Case 2). We finally show that delta hedging does not suffer from the same problems as gamma and can a hedge to maturity perfectly fine given enough re-hedges.

References

[Arm21] John Armstrong. Lecture notes for 7CCMFM06, Computational and Numerical Methods in Mathematical Finance. https://keats.kcl.ac.uk/course/view.php?id=93365, 2021.

[Arm22] John Armstrong. Delta hedging. https://nms.kcl.ac.uk/john.armstrong/courses/fm06/book/matlab-chapter6.pdf, 2022.

[CFI24] CFI. Hedging. https://corporatefinanceinstitute.com/resources/derivatives/hedging/, 2024.

[RJ22] Ashwin Rao and Tikhon Jelvis. Foundations of reinforcement learning with applications in finance. https://stanford.edu/~ashlearn/RLForFinanceBook/appendix3.pdf, 2022.

[Shr05] Steven E. Shreve. *Stochastic calculus for finance I the binomial asset pricing model*. Springer, 2005.

[Shr11] Steven E. Shreve. *Stochastic calculus for finance II*. Springer, 2011.

[YX13] Xisheng Yu and Xiaoke Xie. On derivations of black-scholes greek letters. *Research Journal of Finance and Accounting*, 4(6), 2013.

A Jupyter Notebook

Kernel: Python 3 (system-wide)

Notebook

7CCFM06 Coursework

Setup

The following cells in this section of the notebook contain the nessessary Classes and Functions needed to complete Questions 1, 2, 3 and 4 in the attched problem sheet. Each Class and Function will have a coresponding test function which aim to flag up any anomolies that may occur due to incorrect code.

Import Libarys

Libarys

1. scipy Libary
2. numpy Libary
3. matplotlib.pyplot Libary
4. seaborn Libary
5. warnings Libary

In [3]:
```
### Import Libarys ###

## libarys ##
import scipy as sp
import scipy.stats as stats
from scipy import integrate
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

Random Seed

In [4]:
```
### random seed

## set random seed (# to remove random seed)
np.random.seed(2211)
```

Market Model

Market Model

Classes & Functions

1. market_model() - This Class defines the market parameters of our experiment. That is the risk free rate r, inital stock price S_0, dirft of the stock μ and the volatility function $\sigma(t)$. The class also has embeded functions that compute the mean and variance of the log returns of a stock i.e. $\mathbb{E}(\log \frac{S_t}{S_s})$ and $\text{var}(\log \frac{S_t}{S_s})$ as well as Black-Scholes Implied volaility squared s_t^2 of GBM with time dependent volatility.

2. sigma(self, t) - This is an embeded function in the Calss market_model() that outputs the GBM volatility of the stock at time t

3. mean(self, s, t) - This is an embeded function in the Class market_model() that outputs the mean of the log returns i.e. $\mathbb{E}[\log(\frac{S_t}{S_s})]$

4. var(self, s ,t) - This is an embeded function in the Class market_model() that outputs the variance of the log returns of $\text{var}[\log(\frac{S_t}{S_s})]$

5. s2(self, t) - This is an embeded function in the Class markret_model() that outputs the Black-Scholes implied volaility squared s_t^2 of GBM with time dependent volatility.

In [5]:
```
### Market Model ###

## The Market Model
class market_model():

    # fixed market parameters
    S0 = 105
    mu = 0.12
    r = 0.04

    # max time maturity
    T_max = 1.60

    ## market volaitlity
    # t: (float) input time
    def sigma(self, t):

        if t > 0.5:
            return 0.13 + 0.13 * ((t-0.5) / (self.T_max - 0.5))
        else:
            return 0.13

    ## mean of log returns
    # s, t (float) input time interval such that s < t
    def mean(self, s, t):

        # initalize variables
        nu = self.mu
        T_max = self.T_max

        # integral of f(t) = mean of log returns between the time
interval of the intergal
        # define f(x) depending on integral bounds
```

```
        def f1(x):
            return nu - 0.5 * (0.13 + 0.13 * ((x - 0.5) / (T_max -
0.5)))**2
        def f2(x):
            return nu - 0.5 * (0.13**2)

        # mean value if t > 0.5 and s >= 0.5
        if t > 0.5 and s >= 0.5:
            return integrate.quad(f1, s, t)[0]

        # mean value if t > 0.5 and s < 0.5
        if t > 0.5 and s < 0.5:
            return integrate.quad(f1, 0.5, t)[0] + integrate.quad(f2,
s, 0.5)[0]

        # mean value if t <= 0.5 and s < 0.5
        if t <= 0.5 and s < 0.5:
            return integrate.quad(f2, s, t)[0]

    ## variance of log returns
    # s, t (float) input time interval such that s < t
    def var(self, s ,t):

        # initalize variables
        T_max = self.T_max

        ## integral of g(x) = variance of log returns between the time
interval of the integral
        # define g(x) depending on integral bounds
        def g1(x):
            return (0.13 + 0.13 * ((x-0.5) / (T_max - 0.5)))**2
        def g2(x):
            return 0.13**2

        # var value if t > 0.5 and s >= 0.5
        if t > 0.5 and s >= 0.5:
            return integrate.quad(g1, s, t)[0]

        # var value if t > 0.5 and s < 0.5
        if t > 0.5 and s < 0.5:
            return integrate.quad(g1, 0.5, t)[0] + integrate.quad(g2,
s, 0.5)[0]

        # var value if t <= 0.5 and s < 0.5
        if t <= 0.5 and s < 0.5:
            return integrate.quad(g2, s, t)[0]

    ## s(t)^2
    # t (float) input time
    def s2(self, t):

        # initalize variables
        T_max = self.T_max

        # initalize functions
        variance = self.var

        return (1/(T_max - t)) * variance(t, T_max)
```

Tests

1. test_market_model() - This test double checks variables pre-defined in the Class and checks that the functions sigma(), mean(), var(), s2() output the correct values comparing the output to externaly varified values.

In [6]:
```
### Market model (testing) ###

## test market_model()
def test_market_model():

    # test market paramaters
    assert market_model().r == 0.04, "incorrect risk-free rate"
    assert market_model().mu == 0.12, "incorrect drift"
    assert market_model().T_max == 1.60, "incorrect max time frame"
    assert market_model().S0 == 105, "incorrect inital stock price"

    # test sigma()
    assert market_model().sigma(0.4) == 0.13, "error, sigma for t <=
0.5"
    assert round(market_model().sigma(0.8), 3) == 0.165, "error, sigma
for t > 0.5"

    # test mean()
    assert round(market_model().mean(0.1, 0.3), 3) == 0.022, "error,
mean for s, t < 0.5"
    assert round(market_model().mean(0.4, 0.7), 3) == 0.011 + 0.022,
"error, mean for s < 0.5 and t >= 0.5"
    assert round(market_model().mean(1, 1.3), 3) == 0.030, "error,
mean for s, t >= 0.5"

    # test var()
    assert round(market_model().var(0.3, 0.4), 4) == 0.0017, "error,
variance for s, t < 0.5"
    assert round(market_model().var(0.2, 0.6), 4) == 0.0051 + 0.0018,
"error, variance for s < 0.5 and t >= 0.5"
    assert round(market_model().var(0.7, 0.9), 4) == 0.0055, "error,
variance for s, t >= 0.5"

    # test s2()
    assert round(market_model().s2(0.2), 4) == 0.0346, "error, s2 for
t < 0.5"
    assert round(market_model().s2(0.5), 4) == 0.0394, "error, s2 for
t = 0.5"
    assert round(market_model().s2(1.5), 4) == 0.0646, "error, s2 for
t > 0.5"

    # return
    return print("Test Passed :)")

# run test
test_market_model()
```

Out[6]:
```
Test Passed :)
```

Option Models

Classes & Functions

1. option() - This Class defines a call option with strike price 139 and maturity 1.6 along with its payoff function

2. option_H() - This Class defines a put option with strike price 105 and maturity 1.6 along with its payoff function

In [7]:
```
### Options ###

# Put Option with strike 139
class option():

    # option parameters
    T = 1.60
    K = 139

    ## call option payoff
    # ST: (float - vector/variable) input stock price at maturity
    def payoff_call(self, ST):

        # return payoff with stike 139
        return np.maximum(ST - self.K, 0)

    ## put option payoff
    # ST: (float - vector/variable) input stock price at maturity
    def payoff_put(self, ST):

        # return payoff with strike 139
        return np.maximum(self.K - ST, 0)

# Option with strike S0
class option_H():

    # option parameters
    T = 1.60
    K = 105

    ## call option payoff
    # ST: (float - vector/variable) input stock price at maturity
    def payoff_call(self, ST):

        # return payoff with stike S0
        return np.maximum(ST - self.K, 0)

    ## put option payoff
    # ST: (float - vector/variable) input stock price at maturity
    def payoff_put(self, ST):

        # return payoff with strike S0
        return np.maximum(self.K - ST, 0)
```

Tests

1. test_option() - This test checks that the pre defined variables are set to the correct values as well as checking that the payoff functions defined in the class output the correct values for inputs above and below the strike price $K = 139$

2. test_option_H() - This test checks that the pre defined variables are set to the correct values as well as checking that the payoff functions defined in the class output the correct values for inputs above and below the strike price $K = 105$

In [8]:
```
### Options (testing) ###

## test option()
def test_option():

    # test paramaters
    assert option().T == 1.6, "incorrect option maturity"
    assert option().K == 139, "incorrect option strike price"

    # test payoff functions
    assert option().payoff_call(178) == 178-139, "error, call option
payoff"
    assert option().payoff_call(74) == 0, "error, call option payoff"
    assert option().payoff_put(152) == 0, "error, put option payoff"
    assert option().payoff_put(84) == 139-84, "error, put option
payoff"

    # return
    return print("Test Passed :)")

# run test
test_option()

## test option_H()
def test_option_H():

    # test paramaters
    assert option_H().T == 1.6, "incorrect option_H maturity"
    assert option_H().K == 105, "incorrect option_H strike price"

    # test payoff functions
    assert option_H().payoff_call(149) == 149-105, "error, call
option_H payoff"
    assert option_H().payoff_call(96) == 0, "error, call option_H
payoff"
    assert option_H().payoff_put(200) == 0, "error, put option_H
payoff"
    assert option_H().payoff_put(60) == 105-60, "error, put option_H
payoff"

    # return
    return print("Test Passed :)")

# run test
test_option_H()
```

Out[8]:
```
Test Passed :)
Test Passed :)
```

Simulating Geometric Brownian Motion

Functions

1. simulate_GBM(model, method, N , n, vol) - This function simualtes a stocks price path on a discrete time grid for $n+1$ time steps for N simulations (price paths). This function uses either the "eplicit" or "euler" method to simulate multiple stock price processes and takes into account both "fixed" (constant) or "variable" (time dependent) volaitlity for the GBM of the stock. The spesifics of the GBM of the stock come from the class market_model(). The output of the function is a $[N, n+1]$ matrix with the rows representing different stock price paths and columns the price of a stock for a given time step.

In [9]:

```
### Geometric Brownian Motion ###

## simulate Geormetric Brownian Motion for [ dSt = mu*St*dt +
sigma(t)*St*dWt ]
# model: (class) input market model name followed by ()
# method: (string) input "explicit" for simulation using the explicit
solution of the ODE or "euler" for simulation using euler maruyama
scheme
# N: (int) input the number of simulations of GBM performed
# n: (int) input the number of time steps performed
def simulate_GBM(model, method, N , n, vol):

    # initalise variables
    S0 = model.S0
    mu = model.mu
    T = model.T_max

    # find delta t and times
    dt = T/n

    # create empty matrix
    S = np.zeros([N, n+1])

    # set inital prices
    S[:,0] = S0

    # simulate using the explicit solution of the SDE
    if method == "explicit":

        for i in range(0, n):

            # simulating GBM with a time dependent volaitlity sigma(t)
            if vol == "variable":
                mean = model.mean(i*dt, (i+1)*dt)
                var = model.var(i*dt, (i+1)*dt)
                sd = np.sqrt(var)
                epsilon_alt = np.random.normal(mean, sd, N)
                S[:,i+1] = S[:, i]*np.exp(epsilon_alt)

            # simulating GBM with a time dependent volaitlity sigma(T)
            if vol == ("fixed"):
                epsilon = np.random.normal(0, 1, N)
                S[:,i+1] = S[:, i]*np.exp((mu -
0.5*model.sigma(T)**2)*dt + model.sigma(T)*np.sqrt(dt)*epsilon)

        return S

    # simulate using the euler maruyama scheme
    if method == "euler":
```

```
        for i in range(0, n):

            # simulating GBM with a time dependent volaitlity sigma(t)
            if vol == "variable":
                epsilon = np.random.normal(0, 1, N)
                S[:,i+1] = S[:, i]*(1 + mu*dt +
model.sigma(i*dt)*np.sqrt(dt)*epsilon)

            # simulating GBM with a time dependent volaitlity sigma(T)
            if vol == "fixed":
                epsilon = np.random.normal(0, 1, N)
                S[:,i+1] = S[:, i]*(1 + mu*dt +
model.sigma(T)*np.sqrt(dt)*epsilon)

        return S
```

Tests

1. test_simulate_GBM() - This test takes advantage of the fact that the log returns of the GBM stock proccess should be normaly distirbuted with known mean and variance. The takes the ouput of the function simulate_GBM() and conputs the log difference of two consecuative rows, then checks its mean, variance and normality. The mean and variance can be externaly calculated and checked through means of comparison, as for the normality this is checked though the use of shapiro normality test.

In [10]:

```
### Geometric Brownian Motion (Testing) ###

## test simulate_GBM()
def test_simulate_GBM():

    # set dt
    dt = 1.6/10

    # fixed volatility

    # create a sample of the mean and variance of a sample of log
returns between a time interval (fixed volatility)
    sample_mean = []
    sample_var = []
    for i in range(5000):
        S = simulate_GBM(market_model(), "explicit", 10, 10, "fixed")
        log_return = np.log(S[:, 9]) - np.log(S[:,8])
        sample_mean.append(np.mean(log_return))
        sample_var.append(np.var(log_return))

    # test mean of sample mean (fixed volatility)
    assert round(np.mean(sample_mean), 2) == round((0.12 - 0.5*
(0.26**2))*dt, 2), "incorrect mean of log returns (fixed)"

    # test mean of sample variance (fixed volatility)
    assert round(np.mean(sample_var), 2) == round((0.26**2)*dt, 2),
"incorrect variance of log returns (fixed)"

    # Create a sample of log returns (fixed volatility)
    S = simulate_GBM(market_model(), "explicit", 1000, 10, "fixed")
    log_return = np.log(S[:, 5]) - np.log(S[:,4])
```

```
    # test normaility of log returns (fixed volatility)
    assert stats.shapiro(log_return).pvalue > 0.05, "error, log
returns not normal (fixed)"

    # variable volatility

    # create a sample of the mean and variance of a sample of log
returns between a time interval (variable volatility)
    sample_mean = []
    sample_var = []
    for i in range(5000):
        S = simulate_GBM(market_model(), "explicit", 10, 10,
"variable")
        log_return = np.log(S[:, 3]) - np.log(S[:,2])
        sample_mean.append(np.mean(log_return))
        sample_var.append(np.var(log_return))


    # test mean of sample mean (variable volatility)
    assert round(np.mean(sample_mean), 2) ==
round(market_model().mean(2*dt, 3*dt), 2), "incorrect sample means for
log returns (variable)"

    # test mean of sample variance ((variable volatility)
    assert round(np.mean(sample_var), 2) ==
round(market_model().var(2*dt, 3*dt), 2), "incorrect sample means for
log returns (variable)"

    # Create a sample of log returns (variable volatility)
    S = simulate_GBM(market_model(), "explicit", 1000, 10, "variable")
    log_return = np.log(S[:, 3]) - np.log(S[:,2])

    # test normaility of log returns (variable volatility)
    assert stats.shapiro(log_return).pvalue > 0.05, "error, log
returns not normal (fixed)"

    # return
    return print("Test Passed :)")

# run test
test_simulate_GBM()
```

Out[10]: Test Passed :)

Black-Scholes

Functions

1. N(z) - This function outputs the CDF value of a standard normal distirbution
2. n(z) - This function outputs the PDF value of a standard normal distirbution
3. compute_d1_d2(model, option, St, t, vol) - This function out puts the d1 and d2 values of the Black-Scholes equation using the inputs $(S_t, K, r, \sigma, \tau)$

4. BS_call(model, option, St, t, vol) - This function outputs the Black-Scholes value for a call option using inputs $(S_t, K, r, \sigma, \tau)$

5. BS_put(model, option, St, t, vol) - This function outputs the Black-Scholes value for a Put option using inputs $(S_t, K, r, \sigma, \tau)$

6. BS_delta(model, option, St, t, vol, CorP) - This function outputs the delta of an option (Put or Call) using the Black-Schoels equation and has inputs This function outputs the Black-Scholes value for a call option using inputs $(S_t, K, r, \sigma, \tau, \mathrm{CorP})$

7. BS_gamma(model, option, St, t, vol, CorP) - This function outputs the gamma of an option (Put or Call) using the Black-Schoels equation and has inputs This function outputs the Black-Scholes value for a call option using inputs $(S_t, K, r, \sigma, \tau, \mathrm{CorP})$

In [11]:

```
### Black-Scholes ###

## standard normal cdf
# z: (float) input x such that P(Z<z) = N(z) where Z ~ N(0, 1)
def N(z):
    return sp.stats.norm.cdf(z)

## standard normal pdf
# z: (float) input x such that P(Z<z) = N(z) where Z ~ N(0, 1)
def n(z):
        return sp.stats.norm.pdf(z)

## d1 and d2 values for black-scholes equation
# model: (class) input market model name followed by ()
# options: (class) input option model name followed by ()
# st: (float - vector/variable) input stocks price at time t
# t: (float - vector/varible) input time
# vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
def compute_d1_d2(model, option, St, t, vol):

    # initalize variables
    r = model.r
    K = option.K
    T = option.T
    tau = T - t

    # initalizie sigma
    if vol == "fixed":
        sigma2 = model.sigma(T)**2
    if vol == "variable":
        sigma2 = model.sigma(t)**2
    if vol == "s2":
        sigma2 = model.s2(t)

    # compute d1 and d2
    d1 = (np.log(St/K) + (r + 0.5*sigma2)*tau) * (1 / np.sqrt(sigma2 *
tau))
    d2 = d1 - np.sqrt(sigma2 * tau)

    return d1, d2
```

```
## black-scholes call option price
# model: (class) input market model name followed by ()
# options: (class) input option model name followed by ()
# st: (float - vector/variable) input stocks price at time t
# t: (float - vector/varible) input time
# vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
def BS_call(model, option, St, t, vol):

    # initalize variables
    d1, d2 = compute_d1_d2(model, option, St, t, vol)
    r = model.r
    K = option.K
    T = option.T
    tau = T - t

    return N(d1) * St - N(d2)*K*np.exp(-r*tau)

## black-scholes put option price
# model: (class) inpurt market model name followed by ()
# options: (class) input option model name followed by ()
# st: (float - vector/variable) stocks price at time t
# t: (float -vector/variable) time
# vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
def BS_put(model, option, St, t, vol):

    # initalize variables
    d1, d2 = compute_d1_d2(model, option, St, t, vol)
    r = model.r
    K = option.K
    T = option.T
    tau = T - t

    return N(-d2)*K*np.exp(-r*tau) - N(-d1) * St


## black-scholes delta value
# model: (class) inpurt market model name followed by ()
# options: (class) input option model name followed by ()
# st: (float - vector/variable) stocks price at time t
# t: (float -vector/variable) time
# vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
# CorP: (string) input "C" for call option and "P" for put option
def BS_delta(model, option, St, t, vol, CorP):

    if CorP == "C":
        return N(compute_d1_d2(model, option, St, t, vol)[0])

    if CorP == "P":
        return N(-compute_d1_d2(model, option, St, t, vol)[0])

## black-scholes gamma value
# model: (class) inpurt market model name followed by ()
# options: (class) input option model name followed by ()
# st: (float - vector/variable) stocks price at time t
```

```
# t: (float -vector/variable) time
# vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
# CorP: (string) input "C" for call option and "P" for put option
def BS_gamma(model, option, St, t, vol, CorP):

    # initalize variables
    T = option.T
    tau = T - t

    # initalizie sigma
    if vol == "fixed":
        sigma2 = model.sigma(T)**2
    if vol == "variable":
        sigma2 = model.sigma(t)**2
    if vol == "s2":
        sigma2 = model.s2(t)

    if CorP == "C":
        return n(compute_d1_d2(model, option, St, t, vol)
[0])/(St*np.sqrt(sigma2*tau))

    if CorP == "P":
        return n(compute_d1_d2(model, option, St, t, vol)
[0])/(St*np.sqrt(sigma2*tau))
```

Tests

1. test_N() - This test checks the output of the standard normal CDF with pre-determated values
2. test_n() - This test checks the ouput of the standard normal PDF with pre-determated values
3. test_compute_d1_d2() - This test checks d1 and d2 values with externaly calculated values
4. test_BS_call() - This test checks black-scholes call option values with externaly calculated values for fixed (constant) and variable (time dependent) implied volaitlities
5. test_BS_put() - This test checks black-scholes put options with externaly calculated values for fixed (constant) and variable (time dependent) implied volaitlities
6. test_BS_delta() - This test check black-scholes delta values with externaly calculated values of put and call options with fixed (constant) and variable (time dependent) implied volaitlities
7. test_BS_gamma() - This test check black-scholes gamma values with externaly calculated values of put and call options with fixed (constant) and variable (time dependent) implied volaitlities

In [12]:

```
### Black-Scholes (Testing) ###

# test N()
def test_N():
```

```
    # test Standard Normal CDF values
    assert N(0) == 0.5, "incorrect normal CDF value"
    assert round(N(2), 3) == 0.977, "incorrect normal CDF value"

    # return
    return print("Test Passed :)")

# run test
test_N()

# test n()
def test_n():

    # test Standard Normal PDF values
    assert round(n(0), 3) == 0.399, "incorrect normal PDF value"
    assert round(n(-2), 3) == 0.054, "incorrect normal PDF value"

    #return
    return print("Test Passed :)")

# run test
test_n()

# test  compute_d1_d2()
def test_compute_d1_d2():

    # compute d1 and d2 values
    fd1, fd2 = compute_d1_d2(market_model(), option(), 135, 1.5,
"fixed")
    vd1, vd2 = compute_d1_d2(market_model(), option_H(), 93, 0.9,
"s2")

    # test d1 values
    assert round(fd1, 3) == -0.265, "incorrect d1 value (fixed)"
    assert round(vd1, 3) == -0.415, "incorrect d1 value (variable)"


    # test d2 values
    assert round(fd2, 3) == -0.348, "incorrect d2 value (fixed)"
    assert round(vd2, 3) == -0.599, "incorrect d2 value (variable)"

    # return
    return print("Test Passed :)")

# run test
test_compute_d1_d2()

# test BS_call
def test_BS_call():

    # test BS_call values
    assert round(BS_call(market_model(), option(), 302, 1.4, "s2"), 3)
== 164.108, "incorrect BS call option value (variable)"
    assert round(BS_call(market_model(), option_H(), 64, 0.3,
"fixed"), 3) == 0.697, "incorrect BS call option value (fixed)"

    # return
    return print("Test Passed :)")

# run test
test_BS_call()
```

```
# test BS_put
def test_BS_put():

    # test BS_put values
    assert round(BS_put(market_model(), option_H(), 267, 0.5, "s2"),
3) == 0.000, "incorrect BS put option value (variable)"
    assert round(BS_put(market_model(), option(), 59, 1.1, "fixed"),
3) == 77.248, "incorrect BS put option value (fixed)"

    # return
    return print("Test Passed :)")

# run test
test_BS_put()

# test BS_delta
def test_BS_delta():

    # test BS_delta for call options
    assert round(BS_delta(market_model(), option(), 179, 1.3, "fixed",
"C"), 3) == 0.973, "incorrect BS delta value for call option (fixed)"
    assert round(BS_delta(market_model(), option_H(), 73, 0.2, "s2",
"C"), 3) == 0.099, "incorrect BS delta value for call option (fixed)"

    # test BS_delta for call options
    assert round(BS_delta(market_model(), option_H(), 221, 0.8,
"fixed", "P"), 3) == 0.00, "incorrect BS delta value for call option
(fixed)"
    assert round(BS_delta(market_model(), option(), 140, 0.9, "s2",
"P"), 3) == 0.389, "incorrect BS delta value for call option (fixed)"

    # return
    return print("Test Passed :)")

# run test
test_BS_delta()

# test BS_gamma
def test_BS_gamma():

    # test BS_delta for call options
    assert round(BS_gamma(market_model(), option(), 170, 0.5, "fixed",
"C"), 3) == 0.005, "incorrect BS delta value for call option (fixed)"
    assert round(BS_gamma(market_model(), option_H(), 130, 0.2, "s2",
"C"), 3) == 0.006, "incorrect BS delta value for call option (fixed)"

    # test BS_delta for call options
    assert round(BS_gamma(market_model(), option(), 110, 0.1, "fixed",
"P"), 3) == 0.011, "incorrect BS delta value for call option (fixed)"
    assert round(BS_gamma(market_model(), option_H(), 90, 0.3, "s2",
"P"), 3) == 0.019, "incorrect BS delta value for call option (fixed)"

    # return
    return print("Test Passed :)")

# run tests
test_BS_gamma()
```

Out[12]:
```
Test Passed :)
Test Passed :)
Test Passed :)
Test Passed :)
Test Passed :)
Test Passed :)
Test Passed :)
```

Simulating Hedging Strategies

Functions

1. simulate_delta_headging(model, option, N, n, GBM_vol, BS_vol, method, CorP)
2. half_simulate_gamma_delta_headging(model, option, option_H, N, n, GBM_vol, BS_vol, method, CorP)

In [13]:
```
### Heaging ###

## Delta Headging
# model: (class) inpurt market model name followed by ()
# options: (class) input option model name followed by ()
# N: (int) input the number of simulations of GBM performed
# n: (int) input the number of time steps performed
# GMB_vol: (string) input "fixed" for sigma(T) volatility model or
"variable" for sigma(t) volatility model or "s2" for s(t) volatility
model
# BS_vol: (string) input "fixed" for sigma(T) volatility model or
"variable" or sigma(t) volatility model
# method: (string) input "explicit" for simulation using the explicit
solution of the ODE or "euler" for simulation using euler maruyama
scheme
# CorP: (string) input "C" or "P" for call or put option respectively
def simulate_delta_headging(model, option, N, n, GBM_vol, BS_vol,
method, CorP):

    # Simulate stock prices
    S = simulate_GBM(model, method, N , n, GBM_vol)
    S_0 = model.S0

    # BS option price at time 0
    if CorP == "C":
        V_0 = BS_call(model, option, S_0, 0, BS_vol)

    if CorP == "P":
        V_0 = BS_put(model, option, S_0, 0, BS_vol)

    # initalize variables
    dt = model.T_max/n
    r = model.r
    Pi = np.zeros([N, n+1])
    B = np.zeros([N, n+1])
    q = np.zeros([N, n+1])

    # Set inital wealth, bank and delta
    Pi[:,0] = V_0
    q[:,0] = BS_delta(model, option, S_0, 0, BS_vol, CorP)
```

```
    B[:,0]  = Pi[:,0] - q[:,0]*S_0

    # Simulate delta Headging
    for i in range (0, n-1):

        # Set wealth, delta and bank for time t+dt
        Pi[:, i+1] = (np.exp(r*dt) * B[:, i]) + (q[:,i] * S[:,i+1])
        q[:, i+1] = BS_delta(model, option, S[:,i+1], dt*(i+1),
BS_vol, CorP)
        B[:, i+1] = Pi[:, i+1] - (q[:, i+1] * S[:,i+1] )

    # Set final wealth
    Pi[:, -1] = (np.exp(r*dt) * B[:, -2]) + (q[:,-2] * S[:,-1])

    # calculate error terms
    if CorP == "C":
        error = Pi[:,-1] - option.payoff_call(S[:,-1])
    if CorP == "P":
        error = Pi[:,-1] - option.payoff_put(S[:,-1])

    return S, Pi, error

## Gamma Headging
# model: (class) inpurt market model name followed by ()
# option: (class) input option model name followed by () this should
be the model of the option being headged
# option_H: (class) input option model name followed by () theis
should be model of the option being used in the gamma headging
stratgegy
# N: (int) input the number of simulations of GBM performed
# n: (int) input the number of time steps performed
# GBM_vol: (string) input "fixed" for sigma(T) volatility model or
"variable" or sigma(t) volatility model or "s2" for s(t) volatility
model
# BS_vol: (string) input "fixed" for sigma(T) volatility model or
"variable" or sigma(t) volatility model
# method: (string) input "explicit" for simulation using the explicit
solution of the ODE or "euler" for simulation using euler maruyama
scheme
# CorP: (string) input "C" or "P" for call or put option respectively
def half_simulate_gamma_delta_headging(model, option, option_H, N, n,
GBM_vol, BS_vol, method, CorP):

    # Simulate stock prices
    S = simulate_GBM(model, method, N , n, GBM_vol)
    S_0 = model.S0

    # BS option price at time 0
    if CorP == "C":
        V_0 = BS_call(model, option, S_0, 0, BS_vol)
        PorC = "P"
        BS_market_price = BS_put
    if CorP == "P":
        V_0 = BS_put(model, option, S_0, 0, BS_vol)
        PorC = "C"
        BS_market_price = BS_call

    # initalize variables
    dt = model.T_max/n
    r = model.r
    Pi = np.zeros([N, int(n/2)+1])
```

```
    B = np.zeros([N, int(n/2)+1])
    qS = np.zeros([N, int(n/2)+1])
    qH = np.zeros([N, int(n/2)+1])

    # Set inital wealth
    Pi[:,0] = V_0
    qH[:,0] = BS_gamma(model, option, S_0, 0, BS_vol, CorP) /
BS_gamma(model, option_H, S_0, 0, BS_vol, PorC)
    qS[:,0] = BS_delta(model, option, S_0, 0, BS_vol, CorP) +
BS_delta(model, option_H, S_0, 0, BS_vol, PorC)*qH[:,0]
    B[:,0] = Pi[:,0] - qS[:,0]*S_0 - qH[:,0]*BS_market_price(model,
option_H, S_0, 0, BS_vol)

    # Simulate delta Headging
    for i in range (0, int(n/2)):

        # Set wealth, stock quantility, option quantity and bank for
time t+dt
        Pi[:, i+1] = (np.exp(r*dt) * B[:, i]) + (qS[:,i] * S[:,i+1]) +
(qH[:,i]*BS_market_price(model, option_H, S[:,i+1], (i+1)*dt, BS_vol))
        qH[:, i+1] = BS_gamma(model, option, S[:,i+1], (i+1)*dt,
BS_vol, CorP) / BS_gamma(model, option_H, S[:,i+1], (i+1)*dt, BS_vol,
PorC)
        qS[:, i+1] = BS_delta(model, option, S[:,i+1], (i+1)*dt,
BS_vol, CorP) + BS_delta(model, option_H, S[:,i+1], (i+1)*dt, BS_vol,
PorC)*qH[:, i+1]
        B[:, i+1]  = Pi[:, i+1] - qS[:, i+1]*S[:,i+1] - qH[:,
i+1]*BS_market_price(model, option_H, S[:,i+1], (i+1)*dt, BS_vol)

    # calculate error values
    if CorP == "C":
        error = Pi[:,int(n/2)] - BS_call(model, option, S[:,
int(n/2)], int(n/2)*dt, BS_vol)
    if CorP == "P":
        error = Pi[:,int(n/2)] - BS_put(model, option, S[:, int(n/2)],
int(n/2)*dt, BS_vol)

    return S, Pi, error
```

Testing

1. test_simulate_delta_headging() - This test check for large values of error when delta hedging
2. test_half_simulate_gamma_delta_headging() - this test checks for large values of error when gamma hedging.

In [14]:

```
### Headging (Testing) ###

# test simulate_delta_headging()
def test_simulate_delta_headging():

    # simulate delta hedging (fixed)
    error = simulate_delta_headging(market_model(), option(), 1,
10000, "fixed", "fixed", "explicit", "C")[2]

    # test for large errors for delta headging (fixed)
    assert -0.25 < error < 0.25, "error, large delta hedge error
(fixed)"
```

```
    # simulate delta hedging (variable)
    error = simulate_delta_headging(market_model(), option(), 1,
10000, "variable", "s2", "explicit", "C")[2]

    # test for large errors for delta headging (fixed)
    assert -0.25 < error < 0.25, "error, large delta hedge error
(fixed)"

    # return
    return print("Test Passed :)")

# test half_simulate_gamma_delta_headging()
def test_simulate_gamma_delta_headging():

     # simulate gamma hedging (fixed)
    error = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1, 1000, "fixed", "fixed", "explicit", "C")[2]

    # test for large errors for delta headging (fixed)
    assert -0.05 < error < 0.05, "error, large gamma hedge error
(fixed)"

     # simulate gamma hedging (variable)
    error = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1, 1000, "variable", "s2", "explicit", "C")[2]

    # test for large errors for gamma headging (fixed)
    assert -0.05 < error < 0.05, "error, large gamma hedge error
(variable)"

    # return
    return print("Test Passed :)")

# run tests
test_simulate_delta_headging()
test_simulate_gamma_delta_headging()
```

Out[14]:
```
Test Passed :)
Test Passed :)
```

Questions

Question 1

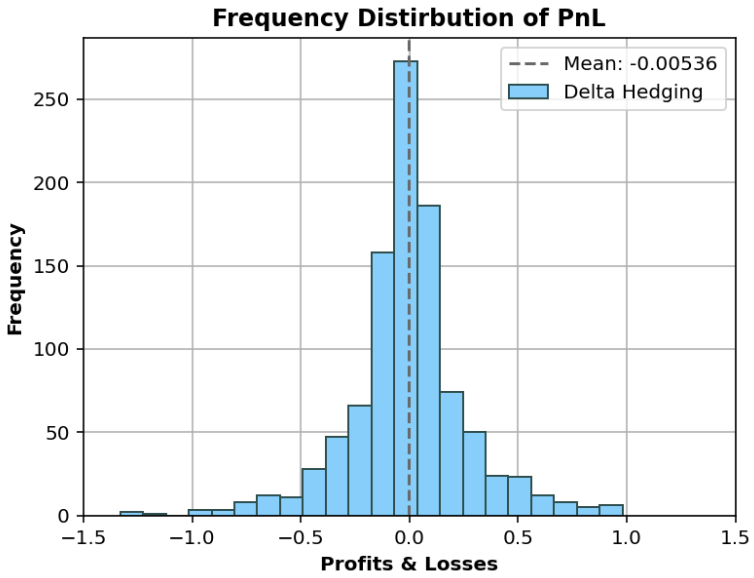
We simulate delta hedging for a call option with strike price $K = 139$, matruity $T = 1.60$ for 1000 simulation and 1000 time steps between time 0 and $T = 1.60$. The true underlying stock price proccess follows $dS_t = S_t\mu dt + S_t\sigma(t)dW_t$ where $\mu = 0.12$ and $\sigma(t) = 0.13 + 0.13\left(\frac{t-0.50}{T-0.50}\right)$ if $t > 0.50$ or $\sigma(t) = 0.13$ if $t \leq 0.50$. We assume during the delta hedging strategy that the stock price process is $dS_t = S_t\mu dt + S_t\sigma(t)dW_t$ and thus we uses $C_t^{\text{BS}}(S_t, K, T, s(t), r)$ where $s(t) = \frac{1}{T-t}\int_t^T \sigma(s)ds$ when calculating the delta and price of the call option.

In [15]:
```
## Simulate Delta Headge
S, Pi, PnL = simulate_delta_headging(market_model(), option(), 1000,
2000, "variable", "s2", "explicit", "C")
```

We can then plot the profit data from the monte-carlo simulation of delta hedging a call option assuming correctly that S_t has time dependent volatility. The plot chosen to represent this data is a frequency histogram.

In [16]:
```
## Histrogram of profits
plt.figure(figsize=(6,4.5))
ax = plt.gca()
ax.grid()
ax.set_axisbelow(True)
ax.hist(PnL, bins=22, ec="darkslategrey", color="lightskyblue")
ax.axvline(np.mean(PnL), color="dimgrey" ,ls="dashed")
ax.set_xlim(-1.5, 1.5)
ax.set_xlabel("Profits & Losses", fontweight="bold")
ax.set_ylabel("Frequency", fontweight="bold")
ax.set_title("Frequency Distirbution of PnL", fontweight="bold")
ax.legend([("Mean: {}".format(round(np.mean(PnL), 5))), "Delta
Hedging"])
```

Out[16]: <matplotlib.legend.Legend at 0x7f7deb561790>



We compute the sample mean for the simulated profit data and the 95% confidence interval for the population mean

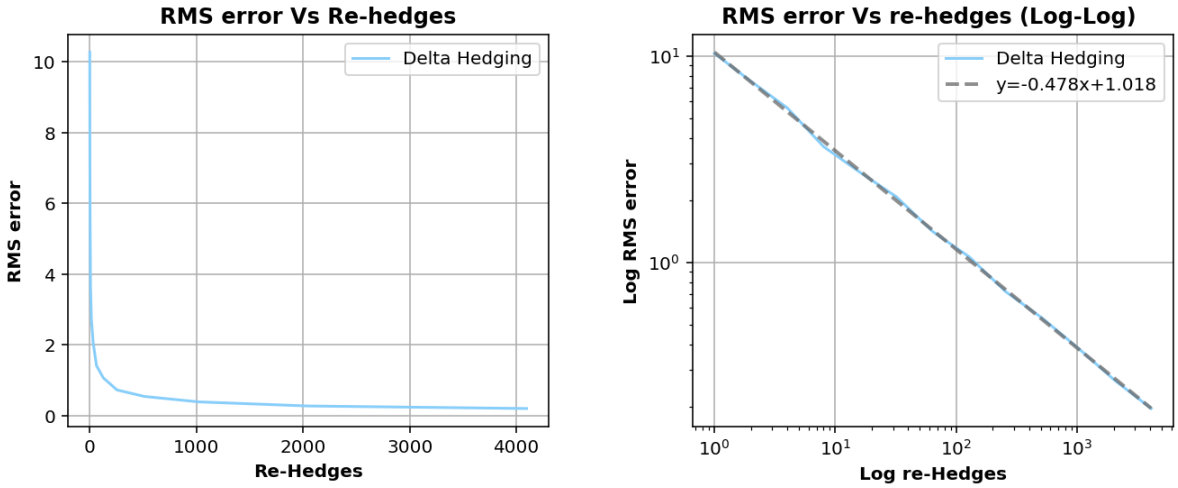
In [17]:
```
## calculate sample mean
print("Sample mean:", np.mean(PnL))

## 95% confidence interval of population mean
CI_sample = [np.mean(PnL)-1.96*np.sqrt(np.var(PnL)/len(PnL)),
np.mean(PnL)+1.96*np.sqrt(np.var(PnL)/len(PnL))]
print("95% Confidence Interval of sample mean:", CI_sample)
```

Out[17]:
```
Sample mean: -0.005357854738382922
95% Confidence Interval of sample mean: [-0.022099230326901514,
0.011383520850135668]
```

We now show it is numerically show it is possible to replicate the call option by plotting the RMS error vs re-hedge and the log-log chart of this, where we find the RMS order of convergence to be aproximately $\mathcal{O}(n^{\frac{1}{2}})$

In [18]:
```
### Log-Log plot

### Comaprison of Delta And Gamma Hedging Plot

# plot settings
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figheight(4)
fig.set_figwidth(11)
plt.subplots_adjust(wspace=0.3)

# plotting points for plots 1 and 2
n_points = 13
n_steps = np.zeros(n_points)
rms_error = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i

    # delta
    S, Pi, error = simulate_delta_headging(market_model(), option(),
1000, int(n_steps[i]), "variable", "s2", "explicit", "C")

    rms_error[i] = np.sqrt(np.mean(error**2))

# Plot 1
ax1.grid()
ax1.plot(n_steps, rms_error, color="lightskyblue")
ax1.set_xlabel("Re-Hedges", fontweight="bold")
ax1.set_ylabel("RMS error", fontweight="bold")
ax1.set_title("RMS error Vs Re-hedges", fontweight="bold")
ax1.legend(["Delta Hedging"])

# Line of best fit Log-Log (gamma)
coefficients = np.polyfit(np.log10(n_steps), np.log10(rms_error), 1)
polynomial = np.poly1d(coefficients)
log10_y_fit = polynomial(np.log10(n_steps))

# Plot 2
ax2.grid()
ax2.loglog(n_steps, rms_error, color="lightskyblue")
ax2.plot(n_steps, 10**log10_y_fit, color="dimgrey", alpha=0.75,
linestyle="dashed", linewidth=2)
ax2.set_xlabel("Log re-Hedges", fontweight="bold")
ax2.set_ylabel("Log RMS error", fontweight="bold")
ax2.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax2.legend(["Delta Hedging", "y={}x+{}".format(round(coefficients[0],
3), round(coefficients[1], 3))])
```

Out[18]: <matplotlib.legend.Legend at 0x7f7de74c4760>



We also compute the black-scholes price of the call option with strike price $K = 139$, matruity $T = 1.60$ at time 0 and inital stock price $S_0 = 105$

In [19]:
```
## Call Option Value at time 0
print("Price of call option at time 0:", BS_call(market_model(),
option(), 105, 0, "s2"))
```

Out[19]: Price of call option at time 0: 2.4240933082154044

Question 2

We simulate delta hedging of a single call option with strike price $K = 139$, matruity $T = 1.60$ for 1000 simulation and 1000 time steps between time 0 and $T/2 = 0.80$. The true underlying stock price proccess follows $dS_t = S_t\mu dt + S_t\sigma(t)dW_t$ where $\mu = 0.12$ and $\sigma(t) = 0.13 + 0.13\left(\frac{t-0.50}{T-0.50}\right)$ if $t > 0.50$ and $\sigma(t) = 0.13$ if $t \leq 0.50$. However in the delta hedging strategy we assume the stock price process is $dS_t = S_t\mu dt + S_t\sigma(T)dW_t$ where $\sigma(T) = 0.26$ and thus we uses $C_t^{\text{BS}}(S_t, K, T, \sigma(T), r)$ to calculate the value and delta of the call option being hedged.

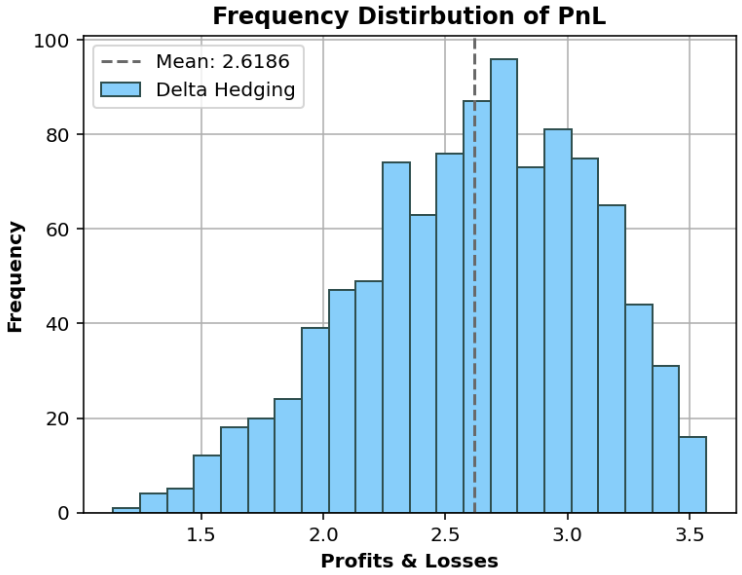
In [20]:
```
## Simulate Delta Headge with ...
S, Pi, error = simulate_delta_headging(market_model(), option(), 1000,
2000, "variable", "fixed", "explicit", "C")

## PnL if Hedging strategy is liquidated at T/2
PnL = Pi[:, 1000] - BS_call(market_model(), option(), S[:, 1000],
0.5*1.6, "fixed")
```

We plot a histogram of the profit data produced by the monte-carlo simulation for a call option hedged using an incorrect volaility model assumption.

In [21]:
```
## Histrogram of profits
plt.figure(figsize=(6,4.5))
```

```
ax = plt.gca()
ax.grid()
ax.set_axisbelow(True)
ax.hist(PnL, bins=22, ec="darkslategrey", color="lightskyblue")
ax.axvline(np.mean(PnL), color="dimgrey" ,ls="dashed")
ax.set_xlabel('Profits & Losses', fontweight="bold")
ax.set_ylabel('Frequency', fontweight="bold")
ax.set_title("Frequency Distirbution of PnL", fontweight="bold")
ax.legend([("Mean: {}".format(round(np.mean(PnL), 5))), "Delta
Hedging"])
```

Out[21]: <matplotlib.legend.Legend at 0x7f7deb278af0>



We Compute the sample mean of the monte-carlo simulation profit data and the 95% confidence interval of the population mean

In [22]:
```
## Mean
print("Sample mean:", np.mean(PnL))

## 95% confidence interval
CI_sample = [np.mean(PnL)-1.96*np.sqrt(np.var(PnL)/len(PnL)),
np.mean(PnL)+1.96*np.sqrt(np.var(PnL)/len(PnL))]
print("Confidence Interval of sample mean:", CI_sample)
```

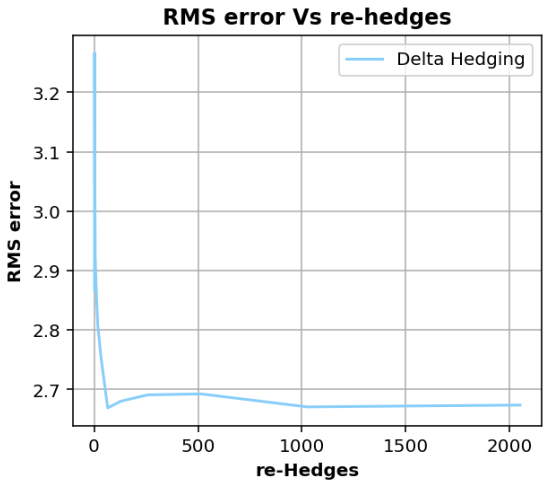
Out[22]:
```
Sample mean: 2.618598115732542
Confidence Interval of sample mean: [2.5888434393185933,
2.648352792146491]
```

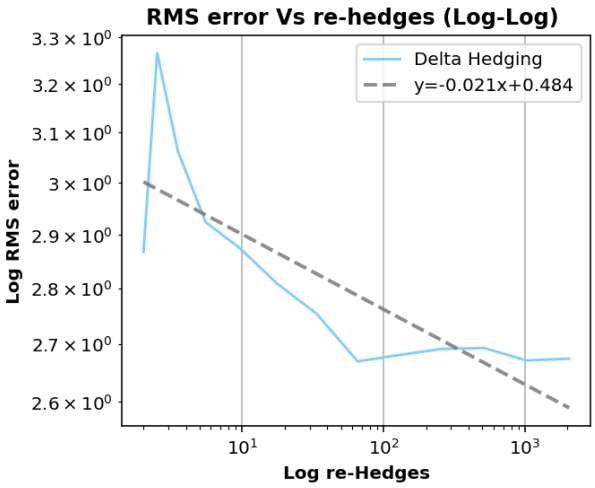
We now show it is numerically show it is not possible to replicate the call option by plotting the RMS error vs re-hedge and a partnered log-log plot, were we do not find a convergence to zero

In [23]:
```
### Comaprison of Delta And Gamma Hedging Plot

# plot settings
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figheight(4)
fig.set_figwidth(11)
plt.subplots_adjust(wspace=0.35)
```

```
# plotting points for plots 1 and 2
n_points = 13
n_steps = np.zeros(n_points)
rms_error = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i+3

    # delta
    S, Pi, error = simulate_delta_headging(market_model(), option(),
1000, int(n_steps[i]), "variable", "fixed", "explicit", "C")
    error = Pi[:, int(n_steps[i]/2)] - BS_call(market_model(),
option(), S[:, int(n_steps[i]/2)], 0.5*1.6, "fixed")

    rms_error[i] = np.sqrt(np.mean(error**2))

# Plot 1
ax1.grid()
ax1.plot(n_steps/2, rms_error, color="lightskyblue")
ax1.set_xlabel("re-Hedges", fontweight="bold")
ax1.set_ylabel("RMS error", fontweight="bold")
ax1.set_title("RMS error Vs re-hedges", fontweight="bold")
ax1.legend(["Delta Hedging"])

# Line of best fit Log-Log (gamma)
coefficients = np.polyfit(np.log10(n_steps/2), np.log10(rms_error), 1)
polynomial = np.poly1d(coefficients)
log10_y_fit = polynomial(np.log10(n_steps/2))

# Plot 2
ax2.grid()
ax2.loglog(n_steps/2, rms_error, color='lightskyblue')
ax2.plot(n_steps/2, 10**log10_y_fit, color="dimgrey", alpha=0.75,
linestyle="dashed", linewidth=2)
ax2.set_xlabel("Log re-Hedges", fontweight="bold")
ax2.set_ylabel("Log RMS error", fontweight="bold")
ax2.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax2.legend(["Delta Hedging", "y={}x+{}".format(round(coefficients[0],
3), round(coefficients[1], 3))])
```

Out[23]: <matplotlib.legend.Legend at 0x7f7de9944fd0>





Question 3

We simulate delta hedging for a call option with strike price $K = 139$, matruity $T = 1.60$ for 1000 simulation and 1000 time steps between time 0 and $T/2 = 0.80$. The true underlying stock price proccess follows $dS_t = S_t\mu dt + S_t\sigma(T)dW_t$ where $\mu = 0.12$ and $\sigma(T) = 0.26$. When delta hedging we assumed the stock price process is $dS_t = S_t\mu dt + S_t\sigma(T)dW_t$ and thus we uses $C_t^{\text{BS}}(S_t, K, T, \sigma(T), r)$ when calculating the delta, gamma and price of the call option.

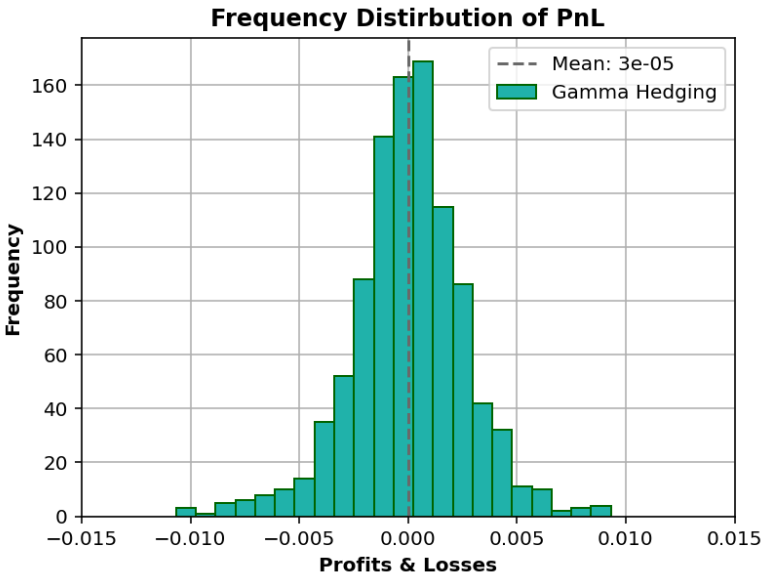
We also add an extra two simulations for comparison of delta and gamma hedging. These simulations simulate the respective hedging strategy for 1000 simulations between time 0 and $T/2 = 0.80$. The set up of the true and assumed unerlying GBM procces is the same as above. analsysis

In [24]:
```
## Simulate half Delta, Gamma hedge strategy (main)
S, Pi, PnL = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, 2000, "fixed", "fixed", "explicit", "C")

## Simulate Delta Hedge stratgey (extra)
S_alt, Pi_alt, error = simulate_delta_headging(market_model(),
option(), 1000, 2000, "fixed", "fixed", "explicit", "C")

## PnL if Hedging strategy (extra) is liquidated at T/2
PnL_alt = Pi_alt[:, 1000] - BS_call(market_model(), option(), S_alt[:,
1000], 0.5*1.6, "fixed")

## Simulate half Delta, Gamma hedge strategy (extra)
S_alt2, Pi_alt2, PnL_alt2 =
half_simulate_gamma_delta_headging(market_model(), option(),
option_H(), 1000, 100, "fixed", "fixed", "explicit", "C")
```

We plot a histogram of the profit data produced by the monte-carlo simulation of Gamma hedging

In [25]:
```
## Histogram of the profits
plt.figure(figsize=(6,4.5))
ax = plt.gca()
ax.grid()
ax.set_axisbelow(True)
ax.hist(PnL, bins=22,  ec="darkgreen", color="lightseagreen")
ax.axvline(np.mean(PnL), color="dimgrey" ,ls="dashed")
ax.set_xlim([-0.015, 0.015])
ax.set_xlabel('Profits & Losses', fontweight="bold")
ax.set_ylabel('Frequency', fontweight="bold")
ax.set_title("Frequency Distirbution of PnL", fontweight="bold")
ax.legend([("Mean: {}".format(round(np.mean(PnL), 5))), "Gamma
Hedging"])
```

Out[25]: <matplotlib.legend.Legend at 0x7f7deb485d90>



We compute the sample mean of the monte-carlo simulation profit data and the 95% confidence interval of the population mean

In [26]:
```
## Mean
print("Sample mean:", np.mean(PnL))

## 95% confidence interval
CI_sample = [np.mean(PnL)-1.96*np.sqrt(np.var(PnL)/len(PnL)),
np.mean(PnL)+1.96*np.sqrt(np.var(PnL)/len(PnL))]
print("Confidence Interval of sample mean:", CI_sample)
```

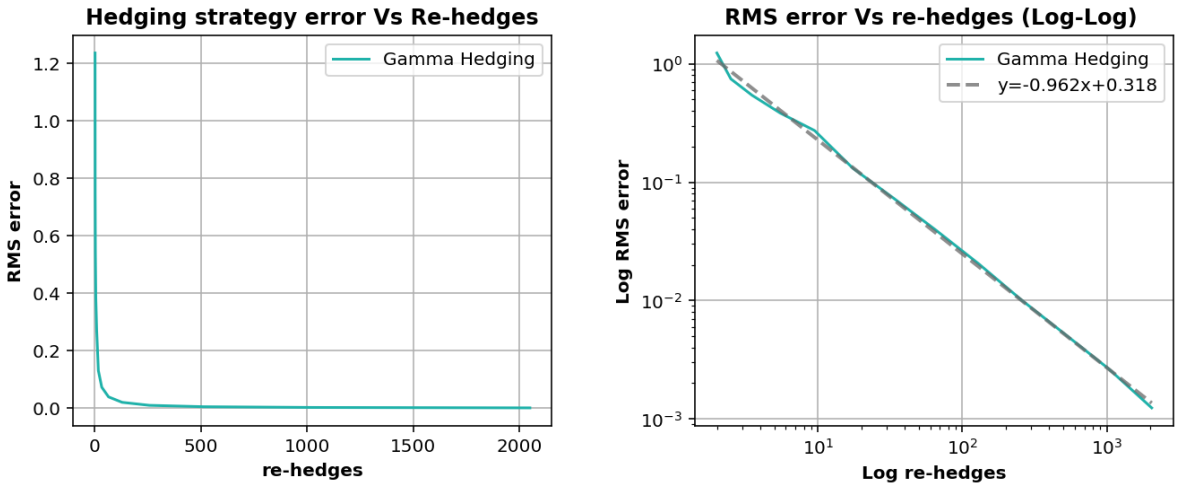
Out[26]:
```
Sample mean: 3.033986826971866e-05
Confidence Interval of sample mean: [-0.00013144177947437222,
0.00019212151601380952]
```

We now show it is numerically show it is not possible to replicate the call option by plotting the RMS error vs re-hedge and a partnered log-log plot. We also find the and finding the RMS order of convergence to be aproximately $\mathcal{O}(n^{-1})$

In [27]:
```
### Comaprison of Delta And Gamma Hedging Plot

# plot settings
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figheight(4)
fig.set_figwidth(11)
plt.subplots_adjust(wspace=0.3)

# plotting points for plots 1 and 2
n_points = 13
n_steps = np.zeros(n_points)
rms_error = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i+3
```

```
    # Gamma
    S, Pi, error = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, int(n_steps[i]), "fixed", "fixed",
"explicit", "C")

    rms_error[i] = np.sqrt(np.mean(error**2))

# Plot 1
ax1.grid()
ax1.plot(n_steps/2, rms_error, color="lightseagreen")
ax1.set_xlabel('re-hedges', fontweight="bold")
ax1.set_ylabel('RMS error', fontweight="bold")
ax1.set_title('Hedging strategy error Vs Re-hedges',
fontweight="bold")
ax1.legend(["Gamma Hedging"])

# Line of best fit Log-Log (gamma)
coefficients = np.polyfit(np.log10(n_steps/2), np.log10(rms_error), 1)
polynomial = np.poly1d(coefficients)
log10_y_fit = polynomial(np.log10(n_steps/2))

# Plot 2
ax2.grid()
ax2.loglog(n_steps/2, rms_error, color='lightseagreen')
ax2.plot(n_steps/2, 10**log10_y_fit, color="dimgray", alpha=0.75,
linestyle="dashed", linewidth=2)
ax2.set_xlabel("Log re-hedges", fontweight="bold")
ax2.set_ylabel("Log RMS error", fontweight="bold")
ax2.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax2.legend(["Gamma Hedging", "y={}x+{}".format(round(coefficients[0],
3), round(coefficients[1], 3))])
```

Out[27]: <matplotlib.legend.Legend at 0x7f7de9944430>



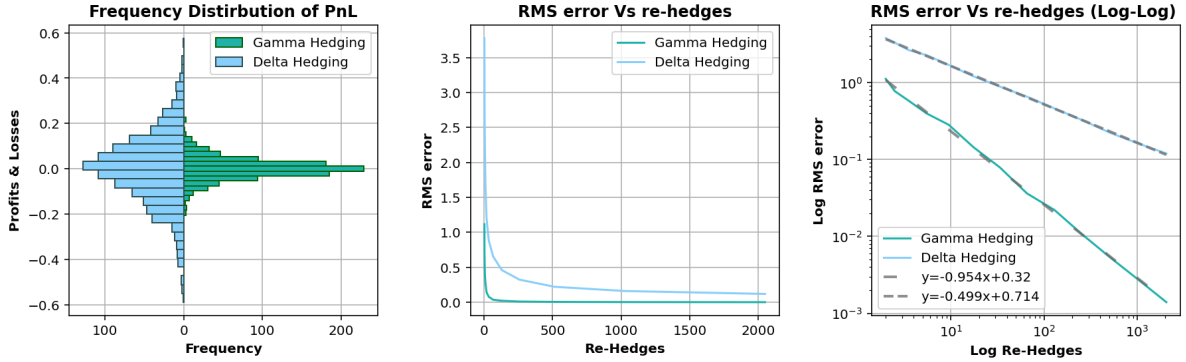
We compare Delta and gamma hedging through graphical means of the following three plots produced. Plot 1 is a side-by-side histogram of delta hedging from 0 to $T/2$ with 1000 steps amd gamma hedging from 0 to $T/2$ with 50 steps. Plot 2 shows the evolution of root mean squared error as the number of steps is increased. Plot 3 shows the log-log plot of the root mean squared errors as the number of steps is increased.

In [28]:

```
### Comaprison of Delta And Gamma Hedging Plot

# ignore warnings
warnings.filterwarnings("ignore")

# plot settings
fig, (ax1, ax2, ax3) = plt.subplots(1, 3)
fig.set_figheight(4)
fig.set_figwidth(15)
plt.subplots_adjust(wspace=0.3)

# Plot 5,1 alternative
#ax1.grid()
#ax1.set_axisbelow(True)
#ax1.hist(PnL_alt, density=False, bins=20, ec="darkslategrey",
color="lightskyblue", alpha=0.75)
#ax1.hist(PnL_alt2, density=False, bins=20, ec="darkgreen",
color="lightseagreen", alpha=0.75)
#ax1.set_xlim([-0.8, 0.8])
#ax1.set_xlabel('Profits & Losses')
#ax1.set_ylabel('Frequency')
#ax1.legend(["Delta: 1000 Steps", "Gamma: 50 Steps"])

#   Plotting point for plot 1
heights, bins = np.histogram(PnL_alt, density=False, bins=30)
heights = heights * -1
bin_width = np.diff(bins)
bin_pos = (bins[:-1] + bin_width / 2) * -1

# Plot (5,1)
ax1.grid()
ax1.set_axisbelow(True)
sns.histplot(y=PnL_alt2, bins=20, edgecolor="darkgreen",
color="lightseagreen", alpha=1, ax=ax1)
ax1.barh(y=bin_pos, height=bin_width, width=heights,
edgecolor="darkslategrey", color="lightskyblue")
labels = [item.get_text() for item in ax1.get_xticklabels()]
labels[1] = "100"
ax1.set_xticklabels(labels)
ax1.set_xlabel('Frequency', fontweight="bold")
ax1.set_ylabel('Profits & Losses', fontweight="bold")
ax1.set_title("Frequency Distirbution of PnL", fontweight="bold")
ax1.legend(["Gamma Hedging", "Delta Hedging"])

# plotting points for plots (5,2) and (5,3)
n_points = 13
n_steps = np.zeros(n_points)
rms_error1 = np.zeros(n_points)
rms_error2 = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i+3

    # Gamma
    S, Pi, error1 = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, int(n_steps[i]), "fixed", "fixed",
"euler", "C")

    ## delta
```

```
    S, Pi, error = simulate_delta_headging(market_model(), option(),
1000, int(n_steps[i]), "fixed", "fixed", "explicit", "C")
    error2 = Pi[:, int(n_steps[i]/2)] - BS_call(market_model(),
option(), S[:, int(n_steps[i]/2)], 0.5*1.6, "fixed")

    #rms error
    rms_error1[i] = np.sqrt(np.mean(error1**2))
    rms_error2[i] = np.sqrt(np.mean(error2**2))

# Plot (5,2)
ax2.grid()
ax2.plot(n_steps/2, rms_error1, color="lightseagreen")
ax2.plot( n_steps/2, rms_error2, color="lightskyblue")
ax2.set_xlabel("Re-Hedges", fontweight="bold")
ax2.set_ylabel("RMS error", fontweight="bold")
ax2.set_title("RMS error Vs re-hedges", fontweight="bold")
ax2.legend(["Gamma Hedging", "Delta Hedging"])

# Line of best fit Log-Log (gamma)
coefficients1 = np.polyfit(np.log10(n_steps/2), np.log10(rms_error1),
1)
polynomial1 = np.poly1d(coefficients1)
log10_y1_fit = polynomial1(np.log10(n_steps/2))

# Line of best fit Log-Log (delta)
coefficients2 = np.polyfit(np.log10(n_steps/2), np.log10(rms_error2),
1)
polynomial2 = np.poly1d(coefficients2)
log10_y2_fit = polynomial2(np.log10(n_steps/2))

# Plot (5,3)
ax3.grid()
ax3.loglog(n_steps/2, rms_error1, color='lightseagreen')
ax3.loglog(n_steps/2, rms_error2, color="lightskyblue")
ax3.plot(n_steps/2, 10**log10_y1_fit, color="dimgrey", alpha=0.75,
linestyle=(0, (5, 10)), linewidth=2)
ax3.plot(n_steps/2, 10**log10_y2_fit, color="dimgrey", alpha=0.75,
linestyle="dashed", linewidth=2)
ax3.set_xlabel("Log Re-Hedges", fontweight="bold")
ax3.set_ylabel("Log RMS error", fontweight="bold")
ax3.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax3.legend(["Gamma Hedging", "Delta Hedging", "y={}x+
{}".format(round(coefficients1[0], 3), round(coefficients1[1], 3)),
"y={}x+{}".format(round(coefficients2[0], 3), round(coefficients2[1],
3)),])

# reset warnings
warnings.resetwarnings()
```

Out[28]:



Question 4

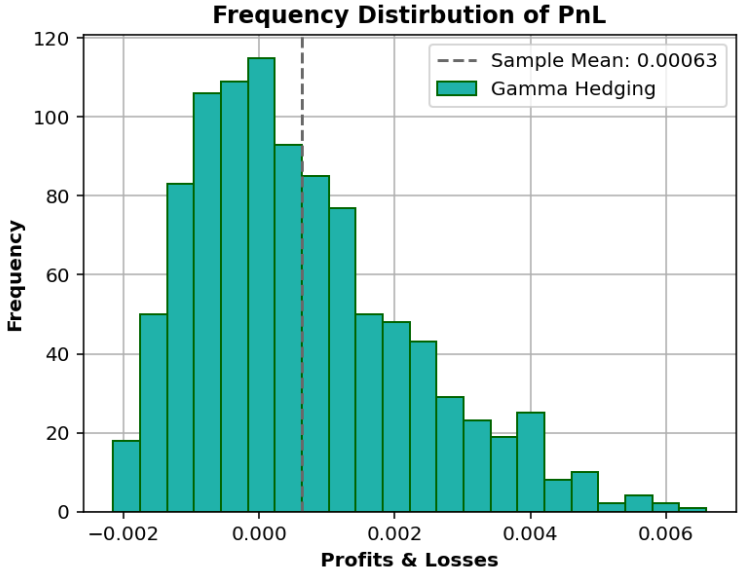
We simulate delta hedging for a call option with strike price $K = 139$, matruity $T = 1.60$ for 1000 simulation and 1000 time steps between time 0 and $T/2 = 0.80$. The true underlying stock price proccess follows $dS_t = S_t\mu dt + S_t\sigma(T)dW_t$ where $\mu = 0.12$ and $\sigma(T) = 0.26$. When delta hedging is performed we assumed the stock price process is $dS_t = S_t\mu dt + S_t\sigma(t)dW_t$ where $\sigma(t) = 0.13 + 0.13\left(\frac{t-0.50}{T-0.50}\right)$ if $t > 0.50$ and thus we uses $C_t^{\text{BS}}(S_t, K, T, s(t), r)$ where $s(t) = \frac{1}{T-t}\int_t^T \sigma(s)ds$ when calculating the delta, gamma and price of the call option.

In [29]:
```
## Simulate Delta, Gamma Headge with ...
S, Pi, PnL = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, 2000, "variable", "fixed", "explicit",
"C")
```

We plot the histogram of the profit data produced by the monte-carlo simulation for gamma hedging with a incorrect volatility assumption

In [30]:
```
## Histrogram of errors
plt.figure(figsize=(6,4.5))
ax = plt.gca()
ax.grid()
ax.set_axisbelow(True)
ax.hist(PnL, bins=22, ec="darkgreen", color="lightseagreen",
density=False)
ax.axvline(np.mean(PnL), color="dimgrey" ,ls="dashed")
ax.set_xlabel('Profits & Losses', fontweight="bold")
ax.set_ylabel('Frequency', fontweight="bold")
ax.set_title("Frequency Distirbution of PnL", fontweight="bold")
ax.legend([("Sample Mean: {}".format(round(np.mean(PnL), 5))), "Gamma
Hedging"])
```

Out[30]: <matplotlib.legend.Legend at 0x7f7de72d4df0>



We compute the sample mean of the profit data and the 95% confidence interval of the population mean.

In [31]:
```
## Mean
print("Sample mean:", np.mean(PnL))

## 95% confidence interval
CI_sample = [np.mean(PnL)-1.96*np.sqrt(np.var(PnL)/len(PnL)),
np.mean(PnL)+1.96*np.sqrt(np.var(PnL)/len(PnL))]
print("Confidence Interval of sample mean:", CI_sample)
```

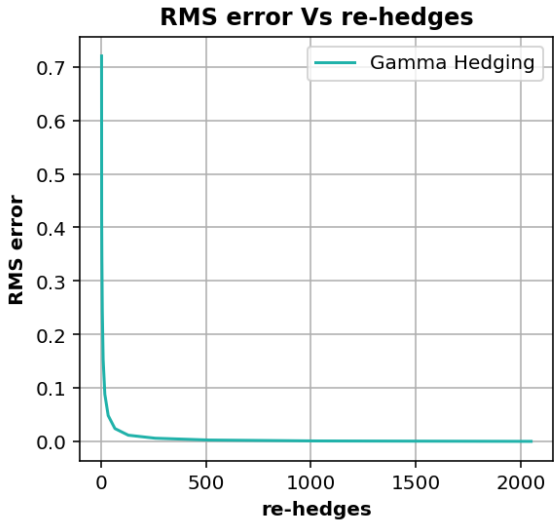
Out[31]:
```
Sample mean: 0.00062745145723028
Confidence Interval of sample mean: [0.0005286760564662733,
0.0007262268579942867]
```

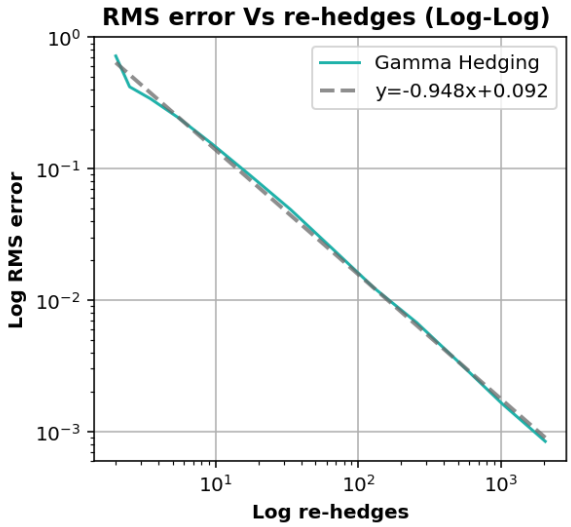
We now show it is numerically show it is possible to replicate the call option by plotting the RMS error vs re-hedge and a partnered log-log plot and finding the RMS order of convergence to be aproximately $\mathcal{O}(n^{-1})$

In [32]:
```
### Comaprison of Delta And Gamma Hedging Plot

# plot settings
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.set_figheight(4)
fig.set_figwidth(10)
plt.subplots_adjust(wspace=0.3)

# plotting points for plots 1 and 2
n_points = 13
n_steps = np.zeros(n_points)
rms_error = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i+3

    # Gamma
    S, Pi, error = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, int(n_steps[i]), "variable", "fixed",
"explicit", "C")

    rms_error[i] = np.sqrt(np.mean(error**2))

# Plot 1
ax1.grid()
ax1.plot(n_steps/2, rms_error, color="lightseagreen")
ax1.set_xlabel("re-hedges", fontweight="bold")
ax1.set_ylabel("RMS error", fontweight="bold")
ax1.set_title("RMS error Vs re-hedges", fontweight="bold")
ax1.legend(["Gamma Hedging"])

# Line of best fit Log-Log (gamma)
coefficients = np.polyfit(np.log10(n_steps/2), np.log10(rms_error), 1)
polynomial = np.poly1d(coefficients)
log10_y_fit = polynomial(np.log10(n_steps/2))

# Plot 2
```

```
ax2.grid()
ax2.loglog(n_steps/2, rms_error, color="lightseagreen")
ax2.plot(n_steps/2, 10**log10_y_fit, color="dimgrey", alpha=0.75,
linestyle="dashed", linewidth=2)
ax2.set_xlabel("Log re-hedges", fontweight="bold")
ax2.set_ylabel("Log RMS error", fontweight="bold")
ax2.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax2.legend(["Gamma Hedging", "y={}x+{}".format(round(coefficients[0],
3), round(coefficients[1], 3))])
```

Out[32]: <matplotlib.legend.Legend at 0x7f7de6e71ee0>





Extra

Quick Checks

This is a quick check that the Confidence intervals are valid by comparing a boatstraping methods to that of the Central Limit Theorm method used

In [33]:

```
### Boatstrap method Vs CLT method

## A check that the CI produced by the CLT is valid

# simulate PnL
S, Pi, PnL = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, 1000, "variable", "fixed", "explicit",
"C")

## 95% confidence interval (boot strap)
bootstrap_means = []
for i in range(100000):
    sample_PnL = np.random.choice(PnL, 1000, replace=True)
    bootstrap_means.append(np.mean(sample_PnL))
dirac = np.mean(bootstrap_means) - bootstrap_means
percentiles = np.percentile(dirac, [2.5, 97.5])
print("Boatstrap CI")
print([np.mean(bootstrap_means)-percentiles[1],
np.mean(bootstrap_means)-percentiles[0]])
```

```
## 95% confidence interval (CLT)
CI_sample = [np.mean(PnL)-1.96*np.sqrt(np.var(PnL)/len(PnL)),
np.mean(PnL)+1.96*np.sqrt(np.var(PnL)/len(PnL))]
print("CLT CI")
print(CI_sample)
```

Out[33]:
```
Boatstrap CI
[0.0010547451532207678, 0.0014407121970840524]
CLT CI
[0.0010543513870328127, 0.0014402169501554524]
```

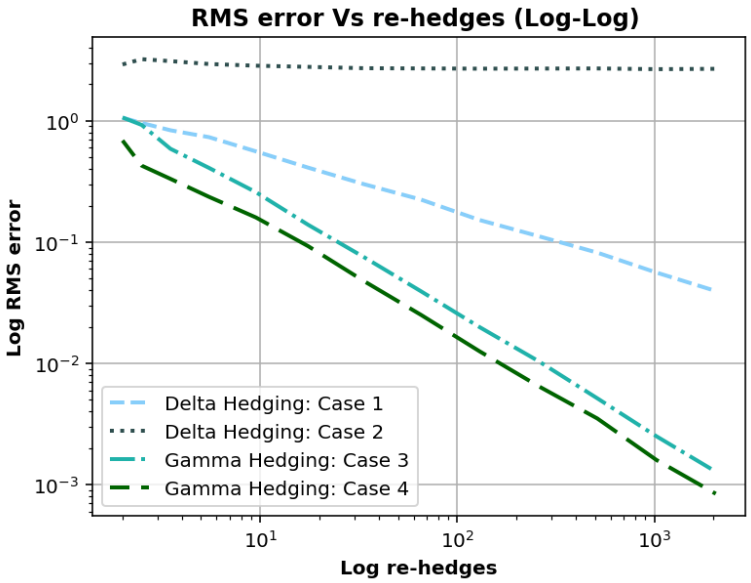
Extra Graphs

This extra graph is a log-log plot of the RMS errors Vs re-hedging for a half period hedge of Cases 1, 2, 3 and 4

In [34]:
```
# Plotting extra graph

plt.figure(figsize=(6,4.5))
ax = plt.gca()
ax.grid()

n_points = 13
n_steps = np.zeros(n_points)
rms_error1 = np.zeros(n_points)
rms_error2 = np.zeros(n_points)
rms_error3 = np.zeros(n_points)
rms_error4 = np.zeros(n_points)
for i in range(0,n_points):

    # steps
    n_steps[i] = 2**i+3

    # case 1 delta hedging
    S, Pi, error1 = simulate_delta_headging(market_model(), option(),
1000, int(n_steps[i]), "variable", "s2", "euler", "C")
    error1 = Pi[:, int(n_steps[i]/2)] - BS_call(market_model(),
option(), S[:, int(n_steps[i]/2)], 0.5*1.6, "s2")

    # case 2 delta hedging
    S, Pi, error2 = simulate_delta_headging(market_model(), option(),
1000, int(n_steps[i]), "variable", "fixed", "explicit", "C")
    error2 = Pi[:, int(n_steps[i]/2)] - BS_call(market_model(),
option(), S[:, int(n_steps[i]/2)], 0.5*1.6, "fixed")

    # case 3 gamma hedging
    S, Pi, error3 = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, int(n_steps[i]), "fixed", "fixed",
"explicit", "C")

    # case 4 gamma hedging
    S, Pi, error4 = half_simulate_gamma_delta_headging(market_model(),
option(), option_H(), 1000, int(n_steps[i]), "variable", "fixed",
"explicit", "C")

    # rms error
    rms_error1[i] = np.sqrt(np.mean(error1**2))
    rms_error2[i] = np.sqrt(np.mean(error2**2))
```

```
    rms_error3[i] = np.sqrt(np.mean(error3**2))
    rms_error4[i] = np.sqrt(np.mean(error4**2))

# plots case 1
ax.loglog(n_steps/2, rms_error1, color="lightskyblue",
linestyle="dashed", linewidth=2)
# plot case 2
ax.loglog(n_steps/2, rms_error2, color="darkslategrey",
linestyle="dotted", linewidth=2)
# plot case 3
ax.loglog(n_steps/2, rms_error3, color="lightseagreen",
linestyle="dashdot", linewidth=2)
# plot case 4
ax.loglog(n_steps/2, rms_error4, color="darkgreen", linestyle=(5, (10,
3)), linewidth=2)
# plot labels
ax.set_xlabel("Log re-hedges", fontweight="bold")
ax.set_ylabel("Log RMS error", fontweight="bold")
ax.set_title("RMS error Vs re-hedges (Log-Log) ", fontweight="bold")
ax.legend(["Delta Hedging: Case 1", "Delta Hedging: Case 2", "Gamma
Hedging: Case 3", "Gamma Hedging: Case 4"])
```

Out[34]: <matplotlib.legend.Legend at 0x7f7de6864c70>



B Individual Question

Individual Question for Student with k-number k23110623

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 $S_0 = 105.00$, $\mu = 0.12$ is a constant, $\sigma(t)$ is given by the deterministic function

$$\sigma(t) = \begin{cases} 0.13 + 0.13 \times \frac{t-0.50}{T-0.50} & t > 0.50 \\ 0.13 & \text{otherwise} \end{cases},$$

and W_t is a Brownian motion. Suppose that there is also a risk-free asset which grows at the continuously compounded rate $r = 0.04$.

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 $K = 139.00$ and maturity $T = 1.60$ 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	k23110623
Price P Confidence interval for p^{Δ} Confidence interval for p^{Γ}	