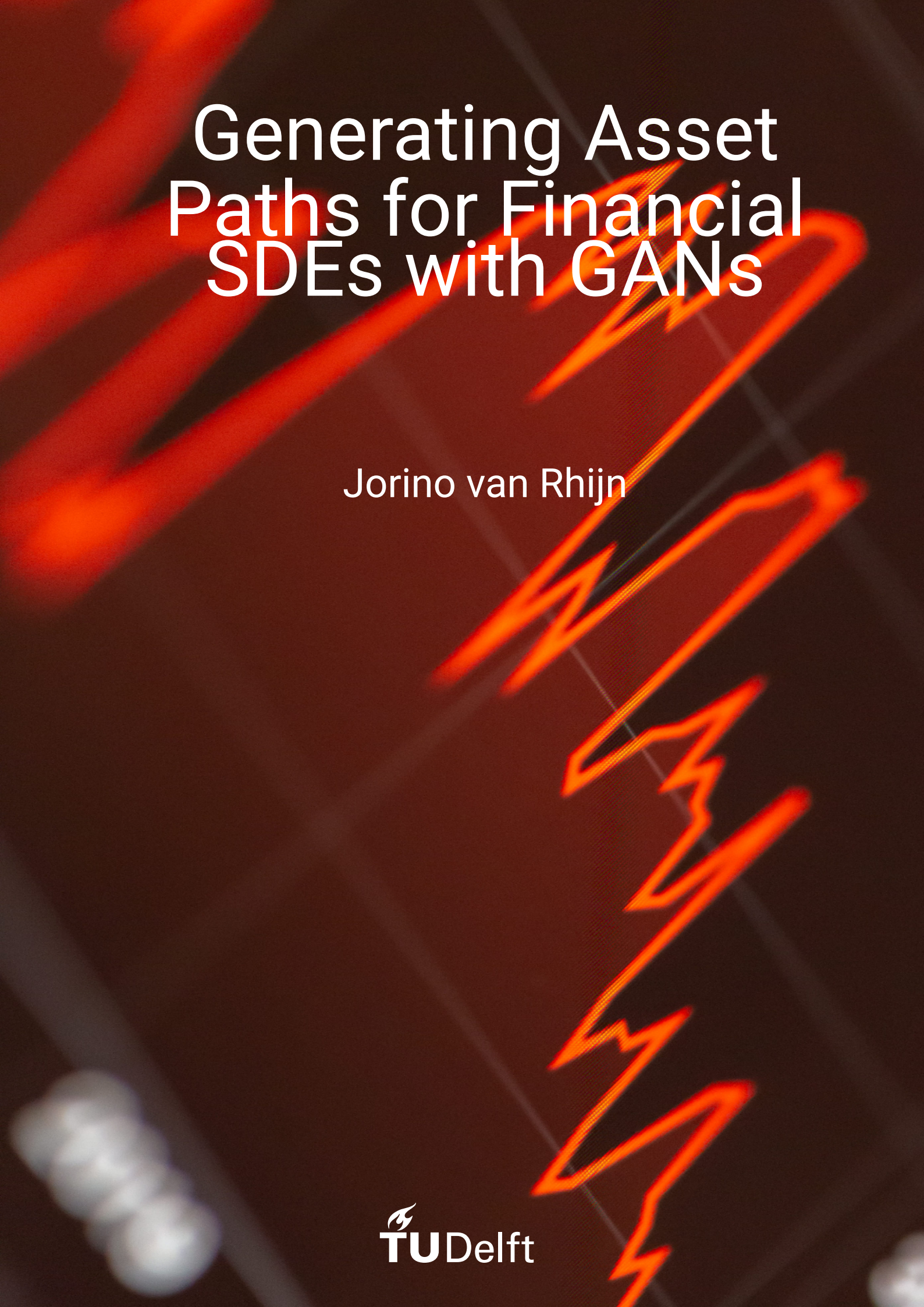


Generating Asset Paths for Financial SDEs with GANs

Jorino van Rhijn

TU Delft

Generating Asset Paths for Financial SDEs with GANs

by

Jorino van Rhijn

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 28 October 2020 at 11:00 AM.

Student number:	4296796	
Specialisation:	Financial Engineering	
Project duration:	3 February 2020 – 28 October 2020	
Thesis committee:	Prof. dr. ir. C.W. Oosterlee	TU Delft and CWI
	Dr. R.J. Fokkink	TU Delft
	Dr. ir. L.A. Grzelak	Rabobank and TU Delft
	S. Liu, MSc	TU Delft

TU Delft

Abstract

Generative adversarial networks (GANs) have shown promising results when applied on partial differential equations and financial time series generation. This thesis investigates if GANs can be used to provide a strong approximation to the solution of stochastic differential equations (SDEs) of the Itō type. Standard GANs are only able to approximate processes in conditional distribution, yielding a weak approximation to the SDE. A novel GAN architecture is proposed that enables strong approximation, called the constrained GAN. The discriminator of this GAN is informed with the random sample that corresponds to the Brownian motion increment between two time steps. This way, the constrained GAN does not only learn the conditional distribution, but the unique map from a random increment to the next asset value along the path, conditional on the previous value. The architecture was tested on geometric Brownian motion (GBM) and the Cox Ingersoll Ross (CIR) process in one dimension, where it was conditioned on a range of time steps and previous values of the asset process. The constrained GAN was shown to outperform discrete-time schemes in strong error on a discretisation with large time steps. It also outperformed the standard conditional GAN when approximating the conditional distribution. A method is proposed to extend the constrained GAN to general one-dimensional Itō SDEs, beyond the SDEs tested in this work. In future work, the constrained GAN should be conditioned on the SDE parameters as well, allowing it to learn an entire family of solutions at once. Furthermore, the architecture could be extended to higher dimensions, including systems of SDEs, such as the Heston model.

Acknowledgements

First and foremost, I would like to thank Prof. dr. ir. Cornelis W. Oosterlee for providing the opportunity to carry out the research of my master thesis. I thank him for the detailed feedback he gave on my work and for supporting me throughout the duration of the project. I thank Dr. ir. Lech A. Grzelak for his critical reflections on my work and his input during our discussions, which provided essential insights along the way. I thank Shuaiqiang Liu, MSc, for our detailed and fruitful discussions about the subject matter, which helped me find my way in the rich body of literature related to this work. I would like to thank all three for the cordial interaction during our weekly online discussions, which I enjoyed very much.

I would also like to thank my friends from the faculty of EEMCS for our frequent moments of working together on our theses - at appropriate distance due to the Covid-19 pandemic.

Finally, I would like to thank my family and friends for their unrelenting support.

Contents

1 Introduction 1

2 Preliminaries 5
2.1 Stochastic Differential Equations . . . 5
2.1.1 Weak and strong solutions . . . 6
2.1.2 Discrete-time approximations . . . 7
2.1.3 Weak and strong convergence for discrete-time approximations . . . 8
2.1.4 SDEs under consideration . . . 8
2.2 Artificial Neural Networks . . . 9
2.2.1 Architecture . . . 9
2.2.2 Universal approximation with neural networks . . . 10
2.2.3 Gradient descent and backpropagation . . . 11
2.2.4 Adam . . . 12
2.2.5 Activation functions . . . 12
2.2.6 Other types of NNs . . . 13

3 Generative Adversarial Networks 15
3.1 Introduction . . . 15
3.2 Theoretical Overview . . . 15
3.2.1 Formal definition . . . 16
3.2.2 Optimisation problem . . . 16
3.2.3 Optimality in the ideal case . . . 17
3.2.4 Connection with the JS-divergence . . . 18
3.2.5 Optimal generator . . . 19
3.2.6 Non-unicity of the optimal generator . . . 19
3.2.7 Non-ideal case: finite parameter set for the discriminator . . . 20
3.2.8 Including the effect of finite datasets . . . 20
3.2.9 Concluding remarks on the theory . . . 22
3.3 Training GANs in Practice . . . 22
3.3.1 Perfect discriminators . . . 22
3.3.2 Vanishing generator gradient problem . . . 23
3.3.3 Mode collapse . . . 23
3.4 Variations of the 'Vanilla GAN' . . . 23
3.4.1 Ideas for stabilising the training process . . . 23
3.4.2 Conditional GAN . . . 24
3.4.3 Other GANs . . . 24
3.4.4 GAN setup used in this thesis . . . 25
3.5 Algorithmic Formulation . . . 25

4 Methodology 27
4.1 Setting . . . 27
4.2 Discrete Approximations with GANs . . . 29
4.2.1 Vanilla GAN . . . 30
4.2.2 Conditional GAN . . . 30
4.2.3 Weak approximation . . . 31
4.2.4 Strong approximation . . . 31
4.3 Constrained GAN . . . 33
4.3.1 Constrained GAN as inverse map . . . 33
4.4 General Case: Construction of the Constrained GAN . . . 34
4.4.1 Train on high quality discrete-time approximation . . . 34

4.5 Data pre- and post-processing . . . 36
4.5.1 Adaptedness of logreturns . . . 37
4.6 Construction and Analysis of Paths . . . 37
4.6.1 Analysing the synthetic paths . . . 37
4.7 Practical Considerations for the CIR Process . . . 38
4.7.1 Discrete-time schemes for the CIR process . . . 38
4.7.2 Pre-processing step if the Feller condition is not satisfied . . . 38

5 Comparing Probability Distributions 41
5.1 Empirical distribution functions . . . 41
5.2 Combination of KS Statistic and 1-Wasserstein distance . . . 41
5.2.1 KS statistic . . . 41
5.2.2 1D Wasserstein distance . . . 42
5.2.3 Implementation for analysing the GAN output . . . 42
5.3 Reference on Distributional Divergences and Distances . . . 43

6 Results 47
6.1 Approximating the Conditional Distribution . . . 47
6.1.1 Statistics using various test sizes. . . . 48
6.1.2 Autocorrelation structure . . . 49
6.1.3 Training process . . . 50
6.2 Constructing Paths . . . 50
6.2.1 Single-step GAN approximation . . . 52
6.3 Analysis of the Constrained GAN . . . 53

7 Discussion and Recommendations 57
7.1 Architecture and training process . . . 57
7.2 Data pre-processing. . . . 57
7.3 Comparison with stochastic collocation . . . 58
7.4 Generality and extensions . . . 58
7.5 Outlook . . . 59

8 Conclusion 61

Bibliography 63

A Appendix 69
A.1 Network Architecture. . . . 70
A.2 Discriminator Output . . . 71
A.3 Empirical validation of section 4.4.1 . . . 71
A.4 Additional Results on Paths . . . 73
A.4.1 GBM . . . 73
A.4.2 Single-step approximation . . . 73
A.4.3 CIR process: Feller condition satisfied. . . . 74
A.4.4 Single-step approximation - Feller condition satisfied. . . . 74
A.4.5 CPU runtimes on single-step approximation . . . 75
A.4.6 Vanilla GAN fails to provide a strong approximation. . . . 75
A.4.7 Strong error over time . . . 76
A.5 Additional Results for Conditional GAN . . . 77
A.5.1 Interpolation capabilities on a single condition . . . 77
A.5.2 Feller condition satisfied: autocorrelation structure. . . . 77
A.5.3 GBM and CIR process - vanilla vs constrained GAN. . . . 78
A.6 Parameter sets and Training Phase . . . 79
A.6.1 SDE parameters . . . 79
A.6.2 Conditional GAN: construction of training set . . . 79
A.6.3 Training process . . . 79
A.6.4 Effect of the Learning Rate Schedule . . . 80
A.7 Further Extensions of the Constrained GAN . . . 81
A.7.1 Karhunen-Loève expansion. . . . 81

A.7.2 Regularisation. 81

1 Introduction

Stochastic differential equations (SDEs) are prevalent in the modelling of stochastic dynamical systems in engineering, physics, healthcare, and many more domains [1]. In finance, they are cornerstone in the modelling of asset prices and interest rates, with applications in portfolio management and the pricing of financial derivatives and related products [2]. In this work, we will exclusively consider SDEs of the Itō type, e.g. of the form:

$$dS_t = A(t, S_t)dt + B(t, S_t)dW_t, \tag{1.1}$$

starting at S_0 and with underlying Brownian motion W_t. We will refer to a realisation of the solution of equation 1.1 over time as a path. In general, the analytical solution to SDEs is not available, which is why practitioners make extensive use of numerical approximations to simulate paths in a Monte Carlo setting [2]. However, high-quality numerical approximation may be too costly in an online setting for practical purposes. Meanwhile, in most practical applications, a continuous representation of the path is not of interest, but rather the solution at specific times along the path. The goal in this work is to let a neural network provide a high-quality approximation of paths corresponding to equation 1.1 on a set of times $\{t_0, t_1, \ldots\}$, but without computing the process on the intermediate time steps. Instead, our goal is to let a neural network 'predict' the solution at time $t + \Delta t$, given the solution at time t, for large time steps Δt. 'Large' here means large in comparison with typical values of Δt when using a discrete-time scheme.

Neural networks have become a popular tool in applied mathematics to approximate functions in high dimensions, or which are otherwise intractable. They have been successfully applied on problems involving partial differential equations (PDEs) [3, 4], time series generation [5–7], chaotic dynamical systems [8], anomaly detection [9] and many more. In particular, generative adversarial networks (GANs) have been shown to produce promising results when solving stochastic PDEs (SPDEs) [10], learning conditional distributions for time series [11], or particle density problems where the GAN is 'informed' with the particle advection dynamics [12]. However, 'informing' a neural network with the SDE dynamics is not straightforward, as its terms are not differentiable. This requires alternative approaches to the ones used on PDEs. The contributions in this work are as follows. Firstly, a novel GAN-based scheme is proposed, the constrained GAN, which approximates the strong solution to SDEs on a time discretisation. Secondly, a method is proposed to extend this architecture to general one-dimensional Itō SDEs. Thirdly, a framework is provided for understanding the map that the generator provides and how it relates to the discriminator output.

Earlier work

Much work has been done on solving PDEs with neural networks. An example of early work dates back to 1998, when in [13] it was proposed to minimise a cost function based on the PDE dynamics, progressively improving Ansatz solutions created by neural networks. Much improvement has been made since, as current neural networks can solve complicated high dimensional PDE problems, as appearing in fluid dynamics, quantum mechanics, quantitative finance, and many more domains [3, 4, 14–17]. Recent work has shown promising results using GANs as well. In particular, Yang et al. [10] encode

the dynamics of a stochastic PDE into a GAN. Xie et al. generate accurate smoke plumes by encoding advection dynamics into separate discriminators. Stinis et al. [18] and Wu et al. [8] use similar constraints with GANs to improve the accuracy of the network and convergence rates.
These networks rely on application of the PDE operator on outcomes generated with a neural network. In the case of Itō SDEs, however, the Brownian motion term precludes differentiability of the dynamics. In [19], a novel method is proposed to find the drift coefficients of SDEs, using generative modelling to sample observations of the underlying process. The authors use a coordinate transformation and Itō's lemma to obtain a differentiable latent process $z(t)$ for both the training data and generated data. Analogous to (S)PDE solving, the derivatives of the latent process $z(t)$ are matched between the observed and generated data. A cost function is minimised to find the model parameters on which the observed data were based. A similar moment-matching technique may be used to define a cost function on Ansatz financial SDE solutions, as is done in PDE solving with neural networks. However, the technique proposed in [19] only works for constant diffusion parameters $B(t, S_t) = B$, which is too restrictive for the purposes in this work.

Instead of focusing directly on solving the SDE, one could treat the underlying stochastic process of the SDE as a time series. If a training set is available, a neural network could be used to construct samples that share the same conditional distribution. Wiese et al. [5] use a wave-net-based GAN architecture, 'QuantGAN', to model financial time series and show promising results on synthetic time series and real-world datasets. With this setup, the 'next step' on a path is predicted, based on a rolling window of realisations from the past. The authors show that the QuantGAN accurately captures the autocorrelation structure in the time series. Recently, a similar approach has been proposed by Ni et al. in [11], called 'Signature Wasserstein GAN'. This architecture uses a fully-connected network, for which the authors define a loss function, based on the difference in the signature of synthetic paths and reference paths [11]. Their work is similar in scope to the QuantGAN approach and shows the ability to accurately capture the underlying autocorrelation structure and marginal density of financial time series data. In [5], it is shown that the output of the neural network is adapted to the input sequence $\{Z_k\}$ of i.i.d. $N(0, 1)$ random variables, which means that it could find a weak solution to the SDE. However, both approaches would provide no guarantees of finding a strong solution to the SDE, i.e. path-wise approximation given the same Brownian motion on which the SDE is defined. Details about the difference between weak and strong solutions will be further explored in chapters 2 and 4.

The works highlighted in [5] and [11] differ with ours in their objective to model general financial time series, e.g. real-world stock market data, while in our work we focus specifically on SDEs. This allows us to exploit the Markov property of SDEs. All information implied in the SDE, including the autocorrelation structure, is contained in the transition distribution $F_{S_{t+\Delta t}|S_t}$ between two time steps, conditional on the current time t, time step Δt and the previous value S_t.

Accurate sampling from this transition distribution is the central idea behind exact simulation schemes for typical SDEs in finance, such as the technique introduced by Broadie and Kaya [20] for the Heston model. The inverse of the transition distribution of the exact solution between two time steps may be approximated using the stochastic collocation Monte Carlo (SCMC) method [21], which allows one to efficiently sample from $F_{S_{t+\Delta t}|S_t}$. The SCMC method approximates the inverse distribution with a polynomial expansion of inexpensive (e.g. normal) prior samples. The SCMC method can be applied on challenging SDEs in finance, such as the SABR model [21]. However, the SCMC method only provides an approximation of $F_{S_{t+\Delta t}|S_t}$ given a single choice of S_t, t and Δt. In [22], this is addressed by combining the SCMC method with a neural network that predicts the collocation points for the SCMC method, conditional on S_t, t, Δt and the model parameters. This provides an almost exact solution, in the sense that the any error would arise from the ability to approximate the exact conditional distribution. In our work, the scope is similar, but the conditional distribution will be approximated directly by a conditional GAN, instead of using the SCMC method.

To the best of our knowledge, no prior work has been done on approximating the conditional transition distribution $F_{S_{t+\Delta t}|S_t}$ with a GAN on SDE problems. Fu et al. [23] use a conditional GAN to reconstruct time series models. The conditional GAN is able to sample from the target distributions given several regimes of parameters. The authors use the conditional GAN to approximate time series models such as AR, GARCH and real-world stock market data, by conditioning the GAN on a single previous step. In our case, the same approach will be used, but this time the GAN is trained on the exact solution to financial SDEs. We will not only condition on the previous values S_t, but also on the time step Δt. By repeated sampling from the conditional GAN approximation of the transition distribution

given S_t and Δt, a path of arbitrary length and with arbitrary time steps can be constructed. Another key distinguishing feature in this work is that we will require the conditional GAN to provide a strong approximation of the SDE, instead of matching the conditional distribution alone, as done in [23] for time series.

Research goals and thesis outline

The first goal is obtain an accurate approximation of the transition distribution $F_{S_{t+\Delta t}|S_t}$ implied by the SDE for large time steps. This approximation will be provided by a conditional GAN, trained on a dataset of samples from the exact solution to the SDE. The second challenge is to ensure that the conditional GAN provides a strong approximation, i.e. one that approaches the exact strong solution path-wise. Two SDEs are under consideration: geometric Brownian motion (GBM) and the Cox, Ingersoll Ross (CIR) process. In all cases, the GAN will be trained, conditional on a range of previous values S_t and time steps Δt, with the SDE parameters held fixed. If the GAN were trained on the SDE parameters as well, it would learn a family of solutions to the SDE. This is left for future work. In this work, we explore the GAN's ability to provide a path-wise approximation on a discretisation. The thesis is structured as follows: in chapter 2, the necessary background will be provided about SDEs, discrete-time approximations and neural networks. Chapter 3 shows in detail how GANs can approximate any probability distribution, given only a training set of samples from the target distribution. In the succeeding chapter 4, it will be shown how GANs can be used to provide a strong approximation to the SDE on a discretisation of a time interval $[0, T]$. The constrained GAN architecture will be introduced, along with extensions beyond the SDEs considered in this thesis. Chapter 5 provides an overview of modern non-parametric techniques for studying the quality of the GAN output. Then, in chapter 6, the results are presented. Chapter 7 provides a reflection of the results, along with recommendations for future work. Finally, chapter 8 concludes.

2

Preliminaries

This chapter explains the theoretical background required for the central themes in this thesis. It will start with a detailed formulation of stochastic differential equations (SDEs), followed by a description of the SDEs under consideration and their properties. In the second part, neural networks are introduced and the key techniques that underlie them are briefly discussed.

2.1. Stochastic Differential Equations

Let $(\Omega, \mathcal{F}, P)$ be a probability space and $\boldsymbol{W}_t = [W_1(t), \dots, W_m(t)]^T$ be an m-dimensional Brownian motion, adapted to the natural filtration $\mathcal{F}_t := \sigma\{\boldsymbol{W}_s : s \leq t\}$. In its most general form, a stochastic differential equation (SDE) of the Itō type on $\mathbb{R}^n$ is then given as follows, following the definition given in [24, ch.26]:

$$\begin{aligned} d\boldsymbol{S}_t &= \boldsymbol{A}(t, \boldsymbol{S}_t)dt + \boldsymbol{B}(t, \boldsymbol{S}_t)\boldsymbol{dW}_t, \\ \{\boldsymbol{S}_t\}_{t=0} &= \boldsymbol{S}_0 \in \mathbb{R}^n, \end{aligned} \tag{2.1}$$

where $\{\boldsymbol{S}_t\}_{t\geq 0}$ is an adapted process, defined by the map $\boldsymbol{S}_t : \Omega \times \mathbb{R}^+ \to \mathbb{R}^n$, $(\omega, t) \mapsto \boldsymbol{S}_t$, for $(\omega, t) \in \Omega \times \mathbb{R}^+$. $\boldsymbol{A}(t, \boldsymbol{S}_t)$ is a random vector on $\mathbb{R}^n$ and $\boldsymbol{B}(t, \boldsymbol{S}_t)$ is a real-valued random matrix of size $n \times m$. $\boldsymbol{A}$ and $\boldsymbol{B}$ have random processes as their elements. Suppose that for each $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$, the maps $(t, x) \mapsto A_i(t, x)$ and $(t, x) \mapsto B_{i,j}(t, x)$ are $\mathcal{F}_t$-measurable. A solution to equation 2.1 is a continuous adapted process on $\mathbb{R}^n$ that satisfies the Itō integral form of the SDE [24]:

$$\boldsymbol{S}_t = \boldsymbol{S}_0 + \int_0^t \boldsymbol{A}(s, \boldsymbol{S}_s)ds + \int_0^t \boldsymbol{B}(s, \boldsymbol{S}_s)\boldsymbol{dW}_s, \quad \forall t \geq 0 \quad P-\text{a.s.} \tag{2.2}$$

Thus, the solution to the SDE is a system of equations for each $S_t^i := \{\boldsymbol{S}_t\}_i, S_0^i := \{\boldsymbol{S}_0\}_i$, where W_t^j resembles the j^{th} Brownian motion:

$$S_t^i = S_0^i + \int_0^t A_i(s, \boldsymbol{S}_s)ds + \sum_{j=1}^m \int_0^t B_{i,j}(s, \boldsymbol{S}_s)dW_s^j, \quad \forall t \geq 0 \quad P-\text{a.s.} \tag{2.3}$$

Note how this formulation links each S_t^i in the system of equations $\boldsymbol{S}_t$ through the matrix $\boldsymbol{B}(t, S_t)$. To better understand the matrix $\boldsymbol{B}$, consider the simple example of a process with correlated Brownian motions, cf. for example [25, p.82], in which case the correlation structure between the standard Brownian motions W_t^j is stored in the matrix B.
Although this formulation may be insightful in the general case, the SDEs in this thesis will be set in $\mathbb{R}$ with a single Brownian motion, i.e. $m = n = 1$. In this case, $A(t, S_t)$ and $B(t, S_t)$ are simply $\mathcal{F}_t$-adapted processes on $\mathbb{R}$. Without loss of generality, we can define the stochastic integral running from a time t to a time set in the future, $t + \Delta t$, for some increment $\Delta t \geq 0$. The stochastic integral can then be written as shown in equation 2.4.

$$S_{t+\Delta t} = S_t + \int_t^{t+\Delta t} A(s, S_s)ds + \int_t^{t+\Delta t} B(s, S_s)dW_s \quad P-\text{a.s.}, \tag{2.4}$$

with initial value $S_t \in \mathbb{R}$. This formulation will be used as the 'default' formulation of the stochastic integral in subsequent sections. The natural filtration is an important concept throughout the rest of this work. Although we will write $\mathcal{F}_t = \sigma(\{W_t\})$, this is a shorthand for the following definition:

Definition 2.1 (**Natural filtration**)**.** Given a probability space $(\Omega, \mathcal{F}, P)$ and standard Brownian motion $\{W_t\}_{t\geq 0}$, the natural filtration is defined as:

$$\mathcal{F}_t := \sigma\left(\{W_s^{-1}(E) : \forall E \in \mathcal{B}(\mathbb{R}) \text{ and } 0 \leq s \leq t\}\right), \tag{2.5}$$

and where W_t^{-1} denotes the pre-image of W_t.

2.1.1. Weak and strong solutions

One may now wonder what characterises the solution to an SDE. Here, the one-dimensional case is discussed. For a treatment of the multi-dimensional case, see [24, p.568-571]. We had already established that a solution should satisfy the stochastic integral in equation 2.4. We will distinguish between a strong solution, in which case the solution is adapted to $\mathcal{F}_t$ and defined on the Brownian motion $\{W_t\}_{t\geq 0}$ from the previous section, and a weak solution, which only satisfies the stochastic integral in distribution, but is potentially adapted to a different filtration and driven by a different Brownian motion [1][26, p.262]. Suppose that the processes of interest are defined on a time interval $[0, T]$, not affecting generality, since any bounded T may be chosen. The notions of weak and strong solutions are more rigorously defined in the following definitions, adapted from [24], chapter 26.

Definition 2.2 (**Weak solution**)**.** Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $S_0 = x \in \mathbb{R}$. Let $\{W_t\}_{t\in[0,T]}$ be a standard Brownian motion adapted to $\mathcal{F}_t = \sigma(\{W_t\})$. A *weak solution* is then given by the pair $\left(\{S_t\}_{t\in[0,T]}, \{W_t\}_{t\in[0,T]}\right)$, which satisfies equation 2.4 between any two times on $[0, T]$. $\{S_t\}_{t\in[0,T]}$ is a continuous-time random process adapted to $\mathcal{F}_t$.
If there exists a process $\{\tilde{S}_t\}_{t\in[0,T]}$, belonging to another weak solution, defined on $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{P})$, such that $\tilde{P} \circ \tilde{S}_t^{\ -1} = P \circ S_t^{-1}$, i.e. equality in law, then the solution is called *weakly unique*.

Definition 2.3 (**Strong solution**)**.** Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $S_0 = x \in \mathbb{R}$. Let $\{W_t\}_{t\in[0,T]}$ be the Brownian motion defined in the SDE formulation. A *strong solution* is then given by the pair of continuous random processes $\left(\{S_t\}_{t\in[0,T]}, \{W_t\}_{t\in[0,T]}\right)$ related by the map:

$$\phi : \mathbb{R} \times C([0, T]; \mathbb{R}) \to C([0, T]; \mathbb{R}), \tag{2.6}$$

i.e. which maps $(S_0, \{W_s\}_{t\in[0,T]}) \mapsto \{S_t\}_{t\in[0,T]}$, and which satisfies the following:

(*i*) The map $(x, w) \mapsto \phi(x, w)$ is measurable with respect to $\mathcal{B}(\mathbb{R}) \otimes \mathcal{H}_t$, where (x, w) are dummy variables for a realisation of S_0 and a Brownian motion, $\mathcal{H}_t$ is the σ-algebra generated by all possible Brownian motions up to time t, i.e. $\mathcal{H}_t := \sigma\{\pi_s : s \in [0, t]\}$ and $\pi_s := C([0, T]; \mathbb{R}) \to \mathbb{R}$ is a Brownian motion map $w \mapsto w(s)$.

(*ii*) The process $\{S_t\}_{t\in[0,T]} = \phi(S_0, \{W_t\}_{t\in[0,T]})$ satisfies equation 2.4 for all $t \in [0, T]$.

Remark 2.4. Note how different the concepts of weak and strong solutions are. Weak solutions may even be defined on a different probability space than the one on which the SDE is defined, but only have to satisfy the Itō integral given $A(t, S_t)$ and $B(t, S_t)$ and some Brownian motion on the probability space.

In [24] it is shown that the map ϕ defining a strong solution is unique. The characterisation of a strong solution as the map ϕ contains the adaptedness requirement to $\sigma(\{W_t\})$ on which the SDE is defined. Given this Brownian motion, for each realisation there is a unique process $\{S_t\}$, adapted to $\sigma(\{W_t\}_{t\in[0,T]})$. A useful concept for understanding the differences between the types of solutions is given in the following definition, reproduced from [24].

Definition 2.5 (**Path-wise uniqueness**)**.** A solution to the SDE (weak or strong) is called *path-wise unique* if for any two weak solutions $(\{S_t\}_{t\in[0,T]}, \{W_t\}_{t\in[0,T]})$ and $(\{\tilde{S}_t\}_{t\in[0,T]}, \{\tilde{W}_t\}_{t\in[0,T]})$ on the same probability space $(\Omega, \mathcal{F}, P)$, the paths are equal P-a. s.:

$$P\left(S_t = \tilde{S}_t \ \forall t \in [0,T]\right) = 1. \tag{2.7}$$

Uniqueness of the strong solution implies that, if it exists, it is path-wise unique.

Remark 2.6. As noted in [24], we should be careful in our interpretation of the 'uniqueness' of the strong solution. The map ϕ is unique, but individual realisations of the Brownian motion and the resulting filtration are not, as they are random variables.

Example 2.7. Consider a trivial example: $dS_t = dW_t$ and $S_0 = 0$. Clearly, $S_t = W_t$ satisfies the SDE. Since it is adapted to $\mathcal{F}_t = \sigma(\{W_t\})$, it forms a strong solution. The map ϕ is the identity in this trivial case. $S'_t = -W_t$ is a weak solution, since $W'_t := -W_t$ is a Brownian motion and S'_t is adapted to $\sigma(\{W'_t\})$. It is not a strong solution, as it is not path-wise unique to $\{S_t\}$, which immediately follows from $P(W_t = W'_t) = 0$. Both solutions are weakly unique, as they are equal in distribution.

A strong solution exists if S_0 is independent of $\mathcal{F}_t$ and $\mathbb{E}[S_0^2] < \infty$, $\sup_{t\in[0,T]} \mathbb{E}[S_t^2] < \infty$ and if $A(t, S_t)$ and $B(t, S_t)$ are K-Lipschitz for some K [2]. A weak solution exists if $A(t, S_t), B(t, S_t)$ are bounded and continuous and $|B(t, S_t)|$ is always greater than some $\varepsilon > 0$ $\forall t \in [0, T]$ [2]. If a process is a strong solution, it is also a weak solution [2, 24], as follows directly from the definitions.
The conditions stated for strong convergence are sufficient, but not necessary, as shown in [2] with an example of an SDE that violates the Lipschitz conditions stated here, but strongly satisfies the Itō integral. Some authors provide slightly different conditions, see for example [26], but an in-depth discussion about these details is not relevant to this thesis. The key idea is that we can distinguish between solutions to the SDE that only satisfy the Itō integral in distribution and those that hold path-wise from the defining Brownian motion of the SDE. In chapter 4, weak and strong solutions will be revisited on a discretisation of the interval $[0, T]$, on which the solution will be approximated by a neural network and discrete-time schemes. In the next section, we discuss these discrete-time approximation schemes, which will be used to construct a reference solution for comparison with the neural network approximation.

2.1.2. Discrete-time approximations

In general, solving an SDE analytically is highly challenging and only possible in special cases [2]. A way to approximate SDEs numerically is by approximating the stochastic integral in equation 2.4, cf. [2, Ch.2]. Kloeden and Platen show [2, p.78] how a Taylor expansion in integral form can be generalised to stochastic calculus by using Itō's lemma in the expansion. Proceeding with the same notation, and assuming that the processes $A(t, S_t)$ and $B(t, S_t)$ are respectively once and twice differentiable w.r.t. S_t, the stochastic Taylor expansion of equation 2.4 is given by [2, p.79]:

$$\begin{aligned} S_{t+\Delta t} = S_t + A(t,S_t)\int_t^{t+\Delta t} ds + B(t,S_t)\int_t^{t+\Delta t} dW_s + B(t,S_t)\frac{\partial B(t,S_t)}{\partial S_t}\int_t^{t+\Delta t}\int_t^s dW_\tau dW_s \\ + \bar{R}(t,S_t), \end{aligned} \tag{2.8}$$

where $\bar{R}(t, S_t)$ is the remainder of the expansion, containing integrals of order higher than dt. Thus, if we were to omit the remainder, we would arrive at an approximation of the stochastic integral up to order dt. If the double integral over the Brownian motions in equation 2.8 is also omitted, one arrives at the Euler-Maruyama scheme as approximation of the stochastic integral [2, 27], given by:

$$\hat{S}_{t+\Delta t} \approx S_t + A(t,S_t)\Delta t + B(t,S_t)\Delta W_t. \tag{2.9}$$

Here, $\Delta W_t := W_{t+\Delta t} - W_t$, i.e. a Brownian motion increment over the time step Δt. If the double integral over the Brownian motions in equation 2.8 is taken into account, omitting only the remainder term, a more accurate approximation is obtained, called the Milstein scheme, cf. [2, 28], given in equation 2.10. The term $\int_t^{t+\Delta t}\int_t^s dW_\tau dW_s$ is given by $\frac{1}{2}\left(\Delta W_t^2 - \Delta t\right)$ [2].

$$\hat{S}_{t+\Delta t} \approx S_t + A(t,S_t)\Delta t + B(t,S_t)\Delta W_t + \frac{1}{2}B(t,S_t)B'(t,S_t)\left(\Delta W_t^2 - \Delta t\right), \tag{2.10}$$

B' denotes $\frac{\partial B}{\partial S_t}$. Note how the Euler and Milstein schemes emphasise the difference between stochastic and deterministic calculus, as commented in [2, p.142]. The extra term in the Milstein scheme captures the additional contribution to the differential operator, that arises from Itō's lemma. The remainder term could be further expanded, but gives rise to less tractable expressions that are harder to simulate numerically [2], while the additional accuracy that this might give is not required in this work. Therefore, in this thesis, the Euler and Milstein scheme are considered as numerical approximations to the stochastic integral.

2.1.3. Weak and strong convergence for discrete-time approximations

Now that methods have been established to approximate the stochastic integral, we need a way to evaluate how well the stochastic integral is satisfied, in the same weak and strong sense that we saw in the previous section. Recall that a sequence of random variables $\{X_n\}_{n\geq 1} \in \mathbb{R}^n$ converges weakly to a random variable $X \in \mathbb{R}^n$, if for all bounded Borel functions f on $\mathbb{R}^n$ we have, see for example [29, ch. 18]:

$$\lim_{n\to\infty} \int f(x)dP_n = \int f(x)dP, \tag{2.11}$$

where $\{P_n\}_{n\geq 1}$ is the sequence of probability measures associated with $\{X_n\}_{n\geq 1}$. Equivalently, one can say that X_n converges in distribution to X, i.e. $X_n \overset{d}{=} X$ [29]. Or, rewriting equation 2.11 in terms of expectations: $\lim_{n\to\infty} \mathbb{E}\left[f(X_n)\right] = \mathbb{E}\left[f(X)\right]$. A weak solution can be evaluated by studying how well the convergence in distribution is achieved, i.e. by considering the moments or other statistics. For discrete-time schemes, the rate of weak and strong convergence can be expressed in terms of the time step Δt [2]. A discrete-time scheme is said to be weakly convergent with order β, if for functions f in the class of real-valued polynomials, $\exists$ constants K and $\beta \in (0,\infty]$, such that:

$$e_w := \left|\mathbb{E}\left[f(S_{t+\Delta t})\right] - \mathbb{E}\left[f\left(\hat{S}_{t+\Delta t}\right)\right]\right| \leq K\Delta t^{\beta}, \tag{2.12}$$

where $\hat{S}$ denotes the discrete-time approximation of the stochastic integral. e_w is called the weak error. The Euler and Milstein schemes are weakly convergent with order $\beta = 1$. Weak convergence only compares the distribution of the discrete approximation to the exact solution at the time $t + \Delta t$. For some applications, a discrete-time scheme should not only converge in distribution at a single time, but converge to the strong solution path-wise, which is denoted by strong convergence [2]. This means that discrete-time schemes will converge to strong solutions to the stochastic integral. A discrete-time approximation of $S_{t+\Delta t}$ converges in the strong sense if $\exists$ constants $\bar{M}$ and $\bar{\gamma} \in (0,\infty]$ such that:

$$e_s := \mathbb{E}\left|S_{t+\Delta t} - \hat{S}_{t+\Delta t}\right| \leq \bar{M}\Delta t^{\bar{\gamma}}. \tag{2.13}$$

e_s is called the strong error. The Euler scheme is strongly convergent with order $\bar{\gamma} = \frac{1}{2}$, while the Milstein scheme achieves strong convergence of order 1 [2].

2.1.4. SDEs under consideration

Two SDEs are under consideration in this thesis: Geometric Brownian motion (GBM) (cf. for example [30]), giving rise to the famous Black-Scholes equation for options, and the Cox Ingersoll Ross (CIR) process [31], which is a model for interest rates. Of both SDEs, the distribution conditional on any 'previous' value is known analytically. The Itō SDEs for both processes are shown in equations 2.14 and 2.15, cf. [30] and [31], respectively.

$$\text{GBM}: \quad dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{2.14}$$

$$\text{CIR}: \quad dS_t = \kappa(\bar{S} - S_t)dt + \gamma\sqrt{S_t}dW_t, \tag{2.15}$$

where both processes start at some initial value $S_0 \in \mathbb{R}$. $\mu, \sigma, \gamma, \kappa$ and $\bar{S}$ are constants.

For GBM, application of Itō's lemma on the process $\log S_t$ allows one to immediately derive [30, p. 226]:

$$\begin{aligned} S_{t+\Delta t} &= S_t e^{(\mu-\frac{1}{2}\sigma^2)\Delta t+\sigma(W_{t+\Delta t}-W_t)} \\ &\overset{d}{=} S_t e^{(\mu-\frac{1}{2}\sigma^2)\Delta t+\sigma\sqrt{\Delta t}Z}, \end{aligned} \tag{2.16}$$

with $Z \sim N(0,1)$. Thus, S_t follows a lognormal distribution.
In the case of the CIR process, one can show that $S_{t+\Delta t} \mid S_t$ follows a scaled non-central χ^2-distribution with some non-centrality parameter ξ, degrees of freedom δ and scaling factor $\bar{c}$ [31]:

$$S_{t+\Delta t} \mid S_t \ \sim \ \bar{c}\, \chi^2(\xi, \delta), \tag{2.17}$$

where $\bar{c}, \xi$ and δ are related to the SDE parameters as shown in equations 2.18-2.20, cf. [31, p.392].

$$\xi(S_t, \Delta t) = \frac{4\kappa S_t e^{-\kappa\Delta t}}{\gamma^2\left(1-e^{-\kappa\Delta t}\right)}, \tag{2.18}$$

$$\delta = \frac{4\kappa\bar{S}}{\gamma^2}, \tag{2.19}$$

$$\bar{c}(\Delta t) = \frac{\gamma^2}{4\kappa}\left(1-e^{-\kappa\Delta t}\right). \tag{2.20}$$

Note that the availability of an exact solution is not required for providing a training set to the neural network, as we could alternatively train on a discrete-time approximation of high accuracy. However, using the exact solution allows the sampling of arbitrary amounts of data for any set of parameters at low computational cost. Secondly, it allows easy comparison of the exact distribution to the output of the neural network approximation that will be obtained, again for any set of parameters. If the SDE were sufficiently complex that a Monte Carlo scheme would be required, for each parameter set, one would have to first obtain a large enough batch of samples to construct an accurate reference solution. This makes both SDEs attractive for the purposes in this thesis. GBM serves as a benchmark, being one of the simplest SDEs in quantitative finance, while the CIR process is richer in complexity, exhibiting near-singular behaviour around zero for choices of parameters such that $\delta < 2$ [31], allowing the process to 'hit zero'. The condition $\delta \geq 2$ ensures that the process does not exhibit this near-singular behaviour near zero, which is known as the Feller condition [32]. In future work, one could consider employing richer models, such as the Heston [33] or SABR [34] models.

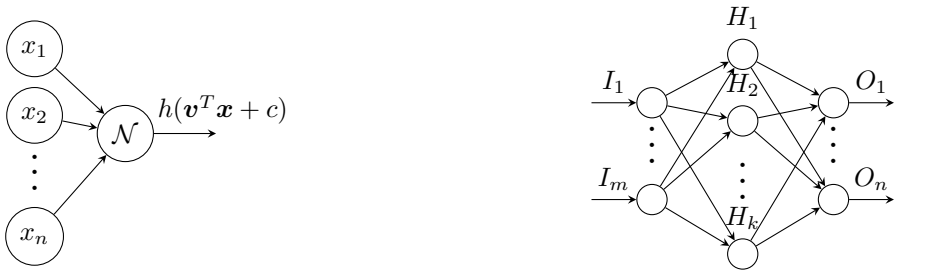
2.2. Artificial Neural Networks

This section explains the key techniques that underpin neural networks. In the subsequent chapter, a specific algorithm involving neural networks is discussed, which forms the main tool for approximating the stochastic integral in equation 2.8. This section motivates the use of neural networks and provides the basic architecture behind them. Then, it is shown how the parameters in such networks are updated if a loss function is provided, with a description of the algorithm used to find the optimal set of parameters. Finally, modern alternatives to the feed-forward neural network are briefly discussed.

2.2.1. Architecture

A neural network is a set of weights, biases and activation functions, whose relations can be modelled as a directed graph, cf. for example [35, ch.5], on which this section is based. Consider first a simple case with a single 'neuron', as shown in figure 2.1a. Suppose that the neuron is connected with a set of n inputs $x_1, \ldots, x_n$. Each connection has its own weight v_i for input i. Let us write the input and weights as vectors $\boldsymbol{x}, \boldsymbol{v} \in \mathbb{R}^n$. The neuron also has an activation function $h(\cdot)$ and bias $c \in \mathbb{R}$. The output of the neuron is then given by: $h(\boldsymbol{v}^T\boldsymbol{x} + c)$. The activation function is typically chosen to be some (piece-wise) differentiable, non-linear function on $\mathbb{R}$, e.g. a sigmoid. More details about the choice of these functions are given in a later section.
A feed-forward neural network or multi-layer perceptron is based on the artificial neuron, repeated over multiple layers of neurons, as illustrated in figure 2.1b. A layer is a collection of nodes which have an equal amount of connections with other nodes, shown as the vertically aligned nodes in the graph.

The input nodes $\{I_i\}_{i=1}^n$ and output nodes $\{O_i\}_{i=1}^d$ in figure 2.1b are themselves shown as layers. Layers that are neither input nor output layers are called hidden layers, containing hidden neurons. Each connection in the graph includes an independently variable weight. In each neuron, the basic operation shown in figure 2.1a is performed. This architecture may be repeated many times, giving rise to multiple hidden layers. The amount of hidden layers is referred to as the network depth, while the amount of nodes in each layer is called the width. The outputs of each layer become the inputs to the subsequent layer.



(a) A single artificial neuron '$\mathcal{N}$'

(b) Feed-forward NN with one hidden layer

Figure 2.1

Although the illustration in figure 2.1 is useful for conceptually understanding neural networks, for a mathematical formulation we only require the weights, biases and activation functions. Suppose we have a feed-forward NN with d hidden layers and $m_i \in \{m_0, \ldots, m_{d+1}\}$ neurons in each layer, with m_0 and m_{d+1} the input and output dimensions of the network, respectively. Let $\mathcal{A}^i$ for $i \in \{1, \ldots, d+1\}$ denote the operator $\mathcal{A}^i(\boldsymbol{x}) := M^i\boldsymbol{x} + \boldsymbol{c}^i$, where $M^i \in \mathbb{R}^{m_i \times m_{i-1}}$ and $\boldsymbol{c}^i \in \mathbb{R}^{m_i}$ are respectively the matrix of weights and the biases in layer i. Let us write the set of all parameters in the neural network as $\boldsymbol{\theta} \in \mathbb{R}^{\bar{\nu}}$, i.e. a vector with a concatenation of all parameters $\{M^i, \boldsymbol{c}^i\}_{i=1}^{d+1}$, and $\bar{\nu} \in \mathbb{N}$. The total amount of parameters is given by $\bar{\nu} = \sum_{i=1}^{d+1} \big[m_i(m_{i-1}+1)\big]$.
A feed-forward neural network with d hidden layers can then be written as the function $g_{\boldsymbol{\theta}} : \mathbb{R}^{\bar{\nu}} \times \mathbb{R}^{m_0} \to \mathbb{R}^{m_{d+1}}$ given in equation 2.21.

$$g_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{L}^{m_{d+1}} \circ \mathcal{A}^{d+1} \circ \ldots \circ \mathcal{L}^{m_2} \circ \mathcal{A}^2 \circ \mathcal{L}^{m_1} \circ \mathcal{A}^1(\boldsymbol{x}), \tag{2.21}$$

where $\mathcal{L}^{m_i} : \mathbb{R}^{m_i} \to \mathbb{R}^{m_i}$ is the vector-valued activation function of the layer i with m_i neurons, i.e. $\mathcal{L}^{m_i}(\boldsymbol{x}) := [h(x_1), h(x_2), \ldots, h(x_{m_i})]$. Note that in principle we could even vary the activation function per layer, but here it is chosen to be the same function h all layers, which is common.

2.2.2. Universal approximation with neural networks

The expressive power in neural networks comes from the non-linear activation function in combination with the hidden layer. It has been shown [36, 37] that the mapping achieved by neural networks with a single hidden layer can approximate any measurable function with an arbitrary degree of accuracy, as long as the width (i.e. nodes in the hidden layer) is large enough. This then automatically holds for any bounded number of hidden layers, as the succeeding layers can approximate the identity function. This theorem is known as the universal approximation theorem [36] and concerns itself with fixed depth and arbitrary width.
The universal approximation theorem has been extended [38, 39] to the case of bounded width but arbitrary depth. As is clear from the formulation of a feed-forward NN in the previous section, the amount of parameters grows rapidly with both the width and depth. For practical purposes, it becomes relevant whether one needs, say, millions of parameters in a single layer or only tens of thousands to approximate the target function of interest. In [38], an extension is given to networks with bounded width and arbitrary depth, for the activation function $f(x) = \max(x, 0)$, while [39] extends this to any non-affine activation function which is continuously differentiable at least at one point. How to balance depth versus width in general is still an open question [38], but modern deep neural networks can reach tens to over one thousand layers, as shown for example in AlexNet (8 layers) [40], VGG (19 layers) [41], GoogLeNet (22 layers) [42] or 152 and even 1001 layers with so-called 'residual networks' [43].

The concept of universal approximation is stated for reference, to show the motivation behind using neural networks as universal function approximators. NNs are able to learn arbitrary mappings from an input to an output space. The algorithm used in this thesis to approximate the stochastic integral requires additional background, as the Itō integral and the output of the neural network are random variables. In the next chapter it is shown how the algorithm used in this thesis converges to the distribution of the target, where neural networks appear as tools for approximating the functions specified in the algorithm.

2.2.3. Gradient descent and backpropagation

Now we turn to the process of updating the weights and biases such that the neural network given in equation 2.21 approximates any target function of interest. This section is based on [35, p.240-245]. First, we need some way to compare the output of the network $g_{\boldsymbol{\theta}}$ from equation 2.21 to the desired target output. On an image classification problem, for example, this could be a Boolean vector $\boldsymbol{y}$ where a 1 in index i means that the output belongs to the class i. Given a dataset $\boldsymbol{x}$, the error of the output could then for example be quantified as $L(g_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) := \|g_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2$, with $\|\cdot\|_2$ the l^2-norm. The function L is called the loss function or cost function and quantifies the error between the target solution and output of the network. From here, let us assume that we always have access to a loss function L for a general problem, which may take more general forms than the l^2 norm.
The weights and biases in the neural network should then be updated such that the loss function L is minimised. Since there is no closed-form expression for the optimal choice of parameters $\boldsymbol{\theta}$ in a neural network, a technique called gradient descent is used to find the optimal parameter set numerically. The idea is to update the weights such that they move in the direction of minima in the hypersurface spanned by the loss function L, as a function of the NN weights $\boldsymbol{\theta}$. I.e, the weights are adjusted in the direction of negative gradient, 'downhill'. If this process is repeated iteratively, one can attempt to approximate the minima in this loss surface. Thus, the weights at iteration k are updated as:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \lambda \nabla_{\boldsymbol{\theta}_k} L, \tag{2.22}$$

for some constant $\lambda \in \mathbb{R}$, called the learning rate, which is a hyperparameter to regulate how much the weights are updated each iteration. There are now two challenges: 1) it is unclear whether gradient descent can find a unique global minimum and 2) obtaining the gradient. Both are now discussed.
It should be stressed that the loss function parameterised by the weights $\boldsymbol{\theta}$ is a highly complex surface in potentially thousands or billions of dimensions. This surface may be highly non-convex and replete with local minima. This is why many adaptations of 'standard' gradient descent have been developed to prevent convergence on local minima and to adapt the learning rate during the training process, cf. [44] for an overview of modern methods. A central challenge is the choice of learning rate λ. Choosing this parameter too small yields a slow 'exploration' of the loss surface, while a too high value may cause the descent operation to 'miss' parts of the loss surface and oscillate around minima [45]. Ideally, the learning rate should be adaptive and sensitive to the curvature of the loss surface, which is the central idea behind innovations to equation 2.22 [45].
Another consideration is that the entire dataset may be too large to be stored in memory [44]. To overcome this problem, one could subdivide the training set into subsets and train successively on each subset [44]. However, it would be required to iterate over all batches before updating the weights of the network, which is inefficient. An alternative called stochastic gradient descent (SGD) has been proposed [46]. In this case the loss function over the whole dataset is replaced by a Monte Carlo estimate of the loss function, by sampling randomly chosen batches from the training set:

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \lambda \nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{\boldsymbol{x}\sim X}\left[L(g_{\boldsymbol{\theta}_k}(\boldsymbol{x}), \boldsymbol{y})\right], \tag{2.23}$$

where X denotes a training set of examples that the neural network is approximating with desired output $\boldsymbol{y}$. This way, the cost of approximating the gradient becomes independent of the size of the training set, but introduces the choice of how large the 'batch' of Monte Carlo samples from X should be.
Now, finally, let us consider how the gradient of the loss function w.r.t the vector $\boldsymbol{\theta}$ may actually be obtained. This may seem daunting given the layered architecture and emerging complexity of the neural network, but can be carried out efficiently by using how each layer depends on the previous one. The formulation in equation 2.21 helps us here. Let us write the output at the ith layer as

$y_i := \mathcal{L}^i \circ \mathcal{A}^i(y_{i-1})$, with $i = 1, \ldots, d+1$. Suppose that the parameter θ_j is present in layer i. Let us assume that L and $h(\cdot)$ are differentiable and write $\hat{L}$ as the Monte Carlo estimate of L, computed with SGD. The derivative of the loss function w.r.t. θ_j can then be found by using the chain rule, as:

$$\frac{\partial \hat{L}}{\partial \theta_j} = \frac{\partial \hat{L}}{\partial y_{d+1}} \frac{\partial y_{d+1}}{\partial y_d} \cdots \frac{\partial y_i}{\partial \theta_j}. \tag{2.24}$$

This technique is known as backpropagation, as the error is propagated 'back' from later layers into earlier ones, introduced in [47] and is crucial to deep neural networks. Since we know the activation functions and weights in each neural network, we do not need any finite differences or similar techniques to approximate equation 2.24. Instead, it can be evaluated analytically, by using another technique called automatic differentiation, cf. [48]. This technique stores the gradients of the output with respect to the parameters at each node in the network as a graph, through which the gradient $\nabla_{\boldsymbol{\theta}}$ can be computed analytically for each θ_j.

2.2.4. Adam

As introduced in the previous section, various methods for stochastic gradient descent have been proposed [44] and covering all of them goes beyond the scope of this thesis. However, one particular weight optimisation algorithm or optimiser appears to be highly popular and effective in modern deep learning research, and will be discussed here. It is called 'Adam' or 'Adaptive Moment Estimation', introduced by [49], in which the authors show that their method outperforms earlier optimisers on a range of typical machine learning problems. Adam updates the weights not on the gradient itself, but on moving average estimates of the first two moments of the gradient at iteration k: $\boldsymbol{m_k}$ and $\boldsymbol{v_k}$. If we let $\hat{\boldsymbol{u}}_k := \nabla_{\boldsymbol{\theta}_k} L$, these estimates can be written as follows [49]:

$$\boldsymbol{m}_k = \beta_1 \boldsymbol{m}_{k-1} + (1-\beta_1)\hat{\boldsymbol{u}}_k, \tag{2.25}$$

$$\boldsymbol{v}_k = \beta_2 \boldsymbol{v}_{k-1} + (1-\beta_2)\hat{\boldsymbol{u}}_k^2, \tag{2.26}$$

where the $\cdot^2$ operation denotes the element-wise square $\hat{\boldsymbol{u}}_k \odot \hat{\boldsymbol{u}}_k$ and $\beta_1, \beta_2 \in [0, 1)$ are hyperparameters. Both these processes are initialised at 0 at $k = 0$, which makes the first iterations biased towards zero. This is corrected by scaling the processes as $\hat{\boldsymbol{m}}_k := \frac{\boldsymbol{m}_k}{(1-\beta_1)}$ and $\hat{\boldsymbol{v}}_k := \frac{\boldsymbol{v}_k}{(1-\beta_2)}$. A small bias remains present due to non-stationarity of the moments, but this is mitigated by the moving average nature of the process, which exponentially decays its dependency on values farther in the 'past'.
Based on these estimates, the weight update of the Adam optimiser is given by:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta\, \hat{\boldsymbol{m}}_k \oslash (\sqrt{\hat{\boldsymbol{v}}_k} + \epsilon), \tag{2.27}$$

where '$\oslash$' denotes element-wise division between vectors, η is a fixed 'base' learning rate and ϵ is some very small value, e.g. chosen in [49] to be 10^{-8}, to keep the update defined for initialisation at 0. The authors of [49] compare the ratio $\hat{\boldsymbol{m}}_k \oslash \sqrt{\hat{\boldsymbol{v}}_k}$ to a 'signal-to-noise ratio' for the gradient, which can be interpreted as follows: if the gradient is high, but additionally its second moment is high as well, then the update tends to be small. This is desirable, as the uncertainty in the value of the gradient would in that case be high. Alternatively, if the gradient is high compared to the second moment, updates will be larger. In [50], Adam is compared with a 'heavy ball with friction rolling downhill', where it is even cast into an ODE form to support the comparison. The relevant intuition is that Adam is less sensitive to local minima than traditional optimisers, as it 'shoots over' them to proceed to wider minima with smoother curvature.
Finally, the authors of [49] show in the seminal paper that the algorithm converges by proving that the sum of past differences between the loss function evaluated at $\boldsymbol{\theta}$ and the optimal choice of $\boldsymbol{\theta}$ is of order $\frac{1}{\sqrt{k}}$, i.e. decays towards zero with increasing k.

2.2.5. Activation functions

One of the key insights from the universal approximation theorem is that it holds for any non-polynomial activation function [39], although in [36] it is already commented that this does not imply that all activations work equally well. The optimal choice of activation function for general

neural network approximations is to this date unclear [39], however, several choices are widely known and shown to work well in practice. Four variants are discussed here: the logistic function, hyperbolic tangent, ReLU [51] and LeakyReLU [52], as they are prevalent in modern architectures [35, 52, 53]. Sigmoids or 'S-shaped' curves, are a common choice in neural networks, the most notable example being the logistic function and hyperbolic tangent. The logistic function is defined by:

$$h_1(x) = \frac{e^x}{1 + e^x}. \tag{2.28}$$

The hyperbolic tangent function is given by:

$$h_2(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.29}$$

Both activation functions share similar properties, being non-monotone, everywhere differentiable functions with an inflection point at 0 and saturating region at $\pm\infty$, i.e. both have a characteristic 'S'-like shape. The tanh function is anti-symmetric about the horizontal axis, i.e. $\tanh(x) = -\tanh(-x)$, which may be desirable if there is a symmetry in the intermediate layer outputs, although this is uncommon [35]. Sigmoids may also be used at the output of the network if it is desirable that the output values remain between 0 and 1 or -1 and 1, respectively. A sigmoid is commonly used in classification problems at the output [35] and also appears in the popular 'DCGAN' architecture [53], which generates images. A potential downside with these two activation functions is that the saturating region may cause gradients that tend towards 0, which is especially problematic in deeper networks [54].
The rectified linear unit (ReLU) proposed in [51] has the shape of a 'hockeystick' and is given in equation 2.30, along with its similar 'LeakyReLU' sibling in equation 2.31, introduced in [52]. The ReLU and LeakyReLU activations are respectively defined as:

$$h_3(x) = \max(x, 0), \tag{2.30}$$
$$h_4(x) = \max(x, 0) + \zeta \min(x, 0), \tag{2.31}$$

where $\zeta \in \mathbb{R}$ is a hyperparameter, typically chosen to be 0.01 or 0.1. A potential advantage of using the ReLU and LeakyReLU activation functions is that they do not saturate at $+\infty$, always providing a constant gradient if the function is 'on', i.e. not close to zero or equal to zero. However, the ReLU activation provides its own challenges, as the gradient is zero for $x < 0$ and undefined for $x = 0$. A gradient of 0 for a large set of input values may unintentionally prevent weights from updating. The LeakyReLU has been proposed in [52] to overcome this first problem, always providing some (although small) gradient in the region $x < 0$. There is, however, still the problem of both these activations being non-differentiable at 0, which is resolved in deep learning implementations by using a convention to cover this single point, e.g. $h'_{3,4}(0) := 0$. In `PyTorch`, for example, the derivative at 0 is set at 0 [55]. This does not seem to provide problems in practice [52].
ReLU and LeakyReLU have been found to perform better than sigmoids in multiple applications [51, 52, 56]. In [52], this is attributed to their lack of vanishing gradients and ability to create sparse representations of the data in intermediate layer outputs, increasing their efficiency.

2.2.6. Other types of NNs

In addition to feed-forward neural networks, there are many types of neural networks that have been proposed in deep learning literature over the past decades. Each serves different goals and has a different architecture, but is based on the same principles that are covered in this chapter. Notable examples of alternative architectures are the convolutional neural network (CNN) [57], residual network (ResNet) [43] and the recurrent neural network (RNN) [58–60]. CNNs use multidimensional arrays to perform convolution operations on input data, to extract high-level features from the data, which can be used for applications such as image classification [57]. ResNets add the inputs from previous layers to the output of succeeding layers. This prevents the gradient from decaying exponentially in hidden layers and allows for arbitrarily deep networks [43]. Finally, recurrent neural networks are a variation of feed-forward neural networks, which can be applied on sequences of arbitrary length. The RNN maintains a hidden state, i.e. the output of one or more hidden layers,

which is applied recursively back to the network, while still connected with the output layers of the network. This way, the network can become sensitive to dependencies over time through the evolution of the hidden state. Such networks and variants have become highly popular and used today for time-varying datasets, such as speech recognition [61], machine translation [62] and many more. This overview of possible NN architectures is by no means complete, but is intended to give an idea of how other architectures may be used to replace feed-forward NNs for specific tasks, including the problem set in this thesis.

Chapter Conclusion

This chapter has covered the necessary background behind the problem setting in this thesis. We distinguished between solutions to the SDE that only satisfy the Itō integral in distribution from solutions that satisfy the Itō integral path-wise. The stochastic integral can be approximated by means of a stochastic Taylor expansion, which is the basis for discrete-time schemes. In this thesis, the distribution of the exact solution to SDEs will be approximated by neural networks, where the universal approximation property will be used to construct mappings from a prior input to the target distribution. The next chapter will describe how neural networks can be used to approximate arbitrary probability distributions.

3

Generative Adversarial Networks

The main tool used throughout the thesis is the Generative Adversarial Network (GAN). This chapter describes the key properties of GANs, starting from a high-level overview, before proceeding to the fundamental mathematics behind them. Subsequently it is shown how the parameterised nature of neural networks affects the GAN. Limitations and variations of GANs are then presented, which are compared to the needs in this work. Finally, the base algorithm used in this work is presented.

3.1. Introduction

Generative Adversarial Networks (GANs) were first introduced by Goodfellow et al. in 2014 [63]. The original paper demonstrated novel and impressive results for the popular MNIST, CIFAR-10 and ImageNet datasets [63]. Follow-up research demonstrated that GANs perform well on higher resolution images of humans and animals, artificial bedrooms [53], image-to-image translation [64] or even text-to-image [65] generation. For an overview of several years of GAN research and a discussion of the many variants up to 2019, see [66]. Most sources mention the increased sharpness in the output compared to other deep generative models, such as variational autoencoders (VAEs).
Aside from images, GANs have also been applied on other problems, such as time series generation [23], PDE solving [10], outlier detection [9], synthetic medical data [67], smoke plume simulations [12], music generation [68] and many more. This wealth of applications is a tribute to the robustness of GANs across problem domains. GANs are interesting to the purposes in this work, as they form a powerful yet flexible tool for approximating complex probability distributions.

3.2. Theoretical Overview

This section is based on the original definition by Goodfellow et al. [63] and theoretical work on GANs by Biau et al. [69]. GANs consist of two neural networks, a generator, G, and a discriminator, D. The basic idea is as follows. The generator maps noise from an inexpensive prior to a potentially highly complicated distribution in some target space, such as images of human faces. The discriminator network alternatingly receives the output of the generator ('fake' sample) or a target sample ('real' sample). The inputs are labeled 0 or 1 for fake and real data, respectively. The discriminator is to predict whether a target is real or fake and its output is a number between 0 and 1. At the same time, the generator is to match the target data as closely as possible, 'fooling the discriminator'. Both networks are thus competing against each other and are jointly trained each iteration. This process should lead the generator to produce samples that the discriminator cannot distinguish from real images. If done well, these samples are not only indistinguishable by the discriminator, but also visually similar to the target data as judged from human observers. A high-level overview of the training procedure is illustrated in figure 3.1.

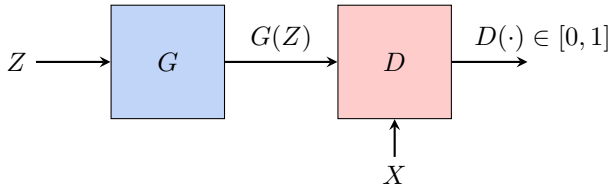


Figure 3.1: GAN schematic: a prior Z is mapped through $G(Z)$. D is trained to determine if a real sample X and a fake sample $G(Z)$ are real or fake. Generally, D and G need not even be neural networks, but could also be other function approximators.

3.2.1. Formal definition

With the basic idea in mind, let us make the definition more rigorous, following the notation in [69]. Suppose we define a Borel set on $\mathbb{R}^n$, denoted by E and define a measure ν on E which dominates all succeeding probability measures in this section. Let $Z \in \mathbb{R}^m$ be the random vector that serves as prior input to the generator. This variable is often denoted the 'latent variable'. Let $(X_1, X_2, \ldots, X_N)$ denote a set of target data of size N, where $X_i \in E \ \forall i$. The generator and discriminator networks can then be written as:

$$G_\theta : \mathbb{R}^m \to E, \tag{3.1}$$

$$D_\alpha : E \to [0, 1], \tag{3.2}$$

where $\theta \in \Theta \subset \mathbb{R}^p$ and $\alpha \in \Lambda \subset \mathbb{R}^q$ are the network parameter sets for some (typically very large) $p, q \in \mathbb{N}$. The dataset $(X_i)_{i=1}^N$ is sampled i.i.d. from the distribution of 'real samples' P^* with density p^* on E. Of course, in the general case, we do not know this density analytically, as we do for several SDE problems. The generator gives rise to a distribution P_θ and associated density parameterised by θ, say p_θ, dominated by ν. We can write $G_\theta(Z) \overset{d}{=} P_\theta$ or $G_\theta(Z) \overset{\mathscr{L}}{=} p_\theta d\nu$, where '$\overset{\mathscr{L}}{=}$' denotes equality in law. In general, p_θ and p^* do not lie in the same space of measures, so we have that $p^* \notin \{p_\theta\}_{\theta\in\Theta}$. This is due to the limited parameterised nature of the generator. The idea is that the GAN will approximate p^* with p_θ as closely as possible. The authors of [69] provide answers to how close the generator can approximate the target density, which will be shown in a later section. First we need to understand the loss function that the GAN attempts to minimise, then we will consider how well G_θ and D_α can find this optimum theoretically.

3.2.2. Optimisation problem

The generator is to maximise the value of the 'discriminator output given a fake image', which is equivalent to minimising $1 - D \circ G_\theta(Z_j)$, while the discriminator tries to maximise the value of the 'discriminator output given a real image AND (1 - discriminator output on fake image)'. The process is similar to a two-player zero-sum game in game theory [63] with value function $U(D, G)$. For more on game-theoretic minimax problems, cf. [70, ch. 12] or [71, p. 302-303]. The two objectives can be cast into the following joint optimisation problem, given dataset X and prior samples Z of sizes N_X and N_Z [69]:

$$\inf_{\theta\in\Theta} \sup_{D\in\mathscr{D}_\infty} U(D, G_\theta) = \inf_{\theta\in\Theta} \sup_{D\in\mathscr{D}_\infty} \left(\prod_{i=1}^{N_X} D(X_i) \prod_{j=1}^{N_Z} (1 - D \circ G_\theta(Z_j)) \right). \tag{3.3}$$

Equivalently, taking the logarithm and using its convexity, the GAN value function $V(D, G)$ is defined as:

$$\inf_{\theta\in\Theta} \sup_{D\in\mathscr{D}_\infty} V(D, G_\theta) = \inf_{\theta\in\Theta} \sup_{D\in\mathscr{D}_\infty} \Big[\sum_{i=1}^{N_X} \log D(X_i) + \sum_{i=j}^{N_Z} \log (1 - D \circ G_\theta(Z_j)) \Big], \tag{3.4}$$

where $\mathscr{D}_\infty$ denotes the set of all discriminators, i.e. all Borel functions from $\mathbb{R}^n \to [0, 1]$. Note that first we limit ourselves to the theoretical case where the discriminator has infinite capacity and is not parameterised by $\alpha \in \Lambda$. This is needed to find the theoretically optimal discriminator given any generator. The generator is given in its parameterised form by $\theta \in \Theta$.

The optimisation criterion can be cast into a loss function[1]. Let us distinguish between the continuous version of the problem with continuous data $x \in E$, and the previously described empirical discrete version with finite data $(X_i)_{i=1}^N$. For $\theta \in \Theta$ and $D \in \mathscr{D}_\infty$, we define a loss function L as:

$$L(\theta, D) = \mathbb{E}_{X\sim P^*}\left[\log(D(X))\right] + \mathbb{E}_{Z\sim P_Z}\left[\log\left(1 - D \circ G_\theta(Z)\right)\right] \tag{3.5a}$$

$$\hat{L}(\theta, D) = \frac{1}{N_X}\sum_{i=1}^{N_X} \log D(X_i) + \frac{1}{N_Z}\sum_{i=1}^{N_Z} \log\left(1 - D \circ G_\theta(Z_i)\right). \tag{3.5b}$$

Equation 3.5b represents the discrete version of equation 3.5a. Now let us define the optimal generator given the optimal choice of discriminator as the neural network with parameters $\hat{\theta}$ such that:

$$\sup_{D\in\mathscr{D}_\infty} \hat{L}(\hat{\theta}, D) \leq \sup_{D\in\mathscr{D}_\infty} \hat{L}(\theta, D) \quad \forall\theta \in \Theta. \tag{3.6}$$

It is easy to see that the choice of $\hat{\theta}$ corresponds to the infimum in equation 3.4, i.e. the minimax choice for the generator. Note that this representation is independent of the choice of the type of network, or even a neural network at all. It could be any approximator defined by parameter sets θ and α. As we will see, both D and G will not exactly reach their theoretical optima in practice. Approximation errors arise from two independent sources: 1) the finite parameter set of the discriminator and 2) the dataset real-world samples being finite and therefore not exactly representing P^*.

3.2.3. Optimality in the ideal case

Let us first treat the theoretical case where the discrminator can take any Borel function on E and the data samples are continuous variables $x \in E$. In this case we can write the densities described earlier explicitly as $p^*(x) = \frac{dP^*}{d\nu}(x)$ and $p_\theta(x) = \frac{dP_\theta}{d\nu}(x)$. In practice ν will denote the Lebesgue measure throughout this thesis.

First, a criterion for evaluating the loss L is that it does not diverge to $-\infty$. This possibility arises from the presence of logarithms of which the argument could become 0. Since $D \in \mathscr{D}_\infty$, this is not automatically excluded. The authors in [69] go around this by excluding such cases and considering the sets they call 'θ-admissible'. Here it will be defined similarly, following the authors:

Definition 3.1. A discriminator $D \in \mathscr{D}_\infty$ is called θ-admissible if $L(\theta, D) > -\infty \quad \nu-$a.e.
The set $\mathscr{D}'_\infty$ denotes the set of discriminators that are θ-admissible.

With this in mind and in the continuous setting, the optimal choice of discriminator given a generator G_θ is given by:

Theorem 3.2. *There exists a θ-admissible optimal discriminator $D \in \mathscr{D}'_\infty$ that attains the value* $\arg\sup_{D\in D'_\infty} L(\theta, D)$. *This discriminator is given by:*

$$D^*_\theta := \frac{p^*}{p^* + p_\theta}.$$

Proof. We can write the expectation in equation 3.5a explicitly, using dummy variable x, as:

$$L(\theta, D) = \int_{\mathscr{X}_1} \log\big(D(x)\big)p^*(x)d\nu(x) + \int_{\mathscr{X}_2} \log\big(1 - D(x)\big)p_\theta(x)d\nu(x). \tag{3.7}$$

Here, $\mathscr{X}_1$ and $\mathscr{X}_2$ represent $\text{supp}(P^*)$ and $\text{supp}(P_\theta)$, respectively. The set $\mathscr{X}_1$ corresponds to all possible observations of real data. The set $\mathscr{X}_2$ corresponds to the possible fake samples from P_θ. Now consider a sample $x' \in \mathscr{X}_1$ but $x' \notin \mathscr{X}_2$, i.e. an observation present in the 'true data' but not in the space of possible generated data. Then we would have $p_\theta(x') = 0$. Similarly, for a measurement y' with $y' \in \mathscr{X}_2$ but $y' \notin \mathscr{X}_1$, we have $p^*(y') = 0$. This step has been added in addition to the proof given by

[1] In practice, the loss function used is $-L(\theta, D)$, since neural network optimisation packages use gradient descent instead of gradient ascent. This is shown in more detail later in this chapter.

Biau et al. We can thus always[2] integrate over the union of both sets: $E \supseteq \mathscr{X} := \mathscr{X}_1 \cup \mathscr{X}_2$. Using this, both integrals can be combined:

$$\begin{aligned} \sup_{D \in D'_\infty} L(\theta, D) &= \sup_{D \in D'_\infty} \int_{\mathscr{X}} \Big(\log(D(x)) p^*(x) + \log\big(1 - D(x)\big) p_\theta(x) \Big) d\nu(x) \\ &\leq \int_{\mathscr{X}} \sup_{D \in D'_\infty} \Big(\log(D(x)) p^*(x) + \log\big(1 - D(x)\big) p_\theta(x) \Big) d\nu(x) \\ &= L(\theta, D^*_\theta). \end{aligned} \tag{3.8}$$

The first inequality is trivial, since for any continuous functional $f(g(x))$ on $\mathscr{X}$ and $g(x) \in \mathscr{G}$, with $\mathscr{G}$ some family of functions on $\mathscr{X}$, we have that: $\int_{\mathscr{X}} f(g(x))dx \leq \int_{\mathscr{X}} \sup_g f(g(x))dx$. But then, taking the supremum over g on both sides, the result follows:

$$\sup_g \int_{\mathscr{X}} f(g(x))dx \leq \sup_g \int_{\mathscr{X}} \sup_g f(g(x))dx = \int_{\mathscr{X}} \sup_g f(g(x))dx.$$

The last step in equation 3.8 follows from the fact that $y \mapsto \alpha \log(y) + \beta \log(1-y)$ is maximised by $\frac{\alpha}{\alpha+\beta}$ for $\alpha, \beta \in \mathbb{R}$, as is easily shown with standard arguments (e.g. setting the derivative w.r.t. y to zero). Since D is a continuous function on $\mathscr{X}$ by construction, the identity holds for the functional of $D(x)$ with coefficients p^* and p_θ as well. This proves the result, as D^*_θ is the discriminator corresponding to the supremum of $L(\theta, D)$.

□

Notice how for $p^* = p_\theta$, the optimal discriminator equals $1/2$, which is in line with intuition: the discriminator flips a coin whether a sample is real or fake, showing it cannot distinguish between the two distributions. This suggests that if the generator approximates p_θ well near a data point x, the discriminator should output a value close to $D(x) = 1/2$ at that data point. The authors of [69] go on to prove uniqueness of the optimal discriminator in the continuous setting.

Theorem 3.3. *The optimal discriminator D^*_θ is unique.*

The proof follows from a convexity argument involving the terms under the integrals in the loss function. The reader is referred to [69, pp.5-6] for a full proof.

3.2.4. Connection with the JS-divergence

The uniqueness of D^*_θ means that the loss function has a global minimum for D, which is also presented in the original paper by Goodfellow et al. [63, p.5]. The minimiser can actually be expressed in a distributional divergence known as the Jensen-Shannon divergence (JS-divergence), as introduced in [72].

To that end, we first need to understand another divergence, know as the Kullback-Leibler divergence (KL-divergence), cf. [73]. Suppose we have two probability distributions, P and Q, with absolutely continuous probability measures, dominated by ν. The KL-divergence of Q from P is then defined as:

$$KL(P\|Q) = \int \log \frac{p(x)}{q(x)} p(x) d\nu(x), \tag{3.9}$$

where $p(x)$ and $q(x)$ are the densities associated with P and Q. It is common to write $KL(p\|q)$, which means the same as equation 3.9. The JS-divergence is defined in terms of the KL-divergence as follows:

$$\begin{aligned} &JS(P\|Q) = \frac{1}{2} KL(P\|M) + \frac{1}{2} KL(Q\|M), \\ &\text{where } \; M := \frac{P+Q}{2}. \end{aligned} \tag{3.10}$$

Now the connection will be shown between discriminator optimality and the JS-divergence, using a similar approach as [63] and [69].

[2] That is, if indeed D is θ-admissible for all $x \in \mathscr{X}$.

Theorem 3.4 (**Connection with the JS-divergence**). *The following statements hold:*

(i) $L(\theta, D_\theta^*)$ *can be written as* $-2\log 2 + 2\ JS(p^* \| p_\theta)$,

(ii) $L(\theta, D_\theta^*)$ *is minimised if and only if* $p^* = p_\theta$.

Proof. From theorem 3.3 we know that the minimiser to equation 3.7 exists and is unique. If we plug D_θ^* into equation 3.7, we readily see the JS-divergence appear:

$$\begin{aligned} L(\theta, D_\theta^*) &= \int_{\mathscr{X}_1} \log\left(\frac{p^*}{p^* + p_\theta}\right) p^* dx + \int_{\mathscr{X}_2} \log\left(\frac{p_\theta}{p^* + p_\theta}\right) p_\theta dx \\ &= -2\log 2 + \int_{\mathscr{X}_1} \log\left(\frac{2p^*}{p^* + p_\theta}\right) p^* dx + \int_{\mathscr{X}_2} \log\left(\frac{2p_\theta}{p^* + p_\theta}\right) p_\theta dx \\ &= -2\log 2 + 2\ JS(p^* \| p_\theta), \end{aligned} \tag{3.11}$$

where in the last step, the symmetry of the JS-divergence has been used. Clearly, the minimum would be attained if $JS(p^* \mid\mid p_\theta) = 0$, due to the non-negativity of the JS-divergence. In [72] it is shown that the JS-divergence achieves 0 if and only if $p_\theta = p^*$. With that, the result follows and the minimum value of L is $-2\log 2$. □

3.2.5. Optimal generator

In general, it may not be possible to have $p_\theta = p^*$, since the parameterised generator may not be in the space of p^*. However, given a parameter space Θ, the question is now whether there exists a $\theta \in \Theta$ such that the final expression in equation 3.11 is minimised over all possible θ's. This will define the optimal generator, with parameter set $\theta^* \in \Theta$ as:

$$L(\theta^*, D_\theta^*) \leq L(\theta, D_\theta^*)\ \forall\ \theta \in \Theta, \tag{3.12}$$

i.e. the optimal generator given the optimal parameterised discriminator. Or equivalently, a generator with parameter set θ^* that satisfies:

$$\theta^* = \arg\min_{\theta \in \Theta}\ JS(p_\theta \| p^*) \tag{3.13}$$

Biau et al. show [69, Theorem 2.2] that since $\{P_\theta\}_{\theta \in \Theta}$ is compact for the metric $\sqrt{JS(\cdot\|\cdot)}$, the map $p \mapsto JS(p, p^*)$ is continuous and thus the minimum p_{θ^*} exists. Furthermore, Biau et al. go on to show that the minimum p_{θ^*} is unique. After that, however, the authors assume that the generator distribution P_θ is identifiable for each $\theta \in \Theta$, which allows them to conclude that θ^* is itself also unique. This assumption is in general violated when using neural networks, as will be shown in the next subsection. Therefore, we should disregard the proofs for unicity of θ^* given in [69]. The existence of θ^* and the uniqueness of p_{θ^*}, do however apply to our setting.

3.2.6. Non-unicity of the optimal generator

The model $\{P_\theta\}_{\theta \in \Theta}$ is identifiable if $G_\theta(Z) \mapsto P_\theta$ is a bijection. Identifiability is thus violated if multiple choices of θ can give the same output distribution P_θ. In general, this could occur arbitrarily often in neural networks. For example: if ReLU activations $\max(x, 0)$ are used in intermediate network layers, infinitely many θ in one layer can output 0, violating identifiability of that layer and consequently of the entire network. Instead, the network could be restricted to bijective activation functions, such as LeakyReLU activations. However, even with bijective activation functions, ambiguity may arise in the model. Consider a layer with two neurons, θ_1 and θ_2, connected to a single input node with value x and no bias or activation function. The output of the layer is given by $y = (\theta_1 + \theta_2)x$. Clearly, for any input-output pair (x, y) there are infinitely many solutions to $\theta_1 + \theta_2 = \frac{y}{x}$, resulting again in non-identifiability of the model. If x is a random input variable, this means that the same distribution can be constructed with infinitely many choices of parameters. Therefore, although the map p_{θ^*} may be unique, θ^* itself is not. There could be many subsets of Θ that arrive at the same density p_{θ^*}. Still, this is not an obstacle in this thesis, since the existence of θ^* is enough for the network to satisfy our needs of approximating the distribution p^*.

3.2.7. Non-ideal case: finite parameter set for the discriminator

So far we have considered the case where the discriminator can attain any Borel function $E \to [0,1]$. In practice, however, the discriminator can only attain mappings within the capacity implied by its parameter set $\alpha \in \mathbb{R}^q$. To stress this, it is written as D_α and the loss function will be written as $L(\theta, \alpha)$. But since in general for the parameterised form we have $D_\alpha \notin \mathscr{D}_\infty$, the minimax solution may be different than in the ideal case if $D \in \mathscr{D}_\infty$. Goodfellow et al. [63] comment on this, mentioning that if G and D have enough capacity, they should at least provide a reasonable estimate of p_{θ^*}, which is found with stochastic gradient descent. Biau et al. [69] analyse this problem in more detail, but have to make several assumptions about the structure of both G and D. The main results will be stated, but the proof will be omitted, since it is rather technical and not informative to the rest of this chapter.

The key consequence of the parametric nature of D_α is that the corresponding optimal generator changes. Suppose we call the parameter set corresponding to this new optimal generator $\bar{\theta}$, such that:

$$\sup_{\alpha \in \Lambda} L(\bar{\theta}, \alpha) \leq \sup_{\alpha \in \Lambda} L(\theta, \alpha) \quad \forall \theta \in \Theta. \tag{3.14}$$

Then it is important that $p_{\bar{\theta}}$ approaches p_θ for the new minimax choices of G_θ and D_α. The following theorem shows that improving the discriminator also improves the generator output.

Theorem 3.5 (**Improving D_α improves G_θ**). *Suppose that the following regularity assumptions hold:*

(H_0)*: There exists a positive constant* $\bar{t} \in (0, 1/2]$ *such that:*

$$\min(D^*_\theta(x), 1 - D^*_\theta(x)) \geq \bar{t} \quad \forall (x, \theta) \in E \times \Theta,$$

i.e. 'D^*_θ is some distance away from both 0 *and* 1*'.*

(H_ε)*: There exists an* $\varepsilon \in (0, t)$ *and a* $\bar{\theta}$*-admissible discriminator* D_α*, with* $\alpha \in \Lambda$ *such that:*

$$\|D_\alpha - D^*_{\bar{\theta}}\|_\infty \leq \varepsilon,$$

i.e. 'there is a discriminator that approaches $D^*_{\bar{\theta}}$ up to a constant ε'.

Then $\exists\ c > 0$*, a positive constant, which depends only on* t*, such that:*

$$0 \leq JS(p_{\bar{\theta}} \| p^*) - JS(p_{\theta^*} \| p^*) \leq c\varepsilon^2. \tag{3.15}$$

Proof. See Biau et al., section 5 [69]. □

This implies that if the parameterised discriminator approaches D^*_θ up to ε, the difference between the JS-divergence in the parametric case and in the ideal case is bounded by a constant of $O(\varepsilon^2)$. This result is important since it shows that letting D_α get closer to $D_{\bar{\theta}}$ actually results in getting $p_{\bar{\theta}}$ closer to p_{θ^*}. This bound scales with ε^2. We should thus expect that as the class $\{D_\alpha\}_{\alpha \in \Lambda}$ becomes richer (i.e. more layers, more neurons per layer), $p_{\bar{\theta}}$ indeed approaches p_{θ^*} more closely. The assumptions seem to be sufficiently mild for this idea to work in practice. H_0 can be seen as a 'softer' version of the admissible discriminator, excluding trivial cases 0 or 1, meaning that the discriminator always at least puts some mass on real or fake data. The second requirement is also reasonable, as we know that neural networks can serve as function approximators that can in theory be arbitrarily accurate as the parameter set increases.

3.2.8. Including the effect of finite datasets

Finally, we should take into account that there is only finite training data available to find the optimal generator and discriminator. In practice, the expectations in the loss function in equation 3.5a will be replaced by the empirical estimates in equation 3.5b. This yet again affects the minimax solution of the generator and discriminator, simply because the integrals in equation 3.7 have been replaced by their sum approximations. All parameters now depend on the dataset used. To stress this, a 'hat' is used for $\hat{L}(\theta, \alpha)$. Let us assume for simplicity that $N_X = N_Z = N$, i.e. the amount of target and 'fake' samples available[3]:

[3]This is without loss of generality, since in the convergence theorem that follows the minimum of both sample sizes could be taken towards infinity.

$$\hat{L}(\theta, \alpha) = \frac{1}{N} \sum_{i=1}^{N} \log(D(X_i)) + \frac{1}{N} \sum_{i=1}^{N} (1 - D \circ G(Z_i)) \,. \tag{3.16}$$

It is easy to see that equation 3.16 is unbiased for equation 3.5a. This time we write $\hat{\theta}$ for the generator parameter set that achieves the infimum of the minimax problem given any discriminator, i.e.:

$$\sup_{\alpha \in \Lambda} \hat{L}(\hat{\theta}, \alpha) \leq \sup_{\alpha \in \Lambda} \hat{L}(\theta, \alpha) \quad \forall \theta \in \Theta. \tag{3.17}$$

Intuitively, we expect that if $N \to \infty$, the sums in equation 3.16 approach the expectations in equation 3.5a and consequently, we recover all results from the previous paragraph.
Interestingly, Biau et al. extend their analysis to study this problem in more detail as well. They provide a convergence theorem for the optimal generator in [69, section 4]. The proof involves a list of assumptions and technical machinery beyond the scope of this thesis. The result is stated here:

Theorem 3.6 (**Convergence theorem for GANs**). *Consider the following regularity assumptions:*

(H_0)*: the same assumption from theorem 3.5.*

(H'_ε)*:* $\exists \varepsilon \in (0, \bar{t})$ *for some positive constant* $\bar{t}$*, such that* $\forall \theta \in \Theta$, $\exists D \in \mathscr{D}$*, a* θ*-admissible discriminator, such that* $\|D - D^*_\theta\|_\infty \leq \varepsilon$*.*

(H^1_{reg}) : $\exists b \in (0, 1/2)$*, such that* $\forall \alpha \in \Lambda : \; b \leq D_\alpha \leq 1 - b$*. Additionally, the map* $(x, \alpha) \mapsto D_\alpha(x)$ *is part of* C^1 *with uniformly bounded differential.*

(H^2_{reg}) : *For any* $Z \in \mathbb{R}^d$*, the map* $\theta \mapsto G_\theta(Z)$ *is part of* C^1*, with uniformly bounded differential.*

(H^3_{reg}) : *For any* $x \in E$*, the map* $\theta \mapsto p_\theta(x)$ *is part of* C^1*, with uniformly bounded differential.*

If assumptions H_0*,* H'_ε *and* $H^{1,2,3}_{\text{reg}}$ *are satisfied, the following statement holds:*

$$\mathbb{E}_{\hat{\theta}} \left[JS(p_{\hat{\theta}} \| p^*) \right] - JS(p_{\theta^*} \| p^*) = O\left(\epsilon^2 + \frac{1}{\sqrt{N}} \right). \tag{3.18}$$

Proof. See Bieau et al., section 4 [69]. □

The dependence on $\hat{\theta}$ is written in $\mathbb{E}_{\hat{\theta}}$ of equation 3.18 to stress the fact that $\hat{\theta}$ is a random variable, depending on the dataset $\{X_i\}_{i=1}^N$ and $\{Z_i\}_{i=1}^N$. The result is interesting, since it shows that we approach the result in equation 3.15 as $N \to \infty$. It is a 'law of large numbers for GANs'. It implies that $p_{\hat{\theta}}$ can get nearly as close to p^* as p_{θ^*} can, if we increase the sample size of our data.

The regularity assumptions mainly involve the continuity and uniform boundedness of the maps $\theta \mapsto G_\theta$, $\theta \mapsto p_\theta$, $(x, \alpha) \mapsto D_\alpha(x)$ and their first derivatives. I.e., if we change the parameters slightly, the resulting change in the neural network and its first derivative are bounded and continuous. How easy it is to satisfy these requirements depends on the choice of activation functions. For example, LeakyReLU activations are not differentiable at 0, but they are piecewise differentiable and continuous outside 0, while sigmoids are smooth on all of $\mathbb{R}$. In the neural network implementation in standard neural network libraries like `PyTorch`, non-differentiability of ReLU type activations is solved with a heuristic at 0, which overcomes the differentiability problem during backpropagation steps. Therefore, in practice the non-differentiability at 0 may not be a problem, not even for this type of activation. In the remainder of this thesis, it will be assumed that we can approach the theoretical p_{θ^*} with stochastic gradient descent by increasing the capacity of the networks and by increasing the sample size of the training set, in the spirit of theorem 3.6. The theoretical work by Biau et al. has provided handles that allow us to better understand what drives the convergence behaviour of GANs and where the ideal case differs from practice.

3.2.9. Concluding remarks on the theory

This section has provided a theoretical framework to understand why the generator and discriminator should approach the target distribution p^*. Even though the optimal generator need not be unique, this is not restrictive to our purposes, since the generator output is of interest, which does have a unique optimum. If the optimal discriminator is replaced by a parameterised approximation, theorem 3.5 shows that as we improve this approximation, we still approach the theoretical optimal generator output p_{θ^*}. With some assumptions about the smoothness of the networks in the space of parameters, the minimax choices of G and D can be approached in the case of both finite network capacity and finite sample size. These results show why we should expect the GAN to converge in distribution to a distribution that resembles the target data and concludes the fundamental theory behind GANs. The optimal discriminator can actually be approximated empirically, by using a kernel method to obtain an approximation of p_θ. This can be used to verify if the discriminator is indeed approaching the optimal choice D^* along the support of p^* and p_θ.

3.3. Training GANs in Practice

With the theoretical motivation behind GANs in mind, the question is now how we can use the loss functions to jointly approximate the minimax solutions of G_θ and D_α. Since neural networks are used, both G_θ and D_α should have their own loss function. The minimax problem can be cast into loss functions L_G for the generator and L_D for the discriminator as follows, as shown in the original paper by Goodfellow et al. [63]:

$$L_D = -\mathbb{E}_{X\sim P^*}\left[\log(D_\alpha(X))\right] - \mathbb{E}_{Z\sim P_Z}\left[\log\left(1 - D_\alpha \circ G_\theta(Z)\right)\right], \tag{3.19a}$$

$$L_G = \mathbb{E}_{Z\sim P_Z}\left[\log\left(1 - D_\alpha \circ G_\theta(Z)\right)\right]. \tag{3.19b}$$

Equation 3.19a is simply a restatement of the original loss function with a minus sign in front of it. This is arbitrary and depends on the choice of gradient ascent or descent. As in practice, neural networks libraries use gradient descent, the negative version is chosen. To see this, recall how the discriminator was to maximise the value function in equation 3.4, while the generator was to minimise it. Equation 3.19b is the same expression without the minus sign and where samples only come from the prior P_Z, so the first term in the value function becomes 0.

A key practical consideration is that the networks should not be trained in succession, but alternatingly. The minimax problem does not suggest this by itself, as we could theoretically first optimise the discriminator and then concern ourselves with the generator. However, as is clear from the loss function formulation, both networks need information about each other in order to be updated. Suppose the generator was initialised and then held fixed, upon which we optimise the discriminator to convergence. Recall that the optimal discriminator is defined only relative to the generator. Thus, if the generator is updated, the discriminator becomes 'outdated' and should be updated again. Performing this iterated operation in a single step is computationally infeasible, as mentioned in [74, p.3], which is why the networks are trained in alternating steps.

3.3.1. Perfect discriminators

Arjovsky et al. have provided a technical paper on the practice of training GANs, supported by theoretical results [75]. They explain how in general, the generator output lies on a low-dimensional manifold in the target space, allowing the discriminator to distinguish regions of the generated data perfectly, always labelling the real data 1 and fake data 0. In our much simpler 1D case, this would mean that there are sets on the real line where the generator puts mass, but the target data has no mass, or vice-versa, i.e.: the supports are disjoint. This would allow D to draw a decision boundary and rule out a set of generator outputs as real data. Arjovsky et al. explain why this is problematic, since it lets the discriminator gradient tend to zero, since a constant output has zero gradient. In the terminology of the GAN theory section, this actually resembles a non-admissible discriminator, which was conveniently ruled out. In practice, it does occur, as the argument shows. If the discriminator gradient is zero, it disallows the generator from improving, which freezes the convergence process. If the discriminator was optimised first while the generator does not yet resemble the distribution, this low-manifold problem is more likely to occur. Training alternatingly allows both networks to improve with a lower chance of giving rise to trivial solutions by the discriminator. However, outside of our simple 1D case, the man-

ifold structure in multiple dimensions can be highly complex. This means that perfect discriminators may arise at various stages during the training process. This is a current frontier of GAN research, where literature focuses on alternate divergence metrics to overcome the problem. This point will be explored further in a later section.

3.3.2. Vanishing generator gradient problem

Arjovsky et al. [75] show that the typical loss function of the generator provided in equation 3.19b will have gradients that tend towards zero as the discriminator approaches D^*_θ. This is an undesirable property, since it could 'freeze' the training process before the generator has converged. Therefore, Arjovsky et al. suggest an alternative loss function for the generator that was already in use by practitioners at their time of writing:

$$\tilde{L}_G = -\mathbb{E}_{Z\sim P_Z}\big[\log\left(D_\alpha \circ G_\theta(Z)\right)\big]. \tag{3.20}$$

This loss gives rise to an equivalent optimisation problem, but does not suffer from the vanishing gradient problem, as shown in [75]. However, this technique has its own problems, as Arjovsky et al. continue to explain. They show on page 8 that the generator gradient becomes unstable as it can be modelled by a Cauchy distribution with infinite mean and variance. This is a notorious problem in GAN literature, as also featured in an extensive review paper on GANs by Hong et al. [66]. This is an active line of GAN research. Still, preventing the gradients of the generator to vanish is an attractive prospect, which is why the modified loss is used in this work as well. The training process will be stabilised 'manually' with a learning rate schedule. Every n_k iterations, the generator learning rate will be additionally decreased by a factor c_{LR}, e.g. around 1.1 to 1.5. See section 2.2.3 from the Preliminaries for more background on the role of the learning rate.

3.3.3. Mode collapse

The final among key challenges described in GAN literature is that of 'mode collapse'. It is described by Hong et al. [66, p.18] as "many modes in the real data distribution are not represented in the generated samples". This is easy to imagine visually if the target distribution is some Gaussian mixture, where the model only centers at one of the Gaussians. Mode collapse occurs often in practice in high-dimensional problems [66, 74]. When classifying handwritten digits in the MNIST dataset, for example, mode collapse could make the GAN output only 1's instead of any number from 1-10, cf. [74, p.6]. This is why this problem is central in GAN literature. The next section briefly discusses the main adaptations that are suggested in GAN literature and compares them to the needs of the problem in this thesis.

3.4. Variations of the 'Vanilla GAN'

Given the popularity of GANs, the body of literature covering them is substantially large. There are many variants of GANs, that mainly focus on stabilising the training process and preventing the mode collapse problem introduced in the previous section. Most of the work is focused on modifying the loss functions for the generator and discriminator. The GAN introduced in the theoretical overview based on the minimax value function will be referred to as the 'vanilla GAN'. Here, the most notable variations are discussed briefly.

3.4.1. Ideas for stabilising the training process

In [74], the authors introduce what they call 'Unrolled GAN', which introduces an additional loss function based on future weight updates to the discriminator. This way, the gradient update to the generator contains information about how the discriminator will respond to succeeding updates of the generator. Including this information penalises trivial outputs on one mode, which, as the authors show in their results, overcomes the mode collapse problem.

Another approach is to replace the loss function based on the JS-divergence by a different distributional metric. In [76], the Wasserstein GAN is introduced, in which the loss function of both networks is based on the Wasserstein distance, defined for two distributions P and Q as:

$$w_p(P,Q) = \left(\inf_{\gamma\in\Gamma(P,Q)} \mathbb{E}_{(X,Y)\sim\gamma}\|X-Y\|_2^p\right)^{\frac{1}{p}}, \tag{3.21}$$

where $\Gamma(P, Q)$ is the set of joint distributions of P and Q. As already briefly discussed in the previous section, the gradients produced by the vanilla GAN generator loss can be unstable. Arjovsky et al. [75, 76] show how using the Wasserstein distance leads to stable and interpretable gradients, contrasting with the KL- and JS-divergences, which tend to 'max out' quickly if the distributions are dissimilar, resulting in zero gradient. They also explain how the w_p metric is 'weaker' than the JS-divergence, which allows the GAN to better capture the distribution as a whole rather than collapse onto one mode. The details of these metrics will be discussed in chapter 5, where it is also explained how the Wasserstein distance will be used in this thesis for analysing the output quality of the GAN, independent of whether it is used in the GAN loss. Thus, the Wasserstein GAN would in theory both solve the stability and mode collapse problems.

In [77], the authors link the problem of mode collapse to sharp discriminator gradients around data from the target distribution P^*. The 'DRAGAN' method is introduced, which adds a regularisation term to the generator loss with the discriminator gradient:

$$\rho \, \mathbb{E}_{(x,\tilde{\delta})} \left[(\|\nabla_x D_\alpha(x + \tilde{\delta})\|_2 - k)^2 \right], \tag{3.22}$$

where $x \sim P^*, \tilde{\delta} \sim N(0, C)$ and ρ is the regularisation parameter, $\tilde{\delta}$ is an $N(0, C)$ noise term with diagonal matrix C, intended to escape local optima and k is an additional constant which the authors set to 1, forcing the gradient to remain near 1. The idea of a gradient penalty has shown promising application in the Wasserstein GAN, where Gulrajani et al. improved the Wasserstein GAN in [78] by adding a gradient penalty to the original Wasserstein loss:

$$\rho \, \mathbb{E}_{\hat{x}} \left[(\|\nabla_x D_\alpha(\hat{x})\|_2 - 1)^2 \right], \tag{3.23}$$

where $\hat{x} \sim P_{\hat{x}}$ is a random convex combination of a variate from P_θ and P^*. The term forces the gradient to stay near 1, which is required for the Wasserstein GAN to converge. The authors show that this technique stabilises the training process. The details of the Wasserstein GAN are omitted as they are beyond the scope of this section. At the end of this section, it will be shown why the vanilla GAN will be sufficient for the purposes in this thesis.

3.4.2. Conditional GAN

At the end of the foundational paper by Goodfellow et al. [63], the authors suggest expanding the GAN model to conditional distributions, i.e. including a class label Y_i for each datapoint X_i to learn $P^*(X_i \mid Y_i)$. This idea has been elaborated and implemented by Mirza et al. [79], dubbed 'conditional GAN' (CGAN). The authors modify the vanilla GAN value function into the form given in equation 3.24.

$$\inf_{\theta \in \Theta} \sup_{\alpha \in \Lambda} \Big[\sum_{i=1}^{N} \log D_\alpha(X_i|Y_i) + \sum_{i=1}^{N} \log \left(1 - D_\alpha \circ G_\theta(Z_i|Y_i)\right) \Big], \tag{3.24}$$

where all that has changed is the conditioning on data labels Y_i. In practice, one would concatenate the labels to the inputs to both G and D, so it is argued for example in [23] that technically the joint distribution $P(X, Y)$ is used. However, through Bayes' rule, we see that the joint density is proportional to the conditional density, so the minimax solution should not change. We could repeat all the results from the theoretical overview for the CGAN as well, obtaining the same results. The conditional GAN will allow us to generate samples conditional on Δt and S_t, or in general the SDE parameters as well.

3.4.3. Other GANs

GAN literature is rich in more ideas and variants, but a detailed description of all setups is beyond the scope of this thesis. Different learning strategies may be proposed, such as evolutionary learning instead of gradient descent [80], GANs may be combined with variational auto-encoders (VAEs) [81], combining CGAN with a Laplacian pyramid [82], and many more. The discriminator may even be replaced entirely by a distributional metric, but typical metrics such as KL-divergence and JS-divergence may be difficult to compute explicitly. One suitable candidate is the maximum mean discrepancy (MMD) metric [83], which is a distance metric for distributions that can be computed efficiently, even in high dimensions.

3.4.4. GAN setup used in this thesis

For each GAN variation, there are arguments to be made why they should improve performance on a typical problem setup. Researchers from Google Brain have performed an extensive comparison study on seven prevalent GAN architectures [84]. They found that as computational budget increases, performance of all methods was similar, while on a lower computational budget, a supposedly superior algorithm may be outperformed by a 'lesser' GAN variant on the same problem. The researchers pointed out how GANs had not been compared properly in previous work and prior researchers had sometimes reported the minimum score achieved by their algorithm, which gives a distorted image of the actual distribution of performance coming from a model. The trend described by Google Brain researchers in [84] can be seen in existing literature as well. Papers comparing multiple architectures sometimes give mixed results, improvements to the vanilla GAN are marginal or the vanilla GAN defeats the majority of proposed alternatives entirely. See for example, [77, fig.4-5], [78, fig.3],[80, fig. 4], [8, fig.8]. The key lesson from the literature seems to be that years since the inception of GANs, there is no 'one-size-fits-all' best choice among all the alternatives proposed, for every computational budget and application, despite the impressive theoretical results on e.g. Wasserstein GANs.

Furthermore, the conditional GAN architecture itself may allow the GAN to distinguish between the modes in the target distribution, mitigating the effect of mode collapse. If dimensions higher than one are chosen in related subsequent works, it is advised to experiment with gradient penalties to stabilise the training process, in the spirit of the analysis given in [77] and [75].

Activation functions were chosen to be of the ReLU type, to prevent the gradient from tending to zero on values far away from the origin. This is relevant for the processes under consideration, which may exhibit heavy tails that persist after normalisation steps. In order to prevent regions with zero gradient that might hinder the training process, LeakyReLU activations are used, as they provide small but non-zero gradients for negative values. The algorithm will be implemented in `PyTorch` [55].

3.5. Algorithmic Formulation

We are now ready to consider the training process in algorithmic form. It is written here as a concrete overview of what is implemented in the results. The inputs to the generator and discriminator will vary as the GAN is modified to the conditional GAN. The vanilla GAN algorithm is similar to the one presented in [63].

Algorithm 1: GAN training process with number of epochs n_E, batch size n_B, number of iterations $n_E \cdot n_B$, number of discriminator steps per generator step n_D, discriminator learning rate λ_D and generator learning rate λ_G. Random mini-batches are drawn to obtain a set of real samples X.

for $m \in \{1, \dots, n_E\}$ **do**
  Sample $\{X_1, X_2, \dots, X_{n_B}\} \overset{iid}{\sim} P^*$
  for $n \in \{1, \dots, n_D\}$ **do**
    Sample $\{Z_1, Z_2, \dots, Z_{n_B}\} \overset{iid}{\sim} P_Z$
    $L_D = \frac{1}{n_B} \sum_{i=1}^{n_B} \Big[\log D_\alpha(X_i) + \log\left(1 - D_\alpha \circ G_\theta(Z_i)\right)\Big]$
    Update the discriminator weights:
    $\alpha \leftarrow \alpha - \lambda_D \nabla_\alpha L_D$
  end
  Sample $\{Z_1, Z_2, \dots, Z_{n_B}\} \overset{iid}{\sim} P_Z$
  $L_G = -\frac{1}{n_B} \sum_{i=1}^{n_B} \Big[\log\left(D_\alpha \circ G_\theta(Z_i)\right)\Big]$
  Update the generator weights:
  $\theta \leftarrow \theta - \lambda_G \nabla_\theta L_G$
end

Notice how the discriminator may be trained more than once per generator update (n_D). This may be advantageous, since the more the discriminator is updated for a fixed generator, the better it approximates the optimal discriminator given the generator. The more the discriminator step is

repeated, however, the more computationally costly the method becomes. In this work, this parameter was set to $n_D = 2$, which led to faster rates of convergence than $n_D = 1$. Due to computational considerations, it was not increased further. An overview of the architectures used in this work is provided in the appendix.

Chapter Conclusion

This chapter has presented the theoretical foundations of GANs. The core result is that the GAN loss function can be written in terms of a distributional metric, the Jensen-Shannon divergence. The GAN can be shown to converge even if we take into account the parameterised nature of the generator and discriminator and the fact that the training set is finite. Many GAN variants exist, but empirical evidence shows no clear best alternative among the variants in the general case. The vanilla GAN and conditional GAN will satisfy the needs in this work as a base architecture. In chapter 4, a modified version of the conditional GAN is proposed, which is still based on the base principles outlined in this chapter. If future work operates in higher dimensions, mode collapse should be a point of attention, in which case an architecture with gradient penalty may be considered.

4

Methodology

This chapter presents our approach to approximate a strong solution to the SDE on a time discretisation. First, the problem setting is discussed in detail. It is shown that vanilla GANs are unable to provide a strong solution to the SDE in general. In this work, a modified GAN called 'constrained GAN' is proposed that overcomes this problem. If a GAN is adapted to the same natural filtration and maps random increments in the same way as the strong solution, it can provide a strong approximation, which is shown in this chapter. Various techniques are discussed to enforce this map on the GAN on general SDE problems, all in the spirit of the constrained GAN architecture. Finally, it is shown how paths are constructed using the GAN and how they are assessed. The next chapter will show how various non-parametric techniques can be used to study the distribution of the GAN output.

4.1. Setting

Before we discuss the implementation of the GAN to approximate the SDE, we will first give a precise description of the problem setting. General SDEs are defined as continuous-time processes, but this is intractable when constructing a dataset of training examples for the GAN. Additionally, in the setting of this work, we are interested in a single time step or several time steps in the future, as opposed to the entire path. Thus, we are concerned with some discretisation of the time domain of interest, i.e. the process at the times $0 = t_0 < t_1 < \ldots < t_k < \ldots < t_N = T$. Suppose we are given a process $\{S_t\}$ with $S_0 = x \in \mathbb{R}$ and a Brownian motion $\{W_t\}$, which together form the strong solution of equation 4.1 on $[0, T]$, adapted to the filtration $\mathcal{F}_t = \sigma(\{W_t\})$.

$$S_{t_k} = S_{t_{k-1}} + \int_{t_{k-1}}^{t_k} A(s, S_s)ds + \int_{t_{k-1}}^{t_k} B(s, S_s)dW_s \quad P-\text{a.s.} \tag{4.1}$$

On our discretisation, the strong solution of the SDE takes values $(\{S_{t_k}\}, \{W_{t_k}\})$, as illustrated in figure 4.1, where the blue line denotes a realisation of the strong solution. The process $\{S_{t_k}\}_{k=0}^N$ is not continuous, i.e. not defined anywhere else but on the discretisation points. The key consequence is that the process $(\{S_{t_k}\}, \{W_{t_k}\})$ for $k = 0, \ldots, N$ is not a strong solution, as it is not continuous. We can no longer distinguish between Brownian motions that take values W_{t_k} on our discretisation, but may take different values on all other points on $[0, T]$. Therefore, in this work, the notions of weak and strong solutions are reintroduced on a discretisation, which we will need to formalise our understanding of the approximation achieved by GANs, trained on a dataset of $\{S_{t_k}\}$.
On a discretisation of the time interval, we do not observe $\{W_t\}$, but $\{W_{t_k}\}$ for all k. Note that $\{W_{t_k}\}$ is not a Brownian motion, since it is not continuous, but a realisation of $\{W_t\}$ at each time t_k. The following will provide the terminology for accurately describing our setting with regard to weak and strong solutions.

Definition 4.1 (**Natural filtration (discrete case)**)**.** Given a discretisation $\{t_k\}_{k=0}^N$ of the interval $[0, T]$ with $T > 0$, the discrete analog of the natural filtration is given by the σ-algebra generated by the realisations of the Brownian motion $\{W_t\}$ at each t_k on the discretisation, i.e.:

$$\mathcal{G}_k := \sigma(\{W_{t_i} : \ 0 \leq i \leq k\}) \ \forall k \in \{0, \ldots, N\}. \tag{4.2}$$

Remark 4.2. For all filtrations of a strong solution $\mathcal{F}_{t_k}$, we have $\mathcal{G}_k \subseteq \mathcal{F}_{t_k} \subseteq \mathcal{F}$. As $t_k - t_{k-1} \to 0$, $\mathcal{G}_k$ will progressively grow to $\mathcal{F}_{t_k}$.

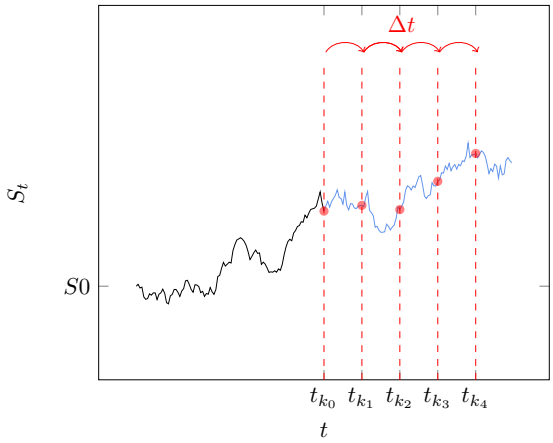


Figure 4.1: A single realisation of a path, corresponding to the strong solution $(\{S_t\}, \{W_t\})$, shown as the solid line, available up to time $S_{t_{k_0}}$. We are interested in extending the strong solution beyond $S_{t_{k_0}}$, i.e. the blue line. The discrete strong solution is a subsampled version of the strong solution, given by $(\{S_{t_k}\}, \{W_{t_k}\})$, shown as the red dots. In the figure, $\Delta t := t_k - t_{k-1}$ is held fixed.

Definition 4.3 (**Discrete strong solution**)**.** Let $(\{S_t\}, \{W_t\})$ with $S_0 = x \in \mathbb{R}$ be a strong solution to the SDE on $(\Omega, \mathcal{F}, P)$. Given a discretisation $\{t_k\}_{k=0}^N$ of the interval $[0, T]$, a *discrete strong solution* on the partition is defined as the pair $\left(\{S_{t_k}\}, \{W_{t_k} - W_{t_{k-1}}\}\right)$ that satisfies P-a. s. $\forall k \in \{1, \ldots, N\}$:

$$S_{t_k} = (S_t)|_{t=k}, \tag{4.3}$$

$$W_{t_k} - W_{t_{k-1}} = (W_t)|_{t=k} - (W_t)|_{t=k-1}. \tag{4.4}$$

Clearly, $\{S_{t_k}\}$ and $\{W_{t_k} - W_{t_{k-1}}\}$ are adapted to $\mathcal{G}_k$ $\forall k \in \{1, \ldots, N\}$. A sequence of exactly simulated points $\{S_{t_k}\}_{k=0}^N$ is not a strong solution to the SDE, since it is not continuous. However, it is a discrete strong solution, adapted to $\mathcal{G}_k$. If we would say that we 'approximate' the strong solution with a discrete strong solution, this statement would be unrelated to the accuracy of the approximation $\{S_{t_k}\}_{k=0}^N$, since by definition the strong solution is equal at every point t_k. The difference arises purely from the transition of a continuous random variable to a discrete one. We can analogously define a weak solution.

Definition 4.4 (**Discrete weak solution**)**.** Let $(\Omega', \mathcal{F}', P')$ be a probability space. Let $(\{S'_t\}, \{W'_t\})$ be a weak solution to the SDE, where both processes are adapted to $\mathcal{F}'_t = \sigma(\{W'_t\})$ and $S'_0 = x \in \mathbb{R}$. Given a discretisation $\{t_k\}_{k=0}^N$ of the interval $[0, T]$, a *discrete weak solution* on the partition is defined as the pair $\left(\{S'_{t_k}\}_{k=1}^N, \{W'_{t_k} - W'_{t_{k-1}}\}_{k=1}^N\right)$ that satisfies P'-a.s. $\forall t_k \in \{t_k\}_{k=1}^N$:

$$S'_{t_k} = (S'_t)|_{t=k}, \tag{4.5}$$

$$W'_{t_k} - W'_{t_{k-1}} = (W'_t)|_{t=k} - (W'_t)|_{t=k-1} \tag{4.6}$$

In analogy with the continuous version by Klenke, definition 26.12 [24], if there exists another discrete weak solution on $(\tilde{\Omega}, \tilde{\mathcal{F}}, \tilde{P})$, denoted by $(\{\tilde{S}_{t_k}\}, \{\tilde{W}_{t_k}\})$ that satisfies $\tilde{P} \circ \tilde{S}_{t_k}^{-1} = P \circ S_{t_k}^{-1}$, S_t, i.e. equality in distribution, it is called *weakly unique*.

Remark 4.5. A discrete weak solution need not be adapted to the filtration of the strong solution $\mathcal{G}_k$, analogous to the continuous case, where a weak solution need not be adapted to $\mathcal{F}_t$. Instead, a discrete weak solution is adapted to $\mathcal{G}'_k := \sigma(\{W'_{t_k}\}_{k=0}^N)$. It does not even need to be defined on the same probability space as the strong solution.

Let us test the introduced terminology on a simple example.

Example 4.6 (**Geometric Brownian motion**)**.** Consider the example of GBM, for which the connection between $\{S_t\}$ and $\{W_t\}$ is known explicitly. Let $\{t_k\}_{k=0}^N$ be a discretisation of $[0,T]$. Suppose that $S_0 = x \in \mathbb{R}$, then the strong solution between any two times $r \leq t$ on $[0,T]$ is given by:

$$S_t \mid S_r = S_r e^{(\mu - \frac{1}{2}\sigma^2)(t-r) + \sigma(W_t - W_r)}, \tag{4.7}$$

while a discrete strong solution is given $\forall k \in \{1, \dots, N\}$ by:

$$S_{t_k} \mid S_{t_{k-1}} = S_{t_{k-1}} e^{(\mu - \frac{1}{2}\sigma^2)(t_k - t_{k-1}) + \sigma(W_{t_k} - W_{t_{k-1}})}. \tag{4.8}$$

The process in equation 4.7 is adapted to the filtration $\mathcal{F}_t = \sigma(\{W_t\})$, while the process in equation 4.8 is adapted to $\mathcal{G}_k$. Suppose the pair $(\{S'_t\}, \{W'_t\})$ also satisfies equation 4.7, but this time $W'_t = -W_t$. This pair forms a weak solution to the SDE, since it satisfies the SDE and is adapted to the filtration $\sigma(\{-W_t\})$. Since $S'_t \overset{d}{=} S_t$, it is weakly unique. However, the process is not a strong solution, as it is not path-wise equal to the strong solution P-a. s. This holds analogously for the discrete variants.
If we were to construct a 'dataset' of the discrete strong solution $\{S_{t_k}\}_{k=0}^N$, we would sample a sequence of random increments $\{W_{t_k}\}_{k=0}^N$ by drawing N numbers $Z \sim N(0,1)$ and multiplying them with $\sqrt{t_k - t_{k-1}}$. Then the pair $(\{S_{t_k}\}, \{W_{t_k} - W_{t_{k-1}}\})$ forms a discrete strong solution to the SDE, where realisations of the Brownian motion increments $\{W_{t_k} - W_{t_{k-1}}\}$ have been sampled using the process $\{Z_k\}$ of i.i.d. standard normal random variables.

Beyond this chapter, we will assume that our discretisation is equally spaced with step size $\Delta t = \frac{T}{N}$, such that $t_k = k\Delta t \ \forall k \in \{0, 1, \dots, N\}$. We could write each 'next step' of the process S_t on the discretisation as $S_{t+\Delta t} \mid S_t$, in the same way as in [85], where paths are simulated for the Heston model. By the Markov property of SDEs [24], each next step $S_{t+\Delta t} \mid S_t$ is independent of $\mathcal{F}_t$. Therefore, on a discretisation with equal spacing and given an SDE and Brownian motion $\{W_t\}$, the entire path on any bounded interval $[0,T]$ is defined by the starting point S_0 and the next step $S_{t+\Delta t} \mid S_t$, as illustrated in figure 4.1. In the remainder of this chapter, we will keep writing $S_{t_k} \mid S_{t_{k-1}}$ to keep the results general for any discretistion.
In the results, paths will be shown as linearly interpolated line segments between the points, which is only for illustration purposes. In the example of figure 4.1, we will attempt to let a neural network approximate 'next red dot given the previous red dot', which is equal on the points $\{t_k\}_{k=1}^N$ to the strong solution given by the blue line. In this work, by exact simulation we will refer to a method that defines a discrete strong solution on all points of the discretisation (i.e. 'construct the red dots').

Remark 4.7 (**Comparison with discrete-time schemes**)**.** Discrete-time approximation schemes such as the Euler and Milstein schemes are also defined in the discrete setting. As we have seen in the preliminaries, such schemes provide approximations to the process $\{S_t\}$ on a discretisation by truncating the stochastic Taylor expansion. They approximate the discrete strong solution at each point on the discretisation, while the discrete strong solution is exactly equal to the strong solution on the discretisation points. In exact simulation, we do not require any condition on the mesh of the discretisation, while discrete-time schemes rely on $\Delta t \to 0$ in order to recover the exact strong solution. The method proposed in this work is to let a neural network 'learn' the map achieved by an exact scheme between any two time steps. It resembles more closely an exact scheme than a discrete-time scheme, while the Euler and Milstein schemes will act as a reference to assess the quality of the generated approximation to the discrete strong solution.

4.2. Discrete Approximations with GANs

In this section, we discuss in what sense a GAN can approximate a weak or strong discrete solution to the SDE. Let us assume in the rest of this chapter that the GAN introduced in chapter 3 approximates the distribution of the trained examples perfectly, i.e. $G_\theta(Z) \overset{d}{=} S_{t_k} | S_{t_{k-1}}$ for all t_{k-1}, t_k and $S_{t_{k-1}}$. This is useful for establishing a theoretical understanding of how a GAN approximates a discrete solution. First, we study the ability of the GAN to sample from $F_{S_{t_k}|S_{t_{k-1}}}$ for a single step and with fixed parameters. This forms the basis for a discrete approximation of the process $\{S_t\}$. Clearly, modelling a single transition $S_{t_k}|S_{t_{k-1}}$ for a fixed $S_{t_{k-1}}$ is not sufficient for approximating a discrete solution on the entire discretisation, either weak or strong, since it will not capture the dependence of the process $\{S_t\}$ on time or on $S_{t_{k-1}}$, i.e. the autocorrelation structure. The

approximation will be extended to $G_\theta(Z, t_k, t_k - t_{k-1}, S_{t_{k-1}}) \overset{d}{=} S_{t_k}|S_{t_{k-1}}$ for any t_{k-1}, t_k and $S_{t_{k-1}}$ on some range of interest with the conditional GAN.

4.2.1. Vanilla GAN

A single step can be modelled using the 'vanilla GAN', as introduced in chapter 3, where the GAN is trained on a dataset of samples with some stationary distribution. This will form a baseline result to establish how well the GAN can approximate a target in distribution. The GAN is trained on a dataset of exact variates corresponding to GBM or the CIR process. This is done by sampling from the analytical formulation of the conditional distribution $S_{t_k} \mid S_{t_{k-1}}$ for fixed parameters, as follows, repeating the equations from the preliminaries:

$$\text{GBM}: \quad S_{t_k} \mid S_{t_{k-1}}(\omega) = S_{t_{k-1}} e^{(\mu - \frac{1}{2}\sigma^2)(t_k - t_{k-1}) + \sigma(W_{t_k} - W_{t_{k-1}})(\omega)} \tag{4.9}$$

$$= S_{t_{k-1}} e^{(\mu - \frac{1}{2}\sigma^2)(t_k - t_{k-1}) + \sigma Z_k(\omega)\sqrt{t_k - t_{k-1}}} \tag{4.10}$$

$$\text{CIR}: \quad S_{t_k} \mid S_{t_{k-1}}(\omega) \sim c\, \chi^2(\xi, \delta)(\omega), \tag{4.11}$$

for all $\omega \in \mathcal{G}_k$ and $k \in \{1, \ldots, N\}$ and the i.i.d. sequence of $N(0,1)$ random variables $\{Z_k\}$. The default parameter sets of choice will be included in the appendix.
Note that in this setting, the GAN does not need to receive the starting value $S_{t_{k-1}}$, nor the remaining SDE parameters at its input, as they remain constant. The GAN 'learns' how to center and scale the distribution purely from the dataset $S_{t_k} \mid S_{t_{k-1}}$. In the next chapter, it is shown how two probability distributions can be compared using techniques from non-parametric statistics.

4.2.2. Conditional GAN

In order to generalise beyond a single step, the GAN can be trained on samples $S_{t_k} \mid S_{t_{k-1}}$ with varying t_{k-1}, t_k and $S_{t_{k-1}}$, by using the conditional GAN introduced by Mirza et al. [79]. Let us define $\mathcal{C}$ to be a set of conditional parameters. We can refer to the conditional parameters as 'labels' or 'classes', which is how they are introduced in [79], although in our case they may be chosen from a continuum of possible choices. The training set now consists of tuples $(S_{t_k} \mid S_{t_{k-1}}, C)$, where $C \in \mathcal{C}$ is a conditional parameter, e.g. $S_{t_{k-1}}$, $\Delta t := t_k - t_{k-1}$, or both. For example, suppose the GAN is to be trained on two time steps, e.g. $\Delta t \in \mathcal{C} := \{0.5, 1\}$ and further that $t_{k-1} = 0$ for simplicity. The training set then consists of tuples $\{(S_{0.5} \mid S_0, 0.5), (S_1 \mid S_0, 1)\}$ and can be constructed by randomly choosing between the classes n times uniformly, with n the desired training set size. Additional parameters could be included as well, such as (μ, σ) for GBM. In principle, the GAN could learn the distribution of an entire family of solutions to the SDE in this way. Training samples are obtained by drawing from the exact distributions given in equations 4.9 and 4.11.
The generator input does not only consist of $Z \sim N(0,1)$, but of the tuple (Z, C). The discriminator alternatingly receives $(S_{t_k} \mid S_{t_{k-1}}, C)$ and $(G_\theta(Z, C), C)$ during training. During inference, the generator is called with (Z, C_{test}) on a test condition of interest. This way, we can assess the GAN output on a cross-section of the hyperplane spanned by all combinations of conditions.
The set of condition labels $\mathcal{C}$ does not need to be discrete, but can be varied on a continuum as well. One could construct an interval $[a, b] \subseteq \mathbb{R}$ to create a test set of N unique samples of C, uniformly sampled from $[a, b]$.

Remark 4.8. To model each step $S_{t_k} \mid S_{t_{k-1}}$ for the GBM and CIR problems, the conditional GAN receives the conditional argument $C = (S_{t_k}, t_k - t_{k-1})$, but not t_k. This is because in both the cases of GBM and the CIR process, $S_{t_k} \mid S_{t_{k-1}}$ does not depend on t_k, but only on the previous value S_{t_k} and the time step $\Delta t := t_k - t_{k-1}$. This holds for all *Itō diffusions*, cf. [1], i.e. where SDE coefficients $A(t, S_t) = A(S_t)$ and $B(t, S_t) = B(S_t)$ do not depend on t. To extend to general SDEs, the conditional GAN should also receive the current time t_k as input.

A schematic overview of the conditional GAN generator is given in figure 4.2. The input conditions are concatenated to the input of the neural network. If the dimension of the conditional parameter set is d, the generator and discriminator receive a $(d+1)$-dimensional input. This problem is more challenging than the vanilla GAN case, as the discriminator must now accurately assign a confidence between 0 and 1 on a $(d+1)$-dimensional plane and consequently capture the dependence of the distribution of $S_{t_k} \mid S_{t_{k-1}}$ on these variables.

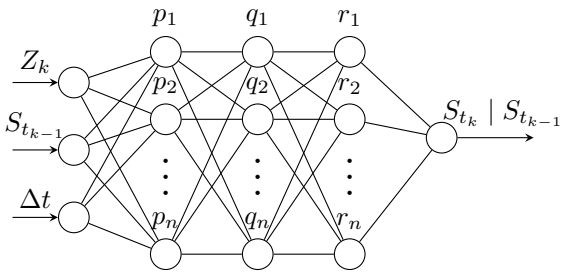


Figure 4.2: Schematic overview of the conditional GAN generator. The discriminator is defined analogously, but instead of Z_k it receives either $S_{t_k} \mid S_{t_{k-1}}$ or a generated solution $G(Z, S_{t_{k-1}}, t_k - t_{k-1})$.

4.2.3. Weak approximation

If the conditional GAN generator, trained on $(S_{t_k} \mid S_{t_{k-1}}, t_k, \Delta t := t_k - t_{k-1})$, approximates its target distribution $S_{t_k} \mid S_{t_{k-1}}$ perfectly, we have:

$$G_\theta(Z_k, S_{t_k}, t_k, \Delta t) \overset{d}{=} S_{t_k} \mid S_{t_{k-1}}. \tag{4.12}$$

If we sample the sequence $\{Z_k\}$, with $Z_k \overset{i.i.d.}{\sim} N(0,1)$, then we can construct a path by iterative application of equation 4.12 for $k \in \{1, \dots, N\}$. However, the key problem is that we do not know whether the GAN output will provide a path-wise approximation to the strong solution, as it is only guaranteed to be equal in distribution at each time t_k to the discrete strong solution. Let us call the generator output $\hat{S}_{t_k}$, then we have $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}} \overset{d}{=} S_{t_k} \mid S_{t_{k-1}}$. Since a strong solution exists, there is also a path-wise unique weak solution, which is weakly unique, cf. the result due to Yamada and Watanabe, given as theorem 26.18 in [24]. This means that there might be another S'_{t_k} for which the generator satisfies equation 4.12, but is adapted to a different filtration than S_{t_k}. The conditional GAN learns the conditional distribution $F_{S_{t+\Delta t}|S_t}$, but we have no information about the filtration or Brownian motion to which it is adapted. Thus, convergence in distribution to the strong solution limits us to the class of discrete weak solutions in general.

4.2.4. Strong approximation

From the analysis so far, we conclude that a standard GAN architecture is not guaranteed to provide a discrete strong solution to the SDE. However, if we could somehow guarantee that the GAN output is adapted to $\mathcal{G}_k$ and shares the same connection with the Brownian motion as the strong solution, we do recover a discrete strong solution, which is shown in this section. For this relation to hold, the GAN should be related to an event $\omega \in \mathcal{G}_k$ as $(\omega, t_k, \Delta t, \hat{S}_{t_{k-1}}) \mapsto \hat{S}_{t_k}(\omega)$, cf. the derivation of theorem 7.1.2 in [1] by Øksendal for Itō diffusions. For each $\omega \in \mathcal{G}_k$, the GAN output must equal the discrete strong solution P-a. s. to represent a strong solution.
As we will be computing moments of the output of the generator, the output must be integrable. To this end, we show that the generator output is in L^p, where we will follow the approach by Wiese et al. [5]. The result is shown in theorem 4.9.

Theorem 4.9 (**The generator output is in L^p**). *Given a conditional GAN generator, which takes as input $t_k \in [0,T]$, $\Delta t := t_k - t_{k-1} \in [0,T]$ and $S_{t_{k-1}} \in [a,b] \subseteq \mathbb{R}$, defined by:*

$$\begin{aligned} &G_\theta : \Omega \times [0,T] \times [0,T] \times [a,b] \to \mathbb{R}, \\ &(\omega, t_k, \Delta t, x) \mapsto G_\theta(Z(\omega), t_k, \Delta t, S_{t_{k-1}}), \end{aligned} \tag{4.13}$$

where $S_0 = x \in \mathbb{R}$. Then the generator output is in $L^p(\mathbb{R})$ $\forall k \in \{0, 1, \dots, N\}$.

Proof. Feed-forward neural networks with ReLU-type activations, as used in this work, are Lipshitz-continuous [86, 87]. From here, we will reproduce the proof by Wiese et al. [5]. Recall that for a Lipschitz-continous function $f : \mathbb{R}^m \to \mathbb{R}^n$ for $m, n \in \mathbb{N}$, there exists a $K > 0$ such that $\|f(x) - f(y)\| \leq K\|x - y\|$ for all $x, y \in \mathbb{R}^m$, where $\|\cdot\|$ is the L^1-norm. If $y = 0$, we have $\|f(x) - f(0)\| \leq K\|x\|$. Consequentially, using the triangle inequality, one can show that: $\|f(x)\| \leq K\|x\| + \|f(0)\|$.

Let us write $\boldsymbol{Z} := (Z, t_k, \Delta t, S_{t_{k-1}})$. Let us first assume that $S_{t_{k-1}} \in L^p(\mathbb{R})$. Since Z is a Gaussian random random variable on $(\Omega, \mathcal{F}, P)$, it is in L^p. Since t_k and Δt are constants on a bounded interval, $\boldsymbol{Z} \in L^p(\mathbb{R}^4)$. Now we apply the triangle inequality:

$$\begin{aligned}\mathbb{E}\left[\|G_\theta(\boldsymbol{Z})\|^p\right] &\leq \mathbb{E}\left[(K\|\boldsymbol{Z}\| + \|G_\theta(\mathbf{0})\|)^p\right] \\ &= \sum_{j=0}^{p} \binom{p}{j} L^j \mathbb{E}\left[\|\boldsymbol{Z}\|^j\right] \|G_\theta(\mathbf{0})\|^{p-j} < \infty,\end{aligned} \tag{4.14}$$

where the last step follows from Newton's binomial expansion. K is the Lipschitz constant of the conditional GAN generator and $\mathbf{0}$ is the vector $(0,0,0,0)$. Finally, we assumed that $S_{t_{k-1}} \in L^p(\mathbb{R})$. At time zero, we have, $S_0 = x \in \mathbb{R}$, which is in L^p. Since $\hat{S}_{t_k}$ and $\hat{S}_{t_{k-1}}$ are related recursively, by induction it follows that $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}} \in L^p(\mathbb{R})\ \forall k$. □

We now have the following: the distribution of the conditional GAN output $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}$ equals $F_{S_{t+\Delta t}|S_t}$, i.e. we assume that the GAN 'perfectly' approximates the target distribution. We also have that $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}$ is in L^p, which will allow us to compute moments of the generator output. If we assume that $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}$ is adapted to $\mathcal{G}_k$ and corresponds to the same Brownian motion increments as the strong solution for each $\omega \in \mathcal{G}_k$, then the generator provides a discrete strong solution, which is shown in theorem 4.12, for which we first consider lemma 4.10.

Lemma 4.10 (**Constructing adapted increments**). *Fix a probability space $(\Omega, \mathcal{F}, P)$. Let $(\{S_t\}, \{W_t\})$ and $(\{S_{t_k}\}, \{W_{t_k} - W_{t_{k-1}}\})$ for all $k \in \{1, \ldots, N\}$ be respectively a strong solution and discrete strong solution of an SDE adapted to the natural filtrations $\mathcal{F}_t = \sigma(\{W_t\})$ and $\mathcal{G}_k = \sigma(\{W_{t_k}\})$. Let $F^{-1}_{W_k}$ denote the inverse distribution of the Brownian motion increment $W_{t_k} - W_{t_{k-1}}$ and $F_{S_k|S_{k-1}}$ denote the CDF of $S_{t_k} \mid S_{t_{k-1}}$. Then $\forall \omega \in \mathcal{G}_k$, the Brownian motion increment between times t_k and t_{k-1} is uniquely given by:*

$$\left(W_{t_k} - W_{t_{k-1}}\right)(\omega) = F^{-1}_{W_k}\left(F_{S_k|S_{k-1}}\left((S_{t_k} \mid S_{t_{k-1}})(\omega)\right)\right). \tag{4.15}$$

Proof. Since the process $\{S_t\}$ contains continuous random variables for any SDE, the joint density of S_t, S_r at times $r \leq t$, denoted by $f_{(S_t,S_r)}$ and the marginal density f_{S_r} are non-zero P-a.s. Then so is the density $f_{S_t|S_r} = \frac{f_{(S_t,S_r)}}{f_{S_r}}$. Consequently, the CDF $F_{S_t|S_r}$ is a bijection, as it is strictly increasing and non-zero P-a.s. Since this holds for all $r, t \in [0,T]$, it also holds for t_k and t_{k-1}. Furthermore, since $W_{t_k} - W_{t_{k-1}} \sim N(0, t_k - t_{k-1})$, its CDF F_{W_k} is a bijection and its inverse distribution exists. Thus, for every realisation of $S_{t_k} \mid S_{t_{k-1}}$, there is a unique realisation of the $N(0, t_k - t_{k-1})$ random variable found by equation 4.15. The final requirement is to show that this $N(0, t_k - t_{k-1})$ random variable coincides with the Brownian motion increment between t_k and t_{k-1}. This follows from uniqueness of the strong solution in the continuous case, i.e. every pair $(S_0, \{W_t\})$ maps uniquely to the process $\{S_t\}$. As this holds for all times, it also holds on our discretisation. Thus, every starting point and increment $(S_{k-1}, W_{t_k} - W_{t_{k-1}})$ map uniquely to a realisation $S_{t_k} \mid S_{t_{k-1}}$. Therefore, the realisation of the random variable given by equation 4.15 must coincide with $W_{t_k} - W_{t_{k-1}}$. By adaptedness of $\{S_{t_k}\}$ to $\mathcal{G}_k$, equality holds for every event $\omega \in \mathcal{G}_k$. □

Remark 4.11. Lemma 4.10 tells us that, given the distribution of the transition $S_{t_k} \mid S_{t_{k-1}}$, we can sample a corresponding Brownian motion increment. Dividing by $\sqrt{t_k - t_{k-1}}$, we obtain a sequence of standard normal random variables, say $\{Z_k\}$, which is also adapted to $\mathcal{G}_k$. For every realisation of $W_{t_k} - W_{t_{k-1}}$, there is a unique realisation of Z_k. This will be used to create an Euler and Milstein approximation of the CIR process, so that we can compare exactly simulated paths using equation 4.11 with the discrete-time approximation path-wise.

Theorem 4.12 (**Discrete strong solution with conditional GAN**). *Let $(\{S_{t_k}\}, \{W_{t_k} - W_{t_{k-1}}\})$ be a discrete strong solution on $(\Omega, \mathcal{F}, P)$, adapted to the natural filtration $\mathcal{G}_k$. Let $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}} \overset{d}{=} S_{t_k} \mid S_{t_{k-1}}$ denote the output of a trained generator given $\hat{S}_{t_{k-1}}$. If the generator output is adapted to $\mathcal{G}_k$ and if $\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}$ and $S_{t_k} \mid S_{t_{k-1}}$ are related by $\left(W_{t_k} - W_{t_{k-1}}\right)(\omega) = F^{-1}_{W_k}\left(F_{S_{t_k}|S_{t_{k-1}}}\left(\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}\right)(\omega)\right)$, then the pair $(\{\hat{S}_{t_k}\}, \{W_{t_k} - W_{t_{k-1}}\})$ is a*

discrete strong solution to the SDE, i.e. $S_{t_k} \mid S_{t_{k-1}}(\omega) = \hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}(\omega)$ P-a. s. *for all* $\omega \in \mathcal{G}_k$. $F^{-1}_{W_k}$ *is the inverse distribution of* $W_{t_k} - W_{t_{k-1}}$.

Proof. By lemma 4.10, for every sample $\left(\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}\right)(\omega) \sim F_{S_{t_k}|S_{t_{k-1}}}$, there is a unique corresponding Brownian motion increment $\left(W_{t_k} - W_{t_{k-1}}\right)(\omega)$. Since we assumed that the generator output is adapted to $\mathcal{G}_k$, this Brownian motion increment is also adapted to $\mathcal{G}_k$. Now, by uniqueness of the strong solution, for each $\left(W_{t_k} - W_{t_{k-1}}\right)(\omega)$, there is a unique $\left(S_{t_k} \mid S_{t_{k-1}}\right)(\omega)$. As this holds for all $k \in \{1, \dots, N\}$ and $\omega \in \mathcal{G}_k$, by uniqueness of the strong solution, we must have $\left(S_{t_k} \mid S_{t_{k-1}}\right)(\omega) = \left(\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}\right)(\omega)$ P-a. s. □

Theorem 4.12 shows that a perfect generator finds a discrete strong solution if it is adapted to $\mathcal{G}_k$ and if it is related to the same Brownian motion increment as the discrete strong solution for every event $\omega \in \mathcal{G}_k$. In other words, the generator would learn the same map as the exact simulation scheme, as opposed to learning any map that corresponds in distribution. However, the adaptedness and connection with $W_{t_k} - W_{t_{k-1}}$ were included as an assumption. In order to enforce these properties, we need a different architecture than the vanilla conditional GAN. This architecture is presented in the following section.

4.3. Constrained GAN

'Constrained GAN' is a variant of the conditional GAN that forces path-wise equality to the strong solution via the discriminator. The key idea behind the constrained GAN is to train the GAN on pairs of $(\{S_{t_k}|S_{t_{k-1}}\}, \{W_{t_k} - W_{t_{k-1}}\})$ instead of only $\{S_{t_k}|S_{t_{k-1}}\}$. This would force the GAN to understand not only the distribution of $S_{t_k}|S_{t_{k-1}}$, but also how each realisation of $S_{t_k}|S_{t_{k-1}}$ relates to the Brownian motion increment. This is equivalent, up to a scaling factor $\sqrt{t_k - t_{k-1}}$, to training on pairs $(\{S_{t_k}|S_{t_{k-1}}\}, \{Z_k\})$, where $\{Z_k\}$ is a sequence of i.i.d. $N(0,1)$ random variables, adapted to the same filtration $\mathcal{G}_k$ as $\{W_{t_k}\}$. This way, the constrained GAN learns both how to sample from the correct distribution and how to correctly map the Brownian motion increment to the next value along the path. This notion is further explained in the following paragraph.

4.3.1. Constrained GAN as inverse map

Consider again figure 4.1, where the red dots form a discrete strong solution to the SDE. Due to the Markov property of SDEs, we could sample an entire path given a starting point $S_{t_{k_0}}$ and a means to sample from the conditional distribution $F_{S_k|S_{k-1}}$. The conditional distribution contains all the information in the SDE between times t_k and t_{k-1}. We can interpret 'satisfying the SDE' as 'following the distribution $F_{S_k|S_{k-1}}$'. This is analogous to finding a discrete weak solution. A strong solution adds an additional constraint on how the output should correspond to the Brownian motion increment. Sampling the next value along the path is equivalent to sampling from the inverse distribution of $F_{S_k|S_{k-1}}$. All of this is summarised in equation 4.16, where we let $(\{S_{t_k}|S_{t_{k-1}}\}, \{W_{t_k} - W_{t_{k-1}}\})$ be a discrete strong solution and $\tilde{S}_{t_k} \mid \tilde{S}_{t_{k-1}}$ denote a sample from a discrete weak solution.

$$\tilde{S}_{t_k} \mid \tilde{S}_{t_{k-1}} \overset{d}{=} (S_{t_k} \mid S_{t_{k-1}})(\omega) = F^{-1}_{S_k|S_{k-1}}\left(F_{W_k}\left(\left(W_{t_k} - W_{t_{k-1}}\right)(\omega)\right)\right). \tag{4.16}$$

As we discussed earlier, a standard GAN is only able to recover the first equality, as it only receives information about the conditional distribution. The idea behind the constrained GAN is to learn the following map instead:

$$(S_{t_k} \mid S_{t_{k-1}})(\omega) = F^{-1}_{S_k|S_{k-1}}\left(F_{W_k}\left(\left(W_{t_k} - W_{t_{k-1}}\right)(\omega)\right)\right), \tag{4.17}$$

or equivalently using an i.i.d. sequence of $\mathcal{G}_k$-adapted standard normal random variates $\{Z_k\}$:

$$(S_{t_k} \mid S_{t_{k-1}})(\omega) = F^{-1}_{S_k|S_{k-1}}\left(F_Z\left(Z_k(\omega)\right)\right). \tag{4.18}$$

Clearly, if the generator learned this map, its output would be adapted to $\mathcal{G}_k$. The generator architecture remains unchanged, given by:

$$G_\theta(Z_k(\omega), t_k, t_k - t_{k-1}, S_{t_{k-1}}) \to \hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}(\omega). \tag{4.19}$$

To achieve this, the GAN must be trained on exactly those $\{Z_k\}$ that correspond to the discrete strong solution training set, i.e. $\{S_{t_k}|S_{t_{k-1}}\}$. The only practical consideration is how the samples $\{Z_k\}$ are obtained. In the GBM case, they are already available, since exact simulation involves constructing the process $\{Z_k\}$ as shown in equation 4.9. In case of the CIR process, the GAN could be trained on 'reconstructed Brownian motion increments' from training data $\{S_{t_k}|S_{t_{k-1}}\}$ using lemma 4.10 and remark 4.11:

$$Z_k(\omega) = F_Z^{-1}\left(F_{S_k|S_{k-1}}\left((S_{t_k} \mid S_{t_{k-1}})(\omega)\right)\right). \tag{4.20}$$

In section 4.4, various techniques will be discussed to extend this technique to the general case, without ever using the conditional distribution $F_{S_k|S_{k-1}}$, which allows extensions to SDEs beyond those considered in this work. The architecture of the constrained GAN is shown in the schematic of figure 4.3. Note how the generator has not changed, but only the discriminator, as it now receives the additional input Z_k.

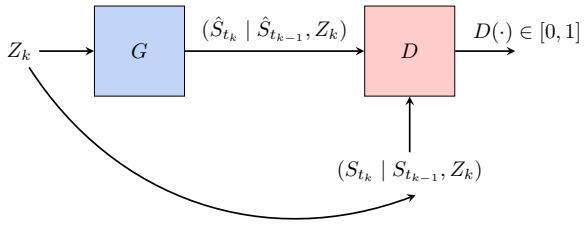


Figure 4.3: Schematic overview of the constrained GAN. The increment Z_k is 're-used' as input to the GAN. Pairs $(\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}, Z_k)$ and $(S_{t_k} \mid S_{t_{k-1}}, Z_k)$ are supplied to the discriminator.

The inclusion of the input Z_k allows the discriminator to 'observe' how the generator maps Z_k to its output and compare this with the pair $(Z_k, S_k \mid S_{k-1})$ by the exact scheme. This will force the generator to resemble the map that the exact scheme makes from Z_k to the output $S_{t_k} \mid S_{t_{k-1}}$.

4.4. General Case: Construction of the Constrained GAN

The construction of the constrained GAN has already been discussed for the examples of GBM and the CIR process, which are the SDEs under consideration in this thesis. However, in order to extend the constrained GAN beyond examples where the conditional CDF is available explicitly, we need methods that work for general SDEs. Three methods are proposed for constructing the constrained GAN architecture in the general case. The first is to invoke a discrete-time scheme on a very small partition to obtain pairs of $\{S_{t_k} \mid S_{t_{k-1}}\}, \{W_{t_k} - W_{t_{k-1}}\}$ on our discretisation. The second option is to use a single-step discrete-time scheme to enforce adaptedness via regularisation of the generator loss function. Thirdly, one could use Karhunen-Loève expansion of the Brownian motion. The first and most straightforward extension is presented here, while the remaining two are included in the appendix.

4.4.1. Train on high quality discrete-time approximation

A way to obtain the $\mathcal{G}_k$-adapted sequence $\{Z_k\}$ for general problems is by constructing the training set through a high-quality discrete-time approximation, instead of using exact simulation. We use the fact that discrete-time schemes such as the Euler and Milstein schemes converge path-wise to the strong solution. Suppose the time of interest is t_k and the starting point is $t_{k-1} \in [0, t_k)$. The interval $[t_{k-1}, t_k]$ is discretised with a very large number of points n, e.g. $O(10^6)$ with time step $\delta t := \frac{t_k - t_{k-1}}{n}$. If, for example, a Milstein scheme is used on this interval, it will produce a high-quality approximation of the stochastic integral at time t_k. The goal of the GAN is then to sample the path from t_{k-1} to t_k in a single step of size $t_k - t_{k-1} \gg \delta t$ (or several steps, each much larger than δt). This setting is illustrated in figure 4.4. In the example of the Milstein scheme, the approximation to the discrete strong solution $\hat{S}_{t_k} \mid S_{t_{k-1}}$ for each $k \in \{1, \ldots, N\}$ is defined by:

$$\hat{S}_{t_k} = S_{t_{k-1}} + \sum_{j=0}^{n-1} \Big[A(\tau_j, \hat{S}_{\tau_j})\delta t + B(\tau_j, \hat{S}_{\tau_j})\sqrt{\delta t}U_{j+1} + \frac{1}{2}\delta t B(\tau_j, \hat{S}_{\tau_j})B'(\tau_j, \hat{S}_{\tau_j})\left(U_{j+1}^2 - 1\right)\Big], \quad (4.21)$$

where τ_j are the times on the fine grid, i.e. $\tau_j = \tau_{j-1} + j\delta t$ and $U_j \overset{iid}{\sim} N(0,1)$. The approximation of the corresponding Brownian motion increment between times t_{k-1} and t_k is given by:

$$\hat{W}_{t_k} - \hat{W}_{t_{k-1}} = \sum_{j=1}^{n} \sqrt{\delta t}U_j. \quad (4.22)$$

To connect this increment to the sequence Z_k as input to the GAN, we will use the following lemma.

Lemma 4.13 (Fine grid to coarse grid). *Let $\{\tau_j\}_{j=0}^n$ form a discretisation of $[t_{k-1}, t_k]$, such that $\tau_j - \tau_{j-1} =: \delta t \ \forall j \in \{1, \dots, n\}$. Let $\hat{W}_{t_k} - \hat{W}_{k-1}$ be a realisation of a Brownian motion increment, given by $\hat{W}_{t_k} - \hat{W}_{t_{k-1}} = \sum_{j=1}^n \sqrt{\delta t}U_j$, where $\{U_j\}_{j=1}^n$ is a sequence of i.i.d. $N(0,1)$ random variables. Then, for every realisation of the increment $\hat{W}_{t_k} - \hat{W}_{t_{k-1}}$, there exists a unique realisation of $Z \sim N(0,1)$, given by:*

$$Z = \frac{\sqrt{\delta t}}{\sqrt{t_k - t_{k-1}}} \sum_{j=1}^{n} U_j \ P\text{-a. s.} \quad (4.23)$$

Proof. Since $\hat{W}_{t_k} - \hat{W}_{t_{k-1}}$ is a continuous random variable, for every realisation of the sequence $\{U_j\}_{j=1}^n$, there is a unique realisation of $\hat{W}_{t_k} - \hat{W}_{t_{k-1}}$. At the same time, we have $\sqrt{t_k - t_{k-1}}Z \sim N(0, t_k - t_{k-1})$, if $Z \sim N(0,1)$, which is also a continuous random variable. Since the map $Z \mapsto \sqrt{t_k - t_{k-1}}Z$ is a bijection, for every realisation of $\hat{W}_{t_k} - \hat{W}_{t_{k-1}}$, there exists a unique realisation of $Z = \frac{1}{\sqrt{t_k - t_{k-1}}}(\hat{W}_{t_k} - \hat{W}_{t_{k-1}})$ P-a. s. Combining this with the definition of the increment $\hat{W}_{t_k} - \hat{W}_{t_{k-1}} = \sum_{j=1}^n \sqrt{\delta t}U_j$ yields equation 4.23. □

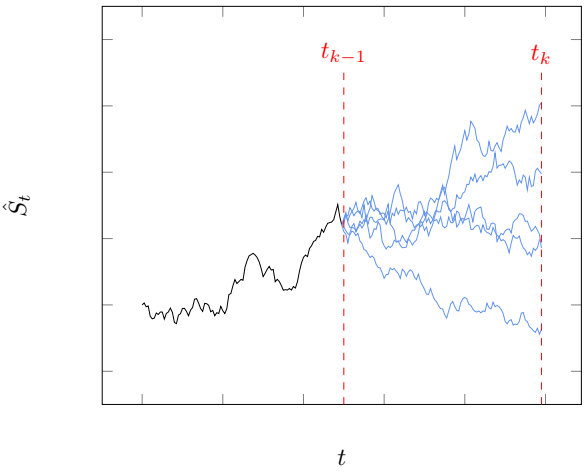


Figure 4.4: Illustration of the extension for general one-dimensional SDEs. This time, the blue solid lines are obtained using a high-quality discrete-time approximation, on which the constrained GAN is trained. The random increment between Z_k times t_{k-1} and t_k can be obtained using lemma 4.13.

Thus, we can recover the $\mathcal{G}_k$-adapted process $\{Z_k\}$ as follows, for all $k \in \{1, \dots, N\}$:

$$Z_k = \frac{\sqrt{\delta t}}{\sqrt{t_k - t_{k-1}}} \sum_{j=1}^{n} U_j, \quad (4.24)$$

i.e. Z_k is the standard normal variate that we would use to sample the Brownian motion increment $\hat{W}_{t_k} - \hat{W}_{t_{k-1}}$ as $\sqrt{t_k - t_{k-1}}Z_k$. The constrained GAN would be trained on pairs $(\{\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}\}, \{Z_{t_k}\})$.

This allows the GAN to be trained on samples with varying $S_{t_{k-1}}$ and t_k, similar to the setting in this thesis, but this time the exact distribution $F_{S_k|S_{k-1}}$ was never used. Thus, even if $F_{S_k|S_{k-1}}$ is not available, the constrained GAN is able to learn an approximation of it via the dataset obtained with the Milstein scheme (or similar discrete-time scheme).
The great advantage of this technique is the ability to parallellise the computation: equation 4.21 allows one to compute many samples at once for each time step, forming the training set. One could define various starting points $S_{t_{k-1}}$ and $t_k - t_{k-1}$ as conditional input to the GAN using a single training set of paths and steps.

4.5. Data pre- and post-processing

The GAN could be trained directly on the process $\{S_{t_k}\}$ and learn a range of distributions starting at different values of $S_{t_{k-1}}$. However, in this work the data is first pre-processed by computing the logreturns, which is also done in the related work by [5]. The logreturns are computed as follows:

$$R_{t_k} := \log\left(\frac{S_{t_k} \mid S_{t_{k-1}}}{S_{t_{k-1}}} + \varepsilon\right), \tag{4.25}$$

for some ε, chosen very small, e.g. 10^{-8} to avoid division by zero (this could form a problem for the CIR process if the Feller condition is not satisfied, and S_{t_k} can get arbitrarily close to zero). The generator is trained to sample the next step of the process $\{R_{t_k}\}$. During inference, the reverse post-processing operation is performed:

$$\hat{S}_{t_k} \mid S_{t_{k-1}} = S_{t_{k-1}} e^{G_\theta(Z, S_{t_{k-1}}, C)}, \tag{4.26}$$

where a hat is used to stress the conditional GAN approximation and $C \in \mathcal{C}$ is some (empty in the vanilla GAN case) tuple of conditional parameters, e.g. including $t_k, t_k - t_{k-1}$ and possibly more parameters. If the GAN predicts a single time step, the 'previous' value came from the discrete strong solution, which is why it does not have a hat. If the GAN is applied iteratively, we should replace $S_{t_{k-1}}$ in equation 4.26 by $\hat{S}_{t_{k-1}}$.
The motivation for using logreturns is two-fold. Firstly, paths are constructed by iteratively computing $S_{t_k} \mid S_{t_{k-1}}$. By using logreturns, the relative increment based on the previous value is computed. This way, the next step is automatically centered near the previous value. This makes the GAN scale-invariant to the asset price under consideration, only considering how the next relative increment follows from the previous one. If this were not the case, a conditional GAN would have to learn how to center the distribution very accurately for a theoretically unbounded range of $[(S_{t_k})_{(0)}, \ldots, (S_{t_k})_{(K)}]$, if the training set consists of K samples and $(S_{t_k})_{(i)}$ denotes the i'th order statistic of the target data at time t_k. It is assumed that a previous value $S_{t_{k-1}}$ is always available. This is not restrictive in the context of asset paths, since the process $\{S_{t_k}\}$ is adapted and has a known starting point S_0. In [5], the dataset is standardised after pre-processing and the neural network is trained on the logreturns, without transforming back to the process itself. Our work differs in that we are interested in the conditional distribution itself at each time step, while the goal in [5] is to reproduce the autocorrelation structure of time series models and real stock market data. Since this work is set in the context of SDEs, the autocorrelation structure is contained entirely in between two time steps, due to the Markov property of SDEs. We do transform back and require a reverse transformation that does not violate adaptedness by including the moments from the dataset.
Secondly, neural networks have been shown to work best if the inputs to the network are centered around zero and have equal variance, cf. LeCun et al. [88]. To this end, the input data is typically standardised. On common problems, such as image classification, this is straightforward: one could force the data to have zero mean and unit variance by standardising the pixel values in each colour channel, see for example [51]. However, in our case, we also need to transform back from the process R_{t_k} to S_{t_k}. If we were to include the first two moments of the training set in the reverse transformation, we would 'build in' information from the training set into the network, which is undesirable. On image problems, this would not be a problem, as pixel values always vary on a finite range, so a scaling factor could be included into the model without loss of generality. In our case, the logreturns allow us to exclusively use the previous asset value $S_{t_{k-1}}$ in the reverse operation, which satisfies the adaptedness requirement of the process $\{S_{t_k}\}$.

It should be noted that the GBM problem is simplified to shifting and scaling the normal distribution after the logreturns transformation, although the GAN must still accurately adjust the mean and variance for each Δt. In the case of the CIR process, the logreturns are not trivial, especially if the Feller condition is not satisfied, as we will see in the results.

4.5.1. Adaptedness of logreturns

Since the GAN will be trained on the logreturns instead of the process itself, we should ensure ourselves that the theory discussed in this chapter still holds for the logreturns process. In the derivation of theorem 7.1.2 in [1] due to Øksendal, the strong solution is defined as a map $\varphi : (\omega, t_{k-1}, t_k, S_{t_{k-1}}) \mapsto S_{t_k}(\omega)$. The process $\{S_{t_k}\}$ is still $\mathcal{G}_k$-measurable after a bounded Borel function $g(\omega, x)$ is applied [1], i.e. if the map defining the strong solution changes from φ to $g \circ \varphi$. Since we have bounded the logreturns from below by $\log(\epsilon)$ and can arbitrarily bound the training data from above, e.g. by assigning a maximum, the map $g(\omega, x) = \log\left(\frac{S_{t_k}(\omega)}{x} + \epsilon\right)$ is a bounded Borel function on $\Omega \times [\log(\epsilon), S_{\max}]$, since its pre-image is in $\mathcal{B}(\mathbb{R})$, ensuring that the logreturns of the process S_{t_k} are still $\mathcal{G}_k$-measurable.

4.6. Construction and Analysis of Paths

Assume for now that we have access to a trained conditional GAN, which was trained on $(t_k, \Delta t := t_k - t_{k-1}, S_{t_{k-1}})$. As noted in remark 4.8, in the case of Itō diffusions, which includes a large range of financial SDEs, we can omit the inclusion of t_k, as the transition from one value along the path to the next is only dependent on $S_{t_{k-1}}$ and Δt. The conditional GAN will be used to construct synthetic paths, which will be compared with a reference of the strong solution and the Euler and Milstein schemes. Suppose, without loss of generality, that we set the problem on a discretisation $\{t_k\}_{k=0}^N$ of $[0, T]$ with equal spacing, $t_k = k\Delta t := k\frac{T}{N}$ and $t_0 = 0$. Each output S_{t_k} becomes the next input to the GAN at time t_{k+1}. This recursive application of the GAN is used to construct a path, as shown in equation 4.27, where the value of the synthetic path is given at time $k\Delta t$.

$$S_{t_{k+1}} = S_{t_k} \exp(G_\theta(Z_k, t_k, \Delta t, S_{t_k}), \tag{4.27}$$

where $\{Z_k\}_{k=1}^N$ is a sequence of i.i.d. $N(0, 1)$ samples. Note that Δt may be varied to construct paths with different time steps (or Δt may even be varied on a single path during inference). The Euler and Milstein schemes provide approximations to the discrete strong solution of varying accuracy, which can be used to 'benchmark' the paths generated by the GAN.
The same random sequence $\{Z_k\}_{i=1}^n$ is used for all sampling techniques: the GAN, exact scheme, Euler scheme and Milstein scheme to construct paths. This chapter has already covered how this sequence may be obtained in various situations. Note the importance of using the same Z_k in all schemes at each t_k, otherwise each path would be adapted to its own filtration and we could only compare them in distribution. Realisations of paths could be plotted together for all four sampling techniques discussed here in a single figure.

4.6.1. Analysing the synthetic paths

Although the GAN output will be compared to discrete-time schemes, we should emphasise how it is based on very different principles than the Euler and Milstein schemes, as pointed out earlier in this chapter. The conditional GAN will depend very differently on the time step Δt than the discrete-time schemes do. Any error in the GAN approximation will be solely due to the GAN's ability to sample from $F_{S_{t_k}|S_{t_{k-1}}}$. In the ideal case, the quality of the GAN approximation would not depend on Δt at all. Therefore, smaller time steps do not automatically translate into a more accurate approximation, as they do for the Euler and Milstein schemes.
Nevertheless, the literature on discrete-time schemes does provide useful concepts for comparing paths, i.e. the weak (e_w) and strong (e_s) errors introduced in the preliminaries. They are restated here:

$$e_w = \left|\mathbb{E}\left(f(S_{t_k+\Delta t})\right) - \mathbb{E}\left(f(\hat{S}_{t_k+\Delta t})\right)\right|, \tag{4.28}$$

$$e_s = \mathbb{E}\left|S_{t_k+\Delta t} - \hat{S}_{t_k+\Delta t}\right|, \tag{4.29}$$

where f is a real-valued polynomial function. Recall how we defined error bounds on the discrete-time approximation in the preliminaries. These bounds, as a function of Δt, gave rise to varying orders of convergence for discrete-time schemes. However, since the GAN is a very different type of approximation, these need not apply to the GAN. With this in mind, we could still use the concepts weak and strong error to compare the paths obtained with the GAN to the exact solution and the approximation obtained with the Euler and Milstein schemes.
Both discrete-time schemes and the GAN approximation have in common that they construct paths recursively. Therefore, an additional benchmark could be defined that is sensitive to how the error accumulates over time. Since all schemes are defined recursively, it is challenging to make general statements about how this error evolves with k for fixed Δt. Still, this error can be measured empirically, using the concept of weak and strong errors, but with varying amount of steps k instead of varying Δt. We will refer to these errors as the weak error over time and strong error over time.

4.7. Practical Considerations for the CIR Process

There are two practical considerations to be taken into account for the CIR process: 1) discrete-time schemes could give rise to negative values, which are problematic when computing the square root in equation 2.15; 2) if the Feller condition is not satisfied, the process $\{S_t\}$ jumps between orders of magnitude, due to the presence of a near-atom of probability mass around zero. The process could jump from e.g. 10^{-6} to 0.1 in a single time step and could hit zero infinitely often on a bounded time interval [89].

4.7.1. Discrete-time schemes for the CIR process

The Euler scheme will be replaced by what we will refer to as the (partially) 'truncated' Euler scheme, as mentioned e.g. in [90]. In the case of the CIR process it is given $\forall k \in \{0, \dots, N-1\}$ by:

$$\hat{S}_{t_{k+1}} = \hat{S}_{t_k} + \kappa(\bar{S} - \hat{S}_{t_k})\Delta t + \gamma\sqrt{\hat{S}^+_{t_k}}\sqrt{\Delta t}Z_k, \tag{4.30}$$

where $\hat{S}_{t_0} = S_{t_0}$ and $\hat{S}^+_{t_k} := \max(\hat{S}_{t_k}, 0)$. $Z_k \sim N(0,1)$. Note that the truncated Euler scheme may still produce negative paths, in which case the term with the Brownian motion equals zero at step $k+1$.
A modified version of the Milstein scheme can be defined as well. In [91], such a Milstein-type scheme is proposed specifically for the CIR process, which will be implemented as a reference to the CIR process. This truncated Milstein scheme is given by:

$$\hat{S}_{t_{k+1}} = \left(\left(\max\left(\frac{1}{2}\gamma\sqrt{\Delta t}, \sqrt{\max\left(\frac{1}{2}\gamma\sqrt{\Delta t}, \hat{S}_{t_k}\right)} + \frac{1}{2}\gamma\sqrt{\Delta t}Z_k\right)\right)^2 + \left(\kappa\bar{S} - \frac{1}{4}\gamma^2 - \kappa\hat{S}_{t_k}\right)\Delta t\right)^+, \tag{4.31}$$

with $\hat{S}_{t_0} = S_{t_0}$ and $(\cdot)^+ := \max(\cdot, 0)$. The one-step order of convergence of this scheme depends on the previous value S_{t_k}, time step Δt and degrees of freedom parameter δ [91]. However, the authors of [91] show that the scheme converges in L^p with order $\frac{1}{2p}\min(1, \delta)$.

4.7.2. Pre-processing step if the Feller condition is not satisfied

If the Feller condition is not satisfied, the logreturns can get arbitrarily close to $\log(\varepsilon)$ and $-\log(\varepsilon)$. Although it is possible to re-normalise the logreturns by a factor $\log(\varepsilon)$, a very small interval around 0, say $(0, q]$, would become a very large interval in the space of logreturns. The GAN would attempt to approximate the conditional distribution of the returns on this large region. However, in practice, we are most likely not interested in the difference between values very close to zero e.g. 10^{-6} versus 10^{-8} or smaller. In the space of logreturns, an interval $[\log 10^{-8}, \log 10^{-6}]$ gives rise to a large interval compared to $[10^{-8}, 10^{-6}]$. In practice, however, we are not interested in precise differences of order 10^{-6} and below. If these quantiles are very close, the difference at the output is benign. Therefore, by using logreturns if the Feller condition is not satisfied, capacity of the neural network would be 'wasted' on very small differences in quantiles, which are not of interest. We will use a different scaling technique if the Feller condition is not satisfied, defined by the pre-processing step in equation 4.32 and post-processing step in equation 4.33.

$$R_{t_k} := \frac{S_{t_k} \mid S_{t_{k-1}}}{\bar{S}} - 1, \tag{4.32}$$

$$\hat{S}_{t_k} \mid S_{t_{k-1}} = \left|(R_{t_k} + 1)\bar{S}\right|. \tag{4.33}$$

This scaling technique centres the distribution around 0 and provides a $\mathcal{G}_k$-measurable output for all k, as desired, on which the GAN is trained. The absolute value ensures that the process $\{\hat{S}_t\}$ remains positive. This was preferred over using a ReLU activation at the output of the generator, as it results in zero gradients on $(-\infty, 0)$, which could negatively impact the training process.

Chapter Conclusion

It has been shown how GANs can be used to obtain weak and strong approximations of SDEs on a discretisation in time. A conditional GAN learns how to sample from the conditional distribution between two points on a path. In general, this yields a discrete weak solution. A modified GAN is required to enforce path-wise equality to the strong solution of the SDE. It was shown how this technique can be applied on GBM and the CIR process and further extended to general problems. The GAN will be used to construct synthetic paths, which are compared with paths obtained by exact simulation and the Euler and Milstein schemes. The next chapter outlines the methods used to compare probability distributions, which will be used in the results to study the distribution of the output generated by the GAN.

5

Comparing Probability Distributions

This chapter describes how non-parametric estimators can be used to compare two distributions. This is required for evaluating the weak approximation to the SDE by a GAN. First, the key techniques that will be used in the results section to report on the accuracy of the GAN output are discussed. In the second part of this chapter, an overview is given of several classes of divergence measures and distances. This is intended to provide the necessary background behind the choice of statistics and implementation of the GAN. Recommendations are provided on which statistics to use if the problem is set in higher dimensions.

5.1. Empirical distribution functions

A useful statistic for analysing the GAN output is the well-known empirical cumulative distribution function (ECDF), cf. [92]. Suppose we have a set of observations $\{X_i\}_{i=1}^n$, with $X_i \in \mathscr{X} \subseteq \mathbb{R}\ \forall\ i$, then the ECDF is given by:

$$\hat{F}_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{\{X_i \leq x\}}(x). \tag{5.1}$$

It is easy to show that if a random variable X follows the distribution measure P with cumulative distribution function (CDF) $F(x) := P(X \leq x)$, then $\hat{F}_n$ is unbiased for F and $\hat{F}_n \xrightarrow{p} F$ as $n \to \infty$ [92]. This estimator is also very easy to implement and can be plotted together with the exact CDF. In this thesis, the statistic is used for visually comparing the ECDF of the GAN output and exact CDF of the target distribution.

5.2. Combination of KS Statistic and 1-Wasserstein distance

In order to quantify the quality of the GAN output, two statistics are used: the KS-statistic and the Wasserstein distance. These statistics are chosen among alternatives, as they allow for a plug-in estimator in 1D and measure different features of the output distribution. For a qualitative comparison of various distributional statistics and reasons why the KS-statistic and Wasserstein distance are preferred in the context of this work, see section 5.3.

5.2.1. KS statistic

The Kolmogorov-Smirnov (KS) statistic is known in probability theory from the Kolmorov-Smirnov test between a CDF and its empirical estimate $\hat{F}_X$ [93]. Suppose we observe n observations $\{X_i\}_{i=1}^n$ of a random variable X with CDF $F_X(x)$. The 2-sided 1-sample KS statistic is defined as, cf. Simard et al. [94]:

$$u_n = \max_{i \in \{1,\dots,n\}} \left| \hat{F}_X(X_i) - F_X(X_i) \right|. \tag{5.2}$$

Let $X_{(i)}$ denote the order statistics of the observed random variates. The KS statistic is implemented

efficiently in the SciPy `stats` package [95] as follows:

$$u^+ = \left[\frac{1}{n}, \frac{2}{n}, \dots, 1\right] - \left[F_X(X_{(1)}), F_X(X_{(2)}), \dots, F_X(X_{(n)})\right], \tag{5.3a}$$

$$u^- = \left[F_X(X_{(1)}), F_X(X_{(2)}), \dots, F_X(X_{(n)})\right] - \left[0, \frac{1}{n}, \dots, \frac{n-1}{n}\right], \tag{5.3b}$$

$$u_n = \max(u^+, u^-). \tag{5.3c}$$

The implementation considers the case of the ECDF being greater (u^+) or less than (u^-) the exact CDF and finds the maximum difference. Now, u_n is itself a random variable and has a distribution that is often referred to as the 'KS distribution' [94]. Its limiting distribution is given by Feller in [96]. Note how the preceding describes a one-sided test, i.e. there is a reference CDF available. In our case, this is possible, since we have the exact solution of the SDEs under consideration at our disposal. If this is not possible, one should consider the two-sided KS-statistic where both terms in equation 5.2 are ECDFs. The distribution of u_n is implemented in SciPy's stats package as `kstwobign` [95]. It can be used to compute a p-value given a single observation of u_n, corresponding to a vector with sample size n. If the p-value is above typical choices of a critical region α (e.g. $\alpha = 0.05$), we cannot reject the hypothesis that the output samples are distributed like the reference distribution.

5.2.2. 1D Wasserstein distance

In the one-dimensional setting, the p-Wasserstein distance between the distributions P and Q is given by [97, 98]:

$$w_1(P, Q) = \left(\int_0^1 |P^{-1}(y) - Q^{-1}(y)|^p dy\right)^{\frac{1}{p}}. \tag{5.4}$$

In practice, setting $p = 1$ is most common [99], which will also be chosen in this work. P and Q can be approximated by their ECDF estimates P_n and Q_n. Note how the Wasserstein distance compares the quantiles of both distributions along their support.
Suppose we observe a random variable $X \sim P$ and another random variable $Y \sim Q$. If both KS and Wasserstein distance are used simultaneously, both the quantiles $P^{-1}(\cdot), Q^{-1}(\cdot)$ of a realisation are compared as well as the values of the distribution function, say, $P(\cdot), Q(\cdot)$. Note, however, the abuse of notation by conflating the probability measure with the distribution function, which is common in literature, but the difference should be clear from the context. In a plot of $P(x)$ and $Q(x)$ versus $x \in \text{supp}(P)$, the KS-statistic compares 'vertical differences' between P and Q, while the Wasserstein distance compares 'horizontal' differences, i.e. quantiles. This interpretation follows the discussion in [97] on the Wasserstein distance. This makes both statistics sensitive to different types of perturbations from the reference distribution.

5.2.3. Implementation for analysing the GAN output

Suppose the exact variates follow the distribution P^* with CDF F^*, which is known analytically. Suppose that the GAN produces an output distribution P_θ, which is not available in closed form. However, we can observe the ECDF, say $\hat{F}_\theta$ associated with a finite set of output samples of size n. Since we have a plug-in estimator for both the KS-statistic and the Wasserstein distance, computation of both statistics is straightforward. However, it is not clear how to interpret the numbers coming out of both statistics without some reference. To this end, a vector of samples of size n is drawn from the exact distribution as well, say $\{X\}_{i=1}^n$, with $X_i \sim P^*$. Let $\hat{F}^*$ denote the ECDF based on the vector of samples X. The statistics on an output vector from the GAN can then be compared with an equally sized reference vector X. The statistics themselves will be random variables, as they depend on the particular realisations of the random input samples. Therefore, the KS-statistic and Wasserstein distance are computed 100 times and the mean will be reported for both the GAN output and the reference.
In addition to a single test set, we can measure the GAN output with a more informative, yet computationally intensive benchmark. In order to get a more complete view on how the statistics change with varying test size N, we can compute the KS-statistic and Wasserstein distance for various

N, say $N \in \{10^2, 10^3, 10^4, 10^5\}$ and plot the resulting value of both statistics versus N. This avoids 'cherry-picking' a test sample size, on which the GAN performs as well as the exact variates. In such a benchmark, the methods considered in this thesis for approximating the stochastic integral $S_{t+\Delta t}$ can be compared in a single figure. This can be done as follows: draw two reference vectors X and Y i.i.d. from P^*, sample the GAN output $X_G \sim P_\theta$, compute X_{Mil} with a Milstein scheme and X_{Eul} with an Euler scheme. The discrete-time schemes are computed using a single step of size Δt. This way, the weak approximation by the GAN is compared to exact simulation of the SDE and two discrete schemes in a single benchmark. The experiment is repeated for each $N \in \{10^2, 10^3, 10^4, 10^5\}$ and repeated 10 times to construct a confidence interval. The KS-statistic and Wasserstein distance of the exact variates should decrease indefinitely as N increases, since exact simulation yields outputs that can get arbitrarily close to P^*. This is not the case for the GAN output or the discrete-time schemes, for which the statistics will converge to their limiting non-zero values. They are not zero, since the distribution from which they are sampled is not equal to P^*, which is only the case if exact simulation is used.

5.3. Reference on Distributional Divergences and Distances

Various measures[1] of similarity between probability distributions are discussed here. These methods are not only useful for analysing the weak approximation by the GAN, but also form the central component of the optimisation objectives in generative models, as we saw in the chapter on theory behind GANs, or can be seen in the loss function of Variational Autoencoders (VAEs) cf. [100]. Two key classes of distributional similarity measures are presented: f-divergences and integral probability metrics [101].

f-divergences

f-divergences are a general class of distributional similarity measures defined for two absolutely continuous probability distributions P and Q, dominated by some measure ν on $\mathscr{X} \subseteq \mathbb{R}$. It is more convenient to write them in terms of the densities $p = \frac{dP}{d\nu}$ and $q = \frac{dQ}{d\nu}$. The f-divergence is then defined as, cf. [102]:

$$\varphi_f(P \parallel Q) = \int_{\mathscr{X}} q(x) f\left(\frac{p(x)}{q(x)}\right) d\nu(x), \tag{5.5}$$

with $f : (0, \infty) \to \mathbb{R}$ a convex function. Notable examples from this class are the KL divergence for $f(y) = y\log(y)$, JS-divergence, for $f(y) = \frac{1}{2}\left[(y+1)\log\left(\frac{2}{y+1}\right) + y\log y\right]$ and total variation distance, for $f(y) = \frac{1}{2}|y-1|$ [102]. These identities can be easily verified by plugging in $f(y)$ into equation 5.5 with $y = \frac{p}{q}$.

KL and JS divergences

Let us consider two f-divergences more closely, which appear in the chapter on the theory behind GANs. The Kullback-Leibler divergence (KL divergence) or 'KL information' of a distribution Q from P on $\mathcal{X}$ is defined as [73]:

$$KL(P||Q) = \int_{\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} d\nu(x). \tag{5.6}$$

In [101] and [102], the KL-divergence is defined to equal $+\infty$ if $P \not\ll Q$, extending to potentially disjoint densities. This would be important if the KL divergence were used to compare two vectors of data with potentially disjoint support. It can be shown that $KL(P\|Q) \geq 0$, with equality if and only if $P = Q$ [73]. Note that the KL divergence is unbounded from above and approaches $+\infty$, if $q(x)$ approaches zero. Also note that this divergence is not a distance, as it is not symmetric, i.e. $KL(P\|Q) \neq KL(Q\|P)$.
Another divergence is the Jensen-Shannon divergence (JS divergence), which is based on the KL divergence. It is defined as follows [72]:

[1] Here, 'measure' means 'a way to quantify performance'. The meaning should further be clear from context in which it appears.

$$JS(P\|Q) = \frac{1}{2}KL(P\|M) + \frac{1}{2}KL(Q\|M), \tag{5.7}$$

where $M = \frac{P+Q}{2}$. This divergence is symmetric and can be cast into a distance metric by taking the square root. Note how this divergence is bounded from above by $2\log(2)$ and 'clips' to this value instead of $+\infty$ if the distributions are not absolutely continuous. In [75, Theorem 2.3], it is shown by Arjovsky et al. that both KL divergence and JS divergence clip to their maximum values, if the manifolds of the GAN output and dataset do not perfectly align. This is an undesirable property when using the divergence for studying the output of the GAN, since there are many cases where the two output distributions will be disjoint.
The KL- and JS-divergences do not have a plug-in estimator and are difficult to approximate efficiently, although several non-parametric methods have been proposed, cf. [102, 103], convergence of these methods is slow with respect to the sample size.

Integral Probability Metrics

A different class of distributional similarity measures is the integral probability metric (IPM). Consider the same setting as in the previous paragraph and let $\mathscr{F}$ be a family of real-valued and bounded functions on $\mathcal{X}$. The general definition of an IPM is then given by [101]:

$$I_{\mathscr{F}} = \sup_{f\in\mathscr{F}} \left| \int_{\mathcal{X}} f dP - \int_{\mathcal{X}} f dQ \right|. \tag{5.8}$$

Just as we could vary f for the f-divergence and obtain different kinds of divergences, we can vary the class of functions $\mathscr{F}$ to obtain metrics with different properties. The most popular IPMs used in generative modelling are the Wasserstein distance and maximum mean discrepancy.

Wasserstein Distance

A common IPM is the Wasserstein distance. In this case, $\mathscr{F} = \{f : f \text{ is 1-Lipschitz}\}$, where the Wasserstein distance arises as the dual form of the IPM formulation under the Kantorovic metric [101, p.2]. The Wasserstein distance is defined between distributions P and Q under the p-norm as [97]:

$$w_p(P,Q) = \left(\inf_{\psi\in\Gamma(P,Q)} \mathbb{E}_{(X,Y)\sim\psi} \|X-Y\|_2^p \right)^{\frac{1}{p}}, \tag{5.9}$$

where $\Gamma(P,Q)$ is the set of joint distributions of P and Q. Note how the L_2-norm is chosen here to compare samples X and Y, but this could more generally include any distance. In the 1D setting considered in this work, the 1-Wasserstein distance between a distribution P and Q reduces to $\int_0^1 |P^{-1}(y) - Q^{-1}(y)|dy$, which can be approximated using the empirical distribution functions P_n and Q_n of P and Q [97, 98]. This makes the Wasserstein distance available in closed form in the 1D setting. In higher-dimensional problems, the distance is not available in closed form and one typically has to rely on more expensive numerical schemes [98]. One could resort to techniques such as 'sliced Wasserstein distances', where 1D projections of P and Q onto their marginals are considered, as reported in [98, 104].
The Wasserstein distance allows for an interesting interpretation, which is why it is also referred to as the 'Earth Mover Distance' [101]. It represents a quantity of mass transported along the distance between the variates X and Y, weighted with the optimal joint density ψ [97]. ψ can be interpreted as the weighting function that minimises the 'cost' of moving the mass from X to Y or vice-versa. This problem is known as the 'optimal transport problem', cf. [105]. This topic itself is beyond the scope of this thesis, but what we can take away from the Wasserstein distance is that it measures the optimal way of weighting differences between the variates themselves, i.e 'horizontally' [97]. To see this, consider the horizontal axis in the 1D case. Then the Wasserstein distance is the distance between points on the support, weighted by joint density ψ.
Arjovsky et al. [76, p.5] show that if the GAN output G_θ is continuous in the parameter set θ, then so is the Wasserstein distance. Arjovsky et al. also show that the same is not true for the KL- and JS-divergences. This property, along with the presence of a closed-form estimator, make the Wasserstein distance suitable for analysing the GAN output in this work.

Maximum Mean Discrepancy

Suppose that we now restrict ourselves to $\mathscr{F} = \{f : \|f\|_{\mathscr{H}_k} \leq 1\}$, with $\mathscr{H}_k$ a so-called 'reproducing kernel Hilbert space' (RKHS) on $\mathcal{X}$ with kernel function $k(\cdot, \cdot)$. Without getting into technical details, a RKHS is a Hilbert space defined by a specific type of kernel, called the 'Mercer kernel' [97] and the condition $\|f\|_{\mathscr{H}_k} \leq 1$. The maximum mean discrepancy (MMD) between distributions P and Q is then defined as [106]:

$$MMD(\mathscr{F}, P, Q) = \sup_{\|f\|_{\mathscr{H}_k} \leq 1} \left(\int_{\mathcal{X}} f dP - \int_{\mathcal{X}} f dQ \right). \tag{5.10}$$

If k is 'characteristic' for $\mathscr{H}$, a property which will not be discussed here as it is beyond the scope of the section, $MMD = 0 \iff P = Q$. A Gaussian is an example of such a kernel. What makes the MMD attractive is that it has an unbiased plug-in estimator, regardless of the dimension of $\mathcal{X}$ [106]. If we draw two n-dimensional vectors $X \sim P$ and $Y \sim Q$, then:

$$MMD_n(X, Y) = \frac{1}{n(n-1)} \sum_{i \neq j} \big[k(X_i, X_j) + k(Y_i, Y_j) - k(X_i, Y_j) - k(X_j, Y_i) \big]. \tag{5.11}$$

In [97, p.7], the MMD is described as "an extremely smoothed Wasserstein distance", where the authors refer to the presence of a kernel with the term 'smoothed'. The MMD depends strongly on the choice of kernel, which is often chosen to be a Gaussian radial basis function, since it satisfies the 'characteristic' property [97, 106] and is easy to implement. In our 1D case, the Wasserstein distance would be a more parsimonious choice, as we do not have to concern ourselves with the choice of kernel. In the RHKS, it is not easy to compare the GAN output variates with the exact distribution. Additionally, applying a kernel may distort the distribution of the output. For example, if the Feller condition is not satisfied for the CIR process, this would be particularly pronounced around 0, where the kernel would put mass on negative values. Still, the MMD is presented here as an alternative in case the problem setting is extended to higher dimensions. The MMD would then be attractive, as the kernel evaluations could be performed efficiently, while most other methods become intractable.

Chapter Conclusion

This chapter has provided an overview of the statistics that will be used to evaluate the weak approximation to the SDE by a GAN. A finite test set is constructed with reference samples to compare to the GAN output. This setup can be validated by drawing test sets of varying size to avoid misinterpreting the statistics. Secondly, a reference of non-parametrics techniques has been provided, including two families of common distributional divergences and distances. This included discussions that were too much in detail in the preceding chapters, but are relevant to understand the design choices and background behind the techniques used in this thesis.

6

Results

Our goal is to approximate the strong solution on a discretisation $\{t_k\}_{k=1}^N$ of $[0,T]$ using a GAN, as described in chapter 4. The output of a trained conditional GAN will form an approximation to the conditional distribution of the process $\{S_t\}$ between two time steps t and $t+\Delta t$, i.e. the transition distribution $F_{S_{t+\Delta t}|S_t}$. The constrained GAN architecture will ensure that the map $Z \mapsto G(Z, S_t, \Delta t)$ approximates the discrete strong solution. We will first test the GAN's ability to sample from $F_{S_{t+\Delta t}|S_t}$. Then, we will study synthetic paths obtained by repeated sampling from this distribution. Finally, we study the map $Z \mapsto G(Z, S_t, \Delta t)$ explicitly for both the vanilla GAN and the constrained GAN. Unless stated otherwise, the results in this chapter are set in the case where the Feller condition is violated, as it forms the most challenging case among the SDEs under consideration. The results for the GBM problem and the CIR process with the Feller condition satisfied are included in the appendix.

6.1. Approximating the Conditional Distribution

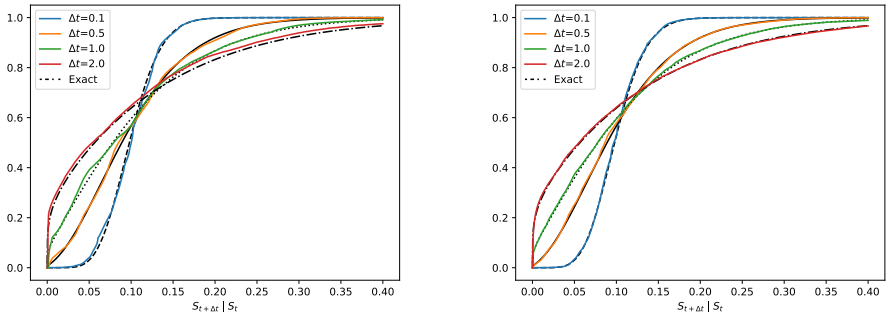
A conditional GAN, as explained in sections 3.4.2 and 4.2.2, is trained on a dataset of 100,000 triplets $\big((S_{t+\Delta t} \mid S_t), S_t, \Delta t)\big)$, over 20,000 iterations. The constrained GAN is trained on quartets $\big((S_{t+\Delta t} \mid S_t), S_t, \Delta t, Z)\big)$, where Z is obtained through the technique explained in lemma 4.10 and remark 4.11. It corresponds to the Brownian motion increment between the time steps t and $t+\Delta t$ and is related to the training samples as shown in equation 6.1.

$$Z = F_Z^{-1}\left(F_{S_{t+\Delta t}|S_t}\left(S_{t+\Delta t} \mid S_t\right)\right), \tag{6.1}$$

where F_Z^{-1} is the quantile function of the $N(0,1)$ distribution. In this case, $F_{S_{t+\Delta t}|S_t}$ resembles the non-central χ^2-distribution. We choose the parameters of the CIR process as $\kappa = 0.1$, $\bar{S} = 0.1$ and $\gamma = 0.3$, such that the degrees of freedom parameter becomes $\delta \approx 0.44$, i.e. well below 2, such that the distribution becomes near-degenerate around 0. A list of the choices of parameters on all problems is included in the appendix. S_t ranged from 10^{-4} to 0.9 and Δt ranged from 0.05 to 2. The GAN is trained to sample from the solution a single time step ahead, while varying the previous value S_t and time step Δt to 'teach' the conditional GAN how the conditional distribution changes with S_t and Δt. This is done for both the vanilla conditional GAN and the constrained conditional GAN using the same training process. This includes identical choices of random seed initialisation, ensuring that the datasets on which the GANs are trained are identical. The generator and discriminator networks were also identical for both GAN architectures, consisting of 4 hidden layers with 200 hidden neurons. This architecture is similar in size to the related work by [23], where the authors use 3 hidden layers with several hundreds of neurons. Our choice of architecture was found to be sufficient to approximate the distribution, while feasible within the constraints of the hardware and time available. A detailed description of the generator and discriminator networks is included in the appendix, along with details about the training process.

From now on, let us refer more compactly to the vanilla conditional GAN and constrained conditional GAN as 'vanilla GAN' and 'constrained GAN', since all results will include the conditional GAN architecture from now.

The ECDF of the GAN approximation to $F_{S_{t+\Delta t}|S_t}$ after a single time step is shown in figure 6.1, for particular choices of Δt and S_t fixed at 0.1. Both GANs are able to adapt the shape of the output distribution in accordance with the input parameters. Note how we need to specify a test set $(S_t, \Delta t)$ to study the GAN's approximation of $F_{S_{t+\Delta t}|S_t}$. The KS-statistic and Wasserstein distance at the test condition $(S_t, \Delta t) = (0.1, 1)$ are shown in table 6.1 on a test set of 2,000 exact samples. These exact samples were drawn from the non-central χ^2-distribution using the corresponding parameters. The constrained GAN outperforms the vanilla GAN in approximating $F_{S_{t+\Delta t}|S_t}$, which is clear by visually comparing the figures and statistics in the table.



(a) Vanilla GAN

(b) Constrained GAN

Figure 6.1: ECDF plots of the conditional GAN output on the CIR process conditional on $(S_t, \Delta t)$, with S_t held fixed at 0.1. The constrained GAN appears to improve the weak approximation as well if both networks are trained on equal settings.

Table 6.1: Conditional GAN results on the GBM problem and the CIR process, both for the vanilla GAN and the constrained GAN. The test conditions were $\Delta t = 1$ and $(S_t, \Delta t) = (0.1, 1)$ for GBM and CIR process, respectively. The statistics are computed using 2,000 generated samples and are compared with an equally sized set of reference samples, e.g. 'KS stat.' versus 'KS stat. ref.'

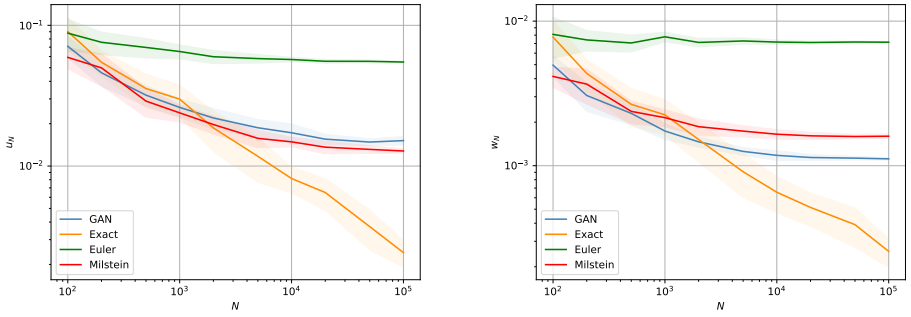
	KS stat.	p-value	KS stat. ref.	Wass. dist.	Wass. dist. ref.
Vanilla GAN	6.5E-02	0.00	1.9E-02	4.7E-03	3.5E-03
Constrained GAN	2.8E-02	0.10	2.0E-02	3.7E-03	3.7E-03

The p-value of the KS-statistic shows that the KS-statistic is is high for both GAN approximations compared to the test set. However, the constrained GAN output has identical Wasserstein distance as the equally sized test set. Although these results seem contradictory, it can be explained by the near-atom near zero and the fact that the KS-statistic and Wasserstein distance measure different features of both distributions. The KS-statistic measures the largest absolute difference between the ECDF and CDF in figure 6.1. Around 0, the slope in the ECDF is very high, which leads to a large difference between CDF and ECDF if there is a small difference in probability mass assigned to a quantile, even though the quantiles may be very close. This makes the KS-statistic highly sensitive due to the near-singular behaviour around the origin. This does not hold for the Wasserstein distance, since in 1D it is sensitive to differences between the quantiles of both distributions. Therefore, the Wasserstein distance is more informative if the Feller condition is not satisfied, as the KS-statistic becomes highly sensitive to small differences in the quantiles near zero. The Wasserstein distance suggests that both distributions are very close. In case the Feller condition was satisfied and on the GBM problem, both the KS-statistic and Wasserstein distance were similar to the test set for the constrained GAN. These results are included in the appendix. The constrained GAN outperformed the vanilla GAN in all cases when approximating the conditional distribution.

6.1.1. Statistics using various test sizes

We study the effect of the size of the test set on the KS-statistic and Wasserstein distance, as it is not clear a priori what would be an appropriate test size to compare the exact variates with the GAN output. In figure 6.2, the KS-statistic and Wasserstein distance corresponding to the constrained

GAN are plotted for the test condition $(S_t, \Delta t) = (0.10, 0.33)$ as a function of the size of the test set, using the setting introduced in section 5.2.3. A single vector of exact samples is drawn for each choice of test size, which is used to compute the statistics on the GAN output, truncated Euler and Milstein schemes and another i.i.d. drawn batch of exact samples. This way, we avoid having to choose a particular test size, while still being able to distinguish between the quality of the different approximations. At $\Delta t = 0.33$, the constrained GAN outperforms the truncated Milstein scheme in terms of the Wasserstein distance, but not in terms of the KS-statistic. If Δt is increased further, the constrained GAN outperforms both the truncated Euler and Milstein schemes on both statistics. This shows that if the time step is large enough, in this case around $\Delta t > 0.33$, the GAN outperforms discrete-time schemes on a single-step approximation. This is unsurprising, since discrete-time schemes rely on the time step being small enough to remain accurate, however, it allows us to establish a baseline for the quality of the approximation in distribution. We could make similar plots for the vanilla GAN, which outperforms the truncated Milstein scheme around $\Delta t > 0.5$. The lower the time step at which the GAN improves on a single-step Milstein approximation, the better the GAN output approximates the conditional distribution. Our results again confirm that the approximation due to the constrained GAN is more accurate than the vanilla GAN.

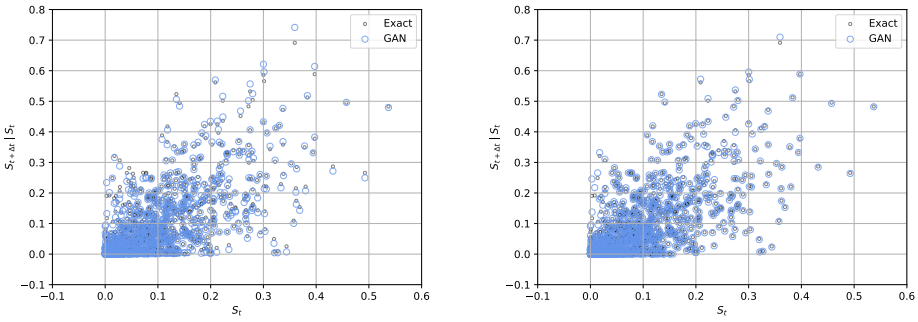


(a) KS-statistic (b) Wasserstein distance

Figure 6.2: Constrained GAN: KS-statistic and Wasserstein distance versus the size of the test set for $\Delta t = 0.33$ and $S_t = 0.1$, if we approximate $F_{S_{t+\Delta t}|S_t}$ with the GAN and truncated Euler and Milstein schemes. The mean is reported based on 20 experiments. The standard deviation is shown as the shaded bands.

6.1.2. Autocorrelation structure

To test if the GAN is indeed able to capture the autocorrelation structure between $S_{t+\Delta t}$ and S_t, 1,000 samples are drawn from the non-central χ^2-distribution given $S_0 = 0.1$ and $\Delta t = 1$. Then, we sample 1,000 exact samples $S_{t+\Delta t} \mid S_t$, using the exact conditional distribution once more. The same 1,000 realisations of S_t are given as conditional input to the vanilla and constrained GAN. Further still, we use equation 6.1 to find the 1,000 realisations of Z that we should provide as input to the GAN to compare the output 'point-by-point'. If the GAN provided a path-wise approximation, not only would the autocorrelation structure be the same, but the output would completely overlap with the 1,000 exactly drawn variates conditional on the batch S_t. In figure 6.3, a scatter plot is shown of $S_{t+\Delta t}$ versus S_t. We see that the shape of the 'cloud' of points is similar, which suggests the autocorrelation structure is similar. Furthermore, we see that the exact samples and GAN approximation indeed overlap on most samples, which suggests that the GAN learns the map $Z \mapsto (S_{t+\Delta t} \mid S_t)$ correctly and provides a path-wise approximation. As explained in section 4.2.4, this allows the GAN to approximate the discrete strong solution on any discretisation. The only limiting factor would be the range of $(S_t, \Delta t)$ on which the GAN is trained. If the Feller condition is satisfied, we find similar results, although this time, the vanilla GAN does not provide a path-wise approximation, as shown in the appendix. On the GBM problem, the autocorrelation structure is completely captured by the logreturns transformation, as the post-processing step scales the logreturns with S_t. Therefore, we do not need to perform this test for GBM.

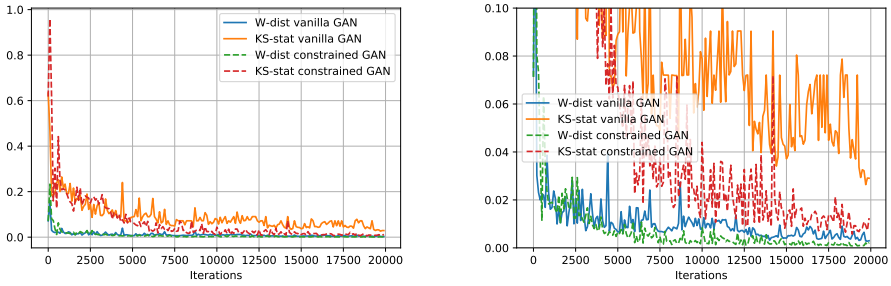


(a) Vanilla GAN (b) Constrained GAN

Figure 6.3: Plot of $S_{t+\Delta t} \mid S_t$ versus S_t to test if the autocorrelation structure of the GAN output matches that of variates obtained through exact simulation.

6.1.3. Training process

Finally, we can compare the KS-statistic and Wasserstein distance during the training phase for both the vanilla and constrained GAN, using a test condition of $(S_t, \Delta t) = (0.1, 1)$. The statistics are computed on a test set of equal size to the training set, i.e. 100,000 samples. The result is plotted in figure 6.4 against the training iterations. The constrained GAN achieves a lower value on both statistics, while the variance in the statistics during training appears to be lower, suggesting a more stable training process. Similar results were found if the Feller condition was satisfied and on the GBM problem, of which the results are included in the appendix.



(a) (b) Zoomed in

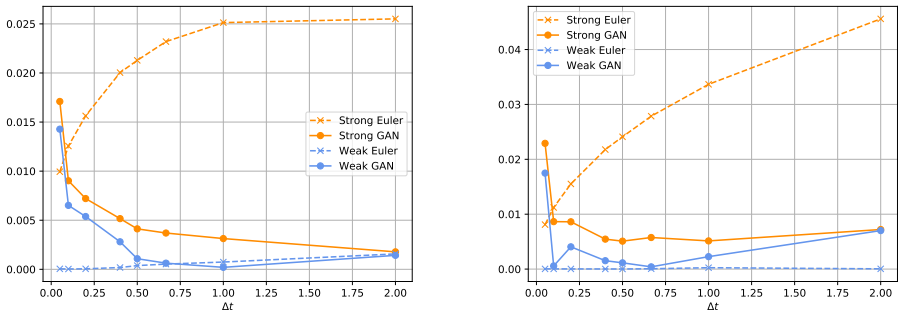
Figure 6.4: KS-statistic and Wasserstein distance on a test set of 100,000 samples during the training phase, both for the vanilla GAN and constrained GAN. The test condition was $(S_t, \Delta t) = (0.1, 1)$.

Summarising, both GANs are indeed able to approximate the distribution $F_{S_{t+\Delta t}|S_t}$ on a range of different parameters S_t and Δt, although the constrained GAN provides a more accurate approximation. Similar behaviour was found when repeating the experiments on the GBM problem and the case if the Feller condition is satisfied. Section 6.3 will explore the map Z to the GAN output in more detail, which may explain why the constrained GAN does not only guarantee a strong approximation, but also provides a more accurate approximation in distribution. We verified that the autocorrelation structure obtained with the vanilla GAN and constrained GAN is similar to that of a set of exactly sampled variates, which is required for constructing paths using our method. The outputs using both GAN architectures even correspond point by point in the plane $S_{t+\Delta t}, S_t$ of figure 6.3, which suggests that both GANs approximate the exact solution path-wise on the discretisation. In section 6.3, we will show that this does not hold in general for the vanilla GAN, but only for the constrained GAN, as shown in chapter 4.

6.2. Constructing Paths

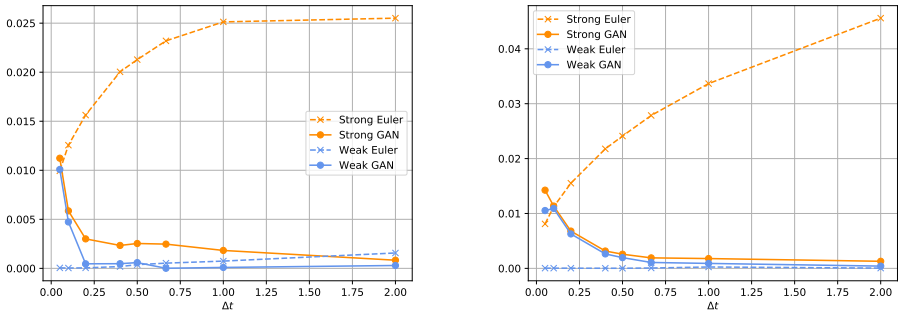
Synthetic paths are constructed with the GAN by iteratively sampling from $F_{S_{t+\Delta t}|S_t}$. We fix a time interval of interest $[0, T]$ and divide it into a discretisation $t_k = k\Delta t = kT/n$, for $k = \{0, 1, \ldots, n\}$.

First, paths constructed with the vanilla GAN and constrained GAN are compared on the same partition $\{t_k\}_{k=0}^n$, where we let n run from 40 to 1, or equivalently, letting Δt run from 0.05 to 2. This way, we compare a fine discretisation to an increasingly coarser one. We will compare the weak and strong errors of the synthetic paths at time $T = 2$ with those of the truncated Euler and Milstein schemes. Between every two time steps, we sample a corresponding normal increment Z using equation 6.1, which is used as input to the GAN and the Euler and Milstein schemes.
The weak and strong errors versus Δt are shown in figure 6.5 for the vanilla GAN and figure 6.6 for the constrained GAN. The truncated Milstein scheme did not achieve a lower weak or strong error than the truncated Euler scheme on this problem, which is why it is not shown in the figures. If the Feller condition was satisfied, it did provide a more accurate approximation, cf. the appendix. Two different starting points S_0 are chosen. If $S_0 = \bar{S}$, the truncated Euler scheme achieves a very low weak error, as the mean of each $\hat{S}_{t+\Delta t}$ equals 0. This is not the case if $S_0 \neq \bar{S}$, in which case the weak error is higher, as is indeed visible in figures 6.5a and 6.6a.



(a) $S_0 = 0.01$ (b) $S_0 = 0.1$

Figure 6.5: Vanilla GAN: weak and strong errors at time $T = 2$, compared with the truncated Euler scheme and exact scheme for various Δt and two different starting points S_0.



(a) $S_0 = 0.01$ (b) $S_0 = 0.1$

Figure 6.6: Constrained GAN: weak and strong errors at time $T = 2$, compared with the truncated Euler scheme and exact scheme for various Δt and two different starting points S_0.

Both GANs achieve a lower strong error than the truncated Euler scheme if $\Delta t \geq 0.1$, but the vanilla GAN has a considerably higher weak error than the constrained GAN on the same time steps. This can be explained by the fact that the constrained GAN provides a better approximation of $F_{S_{t+\Delta t}|S_t}$. This is in agreement with the result in figure 6.1, where the approximation of the conditional distribution at e.g. $\Delta t = 2$ is visibly better in case of the constrained GAN. The weak and strong errors of both GANs at the lower values of Δt are high compared to values at large Δt. This may seem counterintuitive, since the opposite is true for discrete-time schemes. However, as stressed in chapter 4, the GAN approximation is based on different principles than the Euler and Milstein schemes. Any weak error will arise from failure of the GAN to perfectly approximate $F_{S_{t+\Delta t}|S_t}$, while any strong error arises from failure to perfectly reconstruct the map Z to $S_{t+\Delta t} \mid S_t$.

Non-monotonicity in the dependence of the errors on Δt is explained by the fact that the GAN achieved a better approximation on some conditional inputs than others, as is visible in figure 6.1. It is not clear a priori on which conditional classes the GAN will converge best, although ideally it converges equally well for all conditional inputs. There are two reasons why the weak and strong errors on small time steps are relatively high. Firstly, the GAN is repeated more times at lower values of Δt, compounding any error that arises on a single time step. Secondly, the approximation of $F_{S_{t+\Delta t}|S_t}$ turned out to be less accurate on lower values of Δt. On low values of Δt, the variance of the distribution $F_{S_{t+\Delta t}|S_t}$ is low compared to high values of Δt. As explained in [88], the weight updates in a neural network are proportional to the variance of the input samples, which effectively 'underprioritises' samples with lower Δt during training and thus explains our observation. This could be resolved either by choosing the range of Δt less far apart in practical applications or by choosing a pre-processing technique that scales each training point differently for each Δt. The current pre-processing step scales each data point with $\bar{S}$ or involves the logreturns in case of GBM and if the Feller condition is satisfied. If the Feller condition is satisfied and on the GBM problem, similar results were found, which are included in the appendix.

6.2.1. Single-step GAN approximation

The previous experiment tested for which Δt the paths obtained with the GAN outperformed the truncated Euler and Milstein schemes, if they are defined on the same discretisation. This time, we will let the GAN approximate the strong solution at $T = 1$ using a single time step, while we construct paths with the Euler and Milstein schemes on a discretisation with $n = 10$ and $n = 40$ steps. This second experiment tests the accuracy of a single-step approximation with a GAN, compared to both discrete-time schemes on a finer discretisation.

Since the GAN approximates the solution using a single time step T, we need to ensure that the sample Z we provide as input to the GAN corresponds to Brownian motion increments on the finer discretisation $\{t_k\}_{k=0}^n$. Without this step, we could not compare the output path-wise with the solution on the finer discretisation. To this end, we use lemma 4.13 from chapter 4, as follows:

$$Z = \frac{\sqrt{\Delta t}}{\sqrt{T}} \sum_{k=1}^{n} U_k, \tag{6.2}$$

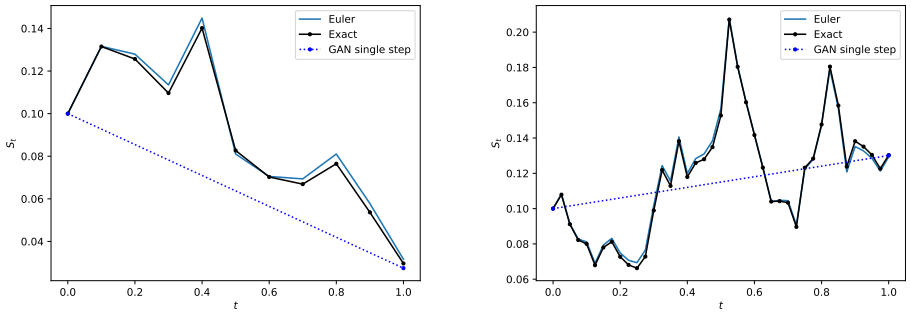
where $U_k \sim N(0, 1)$ are the random variates used on the finer discretisation $\{t_k\}_{k=0}^n$ by the truncated Euler and Milstein scheme. These U_k are in turn found using equation 6.1. This way, we can compare path-wise approximation on both the coarse discretisation (i.e. single step) with the finer Euler and Milstein discretisation and compute the weak and strong error at T for both of them.

The results for the truncated Milstein scheme have again been left out, as it did not achieve lower weak and strong errors on the time steps under consideration. If the Feller condition is satisfied, it does outperform the truncated Euler scheme in strong error, which is shown in the appendix. We will only consider the constrained GAN in this experiment, as we have already found that it outperforms the vanilla GAN in weak and strong error in the previous section. 100,000 paths were generated this way. One of the paths obtained on both experiments is plotted in figure A.5. Note how both the truncated Euler scheme and GAN approximate the exact simulation scheme path-wise at T.

Table 6.2: Weak and strong error at $T = 1$ of the GAN on a single time step and the truncated Euler scheme after 10 or 40 steps.

	Euler	GAN	Euler	GAN
	$n = 10$		$n = 40$	
Weak error	6.0E-05	6.1E-04	5.0E-07	5.9E-04
Strong error	7.8E-03	1.4E-03	4.0E-03	1.5E-03

In table 6.2, the weak and strong errors over all 100,000 paths at final time are shown for both choices of the amount of steps. Since the Euler scheme is of strong order $\frac{1}{2}$, we indeed see that the strong error in the Euler scheme is a factor two smaller, while the strong error in the GAN remains unchanged. The weak error in the Euler scheme is much lower than that of the GAN. Since $S_0 = \bar{S}$, the Euler scheme updates are highly accurate in mean at each step, while the GAN does not have the



(a) 10 steps

(b) 40 steps

Figure 6.7: Example of paths obtained with the single-step approximation by the constrained GAN and multiple steps with the truncated Euler scheme and exact scheme, setting $S_0 = 0.1$.

mean reversion 'built-in'. This is even more pronounced if 40 time steps are used, making the mean reverting behaviour even more accurate. If we repeat the experiment with $S_0 = 0.01$ on 40 time steps for example, the weak error by the Euler scheme was found to be 3.3×10^{-5}, two orders of magnitude higher, but still an order lower than that of the GAN. Thus, the GAN shown here is favourable only if the strong error is more important than the weak error. Similar results were found on the GBM problem and if the Feller condition was satisfied. In particular, on the GBM problem, the GAN even outperformed the Milstein scheme at 10 steps and the Euler scheme at 40 steps in terms of the strong error, while in weak error, it did not do better at 5 steps or more. Further results are included in the appendix. However, this problem was far more challenging than the experiment of figures 6.5 and 6.6 and less in the spirit of large time steps, although the weak error should still be improved. The results on single-step approximation show that lemma 4.13 can be used to connect a fine grid to a coarse grid and that the GAN output still provides a path-wise approximation. The fact that the GAN outperforms the Euler scheme in strong error in figures 6.5 and 6.6 serves as a proof of concept of the technique proposed in this thesis. In the next section, we will see what enables the constrained GAN to find this approximation and how this is not the case for the vanilla GAN in general. We will explore the map from Z to the generator output in more detail and verify the theory behind the constrained GAN, introduced in chapter 4.

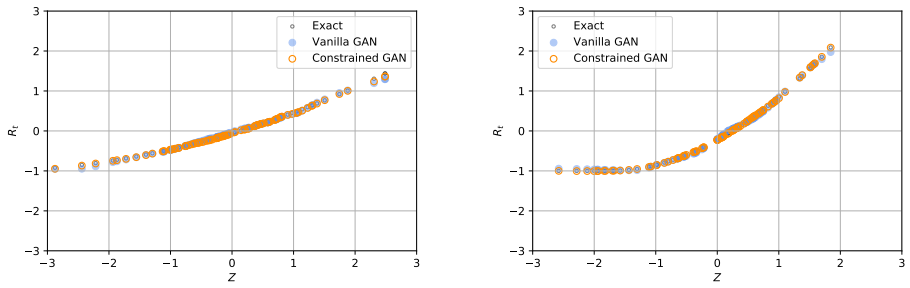
6.3. Analysis of the Constrained GAN

In this section we investigate what distinguishes the constrained GAN from the vanilla GAN. Recall that the vanilla GAN provides the approximation $\hat{S}_{t+\Delta t} \mid \hat{S}_t \overset{d}{\approx} S_{t+\Delta t} \mid S_t$, which does not necessarily equal the exact strong solution path-wise, i.e. having equality of the paths P-almost surely. In our setting, both the vanilla GAN and constrained GAN are adapted to $\mathcal{G}_k = \sigma(\{Z_k\})$, since both receive the input Z_k at each time step t_k. What distinguishes them is the map they represent from Z to the approximation $\hat{S}_{t+\Delta t} \mid \hat{S}_t$. As we will see, a large range of maps could correspond to an approximation of $F_{S_{t+\Delta t}|S_t}$, while these maps themselves may be very different than the map corresponding to the discrete strong solution. The discrete strong solution corresponds to a unique map with input Z, which the constrained GAN is trained to find, while the vanilla GAN is trained to find any map from Z that approximates $F_{S_{t+\Delta t}|S_t}$ in distribution.

By plotting the output of the GAN against the input $Z \sim N(0, 1)$, given some S_t and Δt, we can visualise the map that the GAN represents. We will do this both for the vanilla GAN and the constrained GAN. In figure 6.8, the process R_t (i.e. $(S_{t+\Delta t} \mid S_t)/\bar{S} - 1$) obtained with exact simulation is plotted against Z, which is found by using equation 6.1. The GAN output is shown after providing this same Z as input. This tests if the map corresponding to the exact simulation scheme coincides with the map that the GAN has learned.

In this case, both the vanilla GAN and constrained GAN output are in close agreement with the exact variates, which indicates that both find an approximation to the discrete strong solution. This is in agreement with the fact that the GAN output in figure 6.3 largely overlaps with the exactly sampled points for both GANs. It also agrees with the fact that the strong error in figures 6.5 and 6.6 is lower

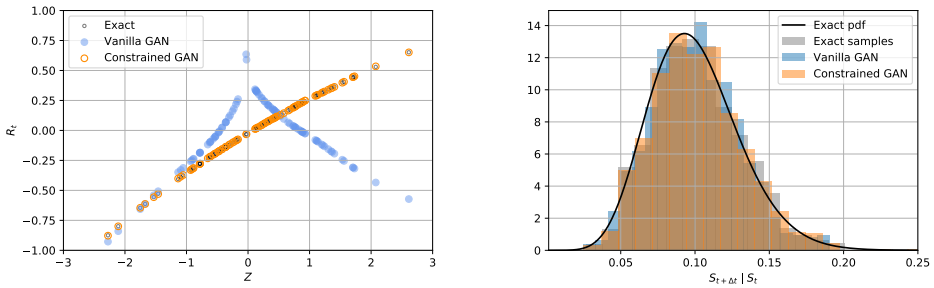
than that of the truncated Euler scheme. If this was not the case, the GAN would not approximate the discrete strong solution path-wise. In this case, the only difference between both GANs is that the approximation of the map in the R_t, Z-plane is more accurate for the constrained GAN. The weak approximation found the vanilla GAN coincides with the strong solution in this example. However, this was found only to be a special case for the vanilla GAN.



(a) $\Delta t = 0.25$ (b) $\Delta t = 1.0$

Figure 6.8: CIR-process if the Feller condition is not satisfied: vanilla GAN and constrained GAN in the R_t, Z-plane, along with exact samples.

For the constrained GAN, a path-wise approximation was found on all settings on which it was tested. However, this was not the case for the vanilla GAN. To illustrate this, another example of the R_t, Z-plane is shown in figure 6.9, which was found after training a vanilla GAN and constrained GAN on the CIR process with the Feller condition satisfied. It reveals why approximating $F_{S_{t+\Delta t}|S_t}$ does not imply approximating the strong solution path-wise on the discretisation, as it does not coincide with the map in the R_t, Z-plane by the exactly sampled variates. If we created a scatter plot similar to figure 6.3, the point-clouds would have a similar shape, but the points would not overlap. The strong error would be much higher than that of the truncated Euler scheme if we created paths with this GAN and the single step-approximation in figure A.5 would not coincide with the exact solution at time T. Examples like this were numerous when training vanilla GANs on the GBM problem and if the Feller condition was satisfied. These examples are included in the appendix. In all cases, the constrained GAN did manage to provide a path-wise approximation.



(a) R_t, Z-plane. (b) Distribution of 1,000 samples.

Figure 6.9: Example of an approximation in distribution by the vanilla GAN, which is not a path-wise approximation. A histogram is shown of the samples corresponding to figure 6.9a. The constrained GAN does represent a path-wise approximation. In this example, $(S_t, \Delta t) = (0.1, 1)$

One may now wonder what enables the constrained GAN to provide the strong approximation, while the vanilla GAN frequently fails to do so in the general case. As explained in chapter 4, the discriminator is 'informed' with the variate Z that corresponds to the sample $S_{t+\Delta t} \mid S_t$ it receives. The discriminator input is given by $(R_t, S_t, \Delta t, Z)$ instead of only $(R_t, S_t, \Delta t)$. Recall that $(S_{t+\Delta t} \mid S_t)$ and Z - and therefore R_t and Z - are coupled using equation 6.1. Had we chosen any Z, the discriminator would only receive i.i.d. noise and would not benefit from the additional input. We can study how the discriminator enforces the correct map from Z to R_t by studying its output. The discriminator input is defined on a 4D plane, spanned by $(R_t, S_t, \Delta t, Z)$. If we fix S_t and Δt, we

obtain the R_t, Z-plane. We can then plot the discriminator output on this plane. Thus, we could 'overlay' the output of the trained discriminator on the result of figure 6.8. The result for the choice $(S_t, \Delta t) = (0.1, 1)$ is shown in figure 6.10.
In the results so far, we observed how the approximation in distribution itself was also more accurate if the constrained GAN was used.

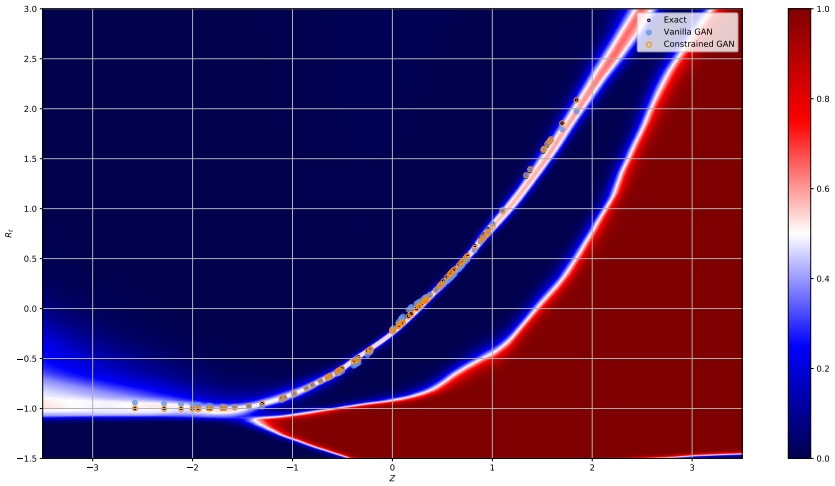


Figure 6.10: Output of the constrained GAN discriminator corresponding to figure 6.8 on the R_t, Z-plane, along with the samples from figure 6.8. $S_t = 0.1$ and $\Delta t = 1.0$.

Recall that the discriminator output is a number between 0 and 1, representing the 'score' that the discriminator assigns to each input it receives. The closer the value is to 1, the more the discriminator considers the input as a sample from the 'real', i.e. exact dataset. The contrary holds for scores close to 0. Thus, if the generator output resembles the exact samples well, the discriminator should assign a score close to 0.5, i.e. it cannot distinguish between the real and generated samples. In figure 6.10, a 'band' of values close to 0.5 is clearly visible. The exact variates and constrained GAN output lie within this band, i.e. the discriminator is unable to distinguish between them. This band implicitly contains all the information about the map from Z to R_t given S_t and Δt. This discriminator confidence surface, along with the constrained generator output, resembles the minimax solution that the generator and discriminator find during the training phase. Note how the constrained GAN discriminator would rule out a map by the vanilla GAN like the one shown in figure 6.9a, which would lie outside the band. The discriminator puts a 'constraint' on which maps by the generator it allows, hence the name of the constrained GAN.
This property also explains why the constrained GAN converges faster and produces more accurate approximations of the conditional distribution, as the discriminator has an additional dimension to guide the generator to find the correct map from Z to the exact solution. It rules out maps that are 'inefficient', such as the one shown in figure 6.9 by the vanilla GAN and directs the generator to the unique map corresponding to $F_Z^{-1} \circ F_{S_{t+\Delta t}|S_t}$. In figure 6.10, some of the output samples of the vanilla GAN lie outside the band. However, the vanilla GAN discriminator does not observe this band, making it harder for the vanilla GAN generator to further improve than the constrained GAN generator.
In all results obtained using the constrained GAN, a similar band was visible. A band remains visible for any choice of S_t and Δt. The discriminator confidence has been included in the appendix on several more examples, for GBM and if the Feller condition is satisfied. The results confirm the ideas proposed in chapter 4, where we presented the constrained GAN as an approximation to the unique map from Z to $S_{t+\Delta t} \mid S_t$.
Note that in figure 6.10, we viewed a 2D cross-section of the 4D input hyper plane $(R_t, S_t, \Delta t, Z)$ that defines the discriminator input. In reality, the band visible in the discriminator output of figure 6.10 is

a 4D region in the space $(R_t, S_t, \Delta t, Z)$. The shape of the band changes for every combination of $S_t, \Delta t$, which allows the GAN to learn the distribution $F_{S_{t+\Delta t}|S_t}$ for a large range of choices Δt and S_t.

Chapter Conclusion

The vanilla GAN and constrained GAN both provide an approximation to the conditional distribution $F_{S_{t+\Delta t}|S_t}$, however, in all the results we found that the constrained GAN provides a more accurate approximation. The constrained GAN provided a path-wise approximation in all results, while the constrained GAN only did so if the Feller condition was not satisfied. If paths are constructed on a discretisation with large time steps, the strong errors of the constrained GAN are lower than those of the Euler and Milstein schemes. On a single-step approximation, the GAN output outperformed discrete-time schemes in terms of the strong error, but not the weak error on all problems the GAN was trained on. The GAN approximation of the conditional distribution should be further improved to improve the weak error. Overall, the results validated the theory proposed in chapter 4 and serve as a proof of concept for the constrained GAN.

7

Discussion and Recommendations

This chapter reflects on the contributions in this thesis and presents an outlook for future work and applications. The design choices in this work are discussed, along with the limitations and generality of the results presented.

7.1. Architecture and training process

The GANs used to generate the results in chapter 6 were sufficient for testing the ideas shown in chapter 4 and for comparing the vanilla GAN to the constrained GAN. We found that the common problem of mode collapse did not occur in our results. This could be due to the fact that the conditional distributions are unimodal, while the conditional GAN is able to make sharp distinctions between the different conditional inputs, cf. for example figure 6.1. The vanishing gradient problem was resolved by using the modified generator loss function proposed in [75], explained in section 3.3.2. In case of the vanilla GAN, we also found that the generator gradients did not 'explode' as sometimes occurs in practice, cf. section 3.3.

When training the constrained GAN, the generator gradients increased sharply on some of the experiments, although this did not have a noticeable effect on the KS-statistic and Wasserstein distance during training. Furthermore, both the vanilla GAN and constrained GAN oscillated around the target distribution during training, arising from the adversarial nature of the training process. The training phase was stabilised by defining a learning rate schedule, which divided the learning rate η of the Adam optimiser by a constant factor 1.05 every 500 iterations. After studying the code used by the authors of the related work in [11], which is publicly available, it was found that they also use this technique.

Although the architecture used to generate the results in chapter 6 was sufficient for comparing the vanilla GAN and constrained GAN on our problem setting, other GAN variants may be explored to test if they provide more accurate approximations of the conditional distribution, by further enhancing the training process. Additionally, increasing the network capacity and size of the training set could further improve the approximation of the conditional target distribution, at the cost of higher computational requirements.

7.2. Data pre-processing

The use of logreturns and scaling with $\bar{S}$ if the Feller condition was not satisfied was found to substantially improve the approximation of the conditional distribution. However, we found that the approximation of the conditional distribution on lower values of Δt was of lower quality than on higher values of Δt. We attributed this to the fact that inputs with low variance are 'underprioritised' compared to inputs with high variance, cf. [88]. An alternative scaling technique may be considered, which applies a different analogue of standardisation for each condition Δt. However, one should ensure that the GAN is still able to distinguish between samples based on their conditional input, as it should be able infer the conditional mean and variance of the output from the training set. The transformation should also not include information about future time steps, violating the adaptedness requirement.

7.3. Comparison with stochastic collocation

Alternative methods are available to sample from the conditional distribution $F_{S_{t+\Delta t}|S_t}$, as briefly introduced in chapter 1. The stochastic collocation Monte Carlo method (SCMC) [21] allows efficient sampling from arbitrary conditional distributions. However, for each change in conditional parameters, model parameters and time step, one would require a new collocation fit. A new method was proposed very recently that overcomes this problem. The '7-League scheme', introduced in [22], combines the SCMC scheme with a neural network. It uses a neural network to learn the collocation points, conditional on the time t, time step Δt, previous realisation S_t and the SDE parameters. Since a standard feed-forward neural network is used, it eliminates the difficulties encountered when training GANs and gives rise to a better interpretable training process. During inference, our method is simpler in setup, as it only requires a trained generator and standard normal variate, while the 7-League scheme requires additional computations for constructing the collocation fit and interpolation, in addition to inference with the neural network to find the collocation points. Additionally, when constructing the training set for our method, CDF inversions can be avoided if a high-quality discrete-time approximation is used as the training set. In this case, the random increments Z needed for the constrained GAN become directly available, as shown in section 4.4.1. However, the authors of [22] propose a way to 'compress' the set of inferred collocation points with the neural network into a matrix. As this matrix also includes the time dimension, it contains all the information needed for constructing paths. Interpolation between the values in this matrix allows for an efficient inference stage, while our method always requires the neural network during inference at each time step. It should be explored if similar improvements to the inference stage can be applied to make sampling with our scheme more efficient.

7.4. Generality and extensions

The choice of SDEs under consideration allowed us to use the analytical conditional distribution $F_{S_{t+\Delta t}|S_t}$, making the link between each $S_{t+\Delta t} \mid S_t$ and normal increment Z available explicitly, using equation 6.1. This allowed convenient construction of a training set for the constrained GAN. However, as shown in section 4.4, any knowledge of the distribution $F_{S_{t+\Delta t}|S_t}$ is not required for the constrained GAN in practice. The most straightforward extension is to create a discrete-time approximation on a very fine discretisation and to reuse the same normal increments to find the input Z to the constrained GAN on large time steps. This still allows the GAN to find the map Z to $S_{t+\Delta t} \mid S_t$ in the space $((S_{t+\Delta t} \mid S_t), S_t, \Delta t, Z)$. The training procedure of the constrained GAN would further be identical, while the only consideration is to provide a rich enough range of S_t and Δt to let the GAN learn the conditional distribution for sufficiently many 'previous values' S_t and time steps.

Secondly, GBM and the CIR process are Itō diffusions, not generalised Itō diffusions that make up general Itō SDEs. This allowed us to leave out dependence of the conditional distribution on the current time t, as the stochastic integral only depended on the time step Δt and the process S_t on the time interval. In general, the constrained GAN should learn the dependence of $F_{S_{t+\Delta}|S_t}$ on time and should thus receive the additional input t. The discriminator would receive $((S_{t+\Delta t} \mid S_t), S_t, t, \Delta t, Z)$, i.e. it would take values on a five-dimensional space instead of a four-dimensional space. This additional dimension makes generalised Itō diffusions more challenging for the GAN to approximate. However, the constrained GAN algorithm would remain unchanged.

Furthermore, the GAN was trained on datasets of GBM and the CIR process using fixed SDE parameters $\mu, \sigma, \kappa, \bar{S}$ and γ. The architecture should be further extended to learn the entire family of solutions of the SDE, as is done in [22]. In this case, the generator would be trained to find a map in the space $((S_{t+\Delta t} \mid S_t), S_t, t, \Delta t, Z, C)$, with C the set of SDE parameters. The condition vector C would be added as conditional input to the generator and discriminator networks. The setup proposed in this work would further remain unchanged, although a larger training set and additional network capacity may be required. This would be particularly relevant for a range of volatility parameters γ of the CIR process, which gives rise to highly non-linear changes in the conditional distribution. In future work, the constrained GAN should be extended to this case of learning families of SDEs, which would allow a single GAN to provide synthetic paths for any desired range of parameters.

This work has been set in 1D, with a single Brownian motion, which simplified the analysis of the GAN output and the construction of a training and test set. However, the use of methods like neural networks becomes more relevant in the high-dimensional setting, as alternative techniques become less tractable due to the curse of dimensionality. The technique proposed in section 4.4.1 automatically

extends to higher dimensions if the Brownian motions are uncorrelated. In this case, the dimension of the input to the generator, Z, increases, but each entry of this vector $\boldsymbol{Z}$ can be found in the same way as in the 1D case, using section 4.4.1. If the Brownian motions are correlated, one could write them as a system of uncorrelated Brownian motions, using Cholesky decomposition of their matrix C, as shown e.g. in [107]. This gives rise to a system of equations based on a triangular matrix, from which the corresponding vector of i.i.d. $N(0,1)$ variates $\boldsymbol{Z}$ can still can be found and included in the training set for the constrained GAN. Future work could extend to systems of SDEs in this way, such as the Heston model.

Finally, in this work we considered Itō SDEs, which allowed us to use the Markov property. However, more general classes of SDEs could be defined, e.g. those based on fractional Brownian motions [108], which have non-Markovian dynamics. This means that the conditional distribution would no longer be dependent on $S_t, t, \Delta t$, but the entire history of the path up to time t. In this case, the method should be further extended by providing the history of the generated process $\{\hat{S}_t\}$ and the sequence $\{Z_k\}$ up to the current time. An architecture similar to those proposed in [5] or [11] could be used, conditioning on a 'rolling window' of previous values of both processes, but this time in the spirit of the constrained GAN, where $\{Z_k\}$ is linked to the process $\{S_t\}$ during training.

7.5. Outlook

Since the constrained GAN was found to outperform the vanilla GAN under identical conditions, the architecture may be used to improve current applications of GANs for sampling from inverse distributions, outside of the realm of SDEs. The map between the prior Z and some conditional target distribution X may be obtained through inexpensive inversion with F_Z^{-1} and the empirical conditional distribution of the training set.

In the context of SDEs, the constrained GAN proposed in this work has been shown to approximate the strong solution on a discretisation. It could serve as a starting point for future applications of GANs on SDE problems. However, the method should be extended to the full parameter sets of the SDEs under consideration and systems of SDEs. Successful extension would lead to efficient sampling of synthetic paths, with potential application on pricing path-dependent derivatives, such as Bermudan options (cf. [109]) and similar applications on a time discretisation that require Monte Carlo simulation of asset paths.

8

Conclusion

In this thesis, a number of novel insights regarding GANs has been presented, in the context of financial SDEs. A new GAN algorithm was proposed, the constrained GAN, which is based on the conditional GAN architecture. It informs the discriminator with the map between the generator input and the target distribution of interest. This led to a more accurate approximation of the conditional target distribution and a more efficient training process on the SDE problems in this work. The link between prior and target distribution was obtained by CDF inversions of the inexpensive prior on the target distribution. An extension was proposed that eliminates the need for CDF inversions on SDE problems by using an accurate discrete-time approximation as training set. It was shown how the constrained GAN architecture allowed for strong approximation of two one-dimensional Itō SDEs on a time discretisation. This is not possible with traditional GANs, as they only learn an approximation in distribution. The constrained GAN links the prior input of the generator to the Brownian motion increment between two time steps, which enables the path-wise approximation. It was shown that the constrained GAN discriminator rules out maps by the generator that do not coincide with the conditional inverse distribution of the exact strong solution. This revealed how the generator learns the map from the prior input to the conditional target distribution. Paths obtained with the constrained GAN outperformed discrete-time schemes on large time steps in terms of the strong error. However, the constrained GAN was trained on one-dimensional SDE problems with fixed parameters. In order to extend to general applications and state-of-the-art alternatives to GANs, the method should be extended to general SDE parameter sets and systems of SDEs, such as the Heston model. The ideas presented in this work serve as a starting point for such extensions to path-wise approximation of general Itō SDEs. Successful extension could give rise to applications in finance, such as the pricing of path-dependent securities and similar applications that require strong approximation with Monte Carlo simulation on large time steps.

Bibliography

[1] Bernt Oksendal. Stochastic differential equations: an introduction with applications. Springer Science & Business Media, 2013.

[2] Peter Eris Kloeden, Eckhard Platen, and Henri Schurz. Numerical solution of SDE through computer experiments. Springer Science & Business Media, 2012.

[3] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 2020.

[4] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561, 2017.

[5] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: deep generation of financial time series. Quantitative Finance, 0(0):1–22, 2020.

[6] Makoto Naruse, Takashi Matsubara, Nicolas Chauvet, Kazutaka Kanno, Tianyu Yang, and Atsushi Uchida. Generative adversarial network based on chaotic time series. Scientific reports, 9(1):1–9, 2019.

[7] Alireza Koochali, Peter Schichtel, Andreas Dengel, and Sheraz Ahmed. Probabilistic forecasting of sensory data with generative adversarial networks–forgan. IEEE Access, 7:63868–63880, 2019.

[8] Jin-Long Wu, Karthik Kashinath, Adrian Albert, Dragos Chirila, Prabhat, and Heng Xiao. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. Journal of Computational Physics, 406:109209, 2020.

[9] Mélanie Ducoffe, Ilyass Haloui, Jayant Sen Gupta, and ISAE Supaero. Anomaly detection on time series with wasserstein gan applied to phm. PHM Applications of Deep Learning and Emerging Analytics. International Journal of Prognostics and Health Management Reviewed (Special Issue), 2019.

[10] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. arXiv preprint arXiv:1811.02033, 2018.

[11] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional sig-wasserstein gans for time series generation, 2020.

[12] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. ACM Transactions on Graphics (TOG), 37(4):1–15, 2018.

[13] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks, 9(5):987–1000, 1998.

[14] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34):8505–8510, 2018.

[15] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of Computational Physics, 375:1339 – 1364, 2018.

[16] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing, 317:28 – 41, 2018.

[17] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. arxiv. arXiv preprint arXiv:1711.10561, 2017.

[18] Panos Stinis, Tobias Hagge, Alexandre M. Tartakovsky, and Enoch Yeung. Enforcing constraints for interpolation and extrapolation in generative adversarial networks. Journal of Computational Physics, 397:108844, 2019.

[19] Gabriele Abbati, Philippe Wenk, Stefan Bauer, Michael A Osborne, Andreas Krause, and Bernhard Schölkopf. Ares and mars-adversarial and mmd-minimizing regression for sdes. arXiv preprint arXiv:1902.08480, 2019.

[20] Mark Broadie and Özgür Kaya. Exact simulation of stochastic volatility and other affine jump diffusion processes. Operations research, 54(2):217–231, 2006.

[21] Lech A Grzelak, Jeroen AS Witteveen, Maria Suarez-Taboada, and Cornelis W Oosterlee. The stochastic collocation monte carlo sampler: highly efficient sampling from 'expensive'distributions. Quantitative Finance, 19(2):339–356, 2019.

[22] Shuaiqiang Liu, Lech A. Grzelak, and Cornelis W. Oosterlee. The seven-league scheme: Deep learning for large time step monte carlo simulations of stochastic differential equations, 2020.

[23] Rao Fu, Jie Chen, Shutian Zeng, Yiping Zhuang, and Agus Sudjianto. Time series simulation by conditional generative adversarial net. arXiv preprint arXiv:1904.11419, 2019.

[24] Achim Klenke. Probability theory. Universitext. Springer-Verlag London Ltd., London, 2008.

[25] Eckhard Platen and Nicola Bruti-Liberati. Numerical solution of stochastic differential equations with jumps in finance, volume 64. Springer Science & Business Media, 2010.

[26] Uwe Hassler et al. Stochastic processes and calculus. Springer Texts in Business and Economics, 2016.

[27] Gisiro Maruyama. Continuous markov processes and stochastic equations. Rendiconti del Circolo Matematico di Palermo, 4(1):48, 1955.

[28] G. N. Milśhtein. Approximate integration of stochastic differential equations. Theory Probab. Appl., 19:3, page 557–562, 1975.

[29] Jean Jacod and Philip Protter. Probability essentials. Springer Science & Business Media, 2012.

[30] Jean-François Le Gall. Brownian motion, martingales, and stochastic calculus, volume 274. Springer, 2016.

[31] John C Cox, Jonathan E Ingersoll Jr, and Stephen A Ross. A theory of the term structure of interest rates. In Theory of valuation, pages 129–164. World Scientific, 2005.

[32] Hansjörg Albrecher, Philipp Mayer, Wim Schoutens, and Jurgen Tistaert. The little heston trap. Wilmott, (1):83–92, 2007.

[33] Steven Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. Review of Financial Studies, 6:327–343, 1993.

[34] PS Hagan, D Kumar, AS Lesniewski, and DE Woodward. Managing smile risk, wilmott magazine, september, 2002.

[35] Christopher M Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[36] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[37] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4(2):251 – 257, 1991.

[38] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In Advances in neural information processing systems, pages 6231–6239, 2017.

[39] Patrick Kidger and Terry Lyons. Universal approximation with deep narrow networks. In Conference on Learning Theory, pages 2306–2327, 2020.

[40] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[43] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[44] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.

[45] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. Neural networks for machine learning. Coursera, video lectures, 264(1), 2012.

[46] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010, pages 177–186. Springer, 2010.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. nature, 323(6088):533–536, 1986.

[48] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. Journal of Machine Learning Research, 18(153):1–43, 2018.

[49] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[50] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In Advances in neural information processing systems, pages 6626–6637, 2017.

[51] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning (ICML-10), pages 807–814, 2010.

[52] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In Proc. icml, volume 30, page 3, 2013.

[53] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434, 2015.

[54] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2):157–166, 1994.

[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, pages 8024–8035, 2019.

[56] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In 2009 IEEE 12th international conference on computer vision, pages 2146–2153. IEEE, 2009.

[57] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In Shape, contour and grouping in computer vision, pages 319–345. Springer, 1999.

[58] C. L. Giles, G. M. Kuhn, and R. J. Williams. Dynamic recurrent neural networks: Theory and applications. IEEE Transactions on Neural Networks, 5(2):153–156, 1994.

[59] Hava T Siegelmann. Theoretical foundations of recurrent neural networks. 1993.

[60] Tony Robinson, Mike Hochberg, and Steve Renals. The Use of Recurrent Neural Networks in Continuous Speech Recognition, pages 233–258. Springer US, Boston, MA, 1996.

[61] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 6645–6649, 2013.

[62] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, pages 3104–3112, 2014.

[63] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

[64] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In The IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[65] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. arXiv preprint arXiv:1605.05396, 2016.

[66] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. How generative adversarial networks and their variants work: An overview. ACM Computing Surveys (CSUR), 52(1):1–43, 2019.

[67] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. arXiv preprint arXiv:1706.02633, 2017.

[68] Sang-gil Lee, Uiwon Hwang, Seonwoo Min, and Sungroh Yoon. Polyphonic music generation with sequence generative adversarial networks. arXiv preprint arXiv:1710.11418, 2017.

[69] Gérard Biau, Benoît Cadre, Maxime Sangnier, and Ugo Tanielian. Some theoretical properties of gans. arXiv preprint arXiv:1803.07819, 2018.

[70] Hans Peters. Game theory: a multi-leveled approach. Springer, 2015.

[71] J v Neumann. Zur theorie der gesellschaftsspiele. Mathematische annalen, 100(1):295–320, 1928.

[72] Jianhua Lin. Divergence measures based on the shannon entropy. IEEE Transactions on Information theory, 37(1):145–151, 1991.

[73] S. Kullback and R. A. Leibler. On information and sufficiency. The Annals of Mathematical Statistics, 22(1):79–86, 1951.

[74] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. arXiv preprint arXiv:1611.02163, 2016.

[75] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862, 2017.

[76] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. arXiv preprint arXiv:1701.07875, 2017.

[77] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of gans. arXiv preprint arXiv:1705.07215, 2017.

[78] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In Advances in neural information processing systems, pages 5767–5777, 2017.

[79] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784, 2014.

[80] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks. IEEE Transactions on Evolutionary Computation, 23(6):921–934, 2019.

[81] Hisham Husain, Richard Nock, and Robert C Williamson. A primal-dual link between gans and autoencoders. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 415–424. Curran Associates, Inc., 2019.

[82] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In Advances in neural information processing systems, pages 1486–1494, 2015.

[83] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In Advances in Neural Information Processing Systems, pages 2203–2213, 2017.

[84] Mario Lučić, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. In Advances in Neural Information Processing Systems, 2018.

[85] Leif BG Andersen. Efficient simulation of the heston stochastic volatility model. Available at SSRN 946405, 2007.

[86] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[87] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. arXiv preprint arXiv:1805.02242, 2018.

[88] Yann LeCun, Leon Bottou, Genevieve B Orr, Klaus-Robert Müller, et al. Neural networks: Tricks of the trade. Springer Lecture Notes in Computer Sciences, 1524(5-50):6, 1998.

[89] Steven E Shreve. Stochastic calculus for finance II: Continuous-time models, volume 11. Springer Science & Business Media, 2004.

[90] Chantal Labbé, Bruno Rémillard, and Jean-François Renaud. A simple discretization scheme for nonnegative diffusion processes, with applications to option pricing. arXiv preprint arXiv:1011.3247, 2010.

[91] Mario Hefter and André Herzwurm. Strong convergence rates for cox–ingersoll–ross processes—full parameter range. Journal of Mathematical Analysis and Applications, 459(2):1079–1101, 2018.

[92] Larry Wasserman. All of nonparametric statistics. Springer Science & Business Media, 2006.

[93] Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, and Ludolf Erwin Meester. A Modern Introduction to Probability and Statistics: Understanding why and how. Springer Science & Business Media, 2005.

[94] Richard Simard, Pierre L'Ecuyer, et al. Computing the two-sided kolmogorov-smirnov distribution. Journal of Statistical Software, 39(11):1–18, 2011.

[95] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272, 2020.

[96] William Feller. On the Kolmogorov–Smirnov Limit Theorems for Empirical Distributions, pages 735–749. Springer International Publishing, Cham, 2015.

[97] Aaditya Ramdas, Nicolás García Trillos, and Marco Cuturi. On wasserstein two-sample testing and related families of nonparametric tests. Entropy, 19(2):47, 2017.

[98] Mark Rowland, Jiri Hron, Yunhao Tang, Krzysztof Choromanski, Tamas Sarlos, and Adrian Weller. Orthogonal estimation of wasserstein distances. arXiv preprint arXiv:1903.03784, 2019.

[99] Marc G. Bellemare, Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Rémi Munos. The cramer distance as a solution to biased wasserstein gradients, 2017.

[100] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. Foundations and Trends® in Machine Learning, 12(4):307–392, 2019.

[101] Bharath K Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Bernhard Schölkopf, and Gert RG Lanckriet. Non-parametric estimation of integral probability metrics. In 2010 IEEE International Symposium on Information Theory, pages 1428–1432. IEEE, 2010.

[102] Friedrich Liese and Igor Vajda. On divergences and informations in statistics and information theory. IEEE Transactions on Information Theory, 52(10):4394–4412, 2006.

[103] Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. Divergence estimation of continuous distributions based on data-dependent partitions. IEEE Transactions on Information Theory, 51(9):3064–3074, 2005.

[104] Soheil Kolouri, Gustavo K Rohde, and Heiko Hoffmann. Sliced wasserstein distance for learning gaussian mixture models. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3427–3436, 2018.

[105] Cédric Villani. Optimal transport: old and new, volume 338. Springer Science & Business Media, 2008.

[106] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In Advances in neural information processing systems, pages 513–520, 2007.

[107] Cornelis W Oosterlee and Lech A Grzelak. Mathematical Modeling and Computation in Finance. World Scientific, 2019.

[108] Benoit B Mandelbrot and John W Van Ness. Fractional brownian motions, fractional noises and applications. SIAM review, 10(4):422–437, 1968.

[109] Desmond J Higham. An introduction to financial option valuation: mathematics, stochastics and computation, volume 13. Cambridge University Press, 2004.

A

Appendix

A.1. Network Architecture

The architectures of the feed-forward neural networks of the generator and discriminator are shown below. In the vanilla unconditional architecture, $c = 0$, while in the conditional GAN architecture, c equals the amount of conditional parameters. If the discriminator is informed with Z, i.e. the constrained GAN, the discriminator input is further increased by 1 for the input Z. The batch size was set to $1,000$. The GAN was trained for a fixed amount of 200 epochs. An additional learning rate schedule was created to stabilise GAN training. The learning rate of the generator was decreased by a factor $n_{\text{LR}} = 1.05$ every $n_{\text{LR}} = 500$ iterations.

Table A.1: Generator

Optimiser	Adam $\eta = 1 \times 10^{-4}, \beta_1 = 0.5, \beta_2 = 0.999$	
Layer	Nodes	Activation
Input layer	$1+c$	LeakyReLU, negative slope=0.1
Hidden 1	200	LeakyReLU, negative slope=0.1
Hidden 2	200	LeakyReLU, negative slope=0.1
Hidden 3	200	LeakyReLU, negative slope=0.1
Hidden 4	200	LeakyReLU, negative slope=0.1
Output layer	1	None

Table A.2: Discriminator

Optimiser	Adam $\eta = 5 \times 10^{-4}, \beta_1 = 0.5, \beta_2 = 0.999$	
Layer	Neurons	Activation
Input layer	$1+c$	LeakyReLU, negative slope=0.1
Hidden 1	200	LeakyReLU, negative slope=0.1
Hidden 2	200	LeakyReLU, negative slope=0.1
Hidden 3	200	LeakyReLU, negative slope=0.1
Hidden 4	200	LeakyReLU, negative slope=0.1
Output layer	1	Sigmoid

The learning rate for the discriminator was set to $5\times$ that of the generator learning rate, which was found to lead to faster convergence in first several epochs and better approximation of the optimal discriminator from theorem 3.2.

A.2. Discriminator Output

The discriminator output is shown for the GANs used to create the results for GBM and the CIR process if the Feller condition was satisfied. In both cases, one can see how the discriminator has identified the map from Z to a return R_t by the 'band' that is visible.

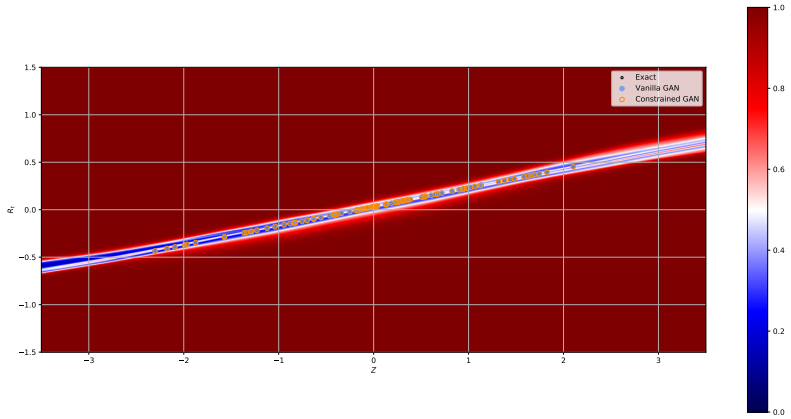


Figure A.1: Example for GBM: output of the constrained GAN discriminator on the R_t, Z-plane, along with samples from the constrained and vanilla GAN. $\Delta t = 1.0$. The discriminator forms a 'band' around the exact samples. In this example, the vanilla GAN output coincides with a strong approximation.

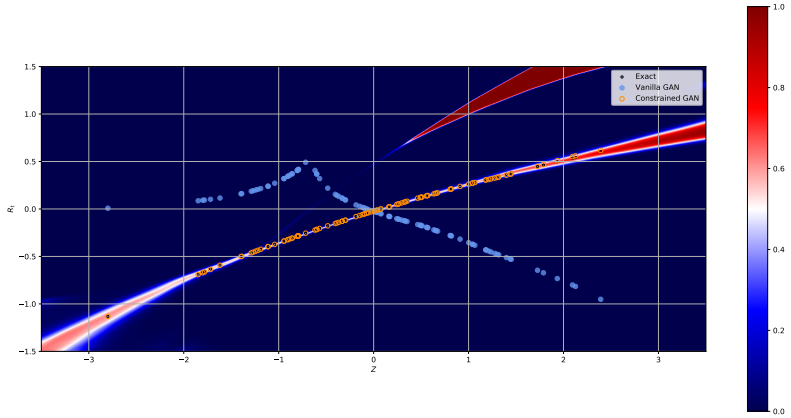
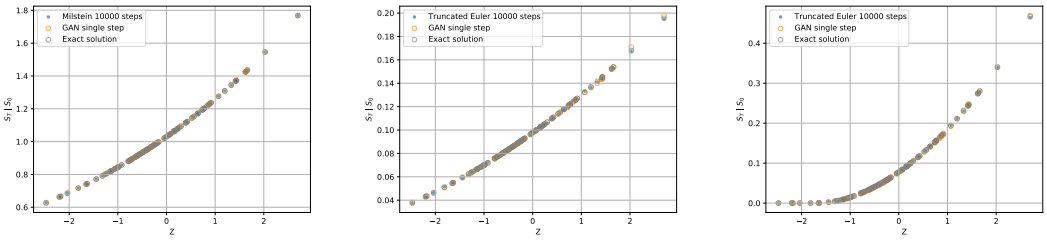


Figure A.2: Example for CIR process with Feller condition satisfied: output of the constrained GAN discriminator on the R_t, Z-plane, along with samples from the constrained and vanilla GAN. $S_t = 0.1, \Delta t = 1.0$. Again, the discriminator forms a 'band' around the exact samples. This time, the vanilla GAN does not coincide with a strong approximation. It has learned a different map, which would have been ruled out by the constrained GAN discriminator.

A.3. Empirical validation of section 4.4.1

The technique proposed for extension in section 4.4.1 is tested using the following setup: a matrix of $100\times10{,}000$ $N(0,1)$ samples was drawn, representing Brownian motion increments on a fine grid of 10,000 steps. These increments are used in a discrete-time approximation of 10,000 time steps of the interval $[0,1]$. For GBM, this was a Milstein scheme and for the CIR process it was the truncated Euler scheme. A random vector Z of size 100 was constructed using the increments on the fine grid, as shown

in lemma 4.13, which was provided as input to the GAN. It was also used to find corresponding exact variates. For the CIR process, this was done as $F^{-1}_{S_T|S_0}(F_Z(Z))$. In all cases, the results in the plane $((S_T \mid S_0), Z)$ overlap, which validates the theory empirically.



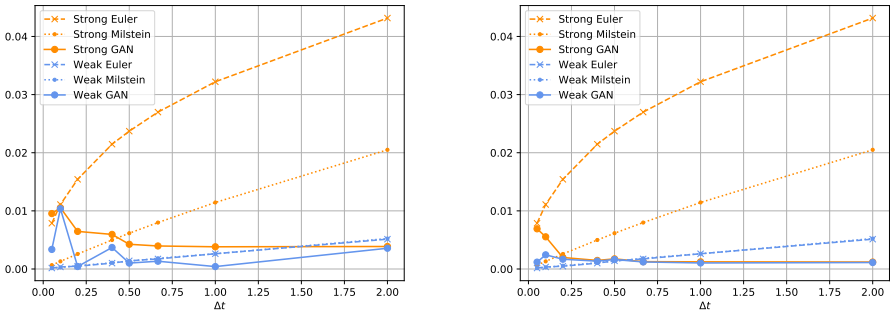
(a) GBM (b) CIR, Feller condition satisfied (c) GBM, Feller condition violated

Figure A.3: Empirical validation of lemma 4.13 and the theory presented in chapter 4. A discrete-time scheme was run on 10,000 time steps on the interval $[0, 1]$. The Z found using lemma 4.13 was provided as input to the GAN and used to sample corresponding exact variates. All results coincide.

A.4. Additional Results on Paths

100,000 paths were constructed using the vanilla GAN and constrained GAN. The results for GBM and the CIR process if the Feller condition is satisfied are shown.

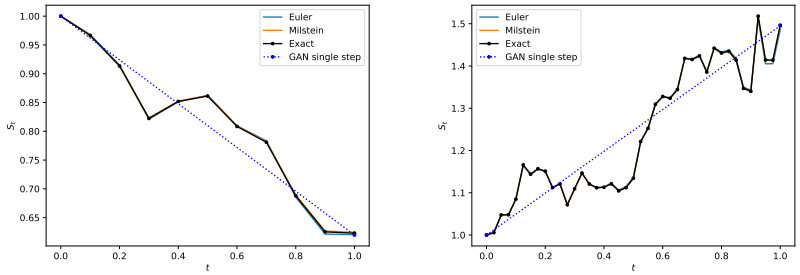
A.4.1. GBM



(a) Vanilla GAN (b) Constrained GAN

Figure A.4: GBM: Weak and strong errors at time $T = 2$ of paths constructed using the vanilla GAN and constrained GAN, compared with the Euler scheme, Milstein scheme and exact scheme. $S_0 = 1$, $\Delta t = 0.05$, i.e. 40 steps between 0 and 2.

A.4.2. Single-step approximation



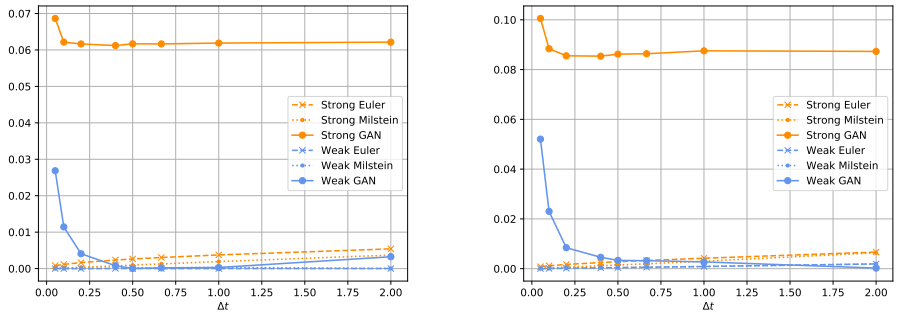
(a) 10 steps (b) 40 steps

Figure A.5: Example of paths obtained with the single-step approximation on the time interval $[0, 1]$ with the constrained GAN and multiple steps with the Euler, Milstein and exact schemes, setting $S_0 = 1$. The technique from lemma 4.13 is used to find the increment Z as input to the GAN.

Table A.3: Single-step approximation on the time interval $[0, 1]$. The weak and strong errors are compared to those of the Euler and Milstein schemes using $n \in \{5, 10, 40\}$ steps to cross the same interval.

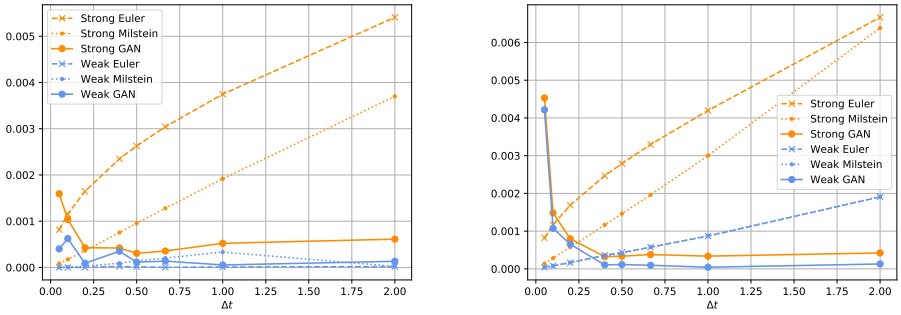
	$n = 5$		$n = 10$		$n = 40$	
Error	Weak	Strong	Weak	Strong	Weak	Strong
Euler	3.3E-04	1.0E-02	1.3E-04	7.4E-03	3.9E-05	3.7E-03
Milstein	2.6E-04	1.7E-03	1.3E-04	8.6E-04	3.4E-05	2.2E-04
GAN	5.0E-04	7.0E-04	4.9E-04	6.9E-04	4.9E-04	6.9E-04

A.4.3. CIR process: Feller condition satisfied



(a) $S_0 = 0.1$ (b) $S_0 = 0.2$

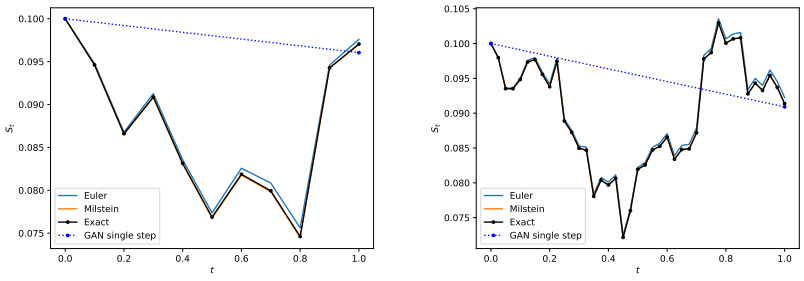
Figure A.6: Vanilla GAN: weak and strong errors at time $T = 2$ of paths constructed, for various Δt, compared with the truncated Euler, Milstein and exact schemes. Two starting values are chosen. The large strong error in both cases is due to the fact that in this case, the vanilla GAN fails to provide a path-wise approximation to the strong solution.



(a) $S_0 = 0.1$ (b) $S_0 = 0.2$

Figure A.7: Constrained GAN: weak and strong errors at time $T = 2$ of paths constructed, for various Δt, compared with the truncated Euler, Milstein and exact schemes. Two starting values are chosen: one equal to $\bar{S}$, in which case the discrete-time schemes perform well as they have the mean reversion 'built-in' and one greater than $\bar{S}$, in which case the GAN performs better.

A.4.4. Single-step approximation - Feller condition satisfied



(a) 10 steps (b) 40 steps

Figure A.8: Example of paths obtained with the single-step approximation on the time interval $[0, 1]$ by the constrained GAN and multiple steps with the Euler, Milstein and exact schemes, setting $S_0 = 1$. The technique from lemma 4.13 is used to find the increment Z as input to the GAN.

Table A.4: Single-step approximation on the time interval $[0, 1]$ with the constrained GAN, corresponding to figure A.8. The weak and strong errors are compared to those of the truncated Euler and Milstein schemes using $n \in \{5, 10, 40\}$ steps to cross the same interval.

	$n = 5$		$n = 10$		$n = 40$	
Error	Weak	Strong	Weak	Strong	Weak	Strong
Euler	1.7E-05	1.2E-03	8.9E-06	8.5E-04	2.3E-06	4.3E-04
Milstein	7.1E-07	2.6E-04	9.1E-07	1.3E-04	3.3E-07	3.2E-05
GAN	6.4E-05	7.1E-04	6.7E-05	7.2E-04	7.3E-05	7.2E-04

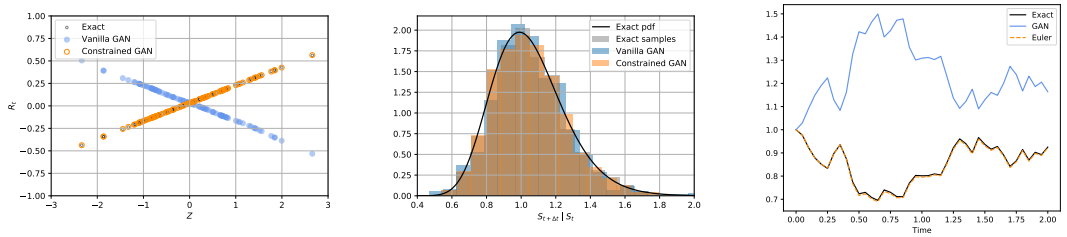
A.4.5. CPU runtimes on single-step approximation

Table A.5: CPU runtimes to construct 100,000 paths on the interval $[0, 1]$ using discrete-time schemes on varying steps, versus the GAN on a single step if the Feller condition is not satisfied. The results were run on an AMD Ryzen 5 3600 processor @3.5GHz. Note that this does not take into account GPU parallellisation, which would make inference with the GAN much faster.

n	Euler	Milstein	GAN (single step)
5	7.0E-03	8.0E-03	3.1E-01
10	1.4E-02	1.5E-02	3.1E-01
40	7.7E-02	9.7E-02	3.5E-01

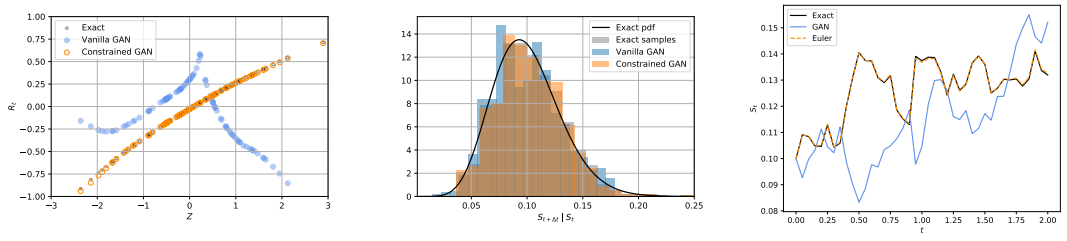
A.4.6. Vanilla GAN fails to provide a strong approximation

Two examples are shown of the failure of the vanilla GAN to provide a strong approximation. In one case, it learns the map with opposite sign of Z, which is equal in distribution, but clearly not path-wise. It is translated into 'mirrored paths'. The second example shows a strange map in the R_t, Z-plane, that is close in conditional distribution, but again not path-wise.



(a) Map in R_t, Z-plane (b) Histogram of $S_{t+\Delta t} \mid S_t$ (c) Example of a corresponding path

Figure A.9: GBM: Example of the failure of the vanilla GAN to provide a strong approximation.

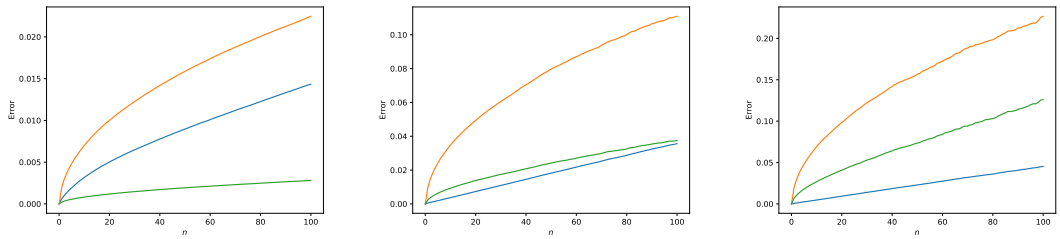


(a) Map in R_t, Z-plane (b) Histogram of $S_{t+\Delta t} \mid S_t$ (c) Example of a corresponding path

Figure A.10: CIR process: Example of the failure of the vanilla GAN to provide a strong approximation.

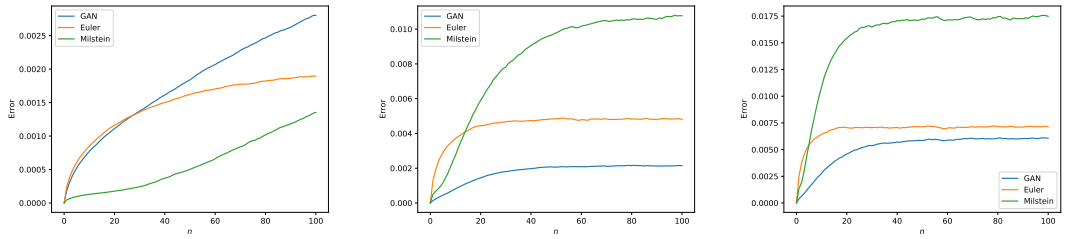
A.4.7. Strong error over time

This time, we test the constrained GAN by sampling iteratively for 100 steps, keeping Δt fixed and varying n. The results indicate that the output remains stable after repeated application of the algorithm. For GBM, paths are discounted with $\exp(-\mu n \Delta t)$, as the process grows exponentially over time. At large time steps, the truncated Milstein scheme underperforms compared to the truncated Euler scheme. The truncated Milstein scheme does not perform well if the Feller condition is not satisfied.



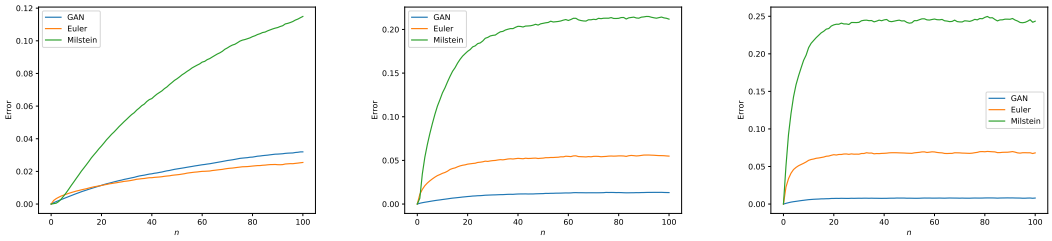
(a) $\Delta t = 0.1$ (b) $\Delta t = 0.5$ (c) $\Delta t = 1.0$

Figure A.11: GBM: strong error over time for discounted synthetic paths obtained with the adapted CGAN on the GBM problem. Various time steps Δt are shown.



(a) $S_0 = 0.1, \Delta t = 0.1$ (b) $S_0 = 0.1, \Delta t = 0.5$ (c) $S_0 = 0.1, \Delta t = 1.0$

Figure A.12: Strong error over time of paths obtained with the constrained GAN on the CIR process if the Feller condition is satisfied. Various time steps Δt are shown.



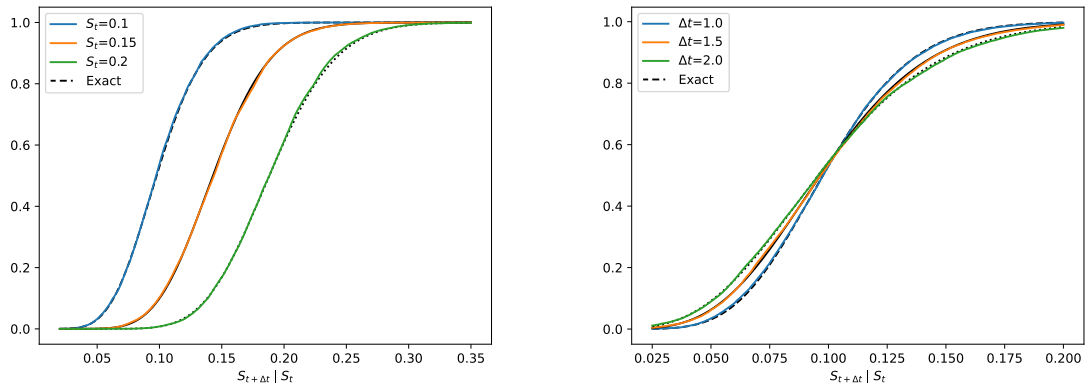
(a) $S_0 = 0.1, \Delta t = 0.1$ (b) $S_0 = 0.1, \Delta t = 0.5$ (c) $S_0 = 0.1, \Delta t = 1.0$

Figure A.13: Strong error over time of paths obtained with the constrained GAN on the CIR process if the Feller condition is not satisfied. Various time steps Δt are shown.

A.5. Additional Results for Conditional GAN

This appendix shows the results using a conditional GAN, both for the vanilla and constrained GAN. First, we show that the conditional GAN is able to interpolate between the conditions it was trained on. Then, the conditional GAN is trained on both S_t and Δt.

A.5.1. Interpolation capabilities on a single condition



(a) Varying S_t

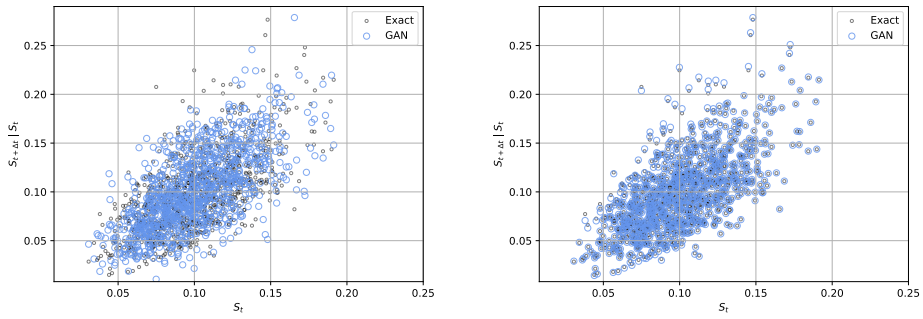
(b) Varying Δt

Figure A.14: ECDF plots of the vanilla conditional GAN output. The distribution corresponds to the CIR process with the Feller condition satisfied. Left: GAN trained on $S_t \in \{0.1, 0.2\}$ and right: GAN trained on $\Delta t \in \{1, 2\}$. The Feller condition was satisfied in both cases. Although trained on only two unique inputs, the conditional GAN can interpolate between the two conditions. The exact CDF is given in black.

Table A.6: KS statistic and Wasserstein distance on test condition half-way between the two training conditions, corresponding to the settings in figure A.14.

Parameter	KS stat.	p-value	KS stat. ref.	Wass. dist.	Wass. dist. ref.
$S_t = 0.15$	2.1E-02	0.41	1.9E-02	1.6E-03	1.5E-03
$t = 1.5$	2.1E-02	0.42	1.9E-02	1.5E-03	1.4E-03

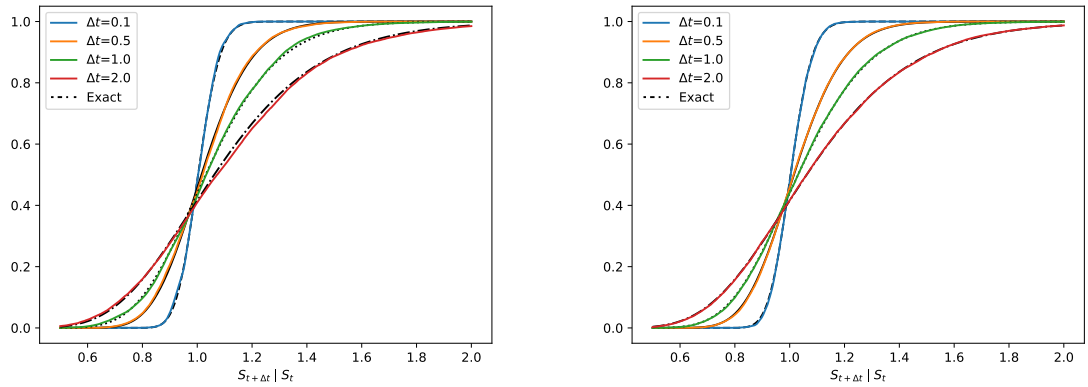
A.5.2. Feller condition satisfied: autocorrelation structure



(a) Vanilla GAN

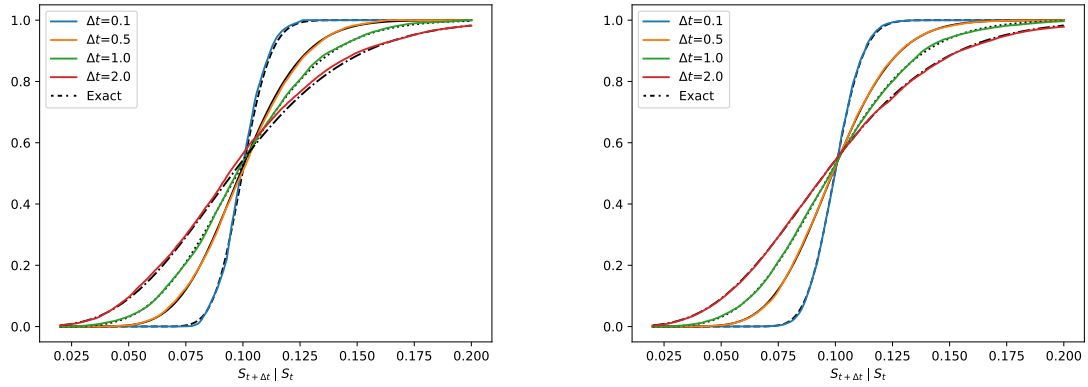
(b) Constrained GAN

Figure A.15: CIR process with the Feller condition satisfied: scatter plot of $S_{t+\Delta t} \mid S_t$ versus S_t. This plot reveals that the autocorrelation structures of the GAN output and reference samples are similar. This time, the vanilla GAN did not manage to provide a path-wise approximation.



(a) Vanilla GAN (b) Constrained GAN

Figure A.16: ECDF plots of the conditional GAN output on the GBM problem for various choices of Δt.



(a) Vanilla GAN (b) Constrained GAN

Figure A.17: ECDF plots of the conditional GAN output on the CIR process with the Feller condition satisfied, for various choices of Δt. S_t was held fixed at 0.1.

A.5.3. GBM and CIR process - vanilla vs constrained GAN

Table A.7: Conditional GAN results on the GBM problem and the CIR process, both for the vanilla conditional GAN and the constrained conditional GAN. The test conditions were $\Delta t = 1$ and $(S_t, \Delta t) = (0.1, 1)$ for GBM and CIR process, respectively. The statistics are computed using 2,000 generated samples and are compared with an equally sized set of reference samples, repeated 100 times. The average of the 100 repetitions is reported in the table. If the Feller condition was not satisfied, the samples were scaled with $\bar{S}$. Otherwise, the logreturns transformation was used.

Vanilla conditional GAN					
SDE	KS stat.	p-value	KS stat. ref.	Wass. dist.	Wass. dist. ref.
GBM	2.4E-02	0.29	1.7E-02	9.9E-03	8.3E-03
CIR, Feller cond. satisfied	2.3E-02	0.33	1.9E-02	1.3E-03	1.2E-03
CIR, Feller cond. not satisfied	6.5E-02	0.00	1.9E-02	4.7E-03	3.5E-03
Constrained conditional GAN					
SDE	KS stat.	p-value	KS stat. ref.	Wass. dist.	Wass. dist. ref.
GBM	2.0E-02	0.46	1.7E-02	8.4E-03	8.3E-03
CIR, Feller cond. satisfied	2.1E-02	0.45	2.0E-02	1.3E-03	1.2E-03
CIR, Feller cond. not satisfied	2.8E-02	0.10	2.0E-02	3.7E-03	3.7E-03

A.6. Parameter sets and Training Phase

A.6.1. SDE parameters

The default parameters for the SDEs considered are given in table A.8.

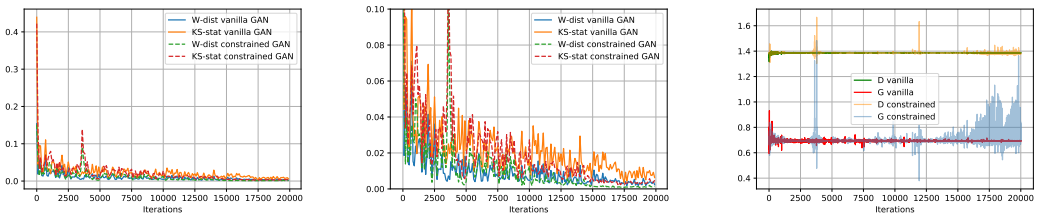
Table A.8: Parameter sets used during the experiments for weak convergence.

Process	Parameter	Value
	Δt	1
GBM	μ	0.05
	σ	0.2
	S_0	1
CIR, Feller cond. satisfied	κ	0.1
	$\bar{S}$	0.1
	γ	0.1
	S_0	0.1
CIR, Feller cond. not satisfied	κ	0.1
	$\bar{S}$	0.1
	γ	0.3
	S_0	0.1

A.6.2. Conditional GAN: construction of training set

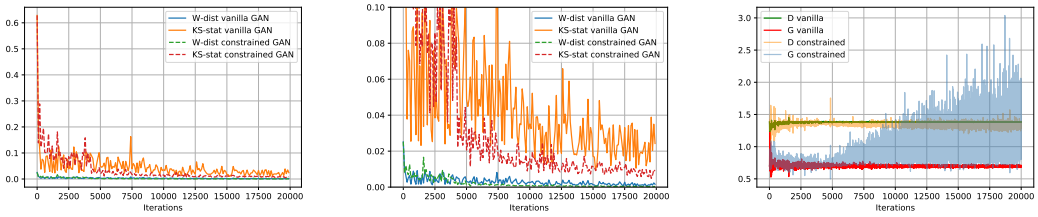
In the GBM case, the conditional GAN was trained only on Δt, as the logreturns transformation remove dependence on S_t. For the CIR process (or general Itō diffusions), two vectors $[\Delta t_1, \dots, \Delta t_m]$ and $[S_t^1, \dots, S_t^n]$ are defined, which form the possible outcomes of the training set. The space of training conditions is then given by $\mathcal{C} = [\Delta t_1, \dots, \Delta t_m] \times [S_t^1, \dots, S_t^n]$. A training set was made of tuples $((S_{t+\Delta t'} \mid S_t'), \Delta t', S_t')$, with $(\Delta t', S_t') \in \mathcal{C}$ and where $S_{t+\Delta t'} \mid S_t'$ is sampled from the exact distribution $F_{S_{t+\Delta t'}|S_t'}$. The experiments were run for $m = n = 100,000$. $[\Delta t_1, \dots, \Delta t_m]$ was chosen to range from 0.05 to 2, while $[S_t^1, \dots, S_t^n]$ contained values between 0.01 and 0.5 on the CIR process if the Feller condition was satisfied. If the Feller condition was not satisfied, the vector of previous samples ranged from 10^{-4} to 0.9. Both vectors were randomly shuffled and then concatenated to represent a random sample from their Cartesian product $\mathcal{C}$. The vectors of Δt and S_t were constructed by repeating a vector of 8 and 10 samples $m/8$ and $n/10$ times, respectively. E.g., for Δt, the vector $[0.05, 0.1, 0.2, 0.4, 0.5, 0.67, 1, 2]$ was repeated 12,500 times and then randomly shuffled uniformly. It was found that letting samples recur many times in this way improved results over defining a continuous range of parameters.

A.6.3. Training process



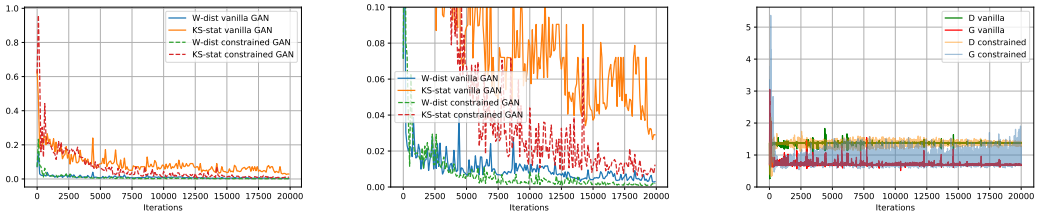
(a) (b) Zoomed in (c) Loss functions

Figure A.18: **GBM**: KS-statistic and Wasserstein distance, and loss function values during the training process. The statistics were computed on the test condition $\Delta t = 1$.



(a) (b) Zoomed in (c) Loss functions

Figure A.19: **CIR, Feller condition satisfied**: KS-statistic and Wasserstein distance, and loss function values during the training process. The statistics were computed on the test condition $(S_t, \Delta t) = (0.1, 1)$. The large values in the generator loss function coincided with large generator gradients, although there is no visibly harmful effect on the training process.

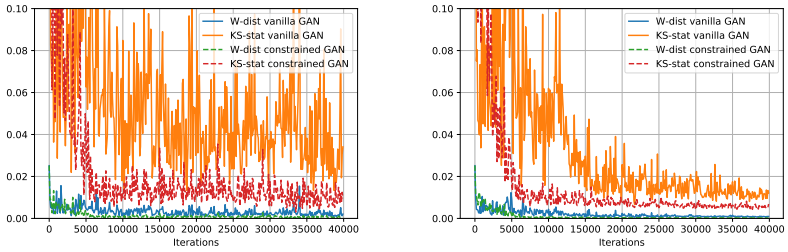


(a) (b) Zoomed in (c) Loss functions

Figure A.20: **CIR, Feller condition not satisfied**: KS-statistic and Wasserstein distance, and loss function values during the training process. The statistics were computed on the test condition $(S_t, \Delta t) = (0.1, 1)$.

A.6.4. Effect of the Learning Rate Schedule

A learning rate schedule was used that divides the base learning rate of Adam by a factor 1.05 every 500 iterations. This means that updates to the generator become exponentially smaller, gradually freezing the training process. In figure A.21, the effect of applying this schedule is shown compared to not applying it. Applying the learning rate schedule stabilises the training process for both GANs. If we would train even longer, e.g. 100,000 steps, we would need to modify this choice of learning rate factor accordingly.



(a) No additional LR schedule (b) Divide LR by 1.05 every 500 iterations.

Figure A.21: Effect of the LR schedule during training process using 40,000 iterations. The KS-statistic and Wasserstein distance are shown during the training process.

A.7. Further Extensions of the Constrained GAN

This appendix provides the additional ideas for extending the constrained GAN to general problems, in addition to the alternative of training on a high-quality discrete-time approximation.

A.7.1. Karhunen-Loève expansion

A very different method involves the explicit construction of the Brownian motion through a Fourier series expansion. This technique is called Karhunen Loève expansion, cf. for example [2, p.57]. On a bounded time interval $0 \leq t \leq T$, a standard Brownian motion can be constructed as follows, for any event $\omega \in \Omega$:

$$W_t(\omega) = \sum_{j=1}^{\infty} Z_j(\omega)\psi_j(t) \quad \forall\ t \in [0,T], \tag{A.1}$$

$$\approx \sum_{j=1}^{K} Z_j(\omega)\psi_j(t), \tag{A.2}$$

$$\psi_j(t) := \frac{2\sqrt{2T}}{\pi(2j+1)} \sin\left(\frac{(2j+1)\pi t}{2T}\right), \tag{A.3}$$

where $Z_j \overset{iid}{\sim} N(0,1)\ \forall i \in \mathbb{N} \cup \{0\}$. By truncating this sum to some K, e.g. at $O(10^3)$ terms, one obtains an approximation to the Brownian motion that is continuous and can be discretised over an arbitrary partition. A high-quality approximation to the strong solution adapted to $\mathcal{F}_t = \sigma(\{W_t\})$ could be obtained by numerically integrating the stochastic integral with respect to equation A.1 on an arbitrarily fine partition. The GAN is then modified as follows: instead of a single random variable Z as input, it receives the K random variables $\{Z_j\}_{j=0}^{K-1}$ that appear in equation A.2. The discriminator also receives these K inputs, which ensures the output remains adapted and that the generator learns the map $\boldsymbol{Z}$ to $S_{t+\Delta t} \mid S_t$. This means that the GAN takes e.g. $O(10^3)$ variables as input. The GAN would be conditioned on $t \in [0,T]$ and trained on the dataset $(t, S_t, (S_{t+\Delta t} \mid S_t)(\omega), \boldsymbol{Z}(\omega))$, where $\boldsymbol{Z}(\omega) \in \mathbb{R}^K$ contains the $Z_j(\omega)$ used in the expansion and $(S_{t+\Delta t} \mid S_t)(\omega)$ are obtained via numerical integration. The generator is then defined by:

$$\left(\hat{S}_{t+\Delta t} \mid \hat{S}_t\right)(\omega) = G_\theta(\boldsymbol{Z}(\omega), t, \Delta t, S_t). \tag{A.4}$$

Again, the central method is to enforce path-wise equality via the discriminator, which alternatingly receives $\left((S_{t+\Delta t} \mid S_t)(\omega), \boldsymbol{Z}(\omega)\right)$ and $\left(\left(\hat{S}_{t+\Delta t} \mid S_t\right)(\omega), \boldsymbol{Z}(\omega)\right)$. However, this method requires more computational overhead due to the size of the input, although $O(10^3)$ inputs is not large compared to other applications of GANs, such as image generation [53]. The methodology is again the same: construct a training set with an expensive numerical scheme to allow the GAN to sample a large time step ahead once it has been trained.

A.7.2. Regularisation

Another way to enforce adaptedness and correspondence of $\{\hat{S}_{t_k} \mid \hat{S}_{t_{k-1}}\}$ and $\{W_{t_k} - W_{t_{k-1}}\}$ is to combine an exact simulation scheme with a discrete-time scheme such as the Milstein scheme. In contrast to the previous example of using the Milstein approximation as a dataset, we could use a discrete-time scheme on a single small time step as regularisation term in the generator loss function. If we again use the example of the Milstein scheme, the regularisation term between times t and $t + \delta t$ would become:

$$L_{\text{Milstein}} = \mathbb{E}_{Z\sim P_Z}\left|G_\theta(Z,t,\delta t,S_t) - S_t - A(t,S_t)\delta t - B(t,S_t)\sqrt{\delta t}Z - \frac{1}{2}\delta t B(t,S_t)B'\left(Z^2-1\right)\right|. \tag{A.5}$$

δt should be chosen as small as possible, or the regularisation term would bias the GAN to the Milstein scheme, however, it must still be part of the range of time conditions, or the generator would not have observed training examples based on δt. The key idea is that the regularisation term 'penalises'

the generator if it uses Z differently than prescribed by the Milstein scheme. Since the Milstein scheme approximates a strong solution, it is adapted to $\mathcal{G}_k$. In example 4.6, where we found that the vanilla GAN may produce 'mirrored' versions of the paths of the strong solution, the regularisation term would force the GAN to choose the correct sign of Z. Note that δt may remain fixed, while the rest of the GAN is trained on a much larger range $\Delta t \in (0, T]$.