Comparing Methodologies for Pricing Barrier Options

FE 620: Pricing and Hedging
Final Project

Written By:
Amod Lamichhane

Andre Sealy

Jeffrey Gebauer

Contents

Chapter 1

Barrier Options

1.1 Background

Barrier options are path-dependent options with price barriers; their price depends on whether the underlying asset's price reaches a certain level during a specific period. Various types of barrier options regularly trade over-the-counter and have done so since 1967 [1]. These exotic options were developed to address the specific hedging concerns and market conditions that European and American options failed to accommodate. Barrier options are very popular for their risk-management solutions, as they allow investors and institutions to take various positions with very specific levels of protection.

As financial markets continued to evolve in the 1990s, barrier options became more standardized and accessible to the financial population. Derivative exchanges and financial institutions offered barrier options on various underlying assets, such as commodities, currencies, and interest rates. The 2008 global financial crisis sparked a renewed interest in derivative products for their risk-management capabilities, and barrier options remained one of the best products for their ability to tailor risk profiles to specific market conditions. In addition, computational tool advancements make pricing these options more manageable, thereby increasing the accessibility to market participants. This report will outline the many analytical and computational tools for pricing barrier options.

1.2 Options Payoffs

For some background, we will briefly discuss the attributes of a vanilla option. A call option gives the holder the right, but not the obligation, to buy a particular number of the underlying assets in the future for a pre-agreed price known as the strike price (put options give the holder the right, but not the obligation, to sell). While European options can only be exercised on the expiration date, American options allow the holder to exercise at any time on or before the expiration date. We will focus on European options throughout this research report.

Let S be the price of an underlying asset and K be the strike price, where $S, K \in \mathbb{R}^+$. Then the payoff for a vanilla call option, V_c, is given by the following

$$V_c(S,T) = \begin{cases} S_T - K & \text{if } S_t > K, \forall t \in [0,T) \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

Likewise, the payoff for a vanilla put option, V_p, derived by the following:

$$V_p(S,T) = \begin{cases} 0 & \text{if } S_t > K, \forall t \in [0,T) \\ K - S_T & \text{otherwise} \end{cases} \tag{1.2}$$

These formulas drive our intuition of how both vanilla and exotic options can be priced.

1.3 Barrier Option Payoffs

As we can see, the payoff of a vanilla option depends only on the terminal value of the underlying asset. However, an exotic option, such as a barrier option, is very different. Its price is determined by whether the underlying asset's price reaches a certain level during a specific period. Barrier options differ from standard vanilla options in several ways.

First, they match the hedging needs more closely than standard options; second, premiums for barrier options are typically lower than vanilla options; and finally, the payoff of a barrier option matches beliefs about the future behavior of the market. These features benefit many different types of investors, regardless of experience or financial needs. Another significant difference between barrier options and vanilla options is that barrier options are path-dependent. This means that the payoff depends on the process of the underlying asset. Another difference involves the possibility of a rebate. A rebate is a positive discount that a barrier option holder may receive if the barrier is never reached. For the purpose of outlining the analytical framework, we will not discuss rebates.

There are four different types of thresholds, or barriers, to consider which are:

- down-and-out
- up-and-out
- down-and-in
- up-and-in

Combined with calls and puts, we have 8 different types of barrier options in total. The payoff for a barrier option is either "knocked out" or "knocked in" if the price of the underlying crosses the barrier.

For example, let B be the barrier threshold and S_0 be the price of the underlying asset at time $t = 0$. Then, for any K the down-and-out call option with constant barrier $B < S_0$ has a payoff if the underlying prices stays below the barrier value until maturity T:

$$\begin{cases} (S_T - K)^+ & \text{if } S_t > B, \forall t \in [0,T) \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

An up-and-out call option with constant barrier $B > S_0$ has a payoff if the underlying price does not go beyond the barrier value until maturity T:

$$\begin{cases} (S_T - K)^+ & \text{if } S_t < B, \forall t \in [0,T) \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

A down-and-in call call option with a constant barrier $B < S_0$ has a payoff if the underlying prices stays below the barrier value until maturity T:

$$\begin{cases} 0 & \text{if } S_t > B, \forall t \in [0, T) \\ (S_T - K)^+ & \text{otherwise} \end{cases} \tag{1.5}$$

An up-and-in call option with a constant barrier $B < S_0$ has a payoff if the underlying prices stays beyond the barrier value until maturity T:

$$\begin{cases} 0 & \text{if } S_t < B, \forall t \in [0, T) \\ (S_T - K)^+ & \text{otherwise} \end{cases} \tag{1.6}$$

There are two main approaches to analytically evaluating the price of a barrier option: the probability method and the partial differential equation (PDE) method. The probability method involves the use of the reflection principal and the Girsanov theorem to estimate the barrier densities. The PDE approach is derived from the intutition that all barrier options satisfy the Black-Scholes PDE but with different domains, expiry conditions, and boundary conditions. Merton was the first to price barrier options using the PDE method, which he used to obtain the theoretical price of a down-and-out call option by using the PDE method to obtain a theoretical price.

Chapter 2

Analytical Solutions for Barrier Options

There are closed-form solutions for pricing European-style barrier options. This means we have an explicit mathematical expression that can be used to compute the value of a function without the need for numerical solutions. However, we will continue to compare closed-form solutions to more rigorous methodologies, such as Monte Carlo, and Lattice (Binomal Trees). Unlike their continuous counterparts, no closed-form solutions exist for discrete-time barrier options (even numerical pricing is a challenge). For this reason, we will only focus on continuous-time, single-barrier options. All of the options that we analyze will be European options unless stated otherwise.

2.1 The Black-Scholes Model

The Black and Scholes model was first published in 1973, named after the two economist who helped to develop it: Fischer Black and Myrion Scholes. (the model is formally known as the Black-Scholes-Merton model) A rigorous derivation of the Wiener process, Ito's lemma, the portfolio process at the risk-free rate gives us the following equation

$$\frac{1}{2}\sigma^2 S^2 \frac{\partial^2 f}{\partial S^2} + rS\frac{\partial f}{\partial S} - \frac{\partial f}{\partial} - rf = 0 \tag{2.1}$$

From here, we solve equation (2.1) to arrive at the following equation

$$f(S,t) = Se^{-qT}N(d_1) - Ke^{-rT}N(d_2) \tag{2.2}$$

where S is the stock price, K is the strike price, r is the risk-free rate, T is the time to expiration, σ is the volatility of the stock, $N(\cdot)$ is the cumulative distribution function, and d_1/d_2 are derived by the following:

$$d_1 = \frac{\ln\left(S_0/K\right) + \left(r + \sigma^2/2\right)T}{\sigma\sqrt{T}}, \quad d_2 = \frac{\ln\left(S_0/K\right) + \left(r - \sigma^2/2\right)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T} \tag{2.3}$$

A more rigourous proof for the solution to the Black-Scholes PDE can be found on A.1 of the appendix.

2.2 Analytical Solution to Barrier Options

Using the example of an up-and-out call barrier option, we start by changing the value of T for $\tau = T - t$. From there, in order to have the PDE solutioni for the up-and-out call option, we begin to alter equation (2.2). Let H be the barrier price. Then when $B \geq K$, we have:

$$C_{\text{up-out}}(S,t) = Se^{-q\tau}\left(N(d_1) - \left(\frac{B}{S}\right)^{2\lambda} N(d_1')\right) - Ke^{-r\tau}\left(N(d_2) - \left(\frac{B}{S}\right)^{2\lambda-2} N(d_2')\right) \tag{2.4}$$

where

$$\lambda = \frac{r-q}{\sigma^2} + \frac{1}{2} \tag{2.5}$$

and d_1'/d_2' is derived by the following

$$d_1' = \frac{\ln\left(\frac{B^2}{SK}\right) + (r - q + \frac{1}{2}\sigma^2)\tau}{\sigma\sqrt{T}}, \quad d_2' = d_1' - \sigma\sqrt{\tau} \tag{2.6}$$

If $S \geq B$ at any time before expiration, the up-and-out call ceases to exist (it is knocked out). If $S < B$ for the entire option's life, the payoff at maturity is just like a standard call, where the payoff is $\max(S_T - K, 0)$

2.3 In-Out Parity

An interesting consequence from the analytical solution to barrier options is the relationship between knock-in and knock-out barrier options. As such, in the same way there is a parity relationship between vanilla puts and calls, there is a parity relationship between knock-out calls/puts and knock-in calls/puts. We refer to this relationship as the up-out/up-in call/put parity. For this example, we will refer to the up-in and up-out call option.

The parity between the up-in and up-out call option demonstrates that the price of an up-and-in call option can be expressed by the following:

$$C_{\text{BSM}} = C_{\text{up-in}} + C_{\text{up-out}} \tag{2.7}$$

For example, consider a call option with $S = 100, K = 105, r = 10\%, \sigma = 20\%,$ and $T = 1$. Also, consider a barrier call option with all of the same parameters, except with a barrier level, $B = 120$. Using equations (2.2), (2.4), and (2.7) we have the following parity:

$$\begin{aligned} C_{\text{BSM}} &= C_{\text{up-in}} + C_{\text{up-out}} \\ 10.521 &= C_{\text{up-in}} + 0.52 \\ 9.99 &= C_{\text{up-in}} \end{aligned}$$

This parity holds under the assumption that no dividends are paid, and other assumptions like interest rates are constant. This relationship works because the call options under the Black-Scholes formula represents a basic option that can be exercised at any point before expiration. The up-and-out call option represents the vanilla call option with a barrier that expires worthless if the underlying asset breaks the barrier. The up-and-in call represents the same option but with a delayed start; only becoming active if the price breaks the barrier.

Since the up-and-in option only activates once the barrier is breached, it's value is lower than the vanilla call option. Using this intuition, we can adjust the up-and-out call value based on the price moving above the barrier. This intuition works for down-in/down-out, as well as puts. As such, we can always find the value of one of the options as long as we know the value of the vanilla option and it's payoff pair (up/down and in/out).

2.4 Surface of the Barrier Option

Barrier Pricing for At-The-Money Options

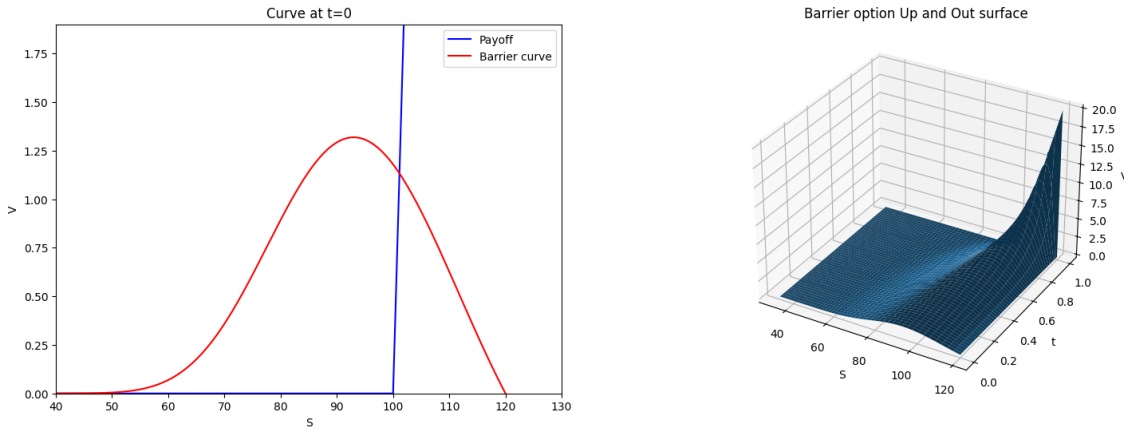

Figure 2.1: At-the-money up-and-out barrier option

Figure (2.1) shows the barrier curve and surface for an at-the-money up-and-out barrier option for $S = 100, K = 100, T = 1, r = 10\%, \sigma = 20\%$, and $B = 120$. The left plot shows the value of an up-and-out barrier option (y-axis) in relation to the underlying (x-axis). The right plot shows a 3-dimensional graphical representation of the valuation of the option (z-axis) in relation to the underlying (x-axis) and time (y-axis). The plot on the left assumes the option is at expiration.

As we can see, the option has no value at any price below 100, as the strike price is also 100. However, the call will retain some value, as the option has not reached the barrier of 120. Where the payoff line (blue) and the barrier curve (red) intersects shows the potential payoff at expiration, which is 1.18 according to Black-Scholes. The surface shows the option value in relation to time and the underlying price. As we can see, the barrier option has the most value when the option is very close to expiration and the stock price is relatively close to the barrier. Otherwise, the option has no value when the underlying is at 40 and the time to expiration is 0. The option also loses it's value when the option is close to the barrier.

Relationship With Black-Scholes

Figure (2.2) shows the same surface plot as Figure (2.2), except with one small alteration: we've increased the barrier to an arbitrarily high level of 250. An interesting consequence of having a barrier level that is too far away from the underlying is that the barrier option price will mirror the Black-Scholes price. This relationship holds for barrier options where $B < S$ (down-and-out, down-and-in), as well as barrier options where $B > S$ (up-and-out, up-and-in).

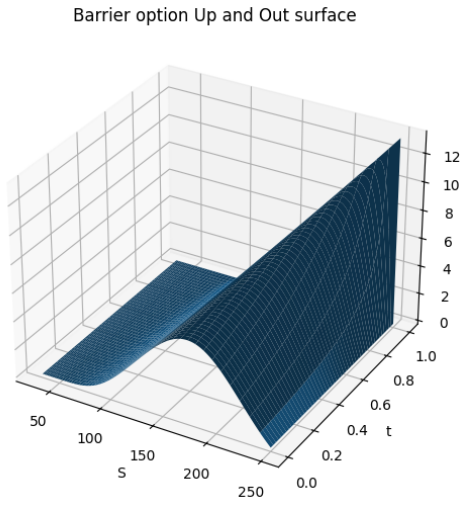

Figure 2.2: At-the-money up-and-out barrier option

Recall that this surface still reflects an at-the-money call option, with $S = 100, K = 100, T = 1, r = 10\%, \sigma = 20\%$, and $B = 120$. When S is far away from K, we see that the option has almost no value (approximately 0.01) when $T = 1$;, but loses all of its value for any $T < 1$. Once the underlying reaches the strike price, the option will reflect the price of the Black-Scholes, which is 13.27. Of course, the option loses all of its value once it reaches the barrier threshold of 250.

2.5 Barrier Option Payoffs

With eight different types of single barrier options comes eight possible payoffs, based on the barrier price. Table (2.5) shows the payoff based on whether the barrier is up or down, whether the stock price in or outside of the barrier, and whether the option type is a call or put. Refer to Appendix A.2

Table 2.1: Theoretical Values of Single Barrier Options

Down/Up	In/Out	Call/Put	Payoff ($K \leq B$)	Payoff ($K \geq B$)
Down	In	Call	$A_1 - A_2 + A_4 + A_5$	$A_3 + A_5$
Up	In	Call	$A_2 - A_2 + A_4 + A_5$	$A_1 + A_5$
Down	In	Put	$A_1 + A_5$	$A_2 - A_3 + A_4 + A_5$
Up	In	Put	$A_3 + A_5$	$A_1 - A_2 + A_4 + A_5$
Down	Out	Call	$A_2 - A_4 + A_6$	$A_1 - A_3 + A_6$
Up	Out	Call	$A_1 - A_2 + A_3 - A_4 + A_6$	A_6
Down	Out	Put	A_6	$A_1 - A_2 + A_3 - A_4 + A_6$
Up	Out	Put	$A_1 - A_3 + A_6$	$A_2 - A_4 + A_6$

Throughout this report, we will be deriving our analysis from the up-and-out call and put option, since it is easier to intuitively understand. The payoffs from equations (1.1) and (1.2), still hold for vanilla calls and puts. However, recall for a knocked-out up option, the option losses value once it has reaches the barrier above the underlying stock price.

Chapter 3

Barrier Option Pricing with Monte Carlo

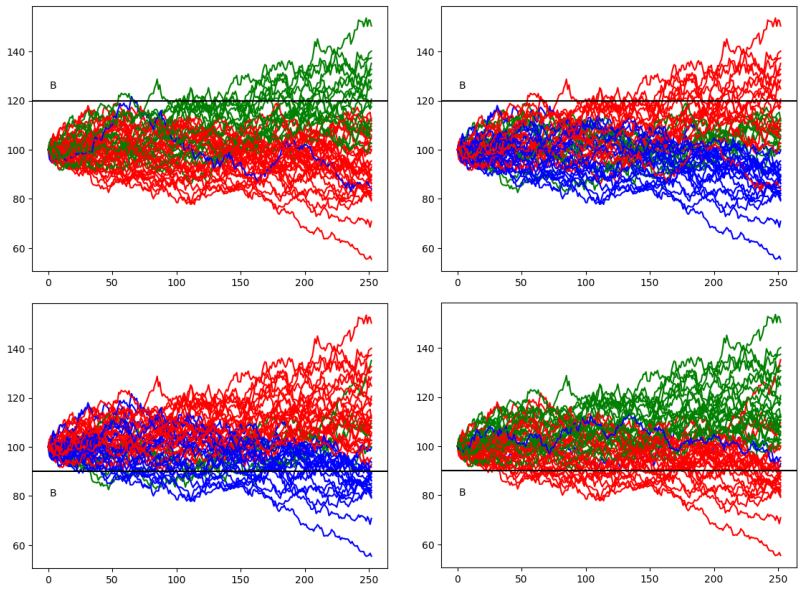

Figure 3.1: Simulated Paths Barrier Option Prices using Monte Carlo

Suppose the asset price follows a Geometric Brownian Motion (GBM):

$$dS_t = rS_t dt + \sigma S_t dW_t \tag{3.1}$$

Where S_t is the asset price at time t, r is the risk-free interest rate, σ is the volatility of the asset, and dW_t is the increment of a Wiener process. With this process in mind, we can discretize time by dividing the total time into smaller length intervals $\Delta t = T/N$. Afterwards, we simulate the asset price paths by generating a random standard normal variable Z and using the random variable to update the asset price with the discretized GBM.

$$S_{t+\Delta t} = S_t \times \exp\left(\left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t} \times 2\right) \tag{3.2}$$

The Monte Carlo simulation is a tool that we can implement to analyze models like GBM. Monte Carlo is a computational algorithm that uses repeated random sampling to obtain numerical results. It is used to predict the behavior of complex systems that are difficult to model analytically (while European Barrier options have an analytical solution, we will still incorporate Monte Carlo). In the context of GBM, Monte Carlo simulations are used to simulate many possible paths of stock prices over time under the assumptions of GBM. This allows for exploring statistical outcomes and quantifying risk in a financial context, where analytical solutions may be challenging to reproduce.

Figure (3.1) shows the simulated Monte Carlo paths for all of the barrier call options: up-and-in (top left), up-and-out (top right), down-and-in (bottom left), and down-and-out (bottom right). We assume the $S = 100, K = 100, T = 1, r = 5\%,$ and $\sigma = 20\%$. We use a barrier, B, of 120 for the up-in/up-out options and 90 for the down-in/down-out options. We highlight the path's different colors based on the outcome at the expiration date.

Take the up-and-in (top-left) barrier option, for example. If the simulated stock path breaches the barrier and stays above the strike price, K, we highlight the path as green. If the stock breaches the barrier but ends below the strike price, we highlight the path as blue (the option becomes exercisable but expires out-of-the-money). All other paths will be highlighted in red if the simulated paths never breach the barrier level. Other simulations utilize the same criteria for their path directions based on the characteristics of the option.

We have only used 100 simulations, as it is easier to display the different outcomes. Regardless, we can see that we can consistently arrive at an option price that mirrors the analytical model simply by averaging the outcomes where $K < S < B$, as we are working with an up-and-out.

3.1 Monte Carlo Simulation

Table (3.1) shows the comparison between the Black-Scholes analytical price and the Monte Carlo simulations for the up-and-out call options. We have chosen options with different Barrier values as well as varying stock values.

Table 3.1: MC Up-and-out call with $q = 0\%, r = 10\%, T = 1, \sigma = 20\%, K = 100$

	Price	MC_{100}	err_{100}	MC_{1000}	err_{1000}	MC_{5000}	err_{5000}	MC_{10000}	err_{10000}
$S_0 = 90$	1.2925	1.8113	-0.5188	1.4851	-0.1926	1.3249	-0.0324	1.3525	-0.06
$B = 120$	(7.3823)	(8.0324)	(-0.6524)	(7.5403)	(-0.1603)	(7.4427)	(-0.0627)	(7.3643)	(0.0157)
$S_0 = 90$	2.9799	3.2746	-0.2947	3.0758	-0.0960	3.0351	-0.0552	2.9991	-0.0192
$B = 130$	(7.5049)	(6.7301)	(0.7748)	(7.9160)	(-0.4111)	(7.4177)	(0.0872)	(7.4826)	(0.0222)
$S_0 = 100$	1.1789	1.5790	-0.4001	1.0808	0.0981 (-	1.2127	-0.0338	1.2060	-0.0271 (-
$B = 120$	(3.5932)	(4.0435)	(-0.4503)	(3.9701)	0.3763)	(3.6234)	(-0.0302)	(3.5990)	0.0058)
$S_0 = 100$	3.5369	4.6782	-1.1413	3.2785	0.2584 (-	3.4410	0.0959 (-	3.5303	0.0066 (-
$B = 130$	(3.7432)	(4.4019)	(-0.6587)	(3.8816)	0.1384)	(3.7508)	0.0076)	(3.7498)	0.0066)
$S_0 = 110$	0.6264	0.7961	-0.1697	0.7819	-0.1555	0.6422	-0.0159	0.6551	-0.0362
$B = 120$	(1.3437)	(1.5428)	(-0.1991)	(1.2988)	(0.0449)	(1.3553)	(-0.0116)	(1.3396)	(0.0041)
$S_0 = 110$	2.9014	2.4881	0.4133	3.1815	-0.2801	2.9130	-0.0116	2.9700	-0.0686
$B = 130$	(1.6735)	(1.5780)	(0.0955)	(1.8227)	(-0.1492)	(1.7561)	(-0.0826)	(1.6167)	(0.0568)
$S_0 = 110$	13.2660	13.4717	-0.2057	13.0827	0.1833 (-	13.1939	0.0721	13.2129	0.0531 (-
$B = 250$	(3.7534)	(4.6529)	(-0.8995)	(4.2545)	0.5011)	(3.5797)	(0.1737)	(3.7706)	0.0172)

These results will be compared with those obtained through alternative variations of Monte Carlo methods. We've used simulations $n = 100$, the same number as in Figure (3.1), but we also use simulations where $n = 1000, 5000,$ and 10000, as well. The errors listed in the tables represent the price deviation from the analytical values. The values in the parenthesis denote the outcome for puts, while the values without parenthesis denote the values for calls. As we can see, the Monte Carlo price gets closer to the analytical solution for $n = 5000$. With the table, we confirm one main aspect of Monte Carlo theory: increasing the number of simulations leads to an improved accuracy for the computation.

We have also used a simulation with an arbitrarily high barrier of 250 and the results simulate what we can expect from the Monte Carlo method, as well as the analytical method. When the barrier is very far away from the underlying price, the option value will mirror the Black-Scholes price.

Chapter 4

Binomial Tree Method

The *binomial tree method* is helpful for pricing derivative instruments, including barrier options. It works by breaking down the time from the current moment ($t = 0$) to the option's expiration, T into a finite number of steps N. The more steps you use, the more accurate the price becomes, eventually converging to the theoretical option price.

4.1 Binomial Tree Structure

A binomial tree models the possible price movements of the underlying asset over time. At each step, the asset price can either go up or down. The price at any node in the tree is calculated using the formula:

$$S_{i,j} = S_0 \cdot u^j \cdot d^{i-j}$$

where:

- $S_{i,j}$ is the stock price at step i, level j,
- S_0 is the initial stock price,
- $u = e^{\sigma\sqrt{\Delta t}}$ is the up factor,
- $d = \frac{1}{u}$ is the down factor,
- $\Delta t = \frac{T}{N}$ is the time increment per step,
- σ is the volatility of the underlying asset.

The risk-neutral probabilities of an upward movement (p) and a downward movement (q) are:

$$p = \frac{e^{r\Delta t} - d}{u - d}, \quad q = 1 - p$$

where r is the risk-free interest rate.

4.2 Example Calculation

Consider an up-and-out put option with these parameters: $S_0 = 100$, $B = 120$, $K = 100$, $r = 0.05$, $T = 1$ year, $\sigma = 0.2$, $N = 3$.

- The time increment per step is:

$$\Delta t = \frac{T}{N} = \frac{1}{3} \approx 0.3333 \,\text{years}$$

- The up and down factors are:

$$u = e^{\sigma\sqrt{\Delta t}} = e^{0.2\sqrt{0.3333}} \approx 1.1224, \quad d = \frac{1}{u} \approx 0.8909$$

- The risk-neutral probabilities are:

$$p = \frac{e^{r\Delta t} - d}{u - d} = \frac{e^{0.05 \cdot 0.3333} - 0.8909}{1.1224 - 0.8909} \approx 0.5438, q = 1 - p = 0.4562$$

- The payoff for an up-and-out barrier put option at maturity is defined as:

$$\text{Payoff} = \begin{cases} \max(K - S_T, 0), & \text{if } \max(S_t) < B \\ 0, & \text{if } \max(S_t) \geq B \end{cases}$$

  where $t \in [0, T]$.

4.3 Tree Construction and Results

Figure (4.1) illustrates the binomial tree for the given parameters, showing the possible stock price movements and the resulting option values at each node. Considering the up-and-out barrier condition, the calculated option price is obtained through backward induction along the tree. This ensures that the barrier condition is applied at each step. The final option price using this method is 6.17. For reference, the actual option price using the analytical formula is 5.36.

The binomial tree method is excellent for pricing various options, including barrier options, because it's intuitive and systematic. As shown in Figure (4.2), the option price gets closer to the analytical solution as the number of steps, N, increases. However, this accuracy comes with higher computational complexity, especially for path-dependent options. In such cases, methods like Monte Carlo simulations or analytical approaches can be more efficient and scalable, providing accuracy without the heavy computational load. Flexibility and efficiency are crucial when choosing the proper pricing method for complex financial derivatives.

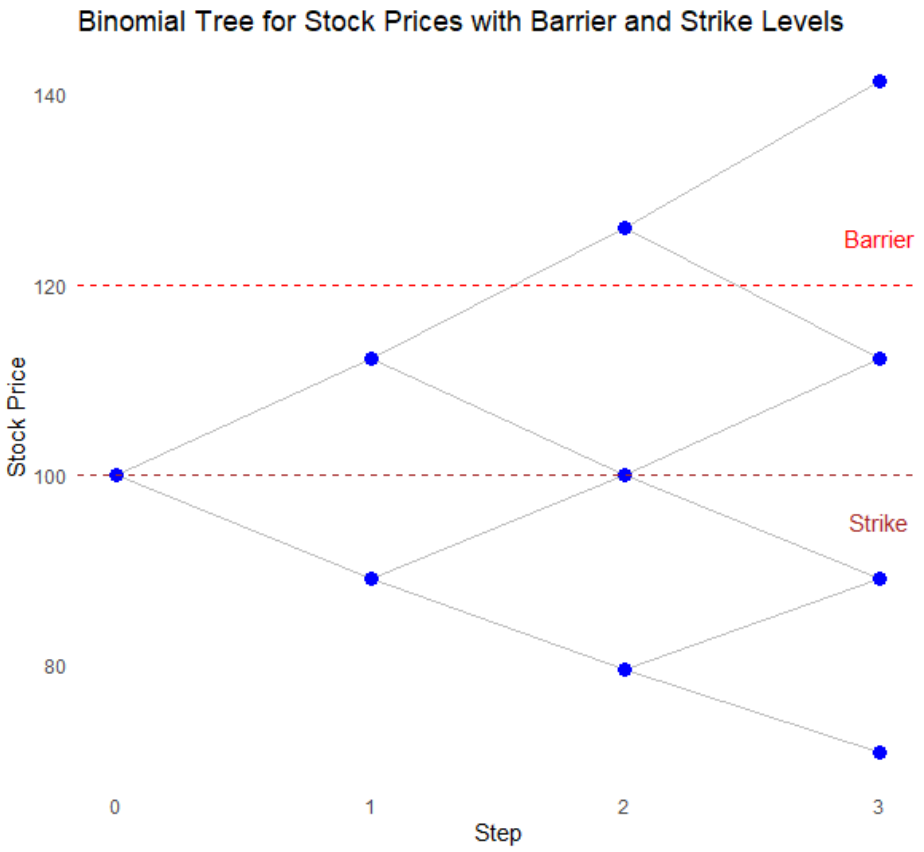

Figure 4.1: Binomial Tree with Barrier Level $B = 120$ and Strike Price $K = S_0$.

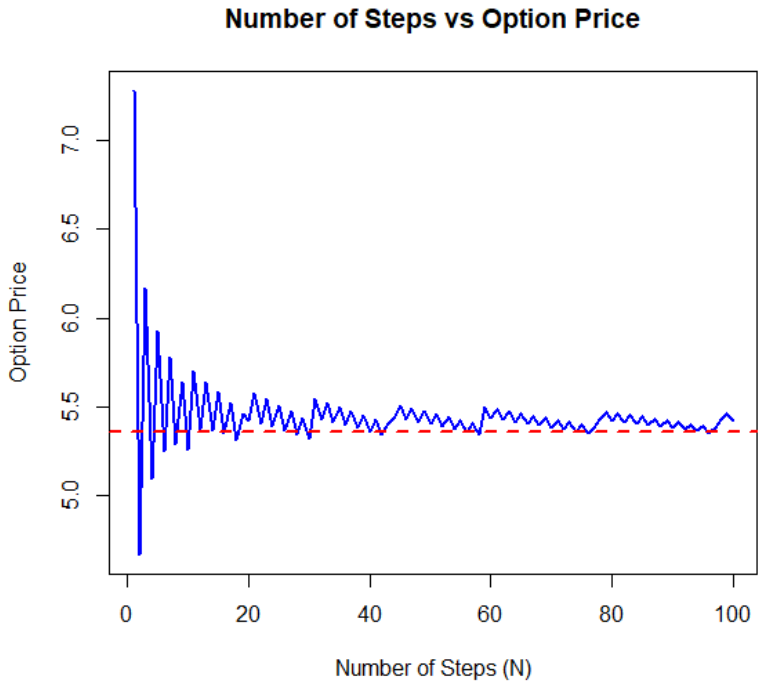

Figure 4.2: Option price converge to analytical solution as number of steps rise.

Chapter 5

The Greeks for Barrier Options

The Greeks are key sensitivities in option pricing that measure how the price of an option changes with respect to various factors such as the underlying asset price, time to maturity, volatility, and interest rates. For barrier options, the calculation of the Greeks is more complex due to the added condition of a barrier, which affects the option's price path and behavior. For the plots in this section, we use assume $S = 40, K = 50, T = 1.0, r = 10\%, \sigma = 20\%,$ and $B = 60$.

5.1 Delta (Δ)

Delta for a barrier option measures the sensitivity of the option price to changes in the underlying asset price S. It represents the rate of change of the option's price with respect to small changes in the underlying price.

$$\Delta = \frac{\partial V}{\partial S}$$

For **knock-in** barrier options, the delta behaves similarly to that of a standard European option but is influenced by the presence of the barrier. It reflects how changes in the underlying price affect the probability of the barrier being breached and the option becoming active. Figure (5.1) shows the delta behavior of an up-and-out put option as a function of the underlying stock price, S. When the stock price is well below the barrier level of B, the delta behaves similarly to a vanilla European put option. As the price approaches the barrier, the delta tends toward zero because the likelihood of the option being knocked out increases, reducing its sensitivity.

To further analyze the delta across different barrier options, refer to the table below:

Table 5.1: Delta behavior for different types of barrier options.

Barrier Type	**Price Far from Barrier**	**Price Near the Barrier**	**Intuition**
Knock-Out	Similar to vanilla options	$\Delta \to 0$ as $S \to B$	Probability of knock-out increases as S nears B.
Knock-In	$\Delta \approx 0$ far from barrier	$\Delta \uparrow$ as $S \to B$	Option becomes active as the price hits the barrier.

Key Intuition:

- **Knock-Out Options:** Lose their sensitivity as Δ tends toward 0 as the likelihood of being knocked out near the barrier increases.
- **Knock-In Options:** Show increasing sensitivity as the underlying price nears the barrier because the probability of activation rises.

The visual analysis in Figure (5.1) shows delta behavior for an up-and-out put option. The insights from the plot match the intuition from Table (5.1), which provides a comprehensive understanding of how delta behaves across different types of barrier options. Before the stock price hits the barrier, the delta is similar to one of a vanilla option. As the stock price hits the barrier, there is a steep decline in delta as the option is knocked-out and the option price is zero.

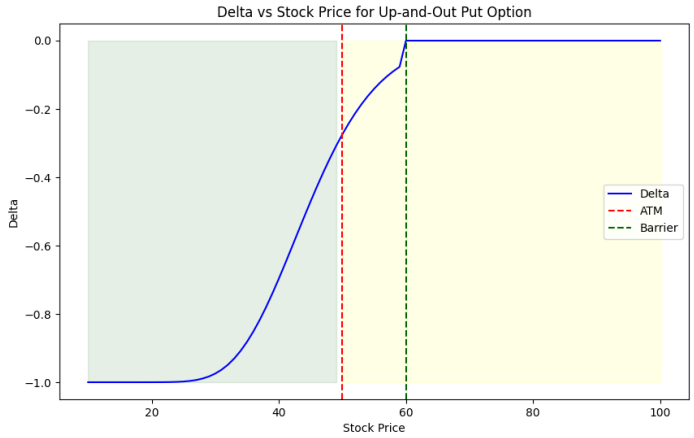

Figure 5.1: Delta of up-and-out Put Option vs. the Stock Price.

5.1.1 Delta Hedging

In this section, we retrieve NVDA prices from October 21-25, 2024, using the parameters $K = 143, T = 1, r = 4.6\%, \sigma = 50\%,$ and $B = 150$, which represent the strike price, time to maturity, risk-free rate, volatility, and barrier level, respectively. We calculated the option price and its delta. The unhedged and hedged changes are presented in Table (5.2), demonstrating the effectiveness of delta hedging in minimizing investment risks. Figure (5.2) provides a visual comparison of hedged versus unhedged changes from October 22 to 24, 2024; illustrating that hedging reduces risk by minimizing the volatility of the option price.

Table 5.2: Option Pricing and Delta Hedging

Date	Stock Price	Option Price	Delta	Unhedged Change	Hedged Change
October 21	143.71	24.23653	-0.3624546	NA	NA
October 22	143.59	24.28006	-0.3630813	0.04353214	3.759243×10^{-5}
October 23	139.56	25.78639	-0.3846464	1.50633247	4.311500×10^{-2}
October 24	140.41	25.46142	-0.3800139	-0.32497748	1.971995×10^{-3}
October 25	141.54	25.03545	-0.3739251	-0.42596805	3.447701×10^{-3}

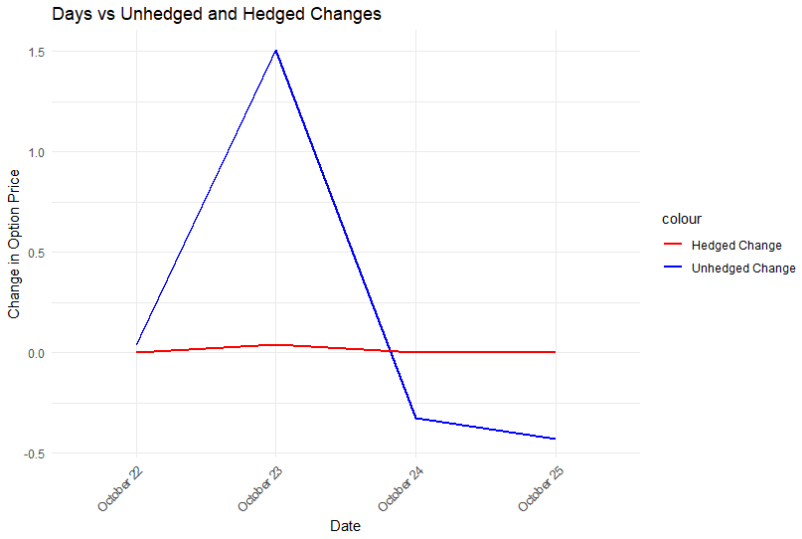

Figure 5.2: Comparison of change in Option Price of a hedged investment vs unhedged investment

5.2 Gamma (Γ)

Gamma (Γ) for a barrier option measures the rate of change of delta (Δ) with respect to changes in the underlying asset price (S). It represents the curvature or sensitivity of delta in response to movements in the underlying price. It provides insights into how rapidly an option's delta will change as the underlying price fluctuates.

$$\Gamma = \frac{\partial^2 V}{\partial S^2}$$

For **knock-in** and **knock-out** barrier options, gamma exhibits unique behavior due to the presence of the barrier. The interaction between the price of the stock, the barrier level and the time to maturity influences how gamma behaves in different scenarios.

Table 5.3: Gamma behavior for Knock-In and Knock-Out barrier options.

Barrier Type	**Price Far from Barrier**	**Price Near the Barrier**	**Intuition**
Knock-Out	Similar to vanilla options	$\Gamma \uparrow$ as $S \to B$	Gamma spikes as the price nears the barrier due to increased risk of knock-out.
Knock-In	$\Gamma \approx 0$ far from barrier	$\Gamma \uparrow$ as $S \to B$	Gamma rises as the stock price approaches the barrier due to increased probability of activation.

Figure (5.3) illustrates the gamma behavior of an up-and-out put option as a function of the underlying stock price (S). As soon as the barrier is hit, the option is knocked out and gamma goes to zero. Gamma hedging is a risk management strategy used in options trading to mitigate the risks associated with large

movements in the underlying asset's price. These large movements may cause large losses in delta hedged portfolio. Figure 5.6 shows that hedging both delta and gamma provides minimum risk investment.

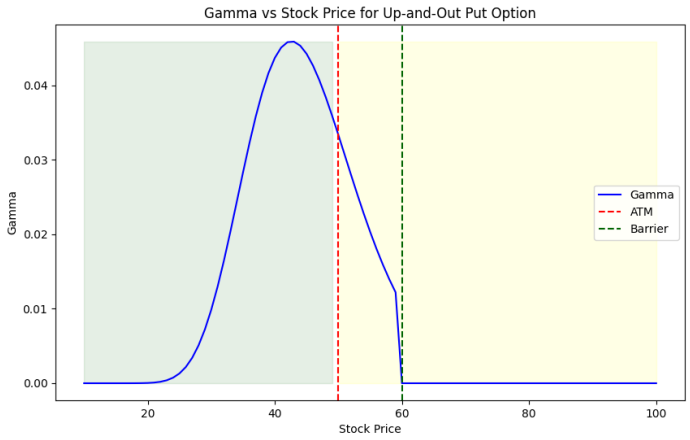

Figure 5.3: Gamma of an up-and-out put option vs. the Stock Price.

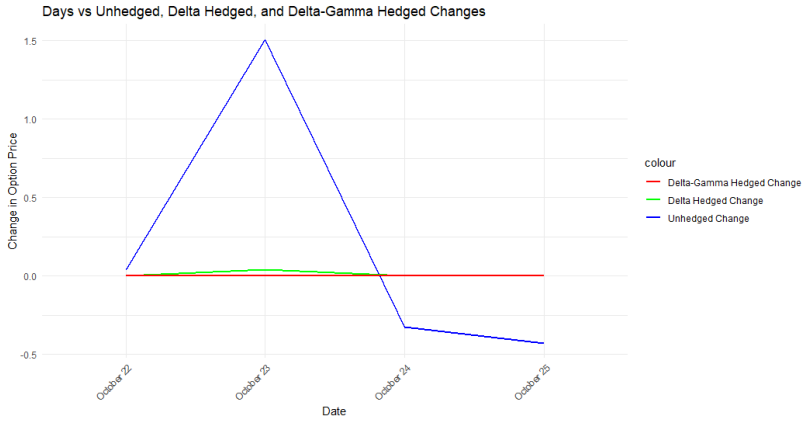

Figure 5.4: Comparing different hedging results

5.3 Vega (ν)

Vega, ν, measures the sensitivity of the option price to changes in volatility. For barrier options, the calculation of vega incorporates adjustments to account for the probability of breaching the barrier level under varying volatility conditions.

$$\nu = \frac{\partial V}{\partial \sigma}$$

- **Knock-In Options:** Increased volatility raises the likelihood of the underlying asset price crossing the barrier level, making the option more valuable. As a result, ν (vega) will generally be positive.
- **Knock-Out Options:** Increased volatility can lead to a higher probability of the option being knocked out, reducing its value. Therefore, ν will tend to be negative for these types of options.

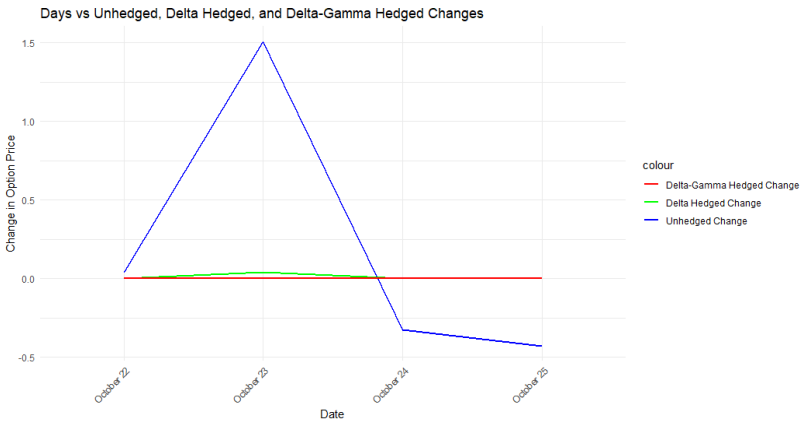

Figure 5.5: Comparing different hedging results

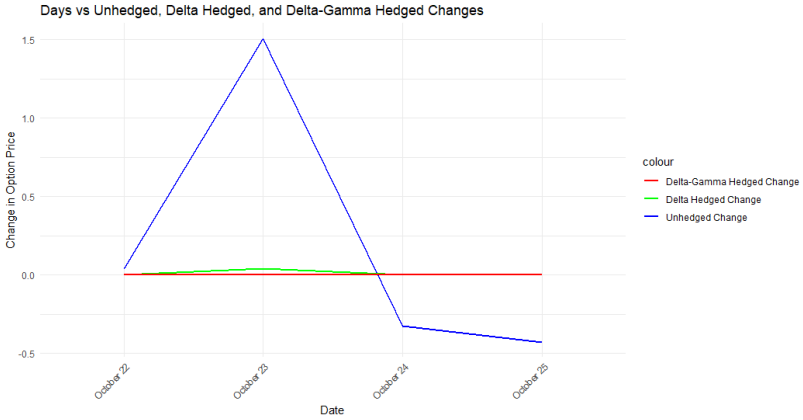

Figure 5.6: Comparing different hedging results

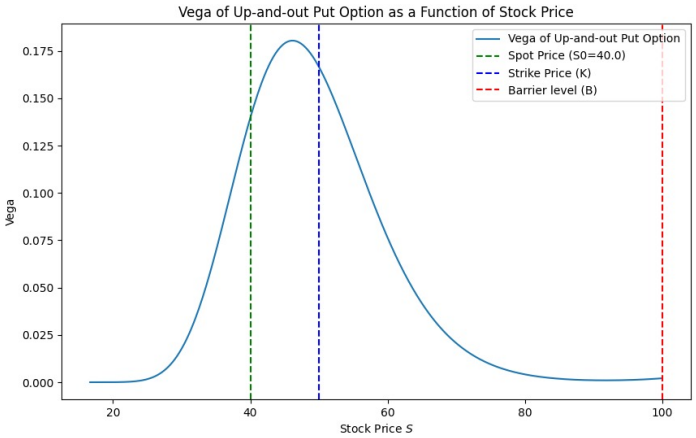

Figure 5.7: Vega vs Stock Price of an Up-and-Out Put Option

The visualization shown in Figure (5.7) provides insights into how Vega behaves with changes in stock price across up-and-out put option. The sensitivity patterns are influenced by the interplay between volatility and the probability of breaching the barrier level. As the barrier is breached, vega immediately goes to zero.

5.4 Theta (Θ)

Theta, Θ, measures the rate of change of an option's price with the passage of time, assuming all other variables remain constant. It is a critical "Greek" that reflects time decay, which is the gradual erosion of the option's value as expiration approaches. In the case of barrier options, theta is influenced not only by the time remaining but also by the interaction with the barrier level and the path dependency of the option.

Barrier options are unique because their payoff depends on the underlying asset price crossing (or not crossing) a specified barrier level during the option's lifetime. As such, the sensitivity of theta changes depending on proximity to the barrier, time remaining until maturity, and whether the option is near the knock-in or knock-out condition.

- **Knock-In Options:** As time to maturity decreases, the value of knock-in options tends to decline, especially when the underlying asset price is far from the barrier. This is because the likelihood of triggering the option by crossing the barrier becomes lower as time runs out.
- **Knock-Out Options:** As time to maturity approaches, the risk of the underlying price reaching the barrier and knocking out the option increases, thereby accelerating time decay.

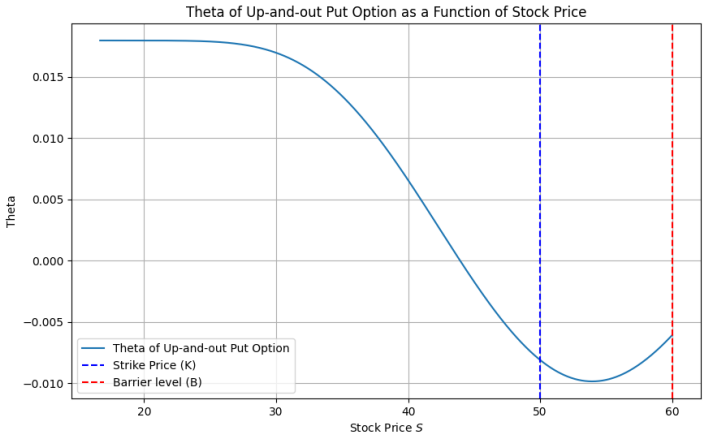

Figure 5.8: Theta vs Stock Price of an Up-and-Out Put Option

The visualization in Figure (5.8) illustrates the relationship between stock price and theta for an up-and-out put option. This analysis highlights how theta's sensitivity varies depending on proximity to the barrier. As the stock price approaches the strike, theta is decreasing. As the stock price approaches the barrier, it is increasing. However, when stock price hits the barrier, theta immediately goes to zero.

5.5 Rho (ρ)

Rho, ρ, for barrier options measures the sensitivity of the option price to changes in the risk-free interest rate (r). It represents how much the value of the option changes when the risk-free rate changes by 1%. Rho is important in understanding the impact of monetary policy, such as changes in interest rates, on the value of barrier options.

$$\rho = \frac{\partial V}{\partial r}$$

When interest rates rise:

- **Knock-In Options:** Tend to show a positive relationship with Rho (ρ). This reflects that higher interest rates increase the present value of the potential payoff, making the knock-in option more valuable.
- **Knock-Out Options:** Tend to show muted sensitivity to changes in the interest rate. This is because the likelihood of being knocked out (and thus losing value) dominates any benefit gained from higher interest rates.

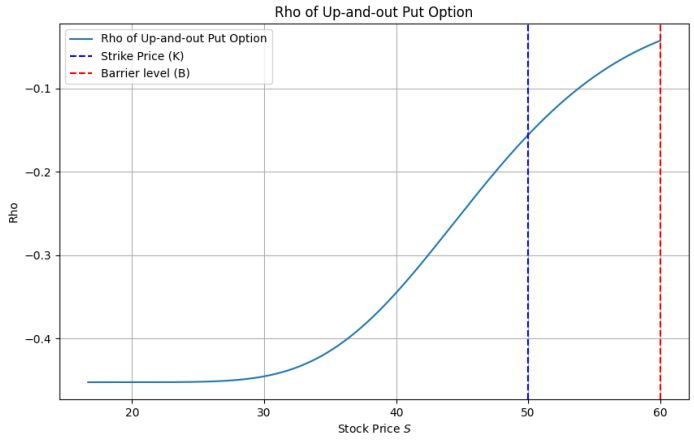

Figure 5.9: Rho vs Stock Price for an Up-and-Out Put Option

The visual behavior in Figure 5.9 illustrates how changes in the stock price (r) impact rho of an up-and-out barrier option. As the stock price increase, rho also increases. However, as stock price breaches the barrier, rho goes to zero.

Bibliography

[1] Cox, John C. *Options Markets.* 1985.

[2] Haug, Espen Gaarder, and Haug, Jørgen Aase. "A Binomial Tree Model for Barrier Options." *Wilmott Magazine,* 2007.

[3] Hull, John and White, Alan. "Dynamic Hedging: Managing Portfolio Risk." *Journal of Finance,* vol. 42, no. 2, 1987, pp. 281–300.

[4] Hull, John C. *Options, Futures, and Other Derivatives.* 10th Edition, Pearson, 2018.

[5] Wilmott, Paul, Howison, Sam, and Dewynne, Jeff. *The Mathematics of Financial Derivatives: A Student Introduction.* Cambridge University Press, 1995.

Appendix A

Formulas and Proofs

A.1 Payoffs Formulas for Single Barrier Options

The formula for Single Barrier options have been proved by Reiner and Rubinstein. The payoff of a Single Barrier call or put depends on the following formulas:

$$
\begin{aligned}
A_1 &= Se^{-q\tau}aN(ax_1) - Ke^{-r\tau}N(ax_1 - a\sigma\sqrt{\tau}) \\
A_2 &= Se^{-q\tau}aN(ax_1) - Ke^{-r\tau}N(ax_2 - a\sigma\sqrt{\tau}) \\
A_3 &= Se^{-q\tau}a\left(\tfrac{B}{S}\right)^{2\lambda+2}N(bx_3) - K\left(\tfrac{B}{S}\right)^{2\lambda}e^{-r\tau}N(bx_3 - b\sigma\sqrt{\tau}) \\
A_4 &= Se^{-q\tau}a\left(\tfrac{B}{S}\right)^{2\lambda+2}N(bx_4) - K\left(\tfrac{B}{S}\right)^{2\lambda}e^{-r\tau}N(bx_4 - b\sigma\sqrt{\tau}) \\
A_5 &= Ke^{-rT}\left[N(bx_2 - b\sigma\sqrt{\tau}) - \left(\tfrac{B}{S}\right)^{2\lambda}N\left(bx_4 - b\sigma\sqrt{\tau}\right)\right] \\
A_6 &= Ke^{-rT}\left[N(bx_5 - b\sigma\sqrt{\tau}) - \left(\tfrac{B}{S}\right)^{2\lambda}N\left(bx_5 - b\sigma\sqrt{\tau}\right)\right]
\end{aligned}
$$

with

$$
\begin{cases}
a = 1, -1 & \text{call or put} \\
b = 1, -1 & \text{out or in}
\end{cases}
$$

where x_1, x_2, x_3, x_4, x_5 is the following:

$$
\begin{aligned}
x_1 &= \frac{\ln\left(\frac{S}{K}\right)}{\sigma\sqrt{\tau}} + (1+\mu)\sigma\sqrt{\tau}, \quad x_2 = \frac{\ln\left(\frac{S}{B}\right)}{\sigma\sqrt{\tau}} + (1+\mu)\sigma\sqrt{\tau} \\
x_3 &= \frac{\ln\left(\frac{B^2}{SK}\right)}{\sigma\sqrt{\tau}} + (1+\mu)\sigma\sqrt{\tau}, \quad x_4 = \frac{\ln\left(\frac{B}{S}\right)}{\sigma\sqrt{\tau}} + (1+\mu)\sigma\sqrt{\tau} \\
x_5 &= \frac{\ln\left(\frac{B}{S}\right)}{\sigma\sqrt{\tau}} + \lambda\sigma\sqrt{\tau}
\end{aligned}
$$

where μ and λ is the following

$$
\mu = \frac{r - q - \frac{\sigma^2}{2}}{\sigma^2}, \quad \lambda = \sqrt{\mu^2 + \frac{2q}{\sigma^2}}
$$

A.2 Code for Monte Carlo Simulated Paths

```
import numpy as np
import pandas as pd
import statistics
import scipy.stats as sp
import time
from datetime import timedelta, date
import warnings
from scipy.stats import norm
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt

def sim_MC_paths(S0, T, r, sigma, n, M):
    """
    Parameters:
    S0: initial price
    T: time to maturity
    r: risk free rate
    sigma: volatility
    n: number of time steps
    M: number of paths
    """
    paths = []
    dt = T / n

    #simulate M paths
    for _ in range(M):
        s = [S0] #declare list of prices
        #increment n timesteps
        for _ in range(n):
            z = np.random.normal(0,1)
            px = s[-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * z) #get
    next price based on prev. price in list
            s.append(px)
        paths.append(s)
    return paths

paths = sim_MC_paths(100,1,0.05,.2,252,10000)

#up and in
for i in range(0,50):
    if(max(paths[i]) > 120):
        if(paths[i][-1] > 100):
            plt.plot(paths[i], color='green')
        else:
            plt.plot(paths[i], color='blue')
    else:
        plt.plot(paths[i],color='red')
plt.annotate('B', (1,125))
plt.axhline(y = 120, color = 'black', linestyle = '-')
plt.show()

#down and out
```

```
for i in range(0,50):
    if(min(paths[i]) < 90):
        plt.plot(paths[i],color='red')
    else:
        if(paths[i][-1] > 100):
            plt.plot(paths[i], color='green')
        else:
            plt.plot(paths[i], color='blue')

plt.annotate('B', (1,80))
plt.axhline(y = 90, color = 'black', linestyle = '-')
plt.show()

#down and in
for i in range(0,50):
    if(min(paths[i]) < 90):
        if(paths[i][-1] > 100):
            plt.plot(paths[i], color='green')
        else:
            plt.plot(paths[i], color='blue')
    else:
         plt.plot(paths[i], color='red')

plt.annotate('B', (1,80))
plt.axhline(y = 90, color = 'black', linestyle = '-')
plt.show()

#up and out
for i in range(0,50):
    if(max(paths[i]) > 120):
        plt.plot(paths[i],color='red')
    else:
        if(paths[i][-1] > 100):
            plt.plot(paths[i], color='green')
        else:
            plt.plot(paths[i], color='blue')
plt.annotate('B', (1,125))
plt.axhline(y = 120, color = 'black', linestyle = '-')
plt.show()
```

A.3 Code for Monte Carlo Pricing

```
1
2  import numpy as np
3  import pandas as pd
4  import statistics
5  import scipy.stats as sp
6  import time
7  from datetime import timedelta, date
8  import warnings
9  from scipy.stats import norm
10 warnings.filterwarnings("ignore")
11 from sklearn.linear_model import LinearRegression
12 import matplotlib.pyplot as plt
13
14
15 def sim_MC_paths(S0, T, r, sigma, n, M):
16     """
17     Parameters:
18     S0: initial price
19     T: time to maturity
20     r: risk free rate
21     sigma: volatility
22     n: number of time steps
23     M: number of paths
24     """
25     paths = []
26     dt = T / n
27
28     #simulate M paths
29     for _ in range(M):
30         s = [S0] #declare list of prices
31         #increment n timesteps
32         for _ in range(n):
33             z = np.random.normal(0,1)
34             px = s[-1] * np.exp((r - 0.5 * sigma**2) * dt + sigma * np.sqrt(dt) * z) #get
    next price based on prev. price in list
35             s.append(px)
36         paths.append(s)
37     return paths
38
39 def calc_BS_Option_Price(So, sigma, t, K, r, optType):
40     d_plus = (np.log(So/K) + (r + (sigma**2)/2)*t)/(sigma*np.sqrt(t))
41     d_minus = d_plus - sigma * np.sqrt(t)
42     if optType == "call":
43         price = So*norm.cdf(d_plus) - K*np.exp(-r*t)*norm.cdf(d_minus)
44     elif optType == "put":
45         price = K*np.exp(-r*t)*norm.cdf(-d_minus) - So*norm.cdf(-d_plus)
46     return price
47
48
49 results = pd.DataFrame(columns=['barrier_Scale','in_call','out_call','in_put','out_put','
    sim_call','sim_put', 'bs_call','bs_put'])
50
```

```
S0 = 100  # Initial stock price
K = 90  # Strike price
r = 0.05  # Risk-free interest rate
sigma = 0.25  # Volatility of the underlying asset
T = 0.5  # Time to maturity in years
n = 252
M = 100
paths = sim_MC_paths(S0,T,r,sigma,n,M)

S0Scales = [60]

for n in S0Scales:
    b =  n
    kicall = []
    kiput = []
    kocall = []
    koput = []
    vput = []
    vcall = []
    for s in paths:
        if b > S0: #up options
            if max(s) > b: #knock in
                kicall.append(max(s[-1] - K, 0))
                kocall.append(0)
                kiput.append(max(K - s[-1], 0))
                koput.append(0)
            else: #knock out
                kicall.append(0)
                kocall.append(max(s[-1] - K,0))
                kiput.append(0)
                koput.append(max(K - s[-1],0))
        else: #down options
            if min(s) < b: #knock in
                kicall.append(max(s[-1] - K, 0))
                kocall.append(0)
                kiput.append(max(K - s[-1], 0))
                koput.append(0)
            else: #knock out
                kicall.append(0)
                kocall.append(max(s[-1] - K,0))
                kiput.append(0)
                koput.append(max(K - s[-1],0))
        #vanilla options with k = b
        vcall.append(max(s[-1] - b, 0))
        vput.append(max(b - s[-1],0))

    price_kicall = np.exp(-r * T) * np.mean(kicall)
    price_kiput = np.exp(-r * T) * np.mean(kiput)
    price_kocall = np.exp(-r * T) * np.mean(kocall)
    price_koput = np.exp(-r * T) * np.mean(koput)
    price_vcall = np.exp(-r * T) * np.mean(vcall)
    price_vput = np.exp(-r * T) * np.mean(vput)
    price_bs_call = calc_BS_Option_Price(S0, sigma, T, b, r, "call")
    price_bs_put = calc_BS_Option_Price(S0, sigma, T, b, r, "put")
```

```
    results.loc[len(results)] = [n,price_kicall,price_kocall,price_kiput,price_koput,
    price_vcall,price_vput,price_bs_call,price_bs_put]


results
```

A.4 Code for Delta (Δ)

```python
# delta_analysis.py
import numpy as np
import scipy.stats as ss
import matplotlib.pyplot as plt
from scipy import sparse
from scipy.sparse.linalg import splu


# Define the BSmodel and BarrierOption classes
class BSmodel:
    def __init__(self, S0, K, T, v, r, q):
        self.S0 = S0
        self.K = K
        self.T = T
        self.v = v
        self.r = r
        self.q = q

    @property
    def d1(self):
        return (np.log(self.S0 / self.K) + (self.r - self.q + 0.5 * self.v**2) * self.T) / (
    self.v * np.sqrt(self.T))

    @property
    def d2(self):
        return self.d1 - self.v * np.sqrt(self.T)


class BarrierOption(BSmodel):
    def __init__(self, S0, K, T, v, r, q, barrier):
        super().__init__(S0, K, T, v, r, q)
        self.H = barrier

    def up_put(self, knock='out'):
        if knock not in ['in', 'out']:
            raise ValueError("knock must be 'in' or 'out'")
        if self.S0 >= self.H:
            return 0  # Knocked out
        vanilla_put = self.vanilla_european_put()
        if knock == 'out':
            return vanilla_put
        else:
            return vanilla_put  # Placeholder for up-and-in put option

    def vanilla_european_put(self):
        p = (self.K * np.exp(-self.r * self.T) * ss.norm.cdf(-self.d2) -
             self.S0 * np.exp(-self.q * self.T) * ss.norm.cdf(-self.d1))
        return p


# Calculate delta using finite difference method for barrier option
def calculate_delta_barrier(S0, K, B, T, sigma, r):
```

```
    epsilon = 1e-4  # Small change in S
    barrier_option.S0 = S0 + epsilon
    price_plus = barrier_option.up_put(knock='out')
    barrier_option.S0 = S0 - epsilon
    price_minus = barrier_option.up_put(knock='out')
    delta = (price_plus - price_minus) / (2 * epsilon)
    return delta


# Define parameters
S0 = 40.0  # spot stock price
K = 50  # strike
T = 1.0  # maturity
r = 0.1  # risk free rate
sigma = 0.2  # diffusion coefficient or volatility
B = 100  # Barrier level

# Initialize barrier option pricing class
barrier_option = BarrierOption(S0=S0, K=K, T=T, v=sigma, r=r, q=0, barrier=B)
# Calculate baseline option price at the initial spot S0
baseline_price = barrier_option.up_put(knock='out')
print(f"Baseline option price at S0={S0}: {baseline_price:.4f}")

# Calculate delta values across a range of stock prices
S_values = np.linspace(K / 3, B - 1e-8, 500)
deltas_barrier = [calculate_delta_barrier(S, K, B, T, sigma, r) for S in S_values]

# Plotting Delta
plt.figure(figsize=(10, 6))
plt.plot(S_values, deltas_barrier, label="Delta of Up-and-out Put Option")
plt.axvline(x=S0, color="g", linestyle="--", label="Spot Price (S)")
plt.axvline(x=K, color="b", linestyle="--", label="Strike Price (K)")
plt.axvline(x=B, color="r", linestyle="--", label="Barrier level (B)")
plt.xlabel("Stock Price $S$")
plt.ylabel("Delta")
plt.title("Delta of Up-and-out Put Option")
plt.legend(loc='upper right')
plt.grid(True)
plt.show()

# Calculate delta at a specific stock price
specific_stock_price = 40.0
delta_value = calculate_delta_barrier(specific_stock_price, K, B, T, sigma, r)

print(f"Delta at S={specific_stock_price}: {delta_value:.4f}")
```

A.5 Code for Gamma (Γ)

```python
# gamma_analysis.py
import numpy as np
import scipy.stats as ss
import matplotlib.pyplot as plt


# Define the BSmodel and BarrierOption classes
class BSmodel:
    def __init__(self, S0, K, T, v, r, q):
        self.S0 = S0
        self.K = K
        self.T = T
        self.v = v
        self.r = r
        self.q = q

    @property
    def d1(self):
        return (np.log(self.S0 / self.K) + (self.r - self.q + 0.5 * self.v**2) * self.T) / (
    self.v * np.sqrt(self.T))

    @property
    def d2(self):
        return self.d1 - self.v * np.sqrt(self.T)


class BarrierOption(BSmodel):
    def __init__(self, S0, K, T, v, r, q, barrier):
        super().__init__(S0, K, T, v, r, q)
        self.H = barrier

    def up_put(self, knock='out'):
        if knock not in ['in', 'out']:
            raise ValueError("knock must be 'in' or 'out'")
        if self.S0 >= self.H:
            return 0  # Knocked out
        vanilla_put = self.vanilla_european_put()
        return vanilla_put

    def vanilla_european_put(self):
        p = (self.K * np.exp(-self.r * self.T) * ss.norm.cdf(-self.d2) -
             self.S0 * np.exp(-self.q * self.T) * ss.norm.cdf(-self.d1))
        return p


# Calculate delta for a given stock price
def calculate_delta(S, K, B, T, sigma, r):
    """Calculate Delta using finite difference approximation."""
    epsilon = 1e-4  # A small stock price change
    # Change the stock price positively
    barrier_option_plus = BarrierOption(S + epsilon, K, T, sigma, r, 0, B)
    price_plus = barrier_option_plus.up_put(knock='out')
```

```
    # Change the stock price negatively
    barrier_option_minus = BarrierOption(S - epsilon, K, T, sigma, r, 0, B)
    price_minus = barrier_option_minus.up_put(knock='out')
    # Approximation of delta
    delta = (price_plus - price_minus) / (2 * epsilon)
    return delta


# Calculate gamma using finite difference
def calculate_gamma(S, K, B, T, sigma, r):
    """Calculate Gamma using finite difference."""
    epsilon = 1e-4  # Small change in stock price
    # Calculate deltas at stock price moved up and down
    delta_plus = calculate_delta(S + epsilon, K, B, T, sigma, r)
    delta_minus = calculate_delta(S - epsilon, K, B, T, sigma, r)
    # Compute gamma
    gamma = (delta_plus - delta_minus) / (2 * epsilon)
    return gamma


# Set parameters for simulation
S0 = 40.0  # Spot stock price
K = 50  # Strike price
T = 1.0  # Maturity
r = 0.1  # Risk-free rate
sigma = 0.2  # Volatility
B = 100  # Barrier level

# Range of stock prices to evaluate gamma across
S_values = np.linspace(K / 3, B - 1e-8, 500)

# Compute gamma values across stock prices
gammas = [calculate_gamma(S, K, B, T, sigma, r) for S in S_values]

# Plotting the Gamma values
plt.figure(figsize=(10, 6))
plt.plot(S_values, gammas, label="Gamma of Up-and-out Put Option")
plt.axvline(x=S0, color="g", linestyle="--", label="Spot Price (S)")
plt.axvline(x=K, color="b", linestyle="--", label="Strike Price (K)")
plt.axvline(x=B, color="r", linestyle="--", label="Barrier level (B)")
plt.xlabel("Stock Price $S$")
plt.ylabel("Gamma")
plt.ylim(-0.01, 0.05)  # Adjust y-axis to zoom in on the range of gamma
plt.title("Gamma of Up-and-out Put Option")
plt.legend(loc='upper right')
plt.grid(True)
plt.show()

specific_stock_price = 40.0
gamma_value = calculate_gamma(specific_stock_price, K, B, T, sigma, r)

print(f"Gamma at S={specific_stock_price}: {gamma_value:.4f}")
```

A.6 Code for Vega (ν)

```python
from BSM_option_class import BarrierOption
import matplotlib.pyplot as plt
import numpy as np


# Define parameters
S0 = 40.0  # spot stock price
X0 = np.log(S0)
K = 50  # strike price
T = 1.0  # maturity
r = 0.1  # risk-free rate
sigma = 0.2  # volatility
B = 100  # Barrier level


# Calculate Vega using finite difference
def calculate_vega(S0, K, B, T, sigma, r, epsilon=1e-4):
    """
    Calculate Vega for an up-and-out put option using finite difference.
    epsilon: Small change in volatility.
    """
    # Increase volatility by a small amount
    price_plus_vol = BarrierOption(S0, K, T, sigma + epsilon, r, 0, B).up_put(knock='out')

    # Decrease volatility by a small amount
    price_minus_vol = BarrierOption(S0, K, T, sigma - epsilon, r, 0, B).up_put(knock='out')

    # Compute Vega
    vega = (price_plus_vol - price_minus_vol) / (2 * epsilon)

    return vega/100


# Compute specific Vega at S0=40
specific_vega = calculate_vega(S0, K, B, T, sigma, r)
print(f"Vega at S0={S0}, T={T}, sigma={sigma}: {specific_vega:.4f}")

# Define a range of stock prices for plotting
S_values = np.linspace(K / 3, B - 1e-8, 500)  # Range of stock prices

# Compute Vega across a range of stock prices
vegas = [calculate_vega(S, K, B, T, sigma, r) for S in S_values]

# Plotting Vega over the range
plt.figure(figsize=(10, 6))
plt.plot(S_values, vegas, label="Vega of Up-and-out Put Option")
plt.axvline(x=S0, color="g", linestyle="--", label=f"Spot Price (S0={S0})")
plt.axvline(x=K, color="b", linestyle="--", label="Strike Price (K)")
plt.axvline(x=B, color="r", linestyle="--", label="Barrier level (B)")
plt.xlabel("Stock Price $S$")
plt.ylabel("Vega")
plt.title("Vega of Up-and-out Put Option as a Function of Stock Price")
```

```
plt.legend(loc="upper right")
plt.grid(False)
plt.show()
```

A.7 Code for Theta (Θ)

```
from BSM_option_class import BarrierOption
import matplotlib.pyplot as plt
import numpy as np


# Define parameters
S0 = 40  # spot stock price
X0 = np.log(S0)
K = 50  # strike price
T = 1.0  # maturity
r = 0.1  # risk-free rate
sigma = 0.2  # volatility
B = 100  # Barrier level


# Calculate Theta using finite difference
def calculate_theta(S0, K, B, T, sigma, r, epsilon=1/252):
    """
    Calculate Theta using finite difference (change in time).
    epsilon: Small change in time (1 trading day expressed in years).
    """
    # Current time value
    barrier_option_now = BarrierOption(S0, K, T, sigma, r, 0, B)
    price_now = barrier_option_now.up_put(knock='out')

    # Slightly earlier time value
    barrier_option_earlier = BarrierOption(S0, K, T - epsilon, sigma, r, 0, B)
    price_earlier = barrier_option_earlier.up_put(knock='out')

    # Theta calculation
    theta = (price_earlier - price_now)
    return theta


# Compute specific Theta at S0=40
specific_theta = calculate_theta(S0, K, B, T, sigma, r)
print(f"Theta at S0={S0}, T={T}, sigma={sigma}: {specific_theta:.4f}")

# Define a range of stock prices for plotting
S_values = np.linspace(K / 3, B - 1e-8, 500)  # Range of stock prices

# Compute Theta across a range of stock prices
thetas = [calculate_theta(S, K, B, T, sigma, r) for S in S_values]

# Plotting Theta over the range
plt.figure(figsize=(10, 6))
plt.plot(S_values, thetas, label="Theta of Up-and-out Put Option")
plt.axvline(x=S0, color="g", linestyle="--", label=f"Spot Price (S0={S0})")
plt.axvline(x=K, color="b", linestyle="--", label="Strike Price (K)")
plt.axvline(x=B, color="r", linestyle="--", label="Barrier level (B)")
plt.xlabel("Stock Price $S$")
plt.ylabel("Theta")
```

```
plt.title("Theta of Up-and-out Put Option as a Function of Stock Price")
plt.legend(loc="upper right")
plt.grid(True)
plt.show()
```

A.8 Code for Rho (ρ)

```python
# rho_analysis.py
import numpy as np
import scipy.stats as ss
import matplotlib.pyplot as plt


# Define the BSmodel and BarrierOption classes
class BSmodel:
    def __init__(self, S0, K, T, v, r, q):
        self.S0 = S0
        self.K = K
        self.T = T
        self.v = v
        self.r = r
        self.q = q

    @property
    def d1(self):
        return (np.log(self.S0 / self.K) + (self.r - self.q + 0.5 * self.v**2) * self.T) / (
    self.v * np.sqrt(self.T))

    @property
    def d2(self):
        return self.d1 - self.v * np.sqrt(self.T)


class BarrierOption(BSmodel):
    def __init__(self, S0, K, T, v, r, q, barrier):
        super().__init__(S0, K, T, v, r, q)
        self.H = barrier

    def up_put(self, knock='out'):
        if knock not in ['in', 'out']:
            raise ValueError("knock must be 'in' or 'out'")
        if self.S0 >= self.H:
            return 0  # Knocked out
        vanilla_put = self.vanilla_european_put()
        return vanilla_put

    def vanilla_european_put(self):
        p = (self.K * np.exp(-self.r * self.T) * ss.norm.cdf(-self.d2) -
             self.S0 * np.exp(-self.q * self.T) * ss.norm.cdf(-self.d1))
        return p


# Calculate option price at a given stock price and risk-free rate
def calculate_option_price(S, K, T, sigma, r, B):
    """Compute the price at a given stock price S with risk-free rate r."""
    barrier_option = BarrierOption(S, K, T, sigma, r, 0, B)
    return barrier_option.up_put(knock='out')


```

```python
# Calculate rho using finite difference
def calculate_rho(S, K, B, T, sigma, r):
    """Calculate Rho by finite difference."""
    dr = 0.0001  # Small change in risk-free rate

    # Compute the option price at risk-free rate r + dr
    V_plus = calculate_option_price(S, K, T, sigma, r + dr, B)

    # Compute the option price at risk-free rate r - dr
    V_minus = calculate_option_price(S, K, T, sigma, r - dr, B)

    # Rho is the rate of change with respect to risk-free rate
    rho = (V_plus - V_minus) / (2 * dr)
    return rho/100


# Set parameters for simulation
S0 = 50  # Spot stock price
K = 50  # Strike price
T = 1.0  # Maturity
r = 0.1  # Risk-free rate
sigma = 0.2  # Volatility
B = 100  # Barrier level

# Range of stock prices to evaluate rho across
S_values = np.linspace(K / 3, B - 1e-8, 500)

# Compute rho values across stock prices
rhos = [calculate_rho(S, K, B, T, sigma, r) for S in S_values]

# Plotting the Rho values
plt.figure(figsize=(10, 6))
plt.plot(S_values, rhos, label="Rho of Up-and-out Put Option")
plt.axvline(x=S0, color="g", linestyle="--", label="Spot Price (S)")
plt.axvline(x=K, color="b", linestyle="--", label="Strike Price (K)")
plt.axvline(x=B, color="r", linestyle="--", label="Barrier level (B)")
plt.xlabel("Stock Price $S$")
plt.ylabel("Rho")
plt.title("Rho of Up-and-out Put Option")
plt.legend(loc='upper right')
plt.grid(True)
plt.show()


specific_stock_price = 50
rho_value = calculate_rho(specific_stock_price, K, B, T, sigma, r)

print(f"Rho at S={specific_stock_price}: {rho_value:.4f}")
```