Towards Arbitrage-free Implied Volatility Surfaces with Diffusion Probabilistic Models

MSc Thesis in Applied Mathematics

Yihan Hu, 2024

TU Delft

MSc thesis in Applied Mathematics

Towards Arbitrage-free Implied Volatility Surfaces with Diffusion Probabilistic Models

Yihan Hu

October 2024

Student number:	5704979
Project duration:	January 2024 – October 2024
Thesis committee:	Dr. Cornelis Vuik, TU Delft, supervisor
	Dr. Shuaiqiang Liu, TU Delft, ING Bank, daily supervisor
	Dr. Antonis Papapantoleon, TU Delft, committee

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Applied Mathematics

Abstract

Implied volatility surfaces (IVS) are essential for option pricing and risk management. Recently, generative deep learning models, such as the Denoising Diffusion Probabilistic Model (DDPM), have gained popularity for generating IVS. However, these machine-generated surfaces do not necessarily meet arbitrage-free constraints, e.g., calendar spread and butterfly arbitrage. This thesis addresses this limitation by integrating arbitrage-free constraints into the DDPM generation process. We introduce a classifier specifically designed to detect violations of arbitrage-free conditions. During the DDPM's iterative generation process, the classifier guides the formation of the IVS at each step, effectively minimizing potential arbitrage opportunities. To evaluate the effectiveness of our approach, we compare the extent of arbitrage violations in IVS generated by the standard DDPM with those produced by our enhanced DDPM-Classifier model. The results demonstrate that the DDPM-Classifier significantly reduces arbitrage opportunities and improves the quality of IVS.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Dr. Shuaiqiang Liu, for his support throughout the entire process of my thesis writing. He took great responsibility for overseeing my thesis work, consistently answering my questions, and providing guidance through weekly meetings. With his help, I have gained substantial knowledge about the forefront of machine learning. Moreover, his feedback and advice on academic writing have greatly contributed to the improvement of my writing skills. I would also like to extend my thanks to Prof. Kees Vuik and Prof. Antonis Papapantoleon, members of my thesis defense committee, for their suggestions and feedback on my thesis, which will further motivate me to keep improving.

During these past two years living abroad, I am deeply grateful for the companionship and help from my friends. First, I want to thank Gu Yingxing. I am especially thankful for the bowl of noodles she made for me when I had a high fever. I miss our late-night hangouts, drinking and chatting, and the New Year's Eve we spent watching fireworks together. I also want to thank my friend, Ji Zhonghui. Despite the time difference, she has always been there to offer me support and encouragement during my difficult times. I am equally grateful to my friend Abby Peng, and I hope we will still have chances to play co-op games together in the future. In addition, I would also like to thank some of my classmates for their help and inspiration in my studies. Last but not least, I want to thank my favorite band, Mayday. Their concerts and songs have provided me with much emotional strength.

And of course, I would also like to extend my heartfelt thanks to my parents, Mr. Hu and Ms. Wang. I am grateful that they have supported my choices and accompanied me through many important moments in my life. I cannot forget to thank my beloved cat, Mr. Xiaomi, for the endless joy and comfort he has brought me. I have already missed your second and third birthday, and I can not wait to return to Beijing to see you soon.

Reflecting on the past two years, I have certainly faced moments of pain and stress, but I have also experienced breathtaking sights and created unforgettable memories. I always feel that life is full of wonderful encounters. For example, I never considered myself good at maths, but I chose to study it as an undergraduate and now I am about to get a master's degree in it. I have to thank myself for my hard work. Perhaps sometimes hard work can compensate for some talent.

"I leave no trace of wings in the air, but I am glad I have had my flight."

Yihan Hu
October 2024

Contents

1 Introduction

1.1 Background

An option is a type of financial derivative that grants the buyer the right, but not the duty, to purchase or sell the underlying asset at a predefined price on or before a certain expiration date. The price of an option is influenced by several key elements, including the underlying asset's price, the striking price, the maturity date (also known as the expiry date), the interest rate, and the volatility of the underlying asset. Out of all these impacts, our main focus will be on volatility, specifically implied volatility. Historical volatility is determined by analyzing past variations in market prices. Conversely, implied volatility (IV) is derived from the present market value of the option. Implied volatility holds significant value in option pricing models like the Black-Scholes model as it accurately represents the market's anticipation of future volatility. Although it cannot be directly observed, it can be determined using the Black-Scholes model. Typically, an increase in implied volatility indicates more expectation of fluctuations in asset prices, which leads to a rise in the option premium. Conversely, a decrease in implied volatility suggests that the expected price movement will reduce and the premium will decrease.

An Implied Volatility Surface(IVS) is a graphical representation in three dimensions of the implied volatility for various options, considering strike prices and expiry dates. The conventional approach for generating the Black-Scholes implied volatility surfaces involves the utilization of mathematical methodologies such as the Heston model. Nevertheless, the Black-Scholes model regards volatility as a constant and employs some other simplifications to minimize complexity, which may fail to describe real financial market conditions.

In recent times, researchers have started employing machine learning approaches to generate IVS. These strategies often produce surfaces by training on a dataset of real IVS and subsequently leveraging deep learning methods to generate new surfaces. Deep learning methods, for instance, generative adversarial networks (GANs) [1, 2], variational autoencoder models (VAE) [3, 4, 5], and diffusion models have demonstrated excellent performance in creating and enhancing IVS. By applying these data-driven methods, it is possible to obtain financial data from real financial markets and generate plenty of realistic and accurate surfaces. Another important advantage is that these methods effectively preserve some of the shape characteristics of IVS.

In our research, the Denoising Diffusion Probabilistic Model(DDPM) will be used as the main deep-learning model. Compared to VAE and GAN, DDPM has higher image generation quality[6], which is why it is more widely used in image generation tasks in various fields, such as medical image synthesis[7], remote sensing change detection[8], etc. DDPM also has a more stable training process compared to the other two deep learning models. The training process of the GAN model is susceptible to the game process between the generator and the discriminator, and sometimes mode collapse occurs. DDPM generates images through a reverse process based on the Markov chain, which is supported by strict mathematical principles. At the same time, DDPM also has better interpretability as its image generation process is generated by a step-by-step denoising process.

As these deep learning models become more mature in generating IVS, more attention is beginning to be paid to the quality of the generated IVS. One of the most common requirements for IVS is the arbitrage-free condition. As showed by the Efficient Market Hypothesis (EMH) proposed by Fama [9], markets are efficient, leaving no room to make excess profits by investing since everything is already fairly and accurately priced. This means there is no arbitrage strategy in efficient markets. If there are arbitrage opportunities in IVS, it implies that trading strategies can be devised to profit without risk, which is not reasonable in an efficient market. Considering some common application scenarios for IVS, for example, traders often rely on IVS to develop hedging strategies and price derivatives. If there are arbitrage opportunities in IVS, wrong decisions of trading strategies may occur, leading to potential losses. Additionally, as we mentioned before, option prices can be affected by implied volatility, and an arbitrage-free IVS can help improve accuracy when computing option prices.

With such financial requirements, researchers have also proposed many kinds of methods for adding financial constraints to the generated models. For VAE and GAN, a common approach is to modify their loss function by adding penalty terms according to financial constraints so that the model can meet the corresponding requirements during training. The feasibility of this approach lies in the fact that both these two model's loss functions are all directly related to the generated samples. However, when we consider using the DDPM to generate samples with constraints, modifying the loss function becomes less realistic and more complex. Since the loss function of DDPM measures the gap between the predicted noise and the real noise added rather than measures the features of the generated image, modifying the loss function during the training process does not bring about a change in the generation of the image.

Recently, some conditional DDPMs have been proposed by researchers. These conditional DDPMs may achieve the purpose of imposing conditions on the generated samples by adding restrictions to the generation process. Another way is to use both conditional and unconditional models jointly during the training process. Such models have produced some great results in the conditional generation of images. At present, however, there is still a lack of research on adding relevant financial constraints during IVS generation.

1.2 Research Objectives

One of the main difficulties in generating implied volatility surfaces is the requirement to guarantee that the surfaces do not contain any arbitrage opportunities, particularly those related to calendar spread and butterfly arbitrage conditions.

Although machine learning methods exhibit superior efficiency and generation capabilities, they also face challenges in this aspect. If the no-arbitrage restrictions are not enforced in the generated IVS, it can result in a decrease in the reliability of IVS.

The main focus of our study is how to modify or adapt the DDPM to generate IVS that satisfy the no-arbitrage requirement. The no-arbitrage requirement considers two key scenarios: the calendar spread and the butterfly arbitrage strategy. In the third chapter, we will describe in detail the form of these two arbitrage strategies and the reasons for them. Our objective is to determine how to integrate these no-arbitrage requirements into the training or generation process of the diffusion model.

Furthermore, it is essential to consider methods for guaranteeing the fulfillment of the no-arbitrage requirements while keeping the quality of the model generation. A key factor that needs to be investigated is how to effectively balance the no-arbitrage requirement with high-quality surface generation. For instance, it is also important to assess if implementing these arbitrage conditions will have negative impacts on the generation of implied volatility surfaces.

1.3 Contributions

As mentioned above, the main contribution of this work focuses on the generation of arbitrage-free implied volatility surfaces using machine learning techniques (in particular, Denoising Diffusion Probabilistic Model) as follows:

1. **Integration of Diffusion Models with Arbitrage-free Conditions:**

   We present an innovative strategy that combines the generation process of diffusion models with a Classifier capable of determining if a surface meets the no-arbitrage condition. Additionally, we utilize a gradient descent algorithm to ensure that the generated implied volatility surfaces (IVS) satisfy the no-arbitrage requirements.

2. **Methods that Integrate Smoothness Constraints into the Generation Process:**

   To enhance the quality of the generated IVS, we also integrate smoothness constraints into the generation process of the diffusion model jointly. This constraint guarantees the avoidance of excessive fluctuations in the generation of surfaces, resulting in a more accurate representation of real market behavior in implied volatility surfaces.

3. **Empirical Analysis and Validation:**

   We conclude with comprehensive data experiments and results to show the efficacy of our approach. This includes a comprehensive analysis of the IVS produced by mathematical techniques, the DDPM without the Classifier, and our DDPM that incorporates the Classifier and smoothness requirements. The results indicate that the newly created surfaces have a better performance in avoiding arbitrage opportunities.

1.4 Thesis Outline

This thesis focuses on how to add financial constraints(i.e.,arbitrage-free conditions) on the generated implied volatility surfaces using a conditional denoising diffusion probabilistic model(DDPM). Firstly, the thesis introduces the background of the problem, and some of the proposed methods for adding constraints to IVS. In Chapter 2, the literature review is aimed to review the recent literature related to the topic. Chapter 3 introduces some financial mathematics knowledge, including the arbitrage-free conditions for the option price surface and implied volatility surface. Chapter 4 introduces the concept of the generative model and details two common conditional DDPMs, classifier and classifier-free DDPM. Chapter 5 describes the detailed implementation of adding financial constraints to the implied volatility surface. Chapter 6 discusses the experimental process and results, including the validation for the no-arbitrage condition, and the effect of adding the no-arbitrage condition on IVS, and compares the difference in results between the DDPM model and the DDPM-Classifier model. Chapter 7 summarises the research and makes recommendations for further research in the future.

2 Literature Review

This literature review aims to provide an overview of the significant advancements in generating Implied volatility surfaces using different machine learning methods, particularly focusing on the deep diffusion model.

Traditional mathematical approaches in generating IVS have relied on models like the Black-Scholes formula and stochastic volatility models like the Heston model. However, both these models often have difficulty in capturing the behavior of data in real financial markets. In real financial markets, the direction of the data may be affected by market sentiment, macroeconomic events, and other factors, leading to sudden changes in implied volatility. The volatility behavior during extreme market conditions (e.g. financial crises) is difficult to model from historical market data alone[10]. All these reasons have led to attempts to apply machine learning models to constructing implied volatility to learn the complex patterns in implied volatility surfaces from large-scale historical data.

In past research, the common machine learning methods for generating IVS are VAE, GAN, and DDPM. In the discussion of these methods, we intend to talk about their advantages, disadvantages, and differences. Another important problem is how the financial requirements(in particular, the no-arbitrage requirement) for IVS can be reasonably integrated.

In addition, through these discussions, we also aim to identify research gaps between the current research literature and our research objectives. This research gap will effectively contribute to our exploration process.

2.1 Mathematical Method for IVS

The Black-Sholes model is one of the most famous mathematical models for dynamic financial markets and is widely used in option pricing[11]. In the B-S model, the volatility of an asset is set as a constant. In constructing the implied volatility surface, we must obtain the implied volatility under different strike prices and maturities. Therefore, in the traditional mathematical approach, people always use the Heston model to generate a series of strike prices and maturities and their corresponding option prices[12]. Then solve the inverse of the Black-Scholes equation to obtain enough implied volatility data to construct the surface. The Heston model is a stochastic volatility model, which assumes that volatility follows a stochastic process rather than being a constant, thus making the calculations more consistent with real-world patterns.

But the drawbacks of the mathematical approach are also obvious, implied volatility surface has many shape features, widely known as volatility skew and volatility smile.

The volatility skew describes how implied volatility changes with different strike prices of options. It shows that the option with strike prices further from the current market price has higher implied volatility, reflecting the market's concern about large price decreases or increases.

The volatility smile is another common feature in that the options with strike prices significantly above or below the current market price have higher implied volatility. This results in a smile curve when plotting implied volatility against strike prices, meaning a higher possibility of extreme price movements in either direction.

In addition to these two common and famous features, real financial markets often exhibit other complex characteristics on implied volatility surfaces that are difficult to capture in math models. These include sudden jumps due to some policy uncertainty and investor's sentiment[13], short-term underreaction or long-term overreaction caused by financial crises[14]. In addition, extreme market conditions may also lead to unusual tailing behavior in volatility[15].

Such shape features often reflect the direction of real financial markets, and it is difficult to mimic them using only mathematical models (e.g., Heston models).

Therefore, when machine learning techniques are developed, in particular some models for generating images, such as VAE, GAN, and DDPM, have been applied to the field of IVS generation. They can acquire the shape features of the images from the training set and retain these learned features in the newly generated images.

2.2 Machine Learning Methods for IVS

The VAE(Variational Autoencoder) model generates new images by learning the latent distribution of the data[16]. It mainly consists of an encoder and a decoder. The encoder transforms the input data into probability distribution parameters of latent space, and through the reparameterization, the sampling process can be backpropagated. Finally, the latent variables are converted back to the original data space by the decoder. During the training process of VAE, the KL divergence is used as part of the loss function and minimized to force the encoder to learn the latent distribution as close as possible to the standard normal distribution. When generating a new image, the model samples the latent variables from the standard normal distribution and then feeds them into the decoder to generate new data.

The GAN(Generative Adversarial Network) model consists of two neural networks, a Generator, and a Discriminator, and is improved by training the two networks against each other[17]. Firstly, the parameters of the Generator and Discriminator are initialized, and the Generator is used to sample from the standard normal distribution and generate the fake data. Afterward, the real data (obtained from the training set) and the fake data generated by the Generator are jointly fed into the Discriminator, and the Discriminator's classification results for both data are obtained. The loss function is first applied in the process of using the discriminator, to make the discriminator discriminate between real and fake data as accurately as possible. The Discriminator and Generator of GAN have different loss functions. For the Discriminator, we aim to raise the output of the classification result for real data close to 1, while for the Generator, we aim to maximize the probability that the Discriminator considers the generated data as real data. When training is complete, generating new data will only require the use of a Generator. New data can be generated by first sampling the noise from a standard normal distribution and then feeding it into the Generator.

The structure of VAE and GAN can be found in Fig 2.1 and Fig 2.2.

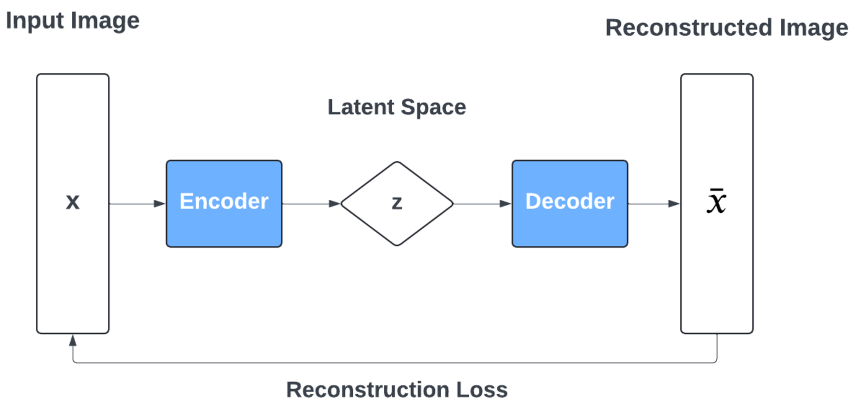


Figure 2.1: Structure of VAE

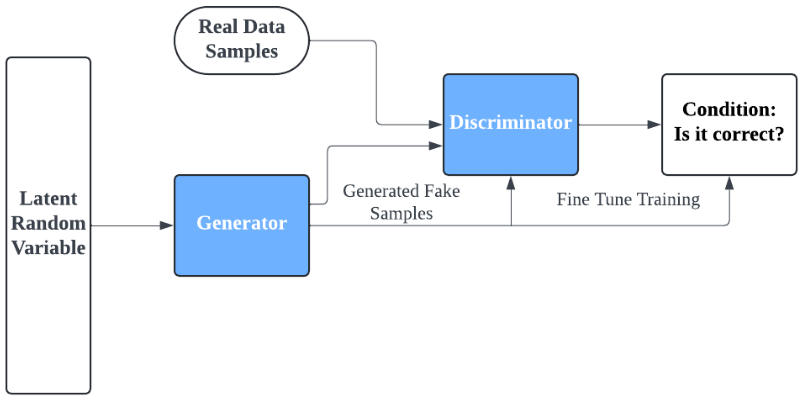


Figure 2.2: Structure of GAN

2.3 Incorporate Arbitrage-free Conditions into Machine Learning Models

In past research, several methods have been proposed on how to integrate arbitrage-free conditions with neural networks, and they fall into two main categories. One is to fundamentally change the structure of the neural network to control the generated results and it can be called adding hard constraints, and the other one is to impose soft constraints by adding penalty terms to the loss functions during training.

2.3.1 Hard Constraints

The first method to guarantee arbitrage-free conditions is adding hard constraints to the architecture of neural network[18].

Typically, we use fully connected neural networks. However, when considering adding hard constraints, it's necessary to modify the neural network's structure. In a sparse neural network, each neuron is connected to only a few neurons in the subsequent layer, rather than all of them.

As shown in Fig 2.3, the first input variable, T, is only connected to the five left-most hidden nodes, and the second input variable k is only connected to the five right-most hidden nodes. Such design enables separate sub-networks within a single layer, each focusing on one input variable, allowing for the imposing of distinct shape constraints relevant to each variable, such as non-negativity and convexity.

In the real case, we consider parameterized maps: $p = p_{\mathbf{W},\mathbf{b}}$

$$(T,k) \ni \mathbb{R}_+^2 \xrightarrow{p} p_{\mathbf{W},\mathbf{b}}(T,k) \in \mathbb{R}_+, \tag{2.1}$$

given as deep neural networks with two hidden layers. We use the following form to show the composition of this neural network:

$$p_{\mathbf{W},\mathbf{b}}(x) = f^{(3)}_{W^{(3)},b^{(3)}} \circ f^{(2)}_{W^{(2)},b^{(2)}} \circ f^{(1)}_{W^{(1)},b^{(1)}}(x), \tag{2.2}$$

where

$$\mathbf{W} = \left(W^{(1)}, W^{(2)}, W^{(3)}\right) \text{ and } \mathbf{b} = \left(b^{(1)}, b^{(2)}, b^{(3)}\right)$$

are weight matrices and bias vectors. And $f^{(l)} := \varsigma^{(l)}\left(W^{(l)}x + b^{(l)}\right)$ are semi-affine transformations for each layer, where non-decreasing activation functions $\varsigma^{(l)}$ applied to their argument componentwise.

As we mentioned before, the neural network is sparsely connected with two inputs T and k. The transformation function then can be written in the following form:

$$f^{(1)}_{W^{(1)},b^{(1)}}(x) = \left[f^{(1,T)}_{W^{(1,T)},b^{(1,T)}}(T), f^{(1,k)}_{W^{(1,k)},b^{(1,k)}}(k)\right] \tag{2.3}$$

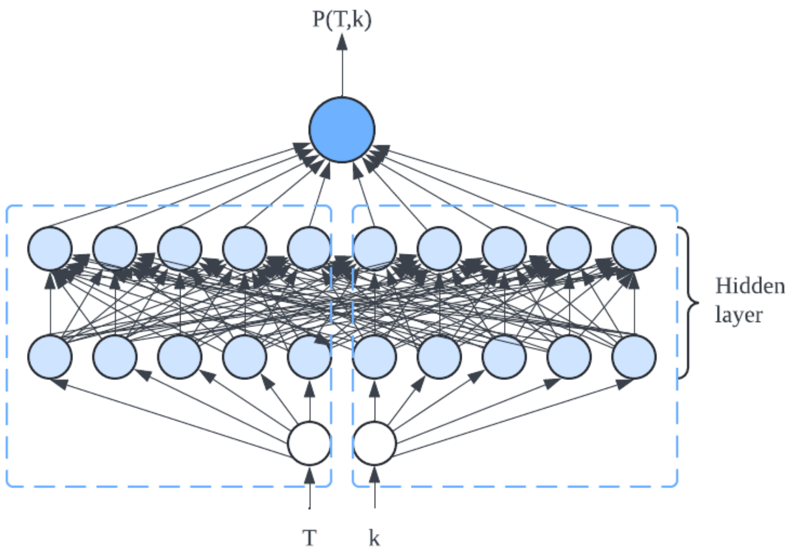


Figure 2.3: A Sparse Neural Network with One Hidden Layer.

where $W^{(1,T)}, b^{(1,T)}$ and $W^{(1,k)}, b^{(1,k)}$ correspond to parameters of sub-graphs for each input, and

$$f^{(1,T)}(T) := \varsigma^{(1,T)}\left(W^{(1,T)}T + b^{(1,T)}\right), f^{(1,k)}(k) := \varsigma^{(1,k)}\left(W^{(1,k)}k + b^{(1,k)}\right) \tag{2.4}$$

Then what we can do is to impose the shape constraints on the non-negative weights and activation functions.

2.3.2 Soft Constraints

In existing research on adding conditions to neural networks, many researchers focus on the loss function of deep learning models. One of the most common ideas is to modify the loss function of the neural network during the training process by adding the conditions as loss terms to the loss function.

For example, for the GAN model mentioned above, the Generator aims to generate samples that are similar to real data, while the Discriminator's goal is to distinguish between real and generated data. The adversarial game between the Generator(G) and the Discriminator(D) is formulated as the following minimax problem[17]:

$$\min_G \max_D L(D, G) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))], \tag{2.5}$$

where:

- x: Real data sample.
- z: Noise vector from the latent space, sampled from a simple distribution such as uniform or Gaussian distribution.
- $p_{\text{data}}(x)$: Distribution of real data.
- $p_z(z)$: Distribution of noise vectors.
- $G(z)$: Data sample generated by the generator.
- $D(x)$: Discriminator's output for input data x, representing the probability that x is a real data sample.
- $D(G(z))$: Discriminator's output for the generated data sample $G(z)$.

During the training process, the Discriminator D tries to maximize this value function while the Generator G aims to minimize this value function to maximize the probability that the Discriminator classifies generated data as real.

The standard form of the loss function is as follows:

1. **Discriminator Loss Function:**

$$L_D = -\mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] - \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.6}$$

2. **Generator Loss Function:**

$$L_G = -\mathbb{E}_{z\sim p_z(z)}[\log D(G(z))] \tag{2.7}$$

To generate an arbitrage-free implied volatility surface, we can incorporate penalty terms into the Generator's loss function. These penalty terms include constraints on calendar spread arbitrage, butterfly arbitrage, and large-moneyness behavior. In the subsequent sections, we will explain these conditions in detail. Additionally, a negative log-likelihood loss term is included to ensure that the generated data distribution is similar to the target data distribution. The improved generator loss function can be found in [1]:

$$L_G = \text{MSE}(y, G(X)) + \lambda_1 L_c + \lambda_2 L_{bf} + \lambda_3 L_\infty + \lambda_4 L_{D_G}, \tag{2.8}$$

where:

- $\text{MSE}(y, G(X)) = \frac{1}{b}\sum_{i=1}^{b}(y_i - G(X_i))^2$: Mean Squared Error, used to minimize the error between the generated data and the real data.
- L_c: Calendar spread arbitrage penalty term.
- L_{bf}: Butterfly arbitrage penalty term.
- L_∞: Large-moneyness behavior penalty term.
- $L_{D_G} = -\frac{1}{b}\sum_{i=1}^{b}\log D(G(z))$: Negative log-likelihood loss, used to ensure the generated data distribution is similar to the target data distribution.

In the improved loss function, the adversarial training is maintained through the negative log-likelihood loss L_{D_G}, which ensures that the distribution of the generated data is similar to the target data distribution. The numerical accuracy optimization is achieved through the MSE loss, which minimizes the error between the generated data and the real data. The arbitrage-free conditions are imposed through the penalty terms L_c, L_{bf}, and L_∞, ensuring that the generated data meets the arbitrage-free conditions of the financial market.

The weights $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are used to adjust the importance of the different constraint conditions.

Then the generator not only can produce realistic data but also ensure that the generated data meets the arbitrage-free conditions.

2.4 Diffusion Models to Generate Data

Diffusion models are a class of generative models that generate data by gradually adding and removing noise. The model has a good performance in generating high-quality image samples and also has a good generative efficiency.

The diffusion model contains two main processes, the forward process and the reverse process. The forward process (diffusion process) gradually adds Gaussian noise to the data. This process is usually defined by stochastic differential equations(SDEs). At the same time, the reverse process removes the noise and generates data from the noise, described by the backward time SDE. Researchers have developed different methods to optimize these models, producing variants such as DDPMs and SGMs.

DDPMs (Denoising Diffusion Probabilistic Models) reverse the forward diffusion process by learning denoising step by step. Training objectives include minimizing the lower bound of evidence (ELBO) of the data likelihood. This approach allows the model to improve its predictions incrementally, starting with pure noise and removing it step by step to generate real data[19].

SGMs(Score-Based Generative Models) generate data by estimating the score (i.e., the gradient of the log probability density) of the noisy data at various noise levels and using this score to gradually denoise. It is shown that this method provides a powerful framework for generative modeling, particularly when combined with neural networks to approximate the score function. By using the score function, SGMs can effectively guide the denoising process, resulting in high-fidelity samples[20].

DDPMs have demonstrated significant versatility and effectiveness in various domains[21]. In computer vision, DDPMs have succeeded in tasks such as image generation, inpainting, restoration, translation, and editing. These models enhance image quality by increasing resolution, filling in missing parts, and translating or editing images, with methods like SR3 and CDM using DDPMs for high-quality image generation[22]. Additionally, DDPMs have been applied to semantic segmentation, enhancing the utilization of labels by using high-level semantic information[23]. In video generation, DDPMs improve the quality and temporal consistency of generated frames[24]. They also solve challenges in 3D representation by inferring missing parts of point clouds, which is crucial for tasks like 3D reconstruction and augmented reality[25]. In the field of natural language processing (NLP), DDPMs are employed in text generation tasks, including character-level text generation and controlling complex sentence attributes. Models like Diffusion-LM utilize DDPMs for hierarchical and continuous latent representations, enabling complex control over text generation[26]. Interdisciplinary applications of DDPMs include generating synthetic financial data and modeling stock prices in finance, assisting in medical image reconstruction and drug design in healthcare by generating realistic synthetic data, and supporting scientific research in molecular and material design by helping to create new molecular structures and predict their properties.

2.5 Advantages Compared to GAN, VAEs

Compared with GAN (Generative Adversarial Networks) and VAE (Variational Autoencoders), DDPM(Denoising Diffusion Probabilistic Models) is currently seen as a model that generates more stable and high-quality results, especially in generating images. Both GAN and VAE face several challenges during the training process, including mode collapse, convergence problems, and KL(Kullback-Leibler divergence) vanishing. Here is a detailed explanation of these issues and their causes.

2.5.1 Problem of GAN

Mode Collapse:
Mode collapse for the GAN model happens when the generator learns a mapping of many input values to only a few outputs within the data space[27]. When facing this problem, the Generator mostly produces samples without diversity, focusing on just a few modes of the training data distribution.

There are several reasons causing mode collapse:

- **Instability in Adversarial Training**: The training of a GAN is considered to be an adversarial process between the Generator and the Discriminator. Simply, within this adversarial process, the Generator attempts to produce realistic samples that it could use to deceive the Discriminator, while the Discriminator aims to distinguish between real data and generated samples. In this process, the Generator might find an easy way of generating a very limited number of samples that are enough to fool the Discriminator.
- **Strong or Weak Discriminator**: A well-fit Discriminator is very important. If the Discriminator is too strong, the Generator will have a hard time learning an effective strategy for generating samples and might end up focusing on generating simple samples. In this way, the adversarial process is meaningless. If the Discriminator is too weak, it will allow the Generator to use samples to fool it quite efficiently, which causes the latter to stop learning and further results in mode collapse.

Convergence Problem:
Another problem that arises during GAN training is the convergence problem. This problem comes from the fact that, during the training process, a proper equilibrium point is not set, i.e., the losses of the Generator and the Discriminator do not continuously decrease or eventually reach an equilibrium state. Under this circumstance, when the Discriminator finally reaches the optimal state, the gradient of the Generator will disappear, which means it will affect the training performance of the Generator and model convergence performance[28].

The following are the specific reasons:

- **Unbalanced Adversarial Process**: GAN training is a min-max problem. If one of the Generator and Discriminator learns too fast or too slow, then it may cause the gradient of another one to vanish or explode, preventing convergence.
- **Local Optima and Saddle Points**: The loss functions of the Generator and Discriminator are non-convex, which makes it possible for the training process to fall into local optima or saddle points, leading to convergence problems.

2.5.2 Problem of VAE

In a VAE, we always aim to maximize the log-likelihood of the data $\ln p(\mathbf{x})$. Since directly computing this log-likelihood is difficult, we introduce a variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ to approximate the true posterior $p(\mathbf{z} \mid \mathbf{x})$. Here, the parameter ϕ represents the weights and biases of this neural network. Then, through the following decomposition, we obtain the ELBO(Evidence Lower Bound) form of VAE:

$$
\begin{aligned}
\ln p(\mathbf{x}) &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x})] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})}\right] \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x} \mid \mathbf{z}) - \ln \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} + \ln \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z} \mid \mathbf{x})}\right]
\end{aligned}
\tag{2.9}
$$

Through decomposition, we obtain:

$$
\ln p(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})] + \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})], \tag{2.10}
$$

where $\mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})] \geq 0$(from its definition).The last component of the expression of $\ln p(\mathbf{x})$, $\mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z} \mid \mathbf{x})]$,is a gap between ELBO and the true log-likelihood. Then we get the ELBO:

$$
\mathrm{ELBO} = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})], \tag{2.11}
$$

During VAE training, we maximize the ELBO to approximately maximize the log-likelihood $\ln p(\mathbf{x})$.

The second term of ELBO, the KL divergence $\mathrm{KL}[q_\phi(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})]$ represents the difference between the variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ and the prior $p(\mathbf{z})$. There exists a potential problem with VAE in this term.

Posterior collapse may happen when the decoder is so powerful that it can generate samples directly from the observed data $\mathbf{x}$ without relying on the latent variable $\mathbf{z}$[29]. This causes $q_\phi(\mathbf{z} \mid \mathbf{x})$ to become very similar to the prior $p(\mathbf{z})$, resulting in the KL divergence term becoming very small, even approaching zero.

The posterior collapse in VAE leads to several negative impacts on the model's performance:

1. **Loss of Information in Latent Variable z:** When posterior collapse occurs, the variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ converges to the prior $p(\mathbf{z})$, which means that the information in the latent variable $\mathbf{z}$ is almost lost. As a result, the latent variable distribution no longer carries important information about the data $\mathbf{x}$ itself, losing the representation capability of the latent space.

2. **Decrease in Reconstruction Quality:** Due to the ignored information of the latent variable $\mathbf{z}$, the model's reconstruction quality of the input data $\mathbf{x}$ may decrease. Although a powerful decoder in VAE can compensate for this loss of information to a certain extent, the decoder is also unable to fully capture the complex structure of the data in this case due to the loss of some information, which leads to a decrease in the quality of the generated samples.

2.5.3 Advantages of DDPM

Compared with the two models mentioned above, since DDPM uses ELBO as an explicit target for training, it rarely suffers from sudden crashes(such as mode collapse) during the training process. It also has better performance in terms of generating efficiency and sample quality.

- **High-quality Image Generation:**

  The diffusion model gradually generates images from noise by denoising. Since its denoising process is not a one-time process, but a multi-step step-by-step denoising process, the granularity of the generation process can be controlled during the noise decrease process. By controlling the gradual reduction of noise and maintaining the details of the image, finally, it can generate high-quality images. The performance of DDPM is usually superior to VAE, and in some cases can even be comparable to the quality of the image generated by the GAN.

- **Scalability and Parallelizability:**

  The framework design of the diffusion model allows it to adapt to datasets of different sizes and resolutions, from common MNIST datasets to complex color images, all of which can be generated at different quality by adjusting the denoising steps. Although the denoising process requires multiple steps, the computation of each step is not quite complex and can be parallelized(i.e., set different batch sizes during generation) to improve efficiency.

2.6 Incorporate Arbitrage-free Conditions with DDPM

DDPM generates data by gradually denoising random noise. Conditional DDPM extends this by incorporating additional information during both the training and generation process to guide the generated samples toward the required conditions:

- **Training Process:** During the training process, the model will accept the noisy data. Under different conditional DDPMs, the training process of DDPM is different. In general, the loss function needs to be adjusted to minimize the difference between the prediction noise and real added noise.

- **Generation Process:** During the generation process, the model generates data conditioned on the provided information or constraints, ensuring that the generated results meet the requirements.

In our study, incorporating arbitrage-free conditions within DDPM is important for generating realistic implied volatility surfaces. Recent research in classifier guidance and classifier-free guidance DDPM methods provide workable frameworks for adding these arbitrage-free conditions into diffusion models.

2.6.1 Classifier Guidance in DDPM

Classifier-guided DDPM uses a pre-trained classifier to guide a pre-trained diffusion model to a specified result. This methodology has been investigated in various scenarios, such as generating images with different labels, and has been verified to yield outcomes that beat those of GAN models in multiple situations, particularly in terms of the outstanding performance and effectiveness of the generated images[6]. To integrate a trained classifier with a DDPM model, follow these steps:

1. **Prepare**: First train a diffusion model and a Classifier on the dataset. To satisfy our requirements in this research, the Classifier should be able to determine if a surface is arbitrage-free and output the corresponding penalty term.

2. **Classifier Function**: The Classifier evaluates each generated surface to determine whether it is free of arbitrage and produces penalties for any violations, such as butterfly arbitrage and calendar arbitrage conditions.

3. **Guiding the Generation Process**: Calculate the penalty term for arbitrage violations for image x_t throughout the generation process of the DDPM. This should be done at each denoising step. Use the gradient algorithm to compute the gradient of the penalty term.

4. **Modify Generation Process**: Modify the generation process by incorporating the calculated gradient into the noise distribution, guaranteeing that the resulting surface minimizes arbitrage violations.

2.6.2 Classifier-Free Guidance in DDPM

Classifier-free guidance is another approach to conditional DDPM methods. In this approach, there is no requirement for an additional classifier to guide the DDPM to generate results in the desired direction. This method integrates conditions directly into the training process of the diffusion model[30].

1. **Joint Training Process:** When using classifier-free DDPM, the models are trained using a joint training method that includes desired conditions, such as specific image labels or financial constraints, as part of the training objective in this case. This involves teaching the model through a combination of condition and un-condition processes. The condition process will incorporate additional information, whereas the un-condition process does not.

2. **Conditional Generating:** The model is trained to provide predictions of denoised data based on noisy inputs. This is achieved by incorporating a desirable condition into the loss function. To satisfy the no-arbitrage requirements during the generation process, one approach is to use a parameter to adjust the output between conditional and unconditional.

3 Financial Options and Implied Volatility Surfaces

3.1 Mathematics on Financial options

Mathematical knowledge of financial options is fundamental to quantitative finance and financial engineering. Mathematics will be applied to model and predict the behavior of option prices and other derivatives under various market conditions. In this section, we will introduce some knowledge about basic financial mathematics to provide a basis for the no-arbitrage theory.

European Call/Put Options
A European option is a type of financial derivative that provides the holder with the exclusive right, but not the duty, to purchase or sell a specific amount of an underlying asset $S(t)$ at a predetermined price K on a specified future date T(i.e., maturity). European options differ from American options in that they can only be exercised on the expiration date.

Black-Scholes Option Pricing Model

The Black-Scholes model is a mathematical equation, specifically a partial differential equation, that governs the price of an option [11]. The model assumes that the price of the underlying asset follows a geometric Brownian motion, with a fixed risk-free rate of r and a constant volatility of σ.

In the Black-Scholes model, the value V of an option is expressed as:

$$V = BS(\sigma, S_t, K, T - t, r), \tag{3.1}$$

where:

- S_t is the current price of the underlying asset.
- K is the strike price of the option.
- $T - t$ is the time remaining until expiration.
- r is the risk-free interest rate.
- σ is the volatility of the underlying asset's returns.

Because of its simplicity, the Black-Scholes model is widely used around the world. The original model makes assumptions about four different aspects, including risky assets, risk-free assets, options, and financial markets[31].

1. **Assumptions about the Risk Asset**
    - Random Walk: Asset prices follow a random walk with no predictable pattern.

- Constant Volatility: The volatility of an asset's return remains constant over time.
- Normal Distribution of Returns: Asset returns are normally distributed.
- No Dividends: The underlying asset does not pay any dividends during the life of the option.

2. **Assumptions about the Risk-free Asset**
   - Fixed risk-free rate:The risk-free rate remains constant and is known in advance.
3. **Assumptions about the Option**
   - European Options: The model only applies to European options that can only be exercised at maturity.
4. **Assumptions about the Market**
   - No Transaction Costs: There are no costs associated with the purchase or sale of an asset or option.
   - Excellent liquidity: The market is extremely liquid, allowing any size of transaction without affecting the price.
   - No Short Selling Restrictions: There are no restrictions on short selling of the asset in question.
   - No Arbitrage Possibilities: There are no risk-free arbitrage opportunities in the market.

The Definition of Black-Scholes Implied Volatility
While the Black-Scholes model assumes a constant volatility, actually in a real financial market, the volatility of an asset is not constant and varies over time. Implied volatility σ^* is the estimation of the asset's future volatility derived from the market price of its options. It can be calculated by inverting the Black-Scholes model:

$$\sigma^* = BS^{-1}(V^{mkt}, S_t, K, T-t, r), \tag{3.2}$$

where:

- V^{mkt} is the observed market price of the option.

This process involves using a numerical method such as the Newton-Raphson or other root-finding algorithms to solve for σ^*, equating the Black-Scholes model price to the observed market price of the option.

Black-Sholes Implied Volatility Surfaces
Implied volatility varies with different strike prices and maturities of the options. Then we can derive a three-dimensional graph from this variation, for example:

- The x-axis represents different strike prices K.
- The y-axis represents different times to maturity $T-t$.
- The z-axis represents the implied volatilities σ^*.

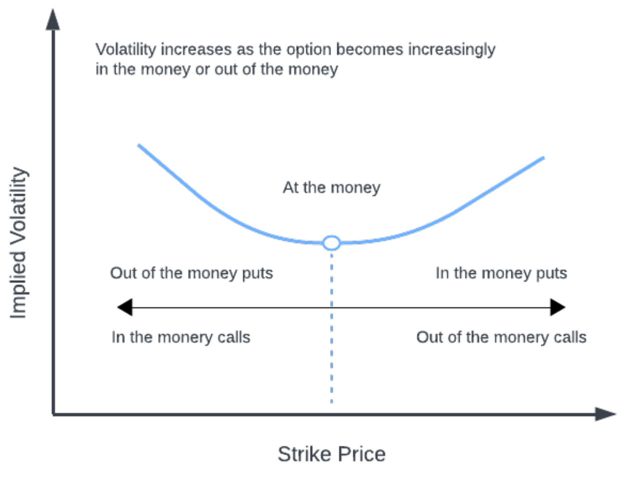


Figure 3.1: Volatility Smile

The shape of the implied volatility surface can provide important information about market conditions, such as the prediction of future economic events. Analysts and traders can study the implied volatility surface to estimate areas of high and low expected volatility, to assist in risk management.

There exist two typical features of implied volatility surface:

- Volatility Smile: For a call option, ”In the Money” means when its strike price is below the current market price of the underlying asset. ”Out of the money” means that the strike price is above the current market price, which means the option is worthless now. Both in-the-money and out-of-the-money options exhibit higher implied volatilities than at-the-money options as shown in Fig 3.1.
- Volatility Skew: At the same time of expiration, if the market is more worried about a certain direction (usually downside risk), the lower strike price may exhibit higher implied volatility, which means it will create a downward trend as shown in Fig 3.2.

The Stochastic Volatility Model
The Heston model, as proposed by Heston[32], is an important tool in financial mathematics. Unlike the traditional Black-Scholes model, which assumes constant volatility, the Heston model supposes that the volatility of the underlying asset follows a stochastic process. The Heston model is characterized by the following two stochastic differential equations:

1. **Underlying Asset Price Process:**

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S \tag{3.3}$$

2. **Variance Process**:

$$dv_t = \kappa(\hat{\theta} - v_t)dt + \hat{\sigma}\sqrt{v_t} dW_t^v \tag{3.4}$$

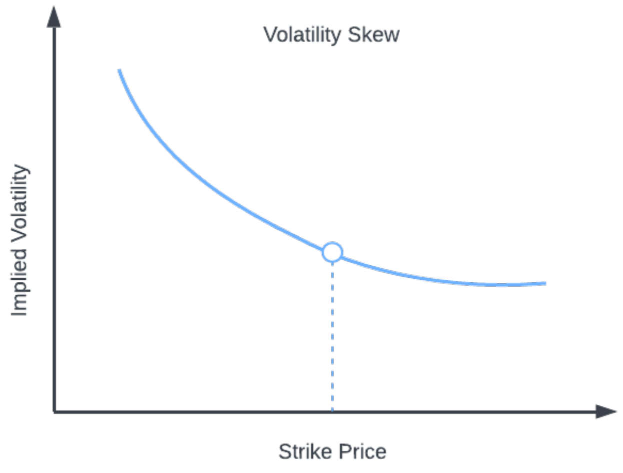


Figure 3.2: Volatility Skew

Where:

- S_t is the price of the underlying asset at time t.
- v_t is the instantaneous variance at time t.
- μ is the expected return of the asset.
- κ is the rate at which the variance reverts to its long-term mean $\hat{\theta}$.
- $\hat{\theta}$ is the long-term mean of the variance.
- $\hat{\sigma}$ is the volatility of the variance, also known as the ”vol of vol.”
- dW_t^S and dW_t^v are two Wiener processes with a correlation coefficient of ρ.

The output of the Heston model is the theoretical prices of call and put options, which reflect the value of the options given the set of input parameters mentioned before. Since the core of the Heston model is stochastic volatility, its analytical solution can be represented through a characteristic function. We can use Monte-Carlo or COS(Fourier-Cosine) methods to solve such a problem, see in [33].

3.2 Arbitrage-free Conditions for Option Price Surface

3.2.1 Option Price Surfaces from the Market and Abirtrage-free Conditions

When considering the arbitrage-free conditions, let us first define the price surface of a call option. The conditions of building an arbitrage-free call price surface are introduced in [34].

Definition 3.2.1. A call price surface parameterised by S(stock price) is a function:

$$C : [0, \infty) \times [0, \infty) \to \mathbb{R}$$
$$(K, T) \mapsto C(K, T)$$

along with a real number $S > 0$.

Arbitrage refers to positive probabilities of earning risk-free profits with a costless trading strategy. Arbitrage opportunity always happens when the market is not efficient. If there is a mispricing of options, there could be an arbitrage opportunity. Before generating the implied volatility surface, we begin with the arbitrage-free conditions for the call option price surface.

Definition 3.2.2. There is no static arbitrage in such a call price surface if there exists a non-negative martingale X on some stochastic basis $\left(\Omega, \mathcal{F}, F = (\mathcal{F}_t)_{t \geq 0}, P\right)$ with $C(K, T) = E\left((X_T - K)^+ \mid \mathcal{F}_0\right)$ for each $(K, T) \in [0, \infty) \times [0, \infty), X_0 = S$. If such a martingale and probability space exists, then we say that the call price surface is free of static arbitrage.

The following are the necessary and sufficient conditions for a call price surface to be free of static arbitrage:

Theorem 3.2.3. *Let $S > 0$ be a constant. Let $C : (0, \infty) \times [0, \infty) \to \mathbb{R}$ satisfy the following conditions:*

1. ***(Monotonicity in T)***
$$C(K, \cdot) \text{ is non-decreasing, } \forall K > 0;$$

2. ***(Convexity in K)***
$$C(\cdot, T) \text{ is a convex function, } \quad \forall T \geq 0$$

3. ***(Large strike limit)***
$$\lim_{K \to \infty} C(K, T) = 0, \quad \forall T \geq 0$$

4. ***(Bounds)***
$$(S - K)^+ \leq C(K, T) \leq S, \quad \forall K > 0, T \geq 0; \text{ and}$$

5. ***(Expiry Value)***
$$C(K, 0) = (S - K)^+, \quad \forall K > 0.$$

Then:
(i) the function

$$\widehat{C} : [0, \infty) \times [0, \infty) \to \mathbb{R}$$
$$(K, T) \mapsto \begin{cases} S, & \text{if } K = 0 \\ C(K, T), & \text{if } K > 0 \end{cases}$$

satisfies assumptions (1)-(5) but with $K \geq 0$ instead of $K > 0$; and
(ii) there exists a non-negative Markov martingale X with the property that

$$\widehat{C}(K,T) = \mathbb{E}\left((X_T - K)^+ \mid X_0 = S\right)$$

for all $K, T \geq 0$.

3.2.2 Condition: Monotonicity in T(Calendar Arbitrage Condition)

There are two common types of arbitrage strategy, calendar arbitrage and butterfly arbitrage.

For call options, calendar spread arbitrage is an options trading strategy that involves buying a long-term call option and selling a short-term call option at the same strike price. This strategy aims to profit from the difference in maturities between these two options. Due to the additional time value, the long-term call option always has a higher value than the short-term call option.

When the underlying asset price is at the strike price of the options at the expiration of the short-term option, traders can make maximum profits. Under this circumstance, the value of the long-term option will be higher due to its longer expiration time, while the short-term option's value will be near zero. From the necessary and sufficient conditions above, if a call price surface satisfies this condition, i.e. Monotonicity in T, we can say there exists no calendar spread arbitrage violations on this surface.

Here, we use $V_c(t_0, S_0, K, T)$ to denote the price of a call option with an underlying price S_0 at the current time t_0.

Then:

$$V_c\left(t_0, S_0; K, T + \Delta T\right) - V_c\left(t_0, S_0; K, T\right) > 0, \tag{3.5}$$

If we divide both sides of the equation by ΔT and we let $\Delta T \to 0$, then:

$$\begin{aligned}\lim_{\Delta T \to 0} \frac{1}{\Delta T} &\left[V_c\left(t_0, S_0; K, T + \Delta T\right) - V_c\left(t_0, S_0; K, T\right)\right] \\ &= \frac{\partial}{\partial T} V_c\left(t_0, S_0; K, T\right).\end{aligned} \tag{3.6}$$

For options with the same strike price K and underlying price S_0 at the current time t_0, the value of an option should increase as the time to maturity T increases, assuming all other factors remain constant.

3.2.3 Condition: Convexity in K(Butterfly Arbitrage Condition)

Here is another important arbitrage-free condition for option prices called the butterfly condition:

$$V_c\left(t_0, S_0; K + \Delta K, T\right) - 2V_c\left(t_0, S_0; K, T\right) + V_c\left(t_0, S_0; K - \Delta K, T\right) \geq 0, \tag{3.7}$$

This condition is based on the construction of a butterfly arbitrage strategy that consists of purchasing a call option with a strike of $K + \Delta K$, selling two call options with a strike of K, and purchasing a call option with a strike of $K - \Delta K$, all with expiry dates of T.

This requirement ensures that the overall return on these positions does not fall below zero, regardless of transaction costs. This is because the loss on calls sold at strike K is limited by the gain on calls purchased at strike $K + \Delta K$ and $K - \Delta K$.

3.2.4 Other Constraints

Condition: Large Strike Limit

$$\lim_{K \to \infty} C(K, T) = 0, \quad \forall T \geq 0$$

This condition states that as the strike price K approaches infinity, the option price $C(K, T)$ should approach zero. This requirement comes from the requirement of the intrinsic value of the option. In practice, as the strike price increases, the intrinsic value of the option (the asset price minus the strike price) becomes smaller, finally resulting in the option being worthless.

If this condition is not satisfied, the investor can sell an option at a positive price, but the option will never be exercised due to the high strike price.

Condition: Bounds

$$(S - K)^+ \leq C(K, T) \leq S, \quad \forall K > 0, T \geq 0$$

$(S - K)^+$ represents the intrinsic value of the option when exercised at maturity and S is the underlying asset's current price. This condition comes from the requirement of the no-arbitrage condition, ensuring that the option price must be reasonably distributed between its intrinsic value and the current price of the underlying asset.

Firstly, the option price can not be lower than its intrinsic value. Under this circumstance, an investor can purchase an option at a price below its intrinsic value and exercise the option immediately, making a risk-free profit.

Same, the option price can not be higher than the current price of the underlying asset. Under this circumstance, there will still exist an arbitrage opportunity. The investor can sell the option and use the proceeds to purchase the underlying asset, also making a risk-free profit.

Condition: Expiry Value

$$C(K, 0) = (S - K)^+, \quad \forall K > 0$$

At maturity, the value of the option is entirely determined by its intrinsic value. If this requirement is not satisfied at the expiration, an arbitrage opportunity will arise. For example, if the option price is higher than its intrinsic value, investors could sell the option and buy the asset in the market for a risk-free profit.

3.3 Transformation from Option Price Surface to Implied Volatility Surface

After calculating the option price using the Heston model or obtaining the option price from the actual financial market, the next step for calculating implied volatility is to invert the Black-Scholes formula to obtain the implied volatility for different strike prices and maturities. The Black-Scholes formula provides a solution for pricing European call options with the assumption of constant volatility, which means we can calculate a series of corresponding implied volatility values.

In the Black-Scholes formula, C represents the price of a European call option. Φ represents the cumulative distribution function (CDF) of the standard normal distribution. This function $\Phi(x)$ gives the probability that a standard normal random variable (with a mean of 0 and a standard deviation of 1) is less than or equal to x.

$$C(S, K, T, r, \sigma) = S_0\Phi(d_1) - Ke^{-rT}\Phi(d_2), \tag{3.8}$$

where d_1 and d_2 are defined as:

$$d_1 = \frac{\ln(S_0/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T},$$

Implied volatility σ is the volatility that matches the price of the Black-Scholes formula to the market option price. When solving for the Black-Scholes implied volatility, we need to use some numerical root-finding method to find the value of implied volatility that minimizes the difference between the Black-Scholes formula price and the market option price.

Finally, using the series of implied volatility values calculated by the B-S formula, we can construct the implied volatility surface. This is a three-dimensional image including implied volatility values, maturities, and strike prices.

3.4 Arbitrage-free Conditions for Implied volatility surface

3.4.1 Conditions

Then, we will turn to arbitrage-free conditions of the implied volatility surface. Also, let us define the implied volatility surface first:

Definition 3.4.1. Time-scaled implied volatility (in log-moneyness form) parameterized by S is a function defined by

$$\begin{aligned} \omega : \mathbb{R} \times [0, \infty) &\to [0, \infty] \\ (k, \tau) &\mapsto \sqrt{\tau}\sigma(S\exp(k), \tau), \end{aligned}$$

where σ is implied volatility and $S > 0$ is the stock price. k is the log-moneyness, i.e., $k = \ln(K/S)$.

Here, w is the total variance of $\sigma(k, \tau)$:

$$\omega(k, \tau) = \sigma^2(k, \tau)\tau.$$

Then, we have the theorem:

Theorem 3.4.2. *Let $S > 0$ and $\omega : \mathbb{R} \times [0, \infty) \to \mathbb{R}$. Let ω satisfy the following conditions:*

1. ***(Positivity)*** *For every $k \in \mathbb{R}$ and $\tau > 0$,*
$$w(k, \tau) > 0.$$
2. ***(Value at Maturity)*** *For every $k \in \mathbb{R}$,*
$$w(k, 0) = 0.$$
3. ***(Smoothness)*** *For every $\tau > 0$, $w(\cdot, \tau)$ is twice differentiable.*
4. ***(Monotonicity in τ)*** *For every $k \in \mathbb{R}$, $w(k, \cdot)$ is non-decreasing.*
5. ***(Durrleman's Condition)*** *For every $\tau > 0$ and $k \in \mathbb{R}$,*
$$0 \leq \left(1 - \frac{k\partial_k\omega}{\omega}\right)^2 - \frac{1}{4}\omega^2\left(\partial_k\omega\right)^2 + \omega\partial_{kk}^2\omega$$
, where w is shorthand for $w(k, \tau)$.
6. ***(Large Moneyness Behavior)*** *For every $\tau > 0$, $\sigma^2(k, \tau)$ is linear for $k \to \pm\infty$.*

Then the call price surface parameterized by s,

$$\widetilde{C} : [0, \infty) \times [0, \infty) \to \mathbb{R}, \quad (K, \tau) \mapsto C(K, \tau)$$

is free of static arbitrage. In particular, there exists a non-negative Markov martingale X with the property that

$$\widetilde{C}(K, \tau) = \mathbb{E}\left[(X_\tau - K)^+ \mid X_0 = S\right]$$

for all $K, \tau \geq 0$.

3.4.2 Explanations about the Conditions

In Theorem 3.4.2, Positivity (1) and Value at Maturity (2) conditions are necessary conditions that any sensible model must satisfy, as shown in [35]. Smoothness (3) is merely sufficient to prove an absence of arbitrage when conditions (4) to (6) are also satisfied. We will explain each condition in detail and find the connection between the price surface and volatility surface:

1. **Positivity:** This condition requires that for all k (log moneyness, $ln(K/S)$) and all positive times τ (maturities), the volatility function $\omega(k,\tau)$ must be positive. In financial markets, volatility represents the uncertainty when the price of the underlying asset moves. This condition shows a common sense in option pricing strategy that volatility should be non-negative.

   When connecting this condition to arbitrage-free conditions of call price surface, we will find that it corresponds to the lower bound(4) requirement of an option price. The positivity of implied volatility ensures that the price of an option will not fall below zero. Even if the volatility is very low(close to zero), which means there is almost no fluctuation in the price of the underlying asset, the option still has a certain time value due to the potential for price changes before expiration. Therefore, the market price of an option will at least reflect this time value.

2. **Value at Maturity:** This condition states that when $\tau = 0$ (at expiration), the volatility function $\omega(k,0)$ is equal to 0. This requirement indicates that the uncertainty about the movement of the option price disappears at expiration time because the value of the option will be equal to its intrinsic value.

   This condition also arises from the constraint on the option price imposed by the no-arbitrage requirement, and we can find a similar condition in 3.2.3. The expiration value of an option indicates that its value is entirely determined by its intrinsic value, which also reflects the fact that uncertainty disappears at expiration. This similarity for both option price and implied volatility illustrates that the transformation from price-surface to implied volatility-surface still maintains the same requirement for the value of the option at expiration.

3. **Smoothness:** The volatility function needs to satisfy second-order differentiability with respect to k to ensure the smoothness requirement for the volatility surface. The smoothness requirement is also found to be implicit in the previously mentioned no-arbitrage requirement for the price surface. According to the convexity condition for price surfaces (2), the convexity of K (strike price) requires that the option price as a function of the strike price K is convex. This means that for any two strike prices K_1 and K_2, and any $\lambda \in [0,1]$, the following inequality holds.

   $$C\left(\lambda K_1 + (1-\lambda)K_2, T\right) \leq \lambda C(K_1,T) + (1-\lambda)C(K_2,T), \tag{3.9}$$

   where $C(K,T)$ denotes the option price at expiration time T and strike price K.

   According to the definition of a convex function, if the function is convex, it is at least second-order derivable in its domain of definition. Also, this implies that the second-order derivative of the option price with respect to K is non-negative:

   $$\frac{\partial^2 C}{\partial K^2} \geq 0, \tag{3.10}$$

This property ensures the smoothness and continuity of the price surface with respect to the strike price.

4. **Monotonicity in τ:** The non-decreasing property of the volatility function $\omega(k,\cdot)$ over time reflects the fact that the uncertainty of the underlying asset's price fluctuation does not diminish as the expiry date approaches. Volatility is a measure of the magnitude of the underlying asset's price fluctuations, while time value is the excess of the option price over its intrinsic value, reflecting the potential profit from changes in the underlying asset's price before expiration.

   This requirement can still be traced back to the no-arbitrage requirement on the price surface. Compared with the monotonicity condition for T in 1, when the volatility function acts as a non-decreasing function, it implies that the uncertainty about the price movement faced by the option does not decrease as the time to expiration increases. This needs to be considered in conjunction with the nature of time value, where the longer the time to expire for an option the greater the chance of the underlying asset generating a favorable price movement. It also implies that the time value of the option will increase or at least not decrease.

   This condition also helps to prevent calendar spread arbitrage opportunities mentioned in 3.2.2 due to unnatural variations in volatility. also essentially uses the time value of options. This strategy involves buying and selling options with the same strike price but different expiration dates at the same time. The profitability of calendar spread arbitrage depends in part on the difference in volatility between options with different expiry dates. If volatility increases over time, rather than monotonically (i.e., violating the monotonicity in the τ condition), then by choosing the right combination of expiry dates, a trader may create an arbitrage opportunity due to the mispricing of options with irregular volatility behavior.

5. **Durrleman's Condition:** This is a condition related to the curvature of the implied volatility surface that ensures that the shape of the volatility surface does not lead to arbitrage opportunities(corresponding to the butterfly arbitrage opportunity). The condition involves the first and second-order derivatives of the volatility function $\omega(k,\tau)$ with respect to k, as well as the volatility function itself. We will explain each part of this condition separately.

   - $\left(1-\frac{k\partial_k\omega}{\omega}\right)^2$ and $-\frac{1}{4}\omega^2\left(\partial_k\omega\right)^2$: This part represents the rate of change of volatility with respect to the log strike price. This component ensures that the volatility does not increase or decrease too quickly in response to changes in the log strike price, maintaining a certain degree of stability in the volatility change.
   - $\omega\partial^2_{kk}\omega$: This part represents the second-order derivative of volatility with respect to the log strike price, which reflects the curvature of the volatility curve with respect to the log strike price. This condition requires that the volatility surface has non-negative curvature (i.e., non-negative second-order derivatives) at each point of the log strike price k, which means that the volatility surface is required to be convex.

   The Durrleman condition ensures that the volatility surface is generally smooth and convex by combining the above conditions and requiring them to be greater than zero. This shape of the volatility surface helps to satisfy the no-arbitrage requirement in the options market because it eliminates the possibility of arbitrage through market price inconsistencies.

Returning to the previously mentioned condition on the price surface, the convexity of the price surface requires that the second-order derivative of the option price with respect to the strike price be non-negative for any two strike prices. As we mentioned earlier, if the price surface is not convex, it is possible to profit from a butterfly arbitrage strategy.

For the implied volatility surface, the Durrleman condition ensures the necessary convexity of the price surface derived from the volatility surface, eliminating arbitrage opportunities constructed through the butterfly arbitrage strategy.

6. **Large Moneyness Behavior:** Implied volatility exhibits asymptotic behavior when values are extreme. This means that in these areas (very high or very low strike prices), the market does not expect volatility to rise or fall suddenly, but to change in a predictable linear pattern. If implied volatility is very non-linear, it is easy to profit from the inconsistency of the price of out-of-the-money options and other extreme strike prices.

4 The Generative Diffusion Models

4.1 Mathematics of Diffusion Models

In this section, we will introduce the mathematical basis of DDPM, that is, how DDPM is guaranteed to be correct in its operation through SDEs.

4.1.1 Stochastic Differential Equations (Score SDEs)

We always describe the form of the diffusion model through stochastic differential equations(SDEs). They represent processes where data is gradually perturbed to noise through an SDE, and then this process is reversed to generate data from noise. We call this formulation Score SDE, and the key idea is to use the score function to guide the denoising process[21].

A Score SDE is determined by the following general form of an SDE:

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{4.1}$$

where $\mathbf{f}(\mathbf{x}, t)$ and $g(t)$ are the drift and diffusion functions of the SDE, and $\mathbf{w}$ is a standard Wiener process.

Forward and Reverse SDEs of DDPM and SGM

In the generative models, the forward process adds noise to the data and is represented by an SDE, while the reverse process removes noise and is also represented by an SDE.

Under specific conditions, Score SDEs can be transformed into the specific forms used in Denoising Diffusion Probabilistic Models(DDPM) and Score-Based Generative Models(SGM). These models can be viewed as different approaches to Score SDEs, depending on the different choices of the drift and diffusion functions.

For Score SDE to become an SGM, different drift and diffusion functions must be chosen. The forward SDE of SGM is:

$$\mathrm{d}\mathbf{x} = \sqrt{\frac{\mathrm{d}[\sigma(t)^2]}{\mathrm{d}t}}\,\mathrm{d}\mathbf{w}, \tag{4.2}$$

where $\sigma(t)$ is a time-dependent standard deviation. The reverse SDE is:

$$\mathrm{d}\mathbf{x} = -\sigma(t)\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\,\mathrm{d}t + \sqrt{\frac{\mathrm{d}[\sigma(t)^2]}{\mathrm{d}t}}\,\mathrm{d}\overline{\mathbf{w}}, \tag{4.3}$$

Here,$\overline{\mathbf{w}}$ denotes another independent standard Wiener process in the reverse process.

The forward SDE for DDPM is:

$$\mathrm{d}\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}\,\mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\mathbf{w}, \tag{4.4}$$

where $\beta(t)$ is a time-dependent noise schedule. And the reverse SDE for DDPM is:

$$\mathrm{d}\mathbf{x} = \left[-\frac{1}{2}\beta(t)\mathbf{x} - \beta(t)\nabla_{\mathbf{x}} \log q_t(\mathbf{x})\right] \mathrm{d}t + \sqrt{\beta(t)}\,\mathrm{d}\overline{\mathbf{w}}, \tag{4.5}$$

The continuous SDEs can be discretized into finite steps to form the DDPM framework. Here, the noise addition process is determined by the forward SDE and is discretized into a series of steps $q(\mathbf{x}_t \mid \mathbf{x}_{t-1})$. Each step adds Gaussian noise with a specific variance β_t.

Similarly, the reverse process is approximated by a learned Gaussian distribution $p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ in DDPM.

4.1.2 DDPM

In this section, we will mainly focus on DDPM. Generative diffusion models always represent a class of latent variable models characterized by the form $p_\vartheta(\mathbf{x}_0) := \int p_\vartheta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, \ldots, \mathbf{x}_T$ are latent variables sharing the same dimensionality as the observed data[36]. The combined distribution $p_\vartheta(\mathbf{x}_{0:T})$, referred to as the reverse process, is modeled as a Markov chain with Gaussian transitions that are learned from data, beginning with $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$:

$$p_\vartheta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T)\prod_{t=1}^{T} p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \quad p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\vartheta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\vartheta(\mathbf{x}_t, t)), \tag{4.6}$$

The forward process $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$, also called the diffusion process of the diffusion model is fixed as a Markov chain that gradually adds Gaussian noise into the data following a predetermined variance schedule $\beta_1, \ldots, \beta_T$:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}), \tag{4.7}$$

We will introduce the forward process and reverse the process in the following, as well as the structure of DDPM, see in Fig 4.1.

Forward Process

- **Definition**: $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ is a fixed Markov chain that gradually adds Gaussian noise to the data.

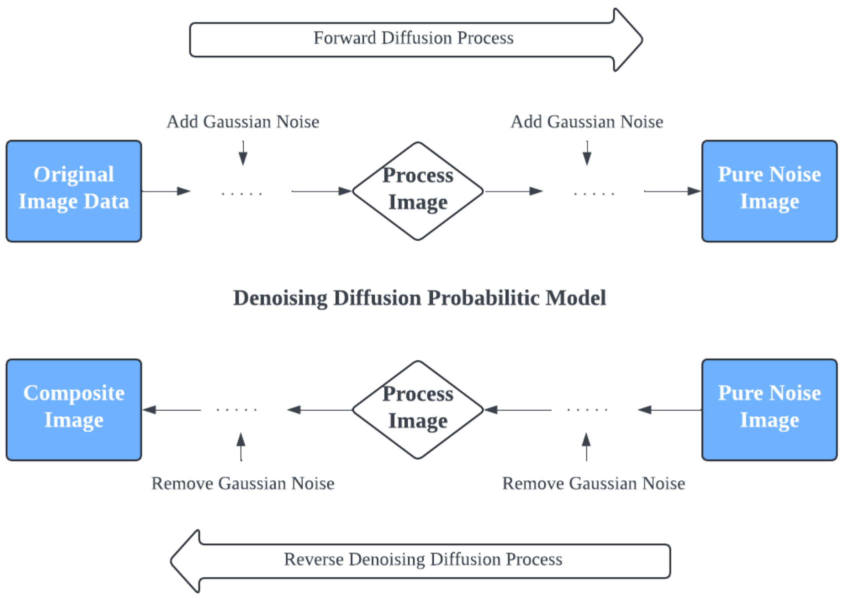


Figure 4.1: Structure of DDPM

- **Formula**:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{4.8}$$

- **Single Step in Forward Process**: Each step of this forward process can be written as:

$$\mathbf{x}_i = \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{z}_{i-1}, \quad \mathbf{z}_{i-1} \sim \mathcal{N}(0, \mathbf{I}), \tag{4.9}$$

  This equation indicates that at each step i, the new state $\mathbf{x}_i$ is a linear combination of the previous state $\mathbf{x}_{i-1}$ and Gaussian noise $\mathbf{z}_{i-1}$.

  We aim to demonstrate that this equation can be seen as a discretization of the Score SDE:

  Let $\Delta t = \frac{1}{N}$ and introduce an auxiliary noise level $\{\bar{\beta}_i\}_{i=1}^{N}$ where $\beta_i = \frac{\bar{\beta}_i}{N}$. Then:

$$\beta_i = \underbrace{\beta\left(\frac{i}{N}\right)}_{\bar{\beta}_i} \cdot \frac{1}{N} = \beta(t+\Delta t)\Delta t, \tag{4.10}$$

  where we assume $\bar{\beta}_i \to \beta(t)$ as $N \to \infty$.

  Also, define

$$\mathbf{x}_i = \mathbf{x}\left(\frac{i}{N}\right) = \mathbf{x}(t+\Delta t), \quad \mathbf{z}_i = \mathbf{z}\left(\frac{i}{N}\right) = \mathbf{z}(t+\Delta t), \tag{4.11}$$

Then we have

$$\begin{aligned}\mathbf{x}_i &= \sqrt{1-\beta_i}\mathbf{x}_{i-1} + \sqrt{\beta_i}\mathbf{z}_{i-1} \\ &= \sqrt{1-\frac{\bar{\beta}_i}{N}}\mathbf{x}_{i-1} + \sqrt{\frac{\bar{\beta}_i}{N}}\mathbf{z}_{i-1} \\ &= \sqrt{1-\beta(t+\Delta t)\Delta t}\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\mathbf{z}(t),\end{aligned} \tag{4.12}$$

For small Δt, we approximate:

$$\sqrt{1-\beta(t+\Delta t)\Delta t} \approx 1 - \frac{1}{2}\beta(t+\Delta t)\Delta t, \tag{4.13}$$

giving:

$$\begin{aligned}\mathbf{x}(t+\Delta t) &\approx \left(1-\frac{1}{2}\beta(t+\Delta t)\Delta t\right)\mathbf{x}(t) + \sqrt{\beta(t+\Delta t)\Delta t}\mathbf{z}(t) \\ &\approx \mathbf{x}(t) - \frac{1}{2}\beta(t)\Delta t\mathbf{x}(t) + \sqrt{\beta(t)\Delta t}\mathbf{z}(t),\end{aligned} \tag{4.14}$$

as $\Delta t \to 0$, we have

$$d\mathbf{x} = -\frac{1}{2}\beta(t)\mathbf{x}dt + \sqrt{\beta(t)}d\mathbf{w}, \tag{4.15}$$

- **Properties**: In the forward process, it is possible to sample $\mathbf{x}_t$ at any timestep t in closed form. Using the notation $\alpha_t := 1-\beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t}\alpha_s$, we have:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{4.16}$$

Reverse Process

- **Definition**:$p_\vartheta(\mathbf{x}_{0:T})$ is a Markov chain that starts from noise and gradually denoises it to recover the data.
- **Formula**:

$$p_\vartheta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T)\prod_{t=1}^{T} p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t), \quad p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\vartheta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\vartheta(\mathbf{x}_t, t)) \tag{4.17}$$

For the reverse process, it can also be proved to be seen as a discretization of Score SDE.

Training Objective:

In the training process of DDPM, we aim to maximize the log-likelihood $\log p_\vartheta(\mathbf{x}_0)$. Due to the computational difficulty of the posterior distribution and the non-negligibility of the log-likelihood, Variational Inference (VI) is finally used as an approximation method.

The Variational Inference approximates the posterior distribution of the latent variable by introducing a computable approximation distribution and indirectly maximizes the log-likelihood by optimizing the Evidence Lower Bound (ELBO). By maximizing the ELBO, the log-likelihood can be maximized and the problem of computing high-dimensional integrals can be avoided.

In DDPM, the ELBO can be expanded by the KL divergence(Kullback-Leibler Divergence) terms.

KL divergence measures the difference between two probability distributions[37]. For two probability distributions P and Q, it is defined as:

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_P\left[\log \frac{P(\mathbf{x})}{Q(\mathbf{x})}\right] = \int P(\mathbf{x}) \log \frac{P(\mathbf{x})}{Q(\mathbf{x})}\, d\mathbf{x} \tag{4.18}$$

The smaller the KL divergence, the more similar the two distributions are. When P and Q are identical, the KL divergence is zero.

The ELBO provides a lower bound on the log-likelihood, and by maximizing this bound, we indirectly maximize the log-likelihood:

$$\mathbb{E}[-\log p_\vartheta(\mathbf{x}_0)] \leq \mathbb{E}_q\left[-\log \frac{p_\vartheta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)}\right], \tag{4.19}$$

Expanding this, we get:

$$\mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\vartheta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right], \tag{4.20}$$

This can be further decomposed into KL divergence terms:

$$\mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\vartheta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)}_{L_{t-1}} \underbrace{- \log p_\vartheta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)}_{L_0}] \tag{4.21}$$

1. **KL Divergence at the Initial Step**:

$$D_{\mathrm{KL}}\left(q\left(\mathbf{x}_T \mid \mathbf{x}_0\right) \| p\left(\mathbf{x}_T\right)\right)$$

2. **KL Divergence at Each Time Step**:

$$D_{\mathrm{KL}}\left(q\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0\right) \| p_\vartheta\left(\mathbf{x}_{t-1} \mid \mathbf{x}_t\right)\right)$$

   Here, the conditional distribution $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0)$ is a Gaussian distribution that can be computed based on the parameters of the forward process:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}) \tag{4.22}$$

where,

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \tag{4.23}$$

and

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t \tag{4.24}$$

3. **Final Reconstruction Error Term**:

$$-\log p_\vartheta\left(\mathbf{x}_0 \mid \mathbf{x}_1\right)$$

From the above derivation, we can see that the ELBO of DDPM can be decomposed into a series of KL divergence terms and a final reconstruction term, which together form the theoretical training objective. However, in practice, the direct computation and optimization of these terms lead to excessive computational effort and complex implementation, especially because detailed computation of the Gaussian distribution is required at each time step.

To simplify the training process in real experiments, we introduce a simplified version of the loss function. This simplified loss function retains the principles of ELBO but represents it as a more efficient optimization objective. This simplification focuses on the mean square error (MSE) between the noise added to the data and the noise predicted by the neural network. Thus, we employ the following simplified loss function in place of the original ELBO for training the model:

$$L_{\text{simple}}(\vartheta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\vartheta\left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t\right)\right\|^2\right], \tag{4.25}$$

Where:

- $L_{\textbf{simple}}(\vartheta)$ is the simplified loss function. ϑ represents the set of network parameters.
- $\boldsymbol{\epsilon}$ is the Gaussian noise sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ added to $\mathbf{x}_0$ during the diffusion process.
- $\boldsymbol{\epsilon}_\vartheta(\mathbf{x}_t, t)$ is the noise predicted by the neural network.

The formula calculates the simplified loss function $L_{\text{simple}}(\vartheta)$, which is used to train the network by minimizing the mean squared error (MSE) between the noise predicted by the network $\boldsymbol{\epsilon}_\vartheta(\mathbf{x}_t, t)$ and the actual noise $\boldsymbol{\epsilon}$ added to the data.

4.2 Deep Learning Framework

Deep learning is a common approach to machine learning that uses neural networks to model complex relationships in data. And the core of deep learning is several neural network architectures. They are the base for advanced models, including those that will be used later in our research such as DDPM. We will provide a brief introduction to the underlying neural network architectures in this subsection, highlighting the CNN networks that will be used in DDPM.

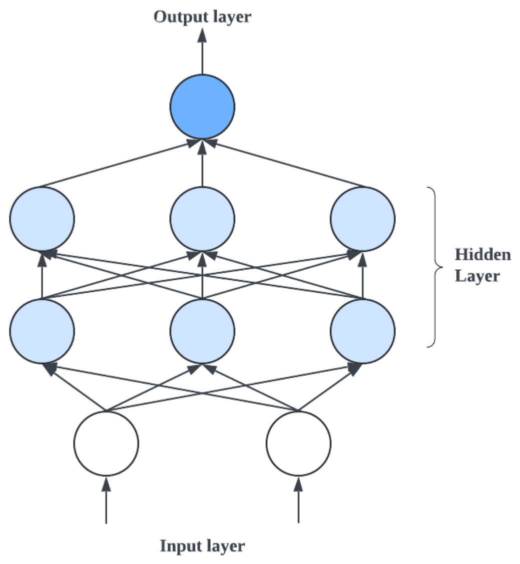


Figure 4.2: Structure of Fully Connected Networks

4.2.1 Fully Connected Networks(FC)

Fully connected networks (FC) are the simplest and most common type of neural network. These networks consist of multiple layers, where each neuron is connected to every neuron in the previous and subsequent layers as shown in Fig 4.2. FC can be applied to a wide range of tasks, from basic classification to more complex regression problems. A Multilayer Perceptron(MLP), which is often used in deep learning, is essentially a network consisting of multiple fully connected layers (FC layers). In other words, an MLP is a specific network structure.

However, the computational cost of FC can easily increase as the neuron or layer grows. The dense connection structure causes higher storage space and can result in FC not being able to have good efficiency and performance when dealing with complex tasks.

4.2.2 Convolutional Neural Network(CNN)

Convolutional neural networks(CNNs) differ from MLPs in that the neurons between the upper and lower layers are not all directly connected. CNNs use convolutional kernels as intermediate connections, and the core of this neural network is also a convolutional layer. Convolutional layers are used to extract features from images, which makes them particularly effective in visual tasks such as image classification, image segmentation, and image generation.

Its key components are:

- **Convolutional Layer:** The convolutional layer uses convolutional kernels to analyze the input data and extract features such as edges, texture, and color variations.

- **Pooling Layer:** The pooling layer reduces the amount of computation by reducing the resolution of the data and keeping the most important features.
- **Fully Connected Layer:** After convolution and pooling, features are usually classified or regressed through the fully connected layer.
- **Activation Functions:** Non-linear activation functions such as ReLU (Rectified Linear Unit) are often used after the convolution layer to increase the expressive power of the network.

4.2.3 Relationships between DDPM and Neural Networks

The Denoising Diffusion Probabilistic Model creates new data samples based on these underlying neural networks and mimics the distribution of the training data. As we mentioned before, there are two main processes in DDPM, the forward diffusion process and the reverse denoising process. The reverse process aims to remove the noise step by step with the help of a neural network that predicts how to subtract the noise from the image at each step.

To achieve the denoising process, DDPM relies on the U-Net architecture, which consists mainly of convolutional layers, as proposed in [38].

U-Net is a convolutional neural network architecture used in image processing tasks. It has an encoder-decoder structure with skip connections. In the encoding stage, CNNs extract multi-scale features and retain important feature information. In the decoding stage, CNNs reconstruct the image with learned features. Due to U-Net introduces skip connections that directly link each layer in the encoder to its corresponding layer in the decoder, it helps the network retain high-resolution information from earlier layers, improving the quality of image reconstruction.

4.3 Conditional Diffusion Models

4.3.1 Classifier-DDPM

The traditional DDPM mentioned before has been widely used in image generation. When returning to our topic, which is the generation of an arbitrage-free implied volatility surface, an improved DDPM method will be applied.

To solve some conditionally required generation problems, adding a classifier to DDPM is one of the improved methods, as mentioned in [6]. By pre-training the Classifier individually and then adding the Classifier to the already trained DDPM to guide the generation process, this process can adjust the result of generating samples.

The whole process of adding a Classifier to a trained DDPM can be summarized as follows and can also be found in the algorithm 1:

1. **Training the Classifier:**

   Considering the classifier first, the user needs to train a Classifier $p_\varphi(y \mid \mathbf{x_t})$, where $\mathbf{x_t}$ represents the surface and y indicates the label of the surface(e.g., in our research, it can indicate whether the surface is arbitrage-free or not). The Classifier is trained by

Algorithm 1 Classifier of DDPM

```
Input: Trained Classifier model $p_\varphi(y \mid \mathbf{x}_t)$, class label $y$, gradient scale $s$
Output: Generated sample $\mathbf{x}_0$
1 $\mathbf{x}_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
2 for $t \leftarrow T$ to 1 do
3     $\boldsymbol{\mu}, \boldsymbol{\Sigma} \leftarrow \boldsymbol{\mu}_\vartheta(\mathbf{x}_t), \boldsymbol{\Sigma}_\vartheta(\mathbf{x}_t)$
4     $\mathbf{x}_{t-1} \leftarrow$ sample from $\mathcal{N}\left(\boldsymbol{\mu} + s\boldsymbol{\Sigma}\nabla_{\mathbf{x}_t} \log p_\varphi(y \mid \mathbf{x}_t), \boldsymbol{\Sigma}\right)$
5 return $\mathbf{x}_0$
```

minimizing a loss function, which helps it to distinguish between surfaces or images with different labels.

2. **Using the Classifier to Guide the Diffusion Process:**

   After training the Classifier, the next step is to condition the original DDPM's reverse noising process on the label y. This is done by modifying the generation process. The unconditional reverse process is given by:

$$p_\vartheta(\mathbf{x}_t \mid \mathbf{x}_{t+1}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \tag{4.26}$$

   where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are the mean and covariance parameters predicted by the model.

   Then, combining the Classifier into this process, the conditional transition probability will be defined as:

$$p_{\vartheta,\varphi}(\mathbf{x}_t \mid \mathbf{x}_{t+1}, y) = Z \cdot p_\vartheta(\mathbf{x}_t \mid \mathbf{x}_{t+1}) \cdot p_\varphi(y \mid \mathbf{x}_t), \tag{4.27}$$

   where Z is a normalizing constant, this modification encourages the generation of samples consistent with the label y.

Now we will focus on the sampling process from $p_{\vartheta,\varphi}(\mathbf{x_t} \mid \mathbf{x_{t+1}}, y)$. First assuming that the term $\log p_\varphi(y \mid \mathbf{x_t})$ has low curvature compared to $\boldsymbol{\Sigma}^{-1}$. Then, we can obtain a Taylor expansion for $\log p_\varphi(y \mid \mathbf{x_t})$ at $\mathbf{x}_t = \boldsymbol{\mu}$:

$$\begin{aligned} \log p_\varphi(y \mid \mathbf{x}_t) &\approx \log p_\varphi(y \mid \mathbf{x}_t)\big|_{\mathbf{x}_t=\boldsymbol{\mu}} + (\mathbf{x}_t - \boldsymbol{\mu}) \nabla_{\mathbf{x}_t} \log p_\varphi(y \mid \mathbf{x}_t)\big|_{\mathbf{x}_t=\boldsymbol{\mu}} \\ &= (\mathbf{x}_t - \boldsymbol{\mu})\,\mathbf{g} + C_1 \end{aligned} \tag{4.28}$$

Here, $\mathbf{g} = \nabla_{\mathbf{x}_t} \log p_\varphi(y \mid \mathbf{x}_t)\big|_{\mathbf{x}_t=\boldsymbol{\mu}}$ and C_1 is a constant. Using this, the combined log-probability can be represented by approximation as:

$$\begin{aligned} \log\left(p_\vartheta(\mathbf{x}_t \mid \mathbf{x}_{t+1})\, p_\varphi(y \mid \mathbf{x}_t)\right) &\approx -\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \boldsymbol{\mu}) + (\mathbf{x}_t - \boldsymbol{\mu})\,\mathbf{g} + C_2 \\ &= -\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu} - \boldsymbol{\Sigma}\mathbf{g})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \boldsymbol{\mu} - \boldsymbol{\Sigma}\mathbf{g}) + \frac{1}{2}\mathbf{g}^T\boldsymbol{\Sigma}\mathbf{g} + C_2 \\ &= -\frac{1}{2}(\mathbf{x}_t - \boldsymbol{\mu} - \boldsymbol{\Sigma}\mathbf{g})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_t - \boldsymbol{\mu} - \boldsymbol{\Sigma}\mathbf{g}) + C_3 \\ &= \log p(\mathbf{z}) + C_4, \quad \mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\Sigma}\mathbf{g}, \boldsymbol{\Sigma}) \end{aligned} \tag{4.29}$$

Here, C_2,C_3 and C_4 are constants. These results show that the new mean is shifted by Σg, leading to:

$$\mathbf{x_t} \sim \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{\Sigma} g, \boldsymbol{\Sigma}). \tag{4.30}$$

This adjustment regulates the diffusion process on the label y by shifting the mean of the distribution.

How does this adjustment work in the application? During the generation process of DDPM, for each timestep t, the model will first predict the mean μ using the original trained DDPM. Then, this prediction can be adjusted by adding the Classifier, using the drift $\boldsymbol{\Sigma} g$, where g is the gradient of the Classifier's output with respect to the sample:

$$\mathbf{x_t} \sim \mathcal{N}(\boldsymbol{\mu} + s \cdot \boldsymbol{\Sigma} g, \boldsymbol{\Sigma}), \tag{4.31}$$

where s is a scaling factor that controls the influence of the Classifier's guidance.

After adding the classifier to DDPM, the gradient result of the Classifier is used to adjust the samples in each step of the generation process, and $\mathbf{x_{t-1}}$ generated in the new step is sampled from the normal distribution after the gradient is added.

During the whole generation process, this adjustment will be repeated at each step. Finally, the DDPM will be guided gradually by the drift Σg, towards conditional generated samples.

4.3.2 Classifier-free DDPM

Another conditional DDPM method is Classifier-free DDPM. Classifier-free guidance DDPM centers on conditioning the DDPM without the need for a separate trained Classifier. This approach jointly trains a diffusion model that can deal with both unconditional and conditional requirements in a single model[30]. That is, for this model, it is the training part rather than the generation part that mainly distinguishes it from the original DDPM.

The training process of Classifier-free DDPM includes training an unconditional denoising model $p_\vartheta(\mathbf{z})$ parameterized by a score estimator $\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda)$ together with a conditional model $p_\vartheta(\mathbf{z} \mid \mathbf{c})$ parameterized by $\boldsymbol{\epsilon}_\vartheta\left(\mathbf{z}_\lambda, \mathbf{c}\right)$. Here, $\mathbf{z}_\lambda$ represents the noisy data with a noise level λ, and $\mathbf{c}$ is the conditional information.

Then jointly train the unconditional and conditional models by randomly deciding whether to include the conditional information during the training process. With some probability p_{uncond}, the conditional information $\mathbf{c}$ is set to a null value $\varnothing$, which means:

$$\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda, \mathbf{c}) = \boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda) \quad \text{when} \quad \mathbf{c} = \varnothing \tag{4.32}$$

Then, during the generation process, a linear combination of both the conditional and unconditional score estimates arises:

$$\tilde{\boldsymbol{\epsilon}}_\vartheta(\mathbf{z}_\lambda, \mathbf{c}) = (1 + w)\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda, \mathbf{c}) - w\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda) \tag{4.33}$$

Here, w is a hyperparameter that controls the strength of the conditional information.

This equation combines the scores from the conditional and unconditional models. There is no classifier gradient in this expression, which means the generation process requires less computation. The algorithm of both the joint training process and generation process can be found in Algorithm 2 and Algorithm 3.

Algorithm 2 Joint training of a diffusion model with classifier-free guidance

Input: Probability of unconditional training p_{uncond}
Output: Trained diffusion model with classifier-free guidance

```
while model not converged do
    $(\mathbf{x}, \mathbf{c}) \sim p(\mathbf{x}, \mathbf{c})$
    $\mathbf{c} \leftarrow \varnothing$ with probability $p_{\text{uncond}}$
    $\lambda \sim p(\lambda)$
    $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    $\mathbf{z}_\lambda = \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon}$
    Take gradient step on $\nabla_\vartheta \left\|\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_\lambda, \mathbf{c}) - \boldsymbol{\epsilon}\right\|^2$
```

Algorithm 3 Conditional sampling with classifier-free guidance

Input: Guidance strength w, conditioning information $\mathbf{c}$, log SNR sequence $\lambda_1, \ldots, \lambda_T$ with $\lambda_1 = \lambda_{\min}$ and $\lambda_T = \lambda_{\max}$
Output: Generated sample $\mathbf{z}_{T+1}$

```
$\mathbf{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t \leftarrow 1$ to $T$ do
    $\tilde{\boldsymbol{\epsilon}}_t = (1 + w)\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\vartheta(\mathbf{z}_t)$
    $\tilde{\mathbf{x}}_t = \left(\mathbf{z}_t - \sigma_{\lambda_t}\tilde{\boldsymbol{\epsilon}}_t\right) / \alpha_{\lambda_t}$
    if $t < T$ then
        $\mathbf{z}_{t+1} \sim \mathcal{N}\left(\tilde{\boldsymbol{\mu}}_{\lambda_{t+1}|\lambda_t}(\tilde{\mathbf{x}}_t, \mathbf{z}_t), \left(\tilde{\sigma}^2_{\lambda_{t+1}|\lambda_t}\right)^{1-v} (\sigma^2_{\lambda_t|\lambda_{t+1}})^v\right)$
    else
        $\mathbf{z}_{t+1} = \tilde{\mathbf{x}}_t$
return $\mathbf{z}_{T+1}$
```

5 Methodology to Generate Arbitrary-free IVS

5.1 Implementation of the Arbitrage-free Conditions of IVS

Research now focuses more on how to generate implied volatility surface with the constraints of arbitrage-free to fit the real financial market. In this section, we would like to introduce methods guaranteeing arbitrage-free as soft constraints.

5.1.1 Define Loss Term

Adding penalty terms to the loss function to handle the arbitrage-free conditions in Theorem 3.4.2 is a common idea[35].

As we mentioned before, Positivity and Value at Maturity are satisfied by the design of the neural network. For smoothness, mapping ω is twice differentiable as long as the activation functions are twice differentiable. And Large Moneyness Behavior only needs to be considered when k approaches infinity. Hence, we only consider the last two conditions, Monotonicity in τ and Durrleman's condition as penalty terms. They correspond to the calendar condition and butterfly condition separately.

Let's recall the detailed description of these two requirements first. As mentioned in 3.4.2, to satisfy the calendar arbitrage-free condition, $w(k, \cdot)$ should be non-decreasing for every k. Then it can be transformed into a mathematical expression, that is:

$$0 \leq \partial_T \omega(k, \tau), k \in \mathbb{R} \tag{5.1}$$

Now we can define the violation value of this term. The violation value for Calendar condition at point (k_i, τ_i) on any implied volatility surface is defined as:

$$L_{\text{cal}}(k_i, \tau_i) = \max\left(0, -\partial_T \omega(k, \tau)\right), \tag{5.2}$$

Similarly, for the Butterfly condition, we need to consider Durrleman's condition, that is:

$$0 \leq \left(1 - \frac{k \partial_k \omega(k, \tau)}{\omega(k, \tau)}\right)^2 - \frac{1}{4}\omega(k, \tau)^2 \left(\partial_k \omega(k, \tau)\right)^2 + \omega(k, \tau) \partial_{kk}^2 \omega(k, \tau) \tag{5.3}$$

By transformation of this condition, it can be represented as[39]:

$$0 \leq \left(1 - \frac{k\partial_k \omega(k,\tau)}{2\omega(k,\tau))}\right)^2 - \frac{\partial_k \omega(k,\tau)}{4}\left(\frac{1}{\omega(k,\tau)} + \frac{1}{4}\right) + \frac{\partial^2_{kk}\omega(k,\tau)}{2} \tag{5.4}$$

Here, we will use $L_{\text{but}}(k_i, \tau_i)$ to denote the right hand side of the expression (5.4).

The violation value at point (k_i, τ_i) is defined as:

$$L_{\text{bf}}(k_i, \tau_i) = \max\left(0, -L_{\text{but}}(k_i, \tau_i)\right), \tag{5.5}$$

To evaluate the whole implied volatility surface rather than a single point, we need to evaluate the average violation value of all the points on the surface. The average violation values for both Calendar spread arbitrage and Butterfly arbitrage conditions are:

$$L_{\text{cal}} = 1/M \sum_{i=1}^{M} L_{\text{cal}}(k_i, \tau_i), \tag{5.6}$$

$$L_{\text{bf}} = 1/M \sum_{i=1}^{M} L_{\text{bf}}(k_i, \tau_i), \tag{5.7}$$

where M is the total number of points on the surface.

Here, for all k_i and τ_i(i.e, all log strike prices and maturities in implied volatility surface), we will calculate the sum of the maximum of $(0, -L_{cal}\,(k_i, \tau_i))$ and sum of the maximum of $(0, -L_{but}\,(k_i, \tau_i))$. If there is no arbitrage violation at the selected point (k_i, τ_i), L_{cal} and L_{but} should be both equal to 0, which means there is no penalty for calendar spread and butterfly arbitrage existing at this point.

5.1.2 Modify the Classifier

As we mentioned in 4.3, to guide the DDPM to generate images that meet requirements, a possible method is to add a Classifier to the trained DDPM. The purpose of this Classifier in our approach is to distinguish whether an implied volatility surface is arbitrage or not. After distinguishing it, the Classifier should be able to make guidance during the DDPM's generating process step by step. And the loss term of arbitrage condition we defined before will both apply in the distinguish process and guidance process.

Before the statements of our detailed design for the Classifier, we would like to define our Classifier. In the original paper on the Classifier-Guidance DDPM, the classifier was applied to classify images. Since the original literature aimed to generate different types of images, the output of the classifier is a probability distribution of the samples belonging to a certain class. However, in our experiments, the output of the Classifier is exact since whether an IVS is arbitrage-free or not can be discriminated by strict mathematical definition. We will treat the Classifier's output as the degree of arbitrage violation of an IVS, as guidance in the DDPM generation process.

The core of DDPM with a Classifier is to modify the mean value of the surface in the generation process. For some color images or images with detailed labels, the function of the Classifier is related to adding a label and making the mean value affected by that label when generating new images from this mean value. For our task, it is desired to generate the implied volatility surface from a mean value that includes a no-arbitrage condition. So during each step of the generation process, we would like to be able to give the mean value a quantity that is relevant to the arbitrage violation value calculated as mentioned in the previous section, so that the next generation step can be guided in the desired direction.

A common and possible approach is the gradient descent method. It is often used in deep learning to minimize an objective function (usually a loss function). However, in DDPM, the loss function is included in the training process and measures the difference between the predicted noise and the actual noise added. The term associated with it is noise but not the property of the surface itself, so it is difficult to add the no-arbitrage condition (i.e., concerns the property of the surface itself) to that loss function.

However, in DDPM's generation process, we can combine the gradient descent method with the function of arbitrage violation value.

Next, we will present a detailed implementation of this gradient approach, in which we will use some penalty terms and scaling parameters. To make it clearer how each of these terms affects the result of the generated surfaces, we will divide them into two main categories.

The first category is about the adjustment of the surface itself, including two loss functions.

Adjustment of the Surface itself:

- **Arbitrage-Free Loss Function**

  Here we use L_{arb} to denote the combination of both the calendar violation term in (5.6) and butterfly violation term in (5.7). The combined loss function is expressed as:

  $$L_{\text{arb}} = \alpha \cdot L_{\text{cal}} + \beta \cdot L_{\text{bf}}, \tag{5.8}$$

  where α, and β are weights of calendar loss term and butterfly loss term.

  The gradient of the arbitrage-free loss with respect to the generated surface, $\mathbf{x_t}$, denoted as $\nabla L_{\text{arb}}(\mathbf{x_t})$, is computed at each step during the generation process. This gradient is then used to adjust the model's predicted mean, guiding the generated surface toward an arbitrage-free direction at each step.

- **Smoothness Constraints**

  During the generation process, the addition of the Classifier's gradients increases the complexity of the whole model, and in some of the generated results we observe some surfaces that are not smooth.

  To further ensure the realism of the generated surfaces, smoothness constraints are also applied during the generation process. This involves adding a smoothness loss term based on the first derivatives with respect to strike price and maturity.

We use L_{smooth} to denote the smoothness loss term, and it is calculated as a weighted combination of the first derivatives of the surface with respect to strike price k and maturity τ:

$$L_{\text{smooth}} = c \cdot \sum \left| \frac{\partial \omega}{\partial \tau} \right| + d \cdot \sum \left| \frac{\partial \omega}{\partial k} \right|, \tag{5.9}$$

where c and d are weights for the derivatives with respect to τ and k.

Then, similar to the arbitrage-free loss function, the gradient of smoothness with respect to the generated surface, $\mathbf{x_t}$, denoted as $\nabla L_{\text{smooth}}(\mathbf{x_t})$, is used to adjust the model's predicted mean, ensuring smooth transitions on the generated surface.

In addition to the adjustment of the property for IVS, we also add some scaling parameters for the generation process. The second category is about control of the DDPM generation process.

Control of the DDPM Generation Process:

- **Time-Dependent Scaling Parameters**

  In a general gradient descent method, the learning rate is an important value to control the speed of convergence. In our research, we also use a similar method to control the whole generation process.

  To control the influence of these gradient-based adjustments, a time-dependent scaling function is applied when calculating the adjusted mean. This ensures that model adjustments can be human-controlled during the generation process. For example, we want the arbitrage violation calculating gradient to grow exponentially. This is because the surface is more like noise in most of the earlier steps of the generation process, and only in the last few steps does the surface have a clear shape features, as shown in Fig 5.1. The same happens for smoothness adjustments, where the condition of imposing smoothness on the surface in the early steps of generation may not affect the smoothness of the surface, but rather bring some negative impacts on the generation performance, which means the adjustment of the smoothness may happen at the last steps of generation process.

  The scaling factor $s(t)$ for arbitrage-loss function is computed using an exponential decay function:

$$s(t) = s_0 \cdot \exp(-\lambda \cdot t) \tag{5.10}$$

  where s_0 is the initial scaling coefficient, λ is the decay rate, and t represents the current timestep. Fig 5.2 describes the process of change in $s(t)$.

  Another scaling factor s_{tm} is introduced to describe the impact of smoothness gradients in the generation process. As we show in Fig 5.1, usually, in most of the previous generation steps, we do not need to consider adding the smoothness constraint to the image because of the large amount of noise in the image. Therefore, in the experiments, this parameter will be zero in most of the generation steps, and we will pick a suitable time step within that time step that will make this parameter non-zero.

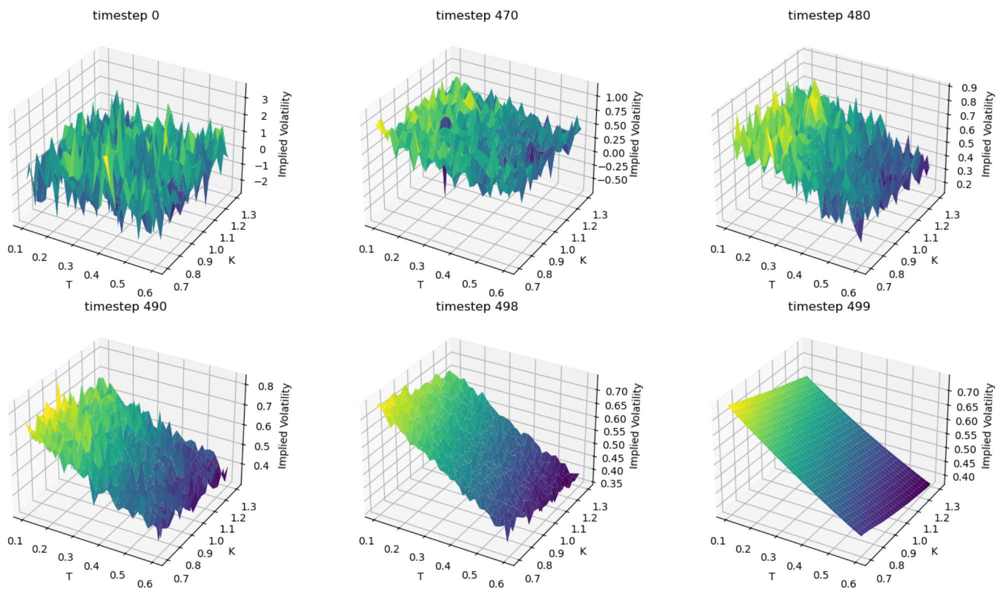


Figure 5.1: The 500 Timesteps of Implied Volatility Surface during Generation

- **Adjusted Mean Calculation**

  Then, the adjusted mean for the generation step is calculated by subtracting the scaled gradients from the model's predicted mean. This adjusted mean is used to generate the next step in the diffusion process:

$$\mu_{\text{adjusted}} = \mu_{\text{original}} - s(t) \cdot \nabla L_{\text{arb}}(\mathbf{x_t}) - s_{tm} \cdot \nabla L_{\text{smooth}}(\mathbf{x_t}), \tag{5.11}$$

  These adjustments help to improve the overall quality and reality of the surfaces generated, ensuring that they fulfill the expectations and theoretical conditions of the financial markets.

We can now add these terms together to the algorithm of DDPM. In our research, the algorithm of DDPM with a Classifier can be written as follows:

Algorithm 4 Classifier of DDPM (For Arbitrage-Free Condition)

Input: Trained Classifier model, gradient scale $s(t)$, and s_{tm}
Output: Generated sample $\mathbf{x_0}$ under arbitrage-free condition

```
22 $\mathbf{x_T} \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
23 for $t \leftarrow T$ to $1$ do
24     $\mu, \Sigma \leftarrow \mu_\theta(\mathbf{x_t}), \Sigma_\theta(\mathbf{x_t})$
25     $\mathbf{x_{t-1}} \leftarrow$ sample from $\mathcal{N}(\mu - s(t) \cdot \nabla L_{\text{arb}}(\mathbf{x_t}) - s_{tm} \cdot \nabla L_{\text{smooth}}(\mathbf{x_t}), \Sigma)$
26 return $\mathbf{x_0}$
```

From the flow of this algorithm, we can find the most significant modification part of our method, that is, for the denoising part of the DDPM generation process.

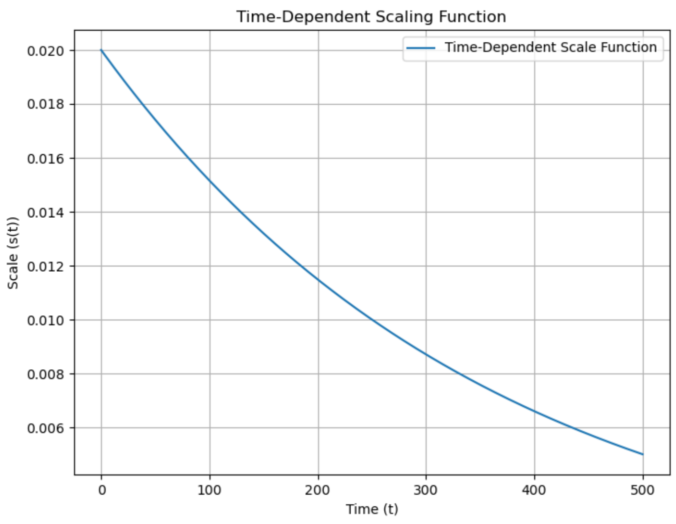


Figure 5.2: Scale Parameter of Arbitrage Calculating Gradient

5.2 Impact of the Loss Terms on IVS

In this section, we will discuss the impact of the loss terms(calendar arbitrage loss term, butterfly arbitrage loss term, and smoothness constraints) on the implied volatility surface.

Impact of L_{cal}, L_{bf} and Smoothness Constraint:

By penalizing violations of these two conditions, L_{cal} and L_{bf}, decrease the arbitrage violation of the IVS in both the strike price and maturity dimensions, preventing local discontinuities or hard variations in the volatility dimension. As for the smoothness constraints, a no-arbitrage IVS should be smooth and continuous. This constraint also helps decrease sharp points and irregular fluctuations on the IVS, ensuring a more realistic surface is generated.

Balancing Convergence and Loss Terms:

However, the introduction of these penalty terms also impacts the original generation process. The introduction of these penalty terms complicates the generation process. This is because the model needs to satisfy the no-arbitrage conditions while learning the features of the training dataset at the same time. Sometimes it is hard to keep the balance between the constraints and the performance of generated surfaces. In practice, proper adjustment of the loss weights is necessary to balance the degree of fulfillment of no-arbitrage conditions with the quality of the final generated surfaces.

6 Results

In this section, we will first validate the effectiveness of the proposed classifier in detecting arbitrage behavior in 6.1. Here, the validity will be verified using the IVS generated by the Heston model. In 6.2, we will provide detailed settings about our experiments and the influence of different weights of loss terms. Then, we will discuss the improved effect of the Classifier in 6.3. The discussion will include several parts: the effect of different parameters on the experimental results, the effect of the presence or absence of smoothness constraints on the generated results, and whether the arbitrage phenomenon of the DDPM is improved with the addition of the Classifier. After that, we will discuss the above results in the last section.

6.1 Test the Classifier

In this part, we aim to validate the reasonableness of the classifier by examining the implied volatility surfaces (IVS) generated under the Heston model. The classifier test includes two key conditions: calendar condition and butterfly condition. We will validate the IVS generated by the Heston model to ensure that the classifier works well and provide a detailed description of the methodology for calculating boundary points that arise during the calculation process.

6.1.1 Classifier Conditions

The classifier includes the following two conditions:

- **Calendar Condition:** For every $k \in \mathbb{R}$, $w(k, \cdot)$ is non-decreasing.
- **Butterfly Condition:** For every $\tau > 0$ and $k \in \mathbb{R}$,

$$0 \leq \left(1 - \frac{k\partial_k \omega}{\omega}\right)^2 - \frac{1}{4}\omega^2 \left(\partial_k \omega\right)^2 + \omega \partial_{kk}^2 \omega,$$

  where w is shorthand for $w(k, \tau)$.

6.1.2 Treatment of Derivatives and Boundary Points

During the calculation process of the classifier, several important treatments of derivatives are influencing the results heavily.

1. **First and Second derivatives:**

   We used the central difference form when calculating the derivatives in 5.6 and 5.7. The first-order derivative using the central difference method is approximated by:

   $$\tilde{f}'(x) =\approx \frac{\tilde{f}(x+h) - \tilde{f}(x-h)}{2h}, \tag{6.1}$$

   where h is the step size.

   The second-order derivative using the central difference method is approximated by:

   $$\tilde{f}''(x) \approx \frac{\tilde{f}(x+h) - 2\tilde{f}(x) + \tilde{f}(x-h)}{h^2}, \tag{6.2}$$

   where h is the step size.

2. **Calculation of Boundary Points:**

   Another important treatment is the calculation method of boundary points. When computing derivatives near the boundaries of the surfaces, we need to pay attention to maintain the accuracy of the calculation. For these boundary points, where the central difference method can not be directly applied due to the lack of neighboring points on one side, we need to use finite difference formulas that achieve second-order accuracy.

   For the first-order derivative at the boundary point x_0, the forward difference formula is:

   $$\tilde{f}'(x_0) \approx \frac{-3\tilde{f}(x_0) + 4\tilde{f}(x_1) - \tilde{f}(x_2)}{2h}, \tag{6.3}$$

   For the last point x_n in the domain, the backward difference formula is:

   $$\tilde{f}'(x_n) \approx \frac{3\tilde{f}(x_n) - 4\tilde{f}(x_{n-1}) + \tilde{f}(x_{n-2})}{2h}, \tag{6.4}$$

   For the second-order derivative at the boundary points, the forward difference at x_0 is given by:

   $$\tilde{f}''(x_0) \approx \frac{2\tilde{f}(x_0) - 5\tilde{f}(x_1) + 4\tilde{f}(x_2) - \tilde{f}(x_3)}{h^2}, \tag{6.5}$$

   And for the last point x_n, the backward difference is:

   $$\tilde{f}''(x_n) \approx \frac{2\tilde{f}(x_n) - 5\tilde{f}(x_{n-1}) + 4\tilde{f}(x_{n-2}) - \tilde{f}(x_{n-3})}{h^2}, \tag{6.6}$$

   These formulas aim to ensure that at the boundaries, the derivatives are approximated with second-order accuracy, to keep the overall accuracy of the calculation across the whole implied volatility surface.

6.1.3 Validation of Heston Model-Generated IVS

To test the validity of the classifier, we first apply it to the implied volatility surfaces generated by the Heston model. Due to the price surfaces generated by the Heston model being arbitrage-free[12], the implied volatility surfaces transformed by Black-Scholes are still arbitrage-free.

The Heston model used in this validation involves the following parameters:

- ρ: Correlation between the asset price and its variance
- κ: Rate of mean reversion
- v_0: Initial variance
- σ: Volatility of variance
- θ: Long-term variance
- r: Risk-free rate

The specific ranges for these parameters used in our tests are summarized in Table 6.1.

Parameter	Description	Range
ρ	Correlation coefficient	[-0.9, -0.1]
κ	Rate of mean reversion	[1.0, 2.0]
v_0	Starting volatility	[0.1, 0.3]
σ	Volatility of variance (Vol of Vol)	[0.1, 0.9]
θ	Long-term mean-variance	[0.1, 0.3]
r	Risk-free rate	[0, 0]

Table 6.1: Parameter ranges used in the Heston model for generating IVS.

The COS method is a numerical method applied in the calculation of the Heston model. We need to calculate the analytical solution of the characteristic function for the Heston model. First, select the convergence interval of the Fourier series expansion and represent the option price as the Fourier series expansion of the characteristic function. Then use the Fast Fourier Transform to calculate the sum of the Fourier series, and finally convert the Fourier series back to the option price.

Also, when using the COS method, we need to set parameters. Here, L represents the length of the truncation interval, which determines the accuracy of the Fourier series expansion. N denotes the number of terms in the series, which affects the resolution and precision of the resulting price calculation. The specific settings of these parameters are summarized in Table 6.2.

Parameter	Value
L	12
N	512

Table 6.2: The parameters of the COS method used for pricing in the Heston model

When generating the grids of implied volatility surfaces, we set up the maturity times T and strike prices K as shown in Table 6.3. The maturity times T range from 0.1 to 0.6, while the strike prices K range from 0.7 to 1.3. Both are uniformly distributed across 28 points along corresponding dimensions.

Parameter	Description	Range
T	Maturity (years)	[0.1, 0.6]
K	Strike Price	[0.7, 1.3]

Table 6.3: Settings for Maturity Times and Strike Prices in the Simulations.

We can then get the implied volatility surface based on the above setup, as shown in Fig. 6.1.

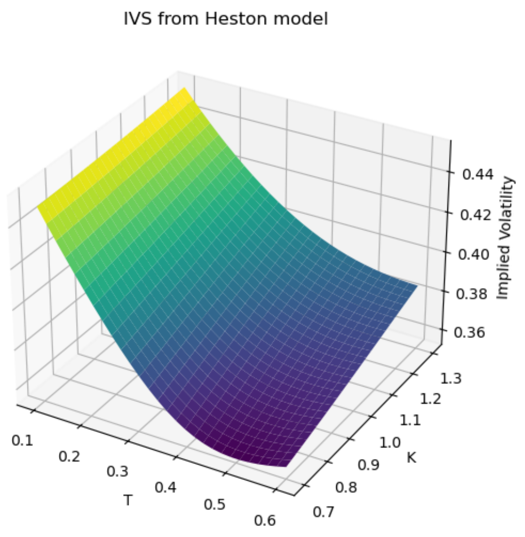


Figure 6.1: Implied volatility surface generated by Heston model

The size of our surface dataset generated by the Heston model is 10,000. To check our classifier, we will randomly choose 1,000 surfaces in the dataset as a test set.

In the testing process, we evaluate the surfaces based on two key arbitrage conditions as mentioned in 6.1.1: the Calendar Condition and the Butterfly Condition. These conditions are applied to every point on the surface, along the strike price and maturity dimensions.

The results of the test will be presented in two parts. On the one hand, it is the statistics of whether there is a violation in a single surface in the test set. On the other hand, it will be for whether there is a violation in the points on each single surface.

Violation Surfaces Assessment:

- **Calendar Condition:**

  For each point on the surface, we compute the condition $w(k, \cdot)$ for every strike k and ensure it is non-decreasing. Any violation (i.e., where the condition resulted in a value less than zero) will be recorded, and the absolute value of these violations will be taken.

- **Butterfly Condition:**

  We compute the butterfly condition for every strike k and maturity τ at each point on the surface. Any violation where the calculated value is less than zero will be recorded and its absolute value taken.

We then calculate the sum of violations for the entire surface and divide this violation value by the number of all points, treating it as the violation value for this surface.

The violation value for the Calendar condition at point (k_i, τ_i) is defined as we mentioned in (5.2).

Similarly, for the Butterfly condition, the violation value at point (k_i, τ_i) is defined as we mentioned in (5.5).

As we use M to denote the number of points on a single surface, here in our experiment, $M = 28 \times 28$.

Violation Points Assessment: In addition to calculating the average arbitrage violation value, we also assess whether each point on the single surface violated the conditions.

- **Boundary Violation:** We checked for violations at the boundaries of the surface grid (i.e., at the minimum and maximum strikes and maturities).
- **Central Violation:** Similarly, violations were checked within the central region of the grid (excluding the boundaries).

The number of points that violated each condition will be recorded in this experiment.

The result of the two tests is listed as shown in Table 6.4:

Violation Type	Surfaces with Violations	Surfaces without Violations	Total Surfaces Tested
Calendar Arbitrage	0 (0%)	1000 (100%)	1000
Butterfly Arbitrage	0 (0%)	1000 (100%)	1000
Boundary Violation	0	1000	1000
Central Violation	0	1000	1000

Table 6.4: Arbitrage and Point Violations for 1000 Tested Surfaces.

From the result, we can find that there is no arbitrage violation in the implied volatility surfaces generated by the Heston model, which means our classifier works well.

6.2 Experiment Setup

In this section, we will introduce the basic setting of the DDPM. Also, we will introduce the specific parameters used in our Classifier. Then we will provide the performance of DDPM with a Classifier under different parameter settings to find the most proper combination of parameters.

6.2.1 Data Description

First, we will select a trained DDPM. In this experiment, we use a dataset generated from the Heston model, containing implied volatility surfaces with various maturities and strike prices. The dataset consists of 50,000 implied volatility surfaces, each representing different market conditions. This dataset will be used as the training set for this DDPM.

6.2.2 Model Setup

The core model used in this experiment is a DDPM built with a U-Net architecture. The U-Net model consists of multiple convolutional layers, with channel multipliers that double the number of channels at each downsampling step[38].

The model was trained using a Gaussian diffusion process with 500 timesteps, and was proposed in [40]. The model will gradually denoise the input data and generate implied volatility surfaces during the whole process. The diffusion model was trained on a GPU-enabled device. Detailed model settings can be found in Table 6.5.

Content	Setup
Model Architecture	U-Net
Channels	96
Time steps	500
Optimizer	Adam
Learning Rate	0.0005
Device	GPU

Table 6.5: Model Configuration and Training Parameters

6.2.3 Incorporation of Classifier Gradients with Generation Process

In addition to the model base settings mentioned above, we also adjust the guidance DDPM with several loss terms. These adjustments include the integration of a Classifier to apply arbitrage-free conditions, the weight of the loss term, the guidance of smoothness, and the use of a time-dependent scaling function to adjust the influence of these gradients over time.

As we mentioned in the Methodology part, we will use the loss function (5.8) to evaluate the arbitrage-free loss value for both the calendar loss term and butterfly loss term for the implied volatility surface.

In our experiment, $\alpha = 1$, $\beta = 1$, which means we assign the same weight to L_{cal} and L_{bf} in the overall gradient calculation.

Also, for the smoothness loss function as shown in (5.9), we will set the weight parameters $c = 0.1$ and $d = 0.05$ in this experiment.

As for the scale function for the arbitrage loss function, $s(t)$ mentioned in (5.10) will gradually increase from 0.005(the initial scaling coefficient) to 0.02 for the whole generation process. Meanwhile, the scale parameter for smoothness will be equal to zero when the step is larger than 30 and equal to 0.001 at the last 30 steps.

Finally, our function for the adjusted mean in this experiment is:

$$\mu_{\text{adjusted}} = \mu_{\text{model}} - 0.005 \cdot \exp(-\lambda \cdot t) \cdot \nabla L_{\text{arb}}(\mathbf{x_t}) - \begin{cases} 0.001 \cdot \nabla L_{\text{smooth}}(\mathbf{x_t}), & \text{if } t \leq 30, \\ 0 \cdot \nabla L_{\text{smooth}}(\mathbf{x_t}), & \text{if } t > 30. \end{cases} \tag{6.7}$$

6.2.4 Influence of Weight Parameter in Arbitrage Loss Function

Before we make the comparison about the effectiveness of the classifier, we will go through the results of some parameter adjustments to illustrate why we have chosen the above parameter settings. In the following experiments, we will use **DDPM** to denote the Denoising Diffusion Probabilistic Model without a Classifier and use **DDPM-Classifier** to denote the model with the Classifier.

Let's recall the loss function first:

$$L_{\text{arb}} = \alpha \cdot L_{\text{cal}} + \beta \cdot L_{\text{bf}}, \tag{6.8}$$

where α, and β are weights of calendar loss term and butterfly loss term.

To illustrate the influence of these two loss terms, we first fix the weight for the calendar arbitrage loss term, setting $\alpha = 1$ and letting β gradually increase. Also, the smoothness constraint was included in this experiment to decrease the fluctuation on the implied volatility surface.

Here, we will use the average arbitrage loss value for both calendar arbitrage condition and butterfly arbitrage condition as evaluation metrics. The formulas (5.6) and (5.7) will be used in the calculations here. We generated 100 implied volatility surfaces under different parameter settings and listed the average for the loss value of all the surfaces as follows.

Parameter	$\alpha = 1$	
β	Calendar Arbitrage loss	Butterfly Arbitrage loss
0.1	9.84E-06	1.08E-01
0.3	2.15E-05	2.42E-01
1.0	4.99E-05	1.28E-02
1.5	8.50E-03	2.08E-01
3.0	1.76E-02	6.55E-01
DDPM(without classifier)	1.42E-06	3.91E-02

Table 6.6: Average Loss Value of IVS for Different β

From Table 6.6, we can find that when we fix the value of α and gradually increase the value of β, the average loss value gradually increases and is higher than the DDPM without a classifier, which means it brings negative impact to the model when the value of beta is less than 1. However, an excellent result occurs when the value of β is equal to 1. In addition, when the value of β is too much higher than 1, it also introduces a lot of noise into the surfaces that cannot be eliminated, resulting in a very high average loss value, and making the final

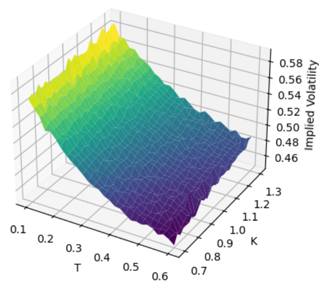


Figure 6.2: α=1, β=1.5

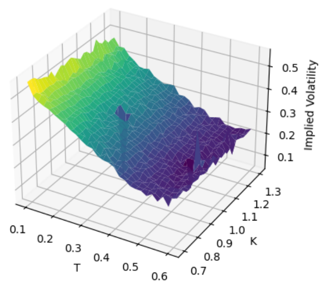


Figure 6.3: α=1, β=3.0

Figure 6.4: Different Performance of Generated IVS under Different β

generated surfaces non-smooth. We can find that in Fig 6.4 there exists peaks and non-smooth points on the IVS.

Another test of parameters occurs in smooth constraints, for which we also set two parameters, c and d:

$$L_{\text{smooth}} = c \cdot \sum \left| \frac{\partial \omega}{\partial \tau} \right| + d \cdot \sum \left| \frac{\partial \omega}{\partial k} \right|, \tag{6.9}$$

Similarly, to choose a proper balance between these two parameters, we also fix the parameter c at 0.1 at first, and gradually increase the value of d during the experiment. We also fix the values of a and b in order to reduce errors.

Parameter	$c = 0.1$	
d	Calendar Arbitrage loss	Butterfly Arbitrage loss
0.01	1.50E-03	4.20E-02
0.03	6.00E-04	4.60E-02
0.05	4.99E-05	1.28E-02
0.1	1.50E-03	9.64E-02
0.5	1.01E-02	1.0282
DDPM(without classifier)	1.42E-06	3.91E-02

Table 6.7: Average Loss Value of IVS for Different d

From Table 6.7, we can find that the value of d can not be too large, otherwise, it will influence the generation performance. When $d = 0.05$, the average loss is smaller than other results. When $d = 0.1$ or much higher, the smoothness constraints will lose effectiveness and negatively affect the performance of generated surfaces.

6.3 Comparison between with and without Arbitrary-free Conditions

The main purpose of our experiment is to verify the effect of the Classifier. For a trained DDPM, we aim to evaluate whether it will decrease the arbitrage violation of the implied volatility surface after adding the Classifier.

6.3.1 Evaluation Metrics

First, we would like to compare the performance of the DDPM-Classifier with or without the smoothness penalty to illustrate why we need to add the smoothness penalty terms to the adjusted mean.

Comparison of Classifier with and without Smoothness Constraints:

During the test process, we initially used the DDPM-Classifier without adding smoothness penalties and sometimes observed implied volatility surfaces with peak or sharp points in the generated surfaces. Such unsmooth surfaces can generate significant arbitrage violations, see in Fig 6.5. Therefore, we would like to explore the impact of adding smoothness constraints to the Classifier during the generating process. The comparison is performed in terms of both arbitrage loss value and point violation count, similar to the metrics we used when checking IVS generated by the Heston model.

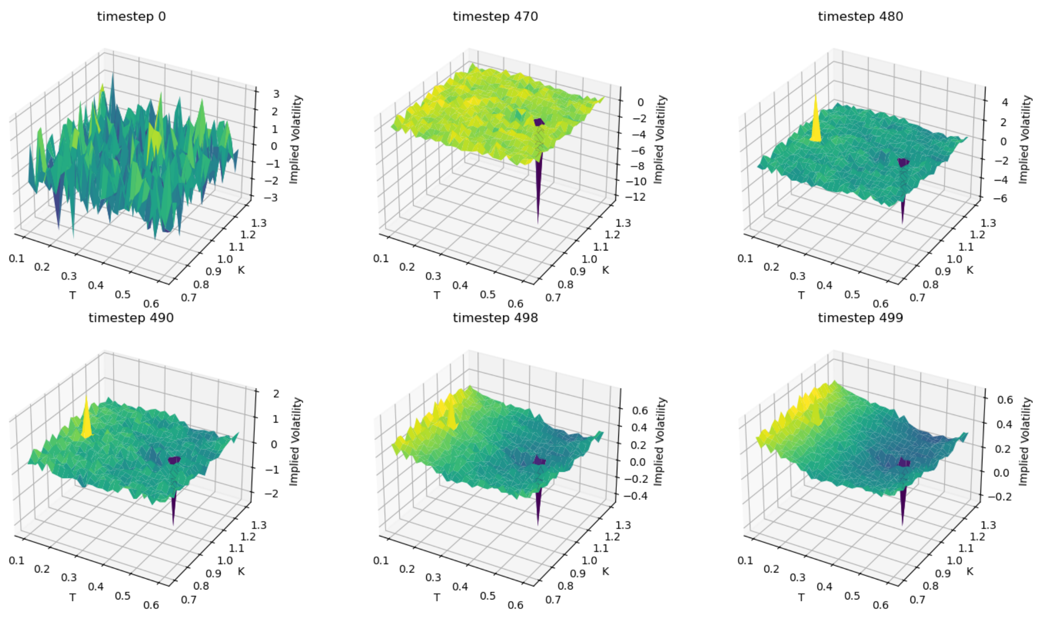


Figure 6.5: Implied Volatility Surface with Peak

- **With Smoothness Constraints:** Surfaces generated with a classifier that includes smoothness constraints are evaluated to determine whether these additional constraints reduce arbitrage violations while maintaining the quality of the surface. We expect that smoothness constraints will lead to fewer fluctuations in volatility, thereby reducing the overall arbitrage violation and point violations.

- **Without Smoothness Constraints:** The performance of surfaces generated without smoothness constraints is also evaluated. This comparison will highlight the trade-offs between enforcing strict no-arbitrage conditions and preserving surface smoothness.

Table 6.8 shows the results of the comparison of average loss value and point violation between DDPM with and without the constraints of smoothness. When there are no smoothness constraints during the generation process, the average violation loss and the number of violation points on IVS will increase.

Metric	DDPM-Classifier with Smoothness	DDPM-Classifier without Smoothness	Total number of surfaces
Average Calendar Violation Value	4.99E-05	1.40E-03	100
Average Butterfly Violation Value	1.28E-02	5.47E-02	100
Average Point Violation Count (Center)	9.9	9.6	100
Average Point Violation Count (Boundary)	10.7	20.5	100

Table 6.8: Comparison of Arbitrage Loss Values and Point Violation Counts with and without Smoothness Constraints

Comparison Between DDPM and DDPM-Classifier

In this section, we would like to evaluate the arbitrage performance of surfaces generated by the DDPM and the DDPM incorporating a classifier. The evaluation will include two main aspects:

- **Arbitrage Loss Value:**

  The arbitrage loss value is calculated for each generated surface, evaluating the degree to which the surface violates the arbitrage-free conditions. The formulas (5.6) and (5.7) will be used for checking calendar arbitrage and butterfly arbitrage in the calculations here, respectively. To confirm the evaluation's reliability and consistency, we will create multiple datasets to demonstrate the classifier's results for the data. Each individual dataset will include data for 20 implied volatility surfaces. We will compute the mean arbitrage loss for each dataset individually. To demonstrate that the results are not due to random chance, we will increase the number of datasets and incorporate additional data into the original dataset. Subsequently, we will calculate the average arbitrage loss by considering the newly added data. The procedure will persist until the average arbitrage loss value reaches a stable state with the addition of new datasets. This indicates that the inclusion of more surfaces does not have a substantial impact on the results, which means we obtain a pretty steady outcome. The stabilized value is thereafter considered as the result.

- **Point Violation Count:**

Count		Model Name			
		DDPM		DDPM-Classifier	
Number of Dataset	Number of Surface	Calendar Average Loss Value	Butterfly Average Loss Value	Calendar Average Loss Value	Butterfly Average Loss Value
1	20	0	4.36E-02	9.75E-05	2.52E-02
2	40	1.11E-06	3.95E-02	4.88E-05	2.24E-02
3	60	7.44E-07	4.01E-02	3.98E-05	1.62E-02
4	80	5.58E-07	3.90E-02	5.37E-05	1.39E-02
5	100	4.46E-07	4.00E-02	4.99E-05	1.28E-02
6	120	3.72E-07	3.80E-02	4.48E-05	1.17E-02
7	140	3.19E-07	3.90E-02	4.29E-05	1.11E-02
8	160	2.79E-07	3.80E-02	4.96E-05	1.10E-02
9	180	1.58E-06	3.90E-02	5.27E-05	1.00E-02
10	200	1.42E-06	3.90E-02	4.77E-05	9.78E-03
Stable Average Loss Value		1.42E-06	3.90E-02	4.77E-05	9.78E-03

Table 6.9: Comparison of Stabilized Average Arbitrage Loss Values of Multiple Datasets

This indicator will show the number of points that violate the no-arbitrage requirements on every implied volatility surface. Our focus will be on identifying breaches relating to calendar arbitrage and butterfly arbitrage, namely on a single point per surface. In a method similar to the computation of Arbitrage Loss Value, we will produce numerous sets of data, each comprising 20 surfaces, and determine the mean count of point violations for each dataset. We shall systematically increase the number of datasets and continue in this procedure until the mean point violation count reaches a steady state. The stabilized value will serve as the ultimate comparison metric. Furthermore, the violated points will be categorized into two groups: center point violations and boundary point violations. This distinction is made because boundary points are more sensitive to being violated during the process of generating the image. We expect that the inclusion of the classifier will have a positive effect on the violation values of the boundary points.

The results of the above metrics are as follows, see in Table 6.9 for the average arbitrage violation loss value and in Table 6.10 for the average number of point violations. Fig 6.6 shows the comparison for the butterfly loss value of the DDPM and DDPM-Classifier.

From the results, we can observe that the addition of the Classifier effectively reduces the butterfly arbitrage on the IVS. For calendar arbitrage, whether a Classifier is added to the DDPM or not, the calendar arbitrage loss value on IVS is very small, not only in terms of the value but also in terms of the number of surfaces where calendar arbitrage occurs. Therefore, adding a soft constraint may not be a good solution to this problem of a very small order of magnitude.

According to the statistics of the distribution of violation points on IVS generated by DDPM without a Classifier, arbitrage is more likely to occur at the boundary points of the surface relative to arbitrage in the central part of the surface. Therefore, we hope that our improvements to DDPM can effectively act on these boundary violation points.

We selected some implied volatility surfaces generated by DDPM with the Classifier and without the classifier and detected their boundary violations. As can be seen in Fig 6.7, we marked the violation points that appear on the boundary with red dots. When the classifier is not

Count		Model Name			
		DDPM		DDPM-Classifier	
Number of Dataset	Number of Surface	Average Point Violation(Center)	Average Point Violation(Boundary)	Average Point Violation(Center)	Average Point Violation(Boundary)
1	20	9.2	20.9	13.9	13.9
2	40	8.9	18.3	11.7	14.8
3	60	9.0	18.0	9.2	12.2
4	80	8.8	17.8	8.9	11.3
5	100	8.9	18.5	10.6	9.9
6	120	8.7	18.2	10.5	9.4
7	140	8.7	18.3	10.5	9.3
8	160	8.5	17.9	10.1	9.2
9	180	8.7	18.1	9.9	9.1
10	200	8.7	18.1	9.7	9.2
Stable Average Violation Point		8.7	18.1	9.7	9.2

Table 6.10: Comparison of Stabilized Average Point Violation Number of Multiple Datasets

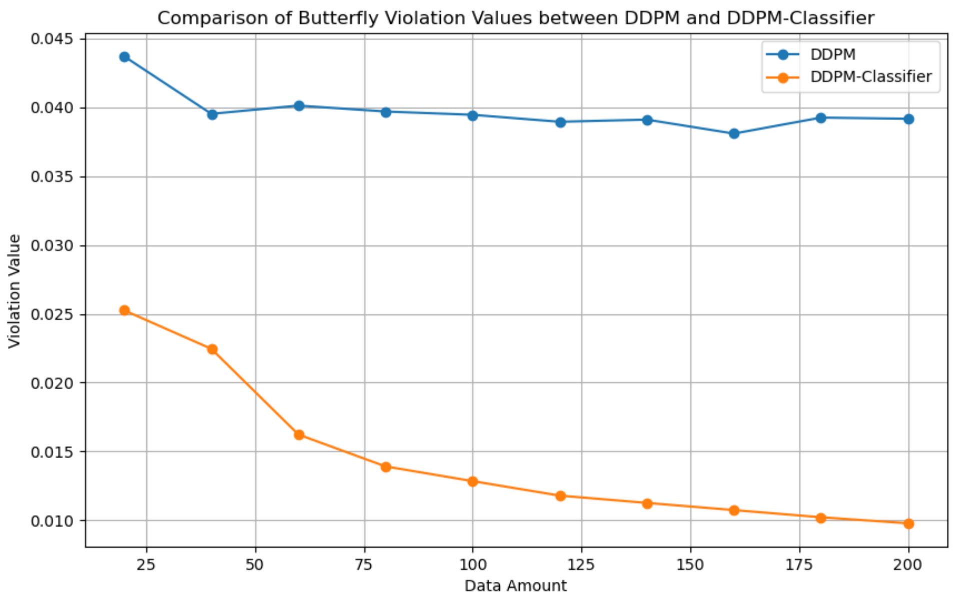
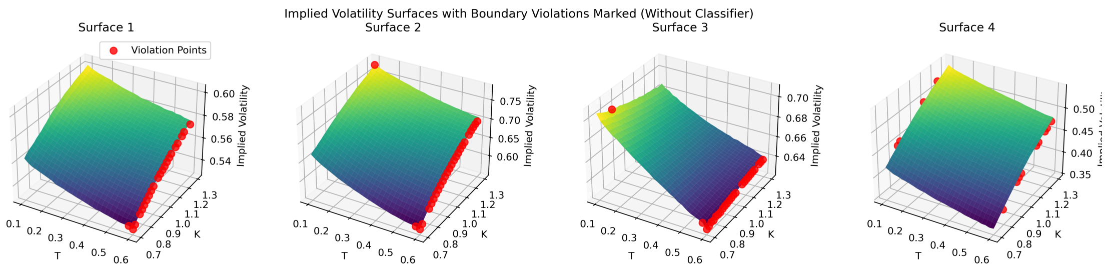
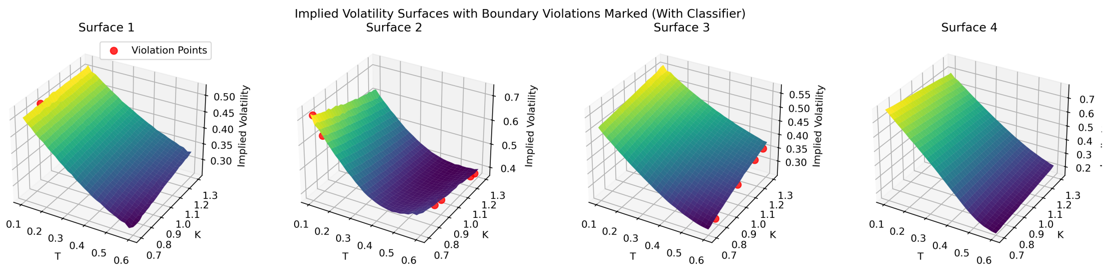


Figure 6.6: Comparison of Butterfly Loss Value between DDPM and DDPM-Classifier



(a) Violations on Boundary for IVS Generated by Standard DDPM



(b) Violations on Boundary for IVS Generated by DDPM-Classifier

Figure 6.7: Effect of the Classifier in Reducing the Number of Boundary Violations

added, arbitrage often occurs when the maturity(T) is large. However, this situation is effectively reduced in the IVS generated by the model with the classifier.

Comparison of Butterfly Arbitrage Violation Value:

A notable observation during the evaluation is that in all surfaces generated by the DDPM, none achieved the requirement of butterfly arbitrage condition, which means the butterfly violation value is non-zero for all surfaces. This indicates that every surface generated without the Classifier violated the butterfly arbitrage condition.

However, when generating surfaces with the DDPM-Classifier, some of the surfaces achieved the requirement of the butterfly arbitrage violation value of zero. This suggests that the Classifier effectively enforces the butterfly arbitrage condition, indicating the improvements brought by the Classifier in compliance with no-arbitrage conditions.

The following Table 6.11 summarizes the comparison between the DDPM and the DDPM-Classifier in terms of the number of surfaces with both calendar and butterfly Arbitrage Violations equal to zero.

Model	Number of Surfaces with Arbitrage Violation equal to 0	Total Number of Surfaces
DDPM	0	200
DDPM-Classifier	21	200

Table 6.11: Comparison of Surfaces with Arbitrage Violation Value Equal to 0

6.3.2 Arbitrage Analysis

In the results listed above, we can observe that adding the Classifier effectively reduces the average butterfly loss value, not only reducing the occurrence of arbitrage behavior on the surface but also reducing the number of arbitrage points, especially for the arbitrage boundary points on the surface.

Another important conclusion is that there are parts of the surfaces that are completely arbitrage-free. In the generation results of the DDPM, the number of completely arbitrage-free surfaces is 0. In the case where we added the classifier, there appear some surfaces on which no arbitrage behavior is generated at all. This suggests that the classifier guidance we added is having a positive effect.

The issue that deserves our attention is the calendar arbitrage phenomenon that still appears in the IVS. After the addition of the classifier, calendar arbitrage does not decrease significantly, but maintains a value almost similar to the DDPM. There are several possible reasons for this problem:

Firstly, among the implied volatility surfaces generated by the DDPM, most surfaces have no calendar arbitrage violations, as can be seen from the average loss value we have calculated, which is a quite small number, 1.42E−06. Therefore, when we use the DDPM-Classifier model, this tiny value is susceptible to fluctuations due to the new loss terms that we have added to the generation process.

On the other hand, the constraints we add are merely a soft guide to the generation process and not strictly hard constraints. Therefore, it is likely that the tiny calendar loss value will be ignored by this soft constraint, or the effect of the soft constraint will be canceled out by the negative effect of introducing a new loss term. As shown in our previous study of smoothness, if we do not add a constraint on smoothness, the Classifier sometimes tends to make the generated surfaces unsmooth or with a peak. Even though this negative effect is mostly eliminated after adding the smoothness constraint, the calendar arbitrage loss value is so small that still sensitive to this negative effect.

7 Conclusion and Future Work

7.1 Conclusion

Our research aims to ensure that the no-arbitrage conditions can be successfully applied to the Implied Volatility Surface (IVS) by using a guidance denoising diffusion probability model (DDPM) with an additional classifier. During the process of our research, we achieved this goal by adding restrictions to the generation process of DDPM.

Our work began with a study of the condition of no-arbitrage. The two most important no-arbitrage conditions are the calendar arbitrage condition and the butterfly arbitrage condition. After establishing the specific form and confirming the validity of these two conditions, we used these conditions to determine the degree to which an implied volatility surface satisfies the no-arbitrage requirements. Some data-driven methods have shown that generating IVS without arbitrage violation is possible. When it comes to the DDPM we used in our research, we mainly considered the DDPM-Classifier to impose financial constraints. In our experimental results, we focused on comparing the impact of the performance of the presence or absence of the classifier on the DDPM to generate IVS. Based on the calendar and butterfly conditions, we used the average loss value as a measure of the degree of arbitrage and also calculated the number of arbitrage points on the IVS as another metric in both cases (with and without the Classifier).

Finally, we found that the addition of our classifier effectively reduces the degree of arbitrage on the IVS. The reduction in the degree of arbitrage on the implied volatility surface is mainly in two areas. The first is the reduction of the average loss value, especially in the reduction of the butterfly arbitrage loss value. For DDPM without the Classifier, all the generated surfaces have a certain degree of butterfly arbitrage. After the addition of the Classifier, not only the average loss value of butterfly arbitrage violation is reduced, but at the same time, a part of the surface completely eliminates butterfly arbitrage. In addition, when counting the number of points with arbitrage violations on the implied volatility surface, it can also be found that after the addition of the Classifier, the number of violation points appearing in the boundary of IVS decreases significantly, which indicates that the classifier has a positive influence on improving the quality of surface generation.

In addition, we also focused on the balance between this newly added impact item and the quality of the generated surfaces. Then, by adding a restriction on the smoothness of the surfaces, we greatly reduced the negative impact that the introduction of the adding of classifier would have had on the DDPM.

However, it can be found in our results that the arbitrage opportunities of IVS generated by DDPM are sometimes not eliminated. Even in the case where a Classifier is used to guide the generation process, not all surfaces are completely arbitrage-free. Since the added constraints are soft rather than strict, these violations may not be completely resolved and may even be affected by the addition of the Classifier. In addition, without smoothing constraints, the model sometimes generates IVS with peaks and fluctuations. Although smoothing constraints can reduce this effect, the tiny arbitrage loss value is still susceptible to these disturbances.

7.2 Future Work

Although the DDPM-Classifier has made some progress in complementing arbitrage-free conditions for IVS, as mentioned before, it sometimes does not completely eliminate the arbitrage opportunities that may exist on an implied volatility surface. Therefore, for further work in the future, it may be a possible approach to focus on how to add strict financial restrictions to IVS generated by DDPM. As we mentioned in the literature review section, another possible way to add hard constraints might start by modifying the structure of the neural network used in the deep learning method. By making the neural network sparsely connected, it may be possible to increase the strength of the constraints imposed on the model.

Moreover, the conditional DDPM used in our research is dependent on the effect of the classifier. For example, in our experiments, the no-arbitrage and smoothness conditions are added through the classifier to the generation process of DDPM. In the experimental results, it can also be observed that when the hyperparameters of the factors used in the classifier are modified, there is a significant impact on the performance of generated surfaces. The other conditional DDPM method mentioned in the previous section, i.e., classifier-free DDPM, does not require the additional classifier to be added during the generation process, but rather the conditional constraints are directly added to the DDPM during the training process.

When using classifier-free DDPM as a solution, researchers can mainly focus on its training process. such as the joint training method mentioned in [30]. This joint training process does not mean training two different models at the same time, but rather combining conditional and unconditional constraints when training a single model to avoid dependency on the additional classifier. For example, conditions can be ignored with a specific probability during the training process so that the model learns unconditional generation in some cases and conditional generation in others (e.g., the condition here could be the no-arbitrage requirements). Then, in the generation process, the conditional and unconditional outputs can be balanced by a parameter w. This parameter is adjusted until the model finally produces the required output.

Bibliography

[1] A. Na, M. Zhang, and J. Wan, "Computing volatility surfaces using generative adversarial networks with minimal arbitrage violations," 2023. [Online]. Available: https://arxiv.org/abs/2304.13128

[2] M. Vuletić and R. Cont, "Volgan: a generative model for arbitrage-free implied volatility surfaces," *SSRN Electronic Journal*, 01 2023.

[3] B. Ning, S. Jaimungal, X. Zhang, and M. Bergeron, "Arbitrage-free implied volatility surface generation with variational autoencoders," *SIAM Journal on Financial Mathematics*, vol. 14, no. 4, pp. 1004–1027, 2023.

[4] Z. Gong, W. Frys, R. Tiranti, C. Ventre, J. O'Hara, and Y. Bai, "A new encoding of implied volatility surfaces for their synthetic generation," 2023. [Online]. Available: https://arxiv.org/abs/2211.12892

[5] M. Bergeron, N. Fung, J. Hull, and Z. Poulos, "Variational autoencoders: A hands-off approach to volatility," 2021. [Online]. Available: https://arxiv.org/abs/2102.03945

[6] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," 2021. [Online]. Available: https://arxiv.org/abs/2105.05233

[7] G. Müller-Franzes, J. M. Niehues, F. Khader, S. T. Arasteh, C. Haarburger, C. Kuhl, T. Wang, T. Han, T. Nolte, S. Nebelung *et al.*, "A multimodal comparison of latent denoising diffusion probabilistic models and generative adversarial networks for medical image synthesis," *Scientific Reports*, vol. 13, no. 1, p. 12098, 2023.

[8] W. G. C. Bandara, N. G. Nair, and V. M. Patel, "Ddpm-cd: Denoising diffusion probabilistic models as feature extractors for change detection," 2024. [Online]. Available: https://arxiv.org/abs/2206.11892

[9] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *The Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970. [Online]. Available: http://www.jstor.org/stable/2325486

[10] D. Sornette, P. Cauwels, and G. Smilyanov, "Can we use volatility to diagnose financial bubbles? lessons from 40 historical bubbles," *Quantitative Finance and Economics*, vol. 2, no. 1, pp. 486–594, 2018.

[11] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of political economy*, vol. 81, no. 3, pp. 637–654, 1973.

[12] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *The review of financial studies*, vol. 6, no. 2, pp. 327–343, 1993.

[13] D. Amengual and D. Xiu, "Resolution of policy uncertainty and sudden declines in volatility," *Journal of Econometrics*, vol. 203, no. 2, pp. 297–315, 2018.

[14] X. Guo, M. McAleer, W.-K. Wong, and L. Zhu, "A bayesian approach to excess volatility, short-term underreaction and long-term overreaction during financial crises," *The North American Journal of Economics and Finance*, vol. 42, pp. 346–358, 2017.

[15] E. Rossi and P. S. De Magistris, "Long memory and tail dependence in trading volume and volatility," *Journal of Empirical Finance*, vol. 22, pp. 94–112, 2013.

[16] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022. [Online]. Available: https://arxiv.org/abs/1312.6114

[17] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[18] M. Chataigner, S. Crépey, and M. Dixon, "Deep local volatility," p. 82, 2020.

[19] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[20] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," 2021. [Online]. Available: https://arxiv.org/abs/2011.13456

[21] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang, "Diffusion models: A comprehensive survey of methods and applications," 2024. [Online]. Available: https://arxiv.org/abs/2209.00796

[22] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, "Cascaded diffusion models for high fidelity image generation," 2021. [Online]. Available: https://arxiv.org/abs/2106.15282

[23] E. B. Asiedu, S. Kornblith, T. Chen, N. Parmar, M. Minderer, and M. Norouzi, "Decoder denoising pretraining for semantic segmentation," 2022. [Online]. Available: https://arxiv.org/abs/2205.11423

[24] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, "Video diffusion models," 2022. [Online]. Available: https://arxiv.org/abs/2204.03458

[25] Z. Lyu, Z. Kong, X. Xu, L. Pan, and D. Lin, "A conditional point diffusion-refinement paradigm for 3d point cloud completion," 2022. [Online]. Available: https://arxiv.org/abs/2112.03530

[26] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto, "Diffusion-lm improves controllable text generation," 2022. [Online]. Available: https://arxiv.org/abs/2205.14217

[27] Y. Kossale, M. Airaj, and A. Darouichi, "Mode collapse in generative adversarial networks: An overview," in *2022 8th International Conference on Optimization and Applications (ICOA)*. IEEE, 2022, pp. 1–6.

[28] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," 2017. [Online]. Available: https://arxiv.org/abs/1701.04862

[29] J. M. Tomczak, *Deep Generative Modeling*, 1st ed. Springer Cham, 2022. [Online]. Available: https://doi.org/10.1007/978-3-030-93158-2

[30] J. Ho and T. Salimans, "Classifier-free diffusion guidance," 2022. [Online]. Available: https://arxiv.org/abs/2207.12598

[31] Black-Scholes Model Assumptions. [Online]. Available: https://www.macroption.com/black-scholes-assumptions/

[32] S. L. Heston, "A closed-form solution for options with stochastic volatility with applications to bond and currency options," *The Review of Financial Studies*, vol. 6, no. 2, pp. 327–343, 1993. [Online]. Available: http://www.jstor.org/stable/2962057

[33] F. Fang, "The cos methodan efficient fourier method for pricing financial derivatives," Ph.D. dissertation, Delft University of Technology, 2010, phD Thesis. [Online]. Available: https://resolver.tudelft.nl/uuid:9aa17357-af21-4c09-86a2-3904ced4b873

[34] M. Roper, "Arbitrage free implied volatility surfaces," *preprint*, 2010.

[35] D. Ackerer, N. Tagasovska, and T. Vatter, "Deep smoothing of the implied volatility surface," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 552–11 563, 2020.

[36] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *CoRR*, vol. abs/2006.11239, 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[37] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[38] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[39] J. Gatheral and A. Jacquier, "Arbitrage-free svi volatility surfaces," pp. 59–71, 2014.

[40] X. Ma, "Diffusion probabilistic model for implied volatility surface generation and completion," Master's Thesis, Delft University of Technology, 2023.