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

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

You are not required to write your name on your work

Candidate number (this is found on your student record account)	AF25002
Module Code and Title	7CCMFM50 Financial Mathematics Project
Title of Project/Coursework	Pricing and Hedging Volaitility Index Options

Declaration

By submitting this assignment I agree to the following statements:
I have read and understand the King's College London Academic Honesty and Integrity Statement that I signed upon entry to this programme of study.
I declare that the content of this submission is my own work.
I understand that plagiarism is a serious examination offence, an allegation of which can result in action being taken under the College's Misconduct regulations.

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

Pricing and Hedging Volatility Index Options

by

Ruben Lewis

Department of Mathematics
King's College London
The Strand, London WC2R 2LS
United Kingdom

Email: ruben.lewis@kcl.ac.uk
Tel: +44 (0)75 7884 9894

29 August 2025

REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MSC IN FINANCIAL MATHEMATICS IN THE UNIVERSITY OF LONDON

Abstract

This dissertation studies the pricing and hedging of volatility index (VIX) derivatives through different methods and techniques available in quantitative finance. First, we calibrate a discretised Ornstein-Uhlenbeck (OU) process to the VIX using 34 years of daily data, the fitted model is used to forecast the VIX over an 18-day time period. Second, we apply the forecasted distribution of the VIX to 3 separate optimisation problems: A static VIX option portfolio optimisation, a VIX digital call option indifference pricing problem and a VIX futures dynamic trading problem using a neural network. The VIX options portfolio and indifference prices are convex optimisation problems computed in a static setting. For our objective function, an entropic risk function is used with two different risk aversion parameters. For the dynamic trading strategy, two time-distributed neural networks are trained on VIX paths using the calibrated OU process to produce two daily trading strategies. The results showed that the OU calibration effectively captured the dynamics of the VIX and could be used to obtain a reliable distribution of the VIX over an 18-day time period. The indifference pricing of the digital options produced economically sensible bid/ask spreads which captured the agents risk-aversion level. The static portfolio optimisation also successfully captured the risk-aversion level of the agent. Finally, the dynamic trading neural networks yielded good strategies, however they were concluded not to be suitable for practical implementation due to the limitations and omissions in creating the dynamic trading strategies.

Acknowledgements

First and foremost, I would like to thank my project supervisor John Armstrong for his guidance and expertise throughout this research.

I would also like to thank my family for their unwavering support during this dissertation and over the course of my studies till this point.

Contents

Introduction

Volatility index options are (VIX) exchange-traded option contracts that use the volatility index as their underlying asset. They provide a more straightforward way for investors to trade volatility as an asset class. Unlike conventional stock options, where the volatility exposure remains vlosely linked to directional price movements of the underlying asset, volatility index options can isolate volatility risk completely. This makes them a useful tool for various trading and risk management applications [10]. In this report, we combine a stochastic model with both static and dynamic optimisation frameworks to obtain insights on investment decisions related to the VIX over an 18-day horizon.

In the first part of this report, we use an Ornstein-Uhlhenbeck (OU) model, calibrated on historical VIX data, to model the logarithm of the VIX over an 18 day time frame. A time series analysis of daily VIX observations over a 34-year period will be conducted to determine the optimal sampling frequency for calibrating the OU model. Once the OU model is fitted, we will use it to create a distribution of the VIX over 18 days. We will obtain the VIX distribution both analytically and through Monte Carlo simulation, providing us with a robust base for the following portfolio optimisation.

Using the forecasts generated from the OU model, we first study a static hedging problem over our 18-day time horizon. The optimisation problem permits a starting wealth of $\$100,000$ and asks the investor to choose from a selection of European-style VIX options, all with the same maturity on 17/07/2024 as well as a risk-free bank account with the borrowing and lending interest rate set to $r = 0$. The VIX options consist of 59 calls and 59 puts with strike prices of 10 to 180. The investor preferences in our static optimisation will be represented through an entropic risk function with two different risk-aversion levels ($\lambda = 2$ vs $\lambda = 20$), allowing for a tractable convex optimisation problem using the MOSEK interior point solver. Budget and box constraints will be applied to the problem, permitting a credible trading problem with close to accurate market conditions. By solving said portfolio optimisation across different levels of risk-aversion we hope to obtain optimal portfolios for varying investors' inclinations.

Following our static portfolio optmisation, we will aim to price 10 digital call options on the VIX with strike prices of 10 to 50. To do this, we will apply indifference pricing to the options using the same investor setup as in the static portfolio optimisation; the investor has an intial wealth of $\$100,000$ with zero liabilities and must choose indifference prices for all the digital options. Thanks to the exponential (entropic) risk functions from the static portfolio optimisation we can reduce the complex indifference pricing problem to a straightforward comparison of certainty equivalents. Once again we will implement this approach across two risk-aversion parameters, hoping to gain insights into the effect of varying degrees of risk aversion on price spreads and option valuation.

Finally, a dynamic trading strategy is developed to explore the use of neural networks in capitalising on daily VIX futures price movements. A time-distributed neural network is trained on OU model simulated futures price paths, using the parameters estimated previously. Two neural networks are trained using separate loss functions: a modified Sharpe loss that should emphasise stable risk-adjusted performance of the trading strategy, and an entropic-risk loss ($\lambda = 2$) that should penalise negative P&L outcomes.

Chapter 1

Literature Review

1.1 What is the VIX?

The VIX, or CBOE Volatility Index, is the most widely used measure of market volatility expectations in equity markets. Created by the Chicago Board Options Exchange (CBOE) in 1993, the VIX calculates expected volatility for the S&P500 index over the next 30 days by analysing the prices of S&P500 options. The basic idea behind the formula which drives the VIX is that when investors expect more market turbulence, they pay higher prices for options, which drives the VIX higher. Conversely, when markets are calm option prices fall and the VIX declines. It is important to understand the VIX is an index rather than a tradeable security itself, therefore, only derivatives of the VIX can be traded [6].

1.1.1 Calculating the VIX

As stated earlier, the VIX measures 30-day expected volatility of the S&P500 Index, and like any other index, the VIX employs rules that govern the selection of its components and a formula to calculate index values. The components of the VIX index are near- and next-term 'out-of-the-money' put and call SPX options centered around an at-the-money strike price [6].

The generalised formula used in the VIX Index Calculation is the following:

$$\sigma^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2} e^{RT} Q(K_i) - \frac{1}{T}[\frac{F}{K_0} - 1]^2 \tag{1.1}$$

Where

σ Annualised standard deviation of volatility

T Time to expiration

F Forward index level derived from SPX option prices

K_0 First strike below the forward index level, F

K_i Strike price of i^{th} out-of-the-money option; a call if $K_i > K_0$ and a put if $K_i < K_0$; or both if $K_i = K_0$

ΔK_i Interval between strike prices

R Risk-free interest rate to expiration

$Q(K_i)$ The midpoint of the bid-ask spread for each option with strike K_i

[6]

Component Sourcing

1. **Time to Expiration (T):**The VIX Index uses near- and next-term SPX options with expirations between 23 and 37 days. The time to expiration is calculated in minutes to replicate the precision that is commonly used by professional option and volatility traders.

2. **Forward Index Level (F):**The forward level F is determined by identifying the strike price (K) at which the absolute difference between the average call and put prices is smallest. The forward level is then obtained from the following formula:

$$\text{F} = \text{Strike Price} + e^{RT} \times (\text{Call Price} - \text{Strike Price})$$

   Separate calculations are performed for near- and next term options (F_1 and F_2).

3. **At-the-money strike (K_0):** The at-the-money strike price (K_0)is determined as the highest strike price immediately below F (for both the near- and next-term options F_1 and F_2).

4. **Out-of-the-money options (K_i):** All strike prices for calls and puts above and below K, respectively, until two consecutive strike prices have no bids. for K_0 both the put and call prices are averaged to form a single value.

5. **Strike interval (ΔK_i):** Half the difference between adjacent strikes, calculated as half the difference between the nearest strike prices on either side of K_i:

$$\Delta K_i = \frac{K_{i+1} - K_{i-1}}{2} \tag{1.2}$$

6. **Risk-Free Rate (R):** Derived from U.S. Treasury yield curve rates using a cubic spline to match option expiration dates. Separate rates (R_1, R_2) are used for near- and next-term options.

7. **Final VIX Calculation** Once the variances (σ_1, σ_2) for near- and next-term options are interpolated to a 30-day weighted average, they are annualised and multiplied by 100 (VIX is expressed as a percentage):

$$\text{VIX} = 100 \times \sqrt{\{T_1\sigma_1^2(\frac{N_{T_2} - N_{30}}{N_{T_2} - N_{T_1}}) + T_2\sigma_2^2(\frac{N_{30} - N_{T_1}}{N_{T_2} - N_{T_1}})\} \times \frac{N_{365}}{N_{30}}} \tag{1.3}$$

where N_{T_1}, N_{T_2}, N_{30} and N_{365} are minutes to expiration for near-term, next-term, 30-day, and annual periods, respectively.

1.1.2 Deriving the VIX formula

As can be seen in section 1.1.1, the VIX is simply the weighted average of two implied volatilities, next-term volatility and long-term volatility. While the CBOE's white paper [6] provides details on the formula's inputs, it does not address the theoretical foundation of the formula itself.

The derivation of the VIX formula is intrinsically linked to fair pricing of variance swaps. A variance swap is an agreement between two parties to exchange the realized variance of an asset between time 0 and time T for a prespecified fixed volatility [8]. Its payout is equal to:

$$(\sigma_R - K_{vol}) \times \text{N} \tag{1.4}$$

where σ_R is the realised variance, K_{vol} is the agreed fixed variance (strike variance) and N is the notional.

Hence, the execution of a volatility swap requires both parties to agree on the fixed volatility level, with the most straightforward approach being to set this parameter to the expected realised variance at maturity under the risk-neutral measure.

$$K_{vol}^2 := \mathbb{E}_{\mathbb{Q}}[\frac{1}{T}\int_0^T \sigma_t^2 \, dt] \tag{1.5}$$

Here is how equation (1.1) is computed:

Following Hull's approach [8], assume the price of a stock has the following dynamics.

$$\frac{dS}{S} = \mu_t \, dt + \sigma_t \, dz \tag{1.6}$$

in a risk-neutral world where σ_t is itself stochastic. From Ito's lemma

$$d(\ln S) = \left(\mu_t + \frac{\sigma_t^2}{2}\right) dt + \sigma_t \, dz \tag{1.7}$$

From Gunderson's notes on deriving the vix [7], we know to subtract equation 1.7 to equation 1.6, giving us

$$\frac{dS}{S} - d(\ln(S) = \frac{\sigma_t^2}{2} \, dt \tag{1.8}$$

Integrating and scaling both sides by 2/T, we get

$$\frac{1}{T}\int_0^T \sigma_t^2 \, dt = \frac{2}{T}\int_0^T \frac{dS}{S} - \frac{2}{T}\int_0^T d(\ln(S)) \tag{1.9}$$

The left-hand-side of equation (1.6) is the total variance from time 0 to time T.

$$V = \frac{1}{T}\int_0^T \sigma_t^2 \, dt$$

Taking expectations under the risk-neutral measure, we obtain:

$$\mathbb{E}_Q(V) = \frac{2}{T}\mathbb{E}_Q\left[\int_0^T \frac{dS}{S}\right] - \frac{2}{T}\mathbb{E}_Q\left[\ln\frac{S_T}{S_0}\right] \tag{1.10}$$

The first expectation equals the risk-rate times T, which gives us:

$$\frac{2}{T}\mathbb{E}_Q\left[\int_0^T \frac{dS}{S}\right] = RT = \ln\left(\frac{F}{S_0}\right) \tag{1.11}$$

where $F = S_0 e^{RT}$. Thus we can write

$$\mathbb{E}_Q[V] = \frac{2}{T}\ln\frac{F}{S_0} - \frac{2}{T}\mathbb{E}_Q(\ln\frac{S_T}{S_0}) \tag{1.12}$$

Now that we have expressed the expected realised variance at maturity, what remains is to express equation (1.12) in terms of option prices (tradeable). This will allow us to construct a portfolio that replicates the strike variance, and the fair price of the variance swap is the cost of constructing this replicating portfolio.

Following [7], the Carr-Madan formula (special case of Taylor's formula) states:

$$\ln S_T = \ln S_0 + \frac{S_T - S_0}{S_0} - \int_0^{S_0} \frac{\max[0, K - S_T]}{K^2}\,dK - \int_{S_0}^{\inf} \frac{\max[0, S_T - K]}{K^2}\,dK \tag{1.13}$$

The integrals represent portfolios of put and call options. Once again, taking expectations under the risk-neutral measure:

$$\mathbb{E}_Q\left[\ln S_T\right] = \ln S_0 + \frac{F - S_0}{S_0} - \int_0^{S_0} \frac{e^{RT}p(K)}{K^2}\,dK - \int_{S_0}^{\infty} \frac{e^{RT}c(K)}{K^2}\,dK \tag{1.14}$$

Remembering $\mathbb{E}_Q\left[\ln\frac{S_T}{S_0}\right] = \mathbb{E}_Q\left[\ln S_T\right] - \ln S_0$ and putting it all together we have obtained the fair price of a variance swap

$$K_{vol}^2 = \frac{2}{T}\ln\frac{F}{S_0} - \frac{2}{T}\left[\frac{F}{S_0} - 1 - \int_0^{S_0}\frac{e^{RT}p(K)}{K^2}\,dK - \int_{S_0}^{\infty}\frac{e^{RT}c(K)}{K^2}\,dK\right] \tag{1.15}$$

Finally, the VIX formula approximates the continous integrals with a sum over a set of options:

$$K_{vol}^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2}e^{RT}Q(K_i) + \frac{2}{T}\ln\frac{F}{S_0} - \frac{2}{T}\left[\frac{F}{S_0} - 1\right] \tag{1.16}$$

And the second-order Taylor approximation:

$$\ln(1+x) \approx x - \frac{x^2}{2} \tag{1.17}$$

where

$$x = \frac{F}{S_0} - 1 \tag{1.18}$$

giving us:

$$\ln(\frac{F}{S_0}) \approx \left[\frac{F}{S_0} - 1\right] + \frac{1}{2}\left[\frac{F}{S_0} - 1\right]^2 \tag{1.19}$$

Hence the complete VIX formula is,

$$K_{vol}^2 = \frac{2}{T}\sum_i \frac{\Delta K_i}{K_i^2}e^{RT}Q(K_i) - \frac{1}{T}\left[\frac{F}{K_0} - 1\right]^2 \tag{1.20}$$

Note the VIX computes the forward price F using the put-call parity, and uses K_0 as an approximation for the spot price S_0. This is why K_0 is used in the final formula.

1.2 Optimal Investment

The portfolio problem of an investor trading in different financial assets is to choose an optimal investment and consumption strategy. More specifically, given an initial capital w, an investor must determine which assets to hold onto in what quantities at what time instant in order to maximise his overall wealth at time horizon T and/or his expected utility of consumption within the time interval [0,T] [9]. The portoflio problem may be modelled in two different ways:

i) Discrete time setting

ii) Continuous time setting

both of which have their own advantages and limitations.

1.2.1 Discrete Time Models

The Markowitz model is the oldest method for resolving the portfolio problem [9]. It is a one-period (static) discrete time model consisting of a one-off decision at the start of the period (t=0) and no further action until the end of the period (t=T). The premise of the Markowitz model lies its formulation of portfolio optimization through mean-variance analysis. Indeed, Markowitz measures portfolio risk through the variance of the position, this allows his method to capture not only the individual volatilities of assets but also their correlations through the covariance matrix.

Assuming the random vector $\mathbf{r}$, whose entries are given by returns of each stock i, follows a multivariate normal distribution with mean μ and covariance matrix $\mathbf{\Sigma}$ [3] and taking a vector of weights $\mathbf{w}$ that determines the portfolio allocations. The portfolio return is given by:

$$r^{\text{portfolio}} = \sum_{i=1}^{n} r_i w_i = \mathbf{r}^T \mathbf{w} \tag{1.21}$$

The expected portfolio return and variance are:

$$\mathbb{E}\left[r^{\text{portfolio}}\right] = \sum_{i=1}^{n} \mu_i w_i = \mu^T \mathbf{w} \tag{1.22}$$

$$\text{Var}\left[r^{\text{portfolio}}\right] = \sum_{i=1}^{n} \sum_{j=1}^{n} \mu_i w_i = \mu^T \mathbf{\Sigma} \mathbf{W} \tag{1.23}$$

Markowitz portfolio optimization therefore consists of achieving a target expected return R while minimizing variance and can be formulated in matrix form as:

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^n} \quad & \mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \\ \text{subject to} \quad & \mathbf{1}^T\mathbf{w} = 1 \\ \text{and} \quad & \boldsymbol{\mu}^T\mathbf{w} = R \end{aligned} \tag{P1}$$

Solving this optimization problem for different values of R generates the efficient frontier [3]. This simplistic approach is highly valued and achieved widespread use amongst practitioners and academic, warranting Markowitz's Nobel prize in economics [9].

Of course, there are many other discrete-time models and corresponding methods for portfolio optimisation. Looking at a problem setup (P1), one could change it to

$$\begin{aligned} \max_{w_i\in\mathbb{R}} \quad & \mathbb{E}(U(R)) \\ \text{subject to} \quad & \sum_{i=1}^{n} w_i = 1, \quad i = 1, \ldots, n \end{aligned} \tag{P2}$$

where we are now maximising a general utility function $U(.)$ of the portfolio returns R, noting that a finite optimal solution can only be obtained for certain utility functions.

So far we have only looked at one-period (static) models, one can create a muti-period model in which the investor can reallocate his investments w_i at some given trading dates 0,1,...,T-1, before the time horizon T. This means the investor must now create a portfolio process $\pi(t)$, where at each timestep the portfolio weights are rebalanced according to $\pi(t)$, without knowledge of asset price movements at later trading dates. Assuming that the investor operates in a self-financing way (i.e., the value of their holdings prior to $\pi(t)$ equals the value of their holdings subsequent to $\pi(t)$), we can formulate a multi-period variant of problem (P2). Assuming that for each trading date t, the set of tradeable security prices is finite, problem (P2) can be resolved using dynamic programming methods [9].

1.3 Pricing Theory

Over 50 years ago, Fischer Black, Myron Scholes and Robert Merton created the famous Black-Scholes option pricing formula which is widely regarded nowadays as one of the cornerstones of mathematical finance. The Black-Scholes-Merton framework introduced the concept of risk-neutral valuation, demonstrating that option prices could be determined by discounting expected payoffs under a risk-neutral probability measure, rather than the actual physical probabilities governing market movements [18]. The approach involves setting up a riskless portfolio consisting of the option and the underlying stock and arguing that the return of the portfolio must be the risk-free return. Using this assumption, we can work out the cost of setting up the portfolio and therefore the option's price [8]. Here is how the Black-Scholes-Merton differential equation is derived according to Hull [8].

The assumptions used to derive the Black-Scholes-Merton differential equation are the following:

1. The stock price follows the process in equation (1.24).
2. The short selling of securities with full use of proceeds is permitted.
3. There are no transaction costs or taxes. All securities are perfectly divisible.
4. There are no dividends during the lifetime of the derivative.
5. There are no riskless arbitrage opportunities.
6. Security trading is continuous.
7. The risk-free interest rate is constant and the same for all maturities.

We consider the the price of a derivative security at time t, where T is the maturity date, so the time to maturity is $T - t$. The underlying stock price process we are assuming is the following:

$$dS = \mu S\, dt + \sigma S\, dz \tag{1.24}$$

where S is the stock price μ is the drift, σ is the volatility and dz is Wiener process increment. Letting $f(S, t)$ be the price of a derivative. From Ito's lemma (see appendix A), it follows that the process followed by a function f of S and t is:

$$df = \left(\frac{\delta f}{\delta S} \mu S + \frac{\delta f}{\delta t} + \frac{1}{2} \frac{\delta^2 f}{\delta S^2} \sigma^2 S^2 \right) dt + \frac{\delta f}{\delta S} \sigma S \, dz \tag{1.25}$$

looking at equation (1.25), we see that df has both a deterministic component (the dt term) and a random component (the dz term). It thus follows that a portfolio of the stock and the derivative can be constructed so that Wiener process is eliminated (eliminate risk). The holder of this portfolio must short one derivative and go long an amount $\frac{\delta f}{\delta S}$ in shares of the underlying. Defining Π as the value of the portfolio, we have

$$\Pi = -f + \frac{\delta f}{\delta S} S \tag{1.26}$$

The change in portfolio value over a small time interval δt is

$$\Delta \Pi = -\Delta f + \frac{\delta f}{\delta S} \Delta S \tag{1.27}$$

Substituting the discrete approximations of equations (1.24) and (1.25) into equation (1.27)

- $\Delta S = \mu S \, \Delta t + \sigma S \, \Delta z$
- $\Delta f = \left(\frac{\delta f}{\delta S} \mu S + \frac{\delta f}{\delta t} + \frac{1}{2} \frac{\delta^2 f}{\delta S^2} \sigma^2 S^2 \right) \Delta t + \frac{\delta f}{\delta S} \sigma S \Delta z$

we obtain:

$$\Delta \Pi = \left(-\frac{\delta f}{\delta t} - \frac{1}{2} \frac{\delta^2 f}{\delta S^2} \sigma^2 S^2 \right) \Delta t \tag{1.28}$$

Since this equation does not involve Δz, the portfolio is risk-free during time Δt. The no-arbitrage assumption listed earlier implies that the portfolio must return the same rate of return as other short term risk-free securities. If it earned more, arbitrageurs could make a riskless profit by borrowing the money to buy the portfolio and investing the proceeds in a risk-free bank account. If it earned less, arbitrageurs could make a riskless profit by short-selling the portfolio and investing the proceeds in a risk-free bank account.

$$\Delta\Pi = r\Pi\,\Delta t \tag{1.29}$$

where r is the risk-free interest rate. Substituting equations (1.23) and (1.25), we have

$$\left(\frac{\delta f}{\delta t} + \frac{1}{2}\frac{\delta^2 f}{\delta S^2}\sigma^2 S^2\right)\,\Delta t = r\left(f - \frac{\delta f}{\delta S}S\right)\,\Delta t \tag{1.30}$$

Rearranging,

$$\frac{\delta f}{\delta t} + rS\frac{\delta f}{\delta S} + \frac{1}{2}\sigma^2 S^2\frac{\delta^2 f}{\delta S^2} = rf \tag{1.31}$$

is the Black-Scholes-Merton differential equation, one of the solutions to this partial differential equation (PDE) is the Black-Scholes formula for the prices of European call and put options. It can be solved analytically subject to European option boundary conditions. However, the key property that arises from this PDE is that it does not contain any variables that are influenced by the investors risk preferences. The PDE includes only the current stock price, the time, the stock price volatility and the risk-free interest rate, all of which are independent of a market participant's risk preferences.

Since the Black-Scholes-Merton differential equation is independent of risk preferences, its solution must also be independent of risk preferences. This observation, combined with the Feynman-Kac theorem (see appendix A) leads to a neat computational trick. Instead of solving the the PDE analytically, the solution can be computed as an expected value under the risk-neutral measure.

Under the risk-neutral assumption, the expected return on any asset is the risk-free interest rate. We can thus price derivatives without explicitly modeling investor risk preferences. The pricing problem is now very straightforward: the derivative's value is equal to the expected payoff under the risk-neutral measure, discounted at the risk-free rate [8]. From the risk-neutral valuation argument, the price of a European call option at time t is given by:

$$f(S,t) := \mathbb{E}_t^{\mathbb{Q}}[e^{-r(T-t)}\max(S_T - K, 0)] \tag{1.32}$$

This approach to derivative pricing relies on many strong assumptions (as listed earlier). These assumptions can hinder the use of risk-neutral pricing in the real world as real markets often don't reflect idealised market conditions due to transaction costs, incomplete markets, liquidity constraints, and varying risk preferences among market participants. These limitations are the reason other pricing methodologies exist, most notably indifference pricing, which incorporate an agent's risk preference into the valuation process.

1.3.1 Indifference Pricing

Instead of seeking a fair price for the asset, indifference pricing admits that the value of a financial instrument may change depending on an agents respective utility functions, current portfolio, and risk tolerances. The theoretical foundation of indifference pricing stems from the asset-liability management (ALM) framework. Asset-liability management can be thought of as a specific kind of portfolio optimisation where an agent's existing liabilities are incorporated alongside their investment decisions, the investor is trying to invest to best cover his obligations as opposed to trying to maximise expected returns or utility [13].

Consider an agent whose financial position is described by an initial wealth $w \in \mathbb{R}$ at time $t = 0$ and a financial liability of delivering a claim $c \in L^0$ at time $t = 1$. The agent can buy a portfolio $x \in \mathbb{R}^J$ at time $t = 0$ and use the proceeds to cover his liabilities. This ALM problem can be written as an optimisation problem of the following form:

$$\begin{aligned} \min \quad & \mathcal{V}(c - s_1 \cdot x) \quad \text{over} \quad x \in D \\ \text{Subject to} \quad & s_0 \cdot x \leq w, \end{aligned} \tag{ALM}$$

where $D \subseteq \mathbb{R}^J$ is a convex set which describes portfolio constraints and the utility function $\mathcal{V} : L^0 \mapsto \bar{\mathbb{R}}$ is a nondecreasing convex function with $\mathcal{V}(0 = 0)$, which is a numerical representation of the agent's risk preferences concerning net expenditure $c - s_1 \cdot x$ at time $t = 1$ [13]. The convexity of D and $\mathcal{V}$ imply that (ALM) is a convex optimisation problem. The optimal value function $\phi : \mathbb{R} \times L^0 \mapsto \bar{\mathbb{R}}$ of (ALM) is given by:

$$\phi(w, c) := \inf\{\mathcal{V}(c - s_1 \cdot x) | x \in D, s_0 \cdot x \leq w\}. \tag{1.33}$$

$\phi(w, c)$ is convex, nonincreasing in w, nondecreasing in c and $\phi(0, 0) = 0$. The proof of this can be found in [13]. These properties of the optimal value function directly inform the structure and behaviour of indifference prices, this will be discussed at the end of this subsection.

The indifference selling price $\pi_s(\bar{w}, \bar{c}; c)$ is thus given as minimum amount of capital w an agent would accept for delivering a claim c without worsening his financial position [13]. It is expressed, for an agent whose financial position is described by $(\bar{w}, \bar{c}) \in \mathbb{R} \times L^0$, as:

$$\pi_s(\bar{w}, \bar{c}; c) = \inf\{w | \phi(\bar{w} + w, \bar{c} + c) \leq \phi(\bar{w}, \bar{c})\} \tag{1.34}$$

This definition captures the economic intuition that an agent would only accept a price that leaves them no worse off than their current position. Similarly, the indifference buy price of an agent whose financial position is described by $(\bar{w}, \bar{c}) \in \mathbb{R} \times L^0$, is given by:

$$\pi_l(\bar{w}, \bar{c}; c) = \inf\{w | \phi(\bar{w} - w, \bar{c} - c) \leq \phi(\bar{w}, \bar{c})\} \tag{1.35}$$

It represents the opposite of the indifference selling price, that is, the maximum amount of money an agent would be willing to pay to reduce their liabilities by c. This formulation incorporates subjective factors that are not present in risk-neutral pricing. These are: the agent's current financial position $(\bar{w}, \bar{c})$, their risk preferences through the utility function V and their probability beliefs through the measure P if the utility function contains an expectation.

Finally, it is important to note that the convexity of the optimal value function $\phi : \mathbb{R} \times L^0 \mapsto \bar{\mathbb{R}}$ is a manifestation of the diversification principle [13], where the risk of combining different financial positions should not be greater than the weighted average of the individual risks. The convexity of ϕ ensures the indifference prices are convex functions of c and thus satisfy economic principles. Notably, it implies that indifference prices are bounded by their supporting hyperplanes, these are known as arbitrage bounds and coincide with replication-based prices from complete market models [13].

Furthermore, the convexity of indifference prices guarantees that the buying price will never exceed the selling price of a claim, maintaining rational market behaviour [13]. This pricing method offers a theoretical foundation for bid-ask spreads provides a mathematical explanation of why different agents with different financial positions are inclined to engage in a trade. Thus, indifference pricng offers a coherent pricing frameowrk in incomplete markets.

1.3.2 The Black-76 Model

Looking at the earlier derivation of the Black-Scholes-Merton differential equation and its solution, the Black-Scholes option pricing formula, it is clear that one of the key assumptions in deriving this formula is the that trading of the underlying variable is possible and happens in continuous time. Thus, a logical question to ask is: How does one price an option for which the underlying is not tradeable (futures based). Fischer Black's 1976 paper 'The Pricing of Commodity Contracts' [4] introduced a modified version of the Black-Scholes model to address this.

The key insight in [4], is that one should use the futures price as the spot price, assuming futures prices have the same lognormal distribution at time T (See appendix B), as well as maintaining the other core assumptions of the Black-Scholes model.

The Black-76 model extended the risk-neutral valuation framework to futures based derivatives. Traders often prefer options written on futures over options written on an underlying assets primarily because futures markets offer superior liquidity and transparency [8]. This is because futures prices can be seen instantly on exchanges while spot prices may not be so easily available. Furthermore, the operational advantages extend further as futures and options trade on the same exchange, this makes hedging and arbitrage strategies more effective [8].

The Black-76 formula, as derived in Black's original work [4], provides a closed-form solution for European options on futures contracts. The pricing equations are:

$$c = e^{-rT}\left[FN(d_1) - KN(d_2)\right] \tag{1.36}$$

$$p = e^{-rT}\left[KN(-d_2) - FN(-d_1)\right] \tag{1.37}$$

where:

$$d_1 = \frac{\ln(F/K) + (\sigma^2/2)T}{\sigma\sqrt{T}} \tag{1.38}$$

$$d_2 = d_1 - \sigma\sqrt{T} \tag{1.39}$$

and the parameters are defined as:

- c European call option price
- p European put option price
- F current futures price
- K strike price
- r risk-free interest rate
- T time to maturity
- σ volatility of the futures price
- $N(\cdot)$ cumulative standard normal distribution function

A critical observation in the Black-76 formula is that it does not require the options contract's maturity T and the futures contract delivery date T' to be the same [8].

1.4 Machine Learning Applications in Finance

The integration of machine learning (ML) technologies, particularly deep neural networks, has extended the boundaries quantitative finance [14]. A 2022 survey by the bank of England reported that 72% of UK financial services firm were already using or developing ML applications [5]. Although most of the current ML applications in finance are not related to quantitative finance, ML can play a key role asset pricing accuracy and determining new investment strategies for asset allocation.

1.4.1 Deep Pricing and Other Deep Learning Techniques in Quantitative Finance

Deep pricing generally refers to the application of deep learning techniques, usually neural networks, to price financial instruments such as options and other assets. As mentioned in the previous section, there are various approaches to asset pricing. However, unquestionably, one of the key elements to asset pricing is prediction.

In the passed, financial research was heavily based on a handful of variables extracted from sources like annual Moody's manuals which summarised corporate financial reports from all companies. More recently, there has been an exponential growth in potentially relevant predictor variables. The SEC's Edgar database alone contains terabytes of financial report data, combined with market transaction history, social media sentiment, satellite imagery and other alternative data sources, investors are overloaded with information, and traditional models do not work. This abundance of information, according to Nagel [11] is the premise for the emergence of deep pricing methods, it is known as the high-dimensional challenge in modern asset pricing.

Deep learning techniques offer a powerful alternative approach to some quantitative finance problems. Firstly, in recent years there has been significant enhancement in the speed and generality of deep learning techniques in quantitative finance [14]. Deep learning provides numerical alternatives to traditional methods for pricing derivatives, portfolio optimization, and forecasting. Unlike traditional linear models that require explicit specification of the included variables, neural networks can automatically learn complex, non-linear relationships from high-dimensional data without requiring researchers to pre-specify the functional form or impose strict sparsity constraints [11]. These algorithms thus overcome the 'curse of dimensionality' and obtain solutions faster than traditional methods.

Other innovations of deep learning in finance include the use of carefully chosen network topologies, designed to automatically derive approximate optimal solutions in scenarios where no theoretical solutions exist [14]. For example, hedging strategies that account for more complex market conditions. This goes hand-in-hand with the progress in advanced market simulation through market generators (market data simulators). Exemplifying how modeling has evolved from accepting simplified assumptions for mathematical convenience to requiring high-quality datasets and advanced model evaluation techniques [14].

These advancements in data focused modeling have opened the door to many prediction algorithm applications. However, Nagel [11] argues the idea that one could make predictions in an entirely data driven automated fashion is too good to be true in the context of asset pricing. Price action datasets and other financial data characterised by low-signal-to-noise ratios and successful prediction strategies being arbitraged away once discovered are examples of the possible limitations of machine learning in finance [11].

Chapter 2

Pricing and Hedging VIX Options in a Static Setting

2.1 Modelling the VIX

Option traders are highly dependent on volatility values when trading. If the implied volatility of an option is deemed too high, the option is considered to be overpriced, and vice versa. Strategies like straddles profit from price movements rather than directional bets on the underlying asset, making implied volatility a critical tool in option trading as well as in risk and portfolio management. Traditionally, implied volatility was derived from models like Black-Scholes, but model-free indices like the VIX now dominate [1].

Due to its popularity, the VIX has been the subject of extensive research and many attempts have been made to try and model the VIX.

In [1], it is said an ARIMA(1,1,1) specification is the best fit to VIX historical data and in terms of forecasting, an ARIMA(1,1,1)-GARCH(1,1) model obtains the best directional accuracy for the VIX index. However, in this study, we will proceed with an AR(1) process as in the task requirements, assuming the daily values of the logarithm of the VIX follow a discrete-time Ornstein-Uhlenbeck process ARIMA(1,0,0).

$$\Delta\xi_t = -a(\xi_{t-1} - \bar{\xi}) + \sigma\epsilon_t t = 0, 1, ... \tag{2.1}$$

where $\Delta\xi_t := \xi_t - \xi_{t-1}$, ϵ_t are i.i.d Gaussian variables with zero mean and unit variance a is the speed of mean-reversion, $\bar{\xi_t}$ is the long-term mean of the logarithm of VIX values and σ is the volatility.

2.1.1 Time Series Analysis

In our approach we are looking at 34 years of daily VIX opening values from 02/01/1990 to to 04/11/2024, we analyse both 1-day and 18-day intervals. The 18-day interval corresponds to the duration of the options in the portfolio optimisation in section 2.2.

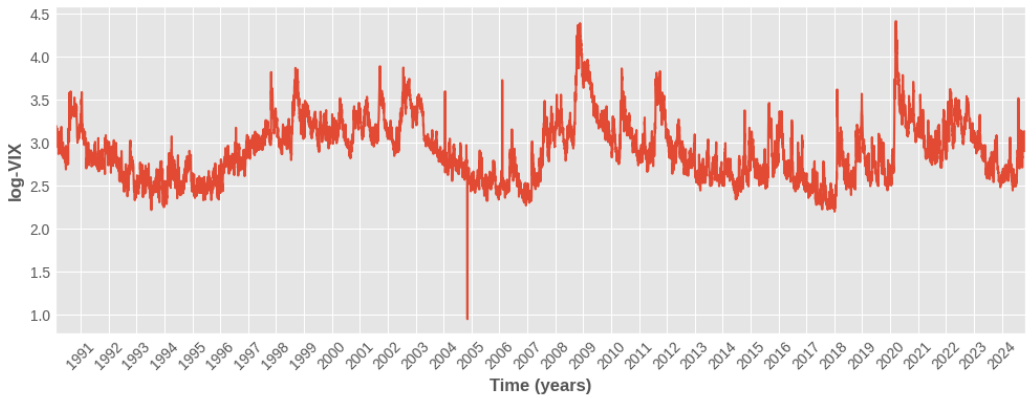


Figure 2.1: VIX Daily Opening Prices

Looking at table 2.1, we see that both time series show strong evidence of stationarity, as the Augmeted-Fuller-Dickey test shows. This supports our hypothesis that the VIX exhibits mean-reverting behaviour, thus validating the use of the Ornstein-Uhlenbeck model.

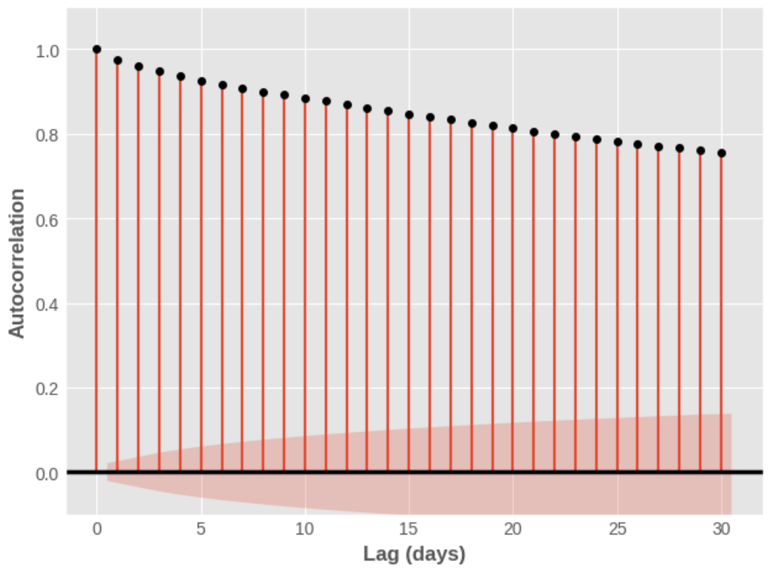


Figure 2.2: VIX Time Series Autocorrelation Plot

The autocorrelation graph in figure 2.2 reveals persistent correlation in the VIX time series. The autocorrelation plot shows a gradual decay from very high correlation at lag 1 that consistently decreases through to lag 30. This pattern is also consistent with mean-reverting AR(1) processes.

Test Statistic	**1 day time period**	**18 day time period**
ADF Score	-6.1637	-4.0612
p-value	7.0744×10^{-8}	1.1220×10^{-3}

Table 2.1: ADF Test Results for Stationarity

The stronger ADF test results for the 1-day interval suggests that daily observations of the VIX captures the mean-reverting dynamics of the VIX more precisely than the 18-day interval data. Therefore, since the volatility's properties are more pronoucned at higher frequencies, we will use the daily observations model for the portfolio optimisation.

2.1.2 Parameter Estimation

In order to estimate the parameters of discretised Ornstein-Uhlenbeck model for VIX data, the least-squares regression technique was used. To set up the least squares regression we first had to rearrange equation 2.1 into standard AR(1) form.

$$\Delta\xi_t = -a(\xi_{t-1} - \bar{\xi}) + \sigma\epsilon_t \tag{2.2}$$

$$\xi_t = a\bar{\xi} + (1-a)\xi_{t-1} + \sigma\epsilon_t$$

.

In standard AR(1) form:

$$\xi_t = \alpha + \beta\xi_{t-1} + \eta_t \tag{2.3}$$

where $\alpha = a\bar{\xi}$ is the intercept term, $\beta = (1-a)$ is the auto-regressive coefficient and η_t is the error term with variance σ^2.

In the context of least-squares regression we use the risk function $r(y, y') = ||y - y'||^2$. Following the notation in [14], letting X and Y denote the dependent and independent variables of the regression task respectively, we assume $\mathcal{R}_Y = \mathbb{R}^q$ and $\mathcal{R}_X = \mathbb{R}^d$. Given a vector $x \in \mathbb{R}^d$, we define the vector $\tilde{x} = \begin{bmatrix} 1 \\ x \end{bmatrix} \in \mathbb{R}^{d+1}$. Then we re-write any linear predictor f, as $f = \beta^T\tilde{x}$, where $\beta = \begin{bmatrix} \beta_0^T \\ B \end{bmatrix} \in \mathbb{R}^{(d+1)\times q}$. Thus we can rewrite equation (2.3) as:

$$Y_t = X_t\beta + \text{residual noise} \tag{2.4}$$

where we define the vectors $Y_t = \xi_t$, $X_t = \begin{bmatrix} 1 \\ \xi_{t-1} \end{bmatrix}$ and $\beta = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$. Under standard statistical assumptions [14], the maximum likelihood estimator of β, based on a training set $T = \{x_1, y_1, ..., x_N, y_N\}$ minimises the residual sum of squares:

$$\text{RSS}(\beta) = N\hat{R}(f) = \sum_{i=1}^{N} ||y_i - f(x_i)||^2 = \sum_{i=1}^{N} ||y_i - \beta^T\tilde{x}_i||^2$$

Thus, to compute the optimal regression parameters, we must solve a system of matrix equations. In our case $d = q = 1$, so we have simple system of linear equations. In matrix form we have:

$$\mathcal{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_1^{(d)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^{(1)} & \dots & x_N^{(d)} \end{bmatrix}, \mathcal{Y} = \begin{bmatrix} y_1^{(1)} & \dots & y_1^{(q)} \\ \vdots & \ddots & \vdots \\ y_N^{(1)} & \dots & y_N^{(q)} \end{bmatrix}$$

we solve the convex minimisation problem

$$\min_{\beta \in \mathbb{R}^{(d+1)\times q}} \text{RSS}(\beta) = ||\mathcal{Y} - \mathcal{X}\beta||_F^2$$

.

The solution to linear least-squares regression can be found in appendix A

Results

Two sets of model parameters were estimated using 8791 daily VIX observations and 462 18-day intervalled VIX observations, both spanning over the same amount of time. Given the ADF score for daily intervals showed better evidence of stationarity than that for 19-day intervals, it was expected that the daily interval dataset would yield more reliable parameters and better calibration.

Parameters	**a**	$\bar{\xi}$	σ
1-day interval	0.0245	2.9101	0.0766
18-day interval	0.1555	2.9132	0.1875

Table 2.2: Ornstein-Uhlenbeck Parameter Estimation

The comparison of both time series' regression results shows small differences in the estimated dynamics of the VIX. The daily observations have a mean-reversion speed of $a = 0.0245$ per day, corresponding to a half life of volatility shocks of 28 days. This means that VIX deviations' from its mean-reversion level revert back in just under a month. In contrast, the 18-day interval data exhibited a slightly slower mean reversion speed of $a = 0.1555$ per 18-day period (0.0086 per day), corresponding to a half of about 29 days. Both time series' had identical long-term mean reversion level of $\bar{\xi} \approx 2.91$, indicating a robust estimation of the equilibrium VIX level around $e^{2.91} \approx 18.4$ days.

Regarding the model fit, the daily intervals achieved a greater accuracy with a mean-squared-error of 0.00587 versus 0.03315 for the 18-day interval time series, suggesting higher volatility in the latter. However, the volatility parameter estimates did not validate this. Using the daily data yielded a volatility of $\sigma = 0.0766$, while the 18 day intervals had a volatility of $\sigma = 0.1875$. In standard Brownian motion, the volatility should scale proportionally with the square root of time, such that $\sigma_{daily} = \sigma_{\text{18-day}} \approx 0.0766 \times \sqrt{18} = 0.3250$, this is almost the double the actual 18-day volatility. This discrepancy reflects the mean-reverting nature of the VIX, where unlike in pure random walks where the shocks can accumulate, much of the volatility is dampened due to the mean-revsersion process, thus leading to lower volatility parameters over longer time periods.

These results demonstrate that while both frequencies capture the fundamental mean-reverting nature of VIX, the daily observations give us a a better idea of the short-term volatility, which could be especially useful in our short-term portfolio optimisation in section 2.2.

2.1.3 Forecasting the VIX

Having established reliable Ornstein-Uhlenbeck parameters using the daily observations time series, we can now derive the theoretical distribution of the VIX at future time horizons. Given our investment framework in section 2.2 requires an 18-day holding period for the options portfolio, the interest is in finding the mean and standard deviation of ξ_T at this specific maturity.

To determine the statistical properties of ξ_T, we employ two different approaches. First, we derive the analytical expectation and variance by exploiting the mathematical tractability of the Ornstein-Uhlenbeck process, which provides closed-form expressions. Second, we implement a Monte Carlo simulation that generates numerous synthetic VIX trajectories, allowing us to empirically estimate the expectation and variance to validate our analytical results.

Derivation of Analytical Mean and Variance:

Transforming the discretised Orstein-Uhlenbeck process in equation 2.1 into a continuous-time stochastic differential equation (SDE), we have:

$$d\xi_t = a(\bar{\xi} - \xi_t)dt + \sigma dW_t \tag{2.5}$$

Solving this SDE requires the use of an integrating factor, thus we multiply by e^{at}

$$e^{at}d\xi_t + e^{at}a\xi_t dt = e^{at}a\bar{\xi}dt + e^{at}\sigma dW_t \tag{2.6}$$

Looking at the left hand side, it is clear it is equal to the differential of $e^{at}\xi_t$

$$\begin{aligned} \text{LHS} &= e^{at}d\xi_t + e^{at}a\xi_t dt && (2.7)\\ &= d(e^{at}\xi_t) && (2.8) \end{aligned}$$

Thus, integrating both sides from t to s:

$$\begin{aligned} e^{as}\xi_s - e^{at}\xi_t &= a\bar{\xi}\int_t^s e^{au}du + \sigma\int_t^s e^{au}dW_u && (2.9)\\ &= \bar{\xi}(e^{as} - e^{at}) + \sigma\int_t^s e^{au}dW_u && (2.10) \end{aligned}$$

Dividing by e^{as} and rearranging

$$\xi_s = \bar{\xi} + (\xi_t - \bar{\xi})e^{-a(s-t)} + \sigma\int_t^s e^{-a(s-u)}dW_u \tag{2.11}$$

gives us the explicit solution to the SDE. Finally, we can obtain an expression for the mean and variance.

$$\mathbb{E}[\xi_s|\xi_t] = \bar{\xi} + (\xi_t - \bar{\xi})e^{-a(s-t)} + \sigma\mathbb{E}\left[\int_t^s e^{-a(s-u)}dW_u\right] \tag{2.12}$$

Since $\mathbb{E}\left[\int_t^s e^{-a(s-u)}dW_u\right] = 0$:

$$\mathbb{E}[\xi_s|\xi_t] = \bar{\xi} + (\xi_t - \bar{\xi})e^{-a(s-t)} \tag{2.13}$$

gives us the conditional mean.

$$\text{Var}(\xi_s|\xi_t) = \text{Var}\left(\sigma\int_t^s e^{-a(s-u)}dW_u\right) = \sigma^2 e^{-2as}\text{Var}\left(\int_t^s e^{au}dW_u\right) \tag{2.14}$$

From Ito isometry we know

$$\mathrm{Var}\left(\int_t^s e^{au}dW_u\right) = \int_t^s e^{2au}du = \frac{1}{2a}(e^{2as} - e^{2at}) \tag{2.15}$$

Therefore,

$$\mathrm{Var}(\xi_s|\xi_t) = \sigma^2 e^{-2as}\frac{1}{2a}(e^{2as} - e^{2at}) \tag{2.16}$$

$$= \frac{\sigma^2}{2a}(1 - e^{-2a(t-s)}) \tag{2.17}$$

gives us the conditional variance.

Monte Carlo simulation:

The Monte Carlo method approximates the expected value of a variable by simulating it through independent repetitions, with the strong law of large numbers ensuring convergence [17]:

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i = \mathbb{E}[X] \;\; P - a.s. \tag{2.18}$$

The value of the expectation of the VIX after 18-days can be estimated by performing random independent simulations and averaging the final simulated values for the VIX at maturity. The Ornstein-Uhlenbeck model with the parameters from section 2.1.2 was used. By applying the Euler-Maruyama scheme to equation 2.5, this continuous time-equation can be formulated into a discrete-time equation [3].

$$\xi_{t+\delta t} = \xi_t - a(\xi_t - \bar{\xi})\delta t + \sqrt{\delta t}\epsilon_t^i \;\; i = 1, 2, \ldots, n \tag{2.19}$$

where $\epsilon_t \sim N(0, 1)$.

Results

We apply the estimated parameters (from the daily time series) to equations 2.13 and 2.17 with $t = 0$, $s = 18$ and $\xi_t = \ln 13.2$ to give us the theoretical mean and standard deviation.

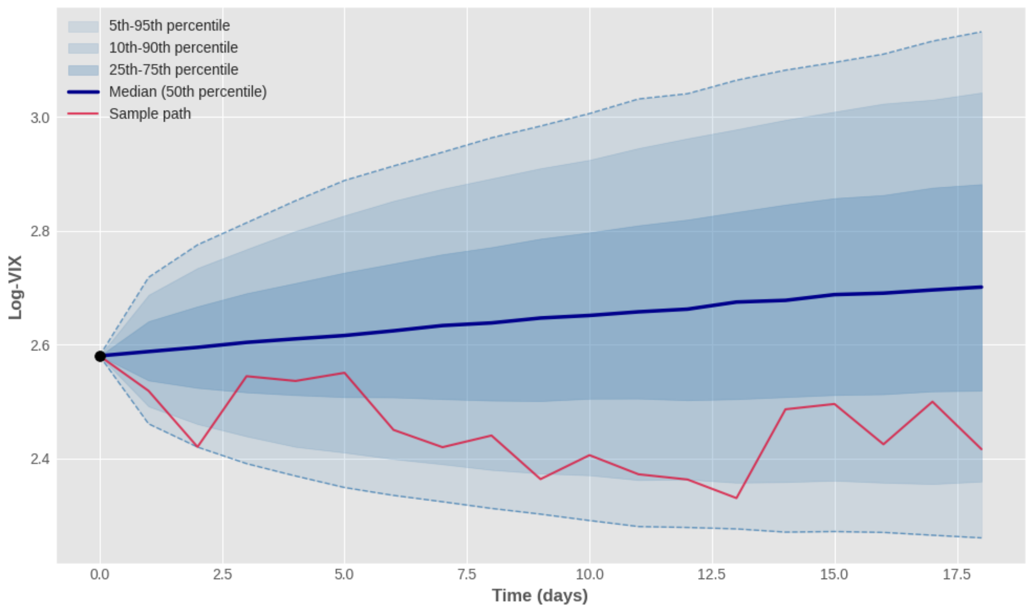


Figure 2.3: VIX Forecast Distribution Portrayed Through a Monte Carlo Simulation Fan Chart

Similarly, for the Monte Carlo solutions, we input the estimated parameters (from the daily time series) into equation 2.19 and simulate 10,000 pathways over the time interval [0,T] where T = 18 days. A fixed timestep of $\delta t = 1$ is employed. The starting value of the VIX is given by $\xi_0 = \ln 13.2 \approx 2.58$ and at each discrete timestep δt, the VIX simulation advances by sampling ϵ_t from a standard normal distribution and applying the Euler-Maruyama discretisation.

Method	$\mathbf{E}[\xi_\mathbf{T}]$	$\boldsymbol{\sigma(\xi_T)}$
Analytical Solutions	2.6979	0.2649
Monte Carlo Solutions	2.7019	0.2682

Table 2.3: VIX Forecast Distribution Comparison

The analytical and Monte Carlo solutions are very similar, validating our results. The $E[\xi_T]$ and $\sigma(\xi_t)$ differ by 0.15% and 1.2% respectively, indicating that 10,000 simulation paths is enough to approximate the true distribution of the VIX at maturity. Both solutions correctly show the log-VIX converging towards the long-term mean $\bar{\xi} = 2.91$, indicating slow upward mean-reversion from the starting value of $\ln 13.2 \approx 2.58$. This can be seen clearly from the fan chart in figure 2.3.

2.2 Portfolio Optimization

In this study, similarly to the Markowitz optimisation explained in section 1.3, we are static hedging. The hedging instruments will consist of cash, 59 calls and 59 puts on the VIX, obtained at 3:45 pm on the 21/06/2024, all with maturity on the 17/07/2024 and a starting value for VIX at 13.20. Each option is finitely available and has a bid- and ask-price quoted in dollars. The risk-free interest rate for both lending and borrowing cash is assumed to be zero. For simplicity, we assume that there are no limits on lending or borrowing cash.

Thus we have a portfolio optimisation problem over 18 days, we will assume the agent has an initial wealth of \$100,000 and 0 liabilities. We want to choose a portfolio of options and cash at time $t = 0$, hold it until time $t = T$ such that we minimise our disutility.

2.2.1 Problem Setup

The hedging problem we are attempting to solve is:

$$\begin{aligned} \min \quad & \mathcal{V}\left(c(\xi_T) - \sum_{j\in J} P^j(\xi_T)x^j\right) \quad \text{over} \quad x \in D \\ \text{subject to} \quad & \sum_{j\in J} S^j(x^j) \leq w, \end{aligned} \tag{P2}$$

where c is a random claim to be hedged, w is a given initial wealth, J is the set of traded assets (with the convention that $0 \in J$ corresponds to the risk-free asset), $S^j(x^j)$ is the cost of buying x^j units of asset j at time t = 0, $D \subseteq \mathbb{R}^J$ is the set of feasible portfolios, $P^j(\xi_T)$ is the payout of one unit asset $j \in J$ at time $t = T$ and $\mathcal{V} : L^0 \mapsto \mathbb{R}$ is a nondecreasing convex utility function in the space of random cash-flows at time $t = T$.

The objective function in equation (2.20) is defined in terms of a function $\mathcal{V}$ on the space L^0 of real valued random variables. The function $\mathcal{V}$ describes the investor's preferences over uncertain expenditure at time $t = T$. In this study, we take

$$\mathcal{V}(c) := \frac{1}{\lambda}\ln\mathbb{E}[\exp(\lambda c/\hat{w})] \tag{2.20}$$

where $\lambda > 0$ is a given "risk aversion" parameter and $\mathbb{E}$ denotes the expectation with respect to the physical probability measure (VIX expectation based on real world probabilities). We set $\hat{w} := 100,000$, this constant simply specifies the units of accounting in the sense that wealth is measured in multiples of \$100,000. The set of feasible portfolios is given by

$$D := \prod_{j\in J}[-q_b^j, q_a^j] \tag{2.21}$$

where q_b^j and q_a^j are the quantities available at the best bid- and the best ask-prices. We assume that the claim c only depends on the value of the VIX at maturity. It follows that all the random quantities in the above problem are uniquely determined by the VIX at maturity.

2.2.2 Resulting Convex Optimization Problem

In order to make the hedging problem (P2) computationally tractable, it must first be transformed into a convex problem. Convex problems have a unique global minimum [17] which can be found efficiently thanks to established reliable optimisation software. In our case we will be using the MOSEK interior point solver, thus we will approximate the objective function by a finite set of convex functions of the portfolio and rewrite the resulting optimisation problem in terms of a finite set of linear inequality constraints.

Following the VIX analysis in section 2.1, we can assume that the distribution of the logarithmic VIX at maturity has probability density function p_{ξ_T} on the real line with expected value $\mu = \mathbb{E}[\xi_T]$ and standard deviation $\sigma = \sigma(\xi_T)$ (See appendix B). Now, the objective function of the portfolio optimisation problem can be rewritten as

$$\mathcal{V}\left(c(\xi_T) - \sum_{j\in J} P^j(\xi_T)x^j\right) = \frac{1}{\lambda}\ln\left(\int_{-\infty}^{\infty} f(x,\xi)p_{\xi_T}(\xi)d\xi\right) \tag{2.22}$$

where

$$f(x,\xi) := \exp\left(\frac{\lambda}{\hat{w}}\left(c(\xi) - \sum_{j\in J} P^j(\xi)x^j\right)\right) \tag{2.23}$$

One can approximate the integral in equation (2.24) by a quadrature of the form

$$\int_{-\infty}^{\infty} f(x,\xi)p_{\xi_T}(\xi)d\xi \approx \sum_{i=1}^{M} f(x,\xi_i)p_{\xi_T}(\xi_i)\Delta\xi_i \tag{2.24}$$

where ξ_i are chosen quadrature points and $\Delta\xi_i = \xi_i - \xi_{i-1}$. In the computation we use the equidistant quadrature points $\{\xi_i\}_{i=0}^{M}$, with $M = 100$, where $\xi_0 = \mu - 9\sigma$ and $\xi_M = \mu + 9\sigma$ (since the quadrature points are equidistant, we have $\Delta\xi_i = 18\sigma/M$).

We now have the objective function:

$$\frac{1}{\lambda}\ln\left(\sum_{i=1}^{M}\exp\left(\frac{\lambda}{\hat{w}}\left(c(\xi_i) - x_0 - \sum_{j\in J/\{0\}} P^j(\xi_i)x^j\right)\right)p_{\xi_T}(\xi_i)\Delta\xi_i\right) \tag{2.25}$$

Looking at the expression (2.25), we see that we see that the expectation in the objective function has now been approximated by a sum of M convex functions. The objective function is now a log-sum-exp (RealSoftMax) function, which is a convex function [19]. Now we reformulate the budget constraint, making sure we have a finite set of linear inequality constraints and we will have created our convex optimization problem.

Let any $j \in J \setminus \{0\}$. The cost functions S^j are given by

$$S^j(x^j) = \begin{cases} p_a^j x^j & \text{if } x^j \geq 0, \\ p_b^j x^j & \text{if } x^j \leq 0, \end{cases} \tag{2.26}$$

where p_b^j and p_a^j are the bid and ask prices of asset $j \in J/\{0\}$ (where we assume that 0 corresponds to the risk-free asset). Such nondifferentiable functions are not supported by common interior point solvers. Thus, we express the trading cost as

$$S^j(x^j) = p_a^j x_+^j - p_b^j x_-^j, \tag{2.27}$$

where x_+^j and x_-^j are the positive and negative parts of x^j. We can thus express the resulting optimization problem in terms of the objective function and three inequality constraints:

$$
\begin{aligned}
\min \quad & \frac{1}{\lambda}\ln\left(\sum_{i=1}^{M} f(x,\xi_i)p_{\xi_T}(\xi_i)\Delta\xi_i\right) && \text{(P3)}\\
\text{subject to} \quad & \sum_{j\in J/\{0\}} (p_a^j x_+^j - p_b^j x_-^j) + x^0 \le w \\
& x^j = x_+^j - x_-^j, \quad \forall j \in J\setminus\{0\} \\
& x_+^j \ge 0, \quad x_-^j \ge 0, \\
& -q_-^j \le x^j \le q_+^j \\
& x_+^j,\, x_-^j \in \mathbb{Z}, && (2.28)
\end{aligned}
$$

where q_-^j and q_+^j are the quantity of bids and asks for asset $j \in J\setminus\{0\}$. The final constraint lets us only trade integer amounts of options, this is a key aspect of the optimization since options, unlike shares, cannot be traded as fractions.

Verification of Convexity

To prove the convexity of the optimisation problem, we must first prove the convexity of the objective function $\mathcal{V}(.)$. From [13], a function f is convex if, for all $x_i \in \mathrm{dom} f$ such that $x_1 < x_2$ and $\alpha \in [0,1]$,

$$
f(\alpha x_1 + (1-\alpha)x_2) \le \alpha f(x_1) + (1-\alpha) f(x_2) \tag{2.29}
$$

Therefore, letting $x_i \in \mathrm{dom}\mathcal{V}$ such that $x_1 < x_2$ and $\alpha \in [0,1]$ and using Hölder's inquality (see appendix B),

$$
\begin{aligned}
\mathcal{V}(\alpha x_1 + (1-\alpha)x_2) =& \frac{1}{\lambda}\ln\mathbb{E}[\exp(\lambda(\alpha x_1 + (1-\alpha)x_2)/\hat{w})] \qquad (2.30)\\
=& \frac{1}{\lambda}\ln\mathbb{E}\left[\exp(\lambda x_1/\hat{w})^{\alpha}\exp(\lambda x_2/\hat{w})^{1-\alpha}\right]\\
\le& \frac{1}{\lambda}\ln\left((\mathbb{E}[\exp(\lambda x_1/\hat{w})])^{\alpha}(\mathbb{E}[\exp(\lambda x_2/\hat{w})])^{1-\alpha}\right)\\
=& \frac{\alpha}{\lambda}\ln(\mathbb{E}[\exp(\lambda x_1/\hat{w}]) + \frac{1-\alpha}{\lambda}\ln(\mathbb{E}[\exp(\lambda x_2/\hat{w})])\\
=& \alpha\mathcal{V}(x_1) + (1-\alpha)\mathcal{V}(x_2)
\end{aligned}
$$

Thus $\mathcal{V}$ is convex.

2.2.3 Results

Risk Aversion	Portfolio Cost ($)	Optimal Portfolio Value
$\lambda = 2$	229,461.76	-1.12677
$\lambda = 20$	177,498.47	-1.02187

Table 2.4: Static Portfolio Optimisation Results

The optimization problem is solved for two distinct risk aversion levels: $\lambda = 2$ representing moderate risk tolerance, and $\lambda = 20$ representing high risk aversion. The economic significance of this lies in understanding how varying the degree of risk aversion influences the choice of portfolio components and the terminal wealth distribution. For lower risk aversion ($\lambda = 2$), we expect more aggressive positioning potentially exploiting high realised volatility, while high risk aversion ($\lambda = 20$), should favour conservative allocations with VIX options potentially serving as portfolio insurance against volatility spikes. The following optimization results provide an insight into these dynamics.

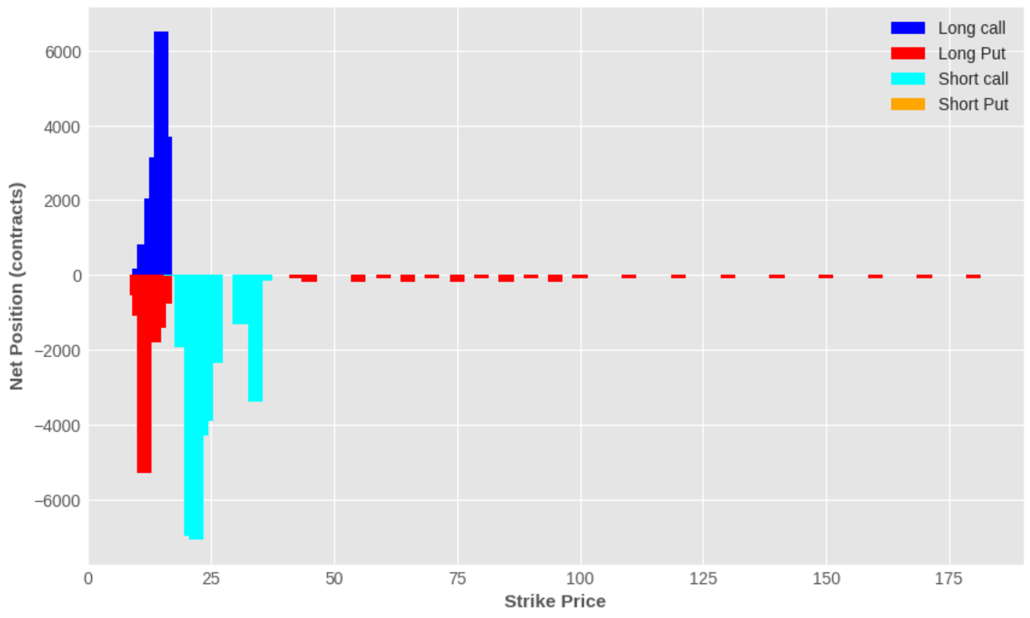


Figure 2.4: Portfolio Allocation By strike price ($\lambda = 2$)

By looking at figures 2.4 and 2.5, we gain some insight into the strategy used by both agents. A straddle-like strategy is used by both agents for near-the-money long calls and puts as well as a general trend of betting on the VIX not reaching levels far above the spot price (short calls and long puts for far above-the-money strikes). Overall, both investment strategies are similar, the essential difference in between both agents is in the amount of options traded and therefore the cost of each portfolio. Indeed, the risk-averse agent has only traded 17067.51 options versus 67159.09 options for the risk prone investor. Hence, the risk prone agent has borrowed \$129461.76 versus only \$77498.47 for the risk averse agent. Overall, the higher leverage reflects the riskier agent's willingness to amplify potential returns by taking more concentrated positions, however, in terms of strategy, they are very similar. This is coherent as the utility function directing their investing is the same with only the coefficient of λ being changed.

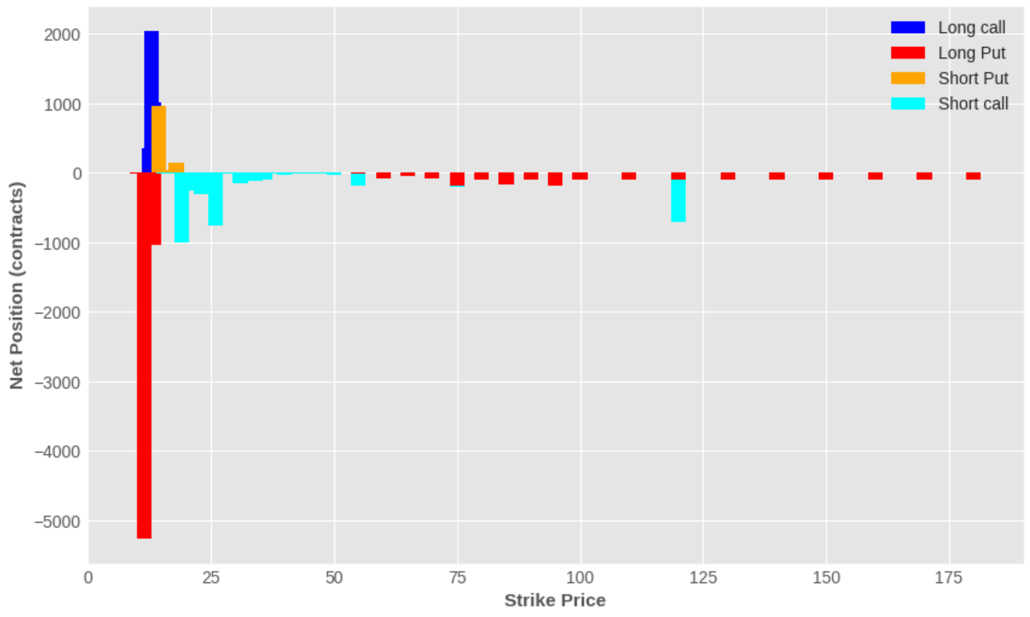


Figure 2.5: Portfolio Allocation by Strike Price ($\lambda = 20$)

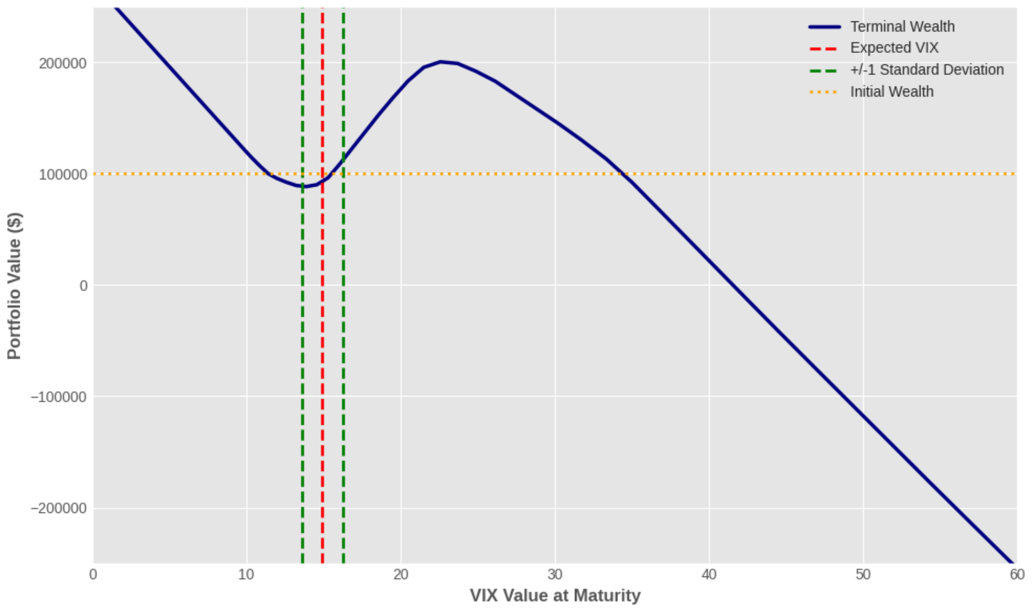


Figure 2.6: Terminal Wealth with Respect to the VIX Value at Maturity ($\lambda = 2$)

Figures 2.6 and 2.7 show the corresponding terminal wealth of both agents with respect to the VIX level at maturity. From these graphs, it is evident that both agents employ almost identical investing strategies, with the key distinction being that the moderately risk-averse agent is prepared to commit more capital to his positions in order to amplify potential returns. The shape of the plot showing the moderately risk averse agent's terminal wealth is the same as that of the highly risk-averse agent except that it has been vertically stretched (greater gains/greater losses). Both portfolios suffer losses at approximately the same VIX levels at maturity, as can be seen in table 2.5.

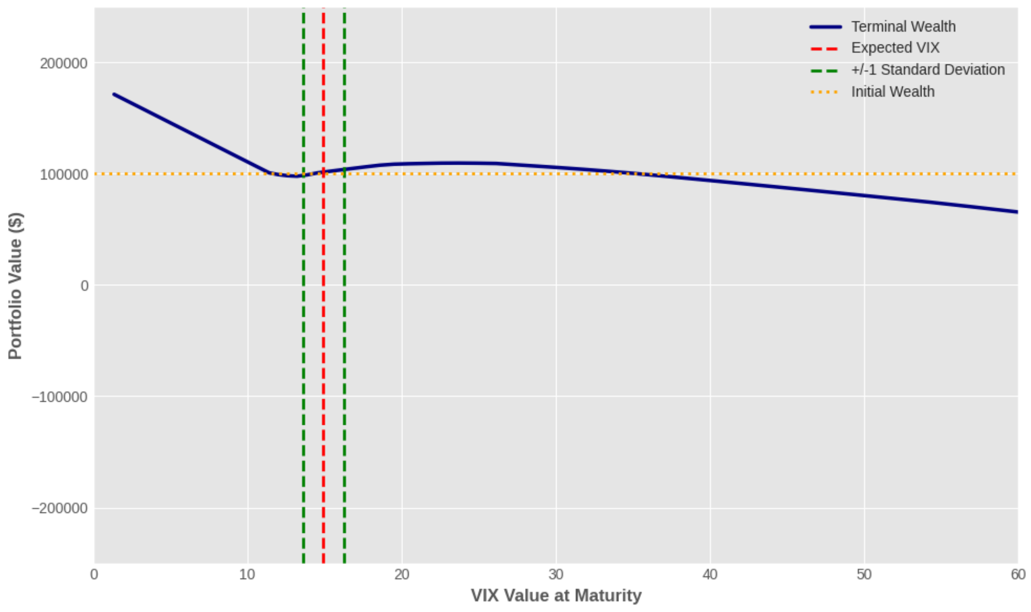


Figure 2.7: Terminal Wealth with Respect to the VIX Value at Maturity ($\lambda = 20$)

Considering the mean-reversion level in our model is approximately 18.4 and the expected VIX level at maturity is approximately 14.9, it may seem counter-intuitive to have said loss region in ones portfolio. However, this is part of the straddle-like strategy used by both agents for near-the-money options. Both agents have chosen to hold positions that profit from more extreme volatility fluctuations while accepting losses in the event of tamer fluctuations.

Agent Type	**Risk Parameter**	**Loss Regions**
Moderately risk-averse	$\lambda = 2$	VIX $\in (11, 16) \cup (34, \infty)$
Highly risk-averse	$\lambda = 20$	VIX $\in (11, 14) \cup (35, \infty)$

Table 2.5: Loss Regions

Figure 2.8 permits to better visualisation the P&L distribution. This KDE plot was created by simulating 10,000 VIX paths and observing the P$L of the portfolios in each case, thus we only see the realistic payouts of the portfolios. The risk-averse investors' P&L is notably more concentrated just ahead of 0 with much tighter tails, especially in the loss direction. This reflects his preference for more predictable outcomes, creating a portfolio which serves primarily as a hedge with limited upside potential but good downside protection, as seen from the statistics in table 2.6.

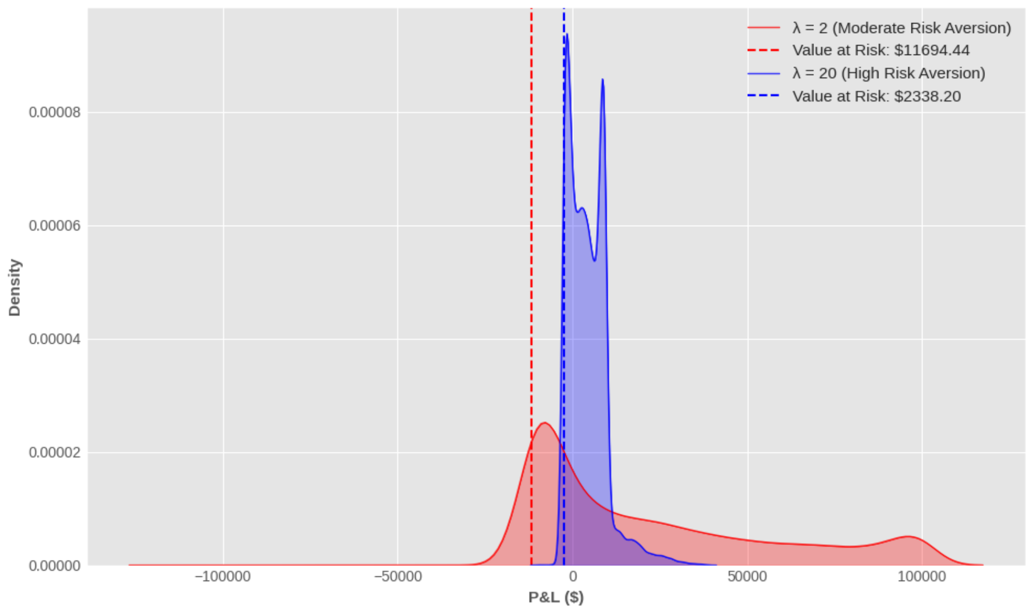


Figure 2.8: P&L Distribution with Risk Metrics (5% VaR)

Conversely, the moderately risk averse investors' P&L distribution shows much fatter tails, particularly in the profit direction. His highly leveraged position means his portfolio is much more sensitive to VIX outcomes, with a P&L spread of almost $200,000. The moderately risk averse investor has a $11,694 value at risk and only a 59.9% profitability rate, suggesting a more speculative investment approach with great upside potential as well as significant downside exposure.

Agent Type	**Profitability Rate**	**5%VaR ($)**	**Range ($)**
Moderately risk-averse	59.9%	11,694	(-110,248, 100,879)
Highly risk-averse	72.16%	2,338	(-9,307, 38,539)

Table 2.6: Risk Metrics

Finally, the real VIX-world value on the close of 17/07/24 was 14.48, thus the moderately risk averse agent would have made a loss of $10,424.99 whilst the highly risk-averse agent would have made a profit of only $462.57. This is an unfortunate outcome for both investors, as their straddle-like strategy did not payoff, however, it does illustrate the advantages of a more risk-averse approach to portfolio optimisation.

2.3 Indifference pricing

Building on the portfolio optimisation framework established in the previous section, we now want to compute the indifference prices for digital VIX options. Whilst the theoretical definitions of the buying and selling indifference prices seen in the literature review provide a clear economic reasoning for the concept of indifference between trading or not trading a claim, they do not provide an easily computable formula. When we want to calculate indifference prices using the standard definitions, we face a significant computational challenge. The indifference selling price of an option $\pi_s(\bar{w}, \bar{c}; c)$ requires us to find the smallest price w such that an agent is indifferent between keeping the option or selling the option at price w.

In order to find said w, from the expression (1.32), the intuitive method would be to implement a line search algorithm that iteratively guesses a price, calculates $\phi(\bar{w} + w, \bar{c} + c)$, compares it with $\phi(\bar{w}, \bar{c})$, checks if the indifference condition is satisfied, and adjusts the price accordingly. This method can be very computationally expensive, especially when extended to multiple options. Therefore, it is useful to derive an alternate formulation of the indifference price that does not require an iterative line search.

The following derivation shows how the indifference buying and selling prices can be reformulated as simple differences of the optimal value function, thereby enabling the efficient computation of the indifference prices:

Starting from the the indifference selling price definition, our utility function and our optimal value function:

$$\pi_s(\bar{w}, \bar{c}; c) = \inf\{w | \phi(\bar{w} + w, \bar{c} + c) \leq \phi(\bar{w}, \bar{c})\} \tag{2.31}$$

$$\mathcal{V}(c) := \frac{1}{\lambda} \ln \mathbb{E}[\exp(\lambda c / \hat{w})] \tag{2.32}$$

$$\phi(w, c) := \inf_{x \in D} \left\{ \mathcal{V} \left(c(\xi_T) - \sum_{j \in J} P^j(\xi_T) x^j \right) \right\} \tag{2.33}$$

we can show that

$$
\begin{aligned}
\pi_s(\bar{w}, \bar{c}; c) &= \hat{w}(\phi(0, \bar{c} + c) - \phi(0, \bar{c})) \\
\pi_l(\bar{w}, \bar{c}; c) &= \bar{w}(\phi(0, \bar{c}) - \phi(0, \bar{c} - c))
\end{aligned} \tag{2.34}
$$

.

Our optimal value function can be rewritten as:

$$
\phi(w, c) = \inf_{x, x_0 \in D} \left\{ \mathcal{V} \left(c(\xi_T) - x_0 - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right) \mid \sum_{j \in J|\{0\}} S^j(x^j) + x_0 \leq w \right\} \tag{2.35}
$$

Let $x_0 = w$ and thus $\sum_{j \in J|\{0\}} S^j(x^j) \leq 0$, thus the optimal value function can be rewritten as:

$$
\phi(w, c) = \inf_{x, x_0 \in D} \left\{ \mathcal{V} \left(c(\xi_T) - w - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right) \mid \sum_{j \in J|\{0\}} S^j(x^j) \leq 0 \right\} \tag{2.36}
$$

Looking at the utility function:

$$
\begin{aligned}
\mathcal{V} \left(c(\xi_T) - w - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right) &= \frac{1}{\lambda} \ln E \exp \left[\left(\frac{\lambda \left(c(\xi_T) - w - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right)}{\hat{w}} \right) \right] \\
&= \frac{1}{\lambda} \ln E \left[\exp \frac{\lambda \left(c(\xi_T) - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right)}{\hat{w}} \cdot \exp \frac{\lambda w}{\hat{w}} \right] \\
&= -\frac{w}{\hat{w}} + \frac{1}{\lambda} \ln E \left[\exp \frac{\lambda \left(c(\xi_T) - \sum_{j \in J|\{0\}} P^j(\xi_T) x^j \right)}{\hat{w}} \right]
\end{aligned} \tag{2.37}
$$

since w is deterministic. Substituting back into the optimal value function:

$$
\begin{aligned}
\phi(w,c) &= \inf_{x,x_0\in D}\left\{-\frac{w}{\hat{w}}+\frac{1}{\lambda}\ln E\left[\exp\frac{\lambda\left(c(\xi_T)-\sum_{j\in J|\{0\}}P^j(\xi_T)x^j\right)}{\hat{w}}\right] \mid \sum_{j\in J|\{0\}}S^j(x^j)\le 0\right\}\\
&= -\frac{w}{\hat{w}}+\inf_{x\in D}\left\{\frac{1}{\lambda}\ln E\left[\exp\frac{\lambda\left(c(\xi_T)-\sum_{j\in J|\{0\}}P^j(\xi_T)x^j\right)}{\hat{w}}\right] \mid \sum_{j\in J|\{0\}}S^j(x^j)\le 0\right\}\\
&= -\frac{w}{\hat{w}}+\phi(0,c) \qquad (2.38)
\end{aligned}
$$

Substituting this formula into the indifference selling price, we get:

$$
\begin{aligned}
\pi_s(\bar{w},\bar{c};c) &= \inf\{w\in\mathbb{R}|\phi(\bar{w}+w,\bar{c}+c)\le\phi(\bar{w},\bar{c})\}\\
&= \inf\{w\in\mathbb{R}|\phi(0,\bar{c}+c)-\frac{\bar{w}+w}{\hat{w}}\le\phi(0,\bar{c})-\frac{\bar{w}}{\hat{w}}\}\\
&= \inf\{w\in\mathbb{R}|\phi(0,\bar{c}+c)-\phi(0,\bar{c})\le\frac{\bar{w}}{\hat{w}}\}\\
&= \hat{w}(\phi(0,\bar{c}+c)-\phi(0,\bar{c}))
\end{aligned}
$$

.

Similarly for the indifference buying price:

$$
\begin{aligned}
\pi_l(\bar{w},\bar{c};c) &= \sup\{w\in\mathbb{R}|\phi(\bar{w}-w,\bar{c}-c)\le\phi(\bar{w},\bar{c})\}\\
&= \sup\{w\in\mathbb{R}|\phi(0,\bar{c}-c)-\frac{\bar{w}-w}{\hat{w}}\le\phi(0,\bar{c})-\frac{w}{\hat{w}}\}\\
&= \sup\{w\in\mathbb{R}|\frac{\bar{w}}{\hat{w}}\le\phi(0,\bar{c})-\phi(0,\bar{c}-c)\}\\
&= \hat{w}(\phi(0,\bar{c})-\phi(0,\bar{c}-c))
\end{aligned}
$$

.

Thanks to this reformulation, we have reduced the number of optimal value calculations to just 12: one for $\phi(0,0)$ plus $\phi(0,c_k)$ for each digital option strike $K=5k$, for $k=0,1,...,10$. This mathematical simplification makes computing the indifference price for multiple digital options computationally feasible, allowing us to analyse how the indifference prices vary with various strike prices.

2.3.1 Results

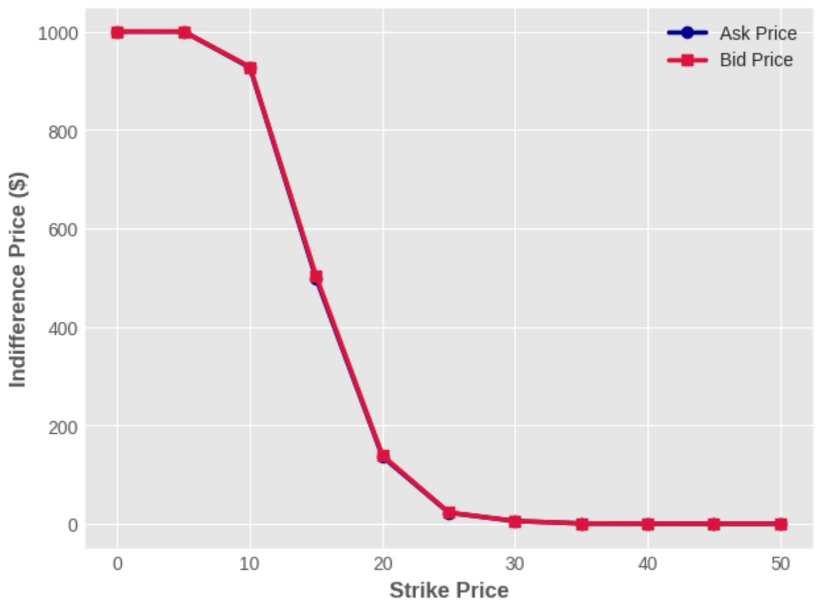


Figure 2.9: Indifference Prices ($\lambda = 2$)

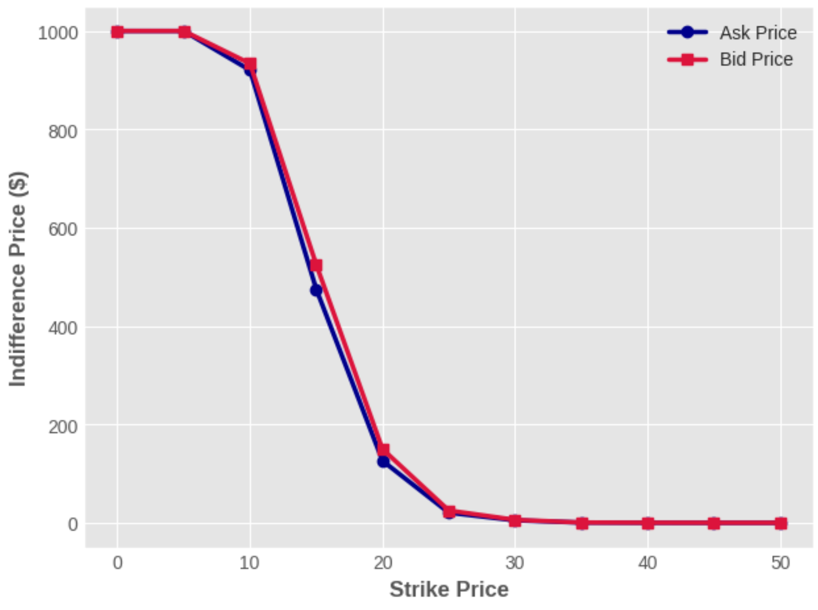


Figure 2.10: Indifference Prices ($\lambda = 20$)

In figures 2.9 and 2.10, the trend in the indifference prices show the steepest decrease happens over the same strike range for both risk-aversion levels, from $K = 10$ to $K = 20$. This range represents a critical transition zone where the digital options go from being in-the-money to out-of-the-money relative to the expected VIX level at maturity $\exp 2.7019 \approx 14.9$ (computed in the Monte Carlo simulation from section 2.1.3). Digital options have a binary payoff (they either pay \$1000 or \$0), this creates an abrupt transition in the payoff at the strike level. For strikes below 10, the payoff is almost certainly \$1000, while for strikes above 20, the payoff is almost certainly \$0, hence the plateau-ing of the price at these strikes. However, within the critical transition zone the probability of the \$1000 payout diminishes rapidly as the strike moves past the expected VIX level at maturity, hence the steep decrease in price in this range. This peculiar characteristic of digital options also explains why the steepest bid-ask spread percentage increase, and percentage price difference between the ask and bid all manifest in the same strike region, see figures 2.11 and 2.12.

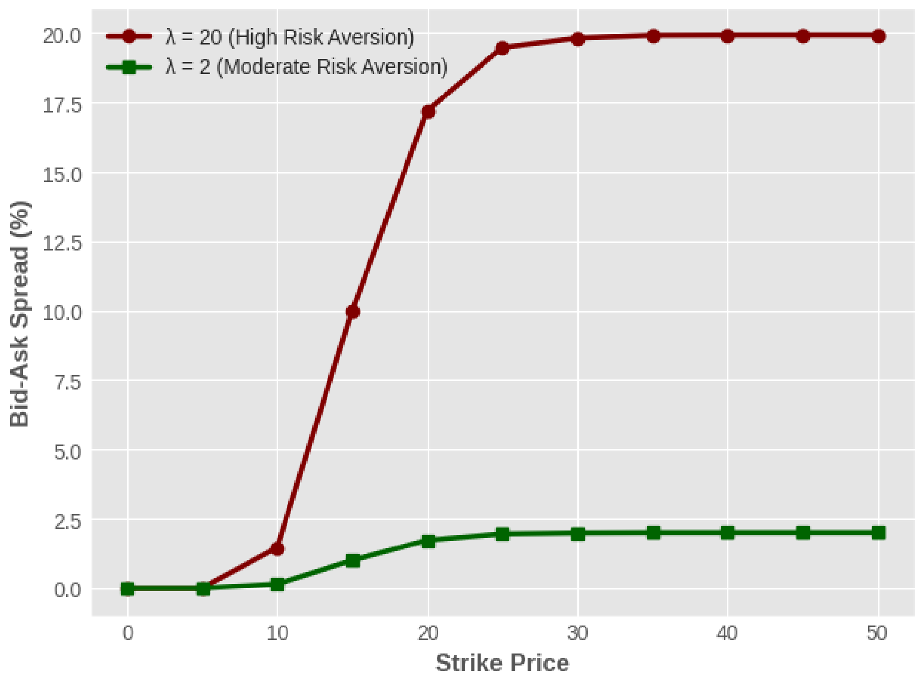


Figure 2.11: Bid-Ask Percentage Spread

The substantial price decay from strike 0 to strike 50 follows the expected convex relationship and selling prices are always greater than or equal to buying prices, for the same level of risk aversion, consistent with the literature covered in section 1. We notice the bid-ask spreads seem remarkably close in table 2.7, however, when looking at figure 2.11, it is clear that the spreads increase as the option is further out-of-the-money. The spread is greater for the more risk-averse agent, leveling just below 20% for high risk-aversion and 2% for moderate risk aversion. This is consistent with general market prices, where the lack of liquidity usually causes a greater spread for far-out-of-the-money options.

Strike	$\lambda = 2$		$\lambda = 20$	
	Selling Price	**Buying Price**	**Selling Price**	**Buying Price**
0	1000.0000	1000.0000	1000.0000	1000.0000
5	999.9694	999.9688	999.9720	999.9658
10	928.0498	926.7029	933.7443	920.2483
15	502.5000	497.5000	524.9584	475.0416
20	138.5157	136.1463	149.7557	126.0395
25	22.8239	22.3820	24.9578	20.5269
30	5.4563	5.3488	5.9767	4.8986
35	0.5294	0.5189	0.5802	0.4751
40	0.1370	0.1343	0.1501	0.1229
45	0.0142	0.0140	0.0156	0.0128
50	0.0027	0.0026	0.0030	0.0024

Table 2.7: Indifference Prices by Risk Aversion Level

The impact of risk-aversion on bid and ask prices can be as seen in figure 2.12, both curves diverge from zero as strike prices increase, the ask prices have an upward divergence which reaches approximately +8.8% versus the bid prices which diverge downwards to approximately −9.2%. Both convergence patterns plateau for further out-of-the-money options. The slight asymmetry in the plateau levels (−9.2% vs +8.8%) indicates that risk aversion has a marginally stronger impact on buying decisions than selling decisions for further out-of-the-money options.

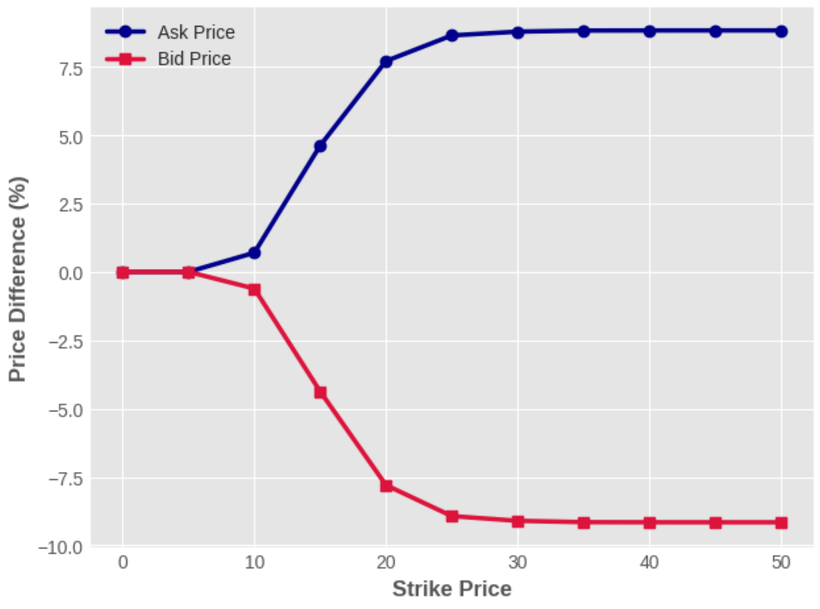


Figure 2.12: Impact of Risk Aversion on Option Pricing ($\lambda = 2$ vs $\lambda = 20$)

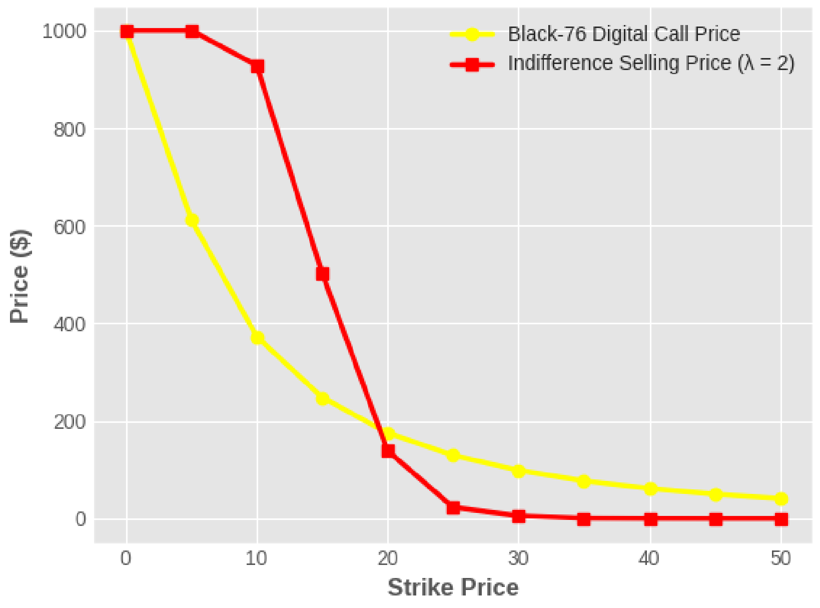


Figure 2.13: Black-Scholes Digital Call Price Versus Indifference Selling Price ($\lambda = 2$)

Finally, figure 2.13, demonstrates one the key advantages of indifference pricing outlined in the literature review. For far-out-of the money options and deep in-the-money options, the Black-Scholes framework often falls apart. As seen in figure 2.13, for strikes $K = 5$ and $K = 50$, the Black-76 prices are \$611.87 and \$40.99 respectively. We can saftely say no trader would buy a digital VIX call option with strike $K = 50$ for \$40.99 in the same way no trader would sell a digital VIX call option with strike $K = 5$ for 611.87 because they would lose money the vast majority of the time. We are able to make this conjecture as we know that even for vastly different risk-aversion levels, the indifference prices do not get close to these values.

Chapter 3

Dynamic Trading using Neural Networks

3.1 Dynamic Portfolio Optimisation

As mentioned in the literature review, machine learning applications in finance have provided a plethora of new approaches for tackling portfolio optimisation problems. In chapter two, we are attempting to solve a static portfolio optimisation problem. One of the key limitations of this setup is that a single investment decision is made at time $t = 0$ and no rebalancing of the portfolio can be made as market conditions change. In this section we will attempt to produce a simplified neural-network based dynamic trading strategy that can adaptively change its positions in response to evolving market conditions.

3.1.1 Problem Setup

For our trading setup, we will allow the agent to trade VIX futures, assuming the logarithm of VIX future prices follows the Ornstein-Ulenbeck process from section 2.1 with the estimated parameters in table 2.2. The initial VIX futures price will be $F_0 = 13.2$. The agent has a daily investment budget of \$5556 (\$100,000/18 days), with the neural network dynamically determining optimal position sizes within this budget based on market information and trying to maximise his risk-adjusted total returns according to the specified utility function. The agent invests and liquidates daily, thus his total P&L is calculated from:

$$\text{P\&L} = \sum_{i=0}^{18} W_i \frac{F_i - F_{i-1}}{F_{i-1}}. \tag{3.1}$$

where F_i is the futures price on day i and W_i is the value of his investment on day i.

We will investigate the effectiveness of neural network in producing trading strategies according the following utility functions:

$$\text{Modified Sharpe Ratio:} \quad \frac{\mathbb{E}[\text{P\&}L]}{\sigma(\text{P\&L})} \tag{3.2}$$

$$\text{Entropic Risk Function:} \quad \mathcal{V}(c) := \frac{1}{\lambda} \ln \mathbb{E}[\exp(\frac{\lambda \times \text{P\&L}}{\hat{w}})] \tag{3.3}$$

The modified Sharpe ratio is an effective utility function as it is numerically stable and captures the fundamental tradeoff between risk and returns. The entropic risk function here is the same as that from the static portfolio optimisation, this time with $\hat{w} = 1000$ as the units of accounting. In the entropic risk function, we will use: $\lambda = 2$ to create a strategy representing a moderately risk-averse investor.

Information Set

The neural network processes a 3-dimensional feature vector at each time step.

1. Time component: Time index (current day)
2. Price component: Standardised current futures price
3. Price momentum: Return from previous day

This provides the neural network with temporal, level and momentum information which will help it to make decisions on position sizes. To obtain the information set, 10,000 Monte Carlo simluations were performed using the VIX futures model, the components were then calculated, resulting in a matrix $\mathcal{X} \in \mathbb{R}^{10000\times 18\times 3}$ where:

$$\mathcal{X}_{n,t,f} \quad \begin{cases} n = 1, \ldots, 10000 & \text{(MC path)} \\ t = 1, \ldots, 18 & \text{(Time index)} \\ f \in \{1, 2, 3\} & \text{(Feature index)} \end{cases}$$

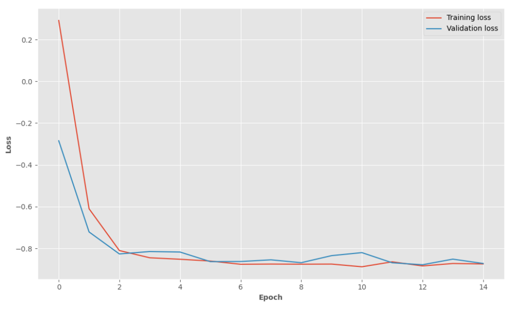
$$\begin{aligned} \mathcal{X}_{n,t,1} &= \frac{\text{Time Horizon}}{\text{Time Steps}}, \\ \mathcal{X}_{n,t,2} &= \frac{F_t^{(n)} - \mathbb{E}[F_T]}{\sigma(F_T)}, \\ \mathcal{X}_{n,t,3} &= \frac{F_t^{(n)} - F_{t-1}^{(n)}}{F_{t-1}^{(n)}} \quad (t \geq 1). \end{aligned}$$

3.1.2 Methodology

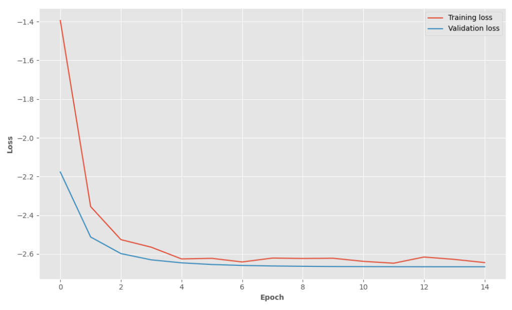
Constructing the neural network required testing multiple different model architectures and evaluating them across multiple different metrics. Firstly, by monitoring the training and validation loss and secondly by examining the performance metrics of the dynamic trading strategy produced by the neural network.

One of the prevalent issues when training the neural network was overfitting. The validation loss would plateau long before the training loss. This issue was overcome through the implementation of L2 regularisers, dropouts and batch normalisation.

When building the network, one of the immediate questions posed was whether or not to use a recurrent neural network to process the sequential data whilst maintaining a memory of previous inputs. After testing a recurrent neural network with Long Short-Term Memory (LSTM) layers, it was concluded that the advantages gained from the network learning temporal patterns were overshadowed by the substantially increased epoch training time. To avoid the computational complexity of using a recurrent neural network, a simple feedforward neural network was used with price momentum incorporated as a feature in the input feature matrix. 1-day, 2-day, and 3-day price momentum features were attempted, however, in the short 18-day time-frame, only the 1-day price momentum produced better results.



(a) Sharpe Ratio



(b) Entropic Risk Function ($\lambda = 2$)

Figure 3.1: Training and Validation Loss

For the training setup, the models were trained for 15 epochs with batches of 32 samples each, using 80% of the data for training and 20% for validation. As can be seen in figure 3.1, for both loss functions the training and validation loss converge to a plateau.

3.1.3 Model Architecture

Our objective is to obtain a function $f : \mathbb{R}^{18\times 3} \to \mathbb{R}^{18}$ that maps the three-dimensional temporal feature space across 18 timesteps to position size predictions for each trading day.

Our function f is a feedforward neural network that processes the entire temporal sequence simultaneously. However, the dense transformations are applied independently to each timestep to preserve the desired sequential structure of the input data. The network contains four hidden layers with 64, 128, 64, and 32 neurons respectively, followed by a final output layer:

$$f = L_6 \circ \sigma_5 \circ L_5 \circ \sigma_4 \circ L_4 \circ \sigma_3 \circ L_3 \circ \sigma_2 \circ L_2 \tag{3.4}$$

where $\sigma_i : \mathbb{R}^{d_i} \mapsto \mathbb{R}^{d_i}$ are activation functions and $L_i : \mathbb{R}^{d_{i-1}} \mapsto \mathbb{R}^{d_i}$ is an affine function of the form

$$L_i(x) = W_i x + b_i, \tag{3.5}$$

parametrized by a weight matrix $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ and a bias vector $b_i \in \mathbb{R}^{d_i}$. Following notational framework in [14], this architecture can be formally represented as:

$$f \in \mathcal{F}_{TD}(3, 64, 128, 64, 32, 1; \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6) \tag{3.6}$$

where the subscript TD denotes that dense layers are applied in a time-distributed manner across the temporal dimension. The activation for the first hidden layer is $\sigma_2(z) = z$, followed by ReLU activation functions $\sigma_i(z) = \max\{0, z\}$ for layers 3 to 5, and a hyperbolic tangent activation for the output layer $\sigma_6(z) = \tanh(z)$. A dropout regularisation is added to the fourth and fifth layers to combat overfitting, with dropout probabilities of 0.2 and 0.1 respectively. These randomly set a fraction of the neurons to zero for each forward pass, encouraging the network to learn more robust representations and prevent over-reliance on specific neurons [14].

Layer normalization is applied to the input sequences to stabilise training by normalising across the feature dimension:

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{3.7}$$

where μ and σ^2 are the mean and variance computed across features for each timestep, and $\gamma, \beta \in \mathbb{R}^3$ are learnable scale and shift parameters [16]. Batch normalisation is applied after the first dense layer to stabilise training by normalising the inputs of each mini-batch:

$$\text{BatchNorm}(x) = \gamma \odot \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{3.8}$$

where μ_B and σ_B^2 are the mean and variance computed across the mini-batch and $\gamma, \beta \in \mathbb{R}^3$ are learnable scale and shift parameters [15].

L2 regularisation or ridge regression is a regularisation method that penalises high-value coefficients. L2 regularisation adds the squared magnitude of the coefficient [12]:

$$\mathcal{L}_{\text{reg}} = \mathcal{L}_{\text{original}} + \lambda \sum_i \|W_i\|_F^2, \tag{3.9}$$

where $\|W_i\|_F^2$ denotes the Frobenius norm of the weight matrix in layer i. This penalty diminishes large coefficients and thereby prevents the network from overfitting the data.

The use of the hyperbolic tangent activation function in the output layer:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.10}$$

constrains position predictions to the interval $(-1, 1)$. This bounded output is useful as it directly gives us the fraction of the daily budget to allocate to the position. The value of an investment on day i is thus equal to:

$$W_i = f_i \times B_{day} \tag{3.11}$$

where $f_i \in (-1, 1)$ is the neural network output for day i, and B_{day} is the daily budget (\$5556). Noting that positive values of f_i indicate a long position while negative values indicate short positions.

Finally, the model is trained using the Adam optimiser (See appendix B)

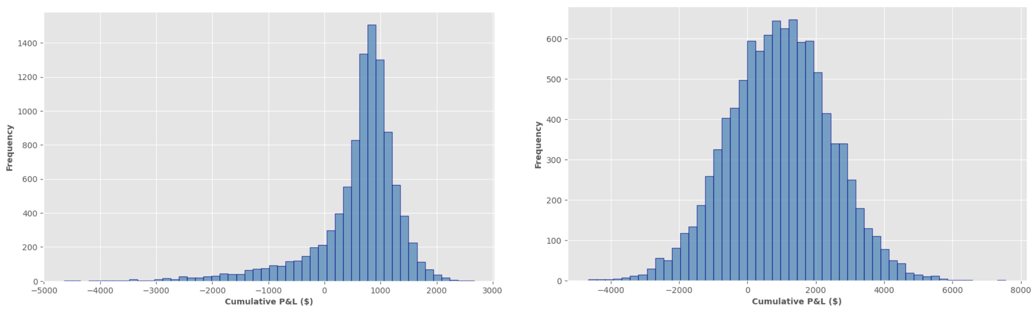
3.1.4 Results

To evaluate both trading strategies, their performance was recorded against the training set as well as a test set. The test set was created by performing another Monte Carlo simulation of the discrete Ornstein-Ulhenbeck process and calculating the new feature matrix.

Metric	**Sharpe Loss**	**ERF Loss**
P&L mean (\$)	608.44	942.62
P&L std (\$)	771.51	1489.56
P&L range (\$)	[-4644.54, 2670.24]	[-4657.78, 7563.55]
Modified Sharpe ratio mean	0.79	0.63
Positive P&L ratio	85.80%	73.66%
Mean absolute position (contracts)	218.43	404.39
Mean value of daily investment (\$)	2666.01	5555.53
Value at Risk (5%)	902.85	1104.04

Table 3.1: Dynamic Trading Strategy Training Set Performance

These results highlight clear differences in the dynamic strategies produced by both neural networks, suggesting the loss functions have effectively dictated their respective dynamic trading strategies. Under the modified Sharpe ratio loss, the strategy delivered more consistent risk-adjusted returns than under the entropic risk function (ERF). A mean cumulative P&L of \$608 with a standard deviation of \$772 versus a mean cumulative P&L of \$943 with a standard deviation of \$1490 shows the Sharpe loss driven strategy was half as volatile as the ERF driven strategy, whilst only sacrificing $\sim 30\%$ less in expected returns. Looking at the histograms in 3.2, it is apparent that under the Sharpe loss, the cumulative P&L distribution is positively-skewed of zero compared to the ERF strategy, which shows a more symmetric distribution. This is reflected in the percentage of positive P&L outcomes, where for the Sharpe loss 85.80% of paths result in positive outcomes, versus only 73.76% for the ERF loss. Interestingly, although the sharpe loss distribution is more positively-skewed, it exhibits a more pronounced left-tail than the ERF loss, who conversely has a right-tail. In fact, the range of outcomes for both strategies has approximately the same lower bound $\sim$ \$4650 and a similar 5% value at risk $\sim$ \$1000. This is surprising considering the ERF strategy demonstrates significantly more agressive positioning, with a mean absolute position of 404.39 contracts versus 218.43. Finally, we note the ERF strategy has substantially more upside potential than the Sharpe strategy, with an upper bound of 7563.55 versus 2670.24 respectively.



(a) Sharpe Ratio (b) Entropic Risk Function ($\lambda = 2$)

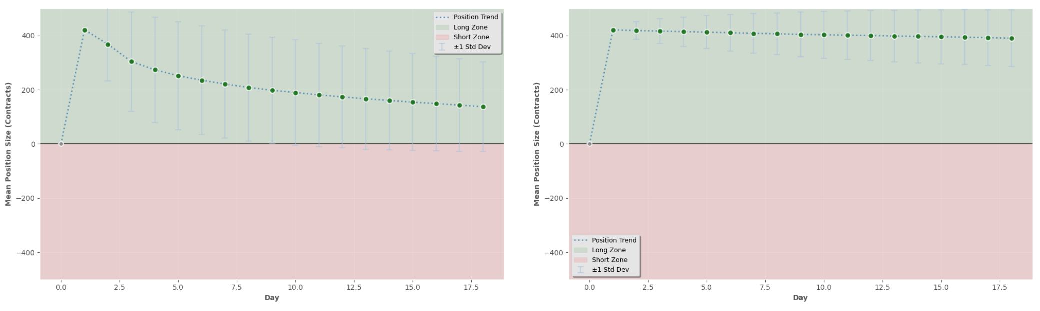
Figure 3.2: Training Set P&L

The results for the training set are very much consistent with the intention for both strategies. The dynamic of the Sharpe loss strategy is to get as greater expected return relative to variance, although the modified Sharpe ratio was only 0.79, it was better than that of the ERF 0.63. Meanwhile, the exponential in the expectation of the ERF penalises downside risk heavily, this was achieved as the lower bound of its P&L and value at risk were better than that of the Sharpe loss, relative to the investment made over 18 days.

Metric	Sharpe Loss	ERF Loss
P&L mean ($)	537.11	919.07
P&L std ($)	1351.19	1750.05
P&L range ($)	[-3730.75, 12610.57]	[-4082.50, 15296.34]
Modified Sharpe ratio mean	0.40	0.53
Positive P&L ratio	63.63%	67.57%
Mean absolute position (contracts)	250.66	404.39
Mean value of daily investment ($)	3534.92	5786.81
Value at Risk (5%)	1248.12	3984.42

Table 3.2: Dynamic Trading Strategy Test Set Performance

The test set results revealed more details about the trading strategies generated by the neural networks. The Sharpe loss driven strategy seems to be significantly less robust than the ERF loss driven strategy. Although its average profit increased, the ratio of scenarios with a positive P&L decreased to 63.63%, suffering a 26% decrease compared to only 11% for the ERF. The test set showed an overall increase in the variance exhibited both strategies, especially the Sharpe loss driven strategy where the upper bound increased by a factor of five. This reveals the neural networks may be overfitting the training set, thus leading to greater variance for results yielded from the test set.



(a) Sharpe Ratio (b) Entropic Risk Function ($\lambda = 2$)

Figure 3.3: Test Set Average Position at Each Timestep

By looking at figure 3.3, it is evident that both strategies are taking advantage of the initial VIX value being significantly below its mean reversion level. Both strategies rely on the VIXs reversion by taking long positions throughout the trading time frame with the difference between both strategies being the Sharpe loss strategy gradually reduces its position size as the VIX gradually gets closer to its mean-reversion level.

Chapter 4

Conclusion

This report offers valuable insight into VIX-linked decision-making and risk management applications over an 18-day horizon. Calibrating the log-VIX to an OU process using daily observations proved to be adequate for short-term forecasts. This finding suggests that volatility forecasting benefits from including short-term fluctuations, especially a short time period. The advantages of using analytically tractable models was also displayed as we were able to confirm our Monte Carlo simulation results thanks to the closed form solution of the continuous Ornstein-Uhlenbeck model.

The static portfolio optimisation revealed key insights into the effect of risk preferences on volatility trading strategies. While both levels of risk-aversion had structurally similar approaches, the difference was in their willingness to leverage their position, as the less risk-averse agent borrowed almost twice as much as the other. By observing the investment allocation of both investors and the profile of the terminal wealth relative to the VIX at maturity, we were able to see that the optimal strategies gravitated towards positions that profit from market movement (straddling) rather than opting for conservative positioning around the mean-reversion level of the VIX. This highlights a an important aspect of volatility trading where real-world traders usually accept small losses in exchange for occasional large gains.

The indifference pricing of the digital options revealed some critical limitations of traditional pricing methods. We observed a widening of the bid-ask spread as the options were further out-of-the-money, especially for higher risk-aversion. These results revealed how the subjective risk preferences can recreate natural market microstructure, where it is common for far-out-of the-money options to have a wider bid-ask spread. Furthermore, the results showed that indifference pricing provides more economically sensible valuations than the Black-Scholes framework, especially for extreme strikes where the Black-Scholes price was unreasonably low for far into-the-money options and overpriced for far out-of-the-money options.

The dynamic trading strategies produced by the neural networks highlighted both the advantages and downsides of leveraging machine learning for dynamic trading strategies. The research revealed that the objective function choice changes the output strategy as well as the network architecture necessary to obtain results. We found that in our problem setup, both strategies seem to rely heavily on the mean-reversion of the VIX, possibly making them inapplicable for scenarios where F_0 is closer to the mean-reversion level. More importantly, the results from the dynamic trading strategy admitted a clear overfitting issue, the strategy produced by both neural networks produced significantly better results for the training set than the test set. These findings highlight the importance of combating overfitting through a heavier regularisation and a larger training set. Finally, it is important to note that this dynamic trading strategy was produced in a highly simplified environment with no trading fees or spreads meaning its applicability to real market scenarios is not yet valid.

Appendix A

Theorems

Theorem 1: Ito's Lemma

For a process X satisfying the SDE:

$$dX_t = b(X_t)dt + \sigma(X_t)dW_t.$$

and a function $f(x,t)$ which is twice differentiable in x and once in t, we have:

$$\begin{aligned} f(X_t,t) &= f(X_0,0) + \int_0^t f_x(X_s,s)\,dX_s + \int_0^t \Big[f_t(X_s,s) + \tfrac{1}{2}f_{xx}(X_s,s)\sigma(X_s)^2\Big]ds \\ &= f(X_0,0) + \int_0^t f_x(X_s,s)\big[b(X_s)ds + \sigma(X_s)dW_s\big] \\ &\quad + \int_0^t \Big[f_t(X_s,s) + \tfrac{1}{2}f_{xx}(X_s,s)\sigma(X_s)^2\Big]ds. \end{aligned}$$

As shorthand, we write the latter in differential form as:

$$df(X_t,t) = f_t(X_t,t)\,dt + f_x(X_t,t)\,dX_t + \tfrac{1}{2}f_{xx}(X_t,t)\sigma(X_t)^2\,dt.$$

Theorem 2: The Feynman-Kac formula

The option price $C(S,t)$ is the discounted expected value of $f(S_t)$ (the payout of the option at time T under the Black-Scholes model) under the risk-neutral-measure $\mathbb{Q}$ where the drift is r not μ.

$$C(S,t) = e^{r(T-t)}\mathbb{E}[f(S_T)|S_t = S].$$

Theorem 3: Solution to linear least-squares regression:

Assuming that rank($\mathcal{X} = d+1$). Then, the RSS is minimised for:

$$\hat{\beta} = (\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T\mathcal{Y}$$

.

Appendix B

Definitions

Definition 1: Lognormal Distribution

A random variable X is said to follow a lognormal distribution if the natural logarithm of X is normally distributed. That is,

$$X \sim \mathrm{LogNormal}(\mu, \sigma^2) \quad \Longleftrightarrow \quad \ln(X) \sim \mathcal{N}(\mu, \sigma^2).$$

For $x > 0$, the probability density function is given by

$$f(x; \mu, \sigma) = \frac{1}{x\,\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right).$$

Definition 2: ADAM Optimiser

ADAM (Adaptive Moment Estimation) is a mini-batch stochastic gradient descent variant. Mini-batch stochastic gradient descent is summarised in the following algorithm [14]:

Algorithm 1 Mini-batch stochastic gradient descent

```
Require: $f,\ x_0 \in \mathbb{R}^n,\ K \in \mathbb{N},\ b \in \{1, \dots, N\}$
  while $k < K$ do
      Sample (uniformly among sets of size $b$) $\mathcal{B}_k \subset \{1, \dots, N\}$
      Choose $\gamma_k > 0$ and perform
                  $x_{k+1} = x_k - \gamma_k \nabla f_{\mathcal{B}_k}(x_k)$
      $k = k + 1$
  end while
  return $x_K$
```

Definition 3: Hölder's inquality

For the probability space $(\Omega, \mathcal{F}, \mathbb{P})$. For real or complex valued random variables X and Y on Ω, Hölder's inequality reads

$$\mathbb{E}[|XY|] \;\leq\; (\mathbb{E}[|X|^p])^{\frac{1}{p}} \, (\mathbb{E}[|Y|^q])^{\frac{1}{q}} \, , \tag{B.1}$$

where $p, q > 1$ and $\frac{1}{p} + \frac{1}{q} = 1$ [2].

Appendix C

Part 2 code

```
!pip install mosek
%pip install cvxpy[MOSEK]
import cvxpy as cp

from google.colab import files
uploaded = files.upload()
import os
```

Show hidden output

```
os.makedirs('/root/mosek', exist_ok=True)
!mv mosek.lic /root/mosek/mosek.lic
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import scipy.stats
import seaborn as sns
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.tsa.stattools import adfuller

import torch
import torch.nn as nn
import torch.nn.functional as F
from sklearn.model_selection import train_test_split

from jax import grad

plt.style.use('ggplot')
```

```
np.set_printoptions(suppress=True)
np.set_printoptions(precision=5)
```

Task 1

```
#VIX data analysis

VIXopeningprices = pd.read_csv('VIX-daily-opening-prices.csv')
X = VIXopeningprices.iloc[:,0]

logVIX = np.log(VIXopeningprices.iloc[:,1])
logVIXshifted = logVIX.shift(-1)
logVIXdifferences = np.diff(logVIX)

result = adfuller(logVIX)
print('ADF Statistic:', result[0])
print('p-value:', result[1])
print('Critical Values:', np.array(result[4]))

X_dates = pd.to_datetime(X)
plt.figure(figsize=(10, 4))
plt.plot(X_dates, logVIX, linewidth=1.5)
plt.xlim(pd.Timestamp('1990-02-01'), pd.Timestamp('2024-11-04'))
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.xlabel('Time (years)', fontsize=12, fontweight='bold')
plt.ylabel('log-VIX', fontsize=12, fontweight='bold')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
plot_acf(np.array(logVIX), lags=30, color='black', linewidth=2.5, markersize=4,
plt.xlabel('Lag (days)', fontsize=12, fontweight='bold')
plt.ylabel('Autocorrelation', fontsize=12, fontweight='bold')
plt.ylim(-0.1, 1.1)
plt.tight_layout()
plt.show()

plt.figure(figsize=(8, 4))
plot_acf(np.array(logVIXdifferences), lags=30, color='black')
plt.ylim(-0.2, 0.2)
plt.title('Autocorrelation of VIX Differences', fontsize=14)
plt.tight_layout()
plt.show()
```

Show hidden output

```
#Model 1 - Simple linear least squares regression
# Linear y = m*x + b
# Matrix d = G @ m
#Therefore

G = np.stack((logVIX[:-1],np.ones(logVIX[:-1].shape)),axis=1)
d = logVIXshifted[:-1]

print(f'Daily Time Series:')

print(G.shape)
print(d.shape)

m, RSS, rank, s = np.linalg.lstsq(G, d)
print(m.T)

print(f'Coefffieients: {m}')
print(f'Residual sum of squares: {RSS}')
print(f'MSE = {RSS/len(d)}')
print(f'rank: {rank}')
print(f'Singular values: {s}')
print('||')
#19day interval

G18 = np.stack((logVIX18day[:-1],np.ones(logVIX18dayshifted[:-1].shape)),axis=1)
d18 = logVIX18dayshifted[:-1]

print(f'18 Day Interval Time Series:')

print(G18.shape)
print(d18.shape)

m18, RSS18, rank18, s18 = np.linalg.lstsq(G18, d18)

print(f'Coefffieients: {m18}')
print(f'Residual sum of squares: {RSS18}')
print(f'MSE = {RSS18/len(d18)}')
print(f'rank: {rank18}')
print(f'Singular values: {s18}')
```

Show hidden output

```
#daily interval

xpred = np.linspace(0,5)
gpred = np.stack((xpred,np.ones(xpred.shape)), axis=1)
ypred = gpred @ m

plt.scatter(logVIX[:-1],logVIXshifted[:-1])
plt.plot(xpred,ypred)
plt.grid()
plt.show()

#19 day interval

xpred18 = np.linspace(0,5)
gpred18 = np.stack((xpred18,np.ones(xpred18.shape)), axis=1)
ypred18 = gpred18 @ m18

print(gpred18.shape)
print(ypred18.shape)

plt.scatter(logVIX18day[:-1],logVIX18dayshifted[:-1])
plt.plot(xpred18,ypred18)
plt.grid()
plt.show()
```

Show hidden output

```
beta = m[0]
```

```
    alpha = m[1]
    print(alpha, beta)

    vol = np.sqrt((RSS)/(8791-2)).item()

    a = 1 - beta
    Vmean = alpha/a


    def OU_prediction(V0, timesteps,Vmean,vol,a,paths):
      dt = 1
      V = np.zeros((paths, timesteps+1))
      V[:,0] = V0
      np.random.seed(42)
      DW = np.random.normal(0,1, size=(paths,timesteps))
      for i in range(timesteps):
        V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
      return V

    predicted_logV = OU_prediction(np.log(13.2), 18, Vmean, vol, a, 10000)
    newvol = predicted_logV[:,-1].std()

    X = np.linspace(0, 18, 19)
    plt.plot(X, np.exp(predicted_logV[0,:]))
    plt.xlabel('Time')
    plt.ylabel('VIX')
    plt.grid()
    plt.show()

    print(np.mean(predicted_logV[:,-1]))
    print('Daily time intervals:')
    print(f'The speed of mean reversion a={a}')
    print(f'The mean-reversion level {Vmean}')
    print(f'The volatility of the process is {vol}')
    print(f'The 18-day volatility of the process is {vol*np.sqrt(18)}')
    print(f'The recorded volatility is: {newvol}')
    print(f'The condition number is: {s[0]/s[1]}')
    print(f'The half-life of volatility shocks = {np.log(2)/-np.log(beta)}')


    beta18 = m18[0]
    alpha18 = m18[1]

    vol18 = np.sqrt((RSS18)/(462-2)).item()

    a18 = 1 - beta18
    Vmean18 = alpha18/a18

    print('||')
    print(f'18 day time intervals:')
    print(f'The speed of mean reversion a={a18}')
    print(f'The mean-reversion level {Vmean18}')
    print(f'The volatility of the process is {vol18}')
```

```
predicted_logV18 = OU_prediction(np.log(13.2), 18, Vmean, vol, a, 10000)
newvol18 = predicted_logV18[:,-1].std()

print(f'The recorded volatility is: {newvol18}')
print(f'The condition number is: {s18[0]/s18[1]}')
print(f'The half-life of volatility shocks = {7 * np.log(2)/-np.log(beta18)}')
```

Show hidden output

```
def simulating_OU(V0, timesteps,Vmean,vol,a,paths):
  dt = 1
  V = np.zeros((paths, timesteps+1))
  V[:,0] = V0
  np.random.seed(42)
  DW = np.random.normal(0,1, size=(paths,timesteps))
  for i in range(timesteps):
    V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
  return V

maturityvix = simulating_OU(np.log(13.2), 18,Vmean,vol,a,10000)[:,-1]
print(maturityvix)
print(np.mean(maturityvix))
print(np.std(maturityvix))
```

Show hidden output

Task 2

```
def meanandstd(startingvalue, longtermmean, t, T):
  mean = longtermmean + (startingvalue - longtermmean)*np.exp(-a*(T-t))
  variance = (vol**2/(2*a))*(1-np.exp(-2*a*(T-t)))
  return mean, np.sqrt(variance)

startingvalue = np.log(13.20)
longtermmean = Vmean
t = 0
T = 18

expectationA, stdA = meanandstd(startingvalue, longtermmean, t , T)

print('Daily Time Series:')
print(f'Analytical Expected logVIX at maturity: {expectationA}')
print(f'Analytical standard deviation of VIX at maturity: {stdA}')
print(f'Variance: {stdA**2}')
expectation19, std19 = meanandstd(startingvalue, Vmean18, t , T)

print('19 Day Interval Time Series:')
print(f'Analytical Expected logVIX at maturity: {expectation19}')
print(f'Analytical standard deviation of VIX at maturity: {std19}')
print(f'Variance: {std19**2}')
```

Show hidden output

```
VIXmedian = np.exp(expectationA)
VIXmedian19 = np.exp(expectation19)
print('Daily interval time series')
print(VIXmedian)
print('19 day interval time series')
print(VIXmedian19)
```

Show hidden output

Task 3 and 4:

```
vixoptionsdata = pd.read_csv('VIX_Data_MScFM_project.csv')

strikes = vixoptionsdata['strike'].tolist()
option_types = vixoptionsdata['option_type'].tolist()
bid_sizes = vixoptionsdata['bid_size_1545'].tolist()
bid_prices = vixoptionsdata['bid_1545'].tolist()
ask_sizes = vixoptionsdata['ask_size_1545'].tolist()
ask_prices = vixoptionsdata['ask_1545'].tolist()

print(" Strike Prices :", strikes[:5])
print(" Option Types:", option_types[:5])
print("Bid Sizes:", bid_sizes[:5])
print("Bid Prices :", bid_prices[:5])
print("Ask Sizes:", ask_sizes[:5])
print("Ask Prices :", ask_prices[:5])
```

Show hidden output

```
def simulating_OU(V0, timesteps,Vmean,vol,a,paths):
  dt = 1
  V = np.zeros((paths, timesteps+1))
  V[:,0] = V0
  np.random.seed(42)
  DW = np.random.normal(0,1, size=(paths,timesteps))
  for i in range(timesteps):
    V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
  return V

futurelogvix = simulating_OU(np.log(13.2),18,Vmean,vol,a,10000)
```

```
std = np.std(futurelogvix[:,-1])

print(np.exp(expectedfuturelogvix))
print(std)

plusstd = expectedfuturelogvix + std
minusstd = expectedfuturelogvix - std

import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 18, 19)
upper = np.percentile(futurelogvix, 95, axis=0)
middle = np.percentile(futurelogvix, 50, axis=0)
lower = np.percentile(futurelogvix, 5, axis=0)
p90 = np.percentile(futurelogvix, 90, axis=0)
p75 = np.percentile(futurelogvix, 75, axis=0)
p25 = np.percentile(futurelogvix, 25, axis=0)
p10 = np.percentile(futurelogvix, 10, axis=0)
plt.figure(figsize=(10, 6))
plt.fill_between(t, lower, upper, alpha=0.15, color='steelblue', label='5th-95th
plt.fill_between(t, p10, p90, alpha=0.2, color='steelblue', label='10th-90th per
plt.fill_between(t, p25, p75, alpha=0.3, color='steelblue', label='25th-75th per
plt.plot(t, middle, color='darkblue', linewidth=2.5, label='Median (50th percent
plt.plot(t, upper, color='steelblue', linewidth=1, alpha=0.8, linestyle='--')
plt.plot(t, lower, color='steelblue', linewidth=1, alpha=0.8, linestyle='--')
plt.plot(t, futurelogvix[1,:], color='crimson', linewidth=1.5, alpha=0.8, label=
plt.xlabel('Time (days)', fontsize=12, fontweight='bold')
plt.ylabel('Log-VIX', fontsize=12, fontweight='bold')
plt.legend(loc='upper left', fancybox=True, shadow=True, fontsize=10)
plt.scatter(0, np.log(13.2), color='black', s=50, zorder=5, label='Initial VIX =
plt.tight_layout()
plt.show()

for i in range(1000):
  plt.plot(t, np.exp(futurelogvix[i,:]))
plt.xlabel('Time')
plt.ylabel('VIX')
plt.grid()
plt.show()
```

Show hidden output

```
print(expectationA)
print(stdA)
```

Show hidden output

```
#Setting up quadrature
M=100
Xi0 = expectedfuturelogvix - 9*newvol
XiM = expectedfuturelogvix + 9*newvol
```

```
Xi = np.linspace(Xi0,XiM,M)

print(Xi)

#Computing probability density at each datapoint:
probdist = scipy.stats.norm(expectedfuturelogvix,newvol)
probdensities = np.log(probdist.pdf(Xi))
print(np.exp(probdensities))

#For value@risk
percent5value = probdist.ppf(0.05)
print(percent5value)


q_buy = cp.Variable(len(strikes), nonneg=True)
q_sell = cp.Variable(len(strikes), nonneg=True)
q_net = q_buy - q_sell #net position
q_riskfree = cp.Variable()

portfoliocost = cp.sum(cp.multiply(ask_prices, q_buy)) - cp.sum(cp.multiply(bic

def payoff(Xi):
    payoffs = np.zeros(len(strikes))
    for i in range(len(strikes)):
      if option_types[i] == 'C':
        payoffs[i] = np.maximum(np.exp(Xi) - strikes[i], 0)
      if option_types[i] == 'P':
        payoffs[i] = np.maximum(strikes[i] - np.exp(Xi),0)
    return payoffs

def objectivefunction(lamda,w,c):
  f = []
  for i in range(len(Xi)):
    portfolioATmaturity = cp.sum(cp.multiply(q_net, payoff(Xi[i]))) + q_riskfre
    f.append((lamda/w)*(c - portfolioATmaturity))

  delta = np.log((18*newvol)/100)
  quadraturepoints = []
  for i in range(len(Xi)):
    quadraturepoints.append(f[i] + probdensities[i] + delta)
  quadrature = cp.log_sum_exp(cp.hstack(quadraturepoints))
  objectivefunction = (1/lamda)*quadrature
  return objectivefunction
```

Show hidden output

```
def simulating_OU(V0, timesteps,Vmean,vol,a,paths):
  dt = 1
  V = np.zeros((paths, timesteps+1))
  V[:,0] = V0
  np.random.seed(42)
  DW = np.random.normal(0,1, size=(paths,timesteps))
  for i in range(timesteps):
    V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
  return V

maturityvix = simulating_OU(np.log(13.2), 18,Vmean,vol,a,10000)[:,-1]
print(maturityvix)
print(np.mean(maturityvix))
futuresstd = np.std(maturityvix)
maturityvix.shape
```

Show hidden output

```
#Portfolio Optimisation for lamda = 2, 20
def portfoliooptimisation(riskaversion):
  objective1 = cp.Minimize(objectivefunction(riskaversion,100000, 0))

  constraints = [q_buy <= ask_sizes,
                 q_sell <= bid_sizes,
                 ask_prices @ q_buy - bid_prices @ q_sell + q_riskfree <= 100000]

  prob = cp.Problem(objective1, constraints)
  result_2 = prob.solve(solver = cp.MOSEK)


  if prob.status == cp.OPTIMAL:
    print(f"Optimal portfolio value for lamda = {riskaversion}: {result_2:.5f}")
    print(f'Risk-Free position: {q_riskfree.value}')
    calculatedportfolio_cost = ask_prices @ q_buy.value - bid_prices @ q_sell.va
    print(f'Portfolio cost: {calculatedportfolio_cost}')
    print(f'Number of options traded: {np.sum(np.abs(q_net.value))}')
    print(f'Total cost of options {calculatedportfolio_cost-q_riskfree.value}')

    portfoliovalues = []
    for i in range(len(Xi)):
      portfoliovalues.append((q_net.value @ payoff(Xi[i])) + q_riskfree.value)

    reasonableportfolios = []
    for i in range(len(maturityvix)):
      reasonableportfolios.append((q_net.value @ payoff(maturityvix[i])) + q_ris

    realwordpayout = (q_net.value @ payoff(np.log(14.48))) + q_riskfree.value -
    print(f'Real-world P&L: {realwordpayout}')

    moneymakers = np.sum(np.array(reasonableportfolios) > 0)
    worstloss = np.min(reasonableportfolios)
    bestwin = np.max(reasonableportfolios)
    print(f'Percentage winners: {100*moneymakers/len(maturityvix)}%')
    print(f'Worst loss: {worstloss}')
    print(f'Best win: {bestwin}')


    Valatrisk = np.percentile(reasonableportfolios, 5)

    shortfalls = []
    for i in range(len(maturityvix)):
      if ((q_net.value @ payoff(maturityvix[i])) + q_riskfree.value -calculatedp
        shortfalls.append(q_net.value @ payoff(maturityvix[i])-calculatedportfol

    expectedshortfall = np.mean(shortfalls)
    print(f'Value at Risk: {Valatrisk}')
    print(f'Expected Shortfall: {expectedshortfall}')

    plt.figure(figsize=(10, 6))
    plt.plot(np.exp(Xi), portfoliovalues, linewidth=2.5, color='navy', label='Te

    plt.axvline(np.exp(np.mean(maturityvix)), linestyle='--', color='red', label
    plt.axvline(np.exp(np.mean(maturityvix))+np.exp(np.std(maturityvix)), linest
    plt.axvline(np.exp(np.mean(maturityvix))-np.exp(np.std(maturityvix)), linest
    plt.axhline(calculatedportfolio_cost, linestyle=':', color='orange', label='

    plt.xlabel('VIX at Maturity', fontsize=12, fontweight='bold')
    plt.ylabel('Portfolio Value ($)', fontsize=12, fontweight='bold')

    plt.legend(loc='best', framealpha=0.9, fontsize=10)
    plt.grid(True, alpha=0.3, linestyle='-', linewidth=0.5)
    plt.xlim(0,60)
    plt.ylim(-250000,250000)
    plt.tight_layout()
    plt.show()

    PNL = np.array(portfoliovalues) - 100000
    workingscenarios = np.vstack((np.exp(Xi), PNL)).T
    #print(workingscenarios)

    long_calls = []
    short_calls = []
    long_puts = []
    short_puts = []

    for i in range(len(strikes)):
        if option_types[i] == 'C':
            if q_net.value[i] > 0:
                long_calls.append(abs(q_net.value[i]))
            else:
                short_calls.append(abs(q_net.value[i]))
        elif option_types[i] == 'P':
            if q_net.value[i] > 0:
                long_puts.append(abs(q_net.value[i]))
            else:
                short_puts.append(abs(q_net.value[i]))



    plt.figure(figsize=(10, 6))
    legend_labels = set()

    for i in range(len(strikes)):
        if option_types[i] == 'C':
            if q_net.value[i] > 0:
```

```
                label = 'Long call' if 'Long call' not in legend_labels else ""
                plt.bar(strikes[i], q_net.value[i], color='blue', width=3, label
                legend_labels.add('Long call')
            else:
                label = 'Short call' if 'Short call' not in legend_labels else "
                plt.bar(strikes[i], q_net.value[i], color='cyan', width=3, label
                legend_labels.add('Short call')
        elif option_types[i] == 'P':
            if q_net.value[i] > 0:
                label = 'Long Put' if 'Long Put' not in legend_labels else ""
                plt.bar(strikes[i], -q_net.value[i], color='red', width=3, label
                legend_labels.add('Long Put')
            else:
                label = 'Short Put' if 'Short Put' not in legend_labels else ""
                plt.bar(strikes[i], -q_net.value[i], color='orange', width=3, la
                legend_labels.add('Short Put')

    plt.xlabel('Strike Price', fontsize=11, fontweight='bold')
    plt.ylabel('Net Position (contracts)', fontsize=11, fontweight='bold')
    plt.legend(loc='upper right')
    plt.show()

  else:
    print(f"Problem status: {prob.status}")

  return portfoliovalues, Valatrisk, expectedshortfall, reasonableportfolios

portfoliovalues2, VatRisk2, shortfall2, reasonableportfolios2 = portfoliooptimis
portfoliovalues20, VatRisk20, shortfall20, reasonableportfolios20 = portfolioopt
#portfoliovalues50, VatRisk50, shortfall50, reasonableportfolios50 = portfolioop

plt.figure(figsize=(10, 6))
sns.kdeplot(reasonableportfolios2, linewidth=1, label='λ = 2 (Moderate Risk Aver
sns.kdeplot(reasonableportfolios2, alpha=0.3, fill=True, color='red')
plt.axvline(VatRisk2, linestyle='--', color='red', label=f'Value at Risk: ${-Vat
#plt.axvline(shortfall2, linestyle=':', color='red', label=f'Shortfall: ${-short

sns.kdeplot(reasonableportfolios20, linewidth=1, label='λ = 20 (High Risk Aversi
sns.kdeplot(reasonableportfolios20, alpha=0.3, fill=True, color='blue')
plt.axvline(VatRisk20, linestyle='--', color='blue', label=f'Value at Risk: ${-V
#plt.axvline(shortfall20, linestyle=':', color='blue', label=f'Shortfall: ${-sho

#sns.kdeplot(reasonableportfolios50, linewidth=1, label='λ = 50 (Very High Risk
#sns.kdeplot(reasonableportfolios50, alpha=0.3, fill=True, color='green')
#plt.axvline(VatRisk50, linestyle='--', color='green', label=f'Value at Risk: ${

plt.xlabel('P&L ($)', fontsize=11, fontweight='bold')
plt.ylabel('Density', fontsize=11, fontweight='bold')
plt.ticklabel_format(style='plain')

plt.legend(fontsize=11, loc='upper right', framealpha=0.9)
plt.tight_layout()
plt.show()
```

Show hidden output

Task 5 and 6

```
def digitalpayoffs(Xi):
  payoffs = np.zeros(11)
  digitalstrikes = [5*k for k in range(11)]
  for k in range(len(digitalstrikes)):
    payoffs[k] = 1000 * (np.exp(Xi) > digitalstrikes[k])
  return payoffs

digitaloptionpayouts = []
for i in range(len(Xi)):
  digitaloptionpayouts.append(digitalpayoffs(Xi[i]))
digitaloptionpayouts = np.array(digitaloptionpayouts)

print(digitaloptionpayouts.shape)

def objectivefunction_with_liability_selling(lamda,w, digitaloptionpayouts, c,
  f = []
  for i in range(len(Xi)):
    F_position = digitaloptionpayouts[i,j]
    f.append((lamda/w)*(c + F_position))

  delta = np.log((18*newvol)/100)
  quadraturepoints = []
  for i in range(len(Xi)):
    quadraturepoints.append(f[i] + probdensities[i] + delta)
  quadrature = cp.log_sum_exp(cp.hstack(quadraturepoints))
  objective = (1/lamda)*quadrature
  return objective

def objective(lamda, w, c, j):
  f = []
  for i in range(len(Xi)):
    f.append((lamda/w)*(c))

  delta = np.log((18*newvol)/100)
  quadraturepoints = []
  for i in range(len(Xi)):
    quadraturepoints.append(f[i] + probdensities[i] + delta)
  quadrature = cp.log_sum_exp(cp.hstack(quadraturepoints))
  objective = (1/lamda)*quadrature
  return objective


def objectivefunction_with_liability_buying(lamda, w, digitaloptionpayouts, c,
  f = []
  for i in range(len(Xi)):
    F_position = digitaloptionpayouts[i,j]
    f.append((lamda/w)*(c - F_position))
  delta = np.log((18*newvol)/100)
  quadraturepoints = []
  for i in range(len(Xi)):
    quadraturepoints.append(f[i] + probdensities[i] + delta)
  quadrature = cp.log_sum_exp(cp.hstack(quadraturepoints))
  objective = (1/lamda)*quadrature
  return objective

def objectivecalculation(riskaversion, initialwealth, c):
  Sobjectives = []
  Bobjectives = []
  Nobjectives = []
  for j in range(11):
    Sobjective = (objectivefunction_with_liability_selling(riskaversion, initia
    Bobjective = (objectivefunction_with_liability_buying(riskaversion, initial
    Nobjective = (objective(riskaversion, initialwealth, 0, j))


    Sobjectives.append(Sobjective.value)
    Bobjectives.append(Bobjective.value)
    Nobjectives.append(Nobjective.value)


  return np.array(Sobjectives), np.array(Bobjectives), np.array(Nobjectives)


def indifferencepricing(Sobj2, Sobj20, Bobj2, Bobj20, Nobj2, Nobj20, c, w):

  indifferencesellingprice2 = np.zeros(11)
  indifferencesellingprice20 = np.zeros(11)
  indifferencebuyingprice2 = np.zeros(11)
  indifferencebuyingprice20 = np.zeros(11)

  for i in range(11):
    indifferencesellingprice2[i] = w * ((Sobj2[i])-(Nobj2[i]))
    indifferencesellingprice20[i] = w * ((Sobj20[i])-(Nobj20[i]))

    indifferencebuyingprice2[i] = w * ((Nobj2[i])-(Bobj2[i]))
    indifferencebuyingprice20[i] = w * ((Nobj20[i])-(Bobj20[i]))
  return indifferencesellingprice20, indifferencesellingprice2, indifferencebuy
```

```
   (100, 11)
```

```
#Calculations with c = 0

Sobj2, Bobj2, Nobj2 = objectivecalculation(2,100000, 0)
Sobj20, Bobj20, Nobj20 = objectivecalculation(20,100000, 0)

price20s, price2s, price20b, price2b = indifferencepricing(Sobj2, Sobj20, Bobj2

print('For Lamda = 20')
print(f'Indifference selling price: {price20s}')
print(f'Indifference buying price: {price20b}')
print('For Lamda = 2')
print(f'Indifference selling price: {price2s}')
print(f'Indifference buying price: {price2b}')

prctdif2 = ((price2s - price2b)/(0.5*(price2b+price2s)))*100
prctdif20 = ((price20s - price20b)/(0.5*(price20b+price20s)))*100

prctdifbuy = ((price2b - price20b)/(0.5*(price20b+price2b)))*100
prctdifsell = ((price2s - price20s)/(0.5*(price20s+price2s)))*100

prctdifbs = ((0.01*prctdifbuy + 0.01*prctdifsell)/(0.5*(0.01*prctdifbuy-0.01*pr
```

Show hidden output

```
print(prctdif2)
print(prctdif20)
print(prctdifbuy)
print(prctdifsell)
print(prctdifbs)
```

Show hidden output

```
plt.plot([5*k for k in range(11)], price20b,
         label='Ask Price',
         linewidth=2.5, color='darkblue', marker='o', markersize=6)
plt.plot([5*k for k in range(11)], price20s,
         label='Bid Price',
         linewidth=2.5, color='crimson', marker='s', markersize=6)
plt.xlabel('Strike Price', fontsize=11, fontweight='bold')
plt.ylabel('Indifference Price ($)', fontsize=11, fontweight='bold')
plt.legend(loc='upper right', framealpha=0.9, fontsize=10)
plt.xticks(range(0, 55, 10), fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()


plt.plot([5*k for k in range(11)], price2b,
         label='Ask Price',
         linewidth=2.5, color='darkblue', marker='o', markersize=6)
plt.plot([5*k for k in range(11)], price2s,
         label='Bid Price',
         linewidth=2.5, color='crimson', marker='s', markersize=6)
plt.xlabel('Strike Price', fontsize=11, fontweight='bold')
plt.ylabel('Indifference Price ($)', fontsize=12, fontweight='bold')
plt.legend(loc='upper right', framealpha=0.9, fontsize=10)
plt.xticks(range(0, 55, 10), fontsize=10)
plt.yticks(fontsize=10)
plt.tight_layout()
plt.show()

plt.plot([5*k for k in range(11)], prctdif20,
         label='λ = 20 (High Risk Aversion)',
         label='λ = 20 (High Risk Aversion)',
         linewidth=2.5, color='maroon', marker='o', markersize=6)
plt.plot([5*k for k in range(11)], prctdif2,
         label='λ = 2 (Moderate Risk Aversion)',
         linewidth=2.5, color='darkgreen', marker='s', markersize=6)

plt.xlabel('Strike Price', fontsize=11, fontweight='bold')
plt.ylabel('Bid-Ask Spread (%)', fontsize=11, fontweight='bold')
plt.legend(loc='upper left', framealpha=0.9, fontsize=10)
plt.xticks(range(0, 55, 10), fontsize=10)
plt.yticks(fontsize=10)

plt.tight_layout()
plt.show()

plt.plot([5*k for k in range(11)], prctdifbuy,
         label='Ask Price',
         linewidth=2.5, color='darkblue', marker='o', markersize=6)
plt.plot([5*k for k in range(11)], prctdifsell,
         label='Bid Price',
         linewidth=2.5, color='crimson', marker='s', markersize=6)

plt.xlabel('Strike Price', fontsize=12, fontweight='bold')
plt.ylabel('Price Difference (%): λ = 2 vs λ = 20', fontsize=11, fontweight='bol
plt.legend(loc='upper left', fontsize=10)
plt.xticks(range(0, 55, 10), fontsize=10)
plt.yticks(fontsize=10)

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

Show hidden output

Appendix D

Part 3 code

```
import numpy as np
import numpy.random as npr
from scipy.stats import norm
import tensorflow.keras as keras
import tensorflow.keras.backend as K
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
import seaborn as sns

plt.style.use('ggplot')


np.set_printoptions(suppress=True)
np.set_printoptions(precision=5)


vol = 0.07659374420551314
Vmean = 2.910104770529511
a = 0.024498317590460172
F0 = 13.2
T=18
Timesteps = 18
paths = 10000
dt = T/Timesteps
BpD = 100000/18

def simulating_OU(V0, timesteps,Vmean,vol,a,paths):
  dt = 1
  V = np.zeros((paths, timesteps+1))
  V[:,0] = V0
  DW = np.random.normal(0,1, size=(paths,timesteps))
  for i in range(timesteps):
    V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
  return V

futuressimulation = np.exp(simulating_OU(np.log(13.2),18,Vmean,vol,a,10000))
returns = np.diff(futuressimulation, axis=1) / futuressimulation[:,:-1]

print(f"Futures simulation shape: {futuressimulation}")
print(f"Price differences shape: {returns.shape}")
print(f"Initial price: {F0}")
print(f"Mean final price: {np.mean(futuressimulation[:, -1]):.2f}")
print(f"Std final price: {np.std(futuressimulation[:, -1]):.2f}")


tim = np.linspace(0,T,Timesteps+1)
X = np.zeros((paths,Timesteps, 3))

for i in range(Timesteps):
  X[:,i, 0] = tim[i]
  X[:,i, 1] = futuressimulation[:,i]
```

```
    if i > 0:
      X[:,i, 2] = (futuressimulation[:,i] - futuressimulation[:,i-1])/futuressimul
    else:
      X[:,i, 2] = 0

  print(f"Shape of X: {X.shape}")

  pricemean = np.mean(X[:,:,1])
  pricestd = np.std(X[:,:,1])

  Xtrain = X.copy()
  Xtrain[:,:,0] = X[:,:,0]
  Xtrain[:,:,1] = (X[:,:,1] - pricemean)/pricestd
  Xtrain[:,:,2] = X[:,:,2]

  print(f"Shape of Xtrain: {Xtrain.shape}")


  cutfuturessimulation = tf.constant(futuressimulation[:,:-1], dtype=tf.float32)


  def metrics(positions, training):


      valueofinvestment = cutfuturessimulation*positions

      averagedailyposition = np.mean(positions, axis=0)
      dailypositionstd = np.std(positions, axis=0)

      print('\nTrading metrics:')
      print(f'Position shape: {positions.shape}')
      print(f'Mean value of investment: {np.mean(valueofinvestment)}')
      print(f'Position range: [{np.min(positions)}, {np.max(positions)}]')
      print(f'Mean absolute position: {np.mean(np.abs(positions)):.2f}')
      print(f'Std position: {dailypositionstd}')

      #Performance metrics

      dailyPnL = positions * cutfuturessimulation * returns
      averagePnL = np.mean(dailyPnL)
      dailyPnLstd = np.std(dailyPnL)
      totalPnL = np.sum(dailyPnL, axis=1)
      meantotalPnL = np.mean(totalPnL)
      totalPnLstd = np.std(totalPnL)
      maxPnL = np.max(totalPnL)
      minPnL = np.min(totalPnL)
      totalPnLstd = np.std(totalPnL)
      wealthevolution = np.cumsum(dailyPnL, axis=1)
      meanwealthevolution = np.mean(wealthevolution, axis=0)
      wealthevolutionstd = np.std(wealthevolution, axis=0)
      sharperatio = np.mean(totalPnL)/np.std(totalPnL)

      print('\nPerformance metrics:')
      print(f'Mean daily PnL: {averagePnL}')
      print(f'Daily PnL std: {dailyPnLstd}')
      print(f'Mean Total PnL: {meantotalPnL}')
      print(f'Total PnL std: {totalPnLstd}')
      print(f'Total PnL range: [{minPnL}, {maxPnL}]')
      print(f'Mean modified Sharpe ratio: {sharperatio}')
      print(f'Mean wealth evolution: {meanwealthevolution}')
      print(f'Wealth evolution std: {wealthevolutionstd}')

      winrate = np.sum(totalPnL > 0) / len(totalPnL)
      print(f'Win rate: {winrate:.2%}')

      valueatrisk = np.percentile(totalPnL, 5)
      print(f'Value at risk (95th percentile): {valueatrisk}')

  #-------------------------------------------------------------------------------

      #plt.figure(figsize=(10, 6))
      #plt.plot(training.history['loss'], label='Training loss')
      #plt.plot(training.history['val_loss'], label='Validation loss')
      #plt.xlabel('Epoch', fontsize=10, fontweight='bold')
      #plt.ylabel('Loss', fontsize=10, fontweight='bold')
      #plt.legend()
      #plt.grid(True)
      #plt.tight_layout()
      #plt.show()

  #-------------------------------------------------------------------------------


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

      n, bins, patches = plt.hist(totalPnL, bins=50, alpha=0.7, color='steelblue'
                                 edgecolor='darkblue', linewidth=0.8, label='Tota

      plt.xlabel('Cumulative P&L ($)', fontsize=10, fontweight='bold')
      plt.ylabel('Frequency', fontsize=10, fontweight='bold')
      plt.figure(figsize=(10, 6))

  #-------------------------------------------------------------------------------

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

      pos_data = positions.flatten()
      pos_data = pos_data[~np.isnan(pos_data)]

      n, bins, patches = plt.hist(pos_data, bins=50, alpha=0.7, color='forestgree
                                 edgecolor='darkgreen', linewidth=0.8)

      plt.xlabel('Position Size (Contracts)', fontsize=10, fontweight='bold')
      plt.ylabel('Frequency', fontsize=10, fontweight='bold')
      plt.tight_layout()
      plt.show()

  #-------------------------------------------------------------------------------
```

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

    wealthevolution0 = np.hstack((np.zeros((10000,1)), wealthevolution))

    t = np.linspace(0, len(tim)-1, len(tim))
    upper = np.percentile(wealthevolution0, 95, axis=0)
    middle = np.mean(wealthevolution0, axis=0)
    lower = np.percentile(wealthevolution0, 5, axis=0)
    p75 = np.percentile(wealthevolution0, 75, axis=0)
    p25 = np.percentile(wealthevolution0, 25, axis=0)

    plt.fill_between(tim, lower, upper, alpha=0.15, color='steelblue', label='5
    plt.fill_between(tim, p75, p25, alpha=0.2, color='steelblue', label='10th-9

    plt.plot(tim, middle, color='darkblue', linewidth=2.5, label='Mean')
    plt.plot(tim, upper, color='steelblue', linewidth=1, alpha=0.8, linestyle='
    plt.plot(tim, lower, color='steelblue', linewidth=1, alpha=0.8, linestyle='


    plt.plot(tim, wealthevolution0[1,:], color='crimson', linewidth=1.5, alpha=
                label='Sample path')

    plt.xlabel('Time (days)', fontsize=10, fontweight='bold')
    plt.ylabel('Total Profit', fontsize=10, fontweight='bold')
    plt.legend(loc='upper left', frameon=True, fancybox=True, shadow=True, font
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

#-------------------------------------------------------------------------------

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

    averagedailyposition0 = np.hstack((0, averagedailyposition))
    dailypositionstd0 = np.hstack((0, dailypositionstd))

    plt.plot(tim, averagedailyposition0, color='steelblue', linewidth=2,
            linestyle=':', alpha=0.8, label='Position Trend')

    plt.errorbar(tim, averagedailyposition0, yerr=dailypositionstd0,
                fmt='none', ecolor='lightsteelblue', alpha=0.6,
                capsize=4, capthick=1.5, elinewidth=1.5,
                label='±1 Std Dev', zorder=3)

    colors = ['darkgreen' if pos > 0.1 else 'darkred' if pos < -0.1 else 'gray'
              for pos in averagedailyposition0]

    sizes = [60 if abs(pos) > 0.5 else 40 for pos in averagedailyposition0]
    for i, (day, pos, color, size) in enumerate(zip(tim, averagedailyposition0,
        plt.scatter(day, pos, color=color, s=size, alpha=0.8,
                  edgecolors='white', linewidths=1.5, zorder=5)

    plt.axhline(y=0, color='black', linestyle='-', linewidth=1.5, alpha=0.6)
    max_pos = max(abs(np.max(averagedailyposition0)), abs(np.min(averagedailypo
    if max_pos > 0:
        plt.axhspan(0.1, max_pos * 10, alpha=0.1, color='green', label='Long Zo
        plt.axhspan(-max_pos * 10, -0.1, alpha=0.1, color='red', label='Short Z

    plt.xlabel('Day', fontsize=10, fontweight='bold')
    plt.ylabel('Mean Position Size (Contracts)', fontsize=10, fontweight='bold'
    plt.ylim(-500, 500)
    plt.legend(loc='best', frameon=True, fancybox=True, shadow=True, fontsize=9
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()


cutfuturessimulation = tf.constant(futuressimulation[:,:-1], dtype=tf.float32)


def betterutility(ytrue, ypred):
  #Budget = £100,000
  #Budgetperday = £100,000/18 days

  valueofinvestment = ypred * BpD

  dailyPnL = valueofinvestment * ytrue
  totalPnL = K.sum(dailyPnL, axis=1)

  meanPnL = K.mean(totalPnL)
  PnLstd = K.std(totalPnL)

  sharperatio = meanPnL/PnLstd

  return -sharperatio


def shtradingNN(Timesteps, inputdimension=3, l2_reg=0.01):
    inputs = keras.layers.Input(shape=(Timesteps, inputdimension))
    x = keras.layers.LayerNormalization()(inputs)

    x = keras.layers.Dense(64, activation='linear')(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Dense(128, activation='relu',
                         kernel_regularizer=keras.regularizers.l2(l2_reg))(x)

    x = keras.layers.Dense(64, activation='relu',
                         kernel_regularizer=keras.regularizers.l2(l2_reg))(x)
    x = keras.layers.Dropout(0.2)(x)

    x = keras.layers.Dense(32, activation='relu',
                         kernel_regularizer=keras.regularizers.l2(l2_reg))(x)
    x = keras.layers.Dropout(0.1)(x)

    position = keras.layers.Dense(1, activation='tanh')(x)
    positions = keras.layers.Reshape((Timesteps,))(position)

    model = keras.Model(inputs=inputs, outputs=positions)
    return model


def shtradingNN1(Timesteps, inputdimension=3, l2_reg=0.01):
    inputs = keras.layers.Input(shape=(Timesteps, inputdimension))
    x = keras.layers.LayerNormalization()(inputs)

    x = keras.layers.Dense(64, activation='linear')(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.Dense(128, activation='relu',
                           kernel_regularizer=keras.regularizers.l2(l2_reg))(x)

    x = keras.layers.Dense(64, activation='relu',
                           kernel_regularizer=keras.regularizers.l2(l2_reg))(x)
    x = keras.layers.Dropout(0.2)(x)

    x = keras.layers.Dense(32, activation='relu',
                           kernel_regularizer=keras.regularizers.l2(l2_reg))(x)
    x = keras.layers.Dropout(0.1)(x)

    position = keras.layers.Dense(1, activation='tanh')(x)
    positions = keras.layers.Reshape((Timesteps,))(position)

    model = keras.Model(inputs=inputs, outputs=positions)
    return model


smodel1 = shtradingNN(Timesteps, inputdimension=3)

smodel1.compile(optimizer='adam', loss=betterutility)
smodel1.summary()

straining1 = smodel1.fit(Xtrain, returns, epochs=15, batch_size=32, validation_
spositions1 = (BpD/cutfuturessimulation)*smodel1.predict(Xtrain, batch_size=32)
spositions1 = np.array(spositions1)


z = metrics(spositions1,straining1)


def ourutility2(ytrue, ypred):
  lamda = 2
  wbar=1000

  valueofinvestment = ypred * BpD
  dailyPnL = valueofinvestment * ytrue
  totalPnL = K.sum(dailyPnL, axis=1)

  C = (lamda * totalPnL)/wbar

  maxC = K.max(C)
  expPnL = K.exp(C - maxC)
  meanexpPnL = K.mean(expPnL)
  disutility = (1/lamda) * (maxC + K.log(meanexpPnL))
  return -disutility


smodel2 = shtradingNN1(Timesteps, inputdimension=3)
smodel2.compile(optimizer='adam', loss=ourutility2)
smodel2.summary()

straining2 = smodel2.fit(Xtrain, returns, epochs=15, batch_size=32, validation_
spositions2 = (BpD/cutfuturessimulation)*smodel2.predict(Xtrain, batch_size=32)
spositions2 = np.array(spositions2)
```

Show hidden output

```
b1 = metrics(spositions2,straining2)
```

Testing

```
def simulating_OU(V0, timesteps,Vmean,vol,a,paths):
  dt = 1
  V = np.zeros((paths, timesteps+1))
  V[:,0] = V0
  DW = np.random.normal(0,1, size=(paths,timesteps))
  for i in range(timesteps):
    V[:,i+1] = V[:,i] - a*(V[:,i] - Vmean)*dt + vol*np.sqrt(dt)*DW[:,i]
  return V

futuressimulationT = np.exp(simulating_OU(np.log(13.2),18,Vmean,vol,a,10000))
returnsT = np.diff(futuressimulationT, axis=1) / futuressimulationT[:,:-1]

print(f"Futures simulation shape: {futuressimulationT}")
print(f"Price differences shape: {returnsT.shape}")
print(f"Initial price: {F0}")
print(f"Mean final price: {np.mean(futuressimulationT[:, -1]):.2f}")
print(f"Std final price: {np.std(futuressimulationT[:, -1]):.2f}")


tim = np.linspace(0,T,Timesteps+1)
Xtest = np.zeros((paths,Timesteps, 3))

for i in range(Timesteps):
  Xtest[:,i, 0] = tim[i]
  Xtest[:,i, 1] = futuressimulationT[:,i]
  if i > 0:
    Xtest[:,i, 2] = (futuressimulationT[:,i] - futuressimulationT[:,i-1])/futur
  else:
    Xtest[:,i, 2] = 0

print(f"Shape of Xtest: {Xtest.shape}")

pricemeanT = np.mean(Xtest[:,:,1])
pricestdT = np.std(Xtest[:,:,1])
```

```
Xtest = Xtest.copy()
Xtest[:,:,0] = Xtest[:,:,0]
Xtest[:,:,1] = (Xtest[:,:,1] - pricemeanT)/pricestdT
Xtest[:,:,2] = Xtest[:,:,2]

cutfuturessimulationT = tf.constant(futuressimulationT[:,:-1], dtype=tf.float32
```

```
    Shape of Xtest: (10000, 18, 3)
```

```
testpositions1 = (BpD/cutfuturessimulationT)*smodel1.predict(Xtest, batch_size=
testpositions1 = np.array(testpositions1)
```

```
test1 = metrics(testpositions1,straining1)
```

```
testpositions2 = (BpD/cutfuturessimulationT)*smodel2.predict(Xtest, batch_size=
testpositions2 = np.array(testpositions2)
```

```
test2 = metrics(testpositions2, straining2)
```

Bibliography

[1] Katja Ahoniemi. Modeling and forecasting the vix index. *SSRN*, 2008.

[2] N. Albuquerque, G. Araujo, D. Pellegrino, and J. Seoane-Sepulveda. Hölder's inequality: some recent and unexpected applications, 2015.

[3] John Armstrong. Numerical and computational methods in finance, 2024. Department of Mathematics, King's College, Lecture Notes.

[4] Fischer Black. The pricing of commodity contracts. *Journal of Financial Economics*, 3, 1976.

[5] Kathleen Blake, Mohammed Gharbawi, Oliver Thew, Seema Visavadia, Leo Gosland, and Henrike Mueller. Machine learning in UK financial services. Report, Bank of England and Financial Conduct Authority, London, UK, October 2022.

[6] CBOE. White paper cboe volatility index. White paper, Chicago Board Options Exchange (CBOE), 2019.

[7] Gregory Gunderson. Deriving the vix. https://gregorygundersen.com/blog/2023/09/10/deriving-vix/, 2023.

[8] John C. Hull. *Options, futures and Other Derivatives.* Pearson, 9th edition, 2015.

[9] Ralph Korn. *Optimal Portfolios, Stochastic Models for Optimal Investment and Risk Management in Continuous Time.* World Scientific Publishing CO. Pte. Ltd, 1999.

[10] Michael Kamal Joseph Zou Kresemir Demeterfi, Emanuel Derman. More than you ever wanted to know about volatiltiy swaps. Research report, Goldman Sachs, 1999.

[11] Stefan Nagel. *Machine Learning in Asset Pricing*. Princeton Lectures in Finance. Princeton University Press, Princeton, 2021.

[12] Anuja Nagpal. L1 and l2 regularization methods, explained, Oct 2024. Updated by Brennan Whitfield.

[13] Teemu Pennanen. Incomplete markets, 2025. Department of Mathematics, King's College, Lecture Notes.

[14] Spyridon Pougkakiotis. *Machine Learning Notes*. Department of Mathematics, King's College, 2025. Department of Mathematics, King's College, Lecture Notes.

[15] PyTorch Team. torch.nn.batchnorm2d. `https://docs.pytorch.org/stable/generated/torch.nn.BatchNorm2d.html`, 2024. PyTorch Documentation.

[16] PyTorch Team. torch.nn.layernorm. `https://docs.pytorch.org/stable/generated/torch.nn.LayerNorm.html`, 2024. PyTorch Documentation.

[17] Markus Riedle. Probability theory, 2023. Department of Mathematics, King's College, Lecture Notes.

[18] Merton Robert C. *The Bell Journal of economics and Management Science*. Journal of Political Economy, 1973.

[19] Bertrand Travacca, Laurent El Ghaoui, and Scott Moura. *Implicit Optimization: Models and Methods*. IEEE, 2020.