Leeds University Business School

UNIVERSITY OF LEEDS

Dissertation/Project Coversheet

Student ID Number:	[redacted]
Student Name	Louis Cooke
Module Code:	MATH5312M
Programme of Study:	Financial Mathematics
Supervisor:	[redacted]
Title:	The Inverse Problem of Option Pricing
Declared Word Count:	9981

Please Note:

Your declared word count must be accurate, and should not mislead. Making a fraudulent statement concerning the work submitted for assessment could be considered academic malpractice and investigated as such. If the amount of work submitted is higher than that specified by the word limit or that declared on your word count, this may be reflected in the mark awarded and noted through individual feedback given to you.

It is not acceptable to present matters of substance, which should be included in the main body of the text, in the appendices ("appendix abuse"). It is not acceptable to attempt to hide words in graphs and diagrams; only text which is strictly necessary should be included in graphs and diagrams.

By submitting an assignment you confirm you have read and understood the University of Leeds **Declaration of Academic Integrity** (http://www.leeds.ac.uk/secretariat/documents/academic_integrity.pdf).

Abstract

Financial institutions rely on quantitative models to price options in the market, while the introduction of the Black-Scholes model was revolutionary for quantitative finance, it lacked critical detail of particular market dynamics. In the direct approach to option pricing, using independently sourced data for the volatility (σ), the interest rate (r) and the dividend yield (D) as parameters in the Black-Scholes solution often misaligns market prices with the Black-Scholes price. Furthermore, we crudely assume that σ, r and D are known constants with this approach. We propose an inverse method to invert and solve the Black-Scholes solution for three unknowns (σ, r and D) using three market option premiums. This approach doesn't require independent sourcing of data and recovers accurate values for σ, r and D. However, introducing larger noise into this non-linear system generates instabilities, which require supplementary numerical methods.

Keywords: *Black-Scholes, direct approach, volatility, interest rate, dividend yield, inverse method, option premiums, noise, instabilities, numerical methods.*

Contents

Chapter 1

Introduction

1.1 Literature review

The quantitative approach to option pricing was accelerated by the introduction of the Black-Scholes model in 1973. The Black-Scholes model produced a framework to price derivatives under the assumption that the returns from the underlying asset are distributed log-normally [1]. The Black-Scholes partial differential equation (PDE) allows us to price derivates in quantitative finance based on the stochastic process of the underlying asset. For many options there is an explicit analytical solution to this PDE which provides a means of pricing the option based on particular market variables.

The Black-Scholes formula calculates the option premium, C as a function of 6 dependent variables. In the direct approach to we find the option price at a given stock, strike price and time to maturity, while the volatility, interest rate, and dividend yield are specified before calculating C. This problematically assumes that the volatility, dividend yield and the risk-free interest rate are known constants. This is an inaccurate assumption which ignores key market trends in volatility, generating a misalignment between option premiums on the market with the Black-Scholes price.

The Volatility of an option is not directly observable, it is implicit through the behaviour of the underlying asset. Volatility is an important quantity used by investors to evaluate their risk. The Black-Scholes model is formulated under the assumption that investors are 'risk-neutral' meaning that in the direct approach, we ignore any dependencies between the option's volatility with the strike, and stock price [2]. Derman and Kani show that options on the S&P500 index with expiry of 44 days exhibit an implied volatility which decreases with strike price, and increases with time to expiry [2, p 278], this is called the volatility skew. Although Derman and Kani exclusively focus on a market that exhibits skews in their 1994 paper, other markets trade derivatives that induce different volatility behaviour like smiles and flat smiles.

Derman and Kani attempt to deduce $\sigma(S,t)$ numerically, from observed option prices that exhibit this skew. Their approach uses the discrete time-spaced, Cox-Ross-Rubinstein binomial tree, calibrated to fit market data. Transition probabilities and node spacings for u and d are adjusted to match all European option prices while still enforcing no-arbitrage conditions. By deriving local volatility from the tree, they show how to deduce volatility dependent on time and stock price [2]. Their approach is effective for pricing options that have properties which are sensitive to the shape of the skew, like barrier options where the probability of striking the barrier causes the skew to rapidly change shape [2]. However, some markets trade options and derivatives with relatively consistent implied volatility making calibration from market data difficult. Bouchouev and Isakov also suggest that there is not enough market data in the time direction to reconstruct accurate local time-dependent volatility functions [3].

Because of this issue, Bouchouev and Isakov assume that volatility is time-independent [3]. They extend the work of Derman and Kani by presenting this inverse problem in the continuous time framework. They derived an iterative method that inputs the option price and the payoff to find the local volatility function that accurately replicates the implied volatility over a short number of iterations [3, p L16]. Local volatility models are deterministic models that describe the volatility across all strikes and maturities. Their method is an inverse problem where the input (option prices) and output (local volatility) satisfies uniqueness and stability, defined by "Theorem 1" in [3, p L13]. Uniqueness is guaranteed if the following pair of functions: the option price and the volatility, satisfy Dupire's equation [3, p L13]. Dupire's equation links the local volatility, σ with the partial derivatives of the

call option premium [2]. The equation has different forms based on the choice of variables (spot vs. strike) and the inclusion of drift terms (dividends and interest rates). All forms are mathematically equivalent under appropriate transformations. In the later revised approach by Bouchouev and Isakov, the underlying stock diffusion is used to find Dupire's equation in the form containing the drift terms [3] [4].

Dupire's equation provides a framework for modelling local volatility, where the option price dynamics evolve smoothly over time. Conversely the Black-Scholes PDE is a backwards equation [5, p 97] which provides a single option price from the diffusion. This is one of the issues with the Black-Scholes equation: Wilmott et al. describe how forward diffusion smooths out the initial jagged data, while backwards diffusion does the opposite [5, p 86]. Inverting the Black-Scholes to recover the implied volatility is therefore inherently unstable: small changes to the call option price tend to cause large changes in the value of σ [5, p 76].

Dupire established the foundation to local volatility modelling in his 1993 work [6]. His research proposed working backwards, using the call option premium to determine the unobservable diffusion equation for the spot price of the underlying asset. That way the local volatility, $\sigma(S,t)$ could be found [6]. He points towards the potential applications of his method in pricing path dependent options and proposes the benefits of his method in hedging [6]. This work established local volatility modelling as a powerful alternative to approaches that sacrifice completeness, such as stochastic volatility or jump-diffusion models [6].

Gatheral reviews Dupire's equation, and shows how its inversion gives the equation for the generalised local volatility as a function of maturity time, T, strike price, K and the initial stock price S [7, p 11]. Gatheral shows how the local volatility function may fit the implied volatility when the local variance fits to no volatility skews, and when the local variance does fit to volatility skews [7]. We find that the local variance is independent of stock price and strike price in the case where we do not fit to the skews, and then the local variance reduces to the Black-Scholes implied variance[4, p 12]. In either case we have to calculate the partial derivatives of the call option premium with respect to time and strike price. However in financial markets we do not have a continuous range of data for strikes and maturities, there are usually only a small, discrete set of these variables [8, p R98].

To recover the volatility surface using Dupire's equation we need to somehow compute the partial derivatives of the call option premium with respect to the strike and maturity. However a lack of data means that we cannot accurately compute small changes in the option price with respect to these variables. Haugh states that that noisy market data must be 'smoothed' to ensure stability when calculating the first and second derivatives [9]. While he suggests that the local volatility model is good at pricing barrier options (described by Derman and Kani [2]), he describes certain pitfalls concerning: unreasonable skew dynamics, underestimates the 'volatility-of-volatility' [9, p 3] and can 'often significantly underprice structured products [9, p 4].

Fengler proposed a method to interpolate missing option price data in the market by fitting cubic splines between data [10]. Using a finite differencing scheme to calculate partial derivatives, Fengler ensured correct calibration to the market data by enforcing no arbitrage conditions [10]. The spline smoothing task was posed as a quadratic optimisation problem, where the constraints were the arbitrage conditions. Using this method the implied volatility surface was recovered [10]. A typical approach to employ such a finite differencing method is through central differencing [11]. It offers a simple way to calculate the derivatives in the Dupire equation for the local volatility. However, exclusively using finite differencing methods makes the recovery of the volatility surface very inaccurate and difficult as the approximated derivatives may introduce instability and arbitrage.

Bouchouev and Isakov survey some of the advantages and disadvantages of the typical numerical methods used for reconstructing local volatility surfaces. They critique parametric approaches that use a least squares method to minimize the square deviation between theoretical and market prices, suggesting that parameter non-uniqueness can cause large numerical instabilities [8, p R106]. While a similar approach is used in fitting the cubic spline in Fengler's method, Fengler ensures uniqueness

in the minimizer [10, p 6]. Direct interpolation is also critiqued for producing non-unique solutions and amplifying numerical errors [8, p R106], such issues are also addressed by Fengler. To improve stability, Bouchouev and Isakov propose two iterative algorithms. The first algorithm derives an integral equation for the volatility function [8, p R106]. This equation is discretized at observed strike/maturity points and solved iteratively [8, p R106]. The method works best for short maturities [8, p R106]. The second algorithm iteratively solves the partial differential equation for the fundamental solution directly [8, p R106]. Both methods avoid interpolation, instead they aim to reconstruct volatility between successive maturities [8, p R106].

While Dupire's local volatility model provides a deterministic framework to calibrate the volatility surface from market prices, its assumption of perfect predictability ignores the random nature of volatility observed in markets. The Heston model introduces stochastic volatility to option pricing by changing the underlying spot price process. Heston modelled the spot price assuming that the volatility follows the square root process, which includes a mean-variance term [12]. The variance follows a stochastic process and includes a term which controls the speed in which variance reverts to its mean [12]. Empirically his model reflects real-market dynamics: when stock prices fall, volatility usually increases [12]. His model is particularly successful in describing the non-Gaussian nature of stock returns which is assumed by the Black-Scholes model, but known to be misleading [12]. However, when comparing the volatility surface generated by the Heston model with the empirical implied volatility surface, Gatheral found that the Heston model struggled to fit to the empirical surface for short term options [7, p 39].

For this reason practitioners tend to favour the local volatility model suggested by Dupire, however improvements can be made to the model by allowing jumps in stock price and volatility [7, p 39]. Merton first presented Jump-diffusion models to challenge the proposition that the stock price can only change by a small amount in a short time interval [13]. This was motivated by the idea that certain market dynamics can cause a larger than marginal change in the stock price, like the arrival of new information about the stock [13, p 127]. The arrival of information in the market is modelled as a Poisson-driven process, where events are independent and identically distributed [13, p 127]. Probabilities are defined for the cases where the event: does not occur, occurs once and occurs more than once in a time interval [13, p 127]. These are then encoded into the stock price process [13, p 130]. Gatheral shows how incorporating jump diffusions into the stochastic volatility fixes the volatility surface generated for short term options. However, the assumption that we know the size of a particular jump in the market sacrifices completeness (if the jump were a jump the size of any real number) [7, p 55].

1.2 Motivation and plan of the Thesis

The inverse method we propose in this thesis uses the Black-Scholes analytical solution for European call options. Under the Black-Scholes framework the dependent variables in this model are:

Deterministic variables:

- Strike price, K.
- Time to maturity, $T-t$.

Random variables:

- Interest rate, r.
- Volatility, σ.
- Continuous dividend yield, D.
- Stock price, S.

The option premium is calculated as a function of these variables. Under the Black-Scholes model we assume that the interest rate r, the volatility, σ and the dividend yield, D are constants. In reality these values are subject to change over the option's life. In this thesis we propose a method that inverts the option premiums to derive a constant set of values for σ, r and D. For many options (especially

short-term options) the random variables r and D hold relatively constant over the option's life, we compare our values for r and D against projected values. The volatility of an option is not directly observed in an option, and thus is usually inferred from the underlying asset. We will use the inverse method to find volatility, and also discuss some of the widely researched methods used to recover volatility trends in markets.

We firstly introduce the direct problem and the Black-Scholes analytical solution used to directly solve the price of an option. We then show the motivation for the inverse method, solving for constant values of: σ, r and D, by inverting the analytical solution. This can be applied for one, two and three unknowns. We finally revise some of the non-constant models for the volatility, σ and finally apply the inverse method to real-world data.

Chapter 2

The Direct Problem

2.1 Option Dynamics

Options are categorized as either path-dependent or path-independent. Path-independent options, like European options, have a payoff determined solely by the underlying asset's price at maturity. This makes them easier to price under the Black-Scholes framework, as they often have a closed-form analytical solution.

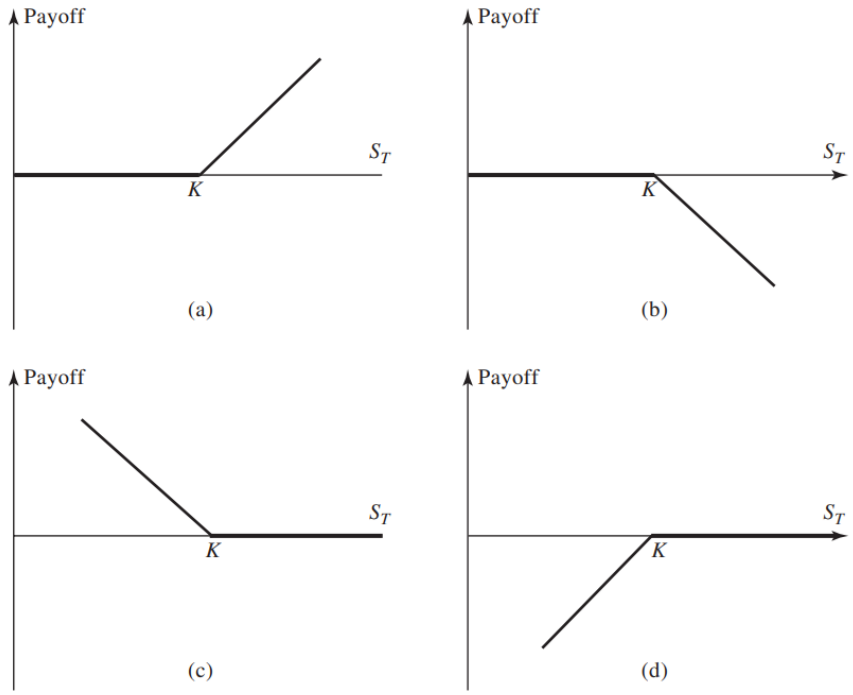

Figure 2.1: Payoff for the following a). Long call b). Short call c). Long put d). Short put, for the strike price K [14, p. 231].

Figure 2.1 shows the payoff for long and short call and put options. The long call is a contract allowing the holder to buy the underlying at K. The payoff is zero when $S \leq K$ for a long call because the holder will not exercise their right to buy at K. When $S \geq K$, the holder can buy the underlying at K and sell it at S to realize a profit of $S-K$. Conversely, the short call involves selling the contract that allows the investor buy the underlying at the price K. Its payoff is zero when $S \leq K$ and $-(S-K)$ when $S \geq K$, as the seller must deliver the asset at the lower price K if the option is exercised.

The long put gives the holder the right to sell the underlying at a fixed price K. The holder gains a positive payoff when $S \leq K$ by buying the asset at S and selling it at K to realize a profit of $K-S$. Conversely, the short put has a negative payoff when $S \leq K$ and zero payoff when $S \geq K$. This is because the seller must buy the stock at the premium price K when the market price S is lower, resulting in a loss of $-(K-S)$ if the option is exercised.

Taking long/short positions with these options requires either paying/issuing an option premium to the seller/buyer of the option to reflect the level of risk in that position. The payoff for European call and put options are given by C and P respectively:

- $C = \max\{0, S-K\}$ or $(S-K)^+$,
- $P = \max\{0, K-S\}$ or $(K-S)^+$.

The quantity S is the stock price at maturity which dictates the payoff of the option at maturity. Hull [14, p. 339] states 'the percentage changes of stock price in a very short period of time are normally distributed':

$$\frac{\Delta S}{S} \sim \phi(\mu\Delta t, \sigma^2\Delta t) \tag{2.1}$$

- The symbol $\sim$ describes how the random variable $\frac{\Delta S}{S}$ is distributed.
- The symbol ϕ is the standard normal probability density function (PDF).

A contingent claim replicates the payoff of an option such as a European call option with the payoff $(S_T - K)^+$. The price at time zero of a contingent claim X in an arbitrage free market where $\mathbb{E}_\mathbb{Q}$ is the expectation under the risk neutral measure $\mathbb{Q}$ is [15, p. 31]:

$$\text{price}_0(X) = \mathbb{E}_\mathbb{Q}\left(\frac{X}{B_T}\right) = \mathbb{E}_\mathbb{Q}\left(e^{-rT}X\right). \tag{2.2}$$

Using the stock price dynamics shown in equation (2.1) and the price of the contingent claim in equation (2.2), we can derive an analytical solution for an option with regular dividend payments.

2.2 The Black Scholes Model

2.2.1 Derivation of the Analytical Solution from the SDE

The Black-Scholes stochastic differential equation (SDE), with a cash dividends payout included is [16]:

$$dS_t = (r-D)S_t\,dt + \sigma S_t\,dW_t^Q, \tag{2.3}$$

where:

- S_t is the stock price at time, t,
- r is the risk-free rate,
- D is the continuous dividend yield,
- W_t^Q is a Brownian motion under the risk-neutral measure Q.

The solution to (2.3) is given by [16]:

$$S_T = S_t e^{(r-D-\sigma^2/2)T+\sigma W_T^Q}, \tag{2.4}$$

Using (2.2) we find that the price at time 0 for a European call option with continuous dividend yield is given by:

$$\text{price}_0(X) = e^{-rT}\mathbb{E}_Q\left[\left(S_0 e^{(r-D-\sigma^2/2)T+\sigma W_T^Q} - K\right)^+\right].$$

W_t is distributed as: $W_t \sim N(0,t)$, so we transform W_t to a standard normal variate, y by multiplying by the standard deviation $\sqrt{t}$ and adding the mean, 0. The payoff is then generalised to the following two cases:

$$(S_T - K)^+ = \begin{cases} S_0 e^{\sigma\sqrt{T}y+(r-D-\sigma^2/2)T} - K & \text{if } y \geq -d_2, \\ 0 & \text{if } y < -d_2, \end{cases}$$

where:

$$d_1 = \frac{\log\left(\frac{S_0}{K}\right) + \left(r - D + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}.$$

We want to find the price given by equation (2.2), the expectation of a random variable is given by:

$$E(X) = \int_{-\infty}^{\infty} x f(x)\, dx,$$

where $f(x)$ is the standard normal PDF given by:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right). \tag{2.5}$$

Equation (2.5) has a known cumulative distribution function (CDF) found by integrating the function from $-\infty$ to z. This is used to isolate our exponent to be a function of y^2 when finding the solution. We use the payoff function to now define the limits of the integral between $-d_2$ and ∞ (because payoff is zero when y is below $-d_2$):

$$\text{price}_0(X) = e^{-rT} \int_{-d_2}^{\infty} \left(S_0 e^{\sigma\sqrt{T}y + \left(r - D - \frac{\sigma^2}{2}\right)T} - K\right) \frac{e^{-\frac{y^2}{2}}}{\sqrt{2\pi}}\, dy.$$

Now by splitting the integral into two parts to simplify the problem we have:

$$\text{price}_0(X) = e^{-rT} \left(\frac{S_0}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{\sigma\sqrt{T}y + \left(r - D - \frac{\sigma^2}{2}\right)T - \frac{y^2}{2}}\, dy - \frac{K}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{-\frac{y^2}{2}}\, dy\right).$$

The second integral is the standard normal CDF evaluated at d_2 as shown by equation (2.5). The first integral is simplified by expanding the bracket, then by completing the square to obtain $-\frac{1}{2}(y - \sigma\sqrt{T})^2 + (r - D)T$ as the exponent:

$$\text{price}_0(X) = \frac{S_0}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{-\frac{1}{2}(y - \sigma\sqrt{T})^2 + (r - D)T - rT}\, dy - K e^{-rT} \Phi(d_2),$$

cancelling the rT terms we have:

$$\text{price}_0(X) = \frac{S_0}{\sqrt{2\pi}} \int_{-d_2}^{\infty} e^{-\frac{1}{2}(y - \sigma\sqrt{T})^2 - DT}\, dy - K e^{-rT} \Phi(d_2).$$

This is an integrable function of the form e^{z^2} where $z = y - \sigma\sqrt{T}$. We integrate with respect to z, replacing the lower limit with $-d_2 - \sigma\sqrt{T} = -d_1$:

$$\text{price}_0(X) = e^{-DT} S_0 \int_{-d_1}^{\infty} e^{-\frac{z^2}{2}} - K e^{-rT} \Phi(d_2),$$

the change of variable gives the standard CDF evaluated at d_1:

$$\text{price}_0(X) = e^{-DT} S_0 \Phi(d_1) - K e^{-rT} \Phi(d_2). \tag{2.6}$$

Equation (2.6) is the analytical solution for a European call option that has a continuous dividend yield, D.

2.2.2 Derivation of the Analytical Solution from the PDE

The Black-Scholes partial differential equation without a dividend yield can be written as [5]:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + rS\frac{\partial C}{\partial S} - rC = 0, \quad (S,t) \in (0,\infty) \times (0,T). \tag{2.7}$$

Where C is the value of the option (the option premium). The partial differential equation has no unique solution and to solve it to find the option's value we must first establish the boundary conditions to define the region of points where the solution lies. We firstly define the payoff for a call option:

$$C(S,T) = \max\{S-K,0\}, \tag{2.8}$$

with boundary conditions:

$$C(0,t) = 0, \quad t \in (0,T), \tag{2.9}$$

$$C(S,t) \sim Se^{-D(T-t)} \quad \text{as} \quad S \to \infty, \quad t \in (0,\infty) \tag{2.10}$$

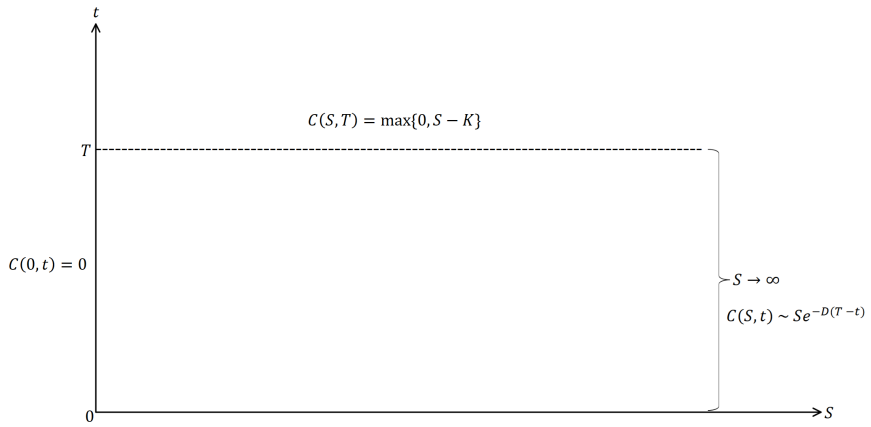

Figure 2.2: The region where the solution lies limited to the boundaries where the stock price approaches infinity and the option reaches maturity

Figure 2.2 shows the region to which the solution of the Black-Scholes PDE lies. The upper boundary represents the option payoff at maturity. The left boundary is limited to zero for European call options. We define the right boundary by setting the x value as some arbitrarily large number for the call option such that the stock price is approximately equal to the payoff from the option.

The Black Scholes PDE can be used to find the price of such an option when re-written to include a continuous dividend yield, D:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} + (r-D)S\frac{\partial C}{\partial S} - rC = 0, \quad (S,t) \in (0,\infty) \times (0,T). \tag{2.11}$$

However the Black Scholes PDE is not typically used to find analytical solutions, we tend to use the heat equation:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}. \tag{2.12}$$

The heat equation is an equation that describes the diffusion of heat through a continuous medium, it does however have useful applications in financial modelling. In particular we will exploit some of its useful properties to find a similarity solution to equation (2.11) [5, p. 80]:

- *Second order* - Highest derivative term is $\frac{\partial^2 u}{\partial x^2}$.
- *Parabolic* - Has two equal roots.
- *Linear* - If u_1 and u_2 are solutions then for constants c_1, c_2 the linear combination $c_1u_1 + c_2u_2$ is also a solution.

We will find the analytical solution from the PDE using a similarity solution [5, p. 98]. The PDE is altered under the following change of variables:

$$S = Ke^x, \quad t = T - \frac{\tau}{\frac{1}{2}\sigma^2}, \quad C(x,t) = Kv(x,\tau). \tag{2.13}$$

Substituting the changed variables into the Black Scholes PDE and simplifying we have:

$$\frac{\partial v}{\partial \tau} = \frac{\partial^2 v}{\partial x^2} + (k-1)\frac{\partial v}{\partial x} - k_1 v, \quad (x,\tau) \in (-\infty,\infty) \times \left(0, \frac{\sigma^2 T}{2}\right), \tag{2.14}$$

where:

$$k = \frac{r-D}{\frac{1}{2}\sigma^2}, \qquad k_1 = \frac{r}{\frac{1}{2}\sigma^2}.$$

For the initial condition, the payoff for a call option [5, p. 98] transforms to:

$$v(x,0) = \max\{e^x - 1, 0\} = \begin{cases} e^x - 1 & \text{if } x \geq 0, \\ 0 & \text{if } x < 0, \end{cases} \quad x \in (-\infty,\infty), \tag{2.15}$$

equation (2.14) is now more similar to the heat equation, we can get closer by using:

$$v(x,\tau) = e^{\alpha x + \beta \tau} u(x,\tau). \tag{2.16}$$

Substituting $v(x,\tau)$ into the PDE and using the product rule by taking partial derivatives with respect to time and space we have:

$$\beta u + \frac{\partial u}{\partial \tau} = \alpha^2 u + 2\alpha \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + (k-1)\left(\alpha u + \frac{\partial u}{\partial x}\right) - k_1 u. \tag{2.17}$$

The exponential terms cancel on both sides leaving an equation that can be simplified to the heat equation. α and β are constants, choosing $\alpha = -\frac{1}{2}(k-1)$ and $\beta = -\frac{1}{4}(k-1)^2 - k_1$ such that the u and $\frac{\partial u}{\partial x}$ terms cancel out leaves the standard heat equation:

$$\frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}. \tag{2.18}$$

We then have:

$$v(x,\tau) = e^{-\frac{1}{2}(k-1)x - \frac{1}{4}(k-1)^2\tau - k_1\tau} u(x,\tau), \tag{2.19}$$

the transformed initial condition is therefore v at $\tau = 0 : (e^x - 1)^+$ multiplied by $e^{-\alpha x - \beta\tau}$ at $\tau = 0$, so $e^{-\alpha x}(e^x - 1)^+$:

$$u(x,0) = \max\left\{e^{\frac{1}{2}(k+1)x} - e^{\frac{1}{2}(k-1)x}, 0\right\}, \quad x \in (-\infty,\infty). \tag{2.20}$$

The standard solution to the equation (2.12) subject to the initial condition from equation (2.20) is:

$$u(x,\tau) = \frac{1}{2\sqrt{\pi\tau}} \int_{-\infty}^{\infty} u(s,0) e^{-\frac{(x-s)^2}{4\tau}}\, ds. \tag{2.21}$$

Using a change of variable to simplify the solution, $x' = -\frac{(x-s)}{\sqrt{2\tau}} \Rightarrow s = x'\sqrt{2\tau} + x$, alongside $\frac{dx'}{ds} = \frac{1}{\sqrt{2\tau}}$, the solution to the transformed PDE is given by:

$$u(x,\tau) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} u(x'\sqrt{2\tau} + x, 0) e^{-\frac{1}{2}x'^2}\, dx'. \tag{2.22}$$

We need to compute the integral in the region where the payoff is non-zero. Equation (2.21) has a non-zero payoff when $u(s,0) > 0$ which occurs when $s > 0$. Our lower limit for x' is therefore set when $s = 0$ leading to $-\frac{x}{\sqrt{2\tau}}$ (from the variable change in equation (2.22)). Then by substituting in the transformed variables and splitting it into two separate integrals, we find that:

$$u(x,\tau) = \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sqrt{2\tau}}}^{\infty} e^{\frac{1}{2}(k+1)(x'\sqrt{2\tau}+x)} e^{-\frac{1}{2}x'^2}\, dx' - \frac{1}{\sqrt{2\pi}} \int_{-\frac{x}{\sqrt{2\tau}}}^{\infty} e^{\frac{1}{2}(k-1)(x'\sqrt{2\tau}+x)} e^{-\frac{1}{2}x'^2}\, dx'. \tag{2.23}$$

Through completion of the square to isolate x' we have $-\frac{1}{2}\left(x' - \frac{(k+1)\sqrt{2\tau}}{2}\right)^2$ and $-\frac{1}{2}\left(x' - \frac{(k-1)\sqrt{2\tau}}{2}\right)^2$ as the exponents of the integral. Then by using the change of variables: $u = x' - \frac{(k+1)\sqrt{2\tau}}{2}$ and $u = x' - \frac{(k-1)\sqrt{2\tau}}{2}$, the integral eventually simplifies to:

$$u(x,\tau) = e^{\frac{1}{2}(k+1)x + \frac{1}{4}(k+1)^2\tau}\Phi(d_1) - e^{\frac{1}{2}(k-1)x + \frac{1}{4}(k-1)^2\tau}\Phi(d_2), \tag{2.24}$$

where Φ is the standard normal CDF. Reverting all the above transformations we get the explicit solution as:

$$C(S,t) = \text{price}_t(X) = e^{-D(T-t)}S\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2), \tag{2.25}$$

for the call option with the payoff $X = (S_t - K)^+$, where:

$$d_1 = \frac{\log(S/K) + \left(r - D + \frac{1}{2}\sigma^2\right)(T-t)}{\sigma\sqrt{T-t}}, \quad d_2 = d_1 - \sigma\sqrt{T-t}. \tag{2.26}$$

This is the same analytical solution obtained in equation (2.7) for the solution at time zero:

$$\text{price}_0(X) = e^{-DT}S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2),$$

where:

$$d_1 = \frac{\log\left(\frac{S_0}{K}\right) + \left(r - D + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}.$$

Hence we have shown how to prove the analytical solution using a similarity solution for the price of an option that includes a continuous dividend payment. Using Python we can plot the call option premium as a function of stock price and time using this solution:

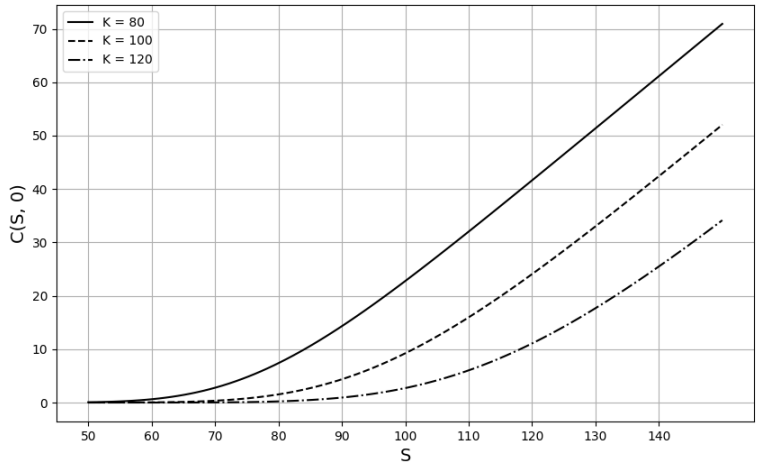

Figure 2.3: The call option premium $C(S)$ shown by equation (2.25) is plotted at $t = 0$ as a function of stock price, S for the strike prices: $K \in \{80, 100, 120\}$, $r = 0.05$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 2.3 shows the monotonically increasing relationship between the option premium and stock price, S which is attributable to the higher expected payoff in European call options at higher stock prices. We also find that the higher strike price options cost a smaller premium to the buyer because the option will have a higher probability of paying nothing when the strike price is higher.

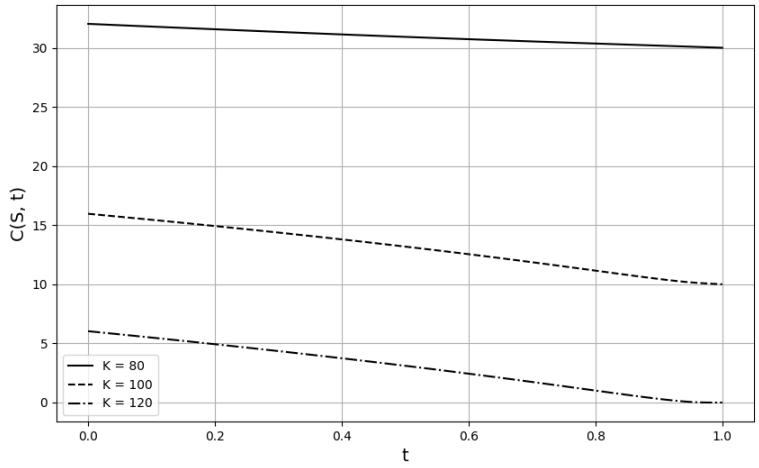

Figure 2.4: The call option premium $C(t)$ shown by equation (2.25) is plotted as a function of time, t by fixing the stock price at, $S = 110$ for the following strike prices $K \in \{80, 100, 120\}$, $r = 0.05$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 2.4 shows the monotonically decreasing relationship for the call option premium as a function of time, t. We notice that alike Figure 2.3 the higher strike price suggests a lower call option premium because the option would be less likely to have a payout above zero. The option premiums at maturity, $t = 1$ reflect the time boundary condition from figure 2.2, where at maturity the option premium equals the payoff.

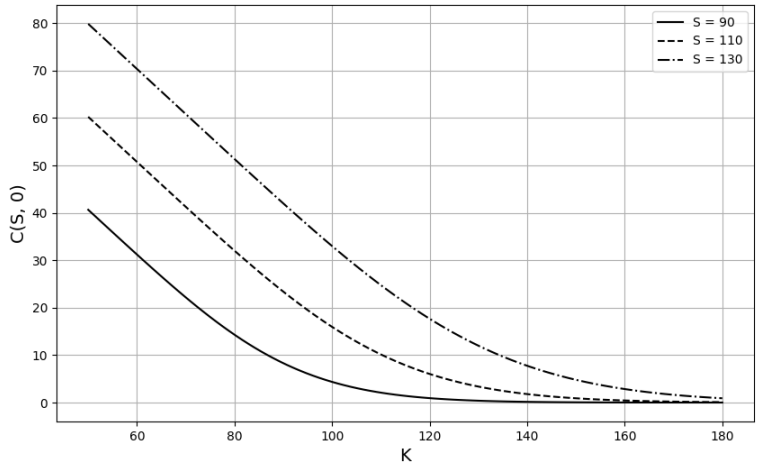

Figure 2.5: The call option premium $C(K)$ shown by equation (2.25) is plotted at $t = 0$ as a function of strike price, K for the following stock prices, $S \in \{90, 110, 130\}$, $r = 0.05$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 2.5 illustrates the monotonically decreasing relationship of the call option premium as a function of the strike price, K. The three different option premiums all appear to approach zero at high strike prices, however they all still have a positive unique price even when they are far out of the money. This is because the options in this plot still have a time to maturity of 1 year, hence they still have the potential to have a non-zero payoff.

In the direct approach to pricing options, we assume that K, σ, r, D and T are all constant, the only variables that change are time to maturity, $T - t$ and stock price, S. We can produce a surface plot in python to investigate the relationship between the call option premium, $C(S,t)$, stock price, S and time t:

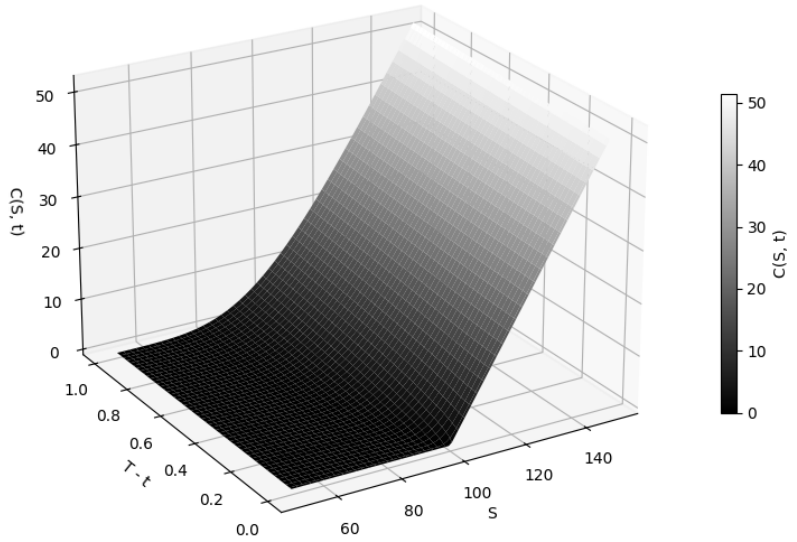

Figure 2.6: The call option premium, $C(S,t)$ shown by equation (2.25) is plotted in a surface plot as a function of stock price, S_t and time, t for the following, $K = 100$, $r = 0.05$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 2.6 shows the call option premium $C(S,t)$ as a function of the time to maturity, $T-t$ and stock price, S. The plot captures the piecewise linear payoff of the European call option at the time to maturity of 0 years shown in equation (2.8) alongside the smooth payoff shown by figure 2.3 at the time to maturity of 1 year.

2.3 Extensions of the Black-Scholes Model

2.3.1 Reduction to the Heat Equation

We have derived the analytical solution for the price of a European call option with a continuous dividend yield, D from the Black-Scholes PDE. We now present extensions of the Black-Scholes PDE to any constant a that multiplies the first derivative, for a general parabolic equation we have [5, Exercise 4, p. 103]:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + a\frac{\partial u}{\partial x} + bu.$$

This can be reduced to the heat equation by using a similar transformation shown previously for the derivation of the analytical solution using the PDE:

$$u(x,t) = e^{\alpha x+\beta t}v(x,t),$$

Substituting $u(x,t)$ into this PDE and taking the partial derivatives we find:

$$\beta v + \frac{\partial v}{\partial t} = \alpha^2 v + 2\alpha\frac{\partial v}{\partial x} + \frac{\partial^2 v}{\partial x^2} + a\left(\alpha v + \frac{\partial v}{\partial x}\right) + bv.$$

Eliminating the $\frac{\partial v}{\partial x}$ and v terms:

$$2\alpha + a = 0 \implies \alpha = -\frac{a}{2},$$

$$\beta = \alpha^2 + a\alpha + b = \frac{a^2}{4} - \frac{a^2}{2} + b = -\frac{a^2}{4} + b,$$

gives the heat equation:

$$\frac{\partial v}{\partial t} = \frac{\partial^2 v}{\partial x^2}. \tag{2.27}$$

Hence for constants a,b the parabolic equation can always be reduced to the heat equation. However, we may also investigate how the heat equation changes for a non-constant, time dependent parameter.

2.3.2 Time Dependent Variables in the Heat Equation

Let the generalised form for a second order equation where $c(t)$ is a non-negative, time dependent parameter be given by:

$$c(t)\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}. \tag{2.28}$$

The transformed time variable τ is:

$$\tau(t) = \int \frac{dt}{c(t)},$$

$$\frac{d\tau}{dt} = \frac{1}{c(t)} \quad \Rightarrow \quad dt = c(t)d\tau.$$

Using the chain rule for the time derivative:

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial \tau}\frac{d\tau}{dt} = \frac{1}{c(t)}\frac{\partial u}{\partial \tau},$$

Substituting this result back into equation (2.28) it simplifies to the heat equation [5, Exercise 4, p. 103]:

$$c(t)\left(\frac{1}{c(t)}\frac{\partial u}{\partial \tau}\right) = \frac{\partial u}{\partial \tau} = \frac{\partial^2 u}{\partial x^2}.$$

Therefore the second order equation with a non-constant, time-dependent parameter also reduces to the heat equation. Now suppose that in the Black-Scholes model σ^2 and r^2 both depend on time, we derive an explicit solution in the next section assuming that $\frac{r}{\sigma^2}$ is a constant.

2.3.3 Black Scholes Solution with Time Dependent Variables

Under the following assumptions, where A is a constant, we will derive an explicit solution for the Black-Scholes formulae [5, Exercise 4, p. 103]:

- $\sigma = \sigma(t)$,
- $r = r(t)$,
- $\frac{r(t)}{\sigma^2(t)} = A$.

The initial Black Scholes PDE is:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2(t)S^2\frac{\partial^2 C}{\partial S^2} + r(t)S\frac{\partial C}{\partial S} - r(t)C = 0, \quad (S,t) \in (0,\infty)\times(0,T). \tag{2.29}$$

Under a similar change of variables seen in equation (2.13) we will now use:

$$S = Ke^x, \quad t = T - \tau, \quad C(x,t) = Kv(x,\tau), \tag{2.30}$$

we have:

$$\frac{\partial v}{\partial \tau} = \frac{1}{2}\sigma^2(\tau)\frac{\partial^2 v}{\partial x^2} + \left(r(\tau) - \frac{1}{2}\sigma^2(\tau)\right)\frac{\partial v}{\partial x} - r(\tau)v.$$

Changing the time scaling with the new time $\hat{\tau}$, and through use of chain rule we find:

$$d\hat{\tau} = \frac{1}{2}\sigma^2(\tau)d\tau \quad \Rightarrow \quad \hat{\tau} = \int_0^\tau \frac{1}{2}\sigma^2(s)ds,$$

which gives:

$$\frac{\partial v}{\partial \tau} = \frac{\partial v}{\partial \hat{\tau}}\frac{\partial \hat{\tau}}{\partial \tau} = \frac{1}{2}\sigma^2(\tau)\frac{\partial v}{\partial \hat{\tau}}.$$

Dividing through by $\frac{1}{2}\sigma^2(\tau)$ on both sides we have:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{\partial^2 v}{\partial x^2} + \left(\frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})} - 1\right)\frac{\partial v}{\partial x} - \frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})}v.$$

From the assumption above that $r(t)/\sigma^2(t) = A$ where A is some constant we have:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{\partial^2 v}{\partial x^2} + (2A-1)\frac{\partial v}{\partial x} - 2Av. \tag{2.31}$$

We then try to predict v in the same way as we did when deriving the analytical solution through the PDE, by trying to find α and β when v is:

$$v = e^{\alpha x + \beta \hat{\tau}} u(x, \hat{\tau}).$$

We then choose $\alpha = \frac{1}{2}(1-2A)$, $\beta = -\frac{1}{4}(2A+1)^2$, which simplifies to the heat equation:

$$\frac{\partial u}{\partial \hat{\tau}} = \frac{\partial^2 u}{\partial x^2}. \tag{2.32}$$

Using the initial condition at $\hat{\tau} = 0$ we can write $u(x,0) = e^{-\alpha x}(e^x - 1)^+$ for a call option. Therefore we have a similar integration as the one done in the derivation of the analytical solution using the PDE, where $u_0(x) = (e^{\frac{1}{2}(2A+1)x} - e^{\frac{1}{2}(2A-1)x})^+$ for the standard solution given by $u(x,\tau) = \frac{1}{2\sqrt{\pi\tau}} \int_{-\infty}^{\infty} u_0(s) e^{-\frac{(x-s)^2}{4\tau}} ds$.

Using the same change of variables as before with $x' = -\frac{(x-s)}{\sqrt{2\tau}}$:

$$u_0(x + x'\sqrt{2\tau}) = \left(e^{\frac{1}{2}(2A+1)(x+x'\sqrt{2\tau})} - e^{\frac{1}{2}(2A-1)(x+x'\sqrt{2\tau})}\right),$$

the solution becomes:

$$u(x,\tau) = \frac{1}{2\sqrt{\pi}} \int_{-\infty}^{\infty} u_0(x + x'\sqrt{2\tau}) e^{-x'^2/2} dx',$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \left[e^{\frac{1}{2}(2A+1)x + \frac{1}{2}(2A+1)x'\sqrt{2\tau}} - e^{\frac{1}{2}(2A-1)x + \frac{1}{2}(2A-1)x'\sqrt{2\tau}}\right] e^{-\frac{1}{2}x'^2} dx'.$$

Splitting the integral into two separate terms:

$$u(x,\tau) = \frac{1}{\sqrt{2\pi}} \left[\int_{-x/\sqrt{2\tau}}^{\infty} e^{\frac{1}{2}(A+1)(x+x'\sqrt{2\tau})} e^{-\frac{1}{2}x'^2} dx' - \int_{-x/\sqrt{2\tau}}^{\infty} e^{\frac{1}{2}(A-1)(x+x'\sqrt{2\tau})} e^{-\frac{1}{2}x'^2} dx'\right];$$

completing the square evaluates the integral as:

$$u(x,\tau) = \frac{1}{\sqrt{2\pi}} \left[e^{\frac{(A+1)x}{2} + \frac{(A+1)^2\tau}{4}} \int_{-x/\sqrt{2\tau}}^{\infty} e^{-\frac{1}{2}(x'-\gamma_1)^2} dx' - e^{\frac{(A-1)x}{2} + \frac{(A-1)^2\tau}{4}} \int_{-x/\sqrt{2\tau}}^{\infty} e^{-\frac{1}{2}(x'-\gamma_2)^2} dx'\right],$$

where $\gamma_1 = \frac{(A+1)\sqrt{2\tau}}{2}$, $\gamma_2 = \frac{(A-1)\sqrt{2\tau}}{2}$. From equation (2.5) it is clear that both integrals can now be evaluated as normal CDF's with means γ_1, γ_2 and variance 1. we convert the integrals to a standard normal CDF using the following solution:

$$u(x,\tau) = e^{\frac{(A+1)x}{2} + \frac{(A+1)^2\tau}{4}} N(d_1) - e^{\frac{(A-1)x}{2} + \frac{(A-1)^2\tau}{4}} N(d_2),$$

where d_1 and d_2 are the points where the standard normal CDF is evaluated at:

$$d_1 = x/\sqrt{2\tau} + \gamma_1,$$
$$d_2 = x/\sqrt{2\tau} + \gamma_2.$$

Using the following initial transformations we revert this solution into an explicit solution:

1. $S = Ke^x$ (so $x = \ln(S/K)$),

2. $C(S,t) = Kv(x,\tau)$,

3. $v(x,\tau) = e^{\alpha x+\beta\tau}u(x,\tau)$,

4. $\alpha = \frac{1-2A}{2}$, $\beta = -\frac{(2A+1)^2}{4}$.

The explicit solution is given below:

$$C(S,t) = S\Phi(d_1) - Ke^{-\int_t^T r(s)ds}\Phi(d_2), \tag{2.33}$$

where:

$$d_1 = \frac{\log(S/K) + \int_t^T \left(r(s) + \frac{1}{2}\sigma^2(s)\right) ds}{\sqrt{\int_t^T \sigma^2(s)ds}}, \quad d_2 = d_1 - \sqrt{\int_t^T \sigma^2(s)ds}. \tag{2.34}$$

This solution requires that $\frac{r(t)}{\sigma^2(t)}$ is a constant, therefore introducing a dividend payment would change the first derivative term violating the constant scaling of $\frac{r(t)}{\sigma^2(t)}$. Therefore this solution cannot be obtained when including a dividend yield.

2.3.4 Extensions to time-dependent financial properties

In this part we will only make the assumption that σ and r are dependent on time and use this to derive a solution to the first order PDE [5, Exercise 5, p. 103]. From section 2.2.4, we generated a second order PDE with σ and r dependent on time:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{\partial^2 v}{\partial x^2} + \left(\frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})} - 1\right)\frac{\partial v}{\partial x} - \frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})}v, \tag{2.35}$$

let $a(\hat{\tau}) = \left(\frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})} - 1\right)$ and $b(\hat{\tau}) = \frac{2r(\hat{\tau})}{\sigma^2(\hat{\tau})}$:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{\partial^2 v}{\partial x^2} + a(\hat{\tau})\frac{\partial v}{\partial x} - b(\hat{\tau})v. \tag{2.36}$$

The first-order PDE obtained from the equation for v may be reduced to the heat equation through a series of techniques, first by omitting the second-order term:

$$\frac{\partial v}{\partial \hat{\tau}} = a(\hat{\tau})\frac{\partial v}{\partial x} - b(\hat{\tau})v. \tag{2.37}$$

The general solution we propose here will have the form:

$$v(x,\hat{\tau}) = F\left(x + A(\hat{\tau})\right)e^{-B(\hat{\tau})}, \tag{2.38}$$

we can prove that this solution is valid by taking partial derivatives, starting with the left hand side of equation [5, Exercise 5, p. 104]:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{\partial F}{\partial \hat{\tau}}e^{-B(\hat{\tau})} + \frac{de^{-B(\hat{\tau})}}{d\hat{\tau}}F(x, A(\hat{\tau})).$$

This result can be simplified further using the following:

- $F(\cdot)$ is an arbitrary differentiable function,

- $A(\hat{\tau})$ and $B(\hat{\tau})$ satisfy:

$$\frac{dA}{d\hat{\tau}} = a(\hat{\tau}), \quad \frac{dB}{d\hat{\tau}} = b(\hat{\tau}). \tag{2.39}$$

The left-hand side is simplified further using the chain rule:

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{dA}{d\hat{\tau}} F'(x+A(\hat{\tau}))e^{-B(\hat{\tau})} - \frac{dB}{d\hat{\tau}} e^{-B(\hat{\tau})} F(x, A(\hat{\tau})),$$

using equation (2.39), this is simplified to:

$$\frac{\partial v}{\partial \hat{\tau}} = a(\hat{\tau}) F'(x+A(\hat{\tau}))e^{-B(\hat{\tau})} - b(\hat{\tau}) e^{-B(\hat{\tau})} F(x, A(\hat{\tau})).$$

Looking at the right hand side of equation (2.37), substituting in $v(x, \hat{\tau})$ and $\frac{\partial v(x,\hat{\tau})}{\partial x}$ gives the exact same result, hence equation (2.32) is a solution to the first order PDE. We will now seek a solution to the full PDE [5, Exercise 5, p. 104] in the form:

$$v(x, \hat{\tau}) = e^{-B(\hat{\tau})} V(\hat{x}, \hat{\tau}), \quad \text{where} \quad \hat{x} = x + A(\hat{\tau}), \tag{2.40}$$

such that V satisfies the heat equation:

$$\frac{\partial V}{\partial \hat{\tau}} = \frac{\partial^2 V}{\partial \hat{x}^2}. \tag{2.41}$$

Firstly we will compute the first time derivative for equation (2.36):

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{dV(\hat{x}, \hat{\tau})}{d\hat{\tau}} e^{-B(\hat{\tau})} - \frac{d\hat{B}}{d\hat{\tau}} e^{-B(\hat{\tau})} V(\hat{x}, \hat{\tau}),$$

$$\frac{\partial v}{\partial \hat{\tau}} = \frac{dV(\hat{x}, \hat{\tau})}{d\hat{\tau}} e^{-B(\hat{\tau})} - b(\hat{\tau}) e^{-B(\hat{\tau})} V(\hat{x}, \hat{\tau}).$$

V is dependent on $\hat{\tau}$ explicitly and implicitly through $\hat{x}$ shown by equation (2.40), so the derivative of V with respect to $\hat{\tau}$ can be represented in the following form:

$$\frac{\partial}{\partial \hat{\tau}} V(\hat{x}, \hat{\tau}) = \underbrace{\frac{\partial V}{\partial \hat{x}} \frac{d\hat{x}}{d\hat{\tau}}}_{\text{Implicit dependence}} + \underbrace{\frac{\partial V}{\partial \hat{\tau}}}_{\text{Explicit dependence}}.$$

From equation (2.40) it can be seen that $\frac{d\hat{x}}{d\hat{\tau}} = \frac{dA}{d\hat{\tau}}$, so we have:

$$\frac{\partial v}{\partial \hat{\tau}} = e^{-B(\hat{\tau})} \left(\frac{\partial V}{\partial \hat{x}} \frac{dA}{d\hat{\tau}} + \frac{\partial V}{\partial \hat{\tau}} \right) - b(\hat{\tau}) e^{-B(\hat{\tau})} V,$$

$$\frac{\partial v}{\partial \hat{\tau}} = e^{-B(\hat{\tau})} \left(a(\hat{\tau}) \frac{\partial V}{\partial \hat{x}} + \frac{\partial V}{\partial \hat{\tau}} \right) - b(\hat{\tau}) e^{-B(\hat{\tau})} V.$$

The first spatial derivative is given by:

$$\frac{\partial v}{\partial x} = e^{-B(\hat{\tau})} \frac{\partial V}{\partial \hat{x}},$$

alongside the second spatial derivative:

$$\frac{\partial^2 v}{\partial x^2} = e^{-B(\hat{\tau})} \frac{\partial^2 V}{\partial \hat{x}^2}.$$

Substituting all these terms into equation (2.36) we have:

$$e^{-B(\hat{\tau})}\left(a(\hat{\tau})\frac{\partial V}{\partial \hat{x}}+\frac{\partial V}{\partial \hat{\tau}}\right)-b(\hat{\tau})e^{-B(\hat{\tau})}V=e^{-B(\hat{\tau})}\frac{\partial^2 V}{\partial \hat{x}^2}+a(\hat{\tau})e^{-B(\hat{\tau})}\frac{\partial V}{\partial \hat{x}}-b(\hat{\tau})e^{-B(\hat{\tau})}V.$$

Dividing through by $e^{-B(\hat{\tau})}$ then cancelling the $b(\hat{\tau})$ and $a(\hat{\tau})$ terms on each side gives the heat equation:

$$\frac{\partial V}{\partial \hat{\tau}}=\frac{\partial^2 V}{\partial \hat{x}^2}. \tag{2.42}$$

In this chapter we introduced the Black-Scholes pricing framework alongside the PDE for pricing a call option (2.7). We showed how to derive an analytical solution from the PDE for the price of an option that includes a dividend continuous yield, D. We then looked at various extensions of the Black-Scholes PDE and the particular cases in which it can be reduced to the heat equation to then be solved analytically. In the next section we will introduce the inverse problem for one, two and three unknowns.

Chapter 3

The Inverse Problem: Fixed Properties

In the direct problem we specify the parameters (K, σ, r, D, T) and we seek the option premium, $C(S,t)$. We now specify the call option premium, $C(S,t)$ and attempt to recover constant values for σ, r and D. The parameters K and T can be treated as known constant properties in the inverse problem.

Before we plotted equation (2.25) as a function of stock price, S strike price, K and time, t. In this chapter we will do the same for the remaining variables, treating each as if they are unknown parameters to be solved for in equation (2.25).

3.1 One Unknown Property

Using Python we plot the call option premium, $C(S,t)$ against the volatility, σ for three different stock prices, $S \in \{90, 110, 130\}$:

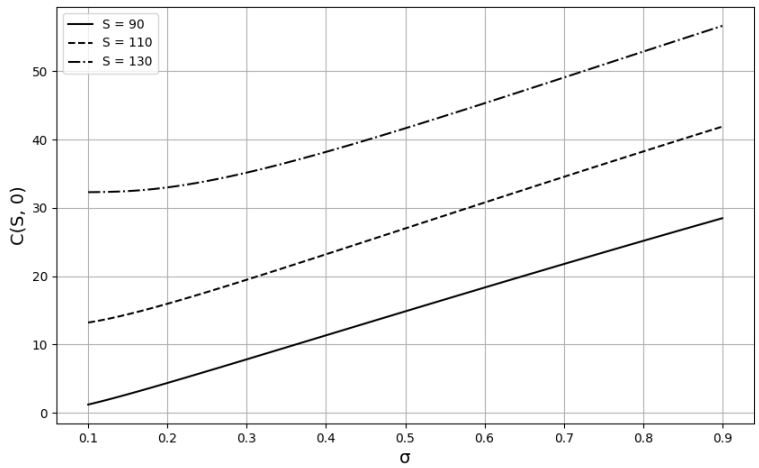

Figure 3.1: The call option premium, $C(S,0)$ shown by equation (2.25) plotted at $t = 0$ as a function of volatility, σ for three different stock prices: $S \in \{90, 110, 130\}$, $K = 100$, $r = 0.05$, $D = 0.02$ and $T = 1$.

Figure 3.1 shows the increasing relationship between the volatility and option price when the remaining six parameters are held constant. In markets this is attributed to the limited downside and unlimited upside of high volatility in European call options [17], shown in figure 2.1. The upper and lower bounds of the call option premiums for $\sigma \in [0.1, 0.9]$ are also found using Python:

$$\begin{cases} C(90,0) \in [1.19, 28.49], \\ C(110,0) \in [13.21, 41.91], \\ C(130,0) \in [32.31, 56.67]. \end{cases} \tag{3.1}$$

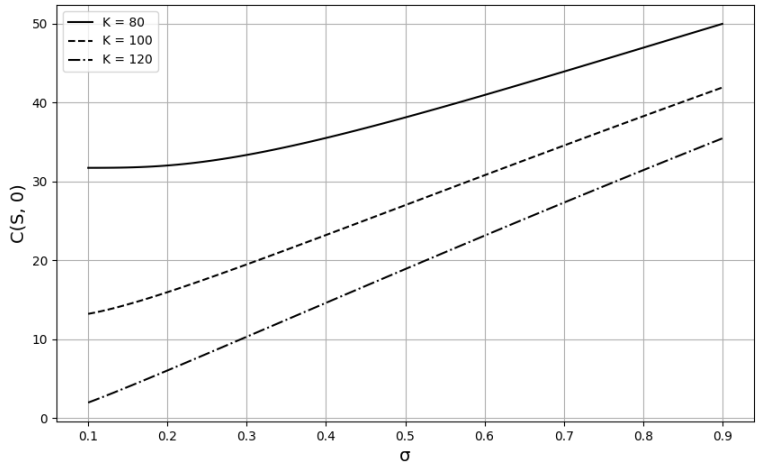

Figure 3.2: The call option premium, $C(S,0)$ shown by equation (2.25) is plotted at $t = 0$ as a function of volatility, σ for three different strike prices: $K \in \{80, 100, 120\}$, $S = 110$, $r = 0.05$, $D = 0.02$ and $T = 1$.

Figure 3.2 illustrates the dependence on σ for a single stock price, $S = 110$ and $K \in \{80, 100, 120\}$. The trend is consistent with Figure 3.1 and the higher strike price options have lower call option premiums. The upper and lower bounds for $\sigma \in [0.1, 0.9]$ here are found using python:

$$\begin{cases} C(110,0)(K = 80) \in [31.72, 49.98], \\ C(110,0)(K = 100) \in [13.21, 41.91], \\ C(110,0)(K = 120) \in [1.96, 35.45]. \end{cases} \tag{3.2}$$

We will now plot the call option premium as a function of the risk free interest rate, r in Python, keeping the remaining 6 parameters constant and unchanged, for the three different stock prices: $S \in \{90, 110, 130\}$.

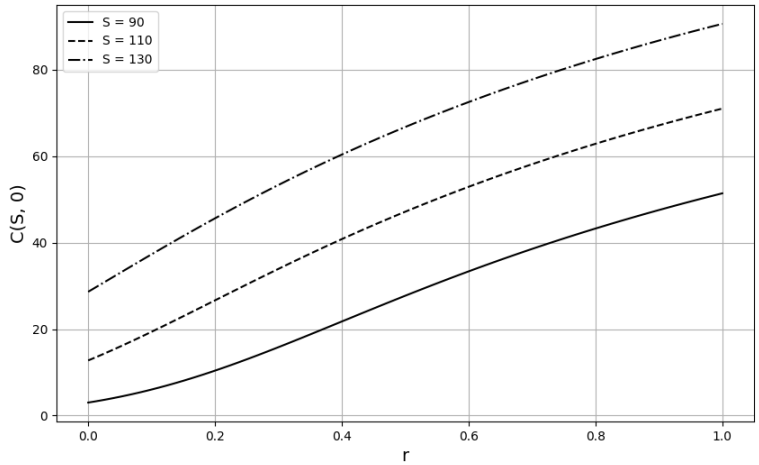

Figure 3.3: The call option premium, $C(S,0)$ shown by equation (2.25) is plotted at $t = 0$ as a function of the risk free interest rate, r for three different stock prices: $S \in \{90, 110, 130\}$, $K = 100$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 3.3 shows the increasing relationship between the call option premium, $C(S,0)$ and the risk-

free interest rate, r. This expected because investors will save money when borrowing rates are high, by borrowing money to buy call options rather than borrowing money to buy the underlying stock [17]. The upper and lower bounds of the call option premium for $r \in [0,1]$ are computed using Python:

$$\begin{cases} C(90,0) \in [3.02, 51.43], \\ C(110,0) \in [12.77, 71.03], \\ C(130,0) \in [28.66, 90.64]. \end{cases} \tag{3.3}$$

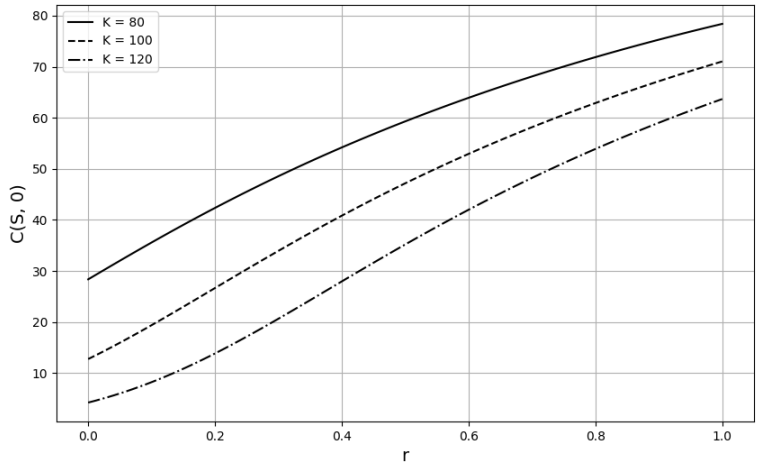

Figure 3.4: The call option premium, $C(S,0)$ shown by equation (2.25) is plotted at $t = 0$ as a function of the risk free interest rate, r for three different strike prices: $K \in \{80, 100, 120\}$, $S = 110$, $\sigma = 0.2$, $D = 0.02$ and $T = 1$.

Figure 3.4 shows the dependence on r when the stock price is fixed at $S = 110$ and $K \in \{80, 100, 120\}$, while the other variables are unchanged. The two plots from Figure 3.3 and Figure 3.4 have the same overall shape. Expectedly the higher strike price trajectories have lower call option premiums. The upper and lower bounds for $r \in [0,1]$ here are computed using Python:

$$\begin{cases} C(110,0)(K = 80) \in [28.37, 78.39], \\ C(110,0)(K = 100) \in [12.77, 71.03], \\ C(110,0)(K = 120) \in [4.25, 63.68]. \end{cases} \tag{3.4}$$

Finally we plot the call option premium as a function of the dividend yield, D in Python, keeping the remaining 6 parameters constant and unchanged, for the three different stock prices: $S \in \{90, 110, 130\}$.

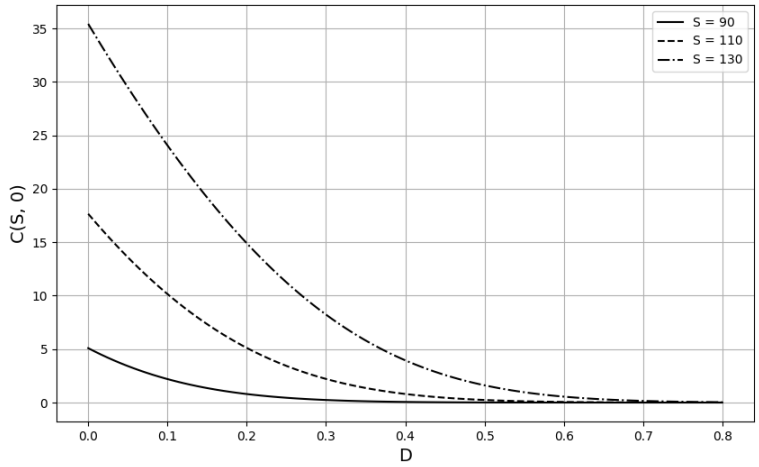

Figure 3.5: The call option premium, $C(S,0)$ shown by equation (2.25) is plotted at $t = 0$ as a function of dividend yield, D for three different stock prices: $S \in \{90, 110, 130\}$, $K = 100$, $\sigma = 0.2$, $r = 0.05$ and $T = 1$.

Figure 3.5 shows the decreasing relationship between the call option premium, $C(S,0)$ and the dividend yield, D, showcasing how the stock price usually falls after dividends are issued [17]. Again the higher stock price lines have a higher call option premium. The upper and lower bounds of the call option premium for $D \in [0, 0.8]$ are computed using Python:

$$\begin{cases} C(90,0) \in [5.09, 2.50 \times 10^{-5}], \\ C(110,0) \in [17.66, 1.92 \times 10^{-3}], \\ C(130,0) \in [35.44, 3.60 \times 10^{-2}]. \end{cases} \tag{3.5}$$

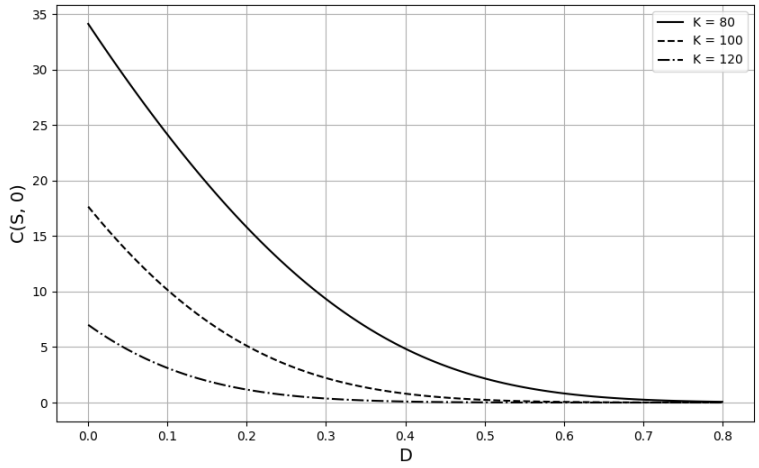

Figure 3.6: The call option premium, $C(S,0)$ shown by equation (2.25) is plotted at $t = 0$ as a function of dividend yield, D for three different strike prices: $K \in \{80, 100, 120\}$, $S = 110$, $\sigma = 0.2$, $r = 0.05$ and $T = 1$.

Figure 3.6 shows the dependence on D when the stock price is fixed at $S = 110$ and $K \in \{80, 100, 120\}$, while the other variables are unchanged. The upper and lower bounds for $D \in [0, 0.8]$ here are computed using Python:

$$\begin{cases} C(110,0)(K=80) \in [34.14, 6.73\times10^{-2}], \\ C(110,0)(K=100) \in [17.66, 1.92\times10^{-3}], \\ C(110,0)(K=120) \in [7.00, 4.63\times10^{-5}]. \end{cases} \tag{3.6}$$

The upper and lower bounds for the call option premium were found by individually plotting the call option premium as a function of σ, r and D. The upper and lower bounds of the call option premiums define where valid solutions to σ, r and D exist, meaning that inputs of option premiums outside of their bounds yield non-existent solutions.

3.2 Two Unknown Properties

We will now solve for a combination of two of the unknown variables by generating surface plots in Python for the call option premium of the following functions: $C(r,\sigma)$, $C(D,r)$, $C(D,\sigma)$.

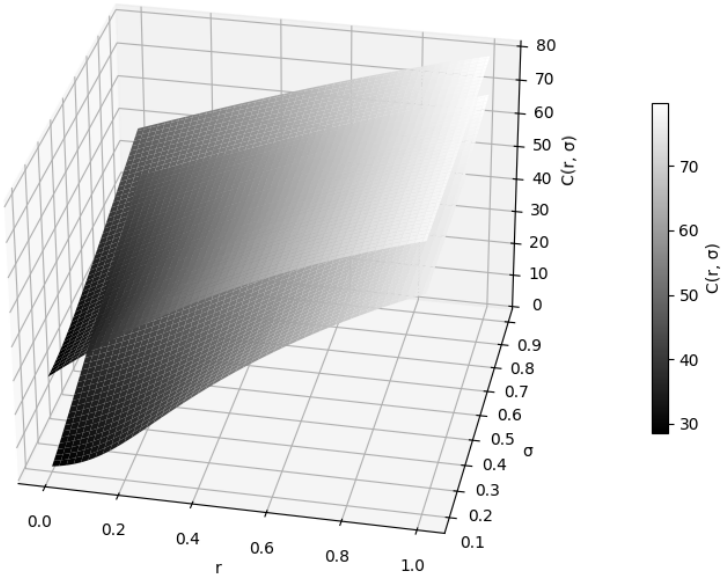

Figure 3.7: The call option premium, $C(r,\sigma)$ shown by equation (2.25) is plotted using a surface plot as a function of the risk free interest rate, r and volatility, σ for the following, $K \in \{80, 120\}$, $S = 110$, $t = 0$, $D = 0.02$ and $T = 1$.

Figure 3.7 shows that higher interest rates and volatility result in higher call option premiums. The steepest premium growth occurs with high volatility and high interest rates, while low volatility and low interest rates yield minimal premiums. We use the appendix code (designed for three unknowns in the next section) to invert the call option premiums and solve for the two unknowns r and σ. The premiums are computed using equation (2.6), we recover r and σ by inverting that same equation for the variables specified in the Python code.

$$\begin{cases} C(110,80,t=0,T=1,r^*,\sigma^*,D^*) \approx 32.0210 \\ C(110,120,t=0,T=1,r^*,\sigma^*,D^*) \approx 6.0332 \end{cases} \Rightarrow \begin{cases} r = 0.0500 \\ \sigma = 0.2000 \end{cases} \tag{3.7}$$

The target variables: $\{r^*,\sigma^*,D^*\} = \{0.05, 0.2, 0.02\}$ were used to generate the call option premiums in equation (2.6) (to four decimal places). Using this method there is zero error (in four decimal places) between the target and recovered values. We now present the same plot for σ and D:

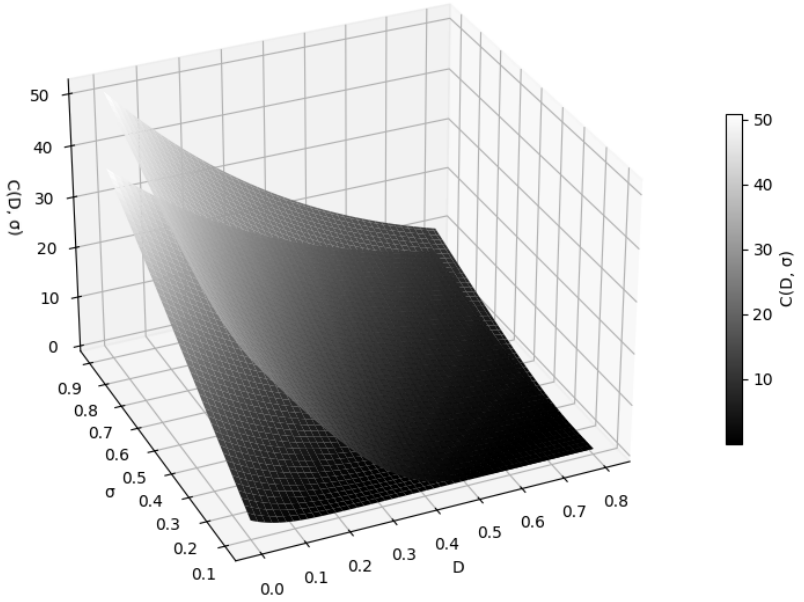

Figure 3.8: The call option premium, $C(D,\sigma)$ shown by equation (2.25) is plotted using a surface plot as a function of dividend yield, D and volatility, σ for the following, $K \in \{80,120\}$, $S = 110$, $t = 0$, $r = 0.05$ and $T = 1$.

Figure 3.8 shows the call option premium as a function of σ and D. Premiums increase with higher volatility and lower dividend payouts, with the highest values occurring in the high-volatility, low-dividend region. Conversely, the lowest premiums result from high dividend payouts and low volatility. We use Python to solve for σ and D in a similar manner as equation (3.7), holding all other parameters the same:

$$\begin{cases} C(110,80,t=0,T=1,r^*,\sigma^*,D^*) \approx 32.0210 \\ C(110,120,t=0,T=1,r^*,\sigma^*,D^*) \approx 6.0332 \end{cases} \quad \Rightarrow \quad \begin{cases} \sigma = 0.2000 \\ D = 0.0200 \end{cases} \tag{3.8}$$

Using the same target values as reference, σ and D are accurate with no error (in four decimal places) from the initial values. We now present the same plot for r and D:

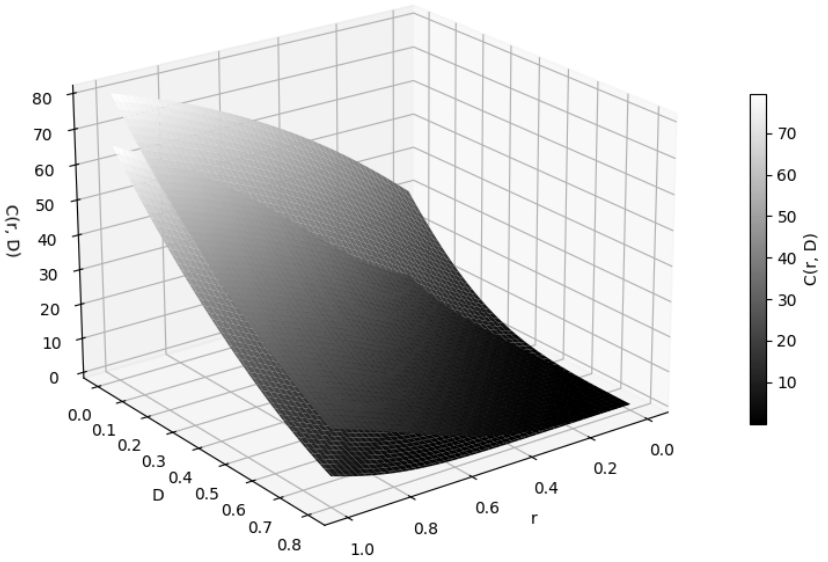

Figure 3.9: The call option premium, $C(r,D)$ shown by equation (2.25) is plotted using a surface plot as a function of the risk free interest rate, r and dividend yield, D for the following, $K \in \{80,120\}$, $S = 110$, $t = 0$, $\sigma = 0.2$ and $T = 1$.

Figure 3.9 shows the call option premium as a function of interest rate, r and dividend yield, D. Premiums increase with higher interest rates and decrease with higher dividend yields. The highest premiums occur when interest rates are high and dividend yields are low, while the lowest premiums result from low interest rates and high dividend yields.

We solve for the two unknowns r and D by inverting the system of equations, holding other parameters constant. The implementation in Python is shown below:

$$\begin{cases} C(110,80,t=0,T=1,r^*,\sigma^*,D^*) \approx 32.0210 \\ C(110,120,t=0,T=1,r^*,\sigma^*,D^*) \approx 6.0332 \end{cases} \quad \Rightarrow \quad \begin{cases} r = 0.0500 \\ D = 0.0200 \end{cases} \tag{3.9}$$

We notice that again the recovered values of r and D have no error (in four decimal places) from the initial values. In the next section we show that this is not always the case. When the number of unknowns increases and we introduce error into our call option prices, the system of equations becomes more unstable.

3.3 Three Unknown Properties

In this section, we solve a non-linear system of three equations for the three unknowns: σ, r, and D. The system uses call option premiums computed at three different stock prices, $S \in \{90,110,130\}$, while keeping the other parameters fixed: $K = 100$, $t = 0$, $T = 1$. The (true) target values are $r^* = 0.05$, $\sigma^* = 0.2$, and $D^* = 0.02$. First, we use these target values to fabricate some call option premiums for each stock price, then we assume the target values are unknown and attempt to recover them by inverting the system of equations:

$$\begin{cases} C(90,100,t=0,T=1,r^*,\sigma^*,D^*) = 4.3599 \\ C(110,100,t=0,T=1,r^*,\sigma^*,D^*) = 15.9613 \\ C(130,100,t=0,T=1,r^*,\sigma^*,D^*) = 33.0040 \end{cases} \quad \Rightarrow \quad \begin{cases} r = 0.0500 \\ \sigma = 0.2000 \\ D = 0.0200 \end{cases} \tag{3.10}$$

Equation (3.10) recovers values for σ, r, and D, to four decimal places the solution exactly matches the target values but at the precision of 15 decimal places the solution differs to the target values. The Python function `fsolve()` solves equation (3.10), taking the initial guess parameter `x0` to set the starting position for root-finding. For this non-linear system, multiple roots may exist (some outside the expected range of $[0,1]$), hence the initial guess of $[0.5,0.5,0.5]$ is used (any initial guess within $[0,1]$ would suffice).

To simulate noisy market data (where observed prices deviate from the theoretical Black-Scholes price), we now round the call option premiums to one decimal place. This tests the numerical stability of the solution with small levels of noise. The system of equations is implemented in Python using `fsolve()` to solve the non-linear system:

$$\begin{cases} C(90,100,t=0,T=1,r^*,\sigma^*,D^*) \approx 4.4 \\ C(110,100,t=0,T=1,r^*,\sigma^*,D^*) \approx 16.0 \\ C(130,100,t=0,T=1,r^*,\sigma^*,D^*) \approx 33.0 \end{cases} \Rightarrow \begin{cases} r = 0.0535 \\ \sigma = 0.2008 \\ D = 0.0266 \end{cases} \tag{3.11}$$

Although the system introduces error when calculating σ, r, and D, the results remain accurate. Volatility (σ) had the smallest error, while r and D exhibit larger errors. The initial guess here is the same guess used in equation (3.10).

We now examine the case where the strike price varies, $K \in \{80,100,120\}$, with a constant stock price $S = 110$. The call option premiums are rounded to one decimal place, and the remaining parameters are unchanged. The Python implementation solves the system for σ, r, and D as follows:

$$\begin{cases} C(110,80,t=0,T=1,r^*,\sigma^*,D^*) \approx 32.0 \\ C(110,100,t=0,T=1,r^*,\sigma^*,D^*) \approx 16.0 \\ C(110,120,t=0,T=1,r^*,\sigma^*,D^*) \approx 6.0 \end{cases} \Rightarrow \begin{cases} r = 0.0682 \\ \sigma = 0.1955 \\ D = 0.0325 \end{cases} \tag{3.12}$$

Using the initial guess from equation (3.10) and varying K, the volatility had the smallest error, while r and D had larger errors; varying K with fixed S introduced greater overall error in σ, r, and D.

Instability occurs as the option approaches maturity for $t > 0.9$ and $T = 1$, σ, r, and D exhibit greater error, and the solution fails to converge. This is because `fsolve()` computes the Jacobian matrix and partial derivatives [18] of the Black-Scholes formula near the money. As shown in figure 2.6, the piecewise linear payoff likely causes computational difficulties. A similar issue occurs for longer maturities ($T > 5$), where the solution becomes unstable again.

To address this problem, we generate random initial guesses for `x0` using a log-normal distribution (mean = -1, variance = 3), favouring values near zero and producing few beyond one. We sample 800 points per variable and refine the 800 total solutions by finding the most often re-occurring solution. This found the optimal solution, dismissing any non-converging roots that returned solutions close to their initial guess. For options with a long maturity ($T = 10$ years), using generic initial guesses is insufficient, we apply the revised method for three stock prices $S \in \{90,110,130\}$, and fixed $K = 100$ when all remaining parameters are unchanged:

$$\begin{cases} C(90,100,t=0,T=10,r^*,\sigma^*,D^*) \approx 23.9 \\ C(110,100,t=0,T=10,r^*,\sigma^*,D^*) \approx 36.8 \\ C(130,100,t=0,T=10,r^*,\sigma^*,D^*) \approx 50.8 \end{cases} \Rightarrow \begin{cases} r = 0.0555 \\ \sigma = 0.1940 \\ D = 0.0222 \end{cases} \tag{3.13}$$

The optimal initial guess was $[0.0222,0.1740,0.0114]$ and the total magnitude of error of between the solution and the target values was 0.0085. Now using three different strike prices: $K \in \{80,100,120\}$, with the constant stock price $S = 110$ we use Python to solve:

$$\begin{cases} C(110,80,t=0,T=10,r^*,\sigma^*,D^*) \approx 45.1 \\ C(110,100,t=0,T=10,r^*,\sigma^*,D^*) \approx 36.8 \\ C(110,120,t=0,T=10,r^*,\sigma^*,D^*) \approx 29.9 \end{cases} \Rightarrow \begin{cases} r = 0.0555 \\ \sigma = 0.1837 \\ D = 0.0212 \end{cases} \tag{3.14}$$

The optimal initial guess used in finding this solution was $[0.0491, 0.1851, 0.0048]$ and the total magnitude of error of our solution from the initial values for $r^* = 0.05$, $\sigma^* = 0.2$, $D^* = 0.02$ was 0.0172.

This concludes our analysis of the methods for recovering σ, r, and D from three distinct call option premiums, focusing on near-the-money options. Varying the strike or stock price had minimal impact on accuracy; the primary determinant was the time to maturity. For very short or long maturities, crude initial guesses failed. We addressed this using a method that randomly generated initial guesses. This worked for option prices with little deviation from the Black-Scholes price.

Chapter 4

The Inverse Problem: Variable Properties

4.1 Volatility Smiles and Skews

The method described in chapter 3 finds constant values for σ, r, and D under the Black-Scholes model, however extensive research on non-constant volatility models has been the interest of many practitioners. Non-constant volatility models describe observable trends between the implied volatility of an option with the parameters of an option, K and T (illustrated by skews or smiles). Using Python we plot the implied volatility (measured as "the estimated volatility of the option strike over the period of the option" [19]) for calls and puts on the S&P500:

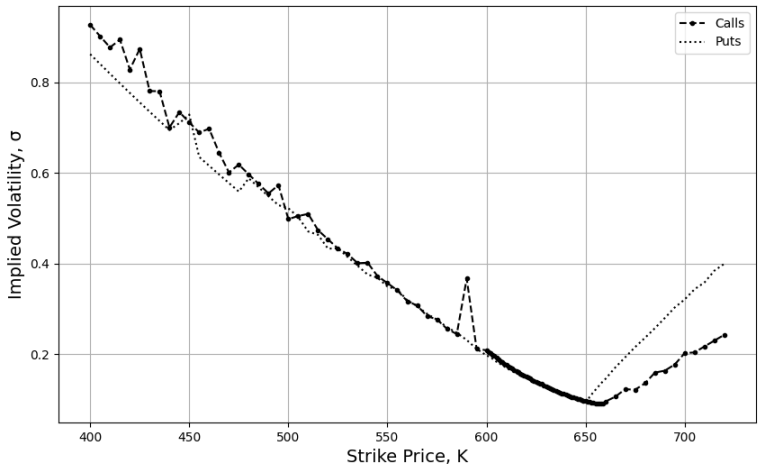

Figure 4.1: Implied volatility, σ plotted against the strike price K for call and put options on the S&P500 SPDR (SPY). The market values are taken on 25-07-2025 and the expirations are for 04-08-2025 (w) [19].

Figure 4.1 shows the implied volatility of weekly issued options (w) against the strike, measured on, 25-07-2025 (at the closing price, \$637.10) for the expiry date, 04-08-2025. The options exhibit volatility smiles for both call and put options for the strike price range, $K = \{400, 720\}$. We notice that the implied volatility approaches a minimum for strikes near the closing market price. While the trend is very similar for strikes below $K = \$650$, the gradient becomes higher for the puts compared to the calls beyond $K = \$650$. We now show a similar plot but for a maturity much further in the future:

Figure 4.2: Implied volatility, σ plotted against the strike price K for call and put options on the S&P500 SPDR (SPY). The market values are taken on 25-07-2025 and the expirations are for 17-12-2027 (m) [19].

Figure 4.2 shows the implied volatility of monthly issued options (m) against the strike, measured on, 25-07-2025 (at the same closing price, \$637.10) for the expiry date 17-12-2027. Options exhibit a volatility smile for puts but a skew for calls when plotted across strikes in the range $K = \{200, 995\}$. We notice the minimum implied volatility is situated at a higher strike in figure 4.2 and the rate of change of volatility against the strike in the range $K \in \{400, 600\}$ is much higher in figure 4.1. We now show the plot for the implied volatility against the maturity for a range of options when the strike price is fixed on the same index:

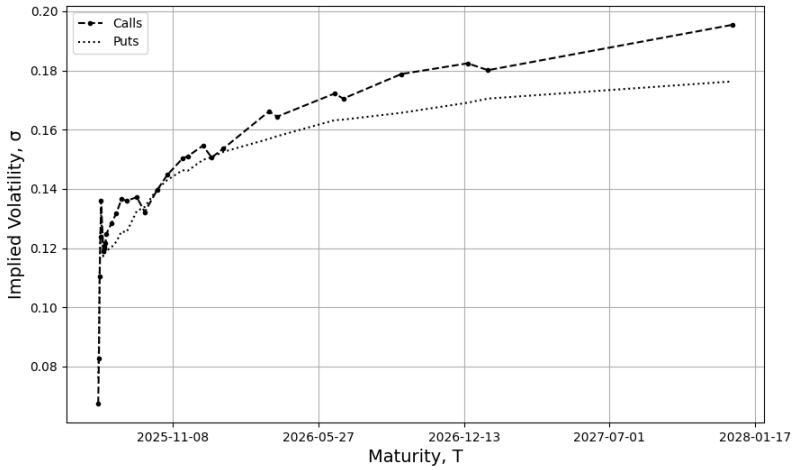

Figure 4.3: Implied volatility, σ plotted against the maturity time, T for call and put options on the S&P500 SPDR (SPY). The market values are taken for the fixed strike of \$635 and the expirations range from 28-07-2025 to 17-12-2027 [19].

Figure 4.3 shows the implied volatility against the maturity time for calls and puts that were near the money ($K =$ \$635 and S = \$637.10). We notice that the implied volatility increases over time with a sharp increase at shorter maturities. The data for figures 4.1, 4.2 and 4.3 were accessed from [19].

4.2 Dupire's Equation

Dupire's equation may be used to model the relationships in figures 4.1, 4.2, and 4.3, by inverting the second-order forward PDE, we may find the local volatility dependent on the strike and maturity [6]. Unlike the backwards Black-Scholes PDE, Dupire's equation is a forward PDE, meaning that small changes in the call option premium do not create disproportionately large changes in σ [5]. We now present the general form of Dupire's PDE [6]:

$$\frac{1}{2}\sigma^2(S,t)S^2\frac{\partial^2 C}{\partial K^2} = \frac{\partial C}{\partial T}. \tag{4.1}$$

Re-arranging equation (4.1) we may obtain the local volatility, $\sigma(S,t)$. Other models re-formulate Dupire's equation so that volatility, $\sigma(K,T)$ is a function of the strike and maturity:

$$\frac{1}{2}\sigma^2(K,T)K^2\frac{\partial^2 C}{\partial K^2} = \frac{\partial C}{\partial T}. \tag{4.2}$$

Equation (4.2) gives the local volatility dependent on the strike, K and maturity, T which useful for calibrating to the relationships shown in figures 4.1, 4.2 and 4.3. Another form of the model includes a drift, giving information on the risk-free rate, r and dividend yield, D, presented in [3] and [4]:

$$\frac{\partial C}{\partial T} = \frac{1}{2}\sigma^2(K,T)K^2\frac{\partial^2 C}{\partial K^2} + (D-r)K\frac{\partial C}{\partial K} - DC. \tag{4.3}$$

Dupire's equation has no analytical solution, but rearranging the PDE for volatility allows calibration of the local volatility to implied volatility. This requires calculating the partial derivatives of the call option premium. However, using simple finite differencing schemes may introduce arbitrage and unstable results due to discontinuities and a discrete range of strikes and maturities [8]. We therefore require numerical methods to replicate the trends in figures 4.1, 4.2 and 4.3.

4.3 Time Dependent Volatility

Local volatility models require calibration to the volatility smiles or skews in market data shown in Figures 4.1, 4.2, and 4.3. We begin by examining models where only time-dependent properties are unknown. Although volatility depends on both stock price and time, Bouchouev and Isakov demonstrate that by separating local volatility into time and stock price dependent components, the local volatility function can be recovered [8, p. 100].

$$\sigma(S,t) = \sigma(S)\rho(t). \tag{4.4}$$

In equation (4.4) $\sigma(S)$ is known and the time dependent part $\rho(t)$ is unknown. Bouchouev and Isakov propose using the Black-Scholes PDE (2.7) to recover the unknown $\rho(t)$. They pose that under zero drift $\mu = r - D = 0$ the time dependence in $\sigma(S,t)$ from the Black-Scholes PDE (2.7) can be removed [8, p R100], using the following variable changes:

$$\tau = \int_t^T \rho^2(\eta)d\eta \quad \text{and} \quad v(S,\tau) = C(S,t)e^{-r(T-t)}, \tag{4.5}$$

the Black-Scholes equation then reduces to [8, p R100]:

$$\frac{\partial v}{\partial \tau} = \frac{1}{2}\sigma^2(S)S^2\frac{\partial^2 v}{\partial S^2}. \tag{4.6}$$

Since the time dependence in equation (4.6) is now eliminated, equation (4.6) becomes much easier to solve [8, p R100]. The time dependent part $\rho(t)$ is found using the equation:

$$v\left(S_0, \int_0^T \rho^2(t)dt\right) = C_0(T)e^{rT}, \tag{4.7}$$

for the initial stock price S_0 and initial time 0. This model reconstructs the time-dependent volatility for market prices across all different maturities, but with the same strike price [8, p R100]. Under this model we cannot fully capture the dynamics of the volatility skews/smiles. In the next section we show how to recover stock price dependent volatility.

4.4 Stock Price Dependent Volatility

Bouchouev and Isakov present a stock price dependent volatility from the Black-Scholes-PDE (2.7), which computes the volatility as a finite sum [3, p L14]. The Black-Scholes PDE (2.7) is first transformed under the following variable changes:

$$\tau = T - t \quad \text{and} \quad y = \ln\frac{x}{K}, \tag{4.8}$$

and let:

$$C(y,\tau;K) = C(x,t;K), \quad a(y) = \sigma(x). \tag{4.9}$$

This transforms the Black-Scholes PDE (2.7) to [3, p L14]:

$$\frac{\partial C}{\partial \tau} = \frac{1}{2}a^2(y)\left(\frac{\partial^2 C}{\partial y^2} - \frac{\partial C}{\partial y}\right) + (R-D)\frac{\partial C}{\partial y} - RU, \quad (y,\tau) \in \mathbb{R}\times(0,T). \tag{4.10}$$

The solution to equation (4.10) is written in integral form, and then reduced to a finite sum to find $\sigma(K_i)$ where $i = 1,\ldots,n$. The result is an iterative volatility function that computes the volatility at the $m+1^{th}$ iteration, given the volatility at the m^{th} iteration [3]. The iterative scheme thus computes a volatility function consistent with both the stock-price process and market data.

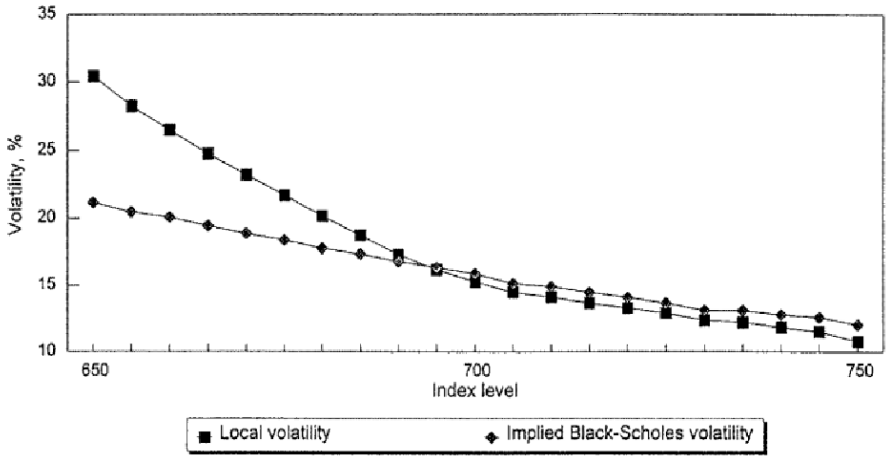

Figure 4.4: Local volatility plotted with the implied Black-Scholes volatility for a range of strikes on the S&P500 Index, 18 October 1996 [3, p L16].

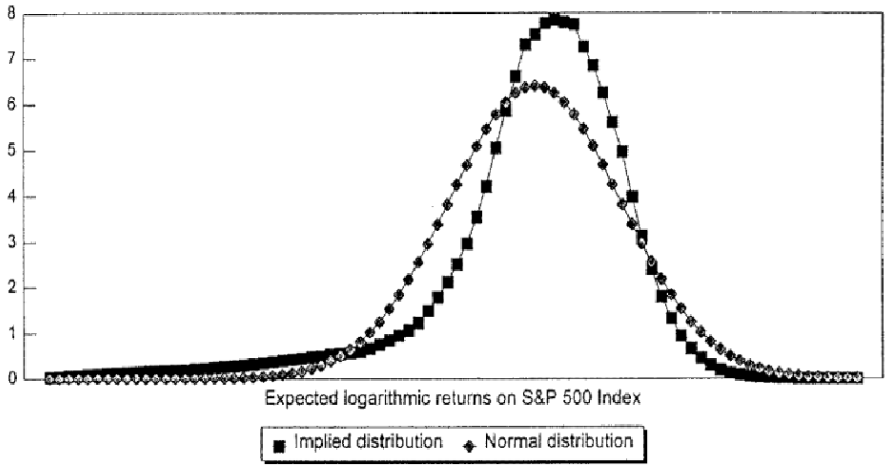

Figure 4.5: Distribution of the logarithmic returns plotted with the normal distribution for options on the S&P500 Index, 18 October 1996 [3, p L16].

The local volatility converges to the implied volatility over 10 iterations [3, p L16]. The distribution of the local volatility is also skewed further right to the normal distribution, showing how the expected logarithmic returns do not exactly match the Black-Scholes model.

Bouchouev and Isakov also present a stock-price dependent volatility model assuming that option prices are given for different strike prices and a fixed maturity [8, p R101]. Contrary to the time-dependent volatility model, they assume $\rho(t)$ is known and $\sigma(s)$ is unknown in equation (4.4). The solution is found by using Dupire's equation with a similar change of variables as shown above [8, p R101].

4.5 Time and Stock Price Dependent Volatility

However most models attempt to incorporate the information of the stock price and the time into the local volatility function, this method was first presented by Dupire in [6]. Dupire proposed the idea using call option premiums to recover the stock price diffusion process:

$$\frac{dS}{S} = r(t)dt + \sigma(t)dW.$$

The PDE he used is shown in equation (4.1). Dupire gave a framework to calibrate the local volatility to market data. Precise methods of doing this with the Black-Scholes implied total variance w [7, p 11] are shown below:

$$w(S_0,K,T) := \sigma_{BS}^2(S_0,K,T)T, \tag{4.11}$$

the Black-Scholes volatility is σ_{BS}. The Black-Scholes formula is transformed to obtain the future value of the option price under the variable change with the risk neutral drift, $\mu = r - D$ [7, p 9]:

$$y = \log\left(\frac{K}{F_T}\right) \quad \text{and} \quad F_t = S_0 \exp\left\{\int_0^T dt\mu(t)\right\}, \tag{4.12}$$

where F_T is the forward price of the stock at time zero and C_{BS} is the Black-Scholes option price:

$$C_{BS}(F_T,y,w) = F_T\left\{\Phi(d_1) - e^y\Phi(d_2)\right\} = F_T\left\{\Phi\left(-\frac{y}{\sqrt{w}} + \frac{\sqrt{w}}{2}\right) - e^y\Phi\left(-\frac{y}{\sqrt{w}} - \frac{\sqrt{w}}{2}\right)\right\}. \tag{4.13}$$

Under the same variable change, Dupire's equation with the local variance v_L transforms to [7, p 11]:

$$\frac{\partial C}{\partial T} = \frac{v_L}{2}\left\{\frac{\partial^2 C}{\partial y^2} - \frac{\partial C}{\partial y}\right\} + \mu(T)C \quad \text{where} \quad v_L = \sigma^2(S_0, K, T). \tag{4.14}$$

Taking partial derivatives of the Black-Scholes solution given in equation (4.13) for $\frac{\partial^2 C_{BS}}{\partial w^2}$, $\frac{\partial^2 C_{BS}}{\partial y \partial w}$ and $\frac{\partial^2 C_{BS}}{\partial y^2}$ - $\frac{\partial C_{BS}}{\partial y}$. We re-write the partial derivatives of Dupire's equation (4.14) using the chain rule and equate an expression for $\frac{\partial C}{\partial T} = \frac{\partial C}{\partial w}\frac{\partial w}{T}$ in the local volatility, with $\frac{\partial C_{BS}}{\partial T} = \frac{\partial C_{BS}}{\partial w}\frac{\partial w}{T}$ in the Black-Scholes implied volatility [7, p 12]. Because of the chosen variables the $\frac{\partial C_{BS}}{\partial w}$ terms cancel and we invert to get an expression for the local variance in terms of the Black-Scholes implied total variance [7, p 13]:

$$v_L = \frac{\frac{\partial w}{\partial T}}{1 - \frac{y}{w}\frac{\partial w}{\partial y} + \frac{1}{4}\left(-\frac{1}{4} - \frac{1}{w} + \frac{y^2}{w^2}\right)\left(\frac{\partial w}{\partial y}\right)^2 + \frac{1}{2}\frac{\partial^2 w}{\partial y^2}}. \tag{4.15}$$

Equation (4.15), obtained by transforming the Black-Scholes PDE with Dupire's equation, links the local variance to the derivatives of the Black-Scholes implied total variance, w. This extracts the local volatility surface directly from market prices, ensuring the model reflects market dynamics. The equation fits market skews, but when $\frac{\partial w}{\partial y} = 0$, local variance reduces to Black-Scholes implied variance and therefore fits to no skew [7, p 13].

Other discrete time methods by Derman and Kani use an adjusted binomial tree to recover local time and stock dependent volatility by fitting to market skews [2]. Recent proposals focus on numerically 'smoothing' the implied volatility surface using splines [10]. While not providing a fully parametric form for $\sigma(S,t)$, this method uses Dupire's partial derivatives and arbitrage constraints to produce an accurately calibrated volatility surface [10].

Chapter 5

Inversion of real-data

5.1 Picking Target Values

Using a similar method to chapter 3, we now apply the inverse method for call option premiums on the S&P500 SPDR (SPY) index, using the Black-Scholes solution to recover values for σ, r and D. We jointly calculate constant values for σ, r and D using future projections for r and σ, and historical data on D as the target values.

The options have strikes ranging from 200 to 995 and the data was measured on 25-07-2025 for the expiry date of 17-12-2027 (m) so $T-t = 2.389$ [19]. To find a target value for r we use the daily treasury par yield curve rates projected for assets with a maturity of 2 and 3 years [20]. Using the projected 2 and 3 year rates for all of the available dates in 2025 we interpolate between the two to find the approximate value for the risk-free return in 2.389 years. The mean target value for r with the given standard deviation is shown below:

$$r^* = 3.981\% \pm 0.196\%$$

The target value for the dividend yield, D was calculated using historical data on the S&P500 SPDR (SPY) index from [21]. We calculate the 3-month dividend yield by:

$$\text{Dividend Yield (\%)} = \frac{\text{Dividend Per Share}}{\text{Index Price}} \times 100 \tag{5.1}$$

The dividend on this index is historically paid every three months, calculating 9 dividend yields over a 2-year period, we find the target value for the 3-month dividend yield. The annual dividend is equal to the average 3-month dividend yield multiplied by four. Below shows the mean target value for D with the standard deviation for the error:

$$D^* = 1.337\% \pm 0.149\%$$

The Black-Scholes analytical formula calculates the call option premium using the continuous dividend yield; since the annual rate is small enough, it is approximately equal to the continuous rate. The estimate is based on historical data but the standard deviation from the mean captures the dividend estimate for 05/08/2025 of 1.22% [22]. We find the target value for σ by looking at the implied volatility in the dataset, which is a future estimate for the volatility of the option over its life [19]. Empirical and Black-Scholes scatter plots are produced in Python to show the call option premium as a function of the implied volatility for a range of different strikes:

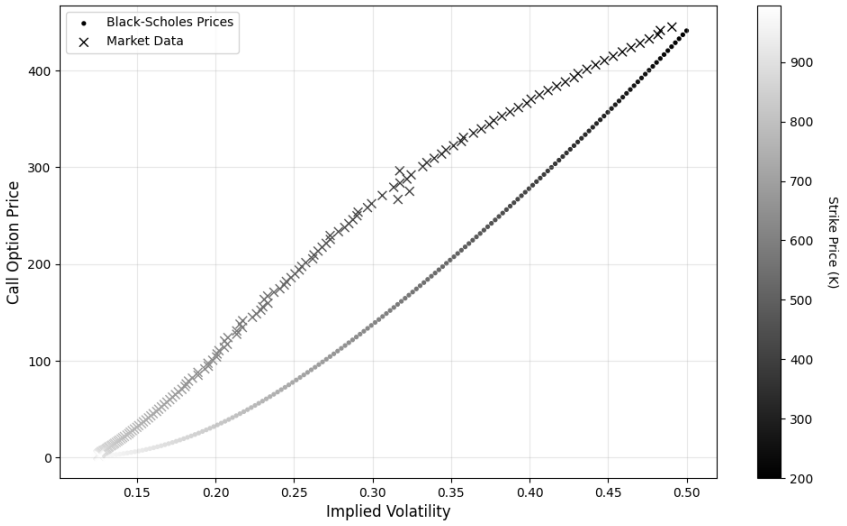

Figure 5.1: Call option Price plotted against the implied volatility for options with strikes ranging from 200 to 995, the crosses represent empirical market data and the points show the Black-Scholes price [19].

The Black-Scholes price is different to the empirical call option prices across the range of strikes and volatilities shown above. The Black Scholes fit, took the parameters: $S = \$637.10$ (the closing price on 25-07-2025), $T = 2.389$, $t = 0$, $r = 3.981\%$ and $D = 1.337\%$. A target value for the volatility σ^* can be picked for any one of the empirical prices on the plot. However, since the implied volatility increases as the strike price decreases for the call options (shown in figure 4.2), solving for constant values of σ, r and D introduces a degree of inaccuracy. For high and low strike price options the empirical prices match very close to the Black-Scholes price. There is also a smaller change in the call option price relative to changes in the strike price for high strike price options.

Hence we expect the inverse method to be accurate at very low or very high strikes, but less accurate for strikes close to the stock price $S = \$637.10$, since the empirical prices deviate further from the Black-Scholes price.

5.2 Applying the Inverse Method

Since the empirical data deviates much further from the Black-Scholes price (look at figure 5.1) compared to the data in chapter 3, we now solve the system of equations based on a least squares optimisation, with Tikhonov Regularisation. Tikhonov regularisation optimises the solution by balancing the payoff between solutions that minimise the least squares error (between the Black-Scholes and empirical option prices) with solutions that minimise the Tikhnov error. Below shows the new optimisation problem:

$$\min_{r,\sigma,D} \left\{ \underbrace{\sum_i (\text{BS Price}_i - \text{MarketPrice}_i)^2}_{\text{Least Squares}} + \underbrace{\lambda_r (r - r_{\text{ref}})^2 + \lambda_\sigma (\sigma - \sigma_{\text{ref}})^2 + \lambda_D (D - D_{\text{ref}})^2}_{\text{Tikhonov}} \right\} \tag{5.2}$$

Equation (5.2) shows how we recover optimal values for σ, r and D. The Black-Scholes price is given by BS Price_i, the market price is given by MarketPrice_i, for three different strikes $(i = K_1, K_2, K_3)$. The Tikhonov term has parameters: λ_r, λ_σ and λ_D which control how much weight is put on the regularisation term [23]. High values for λ_r, λ_σ and λ_D suggest that the solution will be over-regularised

and bias, whereas too low values will lead to unstable, inaccurate results. The terms: r_{ref}, σ_{ref} and D_{ref} are chosen values around which we expect to see the solution lie, these are picked using historical and future estimates (σ_{ref} will vary for different strikes).

The `least_squares()` Python function also takes an initial guess, set to any value between 0 to 1, for generality. This removes the need to generate a large set of random numbers for initial guesses. The parameter, `bounds` limits the range in which the solution can lie, here it is set between 0 and 1 for all three parameters. While interest rates may sometimes be negative in markets, this is a rare occurrence and is not relevant to our data. We solve for σ, r and D using the call option prices: $C = \{\$445.02, \$441.41, \$437.18\}$, and the known variables (which will be constant for the remainder of this section): $S = \$637.10$, $K = \{\$200, \$205, \$210\}$ and $T - t = 2.389$, on the S&P500 SPDR (SPY) index. Using Python the following set of equations are solved:

$$\begin{cases} C_{BS}(637.10, 200, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 445.02 \\ C_{BS}(637.10, 205, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 441.41 \\ C_{BS}(637.10, 210, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 437.18 \end{cases} \Rightarrow \begin{cases} r = 0.0403 \\ \sigma = 0.5194 \\ D = 0.0115 \end{cases} \tag{5.3}$$

The target values for σ, r and D here were:

$$r^* = 3.981\% \pm 0.196\%, \quad \sigma^* = 48.53\% \pm 0.48\% \quad \text{and} \quad D^* = 1.337\% \pm 0.149\%. \tag{5.4}$$

Equation (5.2), finds the solution of σ, r and D in equation (5.3). The weight parameters: $\{\lambda_r, \lambda_\sigma, \lambda_D\}$ in the Tikhonov regularisation part of the equation were $\{100, 0.01, 0\}$ respectively. The reference values in equation (5.2), $\{r_{\text{ref}}, \sigma_{\text{ref}}, D_{\text{ref}}\}$ were $\{0.04, 0.25, 0.01\}$ respectively and the recovered values were all close to the target values. We chose the reference values to be slightly different to the target values so that the robustness of the model could be tested.

Now we solve for options at the money, where $C = \{97.60, 94.53, 91.75\}$ for the corresponding strike prices: $K = \{630, 635, 640\}$, holding the remaining Black-Scholes parameters the same:

$$\begin{cases} C_{BS}(637.10, 630, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 97.60 \\ C_{BS}(637.10, 635, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 94.53 \\ C_{BS}(637.10, 640, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 91.75 \end{cases} \Rightarrow \begin{cases} r = 0.0641 \\ \sigma = 0.1398 \\ D = 0.0130 \end{cases} \tag{5.5}$$

The target values for σ, r and D here were:

$$r^* = 3.981\% \pm 0.196\%, \quad \sigma^* = 19.46\% \pm 0.12\% \quad \text{and} \quad D^* = 1.337\% \pm 0.149\%. \tag{5.6}$$

Again we use equation (5.2) to find the solution of σ, r and D in equation (5.5). The Tikhonov weight parameters, $\{\lambda_r, \lambda_\sigma, \lambda_D\}$, here were $\{1, 0, 100\}$ respectively. For options with these strike prices, the Black-Scholes price deviates furthest from the empirical option prices (see figure 5.1), therefore the reference values, $\{r_{\text{ref}}, \sigma_{\text{ref}}, D_{\text{ref}}\}$ were made to be equal to the target values, $\{0.03984, 0.1946, 0.01337\}$ to increase accuracy. The error from the target values here are much larger; if we choose the reference values with more inaccuracy about the target values, the solution wouldn't have any values within the given target error.

The reduction in accuracy at this strike level is caused by the joint minimisation of the total error combining the least squares part and the Tikhonov penalty term in equation (5.2). If σ, r and D were equal to their target values, the least squares part would dominate and the total error would be larger. The overall error is minimised when the solution is picked to be σ, r and D, shown by the solution (5.6).

Finally we solve for options far out of the money, where $C = \{1.67, 1.58, 1.49\}$ for the respective

strikes: $K = \{985, 990, 995\}$, holding the remaining Black-Scholes parameters the same:

$$\begin{cases} C_{BS}(637.10, 985, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 1.67 \\ C_{BS}(637.10, 990, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 1.58 \\ C_{BS}(637.10, 995, t = 0, T = 2.389, r^*, \sigma^*, D^*) \approx 1.49 \end{cases} \Rightarrow \begin{cases} r = 0.0389 \\ \sigma = 0.1270 \\ D = 0.0113 \end{cases} \tag{5.7}$$

The target values for σ, r and D here were:

$$r^* = 3.981\% \pm 0.196\%, \quad \sigma^* = 12.64\% \pm 0.04\% \quad \text{and} \quad D^* = 1.337\% \pm 0.149\%. \tag{5.8}$$

The solution here is close the corresponding target values, because the option is far out of the money and therefore behaves similarly to the Black-Scholes equation. For low value options, changes in the strike price and volatility cause small changes in the option price (look at figure 5.1), therefore the solution is stable and requires less regularisation to get accurate results. We chose $\{\lambda_r, \lambda_\sigma, \lambda_D\}$ to be $\{10, 0, 10\}$, and $\{r_{\text{ref}}, \sigma_{\text{ref}}, D_{\text{ref}}\}$ to be $\{0.04, 0.25, 0.01\}$ respectively.

The method we proposed in this chapter used the call option premiums from real-market data, alongside the known parameters: strike price (K), current stock price (S), and time to maturity $(T-t)$ to recover the unknown parameters: σ, r and D. The degree of accuracy was assessed for the three solutions, where the options were: far in the money, near the money, and far out of the money. While the standard deviation of the target values were very small and the market noise generated by the empirical option prices were large comparatively, it was expected that for many cases the solutions would lie outside of their corresponding error range.

The target values for σ were different across the three cases investigated, they were constant for r and D. However, our method did not directly seek constants for r and D, instead it reverse-engineered the variables of the diffusion process, equation (2.3), based on the option's value and stock price. This is why the accuracy was lower for options near the money. The empirical prices were much higher than the Black-Scholes price, to account for this the interest rate was over-valued and the dividend yield was undervalued (refer to figures 3.4 and 3.6 for illustration). To improve the accuracy for options with prices that deviate far from the Black-Scholes price, we may adapt the method to find r and D as constants by solving two equations for all strikes, and use the constant estimates to solve for the one unknown σ. In this way we reduce the complexity of the system and calibrate our solution to the entire range of strikes.

Our method does not rely on independently sourced parameters for the volatility, interest rate, and dividend yield, avoiding inconsistencies and arbitrage opportunities when used for pricing. Instead it jointly calibrates a, unified set of solutions to the observed option prices. This ensures internal consistency, filters out noise from illiquid data, and shows promise in the ability to recover missing data.

Chapter 6

Conclusions and Future Work

We addressed the problems of a direct approach to option pricing, the inverse method jointly recovered constant estimates for volatility, risk-free rate, and the dividend yield from market option prices. Robustness through numerical methods ensured valid solutions on the noisy surface. However, the method's accuracy was highly sensitive to moneyness and time to maturity (seen in chapter's 5 and 3).

We explored specific time-dependent volatility cases for analytical solutions in the Black-Scholes PDE in chapter 2, and expanded to more general adaptations of the Black-Scholes PDE in chapter 4. Dupire's equation offered a flexible approach to capturing volatility smiles and skews. However, limitations with data sparsity and numerical instability underscore the need for advanced smoothing and calibration techniques.

In conclusion, inverse methods offer a powerful tool for parameter inference, they also reveal the limitations of the Black-Scholes model. Future work into non-deterministic models capture different market dynamics. We briefly review these methods for option valuation.

6.1 Stochastic Volatility Models

The Heston model assumes a different underlying spot price process, where the volatility itself follows a stochastic process. The process is similar to the diffusion process under the Black-Scholes framework, below we show the diffusion process for the stochastic volatility model [12]:

$$dS(t) = \mu S dt + \sqrt{v(t)} S dW \tag{6.1}$$

The variance here is given by $v(t)$ and follows the square-root process [12]:

$$dv(t) = k[\theta - v(t)]dt + \sigma\sqrt{v(t)}dW \tag{6.2}$$

The mean of the variance is θ, the speed in which the variance, $v(t)$ reverts to its mean is k, and σ is the volatility of the volatility [12]. This model reveals empirical trends between stock price movements and changes in volatility, but struggles to fit the implied volatility surface of short-term options [7].

6.2 Jump-Diffusion Models

To rectify this problem Merton formulated a model that allows jumps in the stock price, specifically to replicate the arrival of new information in the market [13]. The process is Poisson-driven as shown below [13]:

$$\begin{aligned} \frac{dS}{S} &= (\alpha - \lambda k)dt + \sigma dW, \quad \text{if the Poisson event does not occur,} \\ &= (\alpha - \lambda k)dt + \sigma dW + (Y-1), \quad \text{if the Poisson event occurs.} \end{aligned} \tag{6.3}$$

The instantaneous expected return is given by α, the instantaneous variance is σ^2, the parameter defining the mean number of arrivals on new information per unit time is given by λ, and k is the expectation of the variable change in the stock price, $Y-1$ ($k = \mathbb{E}[Y-1]$) [13].

Appendix A

Code

Figure 2.3: Call Option Premium as a Function of Stock Price

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy.optimize import fsolve
from scipy.stats import mode

def black_scholes_call(S, K, T, t, r, sigma, D):
    tau = T - t

    if tau <= 0:
        return max(S - K, 0)

    d1 = (np.log(S/K) + (r - D + 0.5*sigma**2)*tau) / (sigma*np.sqrt(tau))
    d2 = d1 - sigma*np.sqrt(tau)

    return S*np.exp(-D*tau)*norm.cdf(d1) - K*np.exp(-r*tau)*norm.cdf(d2)

K0 = 80
K1 = 100
K2 = 120
r = 0.05
sigma = 0.2
D = 0.02
T = 1.0
t = 0.0

S_values = np.linspace(50, 150, 100)
call_prices0 = [black_scholes_call(S, K0, T, t, r, sigma, D) for S in
    S_values]
call_prices1 = [black_scholes_call(S, K1, T, t, r, sigma, D) for S in
    S_values]
call_prices2 = [black_scholes_call(S, K2, T, t, r, sigma, D) for S in
    S_values]

plt.figure(figsize=(10, 6))
plt.plot(S_values, call_prices0, ls = '-', color = 'k', label = 'K = 80')
plt.plot(S_values, call_prices1, ls = '--', color = 'k', label = 'K = 100')
plt.plot(S_values, call_prices2, ls = '-.', color = 'k', label = 'K = 120')
plt.xlabel('S', fontsize=14)
plt.xticks([50, 60, 70, 80, 90, 100, 110, 120, 130, 140])
plt.ylabel('C(S, 0)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

Figure 2.4: Call Option Premium as a Function of Time

```
# Note: Throughout the rest of the code we will use the function implemented
# above called black_scholes_call to compute call the plots and to solve
# the system of equations for the inverse method.

t_values = np.linspace(0, T, 100)
S_fixed = 110

call_prices_time0 = [black_scholes_call(S_fixed, K0, T, tvals, r, sigma, D)
    for tvals in t_values]
call_prices_time1 = [black_scholes_call(S_fixed, K1, T, tvals, r, sigma, D)
    for tvals in t_values]
call_prices_time2 = [black_scholes_call(S_fixed, K2, T, tvals, r, sigma, D)
    for tvals in t_values]

plt.figure(figsize=(10, 6))
plt.plot(t_values, call_prices_time0, ls = '-', color = 'k', label = 'K = 80
    ')
plt.plot(t_values, call_prices_time1, ls = '--', color = 'k', label = 'K =
    100')
plt.plot(t_values, call_prices_time2, ls = '-.', color = 'k', label = 'K =
    120')
plt.xlabel('t', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

[language=Python]

Figure 2.5: Call Option Premium as a Function of Strike Price

```
K_values = np.linspace(50, 150, 100)
call_prices_K0 = [black_scholes_call(90, K, T, t, r, sigma, D) for K in
    K_values]
call_prices_K1 = [black_scholes_call(S_fixed, K, T, t, r, sigma, D) for K in
     K_values]
call_prices_K2 = [black_scholes_call(130, K, T, t, r, sigma, D) for K in
    K_values]

plt.figure(figsize=(10, 6))
plt.plot(K_values, call_prices_K0, ls = '-', color = 'k', label = 'S = 90')
plt.plot(K_values, call_prices_K1, ls = '--', color = 'k', label = 'S = 110'
    )
plt.plot(K_values, call_prices_K2, ls = '-.', color = 'k', label = 'S = 130'
    )
plt.xlabel('K', fontsize=14)
plt.ylabel('C(S, 0)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

[language=Python]

Figure 2.6: Surface Plot for the Call Option Premium, $C(S,t)$

```
S_grid, t_grid = np.meshgrid(S_values, 1 - t_values)

BS_price = np.zeros((len(t_values), len(S_values)))
for i in range(len(S_values)):
    for j in range(len(t_values)):
        BS_price[j, i] = black_scholes_call(S_values[i], K1, T, t_values[j],
    r, sigma, D)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
surface = ax.plot_surface(S_grid, t_grid, BS_price, cmap='gray', edgecolor='
    none')

ax.set_xlabel('S')
ax.set_ylabel('T - t')
ax.set_zlabel('C(S, t)')

cbar = fig.colorbar(surface, ax=ax, shrink=0.5)
cbar.set_label('C(S, t)')
plt.show()
```

[language=Python]

Figures 3.1 and 3.2: Call Option Premium as a Function of σ

```
# Varying sigma and plotting for 3 different stock prices
sigma_values = np.linspace(0.1, 0.9, 100)
call_prices_sigma0 = [black_scholes_call(90, K1, T, t, r, sigma_val, D) for
    sigma_val in sigma_values]
call_prices_sigma1 = [black_scholes_call(S_fixed, K1, T, t, r, sigma_val, D)
    for sigma_val in sigma_values]
call_prices_sigma2 = [black_scholes_call(130, K1, T, t, r, sigma_val, D) for
    sigma_val in sigma_values]

plt.figure(figsize=(10, 6))
plt.plot(sigma_values, call_prices_sigma0, ls = '-', color = 'k', label = 'S
    = 90')
plt.plot(sigma_values, call_prices_sigma1, ls = '--', color = 'k', label = '
    S = 110')
plt.plot(sigma_values, call_prices_sigma2, ls = '-.', color = 'k', label = '
    S = 130')
plt.xlabel('$\sigma$', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()

# Now varying sigma and plotting for 3 different strike prices
call_prices_sigma3 = [black_scholes_call(S_fixed, K0, T, t, r, sigma_val, D)
    for sigma_val in sigma_values]
call_prices_sigma4 = [black_scholes_call(S_fixed, K1, T, t, r, sigma_val, D)
    for sigma_val in sigma_values]
call_prices_sigma5 = [black_scholes_call(S_fixed, K2, T, t, r, sigma_val, D)
    for sigma_val in sigma_values]

plt.figure(figsize=(10, 6))
plt.plot(sigma_values, call_prices_sigma3, ls = '-', color = 'k', label = 'K
    = 80')
```

```
plt.plot(sigma_values, call_prices_sigma4, ls = '--', color = 'k', label = '
    K = 100')
plt.plot(sigma_values, call_prices_sigma5, ls = '-.', color = 'k', label = '
    K = 120')
plt.xlabel('$\sigma$', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

[language=Python]

Bounds for the Option Premium as a Function of σ

```
Bounds_sigma_S0 = np.array([min(call_prices_sigma0), max(call_prices_sigma0)
    ])
Bounds_sigma_S1 = np.array([min(call_prices_sigma1), max(call_prices_sigma1)
    ])
Bounds_sigma_S2 = np.array([min(call_prices_sigma2), max(call_prices_sigma2)
    ])

Bounds_sigma_K0 = np.array([min(call_prices_sigma3), max(call_prices_sigma3)
    ])
Bounds_sigma_K1 = np.array([min(call_prices_sigma4), max(call_prices_sigma4)
    ])
Bounds_sigma_K2 = np.array([min(call_prices_sigma5), max(call_prices_sigma5)
    ])

print(f"Bounds for S = 90: {Bounds_sigma_S0}")
print(f"Bounds for S = 110: {Bounds_sigma_S1}")
print(f"Bounds for S = 130: {Bounds_sigma_S2}")

print(f"Bounds for K = 80: {Bounds_sigma_K0}")
print(f"Bounds for K = 100: {Bounds_sigma_K1}")
print(f"Bounds for K = 120: {Bounds_sigma_K2}")
```

Figures 3.3 and 3.4: Call Option Premium as a Function of r

```
# Varying r and plotting for 3 different stock prices
r_values = np.linspace(0, 1, 100)
call_prices_r0 = [black_scholes_call(90, K1, T, t, r_val, sigma, D) for
    r_val in r_values]
call_prices_r1 = [black_scholes_call(S_fixed, K1, T, t, r_val, sigma, D) for
     r_val in r_values]
call_prices_r2 = [black_scholes_call(130, K1, T, t, r_val, sigma, D) for
    r_val in r_values]

plt.figure(figsize=(10, 6))
plt.plot(r_values, call_prices_r0, ls = '-', color = 'k', label = 'S = 90')
plt.plot(r_values, call_prices_r1, ls = '--', color = 'k', label = 'S = 110'
    )
plt.plot(r_values, call_prices_r2, ls = '-.', color = 'k', label = 'S = 130'
    )
plt.xlabel('r', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
```

```
plt.legend()
plt.show()

# Varying r and plotting for 3 different strike prices
call_prices_r3 = [black_scholes_call(S_fixed, K0, T, t, r_val, sigma, D) for r_val in r_values]
call_prices_r4 = [black_scholes_call(S_fixed, K1, T, t, r_val, sigma, D) for r_val in r_values]
call_prices_r5 = [black_scholes_call(S_fixed, K2, T, t, r_val, sigma, D) for r_val in r_values]

plt.figure(figsize=(10, 6))
plt.plot(r_values, call_prices_r3, ls = '-', color = 'k', label = 'K = 80')
plt.plot(r_values, call_prices_r4, ls = '--', color = 'k', label = 'K = 100')
plt.plot(r_values, call_prices_r5, ls = '-.', color = 'k', label = 'K = 120')
plt.xlabel('r', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

[language=Python]

Bounds for the Option Premium as a Function of r

```
Bounds_r_S0 = np.array([min(call_prices_r0), max(call_prices_r0)])
Bounds_r_S1 = np.array([min(call_prices_r1), max(call_prices_r1)])
Bounds_r_S2 = np.array([min(call_prices_r2), max(call_prices_r2)])

Bounds_r_K0 = np.array([min(call_prices_r3), max(call_prices_r3)])
Bounds_r_K1 = np.array([min(call_prices_r4), max(call_prices_r4)])
Bounds_r_K2 = np.array([min(call_prices_r5), max(call_prices_r5)])

print(f"Bounds for S = 90: {Bounds_r_S0}")
print(f"Bounds for S = 110: {Bounds_r_S1}")
print(f"Bounds for S = 130: {Bounds_r_S2}")

print(f"Bounds for K = 80: {Bounds_r_K0}")
print(f"Bounds for K = 100: {Bounds_r_K1}")
print(f"Bounds for K = 120: {Bounds_r_K2}")
```

[language=Python]

Figures 3.5 and 3.6: Call Option Premium as a Function of D

```
# Varying D and plotting for 3 different stock prices
D_values = np.linspace(0, 0.8, 100)
call_prices_D0 = [black_scholes_call(90, K1, T, t, r, sigma, D_val) for D_val in D_values]
call_prices_D1 = [black_scholes_call(S_fixed, K1, T, t, r, sigma, D_val) for D_val in D_values]
call_prices_D2 = [black_scholes_call(130, K1, T, t, r, sigma, D_val) for D_val in D_values]

plt.figure(figsize=(10, 6))
```

```
plt.plot(D_values, call_prices_D0, ls = '-', color = 'k', label = 'S = 90')
plt.plot(D_values, call_prices_D1, ls = '--', color = 'k', label = 'S = 110')
plt.plot(D_values, call_prices_D2, ls = '-.', color = 'k', label = 'S = 130')
plt.xlabel('D', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()

# Varying D and plotting for 3 different strike prices
call_prices_D3 = [black_scholes_call(S_fixed, K0, T, t, r, sigma, D_val) for D_val in D_values]
call_prices_D4 = [black_scholes_call(S_fixed, K1, T, t, r, sigma, D_val) for D_val in D_values]
call_prices_D5 = [black_scholes_call(S_fixed, K2, T, t, r, sigma, D_val) for D_val in D_values]

plt.figure(figsize=(10, 6))
plt.plot(D_values, call_prices_D3, ls = '-', color = 'k', label = 'K = 80')
plt.plot(D_values, call_prices_D4, ls = '--', color = 'k', label = 'K = 100')
plt.plot(D_values, call_prices_D5, ls = '-.', color = 'k', label = 'K = 120')
plt.xlabel('D', fontsize=14)
plt.ylabel('C(S, t)', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()
```

[language=Python]

Bounds for the Option Premium as a Function of D

```
Bounds_D_S0 = np.array([max(call_prices_D0), min(call_prices_D0)])
Bounds_D_S1 = np.array([max(call_prices_D1), min(call_prices_D1)])
Bounds_D_S2 = np.array([max(call_prices_D2), min(call_prices_D2)])

Bounds_D_K0 = np.array([max(call_prices_D3), min(call_prices_D3)])
Bounds_D_K1 = np.array([max(call_prices_D4), min(call_prices_D4)])
Bounds_D_K2 = np.array([max(call_prices_D5), min(call_prices_D5)])

print(f"Bounds for S = 90: {Bounds_D_S0}")
print(f"Bounds for S = 110: {Bounds_D_S1}")
print(f"Bounds for S = 130: {Bounds_D_S2}")

print(f"Bounds for K = 80: {Bounds_D_K0}")
print(f"Bounds for K = 100: {Bounds_D_K1}")
print(f"Bounds for K = 120: {Bounds_D_K2}")
```

[language=Python]

Figures 3.7: Surface Plot for the Call Option Premium, $C(r, \sigma)$

```
# Surface plot of c(sigma, r)
r_grid, sigma_grid = np.meshgrid(r_values, sigma_values)
```

```
C_sigma_r1 = np.zeros((len(sigma_values), len(r_values)))
C_sigma_r2 = np.zeros((len(sigma_values), len(r_values)))
for i in range(len(sigma_values)):
    for j in range(len(r_values)):
        C_sigma_r1[i, j] = black_scholes_call(S_fixed, K0, T, t, r_values[j], sigma_values[i], D)
        C_sigma_r2[i, j] = black_scholes_call(S_fixed, K2, T, t, r_values[j], sigma_values[i], D)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
surface1 = ax.plot_surface(r_grid, sigma_grid, C_sigma_r1, cmap='grey', edgecolor='none')
surface2 = ax.plot_surface(r_grid, sigma_grid, C_sigma_r2, cmap='grey', edgecolor='none')

ax.set_xlabel('r')
ax.set_ylabel('$\sigma$')
ax.set_zlabel('C(r, $\sigma$)')

fig.colorbar(surface1, ax=ax, shrink=0.5, label='C(r, $\sigma$)')
plt.show()
```

[language=Python]

Figures 3.8: Surface Plot for the Call Option Premium, $C(D, \sigma)$

```
# Surface plot of c(D, sigma)
D_grid, sigma_grid = np.meshgrid(D_values, sigma_values)
C_sigma_D1 = np.zeros((len(sigma_values), len(D_values)))
C_sigma_D2 = np.zeros((len(sigma_values), len(D_values)))
for i in range(len(sigma_values)):
    for j in range(len(D_values)):
        C_sigma_D1[i, j] = black_scholes_call(S_fixed, K0, T, t, r, sigma_values[i], D_values[j])
        C_sigma_D2[i, j] = black_scholes_call(S_fixed, K2, T, t, r, sigma_values[i], D_values[j])

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
surface1 = ax.plot_surface(D_grid, sigma_grid, C_sigma_D1, cmap='grey', edgecolor='none')
surface2 = ax.plot_surface(D_grid, sigma_grid, C_sigma_D2, cmap='grey', edgecolor='none')

ax.set_xlabel('D')
ax.set_ylabel('$\sigma$')
ax.set_zlabel('C(D, $\sigma$)')

fig.colorbar(surface1, ax=ax, shrink=0.5, label='C(D, $\sigma$)')
plt.show()
```

[language=Python]

Figures 3.9: Surface Plot for the Call Option Premium, $C(r, D)$

```
# Surface plot of c(r, D)
r_grid, D_grid = np.meshgrid(r_values, D_values)
C_r_D1 = np.zeros((len(D_values), len(r_values)))
C_r_D2 = np.zeros((len(D_values), len(r_values)))

for j in range(len(D_values)):
    for i in range(len(r_values)):
        C_r_D1[j, i] = black_scholes_call(S_fixed, K0, T, t, r_values[i],
    sigma, D_values[j])
        C_r_D2[j, i] = black_scholes_call(S_fixed, K2, T, t, r_values[i],
    sigma, D_values[j])

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
surface1 = ax.plot_surface(r_grid, D_grid, C_r_D1, cmap='grey', edgecolor='
    none')
surface2 = ax.plot_surface(r_grid, D_grid, C_r_D2, cmap='grey', edgecolor='
    none')

ax.set_xlabel('r')
ax.set_ylabel('D')
ax.set_zlabel('C(r, D)')

fig.colorbar(surface1, ax=ax, shrink=0.5, label='C(r, D)')
plt.show()
```

[language=Python]

Solving the System of Three Equations With Three Unknowns

```

######################## Varying S ###################
def equations(vars, S, strike_price, maturity, init_t, C):
    r_init, sigma_init, D_init = vars
    eq1 = black_scholes_call(S[0], strike_price[0], maturity, init_t, r_init
    , sigma_init, D_init) - C[0]
    eq2 = black_scholes_call(S[1], strike_price[1], maturity, init_t, r_init
    , sigma_init, D_init) - C[1]
    eq3 = black_scholes_call(S[2], strike_price[2], maturity, init_t, r_init
    , sigma_init, D_init) - C[2]
    return [eq1, eq2, eq3]

S_varied = np.array([90, S_fixed, 130])
strike_price_fixed = np.array([K1, K1, K1])
maturity = 1
init_t = 0
C0 = black_scholes_call(S_varied[0], strike_price_fixed[0], maturity, init_t
    , 0.05, 0.2, 0.02)
C1 = black_scholes_call(S_varied[1], strike_price_fixed[1], maturity, init_t
    , 0.05, 0.2, 0.02)
C2 = black_scholes_call(S_varied[2], strike_price_fixed[2], maturity, init_t
    , 0.05, 0.2, 0.02)

initial_guess = [0.5, 0.5, 0.5]
solution0 = fsolve(equations, initial_guess, args=(S_varied,
    strike_price_fixed, maturity, init_t, [C0, C1, C2]))

```

```
print(f"The value of r that solves the system of equations is: {solution0[0]:.15f}")
print(f"The value of $\sigma$ that solves the system of equations is: {solution0[1]:.15f}")
print(f"The value of D that solves the system of equations is: {solution0[2]:.15f}")

################## Rounding to 1 DP ##################
solution1 = fsolve(equations, initial_guess, args=(S_varied, strike_price_fixed, maturity, init_t, [round(C0, 1), round(C1, 1), round(C2, 1)]))

print(f"The value of r that solves the system of equations is: {solution1[0]:.4f}")
print(f"The value of $\sigma$ that solves the system of equations is: {solution1[1]:.4f}")
print(f"The value of D that solves the system of equations is: {solution1[2]:.4f}")

####################### Varying K ###################
K_varied = np.array([K0, K1, K2])
Stock_price_fixed = np.array([S_fixed, S_fixed, S_fixed])
C3 = black_scholes_call(Stock_price_fixed[0], K_varied[0], maturity, init_t, 0.05, 0.2, 0.02)
C4 = black_scholes_call(Stock_price_fixed[1], K_varied[1], maturity, init_t, 0.05, 0.2, 0.02)
C5 = black_scholes_call(Stock_price_fixed[2], K_varied[2], maturity, init_t, 0.05, 0.2, 0.02)

solution2 = fsolve(equations, initial_guess, args=(Stock_price_fixed, K_varied, maturity, init_t, [round(C3, 1), round(C4, 1), round(C5, 1)]))

print(f"The value of r that solves the system of equations is: {solution2[0]:.4f}")
print(f"The value of $\sigma$ that solves the system of equations is: {solution2[1]:.4f}")
print(f"The value of D that solves the system of equations is: {solution2[2]:.4f}")
```

[language=Python]

Solving difficult Systems by improving the initial guess method

```
def equations_solver(Stockprices, Strikeprices, Maturitytime, Initialtime, interestrate, volatility, dividend_yield):

    Calloptprem1 = black_scholes_call(Stockprices[0], Strikeprices[0], Maturitytime, Initialtime, interestrate, volatility, dividend_yield)
    Calloptprem2 = black_scholes_call(Stockprices[1], Strikeprices[1], Maturitytime, Initialtime, interestrate, volatility, dividend_yield)
    Calloptprem3 = black_scholes_call(Stockprices[2], Strikeprices[2], Maturitytime, Initialtime, interestrate, volatility, dividend_yield)

    solutions = []
    guesses = []
    errors = []
    for i in range(0, 800):
        r_star = np.random.lognormal(-1, 3)
        sigma_star = np.random.lognormal(-1, 3)
```

```
        D_star = np.random.lognormal(-1, 3)

        initialGuess = [r_star, sigma_star, D_star]
        solution = fsolve(equations, initialGuess, args=(Stockprices,
    Strikeprices, Maturitytime, Initialtime, [round(Calloptprem1, 1), round(
    Calloptprem2, 1), round(Calloptprem3, 1)]))

        error = np.array(solution) - np.array(initialGuess)
        if np.sum(error) != 0: # Case where fsolve() takes the guess as
    solution is invalid
            if np.abs(error[0]) < 0.05:
                if np.abs(error[1]) < 0.05:
                    if np.abs(error[2]) < 0.05:
                        solutions.append(solution)
                        guesses.append(initialGuess)
                        errors.append(error)
    return np.array(solutions), np.array(guesses), np.array(errors)

solver1, guess1, errors1 = equations_solver(S_varied, strike_price_fixed,
    10, 1, 0.05, 0.2, 0.02)

def optimal_solution(solver, guess, errors):

    rounded_arr = np.round(solver, 8)
    mode_row = mode(rounded_arr, axis=0, keepdims=False).mode

    matching_indices = np.where((np.round(solver, 8) == mode_row).all(axis
    =1))[0]
    abs_errors = np.sum(errors**2, axis=1)

    err_of_solns = []
    for i in matching_indices:
        err_of_soln = abs_errors[i]
        err_of_solns.append(err_of_soln)

    loc_of_min_err_soln = np.where(abs_errors == np.min(err_of_solns))[0][0]
    return mode_row, guess[loc_of_min_err_soln]

opt_soln1, opt_init_guess1 =  optimal_solution(solver1, guess1, errors1)
err_of_prediction1 = np.linalg.norm(np.array([0.05, 0.2, 0.02]) - opt_soln1)
print(f"The optimal solution is: {opt_soln1}")
print(f"The optimal initial guess is: {opt_init_guess1}")
print(f"The total error is: {err_of_prediction1}")

solver2, guess2, errors2 = equations_solver(np.array([S_fixed, S_fixed,
    S_fixed]), K_varied, 10, 1, 0.05, 0.2, 0.02)
opt_soln2, opt_init_guess2 = optimal_solution(solver2, guess2, errors2)
err_of_prediction2 = np.linalg.norm(np.array([0.05, 0.2, 0.02]) - opt_soln2)
print(f"The optimal solution is: {opt_soln2}")
print(f"The optimal initial guess is: {opt_init_guess2}")
print(f"The total error is: {err_of_prediction2}")
```

[language=Python]

Figures 4.1, 4.2 and 4.3: Plotting volatility smiles and skews

```
import csv
data2025 = []
with open('spy_options_exp_2025_08_04.csv', newline='') as options2025:
```

```
    reader2025 = csv.reader(options2025, delimiter=',')
    for i in range(1):
        next(reader2025)
    for row in reader2025:
        strike = row[0]
        IV = row[11]
        CallPut = row[13]
        bid = row[2]
        mid = row[3]
        ask = row[4]
        data2025.append((strike, IV, CallPut, bid, mid, ask))

data2025 = np.array(data2025)
strikes2025 = data2025[:, 0].astype(int)
iv_values2025 = np.char.rstrip(data2025[:, 1], '%').astype(float) / 100
for i in data2025[:,2]:
    if i == 'Call':
        index_call2025 = np.where(data2025[:,2] == i)[0]
    if i == 'Put':
        index_put2025 = np.where(data2025[:,2] == i)[0]

strikes_call2025 = strikes2025[index_call2025]
iv_values_call2025 = iv_values2025[index_call2025]

strikes_put2025 = strikes2025[index_put2025]
iv_values_put2025 = iv_values2025[index_put2025]

BS_strikes = np.linspace(400, 720, 100)
call_prices_sigma0 = [black_scholes_call(637.1, K1, T, t, r, sigma_val, D)
   for sigma_val in sigma_values]

plt.figure(figsize=(10, 6))
plt.plot(strikes_call2025, iv_values_call2025, '--.', color = 'k', label='
   Calls')
plt.plot(strikes_put2025, iv_values_put2025, ':', color = 'k', label='Puts')
plt.xlabel('Strike Price, K', fontsize=14)
plt.ylabel('Implied Volatility, $\sigma$', fontsize=14)
plt.grid(True)
plt.legend()
plt.show()

from matplotlib.ticker import MaxNLocator
data2027 = []
with open('spy_options_exp_2027_12_17.csv', newline='') as options2027:
    reader2027 = csv.reader(options2027, delimiter=',')
    for i in range(1):
        next(reader2027)
    for row in reader2027:
        strike = row[0]
        IV = row[11]
        CallPut = row[13]
        bid = row[2]
        mid = row[3]
        ask = row[4]
        data2027.append((strike, IV, CallPut, bid, mid, ask))

data2027 = np.array(data2027)
strikes2027 = data2027[:, 0]
strikes2027.astype(int)
iv_values2027 = np.char.rstrip(data2027[:, 1], '%').astype(float) / 100
```

```
for i in data2027[:,2]:
    if i == 'Call':
        index_call2027 = np.where(data2027[:,2] == i)[0]
    if i == 'Put':
        index_put2027 = np.where(data2027[:,2] == i)[0]

strikes_call2027 = strikes2027[index_call2027]
iv_values_call2027 = iv_values2027[index_call2027]

strikes_put = strikes2027[index_put2027]
iv_values_put = iv_values2027[index_put2027]

plt.figure(figsize=(10, 6))
plt.plot(strikes_call2027, iv_values_call2027, '--.', color = 'k', label='
    Calls')
plt.plot(strikes_put, iv_values_put, ':', color = 'k', label='Puts')
plt.xlabel('Strike Price, K', fontsize=14)
plt.ylabel('Implied Volatility, $\sigma$', fontsize=14)
plt.grid(True)
ax = plt.gca()
ax.xaxis.set_major_locator(MaxNLocator(nbins=10))
plt.legend()
plt.show()

import pandas as pd
data = []
with open('spy_data_635.csv', newline='') as options:
    reader = csv.reader(options, delimiter=',')
    for i in range(1):
        next(reader)
    for row in reader:
        date = row[0]
        IV = row[11]
        CallPut = row[13]
        data.append((date, IV, CallPut))

data = np.array(data)
dates = data[:, 0]
cleaned_dates = [date[:-4] for date in dates]
date_ts = pd.to_datetime(cleaned_dates)
iv_values = np.char.rstrip(data[:, 1], '%').astype(float) / 100
for i in data[:,2]:
    if i == 'Call':
        index_call = np.where(data[:,2] == i)[0]
    if i == 'Put':
        index_put = np.where(data[:,2] == i)[0]

dates_call = date_ts[index_call]
iv_values_call = iv_values[index_call]

dates_put = date_ts[index_put]
iv_values_put = iv_values[index_put]

plt.figure(figsize=(10, 6))
plt.plot(dates_call, iv_values_call, '--.', color = 'k', label='Calls')
plt.plot(dates_put, iv_values_put, '-', color = 'k', label='Puts')
plt.xlabel('Maturity, T', fontsize=14)
plt.ylabel('Implied Volatility, $\sigma$', fontsize=14)
plt.grid(True)
ax = plt.gca()
```

```
ax.xaxis.set_major_locator(MaxNLocator(nbins=5))
plt.legend()
plt.show()
```

[language=Python]

Figure 5.1: Plotting Call Option Prices against Implied Volatility for Different Strikes

```
from matplotlib.colors import Normalize

BS_implvols = np.linspace(0.5, 0.12, 160)
strike_range = np.linspace(200, 995, 160)

bids2027 = data2027[:,3].astype(float)
mids2027 = data2027[:,4].astype(float)
asks2027 = data2027[:,5].astype(float)

mids2027Call = mids2027[index_call2027]
mids2027Put = mids2027[index_put2027]

calibrated_BS_prices = [black_scholes_call(637.1, strikes, 2.389, 0,
    0.03981, iv, 0.01337) for strikes, iv in zip(strike_range, BS_implvols)]

plt.figure(figsize=(10, 6))
sc1 = plt.scatter(BS_implvols, calibrated_BS_prices, c=strike_range, cmap='
    gray', norm=Normalize(vmin=min(strike_range), vmax=max(strike_range)),
    marker='.', label='Black-Scholes Prices', s=30)
sc2 = plt.scatter(iv_values_call2027, mids2027Call, c=strike_range, cmap='
    gray', norm=Normalize(vmin=min(strike_range), vmax=max(strike_range)),
    marker='x', label='Market Data', s=50, linewidths=1)

cbar = plt.colorbar(sc2)
cbar.set_label('Strike Price (K)', rotation=270, labelpad=20)

plt.xlabel('Implied Volatility', fontsize=12)
plt.ylabel('Call Option Price', fontsize=12)
plt.grid(True, alpha=0.3)
plt.legend()

plt.tight_layout()
plt.show()
```

[language=Python]

The Inverse Method to Option Pricing for Real-Market Data

```
T_2027 = 2.389
C210_2027 = mids2027Call[2]
C205_2027 = mids2027Call[1]
C200_2027 = mids2027Call[0]

K210_2027 = float(strikes_call2027[2])
K205_2027 = float(strikes_call2027[1])
K200_2027 = float(strikes_call2027[0])

```

```
C640_2027 = mids2027Call[88]
C635_2027 = mids2027Call[87]
C630_2027 = mids2027Call[86]

K640_2027 = float(strikes_call2027[88])
K635_2027 = float(strikes_call2027[87])
K630_2027 = float(strikes_call2027[86])

C995_2027 = mids2027Call[159]
C990_2027 = mids2027Call[158]
C985_2027 = mids2027Call[157]

K995_2027 = float(strikes_call2027[159])
K990_2027 = float(strikes_call2027[158])
K985_2027 = float(strikes_call2027[157])

from scipy.optimize import least_squares
def equations_tikh(vars, S, strike_price, maturity, init_t, market_prices, lambda_r, lambda_sigma, lambda_D, r_ref, sigma_ref, D_ref):
    r_init, sigma_init, D_init = vars
    eq1 = black_scholes_call(S[0], strike_price[0], maturity, init_t, r_init, sigma_init, D_init) - market_prices[0]
    eq2 = black_scholes_call(S[1], strike_price[1], maturity, init_t, r_init, sigma_init, D_init) - market_prices[1]
    eq3 = black_scholes_call(S[2], strike_price[2], maturity, init_t, r_init, sigma_init, D_init) - market_prices[2]
    residuals = np.array([eq1, eq2, eq3])

    reg_r = lambda_r * (r_init - r_ref)**2
    reg_sigma = lambda_sigma * (sigma_init - sigma_ref)**2
    reg_D = lambda_D * (D_init - D_ref)**2

    return np.concatenate([residuals, [np.sqrt(reg_r), np.sqrt(reg_sigma), np.sqrt(reg_D)]])

result1 = least_squares(equations_tikh, x0=[0.1, 0.1, 0.1], bounds=([0, 0, 0], [1, 1, 1]), args=(np.array([637.1, 637.1, 637.1]), np.array([K200_2027, K205_2027, K210_2027]), T_2027, 0, np.array([C200_2027, C205_2027, C210_2027]), 100, 0.01, 0, 0.04, 0.25, 0.01))
lsq_err1 = np.sum(result1.fun[:3]**2)
rglr_err1 = np.sum(result1.fun[3:]**2)
print(f"The total least squares error is {lsq_err1}")
print(f"The total resularised error is {rglr_err1}")
print(f"The solution is {result1.x}")

result2 = least_squares(equations_tikh, x0=[0.1, 0.1, 0.1], bounds=([0, 0, 0], [1, 1, 1]), args=(np.array([637.1, 637.1, 637.1]), np.array([K630_2027, K635_2027, K640_2027]), T_2027, 0, np.array([C630_2027, C635_2027, C640_2027]), 1, 0, 100, 0.03981, 0.1946, 0.01337))
lsq_err2 = np.sum(result2.fun[:3]**2)
rglr_err2 = np.sum(result2.fun[3:]**2)
print(f"The total least squares error is {lsq_err2}")
print(f"The total resularised error is {rglr_err2}")
print(f"The solution is {result2.x}")

result3 = least_squares(equations_tikh, x0=[0.1, 0.1, 0.1], bounds=([0, 0, 0], [1, 1, 1]), args=(np.array([637.1, 637.1, 637.1]), np.array([K985_2027, K990_2027, K995_2027]), T_2027, 0, np.array([C985_2027, C990_2027, C995_2027]), 10, 0, 10, 0.04, 0.25, 0.01))
lsq_err3 = np.sum(result3.fun[:3]**2)
```

```
rglr_err3 = np.sum(result3.fun[3:]**2)
print(f"The total least squares error is {lsq_err3}")
print(f"The total resularised error is {rglr_err3}")
print(f"The solution is {result3.x}")
```

[language=Python]

Bibliography

[1] Black, F. and Scholes, M. The pricing of options and corporate liabilities. *The journal of political economy*, 81(3):637–654, 1973. Availbale from: 10.1086/260062.

[2] Derman, E. and Kani, I. Riding on a smile. *Risk*, 7(2):32–39, 1994. Available from: https://www.researchgate.net/profile/Emanuel-Derman/publication/239059413_Riding_on_a_Smile/links/558950e408ae6d4f27ea5ab4/Riding-on-a-Smile.pdf.

[3] Bouchouev, I. and Isakov, V. The inverse problem of option pricing. *Inverse Problems*, 13: L11–L17, 1997. Available from: 10.1142/9789812704924_0005.

[4] Itkin, A. Local volatility and dupire's equation. In *Fitting Local Volatility Analytic and Numerical Approaches in Black-Scholes and Local Variance Gamma Models*, pages 3–12. World Scientific Publishing, Harlow, England, 2020. Available from: https://ideas.repec.org/h/wsi/wschap/9789811212772_0001.html.

[5] Wilmott, P. Howison, S. and Dewynne, J. *Option Pricing: Mathematical Models and Computation*. Oxford Financial Press, Oxford, 1993. Available from: https://cms.dm.uba.ar/Members/maurette/ACF2022/Paul_Wilmott%2C_Jeff_Dewynne%2C_Sam_Howison-Option_Pricing__Mathematical_Models_and_Computation-Oxford_Financial_Press%281994%29.pdf.

[6] Dupire, B. Pricing with a smile. *Risk Magazine*, 7(1):18–20, 1994. Available from: http://spekulant.com.pl/article/Volatility%20Surface%20Modeling/dupire%20local%20vol.pdf.

[7] Gatheral, J. *The Volatility Surface: A Practitioner's Guide*. Hoboken, New Jersey: John Wiley and Sons, Ltd, 2012. Available from: 10.1002/9781119202073.

[8] Bouchouev, I. and Isakov, V. Uniqueness, stability and numerical methods for the inverse problem that arises in financial markets. *Inverse Problems*, 15(3), 1999. Available from: 10.1088/0266-5611/15/3/201.

[9] Haugh, M. *Financial Engineering: Continuous-Time Models*. Course notes - Columbia University, 2013. Available From: http://www.columbia.edu/~mh2078/ContinuousTimeFinance.html.

[10] Fengler, M. Arbitrage-free smoothing of the implied volatility surface. *Quantitative Finance*, 9 (4):417–428, 2009. Available from: 10.1080/14697680802595568.

[11] Rached, N.B. *Continuous Time Finance* - MATH5330M. 2025.

[12] Heston, S.L. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The Review of Financial Studies*, 6(2):327–343, 1993. Available from: 10.1093/rfs/6.2.327.

[13] Merton, R.C. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 3(1):125–144, 1976. Available from: https://doi.org/10.1016/0304-405X(76)90022-2.

[14] Hull, J.C. *Options, Futures, and Other Derivatives*. Pearson, Harlow, England, eleventh, global edition, 2022.

[15] Lê, K. *Continuous Time Finance* - MATH5330M. 2025.

[16] Haugh, M. *Foundations of Financial Engineering* - IEOR E4706, 2016. Section 7: The Black-Scholes Model, Available from: https://martin-haugh.github.io/files/FoundationsFE/BlackScholes.pdf.

[17] Uddin, M.M. *Financial Derivatives* - LUBS5050M. Lecture 3 Notes, 2025.

[18] Remani, C. Numerical methods for solving systems of nonlinear equations, April 2013. Available from: https://www.lakeheadu.ca/sites/default/files/uploads/77/docs/RemaniFinal.pdf.

[19] Anon. S&P 500 SPDR (SPY), 2025. URL https://www.barchart.com.

[20] Anon. Daily treasury par yield curve rates, 2025. URL https://home.treasury.gov/resource-center/data-chart-center/interest-rates/TextView?type=daily_treasury_yield_curve&field_tdr_date_value=2025.

[21] Anon. S&P 500 ETF SPDR (SPY) historical prices, 2025. URL https://finance.yahoo.com/quote/SPY/history/?filter=history.

[22] Anon. S&P 500 dividend yield by year, 2025. URL https://www.multpl.com/s-p-500-dividend-yield/table/by-year.

[23] Connor, S. Mirjeta, P., Silvia, G. and Ugochukwu, U. TRIPs-py: Techniques for regularization of inverse problems in python. *Numerical Algorithms*, 99(1):285–322, 2025. Available from: 10.21203/rs.3.rs-4001742/v1.