UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering, Mathematics & Computer Science

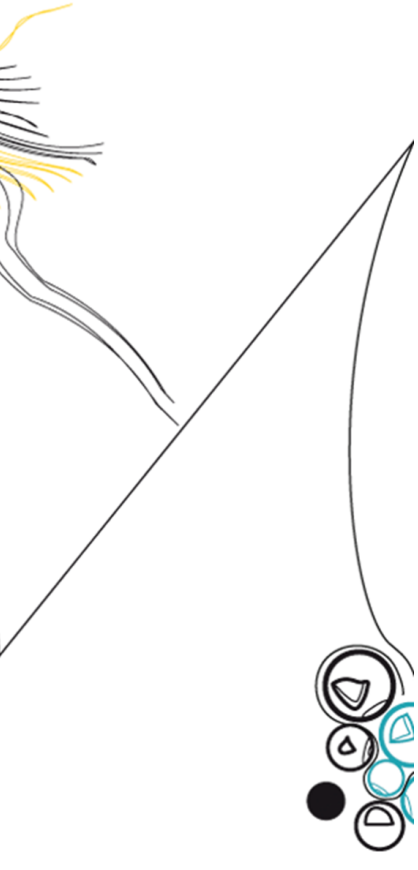
Master thesis - Applied Mathematics

Neural differential equations solving stochastic mean field games

Sven C. Dummer
September 2021

Specialization: Systems Theory, Applied Analysis and Computational Science (SACS)

Chair: Applied Analysis (AA)



Supervisors:
Prof. Dr. Christoph Brune
Len Spek MSc.

Graduation committee:
Prof. Dr. Christoph Brune
Len Spek MSc.
Dr. Matthias Schlottbom

Date of presentation:
13 September, 2021

Abstract

Mean field games (MFG) and mean field control are effective mathematical models for simulating and analysing interactions within large populations of agents. These models have applications for robust control in engineering and robotics, as well as in data science, economics and crowd motion. Recently, to overcome the curse of dimensionality for high-dimensional MFGs, deep learning techniques have been used effectively. Although stochastic MFGs have been well studied, the role of deep learning in stochastic Lagrangian frameworks is still unexplored. Hence, this thesis studies generative neural differential equations like neural ODEs and neural SDEs. To research their advantages, we study two dimensional particle motion examples. Simulations show that a neural SDE generator is more appropriate when information about the Lagrangian view of a stochastic MFG is desired. Moreover, a neural ODE generator can deal with costs in the MFG that cannot be dealt with via a standard neural network. Finally, we show that a neural SDE generator can even stabilize the training compared to standard neural network generators.

Keywords: Mean field games, Hamilton-Jacobi-Bellman, Fokker-Planck equations, optimal transport, stochastic differential equations, deep learning, neural networks, neural ODE

Acknowledgements

This thesis marks the end of five amazing years as an Applied Mathematics student at the University of Twente. Definitely at the start of my student years, I was wondering whether I would be able to finish my study. I remember that I initially had difficulties adapting to the level of mathematics involved and considered quitting. Therefore, the more proud I am today that I actually finished my master Applied Mathematics and continue pursuing a PhD afterwards. Finishing my master via thesis would not have been possible without the help of several people.

First of all, I really want to thank my supervisors Prof. Dr. Christoph Brune and MSc. Len Spek. Even though we had to resort to non-ideal online meetings due to the Coronavirus, they tried their utter best to help me. They really helped me with pursuing my interests and finding a suitable thesis topic. Furthermore, I considered our discussions as very fun and insightful.

Secondly, I want to thank all my friends. Due to the Corona pandemic, most of the days had little variation. All of you made these days more bearable. I especially want to thank Annemarie, Femke, Jarco, Jesse, Leander, Lotte, Lucas, Nienke, Tessa and Wisse, or in short, Shoarma. Playing games with you, talking about the struggles of the master thesis, celebrating discord birthdays and talking about other various topics all helped me get my mind of the master thesis and relax a bit. Furthermore, I genuinely want to thank Theo and Guido. By playing many computer games together, even though you two had to carry me to victory, and all the joking around while playing those games, both of you really helped me get my mind off the, sometimes stressful, master thesis.

Finally, I thank my sister, my father and my mother. Whenever I had trouble with my master thesis, I could always rely on your support. I especially want to thank my mother. She always supported me throughout my study and motivated me to perform as best as I can. I truly believe I would not have made it this far without her and would not have been able to pursue a PhD.

Contents

Chapter 1

Introduction

Mean field games theory studies a non-cooperative game between an infinite number of agents. In this game, the agents want to move through a specific environment. More precisely, the agents want to move optimally given the movement of the other agents. To achieve the optimal movement, the agent can alter their strategy, namely the speed and direction in which he or she wants to move. Hence, depending on the presence of noise on the chosen strategy, the agent's movement follows an ordinary differential equation (ODE) or a stochastic differential equation (SDE). Given only a finite number of agents, the optimal strategy of a single agent depends on the actions of the other agents. However, due to the infinite number of agents, every agent has an infinitesimal effect on the loss of the other agents. Therefore, due to the symmetry, one only has to consider one specific agent to argue about the Nash equilibrium of the game. This observation makes mean field games much easier to solve than a game between a finite number of players.

Mean field games theory has been pioneered by Lasry and Lions. In their celebrated paper [Lasry and Lions, 2007], they, amongst others things, introduce the equations that represent the mean field game, provide some context regarding economics and finance, and introduce a variational formulation for a particular type of mean field game. In the years after this publication, mean field games gained applications in a wide range of fields. In [Lachapelle and Wolfram, 2011] and [Burger et al., 2013], the authors look at crowd modelling. More precisely, [Lachapelle and Wolfram, 2011] investigates congestion and aversion dynamics in pedestrian crowds. Furthermore, mean field games have been applied to the problem of swarm robotics. In swarm robotics, one tries to control the behaviour of several individual robots so that the robots together perform a task beyond the capability of an individual robot. For two works on swarm robotics that use mean field game theory, see [Liu et al., 2018] and [Elamvazhuthi and Berman, 2019]. Finally, two other application fields are finance and economics. See the papers [Achdou et al., 2014], [Gomes et al., 2015], and [Firoozi and Caines, 2017] for works in finance and economics.

Recently [E et al., 2018] connects the theory of mean field games (MFGs) to the theory of deep learning. In particular, the authors reformulate the training problem of residual neural networks as a discretization of a mean field optimal control problem. This optimal control problem can again be related to a mean field game. From this viewpoint, it is interesting to see if there are more connections between neural networks and mean field games. Two particular works that focus on this are [Ruthotto et al., 2020] and [Lin et al., 2020]. These works proposed to solve MFGs using neural networks. The main motivation behind using neural networks to solve MFGs is the curse of dimensionality. Most existing numerical schemes suffer from the curse of dimensionality as these methods use meshes. By using neural networks to represent the solutions of the MFG, one can

circumvent the use of meshes and, therefore, the curse of dimensionality.

Concerning the two mentioned articles that use neural networks, [Ruthotto et al., 2020] solves the curse of dimensionality for a particular class of MFGs, called potential mean field games. They solve this class in case the agents have deterministic strategies. To solve the deterministic potential MFG, the authors utilize the corresponding variational formulation introduced in [Lasry and Lions, 2007]. In addition, they use the fact that the movement of an agent follows an ODE. On the other hand, [Lin et al., 2020] solves a stochastic mean field game by using a min-max formulation. The algorithm presented in this paper does not use the fact that an agent follows an SDE.

The previous paragraph shows a gap between the work of Ruthotto et al. [2020] and Lin et al. [2020]. This gap is the case of solving stochastic MFGs utilizing neural networks and the agent dynamics SDE. Consequently, it is interesting to investigate whether the agent dynamics SDE can be beneficial for solving a high-dimensional stochastic MFG. More specifically, this thesis investigates whether using a neural SDE or a neural ODE is beneficial for solving stochastic potential mean field games. Regarding this context, the focus is on the interpretability of the obtained agent trajectories, training stability and the number of MFG instances that can be solved.

1.1 Contributions

To investigate the advantages of neural differential equations in solving stochastic MFGs, we use a solution algorithm based on a two-player game formulation of potential MFGs. Within this game, the generator for the distribution of the agent positions is present and needs to be learned. This generator is a function that produces samples from the agent position distribution at any time. To learn this generator via the used algorithm, we parametrize the generator by a neural network structure. As a neural differential equation generator is motivated by the Lagrangian view of MFGs, this thesis investigates whether using a neural differential equation as the generator has added benefits over using a more standard neural network architecture like a ResNet.

The contribution of this thesis consists of three parts. The first contribution is related to training stability. To investigate training stability, we construct two mean field optimal control numerical experiments. These numerical experiments are exactly the same except for the terminal cost. One of the two problem instances exhibits training instability when using a generic neural network generator, such as a ResNet or a feedforward neural network generator. When using a neural SDE as the generator instead, both numerical experiments converge. This observation shows that a neural SDE can help in improving the stability of the training procedure.

Second, compared to a standard neural network, such as a ResNet, we find that using a neural SDE as the generator yields more interpretable learned agent trajectories. This observation is important as it facilitates research into the effect of stochasticity on the agent dynamics. As stochastic MFGs can be interpreted as a regularized version of optimal transport, this research can give insight into the effect of stochasticity on the learned optimal transport.

Finally, following the Lagrangian view of an MFG, we show that a neural ODE is a sensible choice for the generator. Additionally, using a neural ODE generator, it is possible to deal with more general problems than the algorithm in [Lin et al., 2020]. In particular, with a neural ODE, one can employ the method in [Ruthotto et al., 2020] to deal with KL-divergences as terminal cost and an entropy as running cost. The latter two costs can be used to deal with optimal transport and crowd aversion problems, respectively. To our knowledge, our extension of the algorithm in [Lin et al., 2020] is the first deep learning based solution method for such more general stochastic potential MFGs.

1.2 Thesis outline

First, Chapter 2 gives some background information on neural networks, neural ODEs/SDEs, mean field games, and optimal transport. Using the background information on mean field games, Chapter 3 first introduces the model used for solving a potential mean field game. Subsequently, we treat the details of the solution algorithms. Using these algorithms, Chapter 4 contains the results of several numerical experiments. Finally, Chapter 5 gives the conclusion of our research and Chapter 6 gives ideas for further research.

Chapter 2

Theoretical background

This chapter contains the necessary background information for discussing the advantages of neural differential equations in solving stochastic potential mean field games. First, we provide an introduction to neural networks, which treats the basics of neural networks, the ResNet architecture, and the recently introduced neural ODEs and neural SDEs. We use the latter neural network structures as well as more standard neural network architectures, such as the ResNet architecture, to solve the MFGs. After introducing neural networks, we define a mean field game via a Hamilton-Jacobi-Bellman equation and a Fokker-Planck equation. Moreover, we present the MFG class treated in this thesis, namely potential mean field games. Subsequently, we introduce optimal transport (OT) and discuss the connection between potential MFGs and OT. We use this discussion to gain a better understanding of MFGs. Finally, we reformulate a mean field game as a two-player game. Our MFG solution algorithm uses a similar two-player game formulation in combination with neural networks to solve MFGs.

2.1 Neural networks

In the last decade, neural networks have seen a surge in new applications. For instance, it has applications in object detection, segmentation [Feng et al., 2021; Sinha and Dolz, 2021], and generative modelling [Karras et al., 2019]. Moreover, it is used in PET image reconstruction [Reader et al., 2021] and the classification of breast abnormalities [Yu et al., 2021]. This section first describes the motivation behind neural networks and the basic operations of a neural network. Subsequently, we discuss the training procedure for a neural network and introduce the well-known ResNet architecture. Finally, neural ODEs and neural SDEs are introduced.

2.1.1 Brief motivation and a simple neural network

The idea of an artificial neural network is based on the workings of neurons in, for example, a human brain. The human brain contains neurons that are connected via axons and dendrites. More specifically, they are connected via synapses, which are the connections between axons and dendrites. This connection between two neurons is depicted in Figure 2.1. Under the influence of external stimuli, the strength of synaptic connections often changes. Due to this change in synaptic strength, learning is possible.

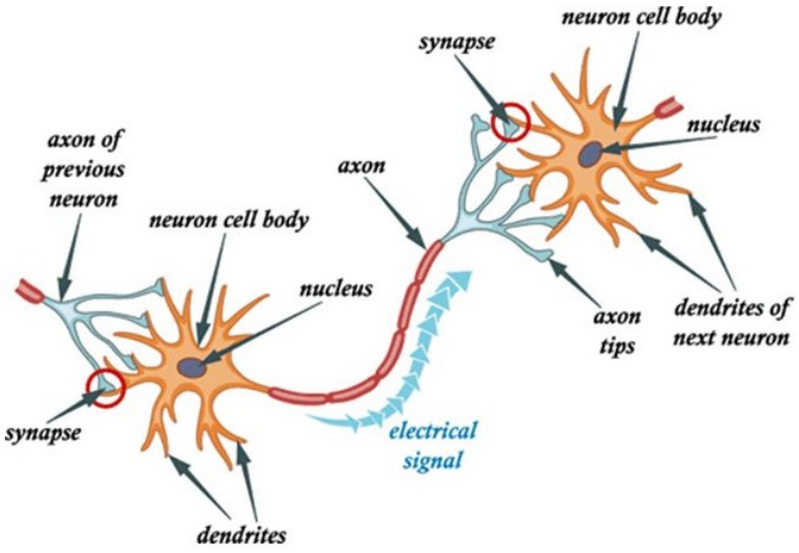


Figure 2.1: Two biological neurons connected via axons, dendrites, and synapses.

Following this biological principle, an artificial neural network is defined as a connected network of cells where each connection between cells has a particular weight. This weight corresponds to the strength of a synapse. For a specific neuron, Figure 2.2 gives a schematic view.

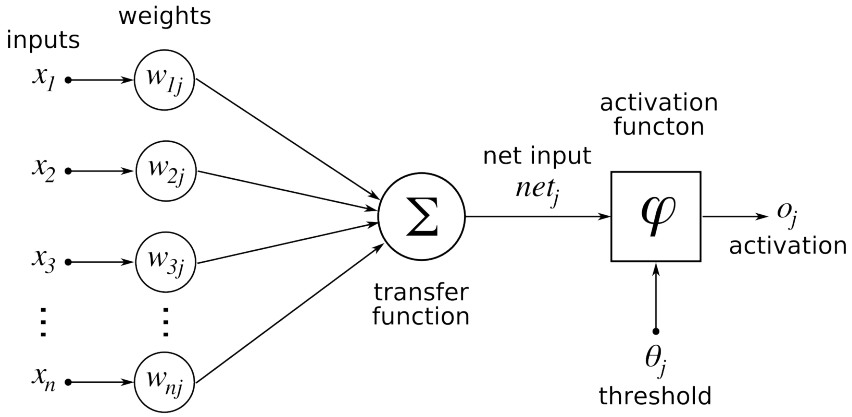


Figure 2.2: A single artificial neuron with input x and output o_j. The signals x_i of other neurons are combined via synaptic weights w_{ij} and the transfer function Σ to obtain the net input. Using the activation function on the net input, one obtains the output o_j of the neuron.

Here one can see that similar to a biological neuron, an artificial neuron receives input signals x_i from the other neurons. Each of these inputs gets a synaptic weight w_{ij}. The neuron combines the inputs with the synaptic weights and outputs a signal o_j via a transfer function and an activation function [Aggarwal, 2018].

The preceding discussion describes the workings of a single artificial neuron on a high level. In particular, it shows that the output of a neuron depends on the output of connected neurons. However, no information is given on how exactly the output of a neuron and the output of the whole neural network are calculated. Moreover, the output of a neuron and the output of a neural network are dependent on the precise connections between neurons. These connections are not discussed either.

In artificial neural networks, the connections between neurons are up to the user. The chosen neural network architecture mainly determines how the output is calculated. To explain the calculation of the neural network output, also called the forward pass or forward propagation, consider the simple fully connected feedforward neural network architecture in Figure 2.3.

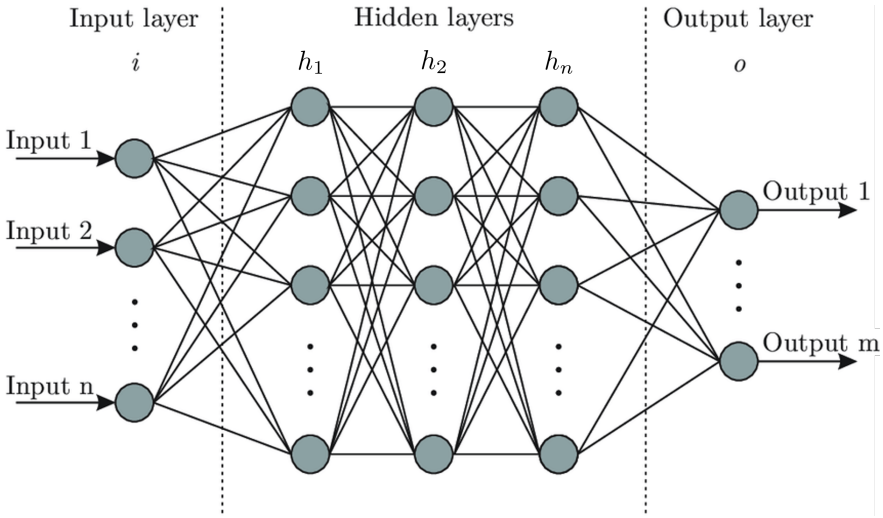


Figure 2.3: The fully connected feedforward neural network architecture. In this architecture, every artificial neuron in a specific layer is connected to each neuron in the previous layer. For a schematic view of a single artificial neuron, see Figure 2.2.

As can be seen in the figure, a simple neural network takes as input a vector with input values. These input values can correspond to any vector of features. For instance, it can be a vector of pixel values for black-white images. Furthermore, the neural network has a couple of layers that consist of several neurons. Every hidden layer takes as input the output of the neurons in the previous layer. Subsequently, hidden layer $h^{(i)}$ combines the outputs of the previous layer via

$$u^{(i)} = \sigma_h(W^{(i)}u^{(i-1)} + b^{(i)}),$$

where σ_h is an activation function that returns a vector of the same size as the input, $W^{(i)} \in \mathbb{R}^{w_i \times w_{i-1}}$ is a matrix belonging to the hidden layer $h^{(i)}$, w_i is the number of neurons in the hidden layer $h^{(i)}$, $u^{(i-1)} \in \mathbb{R}^{w_{i-1}}$ is the output of the previous layer (with $u^{(0)} \in \mathbb{R}^d$ the input), and $b^{(i)} \in \mathbb{R}^{w_i}$ is a bias term. Regarding the choice of σ_h, the linear function $\sigma(x) = x$ is not often used as activation function. This choice is avoided as it makes the neural network an affine mapping. To learn difficult nonlinear functions, nonlinear activation functions are needed. Very often the sigmoid σ, the ReLu or the tanh function are taken as activation function σ_h. These are defined as

$$\sigma = \frac{1}{1+\exp(-x)}, \quad \text{ReLu} = \max(x, 0), \quad \tanh = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}.$$

After having calculated the output of the final hidden state recursively, an additional layer calculates the output. The output $o \in \mathbb{R}^m$ can be calculated by

$$o = \sigma_o(W_o u^{(n)} + b_o)\,,$$

where $u^{(n)} \in \mathbb{R}^{w_n}$ is the output of the last hidden layer, $W_o \in \mathbb{R}^{m \times w_n}$ is a weight matrix, $b_o \in \mathbb{R}^m$ is a bias term, and σ_o is an activation function for the output. The choice of the function σ_o depends on the application. For instance, when one wants to perform regression, one might take σ_o as the identity function. On the other hand, one often uses the softmax function when one deals with a classification task. The i-th entry returned by this function is given by

$$\sigma(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}\,.$$

Hence, the softmax function returns a probability vector. Using this function, the output can be made a vector of probabilities where each entry corresponds to a specific class. By choosing the class with the largest probability, we can classify objects.

2.1.2 Training procedure: loss function and backpropagation

The previous section only describes how to calculate the output given a choice of architecture, activation functions, and weights. However, it does not tell how to choose the correct weights. To find proper weights for the neural network, one trains the neural network. This section describes how to train neural networks.

The neural network is trained by finding the weights that minimize some loss function. Depending on the specific application, some loss functions are more appropriate than others. For instance, assume a regression task needs to be performed on a dataset $\{(x_i, y_i)\}_{i=1}^N$ where $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}^m$. Moreover, assume we use a neural network with weights Φ. Defining $f_\Phi(x)$ as the function that calculates this neural network's output, performing the regression task means that parameters Φ should be found such that $f_\Phi(x_i) \approx y_i$. To find such parameters, an example of a problem that can be solved is

$$\min_\Phi L(\Phi) := \min_\Phi \frac{1}{N} \sum_{i=1}^{N} \|f_\Phi(x_i) - y_i\|^2\,, \tag{2.1}$$

where $\|\cdot\|$ is the standard Euclidean norm.

After a specific loss function and network architecture are chosen, the loss function is minimized using a specific optimizer. The conventional choice for minimizing the loss function is a gradient descent strategy. Examples of optimizers are vanilla gradient descent, RMSProp, and ADAM. Furthermore, when a large amount of data is present, stochastic gradient descent strategies are often applied. For more information on gradient-based optimizers, see [Aggarwal, 2018].

To use such a gradient-based optimizer for training the neural network, the gradient of the loss function with respect to the weights should be calculated. For neural networks, the standard way to do this is called backpropagation. To see how backpropagation works, consider the simple fully connected feedforward neural network as introduced in Section 2.1.1 with σ_h and σ_o taken to be elementwise operating functions. In other words, the functions apply the same function to every entry of the input vector. Moreover, for illustration purposes, take the loss function as given in equation (2.1).

Without loss of generality, consider the weight matrix of hidden layer k. More precisely, consider weight $W_{i,j}^{(k)}$. This is the weight of the connection between neuron i in hidden layer k and neuron

j in hidden layer $k-1$. The desired partial derivative is $\frac{\partial L}{\partial W_{i,j}^{(k)}}$ for the chosen loss function L. To calculate the desired partial derivative, one can use the following identity:

$$\frac{\partial L}{\partial W_{i,j}^{(k)}} = \frac{1}{N}\sum_{l=1}^{N} \frac{\partial}{\partial W_{i,j}^{(k)}} \left\| f_\Phi(x_l) - y_l \right\|^2 .$$

The only quantity that remains unknown is $\frac{\partial}{\partial W_{i,j}^{(k)}} \hat{L}(x_l, y_l)$ with $\hat{L}(x_l, y_l) := \left\| f_\Phi(x_l) - y_l \right\|^2$. Recalling that $W_{i,j}^{(k)}$ is only used in calculating entry i of the kth hidden layer's output, using the chain rule gives

$$\begin{aligned}\frac{\partial}{\partial W_{i,j}^{(k)}} \hat{L}(x,y) &= \frac{\partial \hat{L}}{\partial (u^{(k)})_i} \frac{\partial (u^{(k)})_i}{\partial W_{i,j}^{(k)}} \\ &= \frac{\partial \hat{L}}{\partial (u^{(k)})_i} \sigma_h'((W^{(k)} u^{(k-1)} + b^{(k)})_i)(u^{(k-1)})_j \,,\end{aligned}$$

with $(u^{(k)})_i$ entry i of the output of hidden layer k. The only quantity that needs to be calculated is $\frac{\partial \hat{L}}{\partial (u^{(k)})_i}$. This will be calculated recursively. Using the chain rule, one gets

$$\frac{\partial \hat{L}}{\partial (u^{(k)})_i} = \sum_{l=1}^{w_{k+1}} \frac{\partial \hat{L}}{\partial (u^{(k+1)})_l} \frac{\partial (u^{(k+1)})_l}{\partial (u^{(k)})_i} .$$

Moreover, one has

$$\begin{aligned}\frac{\partial (u^{(k+1)})_l}{\partial (u^{(k)})_i} &= \sigma_h'((W^{(k+1)} u^{(k)} + b^{(k+1)})_l) W_{l,i}^{(k+1)} \,, \quad k < n \,, \\ \frac{\partial (u^{(n+1)})_l}{\partial (u^{(n)})_i} &= \sigma_o'((W_o u^{(n)} + b_o)_l)(W_o)_{l,i} \,,\end{aligned}$$

using $u^{(n+1)} := f_\Phi(x)$.

To summarize, the following equations are obtained:

$$\frac{\partial L}{\partial W_{i,j}^{(k)}} = \frac{1}{N}\sum_{l=1}^{N} \frac{\partial}{\partial W_{i,j}^{(k)}} \left\| f_\Phi(x_l) - y_l \right\|^2 = \frac{1}{N}\sum_{l=1}^{N} \frac{\partial}{\partial W_{i,j}^{(k)}} \hat{L}(x_l, y_l) \,, \tag{2.2}$$

$$\frac{\partial \hat{L}}{\partial W_{i,j}^{(k)}} = \frac{\partial \hat{L}}{\partial (u^{(k)})_i} \frac{\partial (u^{(k)})_i}{\partial W_{i,j}^{(k)}} = \frac{\partial \hat{L}}{\partial (u^{(k)})_i} \sigma_h'((W^{(k)} u^{(k-1)} + b^{(k)})_i)(u^{(k-1)})_j \,, \tag{2.3}$$

$$\frac{\partial \hat{L}}{\partial (u^{(k)})_i} = \sum_{l=1}^{w_{k+1}} \frac{\partial \hat{L}}{\partial (u^{(k+1)})_l} \frac{\partial (u^{(k+1)})_l}{\partial (u^{(k)})_i} \,, \tag{2.4}$$

$$\frac{\partial (u^{(k+1)})_l}{\partial (u^{(k)})_i} = \sigma_h'((W^{(k+1)} u^{(k)} + b^{(k+1)})_l) W_{l,i}^{(k+1)} \,, \tag{2.5}$$

$$\frac{\partial (u^{(n+1)})_l}{\partial (u^{(n)})_i} = \sigma_o'((W_o u^{(n)} + b_o)_l)(W_o)_{l,i} \,. \tag{2.6}$$

Using equation (2.2), the desired derivatives for the gradient-based optimizer are calculated. Hence, for every datapoint (x_l, y_l), the derivative of $\hat{L}$ with respect to the weights needs to be calculated.

In order to do so, one uses equation (2.3). Following this equation, one needs to calculate equation (2.4). First, note that $\frac{\partial \hat{L}}{\partial (u^{(n+1)})_l} := \frac{\partial \hat{L}}{\partial f_\Phi(x)_l}$ is known as $\hat{L}(x,y) = \|f_\Phi(x) - y\|^2$. Moreover, the $\frac{\partial (u^{(k+1)})_l}{\partial (u^{(k)})_i}$ terms are known via equations (2.5) and (2.6). Using this information, equation (2.4) can be used to calculate $\frac{\partial \hat{L}}{\partial (u^{(n)})_i}$ for $i = 1, \ldots, w_n$. However, following the same logic again, using the values of $\frac{\partial \hat{L}}{\partial (u^{(n)})_l}$ in combination with equations (2.4), (2.5) and (2.6), results in $\frac{\partial \hat{L}}{\partial (u^{(n-1)})_i}$. This recursive process can be continued until the desired layer k at which point the partial derivative with respect to the weight are calculated using equation (2.3).

The backpropagation procedure as presented for the simple fully connected neural network might need to be adjusted for other architectures. One scenario is when one deals with the skip connections that we introduce in the next section. However, the basics of backpropagation stay the same. For every architecture, the idea is to calculate the derivatives at the last layers first and use the outcome to calculate the derivatives for the earlier layers. Hence, the backpropagation algorithm moves from the output of the neural network to the input. For more detailed information on backpropagation, see [Bishop, 2006], [Goodfellow et al., 2016], and [Aggarwal, 2018].

To summarize, before training a neural network, a neural network architecture and an appropriate loss function need to be chosen. After that, the backpropagation algorithm is employed to calculate the gradients of the loss with respect to the neural network weights. Finally, using these gradients combined with a gradient-based optimizer, the weights are updated iteratively to train the neural network.

2.1.3 The ResNet architecture

Section 2.1.1 describes the basics of a neural network using the fully connected feedforward neural network architecture. However, not all architectures have the same form. For instance, the ResNet architecture [He et al., 2016] adds additional structure to the aforementioned fully connected feedforward neural network.

A ResNet is created by putting several ResNet modules as depicted in Figure 2.4 in series. The idea behind a ResNet relates to feature learning. For instance, take a picture. Some complicated objects in the picture need many layers to capture them, while other simple objects only need very little nonlinearity. When learning both simple and complicated objects, convergence is slow when using a very deep network with fixed depth across all paths from input to output. By the use of the skip connections as seen in Figure 2.4, the ResNet allows copying. This ensures that there are multiple paths from input to output that do not have the same effective length. In this way, when training the ResNet, the learning procedure can decide how many layers to use to learn each specific feature. For example, for simple objects, many layers are skipped, while for more complicated objects a larger number of connections might be followed to increase the nonlinearity. Hence, the paths make sure that the learning algorithm can decide the appropriate level of complexity needed for a specific input.

Besides the above motivation of ResNets, there is another property of the short and long paths. This property can be observed during the training of ResNets. When training ResNets, the shorter pathways are easier to learn. On the other hand, the longer routes take care of the fine-grained details. Consequently, the deeper paths only learn when it is necessary and only learn a small amount. In other words, the longer routes fine-tune the shorter paths, which are easier to learn [Aggarwal, 2018].

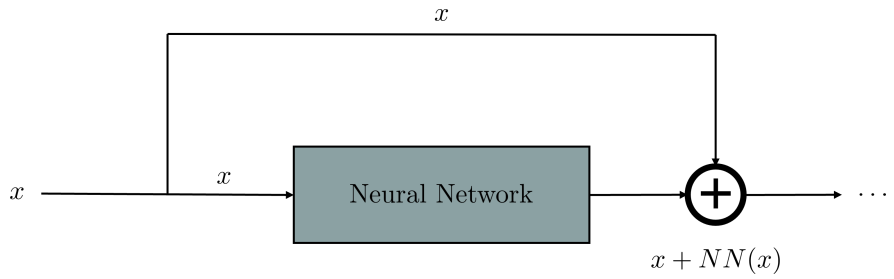


Figure 2.4: A basic ResNet module. The ResNet module takes as input x, calculates the output $NN(x)$ of a neural network and outputs $x + NN(x)$. The neural network corresponding to $NN(x)$ can be any neural network. For instance, it can be the simple feedforward neural network depicted in Figure 2.3.

2.1.4 Neural ODE and neural SDE

Assume a neural network model is given which updates the hidden states using

$$h_{t+1} = h_t + f(h_t, \theta_t)\,,$$

where $t \in \{0, \dots, T\}$, $h_t \in \mathbb{R}^d$ and θ_t is a vector of parameters. For instance, the ResNet discussed in the previous section fits this framework. The recursion resembles a simple Euler discretization with a time step $\delta t = 1$ of the following ordinary differential equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}h(t) = f(h(t), \theta(t))\,, \quad h(0) = h_0\,.$$

From this observation, a neural ordinary differential equation can be defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}h(t) = f(h(t), t, \theta)\,, \quad h(0) = h_0\,, \tag{2.7}$$

where f is a neural network.

This definition of a neural ODE became broadly known mainly due to [Chen et al., 2018]. In this paper, the authors use the adjoint sensitivity method to calculate gradients with respect to the parameters. This method uses an (adjoint) ordinary differential equation to obtain the gradients. One advantage of the adjoint sensitivity method is that one does not have to save the intermediate quantities of the forward pass. This allows model training with a constant memory cost. When one instead backpropagates through the operations of the ODE solver used for the forward propagation, intermediate quantities do have to be saved. A disadvantage of the adjoint sensitivity method is that the adjoint ordinary differential equation could be unstable. In this case, one might need to do checkpointing and save quantities from the forward pass to deal with the instability.

Furthermore, using the neural ODE framework, continuous normalizing flows are introduced. This generative model uses an ordinary differential equation to generate realistic data from a target domain. An application of this generative model is 3d point cloud generative modelling [Yang et al., 2019]. In addition, the original neural ODE paper introduces another generative model, namely a generative time series model. The advantage of using an ordinary differential equation for such a task instead of a standardly used recurrent neural network is the ability to deal with irregular time stamps. Besides the generative modelling applications, neural ODEs have other applications. For instance, one can apply the framework to learn Hamiltonian systems [Toth et al., 2019; Zhong et al., 2019].

After the introduction and application of the neural ODE, extensions of this framework were suggested. A natural extension of the neural ODE framework is replacing the ODE with an SDE. Hence, instead of equation (2.7), one gets:

$$\mathrm{d}x = f_\theta(x,t)\mathrm{d}t + g_\phi(x,t)\mathrm{d}W_t\,,$$

where W_t is a Brownian motion, f_θ is a neural network, and g_ϕ is a neural network. Additionally, such a neural SDE can be obtained as a limiting process of deep latent Gaussian models [Tzen and Raginsky, 2019]. For more information on SDEs, see Appendix A.1.

Neural stochastic differential equations have several applications. In [Kong et al., 2020], a neural SDE is used to quantify both aleatoric and epistemic uncertainty. In short, it is used for uncertainty quantification. In [Oganesyan et al., 2020], neural SDEs are investigated as a stochastic regularization of neural ODEs. The paper [Liu et al., 2019] discusses this topic as well. Moreover, the latter paper treats neural SDEs in the light of adversarial and non-adversarial input perturbations. It is indicated that neural SDEs are more resistant to such perturbations than neural ODEs.

In general, SDEs can be helpful in scenarios where many small and unobserved interactions govern phenomena. Examples are motions of molecules in a liquid, allele frequencies in a gene pool, or prices in a market [Li et al., 2020]. Hence, in cases where the dynamical environments exhibit a stochastic nature due to uncaptured factors, a neural SDE can be useful. In [Look et al., 2020] and [Li et al., 2020], the authors treat several case studies that can be related to this point. In [Li et al., 2020], a generative time series model is built for a stochastic Lorenz attractor system and a 1D geometric Brownian motion. Moreover, they perform predictions on a motion capture dataset. In [Look et al., 2020], besides performing predictions on the same motion capture dataset, a neural SDE is trained to be used for weather forecasting. Neural SDEs driven by stochastic processes with jumps are introduced in [Jia and Benson, 2019] as a generative framework for hybrid dynamical systems with both continuous and discrete behaviour.

To use neural stochastic differential equations for applications, a training procedure should be developed. In [Look et al., 2020] the neural stochastic differential equations are trained using only deterministic approximation methods. A different approach is introduced in [Li et al., 2020]. To train neural stochastic differential equations, they propose to use an SDE version of the adjoint sensitivity training method for neural ODEs. Another method to train neural SDEs backpropagates through the operations of the SDE solver.

2.2 Mean field games and optimal transport

This section treats mean field games and optimal transport. First, the definition of a mean field game is introduced. Subsequently, we treat a special class of mean field games: potential mean field games. This class of mean field games is studied in this thesis and has a connection with optimal transport (OT) theory. Optimal transport is introduced and important results of OT are given. Subsequently, the connection between potential mean field games and optimal transport is discussed. In the end, we treat a reformulation of a specific type of mean field game as a two-player game. This two-player game idea forms the basis of the used MFG solution algorithm.

2.2.1 Derivation of mean field game models

Mean field games model large populations of rational agents that play a (stochastic) differential game. The goal of the model is to find a Nash equilibrium: a state of the model where no agent

wants to alter its actions. To formally introduce MFGs, we follow Section 2.1 of [Gomes and Saúde, 2014].

First, we introduce the movement and the strategies of the agents. Define $f(x,v) : \mathbb{R}^d \times U \to \mathbb{R}^d$ as a vector field where $x \in \mathbb{R}^d$ denotes the position of an agent and $v \in U$ is the control of this agent. Throughout this thesis, we assume that $U = \mathbb{R}^n$ for some n. Furthermore, define a diffusion matrix $\sigma(x,v) : \mathbb{R}^d \times U \to \mathcal{M}_{\mathbb{R}}^{d\times m}$ with $\mathcal{M}_{\mathbb{R}}^{d\times m}$ the set of real $d \times m$ matrices. Given f and σ, the position of the agent is determined through the stochastic differential equation (SDE):

$$\mathrm{d}x = f(x,v)\mathrm{d}t + \sigma(x,v)\mathrm{d}W_t\,, \tag{2.8}$$

where W_t is a Brownian motion. Throughout, this thesis assumes that each agent's dynamics is driven by an independent Brownian motion and that $v = v(x,t)$. So v is a feedback Markovian control.

To define a game, costs need to be introduced for the actions v of the agents. Fix an agent which knows the strategy v of all other agents. This means that the distribution $\theta(x,t)$ of the positions of the other agents is known. Given this known agent distribution $\theta(x,t)$ and defining $\mathcal{P}\left(\mathbb{R}^d\right)$ to be the set of probability distributions (Borel probability measures) on $\mathbb{R}^d$, the cost function that the agent tries to minimize is given by

$$\begin{aligned} J_{x,t}(v) &= \mathbb{E}\left(\int_t^T L(z(s),v(s),\theta(\cdot,s))\mathrm{d}s + G(z(T),\theta(\cdot,T))\right) \\ \text{s.t.}\quad \mathrm{d}z &= f(z,v)\mathrm{d}t + \sigma(z,v)\mathrm{d}W_t\,, \quad z(t) = x\,, \end{aligned} \tag{2.9}$$

where $L : \mathbb{R}^d \times U \times \mathcal{P}\left(\mathbb{R}^d\right) \to \mathbb{R}$ is a loss function for the trajectory of the agent, $G : \mathbb{R}^d \times \mathcal{P}\left(\mathbb{R}^d\right) \to \mathbb{R}$ is a terminal cost, and T is the final time. The value function corresponding to this problem is given by

$$\phi(x,t) = \inf_v J_{x,t}(v)\,.$$

It is known from stochastic optimal control theory that the value function ϕ is a viscosity solution to the Hamilton-Jacobi-Bellman (HJB) equation

$$-\frac{\partial\phi}{\partial t}(x,t) + H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t)) = 0\,, \tag{2.10}$$

where H_ϕ is the hessian of ϕ with respect to the spatial location x and $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{M}_{\mathbb{R}}^{d\times d} \times \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$ is the Hamiltonian given by:

$$H(x,p,M,\theta) = \sup_{v\in U}\left(-\langle p, f(x,v)\rangle_{l^2} - \frac{1}{2}\langle M, \sigma(x,v)\sigma(x,v)^T\rangle_F - L(x,v,\theta)\right)\,. \tag{2.11}$$

Furthermore, the value function ϕ at the final time T satisfies

$$\phi(x,T) = G(x,\theta(\cdot,T))\,. \tag{2.12}$$

Suppose now that ϕ is a smooth enough solution to equation (2.10) with boundary condition given by equation (2.12). Moreover, assume for initial time t_0 there exists a function $\tilde{v} : \mathbb{R}^d \times [t_0,T] \to U$ that attains the supremum in $H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t))$ where H is given in equation (2.11). Then a simple differentiation argument shows that

$$\begin{aligned} f(x,\tilde{v}) &= -\nabla_p H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t))\,, \\ \frac{1}{2}\sigma(x,\tilde{v})\sigma(x,\tilde{v})^T &= -\nabla_M H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t))\,. \end{aligned} \tag{2.13}$$

Moreover, this $\tilde{v}$ is optimal. To summarize, the value function completely determines the optimal strategy of a single agent via the Hamiltonian.

Up to this point, a single agent is considered. For this specific agent, we can determine the distribution θ of the positions of the other agents. However, for another agent, it may be the case that this distribution is different. To see this, consider two agents with distinct initial positions. For agent 1, the initial distribution of the positions of the other agents is determined by the position of agent 2. Similarly, when considering agent 2, the distribution at the initial time is determined by the position of agent 1. As agent 1 and agent 2 are initially positioned at two different locations, the θ for agent 1 differs from the one of agent 2.

In the above argument, a finite number of agents is present. When using an infinite number of agents, the θ distributions of different agents do not differ anymore. The reason is that a single agent has a negligible influence on the distribution of the position of all the agents. Hence, when removing a single agent and considering the position distribution without this agent, the distribution does not change. Consequently, each agent has the same distribution θ of the position of the other agents. As the agents all believe the same θ and act optimally, all agents solve equation (2.9) with the exact same θ. Therefore, the optimal control function $v(x,t)$ is the same for all the agents.

Up to this point, we have considered how an agent moves optimally given an infinite number of agents. We have found that all the agents believe the same distribution θ and, therefore, solve the same optimal control problem given by equation (2.9). As a consequence, all the agents have the same optimal strategy v, which is entirely characterized by the value function ϕ via equation (2.13).

Although the above will be important in deriving the mean field game definition, it does not yet fully characterize the definition of a mean field game. The reason is that a mean field game is the Nash equilibrium solution of the game between the agents. So far, the discussion lacks this Nash equilibrium structure. In our context, a Nash equilibrium is a belief ρ such that the resulting agent position distribution is again ρ. For an arbitrary belief θ, the resulting distribution might differ from the belief. What this possibly different distribution will be, can be characterized by a partial differential equation (PDE). Using this PDE, the Nash equilibrium belief can be characterized. We investigate this now.

Let $\rho(\cdot,t) \in \mathcal{P}\left(\mathbb{R}^d\right)$ for every $t \in [t_0,T]$ be the probability distribution of the positions x of the agents at time t. As the dynamics of all the agents is given by equation (2.8) for a specific control v, $\rho(x,t)$ satisfies the Fokker-Planck (FP) equation and is given by

$$\begin{aligned} \frac{\partial \rho}{\partial t}(x,t) &= -\nabla \cdot (f(x,v(x,t))\rho(x,t)) + \frac{1}{2}\nabla^2 \cdot (\sigma(x,v(x,t))\sigma(x,v(x,t))^T \rho(x,t)) \\ \rho(x,0) &= \rho_0(x)\,, \end{aligned} \tag{2.14}$$

where $\nabla\cdot$ is the divergence operator and ρ_0 is the initial distribution of the position of the agents. The $\nabla^2\cdot$ operator is a function that takes as input a matrix-valued function and is defined by

$$\nabla^2 \cdot h(x) = \sum_{i=1}^{d}\sum_{j=1}^{d} \frac{\partial^2}{\partial x_i \partial x_j}(h_{ij}(x))\,.$$

In equation (2.14), the agent control v is arbitrary. However, from equation (2.13), the optimal controls are known. Hence, given that all agents act optimally and believe the agent distribution is $\theta(x,t)$, then the actual distribution changes according to

$$\begin{aligned} \frac{\partial \rho}{\partial t}(x,t) &= \nabla \cdot (\nabla_p H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t))\rho(x,t)) \\ &\quad - \nabla^2 \cdot (\nabla_M H(x,\nabla\phi(x,t),H_\phi(x,t),\theta(\cdot,t))\rho(x,t)) \\ \rho(x,0) &= \rho_0(x)\,. \end{aligned} \tag{2.15}$$

Up to this point, equations (2.10) and (2.15) are derived given a specific believed agent distribution θ. Hence, solving equations (2.10) and (2.15) yields the resulting agent distribution given a belief θ. To have a Nash equilibrium, one needs that given the distribution θ, the agents choose actions in a certain way such that the resulting distribution is again θ. In other words, θ should satisfy the Fokker-Planck equation in equation (2.15). Using this observation and combining the HJB equation, FP equation, and the corresponding boundary conditions, one gets:

$$\begin{aligned} -\frac{\partial \phi}{\partial t}(x,t) + H(x, \nabla\phi(x,t), H_\phi(x,t), \rho(\cdot,t)) &= 0 \\ \frac{\partial \rho}{\partial t}(x,t) = \nabla \cdot (\nabla_p H(x, \nabla\phi(x,t), H_\phi(x,t), \rho(\cdot,t))\rho(x,t)) & \\ - \nabla^2 \cdot (\nabla_M H(x, \nabla\phi(x,t), H_\phi(x,t), \rho(\cdot,t))\rho(x,t)) & \\ \rho(x,0) = \rho_0(x)\,, \quad \phi(x,T) = G(x, \rho(\cdot,T))\,. & \end{aligned} \tag{2.16}$$

In this system of PDEs, the HJB equation characterizes the optimal strategy of the agent given a belief ρ and the FP equation characterizes the Nash equilibrium belief ρ. Equation (2.16) is taken as a definition of mean field games.

To summarize, a mean field game is a combination of a Hamilton-Jacobi-Bellman equation coming from a (stochastic) optimal control problem and a Fokker-Planck equation coming from the agent dynamics. Moreover, from the derivation of these equations, it is obvious there are two views on MFGs. The first view is the Lagrangian or particle view. The particle view looks at the MFG from the viewpoint of a single agent. The second view is the Eulerian view. This view looks at the MFG via the agent distribution as a function of position and time. Hence, the difference between the Eulerian and the Lagrangian view is that one view looks at the global behaviour of the agents via ρ and the other looks at the local behaviour of a single agent. Mainly the Lagrangian view will play an essential role in the remainder of the thesis.

2.2.2 Potential mean field games

A special instance of a mean field game is the potential mean field game. This MFG class is the main object of study in this thesis. Before discussing the definition of a potential MFG, we must introduce the definition of a variational derivative.

Definition 2.2.1 (Variational derivative for functionals acting on $\mathcal{P}(\mathbb{R}^d)$ [Ruthotto et al., 2020])**.** Assume a functional $\mathcal{F} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$ is given. The variational derivative $F(x,\rho) = \frac{\delta \mathcal{F}(\rho)}{\delta \rho}(x)$ is defined via

$$\lim_{h \to 0} \frac{\mathcal{F}(\rho + hw) - \mathcal{F}(\rho)}{h} = \int_{\mathbb{R}^d} F(x,\rho) w(x) \mathrm{d}x\,, \quad \forall w \in L^2(\mathbb{R}^d)$$

Definition 2.2.2 (Potential MFG [Ruthotto et al., 2020])**.** A potential MFG is a regular mean field game as described by equation (2.11) and (2.16) with the following assumptions:

- The loss function L in equation (2.11) is given by $L(x,v,\rho) = \tilde{L}(x,v) + F(x,\rho(\cdot,t))$.
- $F(x,\rho) = \frac{\partial \mathcal{F}}{\partial \rho}(x)$ for some functional $\mathcal{F} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$.
- $G(x,\rho) = \frac{\partial \mathcal{G}}{\partial \rho}(x)$ for some functional $\mathcal{G} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$.

Potential mean field games are special due to an equivalent fluid dynamics formulation as posed in Section 2.6 of [Lasry and Lions, 2007]:

Theorem 2.2.1 (Variational formulation potential MFGs [Lasry and Lions, 2007])**.** Assume the following potential MFG is given:

$$\begin{aligned} -\partial_t \phi(x,t) - \nu\Delta\phi(x,t) + H(x, \nabla\phi(x,t)) &= F(x, \rho(\cdot,t)) \\ \partial_t \rho(x,t) - \nu\Delta\rho(x,t) - \nabla\cdot(\rho(x,t)\nabla_p H(x,\nabla\phi(x,t))) &= 0 \\ \rho(x,0) = \rho_0(x), \quad \phi(x,T) = G(x,\rho(\cdot,T)), \quad t &\in [0,T]\,, \end{aligned}$$

where H is:

$$H(x,p) = \sup_v(-\langle p, v\rangle_{l^2} - L(x,v))\,.$$

This corresponds to equations (2.11) and (2.16) with $\sigma = \sqrt{2\nu}I$ and I the identity matrix. In the given setting, the MFG system given by a coupled Hamilton-Jacobi-Bellman equation and Fokker-Planck equation coincides with the necessary optimality conditions of the following variational problem:

$$\begin{aligned} \min_{\rho,v} \quad & \int_0^T \left\{ \int_{\mathbb{R}^d} \rho(x,t) L(x, v(x,t)) \mathrm{d}x + \mathcal{F}(\rho(\cdot,t)) \right\} \mathrm{d}t + \mathcal{G}(\rho(\cdot,T)) \\ \text{s.t.} \quad & \partial_t \rho(x,t) - \nu\Delta\rho(x,t) + \nabla\cdot(\rho(x,t)v(x,t)) = 0 \\ & \rho(x,0) = \rho_0(x)\,. \end{aligned}$$

Hence, instead of solving two coupled partial differential equations to solve the potential mean field game, one can solve a minimization problem with only a Fokker-Planck constraint. This is not true for general mean field games. Moreover, this formulation has a natural connection with optimal transport. We discuss this connection in the next section.

2.2.3 Optimal transport and its relationship to MFGs

Optimal transport is the theory of transforming one distribution into another distribution while minimizing a specific loss function. It has several applications. In [Yang and Karniadakis, 2020], a fluid-dynamics formulation of optimal transport is used to create a generative model. For image-to-image translation and domain translation, [Lu et al., 2019] uses optimal transport in combination with CycleGAN to get a one-to-one mapping from one domain to another domain with desired properties. To aid optimization in Generative Adversarial Networks, [Arjovsky et al., 2017] introduces the WGAN. In the WGAN, the Wasserstein distance is used as the loss function. This distance is the solution to an optimal transport problem. In [Dittmer et al., 2020], the authors use optimal transport via a WGAN-like loss function to perform denoising. For more applications, see [Peyré and Cuturi, 2019].

Optimal transport has two formulations: Monge's formulation and Kantorovich's formulation. The formulation by Kantorovich can be seen as a relaxation of Monge's formulation. This thesis only uses Monge's formulation. Before giving the formal definition of Monge's optimal transport problem, an example problem is provided. Consider two density functions $\rho_0(x)$ and $\rho_1(x)$ on some domain $\Omega \subset \mathbb{R}^d$. A map f transports the density ρ_0 to ρ_1 if for all subsets $U \subset \Omega$ and corresponding inverse images $f^{-1}(U)$, one has:

$$\int_U \rho_1(x)\mathrm{d}x = \int_{f^{-1}(U)} \rho_0(x)\mathrm{d}x\,. \tag{2.17}$$

This means that if $x \sim \rho_0$ then $f(x) \sim \rho_1$. To show what such transport maps can look like, take

$$\rho_0(x) = \begin{cases} x-1, & \text{if } 1 \leq x \leq 2 \\ 3-x, & \text{if } 2 \leq x \leq 3 \\ 0, & \text{otherwise} \end{cases}$$

$$\rho_1(x) = \begin{cases} x-4, & \text{if } 4 \leq x \leq 5 \\ 6-x, & \text{if } 5 \leq x \leq 6 \\ 0, & \text{otherwise} \end{cases}$$

These two distributions are shown in Figure 2.5. For this choice of ρ_0 and ρ_1 two possible transport maps are:

- $f_1(x) = x + 3$
- $f_2(x) = 7 - x$

Figure 2.5 uses colours to illustrate which points of ρ_0 are mapped to which points of ρ_1.

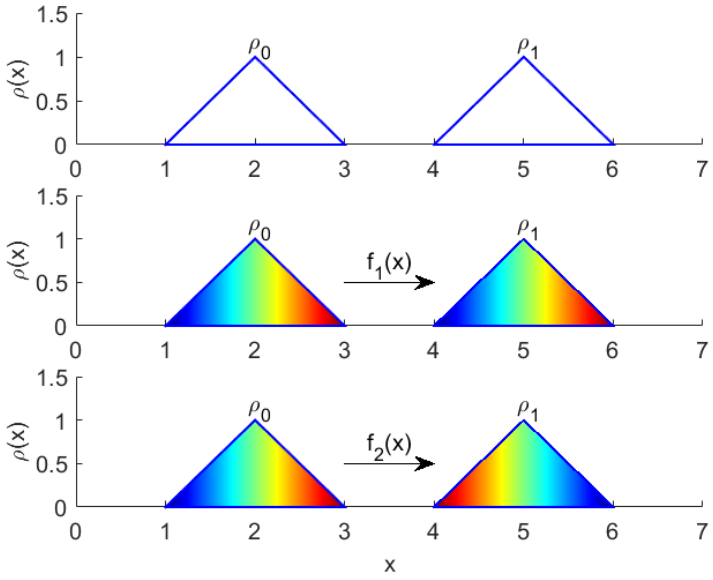


Figure 2.5: Two different transport maps $f_1(x) = x + 3$ and $f_2(x) = 7 - x$ that transport ρ_0 to ρ_1. Colours are used to illustrate how the points from ρ_0 are mapped to points of ρ_1.

From Figure 2.5 it is easy to see that f_1 and f_2 satisfy equation (2.17). Hence, a transport map is not unique. Consequently, it makes sense to introduce a cost functional and pick the transport map with the smallest cost. A possible cost functional is

$$C(f) = \mathbb{E}_{x \sim \rho_0}\left(\|f(x) - x\|_2^2\right) = \int_{\mathbb{R}} \|f(x) - x\|_2^2 \, \rho_0(x) \mathrm{d}x \,.$$

This means that the transport map f should let points $x \sim \rho_0$ travel the smallest amount of distance given $f(x) \sim \rho_1$. Calculating the cost functional for $f_1(x) = x + 3$ and $f_2(x) = 7 - x$ gives:

$$C(f_1) = \int_{\mathbb{R}} \|x + 3 - x\|^2 \rho_0(x) \mathrm{d}x = 9 \int_{\mathbb{R}} \rho_0(x) \mathrm{d}x = 9\,,$$

$$C(f_2) = \int_{\mathbb{R}} \|7 - x - x\|^2 \rho_0(x) \mathrm{d}x = \int_1^2 (7 - 2x)^2 \cdot (x - 1) \mathrm{d}x + \int_2^3 (7 - 2x)^2 \cdot (3 - x) \mathrm{d}x = 9\frac{2}{3}\,.$$

Hence, the transport map f_1 is more optimal than f_2.

Now that we have seen an example, we provide a rigorous mathematical definition of Monge's optimal transport. To accomplish this, we use [Chen et al., 2017] and [Yang and Karniadakis, 2020]. Before giving their definition, the concept of a push-forward should be defined. This definition is based on the image of a measure μ as introduced in [Bogachev, 2007] and push-forward as defined in [Yang and Karniadakis, 2020].

Definition 2.2.3 (Push-forward [Bogachev, 2007; Yang and Karniadakis, 2020])**.** Suppose two spaces X and Y are given with σ-algebras $\mathcal{A}$ and $\mathcal{B}$, respectively. Moreover, assume there is a $(\mathcal{A}, \mathcal{B})$-measurable mapping $f : X \to Y$. Letting f^{-1} be the inverse image, the measure defined by

$$f_{\#}\mu = \mu \circ f^{-1} : B \to \mu(f^{-1}(B))\,, \quad B \in \mathcal{B}$$

is called the image of the measure μ under the mapping f or the push-forward of μ through f.

Definition 2.2.4 (Monge's optimal transport [Chen et al., 2017; Yang and Karniadakis, 2020])**.** Assume a cost function $c : X \times Y \to \mathbb{R}$, $\mu \in \mathcal{P}(X)$ and $\nu \in \mathcal{P}(Y)$ are given. Moreover, define $T := \{f : f_{\#}\mu = \nu\}$ to be the set of all transport maps that map the measure μ to the measure ν. Monge's optimal transport problem is given by

$$\min_{f \in T} \mathbb{E}_\mu \left(c(x, f(x))\right).$$

The minimizer $f^* \in T$ in the above problem is called the optimal transport map.

In the above definition, a minimum is used instead of an infimum. A-priori this may not be clear. For the so-called L^2 Monge-Kantorovich problem, it can be proven that it is indeed a minimum instead of an infimum. In fact, one can show that this minimizer is unique. This result presented in Theorem 2.2.2 can be obtained by applying Theorem 1.2 of [Gangbo and McCann, 1996]. This unique existing minimizer property shows that the L^2 Monge-Kantorovich problem is a special problem. An additional property of this special problem is that it can be recast into a fluid dynamics formulation. This formulation is presented in Theorem 2.2.3.

Theorem 2.2.2 (Existence and uniqueness solution of the L^2 Monge-Kantorovich problem [Gangbo and McCann, 1996])**.** Take Monge's optimal transport problem in Definition 2.2.4 with $c(x, y) = \|x - y\|_{l^2}^2$. This problem is the so-called L^2 Monge-Kantrovich problem. Moreover, assume that μ and ν are absolutely continuous with respect to the Lebesgue measure (hence, they have probability density functions). Then there exists a unique transport map f^* that solves Monge's optimal transport problem.

Theorem 2.2.3 (L^2 Monge-Kantorovich problem in fluid dynamics form [Benamou and Brenier, 2000; Chen et al., 2016])**.** Take Monge's optimal transport problem with $c(x, y) = \|x - y\|_{l^2}^2$ and where we want to transport the measure ρ_0 to the measure ρ_T. In addition, assume that the

measures ρ_0 and ρ_T are absolutely continuous with respect to the Lebesgue measure (hence, they have probability density functions). In this case, the optimal transport problem is equivalent to

$$\begin{aligned} \min_{\rho,v} \quad & T\int_0^T \int_{\mathbb{R}^d} \rho(x,t)\,\|v(x,t)\|_{l^2}^2\,\mathrm{d}x\mathrm{d}t \\ \text{s.t.} \quad & \partial_t\rho(x,t) + \nabla\cdot(\rho(x,t)v(x,t)) = 0 \\ & \rho(x,0) = \rho_0(x)\,, \quad \rho(x,T) = \rho_T(x)\,. \end{aligned}$$

At the minimum (ρ, v) the functional value equals $\mathbb{E}_{\rho_0}\left[\|x - f^*(x)\|^2\right]$ for f^* the optimal transport map and the optimal transport map can be obtained from v. More specifically, the optimality conditions are given by

$$\begin{aligned} v(x,t) &= \nabla\phi(x,t)\,, \\ \partial_t\phi(x,t) + \frac{1}{2}\,\|\nabla\phi(x,t)\|_{l^2}^2 &= 0\,, \end{aligned}$$

and the optimal transport map f^* is determined by

$$\begin{aligned} \frac{\mathrm{d}u(x,t)}{\mathrm{d}t} &= v(x,t)\,, \quad u(x,0) = x\,, \\ f^*(x) &= u(x,T)\,. \end{aligned}$$

The above fluid dynamics formulation of optimal transport can be related to potential MFGs. To relate the two notions, choose $\mathcal{F} = 0$, $L = T\,\|v\|_{l^2}^2$ and $\nu = 0$ in Theorem 2.2.1. Then one gets the following variational problem for the corresponding potential MFG:

$$\begin{aligned} \min_{\rho,v} \quad & T\int_0^T \int_{\mathbb{R}^d} \rho(x,t)\,\|v(x,t)\|_{l^2}^2\,\mathrm{d}x\mathrm{d}t + \mathcal{G}(\rho(\cdot,T)) \\ \text{s.t.} \quad & \partial_t\rho(x,t) + \nabla\cdot(\rho(x,t)v(x,t)) = 0 \\ & \rho(x,0) = \rho_0(x)\,. \end{aligned}$$

Comparing this variational problem to the fluid dynamics formulation in Theorem 2.2.3, one sees many similarities. For instance, if one assumes $\mathcal{G}$ is a hard constraint with target measure ρ_T, the minimization problems are the same. If one supposes that $\mathcal{G}$ is a measure of similarity to a specific desired distribution ρ_T (e.g. the KL-divergence), this shows that potential MFGs allow for a relaxation of the L^2 Monge-Kantorovich optimal transport problem.

The above relationship between optimal transport and MFGs is a connection between unregularized optimal transport and deterministic MFGs. There is also a relationship between regularized optimal transport and stochastic mean field games. To make this connection explicit, the Yasue problem and the Schrödinger bridge problem need to be defined.

Definition 2.2.5 (Yasue Problem and Schrödinger bridge problem [Léger, 2019])**.** Let ρ_0 and ρ_1 be some given initial and terminal measure. The Yasue problem is defined as:

$$\begin{aligned} \min_{\rho,v} \quad & \int_0^1 \int_{\mathbb{R}^d} \frac{1}{2}\rho(x,t)\,\|v(x,t)\|_{l^2}^2 + \frac{\nu^2}{2}\,\frac{\|\nabla_x\rho(x,t)\|^2}{\rho(x,t)}\mathrm{d}x\mathrm{d}t \\ \text{s.t.} \quad & \partial_t\rho(x,t) + \nabla\cdot(\rho(x,t)v(x,t)) = 0 \\ & \rho(\cdot,0) = \rho_0\,, \quad \rho(\cdot,1) = \rho_1\,. \end{aligned}$$

On the other hand, the forward Schrödinger Bridge problem is defined as:

$$\begin{aligned} \min_{\rho,v} \quad & \int_0^1 \int_{\mathbb{R}^d} \frac{1}{2}\rho(x,t)\,\|v(x,t)\|_{l^2}^2\,\mathrm{d}x\mathrm{d}t \\ \text{s.t.} \quad & \partial_t\rho(x,t) + \nabla\cdot(\rho(x,t)v(x,t)) = \nu\Delta\rho(x,t) \\ & \rho(\cdot,0) = \rho_0\,, \quad \rho(\cdot,1) = \rho_1\,. \end{aligned}$$

In the light of Theorem 2.2.3, the Yasue problem can be interpreted as a regularized version of optimal transport. On the other hand, a stochastic potential mean field game can be interpreted as the relaxation of the Schrödinger Bridge problem via Theorem 2.2.1.

In [Léger, 2019], the Schrödinger bridge problem and the Yasue problem are related. In that work, the following is proven:

Theorem 2.2.4 (Equivalence of Yasue Problem and Schrödinger Bridge problem [Léger, 2019])**.** The Schrödinger bridge problem and the Yasue problem defined in Definition 2.2.5 are equivalent. More precisely, in terms of minimizers, the densities ρ obtained in both problems are the same.

Hence, in terms of the distribution ρ, solving the Schrödinger Bridge problem is equivalent to solving the Yasue problem. By our earlier discussion, relaxing the final constraint $\rho(\cdot,1) = \rho_1$ in both problems and adding an appropriate terminal loss function (e.g. the KL-divergence) to the loss functional, one can see that solving a particular stochastic MFG amounts to solving a relaxed regularized optimal transport problem. Regularizing an optimal transport problem might be beneficial. For instance, the regularity could introduce a trade-off that prevents overfitting on data measures [Paty and Cuturi, 2020].

2.2.4 Mean field games as a game between two players

This thesis introduced MFGs as a game between an infinite number of infinitesimal agents that each has a stochastic optimal control problem. Using a suitable reformulation, one can see the MFG as a game between two players. This two-player game formulation is used in [Lin et al., 2020] to solve a stochastic MFG and, under certain assumptions, is proven in [Cirant and Nurbekyan, 2018]. Our solution algorithm for stochastic potential MFGs, which is introduced in Chapter 3, uses a similar procedure as the algorithm presented in [Lin et al., 2020]. More concretely, it uses a two-player game formulation to solve the MFG. As the used two-player game formulation is similar to the two-player game formulation in [Cirant and Nurbekyan, 2018], this section discusses the results of [Cirant and Nurbekyan, 2018].

Letting $\mathbb{T}^d = [0,1]^d$ and T the final time, consider the following PDE perspective of a typical MFG system:

$$\begin{aligned} -\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t),\rho(x,t)) = 0 \\ \rho_t(x,t) - \nu\Delta\rho(x,t) - \nabla\cdot(\rho\nabla_p H(x,\nabla\phi(x,t),\rho(x,t))) = 0 \\ \rho > 0\,,\ \rho(x,0) = \rho_0(x)\,,\ \phi(x,T) = G(x),\ (x,t)\in\mathbb{T}^d\times[0,T]\,. \end{aligned} \tag{2.18}$$

where $H : \mathbb{T}^d\times\mathbb{R}^d\times\mathbb{R}_{++}\to\mathbb{R}$. To treat results regarding this PDE system, [Cirant and Nurbekyan, 2018] introduces several functionals and notations. First, the paper defines $\Omega_T = \mathbb{T}^d\times[0,T]$ and

$$F_H(x,p,\rho) = \int_0^\rho H(x,p,z)\mathrm{d}z\,.$$

In addition, the authors define two functionals that are used inside the game between the two players. Letting $\phi_t := \frac{\partial \phi}{\partial t}$, these functionals are given by:

$$\begin{aligned}\Psi_1(\rho,\phi) &= \int_{\Omega_T} (-\phi_t(x,t) - \nu\Delta\phi(x,t))\rho(x,t) + F_H(x,\nabla\phi(x,t),\rho(x,t))\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{T}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \int_{\mathbb{T}^d} \rho_0(x)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho(x,T)G(x)\mathrm{d}x \\ &= \int_{\Omega_T} (\rho_t(x,t) - \nu\Delta\rho(x,t))\phi(x,t) + F_H(x,\nabla\phi(x,t),\rho(x,t))\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{T}^d} \rho(x,0)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho_0(x)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho(x,T)G(x)\mathrm{d}x\end{aligned} \tag{2.19}$$

$$\begin{aligned}\Psi_2(\rho,\phi) &= \int_{\Omega_T} \left(-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t),\rho(x,t))\right)\rho(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{T}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \int_{\mathbb{T}^d} \rho_0(x)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho(x,T)G(x)\mathrm{d}x \\ &= \int_{\Omega_T} \left(\rho_t(x,t) - \nu\Delta\rho(x,t)\right)\phi(x,t) + H(x,\nabla\phi(x,t),\rho(x,t))\rho(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{T}^d} \rho(x,0)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho_0(x)\phi(x,0)\mathrm{d}x - \int_{\mathbb{T}^d} \rho(x,T)G(x)\mathrm{d}x\end{aligned} \tag{2.20}$$

Besides these definitions, the paper by Cirant and Nurbekyan [2018] makes the following assumptions about the MFG system in equation (2.18):

- $H \in C^2(\mathbb{T}^d \times \mathbb{R}^d \times \mathbb{R}_{++})$ where $\mathbb{R}_{++}$ are the positive numbers.
- $\rho_0, G \in C^2(\mathbb{T}^d)$.
- $\rho_0 > 0$.
- $\mathcal{H}_p(x,p,\rho) \succcurlyeq 0$ for all $(x,p,\rho) \in \mathbb{T}^d \times \mathbb{R}^d \times \mathbb{R}_{++}$ where $\mathcal{H}_p(x,p,\rho)$ is the hessian of H with respect to p.
- $\frac{\partial}{\partial \rho} H(x,p,\rho) \leq 0$ for all $(x,p,\rho) \in \mathbb{T}^d \times \mathbb{R}^d \times \mathbb{R}_{++}$.

Given the above assumptions and notations, the [Cirant and Nurbekyan, 2018] paper proves the following statement

Theorem 2.2.5 (Two-player game formulation [Cirant and Nurbekyan, 2018])**.** For any $\rho^*, \phi^* \in C^2(\Omega_T)$ such that $\rho^* > 0$, one has

$$\begin{aligned}\Psi_1(\rho^*,\phi^*) &= \sup_{\rho \in C^2(\Omega_T), \rho > 0} \left(\Psi_1(\rho,\phi^*)\right), \\ \Psi_2(\rho^*,\phi^*) &= \inf_{\phi \in C^2(\Omega_T)} (\Psi_2(\rho^*,\phi))\end{aligned}$$

if and only if (ρ^*, ϕ^*) is a smooth solution to equation (2.18). Here Ψ_1 and Ψ_2 are given by equations (2.19) and (2.20), respectively.

Remark 2.2.1. In proving Theorem 2.2.5, Cirant and Nurbekyan [2018] use integration by parts. For instance, they use integration by parts to go from the first equality in equations (2.19) and (2.20) to the second equality. When one carefully inspects the transition from the first to the second equality, one can see that the integration by parts is not done entirely correct. Due to integration by parts, several spatial boundary integrals should appear. However, these do not appear in equations (2.19) and (2.20). The authors do not say a word on why they omit the missing spatial boundary integrals. The book by Gomes et al. [2016] mentions that to avoid difficulties, it is common to use $\mathbb{T}^d \times [0,T]$ as the spatial-time domain in combination with periodic boundary conditions. As the [Cirant and Nurbekyan, 2018] paper treats $\mathbb{T}^d = [0,1]^d$ as spatial domain and the boundary integrals vanish when using periodic boundary conditions, the authors most likely proved the theorem assuming periodic boundary conditions for both the Hamilton-Jacobi-Bellman equation and the Fokker-Planck equation. For our purposes for Theorem 2.2.5, periodic boundary conditions are sufficient. Hence, we should keep in mind that in Theorem 2.2.5 periodic boundary conditions to the MFG system apply.

Remark 2.2.2. In the previous remark, it is mentioned that [Cirant and Nurbekyan, 2018] and many other works use $\mathbb{T}^d \times [0,T]$ as domain in combination with periodic boundary conditions. When deriving the MFG framework in Section 2.2.1, an agent view is used. This derivation is based on a domain $\mathbb{R}^d$. When restricting to a compact domain and using periodic boundary conditions, it is not clear whether the same derivation applies. For instance, as one works with a bounded domain, the position of the agent should stay in this domain. When using an SDE, it could be that the agent goes over the boundary, which is not desired. Hence, the agent dynamics should be altered in one way or the other. When using periodic boundary conditions, it would make sense to let the agent appear at the opposite boundary. This change to the agent dynamics is introduced by means of random walks in Section 4.1 of [Pavliotis, 2014]. However, in this case, one has to figure out the effect on the stochastic optimal control problem and the corresponding Hamilton-Jacobi-Bellman equations. Moreover, as the correspondence between agent dynamics and Fokker-Planck is derived for $x \in \mathbb{R}^d$, one needs to figure out the changes (if any) to this relationship.

On the other hand, when one chooses $[-a,a]^d$ as the domain for very large a and makes sure that the initial distribution has most of its mass in a relatively small subset, the agent dynamics most likely does not come close to the boundary at all. In this case, one does not have to deal with the boundary and one can just use the original agent dynamics. Although not mathematically rigorous, in solution algorithms, the sample paths of the agents most likely will not be close to the boundary given a is sufficiently large. Hence, restricting to a compact domain would not alter the solution algorithms and one does not have to deal with a change in the agent dynamics.

All in all, the above discussion shows that when using Theorem 2.2.5, we have to be aware of a gap between this theorem and the intuitive derivation of the MFG model in Section 2.2.1.

Theorem 2.2.5 nicely illustrates the connection between MFGs and two-player games. The game in Theorem 2.2.5 is not necessarily a zero-sum game as $-\Psi_1 + \Psi_2$ is not always equal to zero. The paper by Cirant and Nurbekyan [2018] has an additional result similar to Theorem 2.2.5 that does involve a zero-sum game.

To get this zero-sum game result, we need a specific choice of the Hamiltonian H. This specific choice is $H(x,p,\rho) = H_0(x,p) - F(x,\rho)$. Given this choice, define

$$\mathcal{F}(x,\rho) = \int_0^\rho F(x,z)\mathrm{d}z\,.$$

Using this definition and the specific choice of H, the formula's for Ψ_1 and Ψ_2 become

$$\begin{aligned}\Psi_1(\rho,\phi) &= \int_{\Omega_T}(-\phi_t(x,t)-\nu\Delta\phi(x,t))\rho(x,t)+H_0(x,\nabla\phi(x,t))\rho(x,t)-\mathcal{F}(x,\rho(x,t))\mathrm{d}x\mathrm{d}t\\ &+\int_{\mathbb{T}^d}\rho(x,T)\phi(x,T)\mathrm{d}x-\int_{\mathbb{T}^d}\rho_0(x)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho(x,T)G(x)\mathrm{d}x\\ &=\int_{\Omega_T}(\rho_t(x,t)-\nu\Delta\rho(x,t))\phi(x,t)+H_0(x,\nabla\phi(x,t))\rho(x,t)-\mathcal{F}(x,\rho(x,t))\mathrm{d}x\mathrm{d}t\\ &+\int_{\mathbb{T}^d}\rho(x,0)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho_0(x)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho(x,T)G(x)\mathrm{d}x\end{aligned} \tag{2.21}$$

$$\begin{aligned}\Psi_2(\rho,\phi) &= \int_{\Omega_T}\left(-\phi_t(x,t)-\nu\Delta\phi(x,t)\right)\rho(x,t)+\left(H_0(x,\nabla\phi(x,t))-F(x,\rho(x,t))\right)\rho(x,t)\mathrm{d}x\mathrm{d}t\\ &+\int_{\mathbb{T}^d}\rho(x,T)\phi(x,T)\mathrm{d}x-\int_{\mathbb{T}^d}\rho_0(x)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho(x,T)G(x)\mathrm{d}x\\ &=\int_{\Omega_T}\left(\rho_t(x,t)-\nu\Delta\rho(x,t)\right)\phi(x,t)+\left(H_0(x,\nabla\phi(x,t))-F(x,\rho(x,t))\right)\rho(x,t)\mathrm{d}x\mathrm{d}t\\ &+\int_{\mathbb{T}^d}\rho(x,0)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho_0(x)\phi(x,0)\mathrm{d}x-\int_{\mathbb{T}^d}\rho(x,T)G(x)\mathrm{d}x\end{aligned} \tag{2.22}$$

To get the result involving a zero-sum game, we want to use Theorem 2.2.5. To use it, several assumptions are needed. First, assume $H_0 \in C^2(\mathbb{T}^d\times\mathbb{R}^d)$ and $F\in C^1(\mathbb{T}^d\times\mathbb{R}_{++})$. Moreover, letting $\mathcal{H}_{0p}$ be the hessian of H_0 with respect to p, assume for all $(x,p,\rho)\in\mathbb{T}^d\times\mathbb{R}^d\times\mathbb{R}_{++}$ that $\mathcal{H}_{0p}(x,p)\succcurlyeq 0$ and $\frac{\partial}{\partial\rho}F(x,\rho)\geq 0$. Using these assumptions, almost all the assumptions of theorem 2.2.5 are met. The only assumption that is not necessarily met is $H\in C^2(\mathbb{T}^d\times\mathbb{R}^d\times\mathbb{R}_{++})$. Even though this assumption is used in Theorem 2.2.5, the provided assumptions on H_0 and F are sufficient for using Theorem 2.2.5.

Theorem 2.2.5 asserts that an optimal ϕ^* is a minimizer of $\Psi_2(\rho^*,\phi)$. Therefore, if a function that only depends on ρ is added to Ψ_2, this does not change the minimization problem in ϕ. Comparing equations (2.21) and (2.22), Ψ_2 differs from Ψ_1 by only a ρ dependent term: $\int_{\Omega_T}F(x,\rho(x,t))\rho(x,t)-\mathcal{F}(x,\rho(x,t))\mathrm{d}x\mathrm{d}t$. Adding $\int_{\Omega_T}F(x,\rho(x,t))\rho(x,t)-\mathcal{F}(x,\rho(x,t))\mathrm{d}x\mathrm{d}t$ to Ψ_2 and following the previous reasoning, one gets:

Theorem 2.2.6 (Two-player zero-sum game formulation [Cirant and Nurbekyan, 2018])**.** For any $\rho^*,\phi^*\in C^2(\Omega_T)$ such that $\rho^*>0$, one has

$$\begin{aligned}\Psi_1(\rho^*,\phi^*) &= \sup_{\rho\in C^2(\Omega_T),\rho>0}\left(\Psi_1(\rho,\phi^*)\right),\\ \Psi_1(\rho^*,\phi^*) &= \inf_{\phi\in C^2(\Omega_T)}(\Psi_1(\rho^*,\phi))\end{aligned}$$

if and only if (ρ^*,ϕ^*) is a smooth solution to equation (2.18) with $H(x,p,\rho)=H_0(x,p)-F(x,\rho)$. Here Ψ_1 is given by equation (2.21).

From this formulation, a saddle point of Ψ_1 results in the solution of the MFG. In our solution algorithm for stochastic potential MFGs, which is introduced in Chapter 3, we use a similar saddle point problem to obtain the solution.

Chapter 3

The solution algorithm for potential MFGs

To investigate the advantages of neural differential equations in solving stochastic potential MFGs, we solve several MFGs in Chapter 4. To solve these potential MFGs, a solution algorithm is needed. This chapter uses information from Chapter 2 and discusses the used algorithm to solve stochastic potential MFGs. First, the used potential MFG model is introduced together with a corresponding two-player game formulation. This two-player game formulation is similar to the ones in Section 2.2.4 and is used to obtain the solution to the potential MFG. Subsequently, we give a general outline of the solution algorithm. Within this algorithm, a functional has to be calculated. Two different ways of calculating this functional are discussed. One is only applicable to a specific type of running cost $\mathcal{F}$ and terminal cost $\mathcal{G}$, while the other is more widely applicable.

3.1 The used mean field game model

For solving stochastic potential MFGs, the starting point is the model used in [Lin et al., 2020]. This model is given by:

$$\begin{aligned} -\partial_t \phi(x,t) - \nu \Delta \phi(x,t) + H(x, \nabla \phi(x,t)) &= F(x, \rho(\cdot,t)) \\ \partial_t \rho(x,t) - \nu \Delta \rho(x,t) - \nabla \cdot (\rho(x,t) \nabla_p H(x, \nabla \phi(x,t))) &= 0 \\ \rho(x,0) = \rho_0(x), \quad \phi(x,T) &= G(x, \rho(\cdot,T)) \,, \end{aligned} \tag{3.1}$$

where F and G are variational derivatives of functionals $\mathcal{F} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$ and $\mathcal{G} : \mathcal{P}(\mathbb{R}^d) \to \mathbb{R}$, respectively. The first equation is a Hamilton-Jacobi-Bellman equation (HJB) and the second equation is a Fokker-Planck equation (FP). Moreover, the earlier mentioned paper uses the following definition of the Hamiltonian H:

$$H(x,p) = \sup_v \left(-\langle p, v \rangle - L(x,v) \right) \,, \tag{3.2}$$

where L is some cost function. For this Hamiltonian, the Fokker-Planck equation in the MFG corresponds to an agent with the following dynamics:

$$\mathrm{d}z = v(z,t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t \,, \; z \sim \rho_0 \,,$$

where W_t is a Brownian motion and at optimality $v(z,t) = -\nabla_p H(z, \nabla \phi(z,t))$.

To find a solution to the paired Hamilton-Jacobi-Bellman and Fokker-Planck equations, we use the approach proposed in [Lin et al., 2020] where a min-max problem is used. Such a two-player game formulation is proven in, for instance, [Cirant and Nurbekyan, 2018]. The results of this paper are already discussed in Section 2.2.4.

In the used two-player game formulation, the following functional is used:

$$\begin{aligned}\Psi(\rho,\phi) = &\int_0^T \left\{ \int_{\mathbb{R}^d} \left(-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t))\right)\rho(x,t)\mathrm{d}x - \mathcal{F}(\rho(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi(x,0)\mathrm{d}x - \mathcal{G}(\rho(\cdot,T))\,,\end{aligned} \tag{3.3}$$

where $\Omega_T = \mathbb{R}^d \times [0,T]$ and $\mathcal{F}$ and $\mathcal{G}$ are functionals with variational derivatives F and G, respectively. Similar to what we have seen in Section 2.2.4, the solution to the MFG system in equation (3.1) can be characterized by the solution of a two-player zero-sum game based on Ψ. This fact is used in [Lin et al., 2020] to devise an algorithm that aims to find the solution. Hence, similar to that paper, this thesis uses the following two-player game to solve the MFG system:

$$\begin{aligned}\Psi(\rho^*,\phi^*) &= \sup_{\rho}(\Psi(\rho,\phi^*)) \\ \Psi(\rho^*,\phi^*) &= \inf_{\phi}(\Psi(\rho^*,\phi))\end{aligned} \tag{3.4}$$

In the used MFG model, this thesis uses as spatial domain $\Omega = \mathbb{R}^d$. The reason is that the agent dynamic derivation of the MFG system makes the most sense for $\Omega = \mathbb{R}^d$. This is discussed in Remark 2.2.2. However, as remarked in Remark 2.2.1, one has to note that the results in Section 2.2.4 used to arrive at equation (3.4) are derived for situations with a bounded domain Ω and periodic boundary conditions. In the $\Omega = \mathbb{R}^d$ case, these results do not immediately extend. What one can prove is the following theorem:

Theorem 3.1.1. Assume the MFG PDE system in equation (3.1) is given with $\rho_0(x) \in C^2(\mathbb{R}^d)$. Defining $C_c^2(\mathbb{R}^d)$ as the set of twice continuously differentiable functions with compact support, let F and G be functionals that are defined via:

- $\lim_{h\to 0} \frac{\mathcal{F}(\rho+hm)-\mathcal{F}(\rho)}{h} = \int_{\mathbb{R}^d} F(x,\rho)m(x)\mathrm{d}x$ for all $m \in C_c^2(\mathbb{R}^d)$. This is a variational derivative of some functional $\mathcal{F} : C^2(\mathbb{R}^d) \to \mathbb{R}$. We assume $\mathcal{F}$ is bounded.
- $\lim_{h\to 0} \frac{\mathcal{G}(\rho+hm)-\mathcal{G}(\rho)}{h} = \int_{\mathbb{R}^d} G(x,\rho)m(x)\mathrm{d}x$ for all $m \in C_c^2(\mathbb{R}^d)$. This is a variational derivative of some functional $\mathcal{G} : C^2(\mathbb{R}^d) \to \mathbb{R}$. We assume $\mathcal{G}$ is bounded.

Assume that H is continuous in both arguments. Moreover, assume that the functionals $F(x,\rho) : \mathbb{R}^d \times C^2(\mathbb{R}^d)$ and $G(x,\rho) : \mathbb{R}^d \times C^2(\mathbb{R}^d)$ are continuous in x given ρ. If $(\rho^*,\phi^*) \in C^2(\Omega_T) \times C^2(\Omega_T)$ is a saddle point of the functional Ψ given in equation (3.3):

$$\begin{aligned}\Psi(\rho^*,\phi^*) &= \sup_{\rho\in C^2(\Omega_T)} (\Psi(\rho,\phi^*))\,, \\ \Psi(\rho^*,\phi^*) &= \inf_{\phi\in C^2(\Omega_T)} (\Psi(\rho^*,\phi))\,,\end{aligned}$$

and $\Psi(\rho^*,\phi^*)$ is finite, then (ρ^*,ϕ^*) is a smooth solution to the PDE system.

Remark 3.1.1. A-priori, it might not be clear what type of functionals $\mathcal{F}$ and $\mathcal{G}$ return a finite value for all $C^2(\mathbb{R}^d)$ functions and are useful in the MFG framework. As an example of such a

functional $\mathcal{F}$, we present a modification of the entropy running cost used to model crowd aversion in Section 4.2. This modification is given by:

$$\mathcal{F}(\rho) = \beta\left(\int_{\mathbb{R}^d} \sigma(\rho(x))\log(\sigma(\rho(x)) + \delta)\mathrm{d}x\right). \tag{3.5}$$

where δ is a small positive constant that makes sure that the variational derivative is still continuous. When using $\beta(x) = \sigma(x) = x$, the entropy running cost is obtained. However, it is not properly defined for all functions in $C^2(\mathbb{R}^d)$. To make the function properly defined, $\beta(x)$ and $\sigma(x)$ can be suitably chosen.

For instance, $\beta(x)$ can be defined as a smooth cut-off function. For a small positive scalar ϵ and an upper bound M, $\beta(x)$ can be defined as:

$$\beta(x) = \begin{cases} x, & \text{if } |x| < M - \epsilon \\ \mathrm{sgn}(x)M, & \text{if } |x| > M \\ \tilde{\beta}(x), & \text{if } M - \epsilon \leq |x| \leq M \end{cases}$$

with $\tilde{\beta}(x)$ a smooth function that satisfies $M - \epsilon \leq |\tilde{\beta}(x)| \leq M$ for $M - \epsilon \leq |x| \leq M$, $\beta(x) = \mathrm{sgn}(x)(M - \epsilon)$ if $|x| = M - \epsilon$, and $\beta(x) = \mathrm{sgn}(x)M$ if $|x| = M$. Moreover, when the integral input in equation (3.5) is undefined, we assume the function β returns the value M. In addition, to ensure that similar to probability distributions the inputted functions are always positive, $\sigma(x)$ can be chosen as a smooth approximation to the absolute value function.

Using the choices for β and σ, the functional in equation (3.5) satisfies the requirements of Theorem 3.1.1 and can be used. Moreover, it serves as a suitable replacement of the entropy running cost for sufficiently large M, sufficiently small ϵ, and sufficiently small δ.

This theorem is proven in Appendix C. Hence, one can prove a necessary condition. The sufficient condition is less straightforward. The main difficulty stems from the arising boundary integrals that may not vanish when evaluating them on the solution of the MFG system. However, if the solution ρ and $\nabla_x \rho$ tend to zero quick enough as $\|x\| \to \infty$, these boundary integrals vanish and one can prove a sufficient condition. Theorem 4.1 of [Pavliotis, 2014] gives conditions when a solution to a Fokker-Planck equation decays exponentially in the spatial coordinates. Hence, it may be that under certain conditions the distribution decays very fast, implying the existence of a sufficient condition. The necessary condition, the possible sufficient condition in combination with Remark 2.2.2 motivates the use of the two-player game in $\mathbb{R}^d$ to find the solution of the MFG.

Furthermore, one has to note that the results in Section 2.2.4 do not incorporate the potential MFG scenario. For instance, equations (2.21) and (2.22) only have a $\int_{\mathbb{T}^d} \rho(x,T)G(x)\mathrm{d}x$ term. To have a more general potential MFG, one needs to replace this with a functional $\mathcal{G}(\rho(\cdot,T))$. This change does not pose any additional difficulties in the proofs. As a consequence, the results in Section 2.2.4 still hold in the potential MFG regime. To get an idea on how to include the more general potential MFG terms, see the proof of Theorem 3.1.1 in Appendix C.

3.2 A general outline of the algorithm

This section discusses the general structure of the algorithm that solves the problem described by equation (3.4). First, we elaborate on the meaning of a parametrization of the generator for the agent position distribution. Subsequently, the general structure of the solution algorithm is treated. The pseudo-code of the algorithm is given as well. Finally, as the algorithm uses neural

network parametrizations, we discuss several possible parametrizations. This discussion is mainly aimed to introduce the used parametrizations of the generator for the agent position distribution. Introducing these parametrizations also gives a better intuition on the meaning of a generator for the agent position distribution.

As mentioned in the introduction of this section, before going into the algorithm structure, we elaborate on what we mean by a parametrization of the generator for the agent position distribution $\rho(x,t)$. In our problem setting, only the initial distribution $\rho_0(x)$ is known and we want to find the optimal agent position distribution $\rho^*(x,t)$. For our algorithm it suffices to have a function that generates samples from the optimal agent distribution. Hence, we want to find this optimal generator. To do so, we parametrize a function that samples points from $\rho_0(x)$ at initial time $t = t_0 = 0$ and at any other time can generate any points. This generator of points is trained such that after training, the generator samples points from the optimal agent distribution $\rho^*(x,t)$. Hence, at time t, this trained generator samples points from $\rho(x,t) \approx \rho^*(x,t)$.

As it is clear what we mean with a parametrization of the generator for the agent position distribution, we can continue with the general structure of the algorithm. This general structure is the same as in the work of Lin et al. [2020]. In their paper, a GAN-like training procedure is used in combination with several neural networks to solve the two-player game. More concretely, one first parametrizes both the value function ϕ of equation (3.1) and the generator that produces samples from the agent distribution ρ. Subsequently, the training of the value function ϕ and the generator for ρ are continuously alternated. Training in this way resembles the saddle point structure in equation (3.4) and hopefully returns a decent solution.

To summarize, the general outline of the algorithm is given by:

Algorithm 1: General outline stochastic MFG solution algorithm

Result: Approximation of ϕ and generator $\mathbb{G}$ for ρ where (ϕ, ρ) satisfies the stochastic MFG system in equation (3.1).

Input: Number of switches between training ϕ and $\mathbb{G}$, number of iterations to train ϕ, number of iterations to train $\mathbb{G}$

Initializations: $\phi_{N_\theta}(x,t)$, $\mathbb{G}_{N_\theta}$.

for *Max number of switches* **do**
- `/* Train the agent distribution` ρ_{N_θ} `by training its generator` $\mathbb{G}_{N_\theta}$ `*/`
- **for** *Number of training iterations generator* $\mathbb{G}_{N_\theta}$ **do**
  - Calculate Ψ given in equation (3.3) only with the terms depending on ρ;
  - Calculate the gradients with respect to the NN weights of the generator;
  - Adjust the weights with a specific optimizer (e.g. ADAM) to maximize Ψ;
- **end**
- `/* Train the value function` ϕ_{N_θ} `*/`
- **for** *Number of training iterations value function* ϕ_{N_θ} **do**
  - Calculate Ψ given in equation (3.3) only with the terms depending on ϕ;
  - Calculate the gradients with respect to the NN weights of the value function ϕ_{N_θ};
  - Adjust the weights with a specific optimizer (e.g. ADAM) to minimize Ψ;
- **end**

end

The parametrization of the value function ϕ and the generator, which are needed for the above algorithm, are not specified yet. The parametrization of ϕ that we use in the numerical experiments

is introduced in Section 3.3. For the generator, several sensible parametrization choices exist. All the parametrization choices for the generator that we discuss now are explored in the numerical experiments.

First of all, a standard neural network could be used as done in [Lin et al., 2020]. One motivation for this choice is the easy calculation of samples. To make such a standard neural network generator, [Lin et al., 2020] parametrizes the generator as

$$\mathbb{G}_{N_\theta}(z,t) = \frac{T-t}{T}z + \frac{t}{T}N_\theta(z,t)\,, \tag{3.6}$$

where $z \sim \rho_0$ and $N_\theta(z,t)$ is the output of a standard neural network, such as a simple feedforward neural network or a ResNet. The latter neural network takes as input the initial position and the time t at which we want to generate a sample. Hence, our standard neural network generator generates samples at time t by first generating an initial point $z_0 \sim \rho_0$ and then calculating $\mathbb{G}_{N_\theta}(z_0,t)$.

Another generator choice is a neural SDE. By our discussion in Section 3.1, the solution distribution ρ^* is a solution to a Fokker-Planck equation belonging to an SDE agent dynamics. This agent dynamics completely determines the agent distribution ρ^*. The reason is that given an initial position $x \sim \rho_0$, the dynamics completely determines where the agents can be. As a consequence, the generator for the distribution ρ^* first samples initial points $x_0 \sim \rho_0$ and subsequently solves the agent dynamics SDE given the initial condition x_0. Following this reasoning, it makes sense to use a neural SDE as the generator parametrization.

When dealing with the functional calculation algorithm for the more general case in Section 3.4, we also make a neural ODE a sensible choice. To generate samples with a neural ODE, the same procedure is used as used with a neural SDE.

3.3 Calculating Ψ for specific running and terminal costs

To update the value function ϕ and the generator $\mathbb{G}$ in Algorithm 1, the Ψ functional, which is given in equation (3.3), needs to be calculated. As Ψ can not be calculated analytically, it needs to be approximated. When the running and terminal costs are of a specific form, we estimate Ψ using the estimation approach introduced in [Lin et al., 2020]. In this case, Algorithm 1 becomes the stochastic MFG solution algorithm from [Lin et al., 2020]. This section first defines the specific form of the running and terminal cost for which we use the approximation method of [Lin et al., 2020]. Subsequently, we introduce the approximation method.

The specific form of the running cost $\mathcal{F}$ and the terminal cost $\mathcal{G}$ in equation (3.3) are

$$\begin{aligned} \mathcal{F}(\rho) &= \int_{\mathbb{R}^d} F(x)\rho(x)\mathrm{d}x\,, \\ \mathcal{G}(\rho) &= \int_{\mathbb{R}^d} G(x)\rho(x)\mathrm{d}x\,. \end{aligned} \tag{3.7}$$

Specific examples are:

$$\begin{aligned} \mathcal{F}(\rho) &= \int_{\mathbb{R}^d} \epsilon_1 \eta\left(\epsilon_2 \left\|x - \tilde{f}(x)\right\|^2\right)\rho(x)\mathrm{d}x\,, \\ \mathcal{G}(\rho(\cdot,T)) &= \int_{\mathbb{R}^d} \|x - x_T\|\,\rho(x,T)\mathrm{d}x\,, \end{aligned}$$

where $\tilde{f}(x)$ is some function, η is the mollifier function introduced in Appendix B and the ϵ's are scaling factors. Here the terminal cost $\mathcal{G}$ indicates that in expectation the agents should be close

to the point x_T at final time T. The $\mathcal{F}$ cost models walls. When a trajectory is very close to the wall parametrized by $\tilde{f}(x)$, a significant cost is incurred. However, when one is far enough from this wall, no cost is incurred at all.

To approximate the Ψ functional in equation (3.3), note that for this specific type of running and terminal cost, almost the whole functional Ψ turns into an expectation. To see this, substitute the particular class of loss functions into equation (3.3). This yields:

$$\begin{aligned}\Psi(\rho,\phi) &= \int_0^T \int_{\mathbb{R}^d} (-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t)) - F(x))\rho(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho(x,T)(\phi(x,T) - G(x))\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi(x,0)\mathrm{d}x \\ &= \int_0^T \mathbb{E}_{x\sim\rho(\cdot,t)} \left[-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t)) - F(x)\right] \mathrm{d}t \\ &+ \mathbb{E}_{x\sim\rho(\cdot,T)} \left[\phi(x,T) - G(x)\right] - \mathbb{E}_{x\sim\rho_0(\cdot)} \left[\phi(x,0)\right]\end{aligned} \tag{3.8}$$

From the last equation, it is easily seen that the functional is a combination of expectations and time integrals. As a consequence, to achieve our goal of approximating Ψ, it might be sensible to use a Monte Carlo algorithm that approximates the expectations and an integral approximation method that estimates the time integral. This is precisely what [Lin et al., 2020] does. For more information on Monte Carlo simulation, see Appendix A.2.

Moreover, [Lin et al., 2020] observes from the MFG system in equation (3.1) that the value function should satisfy $\phi(x,T) = \frac{\partial}{\partial\rho}\mathcal{G}(x,\rho) = G(x)$. As G is known, it may be beneficial to adjust our ϕ neural network parametrization to include knowledge about G. To do this, [Lin et al., 2020] uses the following neural network parametrization for the value function:

$$\phi_{N_\theta}(x,t) = \frac{T-t}{T}\tilde{\phi}(x,t) + \frac{t}{T}G(x) \tag{3.9}$$

with $N_\theta = \tilde{\phi}$ a neural network. In our case, $\tilde{\phi}$ is parametrized as the ResNet architecture from [Ruthotto et al., 2020].

The architecture used in [Ruthotto et al., 2020] consists of two components: a ResNet and a polynomial part. Let $N(s,\Theta)$ be the output of the ResNet with weights Θ. Assuming an input $s \in \mathbb{R}^{d+1}$ with as first d components the spatial coordinates and the last component the time, the ResNet calculates the corresponding output according to

$$\begin{aligned} u_0 &= \sigma(K_0 s + b_0)\,, \\ u_1 &= u_0 + h\sigma(K_1 u_0 + b_1)\,, \\ &\vdots \qquad\qquad \vdots \\ u_M &= u_{M-1} + h\sigma(K_M u_{M-1} + b_M)\,, \\ N(s,\Theta) &= u_M\,. \end{aligned}$$

Here $\sigma : \mathbb{R} \to \mathbb{R}$ is an element-wise function and $h > 0$ a fixed step size. Furthermore, letting m be the number of neurons in each hidden layer, the parameters Θ of this ResNet are the matrix $K_0 \in \mathbb{R}^{m\times(d+1)}$, the matrices $K_i \in \mathbb{R}^{m\times m}$ and the bias vectors $b_i \in \mathbb{R}^m$. The output $N(s,\Theta) \in \mathbb{R}^m$ of the ResNet is not the output of the architecture. As mentioned, several polynomial operations still need to be performed to get the output. More specifically, the output is calculated via:

$$\tilde{\phi}(s) = w^T N(s,\Theta) + \frac{1}{2}s^T(A + A^T)s + c^T s + b\,,$$

where the weights are $w \in \mathbb{R}^m$, $c \in \mathbb{R}^{d+1}$, $b \in \mathbb{R}$, $A \in \mathbb{R}^{(d+1)\times(d+1)}$ and Θ .

Using the parametrization ϕ_{N_θ} given in equation (3.9), the final time condition $\phi(x,T) = G(x)$ for the value function is satisfied. Furthermore, this specific choice of parametrization has an effect on the functional in equation (3.8), which is the functional we set out to approximate in this section. As $\phi_{N_\theta}(x,T) = G(x)$, the $\mathbb{E}_{x\sim\rho(\cdot,T)}\left[\phi(x,T) - G(x)\right]$ term vanishes. Hence, the functional we need to approximate is given by

$$\begin{aligned}\Psi(\rho,\phi) = \int_0^T \mathbb{E}_{x\sim\rho(\cdot,t)}\left[-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t)) - F(x)\right]\mathrm{d}t \\ - \mathbb{E}_{x\sim\rho_0(\cdot)}\left[\phi(x,0)\right] .\end{aligned} \tag{3.10}$$

In this formula, the generator only appears in the first term via $\rho(\cdot,t)$, while the value function ϕ appears in both terms. Consequently, we only need to approximate the first term in Ψ for updating the generator. For updating the value function ϕ, all the terms need to be approximated.

Using the preceding observations, the important terms of the Ψ functional can be calculated when we use Algorithm 1 to update the generator or the value function ϕ. First of all, the expectations in equation (3.10) are approximated via Monte Carlo simulation. Using the approximation of the first expectation, we can approximate the time integral by any integral approximation method. For our purposes, the trapezoidal rule is used with equal time intervals. To give pseudo-code of the approximation algorithms, define R_1 and R_2 by:

$$R_1(z,t) := -\partial_t\phi(z,t) - \nu\Delta\phi(z,t) + H(z,\nabla\phi(z,t)) - F(z)\,,$$

$$R_2(z,t) := -\partial_t\phi(z,t) - \nu\Delta\phi(z,t) + H(z,\nabla\phi(z,t))\,.$$

Using these definitions with the trapezoidal rule, we obtain the following two algorithms:

Algorithm 2: Special approximation of the functional used in Algorithm 1 to update the generator $\mathbb{G}$. This algorithm can only be used when the running and terminal costs $\mathcal{F}$ and $\mathcal{G}$ present in equation (3.3) are given by equation (3.7).

Result: The functional value used for updating the generator $\mathbb{G}$.

Input: Number of samples N, number of times M, initial distribution ρ_0

Produce a set $B := \{x_i\}_{i=1}^N$ of N samples from ρ_0;
Create equidistant times t_j for $j = 0, 1, \ldots, M+1$ at which one wants to generate observations (including $t_0 = 0$ and $t_{M+1} = T$);
for *every sample* x_i **do**
  for *every time* t_j **do**
    Generate a sample at time t_j given initial condition x_i by means of the generator: $z(x_i,t_j) = \mathbb{G}(x_i,t_j)$ (Note: $\mathbb{G}(x,0) = x$ should hold). For examples of generators, see Section 3.2;
  end
end
Defining $I_j = \frac{1}{2}$ if $j \in \{0, M+1\}$ and $I_j = 1$ if $j \in \{1,\ldots,M\}$, approximate the functional that only consists of the ϕ dependent terms in the functional Ψ using:

$$\frac{1}{N(M+1)}\sum_{i=1}^{N}\sum_{j=0}^{M+1} R_1(z(x_i,t_j),t_j)I_j\,.$$

Algorithm 3: Special approximation of the functional used in Algorithm 1 to update ϕ. This algorithm can only be used when the running and terminal costs $\mathcal{F}$ and $\mathcal{G}$ present in equation (3.3) are given by equation (3.7).

Result: The functional value used for updating the value function ϕ.

Input: Number of samples N, number of times M, initial distribution ρ_0, regularization constant λ

Produce a set $B := \{x_i\}_{i=1}^N$ of N samples from ρ_0;
Create equidistant times t_j for $j = 0, 1, \ldots, M+1$ at which one wants to generate observations (including $t_0 = 0$ and $t_{M+1} = T$);
for *every sample* x_i **do**
  for *every time* t_j **do**
    Generate a sample at time t_j given initial condition x_i by means of the generator: $z(x_i, t_j) = \mathbb{G}(x_i, t_j)$ (Note: $\mathbb{G}(x, 0) = x$ should hold). For examples of generators, see Section 3.2;
  end
end
Defining $I_j = \frac{1}{2}$ if $j \in \{0, M+1\}$ and $I_j = 1$ if $j \in \{1, \ldots, M\}$, approximate the functional that only consists of the ρ dependent terms in the functional Ψ using::

$$\left(\frac{1}{N}\sum_{i=1}^{N} -\phi(x_i, 0)\right) + \frac{1}{N(M+1)}\sum_{i=1}^{N}\sum_{j=0}^{M+1} R_2(z(x_i, t_j), t_j) I_j \,.$$

Optionally one can add a regularization term:

$$\lambda\left(\frac{1}{N(M+1)}\sum_{i=1}^{N}\sum_{j=0}^{M+1} |R_1(z(x_i, t_j), t_j)| I_j\right) .$$

3.4 Dealing with more general potential MFGs

Algorithm 1 for solving potential stochastic MFGs requires approximating the functional Ψ in equation (3.3). The previous section introduced approximation algorithms for Ψ when the potential MFG described by equation (3.1) has a specific form. Namely, the running cost $\mathcal{F}$ and the terminal cost $\mathcal{G}$ in equation (3.3) are given by equation (3.7). In this case, the variational derivatives F and G in equation (3.1) do not depend on ρ but only on the spatial location of the agents. The used variational derivatives in [Lin et al., 2020] only depend on the spatial location of the agents as well. In [Ruthotto et al., 2020], one deals with potential MFGs where the variational derivatives can depend on both ρ and the spatial location x. However, in that work, only deterministic MFGs are considered. Their approximation approach that deals with these more general running and terminal costs does not immediately extend to stochastic MFGs. Nevertheless, it is possible to use their approach via our min-max formulation in equation (3.4). To our knowledge, this results in the first deep learning based solution method for stochastic potential MFGs with running and terminal costs having an agent distribution dependent variational derivative.

First, this section discusses on a high level how to approximate Ψ for these more general stochastic

MFGs. More specifically, we argue that this can be achieved by combining the min-max formulation in equation (3.4) with the approach in [Ruthotto et al., 2020]. Subsequently, the details of the proposed algorithm are treated.

3.4.1 Possible extension to the stochastic case

This section discusses on a high level how we can deal with the more general stochastic potential MFGs by combining the min-max formulation in equation (3.4) with the approach in [Ruthotto et al., 2020]. This discussion shows that a neural ODE needs to be used as the generator in Algorithm 1. In addition to this motivation to use a neural ODE as the generator, we provide two more reasons why a neural ODE is a sensible choice. These two additional reasons provide additional motivation for the neural ODE generator choice. To recall how a neural ODE can be used as a generator, see Section 3.2.

In [Ruthotto et al., 2020], the authors use the following theorem:

Theorem 3.4.1 (From [Ruthotto et al., 2020])**.** Let ρ satisfy the following continuity equation

$$\frac{\partial}{\partial t}\rho(x,t) + \nabla \cdot (\rho(x,t)v(x,t)) = 0$$

with initial condition $\rho(x,0) = \rho_0(x)$. Moreover, let $z(x,t)$ be the solution to

$$\frac{\mathrm{d}}{\mathrm{d}t}z(x,t) = v(z,t)\,, \quad z(x,0) = x\,.$$

Then the following identity holds

$$\rho(z(x,t),t)\det(\nabla_x z(x,t)) = \rho_0(x)\,.$$

In the latter paper, this result is not rigorously proven. Appendix D of this thesis gives a proof of this result. The authors of the paper use $\rho(z(x,t),t)\det(\nabla_x z(x,t)) = \rho_0(x)$ to replace $\rho(x,t)$ with known quantities. Using this replacement, the authors deal with variational derivatives of running costs $\mathcal{F}$ and terminal costs $\mathcal{G}$ that depend on the distribution $\rho(x,t)$.

To replicate their approach, it needs to be noted that in Theorem 3.4.1 one needs the solution to an ordinary differential equation. In [Ruthotto et al., 2020], the used ordinary differential equation is employed as the generator and must be the agent dynamics. In our case, the agent dynamics is given by a stochastic differential equation. It is difficult to extend the whole approach of [Ruthotto et al., 2020] to such agent dynamics as Theorem 3.4.1 can not easily be extended to a stochastic differential equation. The main difficulty in extending this theorem lies in its proof. This proof uses the method of characteristic on the Fokker-Planck equation corresponding to an ODE. The method of characteristics can not be applied in the same way when dealing with a Fokker-Planck equation corresponding to an SDE.

The previous paragraph shows that the model used in [Ruthotto et al., 2020] must use the agent dynamics as the generator. On the other hand, the min-max game introduced in Section 2.1 does not pose constraints on the generator and the discriminator. Hence, even though it makes sense to use a neural SDE, as discussed in Section 3.2, any other generator parametrization can be used. For instance, an appropriately chosen ordinary differential equation can be used. This makes it possible to apply Theorem 3.4.1 and deal with running costs $\mathcal{F}$ and final costs $\mathcal{G}$ whose variational derivative depends on ρ.

A-priori it is not clear why an ordinary differential equation is a suitable replacement for the stochastic differential equation. We provide two reasons to further motivate the use of a neural ODE as the generator. First of all, it is the closest to the stochastic differential equation. If the stochasticity is small, a simulation of the SDE and a simulation of the ODE are expected to not differ a lot. Moreover, ρ is the interest and not necessarily the agent dynamics. Hence, it might be possible that for a suitably chosen ordinary differential equation, the resulting probability distribution is the same as when one uses the SDE. In fact, the following theorem gives such an 'equivalent' ODE.

Theorem 3.4.2 (From [Song et al., 2020])**.** Assume a smooth solution exists to the following Fokker-Planck equation:

$$\frac{\partial}{\partial t}\rho(x,t) = -\nabla \cdot (f(x,t)\rho(x,t)) + \frac{1}{2}\nabla^2 \cdot (D(x,t)D(x,t)^T\rho(x,t))\,, \tag{3.11}$$

where f is a vector-valued function, D a matrix-valued function and $\nabla^2\cdot$ is defined by:

$$\nabla^2 \cdot A(x) := \sum_{i=1}^{d}\sum_{j=1}^{d} \frac{\partial^2}{\partial x_i \partial x_j}\left(A_{ij}(x)\right)\,.$$

Moreover, assume that $\nabla_x \log(\rho(x,t))$ exists for all (x,t). Then the solution to the following Fokker-Planck equation also solves equation (3.11):

$$\frac{\partial}{\partial t}\rho(x,t) = -\nabla \cdot (\tilde{f}(x,t)\rho(x,t))\,,$$

where

$$\tilde{f}(x,t) := f(x,t) - \frac{1}{2}\nabla \cdot \left(D(x,t)D(x,t)^T\right) - \frac{1}{2}D(x,t)D(x,t)^T\nabla_x \log(\rho(x,t))\,. \tag{3.12}$$

The divergence operator on the matrix $D(x,t)D(x,t)^T$ is taken row wise.

The first Fokker-Planck equation presented in Theorem 3.4.2 corresponds to an arbitrary stochastic differential equation, while the second Fokker-Planck equation corresponds to an ordinary differential equation. Recall that a Fokker-Planck equation characterizes the distribution over time of the corresponding differential equation. Hence, the theorem says that given an initial distribution ρ_0, the ordinary differential equation has the same distribution as the stochastic differential equation. This makes an ODE a suitable replacement for the SDE. However, one does have to be careful. The preceding theorem only holds when $\nabla_x \log(\rho(x,t))$ exists for all (x,t). When $\rho(x,t) = 0$ at some places, this might not hold. However, in the regions where $\rho(x,t) = 0$, an agent does not appear anyhow. Hence, these regions are not important for the agent dynamics. This may make the non-existence of $\nabla_x \log(\rho(x,t))$ not an issue. On the other hand, one can take a function that is very close to $\nabla_x \log(\rho(x,t))$ and does not have singularity issues. Using this in $\tilde{f}$ (in equation (3.12)) instead of $\nabla_x \log(\rho(x,t))$, one hopefully obtains an agent dynamics that approximately has $\rho(x,t)$ as distribution. As we are interested in approximately solving the Fokker-Planck equation, this suffices.

3.4.2 The approximation algorithm for the stochastic case

The previous section treats on a high level an algorithm to approximate the functional Ψ in equation (3.3). We want to use this algorithm to treat stochastic potential MFGs with variational derivatives dependent on the agent distribution ρ. In the current section, we discuss the details of this algorithm.

By our discussion in the previous section, we parametrize the generator $\mathbb{G}$ by:

$$\frac{\mathrm{d}z}{\mathrm{d}t} = f_{NN}(z,t)\,, \quad z(x,0) = x \tag{3.13}$$

where x is a sample from the initial distribution $\rho_0(x)$, $z(x,t)$ is the generated sample at time t given initial point x, and $f_{NN}(z,t)$ can be any neural network. At the numerical experiments, specific architectural choices for the neural network are discussed. Given a specific parametrization, we want to calculate the functional value in equation (3.3) for more general running and terminal costs than the costs allowed in Algorithms 2 and 3. We use a combination of the algorithms in [Lin et al., 2020] and [Ruthotto et al., 2020] to achieve this. To demonstrate how the functional value is calculated for this more general case, the $(-\phi_t - \nu\Delta\phi + H(\cdot, \nabla\phi))\rho$ part is treated first. Subsequently, we discuss how to deal with the $\mathcal{F}$ and $\mathcal{G}$ parts.

To calculate the $(-\phi_t - \nu\Delta\phi + H(\cdot, \nabla\phi))\rho$ part, we parametrize the agent position distribution as ρ_{NN}, which is defined via the following Fokker-Planck equation:

$$\frac{\partial}{\partial t}\rho_{NN}(x,t) + \nabla\cdot(\rho(x,t)f_{NN}(x,t)) = 0\,, \quad \forall (x,t) \in \mathbb{R}^d \times [0,T]\,.$$

Here $\rho_{NN}(x,0) = \rho_0(x)$. Using Theorem 3.4.1 in combination with our generator parametrization given in equation (3.13), we get

$$\begin{aligned}
&\int_{\mathbb{R}^d}\int_0^T (-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t)))\rho(x,t)\mathrm{d}t\mathrm{d}x \\
&= \int_{\mathbb{R}^d}\int_0^T (-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t),\nabla\phi(z(x,t),t)))\rho(z(x,t),t)\det(\nabla_x z(x,t))\mathrm{d}t\mathrm{d}x \\
&= \int_{\mathbb{R}^d}\int_0^T \rho_0(x)(-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t),\nabla\phi(z(x,t),t)))\mathrm{d}t\mathrm{d}x \\
&= \int_{\mathbb{R}^d} \rho_0(x)\int_0^T (-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t),\nabla\phi(z(x,t),t)))\mathrm{d}t\mathrm{d}x \\
&= \mathbb{E}_{x\sim\rho_0}\left[\int_0^T (-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t),\nabla\phi(z(x,t),t)))\mathrm{d}t\right].
\end{aligned}$$

In our algorithm, the time integral is calculated using the trapezoidal rule. In addition, the expectation is approximated by Monte Carlo methods. This approach to calculate this part of the Ψ functional is the same as the approximation method introduced in Section 3.3.

The above approach of deriving the Monte Carlo technique of Section 3.3 plays an important role when dealing with $\mathcal{F}$ and $\mathcal{G}$. As one deals with the $\int_0^T \mathcal{F}(\rho(\cdot,t))\mathrm{d}t$ part in the same way as the $\mathcal{G}(\rho(\cdot,T))$ part in combination with a time integral approximation, only the $\mathcal{G}$ part is dealt with. In order to treat this component, the specific terminal cost that is taken is the KL-divergence between $\rho(x,T)$ and some distribution $\rho_T(x)$. Following similar reasoning to when we dealt with the $(-\phi_t - \nu\Delta\phi + H(\cdot,\nabla\phi))\rho$ part of the functional, one can rewrite the KL-divergence.

Before rewriting the KL-divergence, we must define the KL-divergence. The KL-divergence is defined as:

$$\mathcal{G}(\rho(\cdot,T)) = \int_{\mathbb{R}^d} \rho(x,T)\log\left(\frac{\rho(x,T)}{\rho_T(x)}\right)\mathrm{d}x\,,$$

where ρ_T is some chosen distribution. To rewrite the KL-divergence, we change variables based on the $z(x,t)$ variable given in equation (3.13). We achieve this by first replacing the x variables in the

integrand with $z(x,T)$. Subsequently, we multiply the integrand with a $\det(\nabla_x z(x,T)))$ factor to complete the change of variables. Performing both of these operations yields:

$$\begin{aligned}\mathcal{G}(\rho(\cdot,T)) &= \int_{\mathbb{R}^d} \rho(x,T)\log\left(\frac{\rho(x,T)}{\rho_T(x)}\right)\mathrm{d}x \\ &= \int_{\mathbb{R}^d} \rho(z(x,T),T)\log\left(\frac{\rho(z(x,T),T)}{\rho_T(x)}\right)\det(\nabla_x z(x,T))\mathrm{d}x\end{aligned}$$

By using Theorem 3.4.1 on $\rho(z(x,T),T)$ in the last integral, we get:

$$\begin{aligned}\mathcal{G}(\rho(\cdot,T)) &= \int_{\mathbb{R}^d} \rho_0(x)\log\left(\frac{\rho(z(x,T),T)}{\rho_T(x)}\right)\mathrm{d}x \\ &= \int_{\mathbb{R}^d} \left(\log(\rho(z(x,T),T)) - \log(\rho_T(x))\right)\rho_0(x)\mathrm{d}x \\ &= \int_{\mathbb{R}^d} \left(\log(\rho_0(x)) - \log\left(\det(\nabla_x z(x,T))\right) - \log(\rho_T(x))\right)\rho_0(x)\mathrm{d}x\,.\end{aligned}$$

In the last formulation, one does not have to deal with the $\rho(x,T)$ term that is hard to determine. Given formulas for ρ_0 and ρ_T the last integral can be approximated via Monte Carlo methods by sampling from ρ_0. The only term that is not known is $\log(\det(\nabla_x z(x,t)))$. This term can be calculated using an ordinary differential equation:

Theorem 3.4.3 (An adaptation of a result used in [Ruthotto et al., 2020])**.** Let the log-determinant be denoted by $l(x,t) = \log(\det(\nabla_x z(x,t)))$, $z(x,t)$ be the solution to $\frac{\partial}{\partial t}z(x,t) = v(z,t)$ with $z(x,0) = x$ and assume $\nabla_x z(x,t)$ is invertible (or equivalently, its determinant unequal to zero). Then

$$\begin{aligned}\frac{\partial}{\partial t}l(x,t) &= \nabla_z \cdot v(z(x,t),t)\,, \\ l(x,0) &= 0\,.\end{aligned}$$

Proof. By the chain rule, it is known:

$$\frac{\partial}{\partial t}l(x,t) = \frac{\partial}{\partial t}\log(\det(\nabla_x z(x,t))) = \frac{1}{\det(\nabla_x z(x,t))}\frac{\partial}{\partial t}\det(\nabla_x z(x,t))\,.$$

Using derivative identities of the Matrix Cookbook [Petersen and Pedersen, 2012], one gets

$$\begin{aligned}\frac{\partial}{\partial t}l(x,t) &= \frac{1}{\det(\nabla_x z(x,t))}\det(\nabla_x z(x,t))\mathrm{Tr}\left(\left(\nabla_x z(x,t)\right)^{-1}\frac{\partial}{\partial t}\nabla_x z(x,t)\right) \\ &= \mathrm{Tr}\left(\left(\nabla_x z(x,t)\right)^{-1}\nabla_x v(z(x,t),t)\right) = \mathrm{Tr}\left(\left(\nabla_x z(x,t)\right)^{-1}\nabla_x z(x,t)\nabla_z v(z(x,t),t)\right) \\ &= \mathrm{Tr}\left(\nabla_z v(z(x,t),t)\right) = \nabla_z \cdot v(z(x,t),t)\,.\end{aligned}$$

■

Hence, using the above theorem and the functions $\rho_0(x)$ and $\rho_T(x)$, the Kullback-Leibler divergence can be approximated via Monte Carlo methods when using $\frac{\partial}{\partial t}z = v$ as the generator. In other words, one can estimate KL-divergence term by:

$$\frac{1}{N}\sum_{k=1}^{N}(\log(\rho_0(x_k)) - l(x_k,T) - \log(\rho_T(x_k)))$$

with $x_k \sim \rho_0(x)$ and N the number of samples used. Such an approximation is not possible with the approach in [Lin et al., 2020].

At this point, we discussed the approximation of all the components that are needed to approximate Ψ in equation (3.3). However, additional regularization terms might still need to be approximated to help converge to the correct solution. For instance, it becomes clear from equation (3.1) that the value function satisfies the following two equations

$$\int_0^T \int_{\mathbb{R}^d} \|-\phi_t(x,t) - \nu\Delta\phi(x,t) + H(x,\nabla\phi(x,t)) - F(x,\rho(\cdot,t))\|\,\rho(x,t)\mathrm{d}x\mathrm{d}t = 0\,,$$

$$\int_{\mathbb{R}^d} \|\phi(x,T) - G(x,\rho(\cdot,T))\|\,\rho(x,T)\mathrm{d}x = 0\,,$$

where $F = \frac{\partial \mathcal{F}}{\partial \rho}$ and $G = \frac{\partial \mathcal{G}}{\partial \rho}$ are variational derivatives. Even though this prior knowledge can be used in updating the generator as well as in updating the value function ϕ, we only use it when optimizing ϕ. The main reason is that the above equations correspond to the HJB equation, which mainly characterizes the value function ϕ. The prior knowledge is incorporated into the algorithm for the value function ϕ by adding the left-hand sides multiplied by a regularization constant to the Ψ functional. The way of calculating the regularization terms is analogous to our discussion before.

Using the previous discussions, we obtain Algorithms 4 and 5 for approximating the functional values used in Algorithm 1. These values are used to update either the generator or the value function ϕ. Algorithms 4 and 5 are applicable to more general running cost and terminal cost than Algorithms 2 and 3. To be specific, they can treat costs whose variational derivative depends on ρ.

Algorithm 4: More general approximation of the functional used in Algorithm 1 to update the generator. This algorithm can be used for more general running cost $\mathcal{F}$ and terminal cost $\mathcal{G}$ in equation (3.3) than Algorithm 2. To be specific, it is applicable to running and terminal costs whose variational derivative depends on ρ.

Result: The functional value used for updating the generator $\mathbb{G}$

Input: Number of samples N, number of times M, initial distribution ρ_0

Produce N samples from ρ_0 and create equidistant times t_j for $j = 0, 1, \ldots, M+1$ at which one wants to generate observations (including $t_0 = 0$ and $t_{M+1} = T$);

for *every sample* x_i **do**

Solve the following ODE:

$$\frac{\partial}{\partial t}\begin{bmatrix} z(x,t) \\ l(x,t) \end{bmatrix} = \begin{bmatrix} f_{NN}(z(x,t),t) \\ \nabla_z \cdot f_{NN}(z(x,t),t) \end{bmatrix}, \quad \begin{bmatrix} z(x,0) \\ l(x,0) \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}.$$

Note that the log-determinant $l(x,t)$ can pop up when dealing with the running cost $\mathcal{F}$ and the terminal cost $\mathcal{G}$.

end

Using the samples x_k, approximate the following functional by means of Monte Carlo and the trapezoidal rule:

$$\int_0^T \Big\{ \int_{\mathbb{R}^d} \rho_0(x)(-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t),\nabla\phi(z(x,t),t)))\mathrm{d}x$$
$$-\hat{\mathcal{F}}(z(\cdot,t),\rho_0)\Big\}\,\mathrm{d}t + \int_{\mathbb{R}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \hat{\mathcal{G}}(z(\cdot,T),\rho_0)\,.$$

Algorithm 5: More general approximation of the functional used in Algorithm 1 to update ϕ. This algorithm can be used for more general running cost $\mathcal{F}$ and terminal cost $\mathcal{G}$ in equation (3.3) than Algorithm 3. To be specific, it is applicable to running and terminal costs whose variational derivative depends on ρ.

Result: The functional value used for updating the value function ϕ

Input: Number of samples N, number of times M, initial distribution ρ_0, regularization constants λ_1 and λ_2

Produce N samples from ρ_0 and create equidistant times t_j for $j = 0, 1, \dots, M+1$ at which one wants to generate observations (including $t_0 = 0$ and $t_{M+1} = T$);

for *every sample* x_i **do**

Solve the following ODE:

$$\frac{\partial}{\partial t}\begin{bmatrix} z(x,t) \\ l(x,t) \end{bmatrix} = \begin{bmatrix} f_{NN}(z(x,t),t) \\ \nabla_z \cdot f_{NN}(z(x,t),t) \end{bmatrix}, \quad \begin{bmatrix} z(x,0) \\ l(x,0) \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}.$$

Note that the log-determinant $l(x,t)$ can pop up when dealing with the running cost $\mathcal{F}$ and the terminal cost $\mathcal{G}$.

end

Using the samples x_k, approximate the following functional by means of Monte Carlo and the trapezoidal rule:

$$\begin{aligned}
&\int_0^T \int_{\mathbb{R}^d} \rho_0(x) \left(-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t), \nabla\phi(z(x,t),t))\right) \mathrm{d}x\mathrm{d}t \\
&+ \int_{\mathbb{R}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi(x,0)\mathrm{d}x \\
&+ \lambda_1 \int_{\Omega_T} \rho_0(x) \left|-\phi_t(z(x,t),t) - \nu\Delta\phi(z(x,t),t) + H(z(x,t), \nabla\phi(z(x,t),t)) - \hat{F}(z(\cdot,t),\rho_0)\right| \mathrm{d}x\mathrm{d}t \\
&+ \lambda_2 \int_{\mathbb{R}^d} \rho_0(x) \left|\phi(z(x,T),T) - \hat{G}(z(\cdot,T),\rho_0)\right| \mathrm{d}x\,.
\end{aligned}$$

The functions $\hat{\mathcal{F}}$ and $\hat{\mathcal{G}}$ appear in the above two algorithms. These functions are obtained by applying Theorem 3.4.1 to the functionals $\mathcal{F}$ and $\mathcal{G}$, respectively. For an example to go from $\mathcal{G}$ to $\hat{\mathcal{G}}$, see the KL-divergence part in this section. In addition, the functions $\hat{F}$ and $\hat{G}$ are used. These functions are obtained by applying Theorem 3.4.1 to the functions $F = \frac{\partial}{\partial\rho}\mathcal{F}$ and $G = \frac{\partial}{\partial\rho}\mathcal{G}$. As these two functionals depend on the specific functions $\mathcal{F}$ and $\mathcal{G}$, no elaborate explanation is given here. By going through the numerical experiment in Section 4.2, one gets a feel of how to construct these functions.

In addition, for a specific class of functionals $\mathcal{G}$, some of the calculated components in algorithms 4 and 5 vanish. The specific class of $\mathcal{G}$ functionals is

$$\mathcal{G}(\rho) = \int_{\mathbb{R}^d} G(x)\rho(x)\mathrm{d}x\,.$$

This is the same class as used in Section 3.3. Using the same value function parametrization as in Section 3.3 and following the same considerations, the $\int_{\mathbb{R}^d} \rho(x,T)\phi(x,T)\mathrm{d}x - \hat{\mathcal{G}}(z(\cdot,T),\rho_0)$ term can be removed from algorithm 4. Using similar reasoning on algorithm 5, we can remove the $\rho(x,T)\phi(x,T)$ integral and the second regularization term.

Chapter 4

Numerical experiments

The purpose of this chapter is to investigate the advantages of neural differential equations in solving stochastic mean field games. To investigate these advantages, this chapter discusses several two-dimensional stochastic potential MFGs, which are solved using the algorithms in Chapter 3. First, a destination problem is treated where each agent wants to move to a specific destination and no obstacles are present. Moreover, the agent wants to move with minimal effort and agents do not want to avoid each other. For this problem, we compare generic neural networks, such as a simple feedforward neural network or a ResNet, to neural SDEs concerning training instability. The problem is also used to argue about the interpretability of the agent trajectories obtained when using a generic neural network as the generator. Subsequently, the same problem is treated, however, with the addition of a simple obstacle that the agents have to avoid. This problem is solved using a neural SDE as the generator in Algorithm 1. Using this experiment in combination with the earlier introduced interpretability experiment, we compare generic neural networks to neural SDEs regarding the interpretability of the obtained agent trajectories. In the final section of this chapter, we treat again a destination problem without obstacles. However, contrary to the previous cases, the agents do want to avoid each other. In performing this crowd aversion experiment, we show that the algorithm in Section 3.4 allows more general cost functionals in the MFG than the solution algorithm in [Lin et al., 2020].

4.1 Numerical experiments for a specific type of running and terminal costs

This section treats two of the three destination problems introduced in the introduction of this chapter. These two problems are the two problems in which the agents do not want to avoid each other. We use these problems to compare neural SDE generators to generic neural network generators concerning training instability and interpretability of the obtained agent trajectories. For the mentioned problems, the scenario as described in Section 3.3 is used. More specifically, the used running and terminal costs are defined by:

$$\begin{aligned} \mathcal{F}(\rho) &= \alpha \int_{\mathbb{R}^d} f(x)\rho(x)\mathrm{d}x\,, \\ \mathcal{G}(\rho) &= \int_{\mathbb{R}^d} g(x - x_T)\rho(x)\mathrm{d}x \end{aligned} \tag{4.1}$$

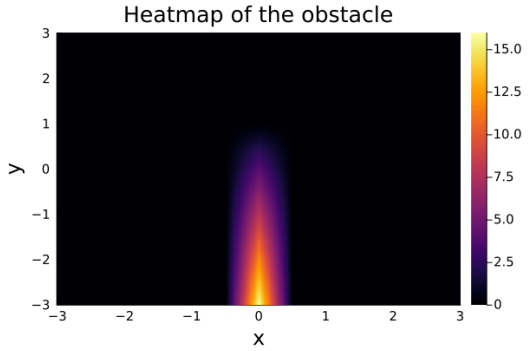
with

- $g(x)$: a function comparing x to the origin.
- $x_T = \begin{bmatrix} 2.0 & 0.0 \end{bmatrix} \in \mathbb{R}^2$ the destination.
- f: a function that represents a wall.
- $\alpha \in \mathbb{R}$: a constant determining the strength of the wall.

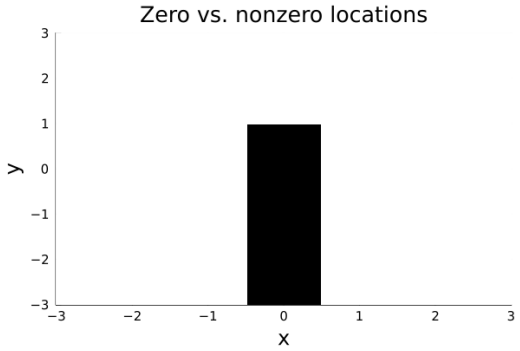
The other parameters that are fixed in this section are:

- The initial agent position distribution $\rho_0 = \mathcal{N}(\mu, \Sigma)$.
- $\mu = \begin{bmatrix} -2.0 & 0.0 \end{bmatrix}$.
- $\Sigma = \sigma^2 I$ with $\sigma = 0.2$.
- The final time $T = 1$.

For our purposes, a specific f is chosen. The values the function f takes can be seen in Figure 4.1.



(a) A heatmap of the function f. This plot shows that the obstacle is a continuous hill instead of a discontinuous wall.



(b) A picture indicating the location of the present obstacle. The black values are the values where $f \neq 0$ and the white values where $f = 0$.

Figure 4.1: Two representations of the obstacle structure used in the numerical experiments of Section 4.1. This structure is parametrized via the function f in equation (4.1). The full wall is determined in equation (4.1) by this function f and the strength of the wall α. Here x and y are the first and second component of the state vector, respectively.

The purpose of the two functions $\mathcal{F}$ and $\mathcal{G}$ is for the agents to move from their initial starting position to x_T while avoiding the wall parametrized by f. However, as many different paths achieve this, an appropriate path should be chosen. In our experiments, the path that results in the least amount of effort for the agent should be chosen. In other words, the path should be chosen that minimizes the kinetic energy. To achieve this, $L = \frac{1}{2} \|v\|^2$ is chosen in equation (3.2). This means the Hamiltonian given in equation (3.2) becomes $H(x, p) = \frac{1}{2} \|p\|^2$.

4.1.1 An unstable destination problem without obstacles

To study training instability issues, this section treats two simple destination problems, which are solved with a ResNet-based generator and a neural SDE generator. How a neural SDE can be used as a generator is treated in Section 3.2. The ResNet-based generator is the G_{N_θ} generator in equation (3.6) with N_θ a ResNet. Hence, when using the term ResNet-based generator, we mean a generator parametrized by equation (3.6) with N_θ a ResNet.

When using a ResNet-based generator, one of the two destination problems exhibits training instability. On the other hand, when using a neural SDE generator for this problem, we observe stable training. Before treating the unstable destination problem, a numerical experiment is performed where both a ResNet-based generator and a neural SDE generator perform well. This case can be parametrized with the following parameters:

- $\alpha = 0$
- $g(x) = 5 \|x\|_2$.
- $\nu = 0.2$

Before treating the experiments, we introduce the concept of a primal-dual gap plot. In this thesis, the saddle point problem in equation (3.4) needs to be solved. Note that for such a saddle point problem, the following holds:

$$\Psi(\rho, \phi^*) \leq \sup_{\rho} \Psi(\rho, \phi^*) = \inf_{\phi} \Psi(\rho^*, \phi) \leq \Psi(\rho^*, \phi)\,.$$

To summarize, at the solution (ρ^*, ϕ^*) to the saddle point problem, $\Psi(\rho, \phi^*) \leq \Psi(\rho^*, \phi)$ for all other ρ and ϕ choices. Hence, to verify whether our found solution makes sense, one can verify whether $\Psi(\rho, \phi^s) \leq \Psi(\rho^s, \phi)$ where (ϕ^s, ρ^s) are the found solutions. In a primal-dual gap plot, this is verified by monitoring all the found (ρ, ϕ) pairs during training and checking $\Psi(\rho, \phi^s) \leq \Psi(\rho^s, \phi)$ for those pairs. This is done by plotting both values present in the inequality in one figure.

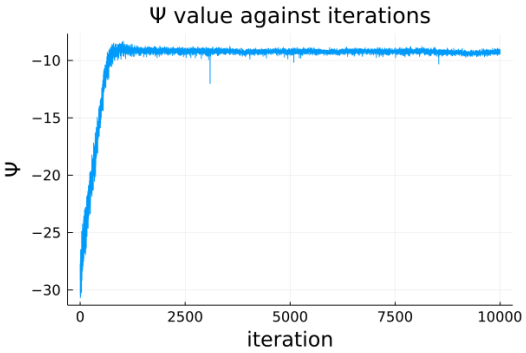
As the primal-dual gap plot is introduced and the problem is defined, the training settings are discussed. First, the ResNet-based generator case is treated. The parametrization of ϕ in the value function parametrization in equation (3.9) uses 3 skip-connections with 5 neurons each. Moreover, the timestep constant is 0.33 and the activation function is the tanh function. The generator is parametrized via equation (3.6). The neural network in this parametrization is a ResNet with 3 skip connections and a hidden layer size of 16. In training this generator and the value function, we use an HJB regularization parameter value of 1.0. Moreover, the functional used in the solution algorithm is approximated using 50 trajectories and each trajectory is sampled every 0.05 seconds. The position of the trajectory at the sampled values is used to approximate the time integrals in this functional. Furthermore, we train the value function for one iteration before switching to training the generator. The same holds when training the generator and switching to the value function. When training the value function, the ADAM optimizer is used with a learning rate of $5 \cdot 10^{-4}$. For training the generator, we use the ADAM optimizer with a learning rate of $5 \cdot 10^{-5}$. These learning rates are chosen as this is the only choice that shows proper training amongst several test learning rate pairs. The momentum parameters are the standard momenta as used in the Flux package of the Julia programming language.

When using a neural SDE as a generator instead, almost all the training settings are the same. The only settings that are different are the parametrization of the generator and the learning rates. The generator is parameterized as

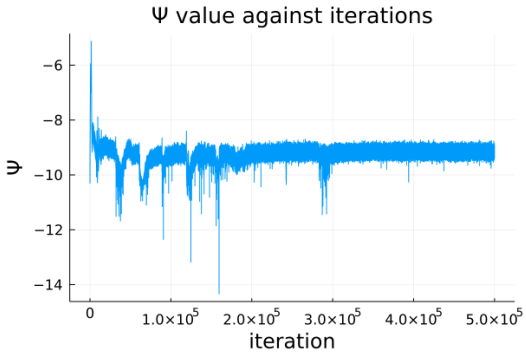
$$\mathrm{d}x = -\nabla\Phi(x, t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t\,, \tag{4.2}$$

where Φ is an independent copy of the value function parametrization. The reason for this parametrization becomes clear later on. The neural SDE generates trajectories by numerically solving equation (4.2) using the Euler-Maruyama scheme, which is introduced in Appendix A.1. For our purpose, this algorithm uses a time step of 0.05. In addition, the learning rate for the value function is increased to $5 \cdot 10^{-3}$. We increase the learning rate corresponding to the generator to 10^{-3}. The momentum parameters for the two ADAM optimizers are the standard momenta as used in the Flux package of the Julia programming language. Furthermore, to calculate the gradients that are used by the ADAM optimizer of the neural SDE generator, we use backpropagation on the operations of the SDE solver.

Using the chosen training settings, we train the ResNet-based generator case for 500000 iterations and the neural SDE case for 10000 iterations. This yields Figures 4.2 up to and including 4.4.

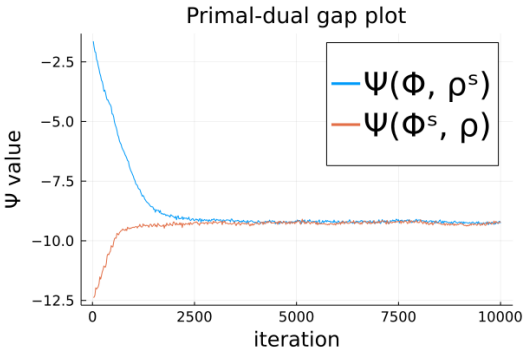


(a) The Ψ functional value over the iterations for the neural SDE generator.

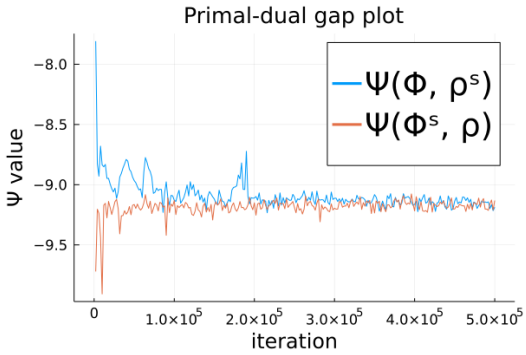


(b) The Ψ functional value over the iterations for the ResNet-based generator.

Figure 4.2: The Ψ functional values over the iterations for two generator types. The problem that is solved is a simple destination problem with a diffusion level of $\nu = 0.2$. It is solved once with a ResNet-based generator and once with a neural SDE as the generator.

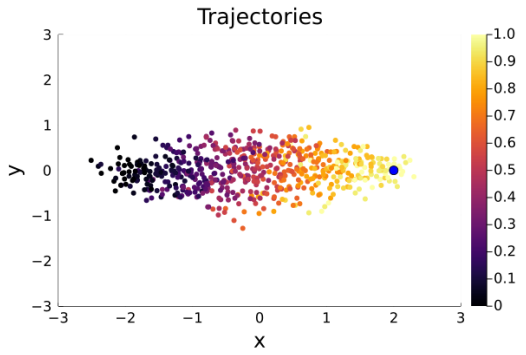


(a) Primal-dual gap plot for the neural SDE generator.

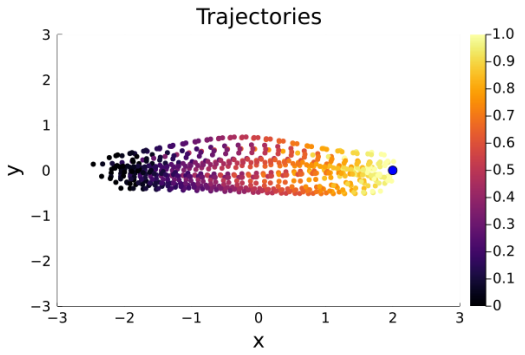


(b) Primal-dual gap plot for the ResNet-based generator.

Figure 4.3: Primal-dual gap plots for a simple destination problem with a diffusion level of $\nu = 0.2$. This is the same problem as treated in Figure 4.2. In the primal-dual gap plot, the blue line corresponds to $\Psi(\phi, \rho^s)$ and the orange line to $\Psi(\phi^s, \rho)$.



(a) Trajectories of the final neural SDE generator.



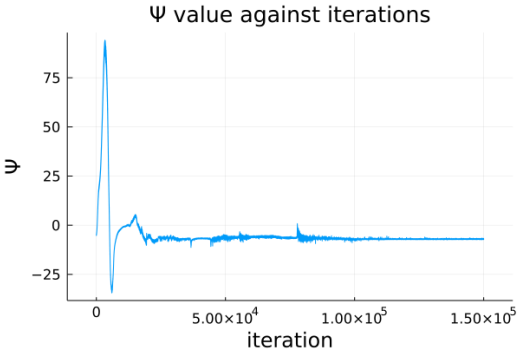
(b) Trajectories of the ResNet-based generator.

Figure 4.4: Trajectories for two different generator types used in solving a simple destination problem with a diffusion level of $\nu = 0.2$. This is the same problem as treated in Figure 4.2. The blue dot is the desired destination and the colours of the other dots represent a time between 0 and 1.

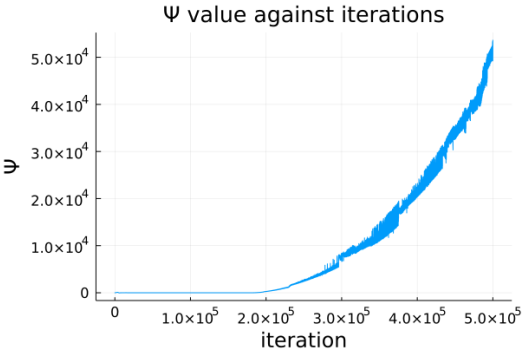
From the primal-dual gap plots and the value function loss against iterations plots, stable training is apparent. It is interesting to investigate if the stable training behaviour is robust under parameter changes. To investigate this, we introduce a second problem with the following parameters:

- $\alpha = 0$
- $g(x) = 2 \|x\|_2^2$.
- $\nu = 0.2$

Comparing these parameters with the ones of the first numerical experiment, only the terminal cost differs. More precisely, it only differs in the constant in front of the norm and the replacement of the norm with the squared norm. As a consequence, it seems sensible to train the neural network using the same parameters. Doing so and training for 500000 iterations, results in Figure 4.5.



(a) The Ψ functional value over the iterations for the ResNet-based generator up to and including iteration 150000.

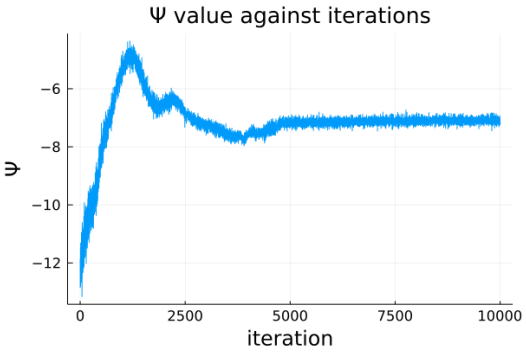


(b) The Ψ functional value over the iterations for the ResNet-based generator up to and including iteration 500000.

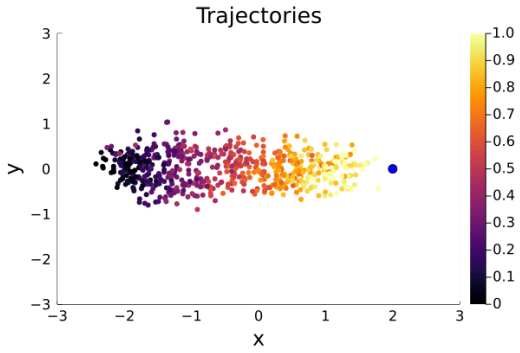
Figure 4.5: The Ψ functional value over the iterations when solving a difficult to solve destination problem. The pictures correspond to solving this problem using a ResNet-based generator.

Inspecting Figure 4.5, the functional value does not converge at all. The functional value even explodes after approximately 200000 iterations. As a consequence of this behaviour, the trajectories generated by the found ResNet-based generator do not make much sense. In fact, when generating trajectories, the trajectories explode away from their initial position and do not even come close to the desired destination.

All in all, as all but one of the problem and training parameters are the same as in the first problem in this section, the above observations show that a single adjustment can make training with a standard neural network, such as a ResNet, quite unstable. Now the question is what happens when the same is done with the neural SDE generator. Using the same training parameters as for the first numerical experiment and training for 10000 iterations, Figure 4.6 is obtained.



(a) The Ψ functional value over the iterations for the neural SDE generator.



(b) Plotting a couple of trajectories produced by the neural SDE generator. The blue dot is the desired destination and the colours of the other dots represent a time between 0 and 1.

Figure 4.6: Plots for a difficult to solve destination problem. This is the same problem as solved in Figure 4.5. The pictures correspond to solving this problem using a neural SDE as the generator instead of the ResNet-based generator used in Figure 4.5.

In Figure 4.6, the Ψ value seems to stabilize. In addition, a reasonable solution seems to be obtained. Hence, in contrast to the ResNet-based generator, the terminal cost change did not result in unstable training. From this observation, we obtain indications that the neural SDE generator exhibits more stable training behaviour than a standard neural network parametrization such as a ResNet.

A possible explanation for this observation could be that the SDE generator is in line with theory. Using equation (2.13) and the fact that the Hamiltonian is $H(x,p) = 0.5\,\|p\|^2$, it is known that the solution generator should be an SDE with constant stochasticity $\sqrt{2\nu}$ and drift term $-\nabla_x\phi^*$ with ϕ^* the value function solution. As the value function ϕ is parametrized via equation (3.9) and the generator is parametrized as

$$\mathrm{d}x = -\nabla\Phi(x,t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t \tag{4.3}$$

with Φ an independent copy of equation (3.9), it is possible to obtain the optimal value function and generator. More specifically, this happens when $\phi = \Phi = \phi^*$. When using a neural network as the generator, this structure is not present and is difficult to enforce. To elaborate on why this structure is important, consider the second problem treated in this section. For this specific problem, it is known that $\phi(x,T) = g(x - x_T)$ should hold and $\nabla g(x - x_T) = 2(x - x_T)$. The first constraint

can easily be enforced on the value function parametrization via equation (3.9). Parametrizing the neural SDE as in (4.3), one gets

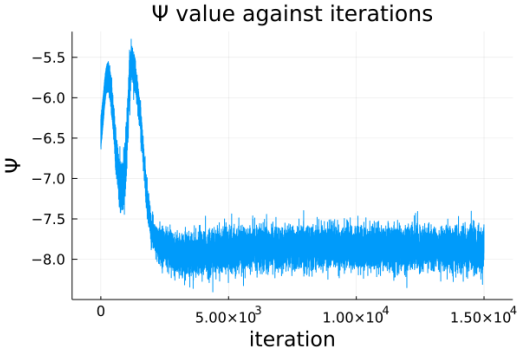
$$\mathrm{d}x = -\nabla\Phi(x,t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t = -((1-t)\nabla\tilde{\Phi}(x,t) + t\nabla g(x-x_T))\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t\,. \tag{4.4}$$

Inspecting equation (4.4) and using $\nabla g(x-x_T) = 2(x-x_T)$, the $\nabla g(x-x_T)$ term motivates the trajectories to move to the final destination. Hence, the additional knowledge helps to get appropriate trajectories. This is not possible with a neural network generator. To summarize, the neural SDE generator allows for more knowledge to be included in the generator, which could enhance stability.

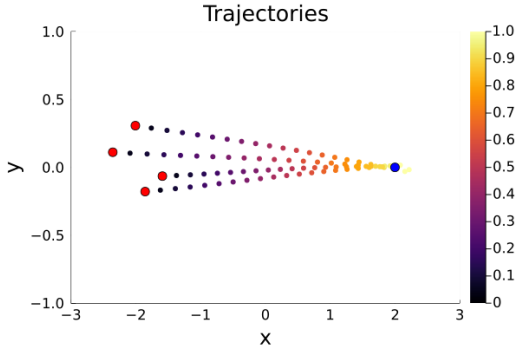
All in all, we have provided an example where a neural SDE proves more robust than a ResNet-based generator. Furthermore, we have given a possible explanation for this phenomenon. However, we have to be aware of the fact that only a limited number of simulations are performed. It could be that we have hit a bad problem for the ResNet-based generator case. Hence, fueled by the motivation given in the previous paragraph, we can only say that a neural SDE might help in stabilizing the procedure.

4.1.2 Interpretation of the trajectories: an obstacle destination problem

As described in Section 3.2, we can use a neural SDE as the generator in Algorithm 1. To see how such a neural SDE generator overcomes the limitations regarding interpretability of a more standard neural network as the generator, these limitations are discussed first. To show these limitations, we solve the first example in the previous section with diffusion level $\nu = 0.0001$ instead of $\nu = 0.2$. This case is only solved by using a ResNet-based generator, which is a generator parametrized by equation (3.6) with N_θ a ResNet. Using the same settings as in the previous section and training for 15000 iterations yields Figure 4.7.



(a) The Ψ functional value over the iterations.



(b) Plotting a couple of trajectories produced by the ResNet-based generator. The blue dot is the desired destination and the red dots are the initial points. The colours of the other dots represent a time between 0 and 1.

Figure 4.7: Plots for a low diffusion destination problem solved using a ResNet-based generator. The problem that is solved is the same as in Figures 4.2 up to and including 4.4, however, with a diffusion level of $\nu = 0.0001$ instead of $\nu = 0.2$.

First of all, note that the trajectories are straight lines. As the diffusion is very small, this is

expected. The reason is that the Hamiltonian corresponds to a problem that wants the shortest paths. Comparing this situation to Figure 4.4b, one sees that in a more stochastic case, we do not obtain straight lines connecting the initial point and the destination. The role of the generator is to produce samples from the optimal $\rho(\cdot, t)$. Due to the stochasticity in the problem, this ρ is wider than the optimal ρ with smaller stochasticity. This can also be inferred from Theorem 2.2.4 where a stochastic MFG is reinterpreted as a deterministic MFG with an additional running cost. This running cost says that the distribution can not be too peaked. Therefore, the agent distribution should be wider. As a consequence, the obtained trajectories must adjust for the stochasticity and produce non-straight trajectories.

One problem with the above observation is that, as a consequence, the trajectories do not correspond to the actual trajectories of an agent. They are just trajectories that create the same (optimal) agent distribution $\rho(\cdot, t)$. Another way to see that the created trajectories are not agent trajectories is the fact that the created trajectories do not have any stochasticity in them. In the particle view of the MFG, the trajectories are SDEs and therefore intrinsically stochastic. Moreover, due to the absence of stochasticity in the generated trajectories, one can not argue what the learned trajectories would look like if the diffusion level in the trajectories is lowered. When using a neural SDE as the generator, this can be done.

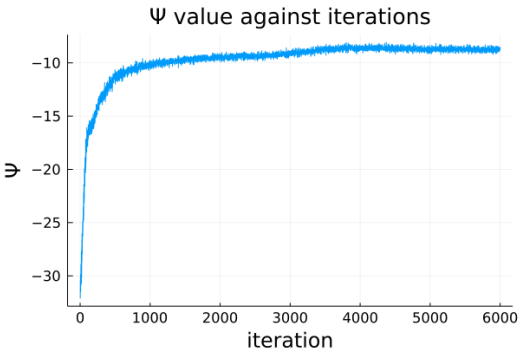
The previous paragraph gives a limitation of a neural network generator and indicates why neural SDE generators can solve this limitation. We give an example to elaborate on this. This example is given by:

- $\alpha = 100.0$
- $g(x) = 7.5 \left\|x\right\|_2$

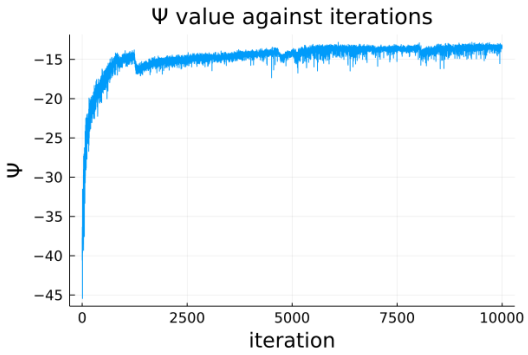
The above problem is solved for $\nu = 0.0001$ and $\nu = 0.2$ using a neural SDE generator. In solving the problem, we use the same training parameters for both diffusion levels. These parameters are discussed now.

First of all, the value function is parametrized via equation (3.9). The specific parametrization of $\tilde{\phi}$ uses 3 skip connections and a step size of 0.33. Moreover, each hidden layer has 16 neurons and the used activation function is tanh. The neural SDE parametrization follows the same logic as in Section 4.1.1 and is given by equation (4.2). When training the value function and the neural SDE generator, we use an HJB regularization parameter of 1. In addition, we use 50 trajectories to approximate the functionals given in the Monte Carlo methods in Section 3.3. These trajectories are generated by solving the neural SDE with the Euler-Maruyama method. For our purposes, this method uses a time step of 0.05. Hence, trajectories are sampled every 0.05 seconds to estimate the time integrals. Furthermore, after the value function is trained for 1 iteration, the generator is trained. The same holds when training the generator and switching to the value function. The optimizer for the value function is the ADAM optimizer with a learning rate of $5 \cdot 10^{-3}$. On the other hand, the ADAM optimizer for the generator uses a learning rate of 10^{-3}. The momentum parameters for the two ADAM optimizers are the standard momenta as used in the Flux package of the Julia programming language. Furthermore, to calculate the gradients that are used by the ADAM optimizer of the neural SDE generator, we use backpropagation on the operations of the SDE solver.

Using the specified training settings and training the $\nu = 0.0001$ case for 6000 iterations and the $\nu = 0.2$ case for 10000 iterations, Figures 4.8 up to and including 4.10 are obtained. In Figure 4.9 trajectories with diffusion are shown and in Figure 4.10 trajectories without diffusion. With diffusion, the trajectories satisfy $\mathrm{d}x = f_\nu(x, t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t$ for some f_ν. Without this diffusion, they satisfy $\mathrm{d}x = f_\nu(x, t)\mathrm{d}t$ for the same f_ν. In both cases, the initial locations are sampled from ρ_0.

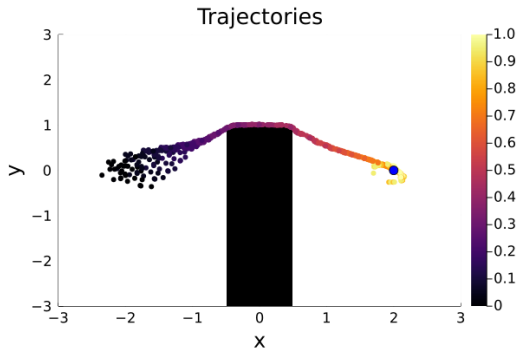


(a) The Ψ functional value against the iterations for the case with a diffusion level of $\nu = 0.0001$.

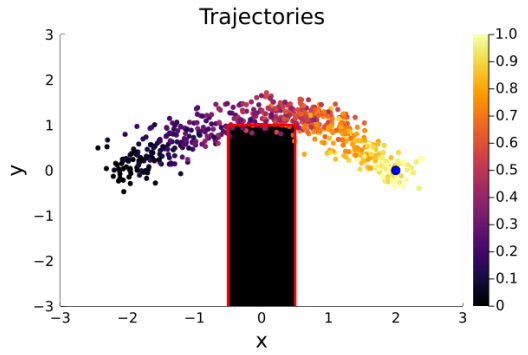


(b) The Ψ functional value against the iterations for the case with a diffusion level of $\nu = 0.2$.

Figure 4.8: The Ψ functional values against the iterations for a destination problem involving an obstacle. A low and high diffusion instance are treated. Inside the solution algorithm, a neural SDE is used as the generator.



(a) Agent trajectories corresponding to the found solution of the case with a diffusion level of $\nu = 0.0001$.

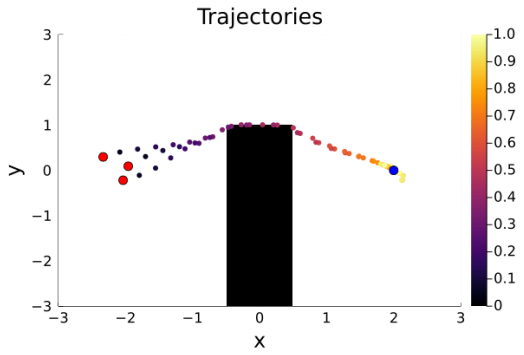


(b) Agent trajectories corresponding to the found solution of the case with a diffusion level of $\nu = 0.2$.

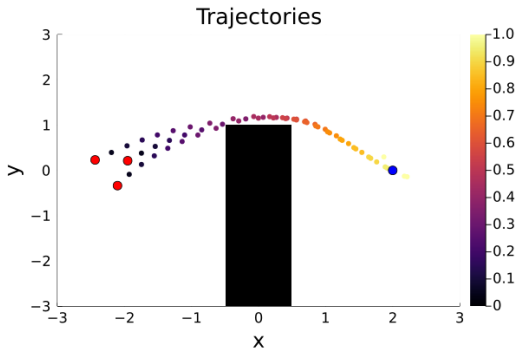
Figure 4.9: Agent trajectories corresponding to the found solutions for the obstacle destination problems treated in Figure 4.8. These trajectories follow an SDE of the form $\mathrm{d}x = f_\nu(x,t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t$ for some f_ν.

From Figure 4.9 it can be noted that in both the low and high diffusion case, we obtain trajectories that move to the final destination while avoiding the obstacle. In the low diffusion case the trajectories go very close to the boundaries and relatively straight lines are obtained. On the other hand, in the problem with more diffusion, some points are further from the boundary. Moreover, the points are more spread out due to the higher diffusion level.

For another difference between the low diffusion level case and the high diffusion level case, we inspect Figure 4.10 below.



(a) The non-diffusion trajectories for the $\nu = 0.0001$ case.



(b) The non-diffusion trajectories for the $\nu = 0.2$ case.

Figure 4.10: Trajectories without diffusion for the problems treated in figures 4.8 and 4.9. Instead of the neural SDE trajectories $\mathrm{d}x = f_\nu(x,t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t$ as plotted in Figure 4.9, non-diffusion trajectories $\mathrm{d}x = f_\nu(x,t)\mathrm{d}t$ are plotted.

It can be seen that in the $\nu = 0.2$ case the non-diffusion trajectories are not following the edge of the wall, while in the $\nu = 0.0001$ case the trajectories do. This makes perfect sense. As the stochasticity in the $\nu = 0.2$ case is larger, the chance of hitting the wall is larger. As a consequence, to avoid the wall with high cost, the trajectories, in general, have to be some distance away from the boundary. In the almost deterministic case, this is not necessary and the trajectories can hug the wall to make sure the shortest path is taken.

The above considerations show that a neural SDE generator can be used to reason about the properties of the agent trajectories. For instance, the effect of diffusion on the agent dynamics can be investigated as is done in Figures 4.9 and 4.10. On the other hand, when using a neural network as a generator, some of the earlier arguments can not be made. For example, a transition similar to the transition from Figure 4.9 to Figure 4.10 can not be made as the neural network only generates deterministic trajectories. Hence, when properties of the agent trajectories are desired, a neural SDE as the generator in Algorithm 1 is preferred over a neural network as the generator.

Research into the properties of the agent trajectories is important. An application of this research is regularized optimal transport. Via theorem 2.2.4, a stochastic MFG can be seen as a relaxed version of a regularized optimal transport problem. Hence, research into the effect of stochasticity in the MFG on the agent dynamics can give insight into the effect of the stochasticity parameter on the learned (regularized) optimal transport. This is important as it helps to get a good understanding of the obtained transport map and gives an indication whether this transport map is useful.

4.2 Crowd aversion experiment: MFG with entropy running cost

To illustrate that the algorithm in Section 3.4 works and can treat more general costs than the algorithm presented in [Lin et al., 2020], a crowd aversion experiment is treated. More precisely, we model crowd aversion by means of an entropy cost. It is not possible to use such an entropy in the algorithm presented in [Lin et al., 2020]. To our knowledge, the algorithm in Section 3.4 is the first deep learning based solution of stochastic MFGs that can deal with such running costs. First, we define the entropy running cost and derive the corresponding variational derivative. Additionally,

the $\hat{\mathcal{F}}$ and $\hat{F}$ functionals appearing in the algorithm of Section 3.4 are given. Subsequently, the remaining problem parameters are introduced and the problem is solved for two diffusion levels.

The entropy running cost that is used in our numerical experiment is given by

$$\mathcal{F}(\rho) = \int_{\mathbb{R}^d} \rho(x) \log(\rho(x)) \mathrm{d}x \,.$$

To use the algorithm introduced in Section 3.4, we need to derive the variational derivative of $\mathcal{F}$. Inspecting Definition 2.2.1, it suffices to calculate $\frac{\mathrm{d}}{\mathrm{d}h}\mathcal{F}(\rho + hw)\Big|_{h=0}$

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}h}\mathcal{F}(\rho + hw)\Big|_{h=0} &= \int_{\mathbb{R}^d} \frac{\mathrm{d}}{\mathrm{d}h} \left((\rho(x) + hw(x)) \log(\rho(x) + hw(x))\right) \mathrm{d}x\Big|_{h=0} \\
&= \int_{\mathbb{R}^d} w(x) \log(\rho(x) + hw(x)) + (\rho(x) + hw(x)) \frac{w(x)}{\rho(x) + hw(x)} \mathrm{d}x\Big|_{h=0} \\
&= \int_{\mathbb{R}^d} (\log(\rho(x)) + 1) w(x) \mathrm{d}x \,.
\end{aligned}$$

From the definition of a variational derivative, it follows that the variational derivative F is given by:

$$F(x, \rho) := \frac{\partial \mathcal{F}}{\partial \rho}(\rho) = \log(\rho(x)) + 1 \,.$$

Using the definition of $\mathcal{F}$ and F, the $\hat{\mathcal{F}}$ and $\hat{F}$ functionals used in the algorithm are derived. Both derivations follow a similar procedure. First, recall that the generator is given by

$$\frac{\mathrm{d}z}{\mathrm{d}t} = f_{NN}(z, t) \,, \quad z(x, 0) = x \,.$$

As a consequence, from Theorem 3.4.1 it is known that $\rho(z(x,t),t) \det(\nabla_x z(x,t)) = \rho_0(x)$. Using this on $F(x, \rho)$, $\hat{F}$ is obtained:

$$\hat{F}(x, \rho_0) = \log(\rho(z(x,t)) + 1 = \log(\rho_0(x)) - l(x,t) + 1 \,,$$

where $l(x,t) := \log(\det(\nabla_x z(x,t)))$. Using the same considerations for $\mathcal{F}$, we obtain $\hat{\mathcal{F}}$:

$$\begin{aligned}
\hat{\mathcal{F}}(x, \rho_0) &= \int_{\mathbb{R}^d} \rho(x) \log(\rho(x)) \mathrm{d}x = \int_{\mathbb{R}^d} \rho(z(x,t)) \log(\rho(z(x,t))) \det(\nabla_x z(x,t)) \mathrm{d}x \\
&= \int_{\mathbb{R}^d} \rho_0(x) (\log(\rho_0(x)) - l(x,t)) \mathrm{d}x \,.
\end{aligned}$$

Besides the above quantities related to the running cost, we need to define other problem parameters such as the Hamiltonian and the terminal cost. The remaining parameters concerning the desired movement of the agents are:

- Hamiltonian: $H(x,p) = \frac{1}{2} \|p\|^2$
- Terminal cost: $\mathcal{G}(\rho(\cdot, T)) = \int_{\mathbb{R}^d} g(x - x_T)\rho(x,T)\mathrm{d}x$ with $g(x) = 5\|x\|$
- Destination: $x_T = \begin{bmatrix} 2 & 0 \end{bmatrix}$
- Final time: $T = 1$

The last problem parameter that we need to define is the initial distribution of the agent positions:

- The initial distribution of the agent positions: $\rho_0 = \mathcal{N}(\mu, \Sigma)$
- $\mu = \begin{bmatrix} -2.0 & 0.0 \end{bmatrix}$
- $\Sigma = \sigma^2 I$ with $\sigma = 0.2$

To show that the algorithm in Section 3.4 works and can treat more general costs than the algorithm presented in [Lin et al., 2020], the above problem is solved for two diffusion levels, namely $\nu = 0.0001$ and $\nu = 0.2$. To find the solutions, the training settings need to be chosen. First, we treat the parametrizations of the value function and the neural ODE. To find an appropriate neural network architecture for the value function, note that the terminal cost has the form as used in Section 3.3. At the end of Section 3.4, it is mentioned that in this case an adaption of algorithms 4 and 5 can be used where some parts of the calculated functionals vanish. Using this observation in combination with the value function parametrization given by equation (3.9), we solve the problem. For the $\tilde{\phi}$ part in this parametrization, we use 2 skip-connections with a step size of 0.5. The number of neurons per hidden layer is set to 16 and a smoothed ReLu activation function is used. The smoothed ReLu function is given by:

$$\mathrm{ReLu}_s(x) = \frac{1}{\beta} \log(1 + e^{\beta x}),$$

where we choose $\beta = 2$ for our numerical experiments.

We still need to define the neural ODE generator. To define it, first note by equation (2.13) that the optimal agent trajectories follow:

$$\mathrm{d}x = -\nabla_x \phi^*(x, t)\mathrm{d}t + \sqrt{2\nu}\mathrm{d}W_t$$

with ϕ^* the optimal value function. As our algorithm only cares about the corresponding distribution ρ, via Theorem 3.4.2 an equivalent ODE can be used as a generator as well. In our case, this is

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -\nabla_x \phi^*(x, t) - \nu \nabla_x \ln(\rho^*(x, t))$$

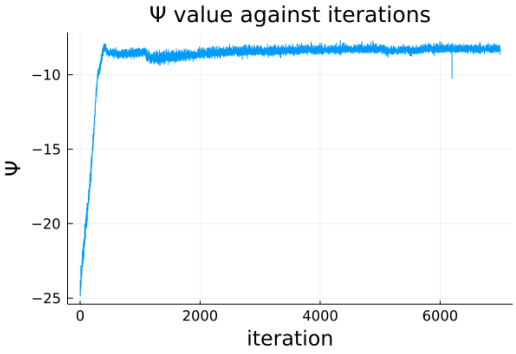
with $\rho^*(x, t)$ the optimal agent distribution. Consequently, in this numerical experiment, the neural ODE is parameterized as

$$\frac{\mathrm{d}x}{\mathrm{d}t} = -(1 - t)\nabla_x \Phi_1(x, t) - t\nabla_x g(x - x_T) - \nu \nabla_x \Phi_2(x, t)$$

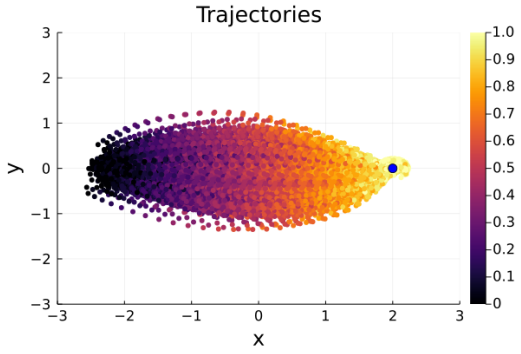
with Φ_1 and Φ_2 independent copies of $\tilde{\phi}$ in the value function parametrization given by equation (3.9).

Regarding training the above parametrizations, we use ADAM optimizers. The learning rate for the value function is $5 \cdot 10^{-3}$ and for the generator 10^{-3}. The momentum parameters for the two ADAM optimizers are the standard momenta as used in the Flux package of the Julia programming language. Furthermore, we use backpropagation on the operations of the ODE solver to calculate the gradients used by the ADAM optimizer of the neural ODE generator. After training the generator using the corresponding optimizer for one iteration, the value function is trained and vice versa. When updating the value function, we use an HJB regularization parameter of 1. The functionals appearing in the algorithm are approximated using 50 trajectories. These trajectories are obtained by solving the neural ODE with the forward Euler method and a step size of 0.05. Hence, the trajectories are sampled every 0.05 seconds.

By training the neural networks using the chosen parameter settings, we can show that the algorithm in Section 3.4 works and can treat more general costs than the algorithm presented in [Lin et al., 2020]. Training the neural networks using the training parameters for 7000 iterations when $\nu = 0.0001$ and for 20000 iterations when $\nu = 0.2$, Figures 4.11 and 4.12 are obtained.

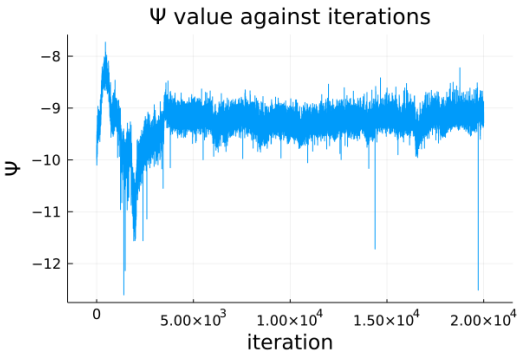


(a) The Ψ functional value over the iterations

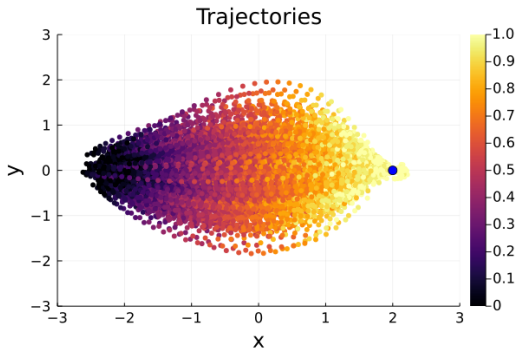


(b) Plotting 1000 trajectories produced by the neural ODE generator. The blue dot is the desired destination and the colours of the other dots represent a time between 0 and 1.

Figure 4.11: Plots for a crowd aversion problem with low diffusion.



(a) The Ψ functional value over the iterations



(b) Plotting 1000 trajectories produced by the neural ODE generator. The blue dot is the desired destination and the colours of the other dots represent a time between 0 and 1.

Figure 4.12: Plots for a crowd aversion problem with high diffusion.

In Figure 4.11, the almost deterministic case is shown. To see that the algorithm works, compare this figure to Figure 4.7. Here one can see that in absence of an entropy running cost, straight lines are obtained in an almost deterministic setting. On the other hand, when an entropy is present, curved trajectories are obtained. This makes sense as the agents want to avoid each other. In Figure 4.12, this effect can be seen as well.

Comparing Figures 4.11b and 4.12b, one can note several things. First of all, in the low diffusion case, most y values are between -1 and 1. In the high diffusion instance, the absolute value of y is

more often larger than 1. This effect can be caused by the higher diffusivity. This effect of diffusivity can be seen in Figure 4.4a as well. In addition, Figure 4.12b has a larger yellow area than Figure 4.11b. This indicates that in the higher diffusion case it is harder to reach the final destination. Due to the higher diffusion, the agents, in general, have to perform more effort to move to the final destination. Due to the penalty on this effort, the agents stop further away from the destination.

Both the above observations indicate that the trajectories in Figure 4.12b are decent. This shows that the algorithm works for the more stochastic case as well. All in all, we illustrated that the algorithm in Section 3.4 works for a stochastic MFG problem with an entropy running cost. It is not possible to solve this problem with the algorithm of [Lin et al., 2020]. Therefore, the algorithm in Section 3.4 can treat more general costs than the algorithm presented in [Lin et al., 2020].

Chapter 5

Conclusions

Using two similar simple destination problems, we investigated the training stability of our algorithm, which solves stochastic potential MFGs via a two-player game formulation. More specifically, we investigated the training stability of the algorithm described in Sections 3.2 and 3.3. For the two destination problems, we trained a ResNet-based generator as well as a neural SDE generator. Both generator types exhibited stable training for one of the two problems. On the other hand, only the algorithm with the neural SDE is stable for the other problem. As the changes between the two settings in terms of problem and training settings were minimal, we concluded that neural SDE generators allow for more stable training in certain cases.

Furthermore, we investigated the interpretability of the trajectories produced by a standard neural network generator, such as a ResNet-based generator. Using a simple destination MFG problem, we show that a standard neural network generator does not allow proper agent trajectories. The main reason is that such generators can not generate stochastic trajectories. Consequently, the generator has to create trajectories that account for the diffusion to still be able to match the correct agent distribution. This results in trajectories that the agents do not actually follow. As a consequence, it is harder to study the actual agent trajectories. When using a neural SDE as the generator instead, we can explore the agent trajectories and the effect of the diffusion on them. More specifically, in the presence of an obstacle, more diffusion results in trajectories that are further away from the obstacle. Hence, when properties of the agent trajectories are desired, we prefer a neural SDE generator over a neural network as the generator. Research into the properties of the agent trajectories is important. An application of this research is regularized optimal transport. Via theorem 2.2.4, a stochastic MFG can be seen as a relaxed version of a regularized optimal transport problem. Hence, research into the effect of stochasticity in the MFG on the agent dynamics can give insight into the effect of the stochasticity parameter on the learned (regularized) optimal transport. This is important as it helps to get a good understanding of the obtained transport map and gives an indication whether this transport map is useful.

Lastly, following [Ruthotto et al., 2020], the algorithm of [Lin et al., 2020] is extended such that it can deal with more general costs. The basis for the extension is using a neural ODE as generator. This neural ODE generator is motivated by the Lagrangian view of an MFG and the properties of the Fokker-Planck equation. Using the neural ODE in combination with the loss calculation as in [Ruthotto et al., 2020], new costs can be treated such as an entropy running cost. Using this cost a crowd aversion problem is solved, showing that the presented algorithm works. To our knowledge, this is the first deep learning based solution algorithm for stochastic MFGs that can deal with such running costs.

Chapter 6

Future work

This thesis solves stochastic mean field games using a two-player game formulation. For this formulation, we showed that using a neural SDE or neural ODE as the generator for the agent distribution has several advantages over using a standard neural network such as a ResNet or a simple feedforward neural network. To further improve neural network based solutions of stochastic mean field games, note that, in general, saddle point problems are challenging to solve. Hence, using a minimization problem may be better than using our saddle point problem. One minimization problem is introduced in [Lin et al., 2020] and is a stochastic generalization of the used minimization problem in [Ruthotto et al., 2020]. To solve this stochastic generalization, perhaps a similar approach as in [Ruthotto et al., 2020] can be taken. In this paper, the authors use the method of characteristics on the deterministic Fokker-Planck equation to sample from the agent distribution, to deal with terminal costs such as the KL-divergence, and to deal with running costs such as an entropy. However, this does not immediately extend to the Fokker-Planck equation corresponding to an SDE. Consequently, the method as in [Ruthotto et al., 2020] can not immediately be applied. However, Theorem 2.2.4 states that solving the stochastic MFG is equivalent to solving a deterministic MFG with an additional running cost. As this formulation is similar to the variational formulation used in [Ruthotto et al., 2020], it may be possible to use the method in [Ruthotto et al., 2020] to solve the stochastic MFG. If it is possible to construct such an algorithm, it is interesting to see whether this algorithm performs better than the algorithm based on the two-player game formulation.

As mentioned in the previous paragraph, our research focuses on solving stochastic mean field games using a two-player game formulation. Another restriction of the performed research is that the Fokker-Planck equation in the MFG corresponds to an SDE with a constant isotropic diffusion matrix. It might be interesting to construct an algorithm that deals with an SDE that does not have such a constant isotropic diffusion matrix. Two examples of such cases are diffusion terms described by a nonlinear function that outputs a matrix and a learnable diffusion. To our knowledge, a deep learning based algorithm for this case has not yet been developed. As this thesis shows the benefits of a Lagrangian view inspired deep learning algorithm, it might be interesting to investigate whether the Lagrangian view can help in creating a deep learning based algorithm for the aforementioned case. A possible application of this new algorithm is learning good Markov Chain Monte Carlo algorithms using ρ-ergodic SDEs [Ma et al., 2015; Hodgkinson et al., 2020].

Furthermore, we indicated that the designed algorithm in Section 3.4 can deal with terminal costs such as a KL-divergence and running costs such as the entropy. In the numerical experiments, it is shown that the algorithm works for an entropy running cost. However, due to time restictions, no numerical experiment is done with a KL-divergence terminal cost. It is interesting to see if everything

still works properly in this case. Moreover, if it works properly, via Theorem 2.2.4 and the discussion above and below this theorem, it might help gain more insight about stochastic optimal transport in comparison to regular optimal transport.

Lastly, this thesis motivates that a neural SDE and a neural ODE are a sensible choice as generator parametrization by means of the Lagrangian view of the MFG. Given this view, the solution generator SDE of the agent distribution and the solution value function are connected as shown in equation (2.13) of Section 2.2.1. In our parametrization of the value function and the generator, we use this connection. For examples of how we use this connection, see equations (3.9) and (4.2). In particular, the drift term in the used neural SDE parametrizations is equation (2.13) with an independent copy of the value function neural network inside the Hamiltonian. However, we do not use the fact that the value function and its independent copy used in the generator should be the same at the solution. As a consequence, it might be that we can further improve the algorithm by imposing this constraint. For instance, one can add a regularization term that promotes the value function and the independent copy of the value function to be the same. We can investigate whether this regularization improves the algorithm.

References

[Achdou et al., 2014] Achdou, Y., Buera, F. J., Lasry, J.-M., Lions, P.-L., and Moll, B. (2014). Partial differential equation models in macroeconomics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 372:20130397.

[Aggarwal, 2018] Aggarwal, C. C. (2018). *Neural networks and deep learning*. Springer International Publishing, 1st edition.

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR.

[Bellman, 1997] Bellman, R. (1997). *Introduction to Matrix Analysis*. Society for Industrial and Applied Mathematics, 2nd edition.

[Benamou and Brenier, 2000] Benamou, J.-D. and Brenier, Y. (2000). A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393.

[Bishop, 2006] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag.

[Bogachev, 2007] Bogachev, V. I. (2007). *Measure theory*. Springer-Verlagg, Berlin Heidelberg, 1st edition.

[Burger et al., 2013] Burger, M., Francesco, M. D., Markowich, P., and Wolfram, M.-T. (2013). Mean field games with nonlinear mobilities in pedestrian dynamics. *arXiv preprint arXiv:1304.5201*.

[Chen et al., 2018] Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*.

[Chen et al., 2016] Chen, Y., Georgiou, T. T., and Pavon, M. (2016). On the relation between optimal transport and schrödinger bridges: A stochastic control viewpoint. *Journal of Optimization Theory and Applications*, 169(2):671–691.

[Chen et al., 2017] Chen, Y., Georgiou, T. T., and Pavon, M. (2017). Optimal transport over a linear dynamical system. *IEEE Transactions on Automatic Control*, 62(5):2137–2152.

[Cirant and Nurbekyan, 2018] Cirant, M. and Nurbekyan, L. (2018). The variational structure and time-periodic solutions for mean-field games systems. *arXiv preprint arXiv:1804.08943*.

[Dittmer et al., 2020] Dittmer, S., Schönlieb, C.-B., and Maass, P. (2020). Ground truth free denoising by optimal transport. *arXiv preprint arXiv:2007.01575*.

[E et al., 2018] E, W., Han, J., and Li, Q. (2018). A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10.

[Elamvazhuthi and Berman, 2019] Elamvazhuthi, K. and Berman, S. (2019). Mean-field models in swarm robotics: a survey. *Bioinspiration & Biomimetics*, 15(1):015001.

[Evans, 2010] Evans, L. (2010). *Partial differential equations*. American Mathematical Society, Providence, R.I, 2nd edition.

[Evans, 2006] Evans, L. C. (2006). An introduction to stochastic differential equations version 1.2. *Lecture Notes, UC Berkeley*.

[Feng et al., 2021] Feng, D., Haase-Schütz, C., Rosenbaum, L., Hertlein, H., Gläser, C., Timm, F., Wiesbeck, W., and Dietmayer, K. (2021). Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1341–1360.

[Firoozi and Caines, 2017] Firoozi, D. and Caines, P. E. (2017). An optimal execution problem in finance targeting the market trading speed: An mfg formulation. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 7–14.

[Gangbo and McCann, 1996] Gangbo, W. and McCann, R. J. (1996). The geometry of optimal transportation. *Acta Mathematica*, 177(2):113–161.

[Gomes et al., 2015] Gomes, D., Nurbekyan, L., and Pimentel, E. (2015). *Economic models and mean-field games theory*.

[Gomes et al., 2016] Gomes, D. A., Pimentel, E. A., and Voskanyan, V. (2016). *Regularity theory for mean-field game systems*. SpringerBriefs in Mathematics. Springer International Publishing, 1st edition.

[Gomes and Saúde, 2014] Gomes, D. A. and Saúde, J. (2014). Mean field games models—a brief survey. *Dynamic Games and Applications*, 4(2):110–154.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[Hodgkinson et al., 2020] Hodgkinson, L., van der Heide, C., Roosta, F., and Mahoney, M. W. (2020). Stochastic normalizing flows. *arXiv preprint arXiv:2002.09547*.

[Jia and Benson, 2019] Jia, J. and Benson, A. R. (2019). Neural jump stochastic differential equations. *arXiv preprint arXiv:1905.10403*.

[Karras et al., 2019] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 4396–4405.

[Kong et al., 2020] Kong, L., Sun, J., and Zhang, C. (2020). Sde-net: Equipping deep neural networks with uncertainty estimates. *arXiv preprint arXiv:2008.10546*.

[Lachapelle and Wolfram, 2011] Lachapelle, A. and Wolfram, M.-T. (2011). On a mean field game approach modeling congestion and aversion in pedestrian crowds. *Transportation Research Part B: Methodological*, 45(10):1572–1589.

[Lasry and Lions, 2007] Lasry, J.-M. and Lions, P.-L. (2007). Mean field games. *Japanese journal of mathematics*, 2(1):229–260.

[Léger, 2019] Léger, F. (2019). A geometric perspective on regularized optimal transport. *Journal of Dynamics and Differential Equations*, 31(4):1777–1791.

[Li et al., 2020] Li, X., Wong, T.-K. L., Chen, R. T. Q., and Duvenaud, D. (2020). Scalable gradients for stochastic differential equations. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 3870–3882. PMLR.

[Lin et al., 2020] Lin, A. T., Fung, S. W., Li, W., Nurbekyan, L., and Osher, S. J. (2020). Apac-net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games. *arXiv preprint arXiv:2002.10113*.

[Liu et al., 2019] Liu, X., Si, S., Cao, Q., Kumar, S., and Hsieh, C.-J. (2019). Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*.

[Liu et al., 2018] Liu, Z., Wu, B., and Lin, H. (2018). A mean field game approach to swarming robots control. In *2018 Annual American Control Conference (ACC)*, pages 4293–4298.

[Look et al., 2020] Look, A., Qiu, C., Rudolph, M., Peters, J., and Kandemir, M. (2020). Deterministic inference of neural stochastic differential equations. *arXiv preprint arXiv:2006.08973*.

[Lu et al., 2019] Lu, G., Zhou, Z., Song, Y., Ren, K., and Yu, Y. (2019). Guiding the one-to-one mapping in cyclegan via optimal transport. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4432–4439.

[Ma et al., 2015] Ma, Y.-A., Chen, T., and Fox, E. B. (2015). A complete recipe for stochastic gradient mcmc. *arXiv preprint arXiv:1506.04696*.

[Oganesyan et al., 2020] Oganesyan, V., Volokhova, A., and Vetrov, D. (2020). Stochasticity in neural odes: An empirical study. *arXiv preprint arXiv:2002.09779*.

[Paty and Cuturi, 2020] Paty, F.-P. and Cuturi, M. (2020). Regularized optimal transport is ground cost adversarial. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 7532–7542. PMLR.

[Pavliotis, 2014] Pavliotis, G. A. (2014). *Stochastic Processes and Applications: Diffusion Processes, the Fokker-Planck and Langevin Equations*. Texts in Applied Mathematics. Springer, New York, NY.

[Petersen and Pedersen, 2012] Petersen, K. B. and Pedersen, M. S. (2012). The matrix cookbook. `"http://www2.compute.dtu.dk/pubdb/pubs/3274-full.html"`. Version 20121115.

[Peyré and Cuturi, 2019] Peyré, G. and Cuturi, M. (2019). Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607.

[Reader et al., 2021] Reader, A. J., Corda, G., Mehranian, A., Costa-Luis, C. d., Ellis, S., and Schnabel, J. A. (2021). Deep learning for pet image reconstruction. *IEEE Transactions on Radiation and Plasma Medical Sciences*, 5(1):1–25.

[Ross, 2007] Ross, S. M. (2007). *Introduction to Probability Models*. Academic Press, ninth edition.

[Ruthotto et al., 2020] Ruthotto, L., Osher, S. J., Li, W., Nurbekyan, L., and Fung, S. W. (2020). A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193.

[Sauer, 2012] Sauer, T. (2012). *Numerical Solution of Stochastic Differential Equations in Finance*, page 529–550. Springer Berlin Heidelberg.

[Sinha and Dolz, 2021] Sinha, A. and Dolz, J. (2021). Multi-scale self-guided attention for medical image segmentation. *IEEE Journal of Biomedical and Health Informatics*, 25(1):121–130.

[Song et al., 2020] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. (2020). Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

[Toth et al., 2019] Toth, P., Rezende, D. J., Jaegle, A., Racanière, S., Botev, A., and Higgins, I. (2019). Hamiltonian generative networks. *arXiv preprint arXiv:1909.13789*.

[Tzen and Raginsky, 2019] Tzen, B. and Raginsky, M. (2019). Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*.

[Yang et al., 2019] Yang, G., Huang, X., Hao, Z., Liu, M.-Y., Belongie, S., and Hariharan, B. (2019). Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4541–4550.

[Yang and Karniadakis, 2020] Yang, L. and Karniadakis, G. E. (2020). Potential flow generator with l2 optimal transport regularity for generative models. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–11.

[Yu et al., 2021] Yu, X., Kang, C., Guttery, D. S., Kadry, S., Chen, Y., and Zhang, Y.-D. (2021). Resnet-scda-50 for breast abnormality classification. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(1):94–102.

[Zhong et al., 2019] Zhong, Y. D., Dey, B., and Chakraborty, A. (2019). Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*.

Appendix A

Stochastic differential equations and Monte Carlo simulation

This appendix gives background information on stochastic differential equations (SDEs) and Monte Carlo simulation. First, we define SDEs as a stochastic version of ordinary differential equations (ODEs). Subsequently, the Euler-Maruyama scheme for numerically solving SDEs is introduced as a generalization of the forward Euler method for ODEs. After these SDE topics, we introduce Monte Carlo simulation.

A.1 Stochastic differential equations

Many physical phenomena can be described by an ordinary differential equation. In other words, the phenomena can be described by the following equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}x = f(x,t)\,, \quad x(t_0) = x_0$$

with initial condition $x_0 \in \mathbb{R}^d$ and $f : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$. This formula can be rewritten into integral form:

$$x(t) = x_0 + \int_{t_0}^{t} f(x(\tau),\tau)\mathrm{d}\tau\,. \tag{A.1}$$

The values of $x(t)$ described by equation (A.1) are completely deterministic. As some processes are not deterministic, such as stock prices, a stochastic extension is desired. This stochastic extension is obtained by adding a stochastic element to equation (A.1). The result is a stochastic differential equation (SDE) in Itô form and is defined by:

$$X_t = X_{t_0} + \int_{t_0}^{t} f(X_\tau,\tau)\mathrm{d}\tau + \int_{t_0}^{t} \sigma(X_\tau,\tau)\mathrm{d}W_\tau\,, \tag{A.2}$$

where X_t is the random variable at time t, X_{t_0} is an initial condition, $W_t \in \mathbb{R}^m$ is a Brownian motion and $\sigma : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^{d\times m}$ is called a diffusion matrix. More often, this SDE is written in differential form as:

$$\mathrm{d}X_t = f(X_t,t)\mathrm{d}t + \sigma(X_t,t)\mathrm{d}W_t\,. \tag{A.3}$$

To understand equation (A.2), we must define the last integral. Before we define this integral, the Brownian motion W_t is further specified. A Brownian motion is a continuous-time stochastic process with the following properties:

1. W_0 equals the zero vector.
2. At each $t \geq 0$, W_t is a Gaussian random vector with mean the zero vector and tI the covariance matrix.
3. For each $t_1 < t_2$, the Gaussian random vector $W_{t_2} - W_{t_1}$ is independent of all W_t for $0 \leq t \leq t_1$.
4. Every sample path of the Brownian motion W_t is continuous almost surely.

The Brownian motion stochastic process is used in the Itô integral $\int_{t_0}^{t} \sigma(X_\tau, \tau)\mathrm{d}W_\tau$ to add stochastic behaviour to an ordinary differential equation. This Itô integral is a random variable and is defined by:

$$\int_{t_0}^{t} \sigma(X_\tau, \tau)\mathrm{d}W_\tau = \lim_{\max_i(t_i - t_{i-1}) \to 0} \sum_{i=1}^{N} \sigma(X_{t_{i-1}}, t_{i-1})(W_{t_i} - W_{t_{i-1}})\,, \tag{A.4}$$

where the convergence of the random variables is convergence in probability.

To summarize, the stochastic differential equation can be described via equation (A.2) with a deterministic part and a stochastic part defined in equation (A.4) [Evans, 2006; Sauer, 2012].

Similar to ordinary differential equations, the solution of most SDEs can not be obtained analytically. Hence, numerical schemes have to be used to simulate SDEs. The most simple method, which is also used in this thesis, is the Euler-Maruyama scheme [Sauer, 2012]. This scheme is an extension of the forward Euler method for ODEs to SDEs. The Euler-Maruyama method produces a sample path z_i. This sample path is an approximate realization of the solution stochastic process X_t, which is defined via the SDE given in equation (A.3). Given an initial condition X_0, times t_i $(i = 0, 1, \ldots, N)$, and defining $\Delta t_i = t_i - t_{i-1}$, the Euler-Maruyama scheme produces such sample paths via:

$$\begin{aligned} z_0 &= X_0\,, \\ z_{i+1} &= z_i + f(z_i, t_i)\Delta t_{i+1} + \sigma(z_i, t_i)\Delta W_{i+1}\,, \end{aligned}$$

where $\Delta W_{i+1} = W_{t_{i+1}} - W_{t_i} \sim N(0, (t_{i+1} - t_i)I)$. This approach illustrates the stochastic nature of the SDE. Given the same initial condition X_0, we obtain different paths depending on the specific sample path of the Brownian motion W_t.

The just presented Euler-Maruyama scheme is weakly convergent with order 1 and strongly convergent with order $\frac{1}{2}$. In other words, letting T denote the final time and $\Delta t = t_i - t_{i-1}$ the chosen constant step size, for a randomly produced sample Z_T at the final time, we have

$$|\,\mathbb{E}(f(X_T)) - \mathbb{E}(f(Z_T))| = O(\Delta t)$$

for all polynomials f and

$$\mathbb{E}(|X_T - Z_T|) = O((\Delta t)^{\frac{1}{2}})$$

A.2 Monte Carlo simulation

The main goal of Monte Carlo simulation is to approximate expectations given a certain probability density function $\rho(x)$ with $x \in \mathbb{R}^d$. Hence, for a continuous random variable $\mathbf{X}$, the goal of Monte

Carlo simulation is to approximate

$$\mathbb{E}\left[g(\mathbf{X})\right] = \int_{\Omega} g(x)\rho(x)\mathrm{d}x\,, \tag{A.5}$$

where Ω is the domain on which $\mathbf{X}$ takes values. This integral expression needs to be estimated as it is not always analytically possible to compute it exactly. However, standard numerical schemes for approximating integrals within a given accuracy may be too expensive. In addition, ρ is not always known. Only a method to generate samples from ρ could be known. To overcome these difficulties, Monte Carlo simulation approximates the integral via simulation.

Before we explain how Monte Carlo simulation approximates the expectation in equation (A.5), we introduce the notion of a generator for the distribution ρ. A generator $\mathbb{G}$ for the distribution ρ produces samples x from the distribution ρ. Hence, the generator $\mathbb{G}$ produces sample points x such that $x \sim \rho$.

To approximate the expectation in equation (A.5), the generator for the distribution ρ generates N independent samples x_i for some user specified integer N. Subsequently, we calculate the variables $y_i = g(x_i)$. By the strong law of large numbers, we know:

$$\lim_{r\to\infty} \frac{\sum_{i=1}^{r} y_i}{r} = \mathbb{E}\left[y_i\right] = \mathbb{E}\left[g(\mathbf{X})\right]\,.$$

From this observation, the following estimate of $\mathbb{E}\left[g(\mathbf{X})\right]$ is obtained:

$$\frac{1}{N}\sum_{i=1}^{N} y_i\,.$$

This approach to estimate $\mathbb{E}\left[g(\mathbf{X})\right]$ is called Monte Carlo simulation [Ross, 2007].

To further illustrate how we can use Monte Carlo simulation, consider the SDE given in equation (A.3) and assume $x_0 \sim \rho_0$ with x_0 the initial condition of the SDE and ρ_0 some distribution. Moreover, assume $t_0 = 0$ and that we want to know

$$\mathbb{E}\left[X_1\right] = \int_{\mathbb{R}^d} x\rho(x,1)\mathrm{d}x \tag{A.6}$$

with $\rho(\cdot,1)$ the distribution of the SDE solution values at time $t = 1$. To approximate this expectation using Monte Carlo simulation, we need samples from $\rho(\cdot,1)$. In order to get these samples, the generator for $\rho(\cdot,1)$ is needed. This generator functions as follows. First of all, the generator samples a point x_0 from the distribution ρ_0. Subsequently, given the initial condition x_0, the generator samples an SDE path from the SDE given by equation (A.3). The final generated sample x_i is the value of this sample path at $t = 1$. Finally, generating N samples using this generator, the expectation in equation (A.6) is estimated by

$$\frac{1}{N}\sum_{i=1}^{N} x_i\,.$$

Appendix B

Mollification

This appendix gives information about the standard mollifier by following [Evans, 2010]. In the proof in the next appendix, we use this concept.

Before continuing with the definition of the standard mollifier, some notation needs to be introduced.

Notation. Given a set S, let $\text{dist}(x, S) := \inf\left(\|x - s\|_2 \,\middle|\, s \in S\right)$. If $U \subset \mathbb{R}^d$, ∂U its boundary and $\epsilon > 0$, then:

$$U_\epsilon := \left\{ x \in U \,\middle|\, \text{dist}(x, \partial U) > \epsilon \right\} .$$

Using this notation, the definition of the standard mollifier and the standard mollification can be given.

Definition B.1 (Standard mollifier [Evans, 2010])**.** Define the standard mollifier $\eta \in C^\infty(\mathbb{R}^d)$ by

$$\eta(x) := \begin{cases} C \exp\left(\frac{1}{\|x\|^2 - 1}\right), & \text{if } \|x\| < 1 \\ 0, & \text{if } \|x\| \geq 1 \end{cases} ,$$

where the constant $C > 0$ is selected such that $\int_{\mathbb{R}^d} \eta(x) \mathrm{d}x = 1$.

Definition B.2 (Standard mollification [Evans, 2010])**.** For $x \in \mathbb{R}^d$, define η_ϵ as

$$\eta_\epsilon(x) := \frac{1}{\epsilon^d} \eta\left(\frac{x}{\epsilon}\right) .$$

The functions η_ϵ are C^∞ and satisfy

$$\int_{\mathbb{R}^d} \eta_\epsilon(x) \mathrm{d}x = 1 \,,\ \text{spt}(\eta_\epsilon) \subset B(0, \epsilon) \,,$$

where $\text{spt}(f)$ means the support of the function f and $B(x, \epsilon)$ is an open ball around x with radius ϵ. Using the definition of η_ϵ, we can define the standard mollification.

Assume a function $f : U \to \mathbb{R}$ is locally integrable for some set $U \subset \mathbb{R}^d$. The standard mollification of f is defined by

$$f^\epsilon(x) := (\eta_\epsilon * f)(x) \,,\ x \in U_\epsilon \,,$$

where the $*$ denotes a convolution operation. That is,

$$f^\epsilon(x) = \int_U \eta_\epsilon(x-y)f(y)\mathrm{d}y = \int_{B(0,\epsilon)} \eta_\epsilon(y)f(x-y)\mathrm{d}y\,,\ \text{for } x \in U_\epsilon\,.$$

The mollification of a function has several properties. The following theorem presents two of those properties.

Theorem B.1 (Properties of standard mollification [Evans, 2010])**.** Assume a function $f : U \to \mathbb{R}$ for $U \subset \mathbb{R}^d$ is given. Then:

- $f^\epsilon \in C^\infty(U_\epsilon)$.
- $f^\epsilon \to f$ *a.e.* as $\epsilon \to 0$.

Appendix C

Proof two-player game formulation in $\mathbb{R}^d$

Theorem. Assume the following MFG PDE system is given:

$$\begin{aligned} -\partial_t \phi(x,t) - \nu \Delta \phi(x,t) + H(x, \nabla \phi(x,t)) &= F(x, \rho(\cdot,t)) \\ \partial_t \rho(x,t) - \nu \Delta \rho(x,t) - \nabla \cdot (\rho(x,t) \nabla_p H(x, \nabla \phi(x,t))) &= 0 \\ \rho(x,0) = \rho_0(x), \quad \phi(x,T) &= G(x, \rho(\cdot,T)) \,, \end{aligned}$$

where $\rho_0(x) \in C^2(\mathbb{R}^d)$. Defining $C_c^2(\mathbb{R}^d)$ as the set of twice continuously differentiable functions with compact support, let F and G be functionals that are defined via:

- $\lim_{h \to 0} \frac{\mathcal{F}(\rho + hm) - \mathcal{F}(\rho)}{h} = \int_{\mathbb{R}^d} F(x,\rho) m(x) \mathrm{d}x$ for all $m \in C_c^2(\mathbb{R}^d)$. This is a variational derivative of some functional $\mathcal{F} : C^2(\mathbb{R}^d) \to \mathbb{R}$. We assume $\mathcal{F}$ is bounded.
- $\lim_{h \to 0} \frac{\mathcal{G}(\rho + hm) - \mathcal{G}(\rho)}{h} = \int_{\mathbb{R}^d} G(x,\rho) m(x) \mathrm{d}x$ for all $m \in C_c^2(\mathbb{R}^d)$. This is a variational derivative of some functional $\mathcal{G} : C^2(\mathbb{R}^d) \to \mathbb{R}$. We assume $\mathcal{G}$ is bounded.

Define the functional Ψ as:

$$\begin{aligned} \Psi(\rho, \phi) = \int_0^T \left\{ \int_{\mathbb{R}^d} \left(-\phi_t(x,t) - \nu \Delta \phi(x,t) + H(x, \nabla \phi(x,t)) \right) \rho(x,t) \mathrm{d}x - \mathcal{F}(\rho(\cdot,t)) \right\} \mathrm{d}t \\ + \int_{\mathbb{R}^d} \rho(x,T) \phi(x,T) \mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x) \phi(x,0) \mathrm{d}x - \mathcal{G}(\rho(\cdot,T)) \,, \end{aligned}$$

Assume that H is continuous in both arguments. Moreover, assume that the functionals $F(x,\rho) : \mathbb{R}^d \times C^2(\mathbb{R}^d)$ and $G(x,\rho) : \mathbb{R}^d \times C^2(\mathbb{R}^d)$ are continuous in x given ρ. If $(\rho^*, \phi^*) \in C^2(\Omega_T) \times C^2(\Omega_T)$ is a saddle point of the functional Ψ:

$$\begin{aligned} \Psi(\rho^*, \phi^*) &= \sup_{\rho \in C^2(\Omega_T)} \left(\Psi(\rho, \phi^*) \right) , \\ \Psi(\rho^*, \phi^*) &= \inf_{\phi \in C^2(\Omega_T)} \left(\Psi(\rho^*, \phi) \right) , \end{aligned}$$

and $\Psi(\rho^*, \phi^*)$ is finite, then (ρ^*, ϕ^*) is a smooth solution to the PDE system.

Proof. To prove the statement, we use a calculus of variations approach on Ψ. First, we prove that the solution to the saddle point problem must satisfy the Hamilton-Jacobi-Bellmann equation in combination with the boundary condition for ϕ. Subsequently, we prove that the solution should satisfy the Fokker-Planck equation and the initial condition for ρ.

Solution should satisfy HJB equation and terminal condition for ϕ
Given the saddle point (ρ^*, ϕ^*), define for $h \in \mathbb{R}$:

$$I_1(h) = \Psi(\rho^* + h \cdot m, \phi^*) = \Psi(\rho_h, \phi^*)\,, \text{ for } m \in C^2(\Omega_T), \quad \bigcup_{t\in[0,T]} \text{spt}(m(\cdot,t)) \subset U_m\,,$$

where $\text{spt}(f)$ is the support of f, $\Omega_T = \mathbb{R}^d \times [0,T]$ and $U_m \subset \mathbb{R}^d$ is some compact set depending on the function m.

First, we show that given a $m \in M := \{m(x,t) \in C^2(\Omega_T) | \bigcup_{t\in[0,T]} \text{spt}(m(\cdot,t)) \subset U_m, U_m \subset \mathbb{R}^d \text{ any compact set}\}$, $I_1(h)$ is finite for all $h \in \mathbb{R}$. The functions $m \in M$ have that $\text{spt}(m(\cdot,t)) \subset U_m$ for all $t \in [0,T]$.

Calculating $I_1(h)$ for $m \in M$ gives

$$\begin{aligned} I_1(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))\rho_h(x,t)\mathrm{d}x - \mathcal{F}(\rho_h(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho_h(x,T)\phi^*(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi^*(x,0)\mathrm{d}x - \mathcal{G}(\rho_h(\cdot,T))\,. \end{aligned}$$

Using the definition of ρ_h, several integrals can be split into two:

$$\begin{aligned} I_1(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))\rho^*(x,t)\mathrm{d}x - \mathcal{F}(\rho_h(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)\phi^*(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi^*(x,0)\mathrm{d}x - \mathcal{G}(\rho_h(\cdot,T)) \\ &+ h\int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))m(x,t)\mathrm{d}x\mathrm{d}t \\ &+ h\int_{\mathbb{R}^d} m(x,T)\phi^*(x,T)\mathrm{d}x \end{aligned}$$

The integrals on the last two lines are finite by the assumption that $m \in M$, $\phi^* \in C^2(\Omega_T)$ and H is continuous in both arguments. Subsequently, we add a clever zero by adding $-\mathcal{F}(\rho^*(\cdot,t)) + \mathcal{F}(\rho^*(\cdot,t))$ and $-\mathcal{G}(\rho^*(\cdot,T)) + \mathcal{G}(\rho^*(\cdot,T))$. Using $D_{\mathcal{F}}(h,t) = (\mathcal{F}(\rho^*(\cdot,t)) - \mathcal{F}(\rho_h(\cdot,t)))$ and $D_{\mathcal{G}}(h) = (\mathcal{G}(\rho^*(\cdot,T)) - \mathcal{G}(\rho_h(\cdot,T)))$, we get:

$$\begin{aligned} I_1(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))\rho^*(x,t)\mathrm{d}x - \mathcal{F}(\rho^*(\cdot,t)) + D_{\mathcal{F}}(h,t) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)\phi^*(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi^*(x,0)\mathrm{d}x - \mathcal{G}(\rho^*(\cdot,T)) + D_{\mathcal{G}}(h) \\ &+ h\int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))m(x,t)\mathrm{d}x\mathrm{d}t \\ &+ h\int_{\mathbb{R}^d} m(x,T)\phi^*(x,T)\mathrm{d}x \end{aligned}$$

As $\mathcal{F}$ is assumed bounded, the time integral over $D_{\mathcal{F}}$ is finite and we can get this integral outside the big integral. Doing so, we get:

$$\begin{aligned} I_1(h) &= \Psi(\rho^*, \phi^*) + h \int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))m(x,t)\mathrm{d}x\mathrm{d}t \\ &\quad + \int_0^T D_{\mathcal{F}}(h,t)\mathrm{d}t + h\int_{\mathbb{R}^d} m(x,T)\phi^*(x,T)\mathrm{d}x + D_{\mathcal{G}}(h) \\ &= \Psi(\rho^*, \phi^*) + h \int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))m(x,t)\mathrm{d}x\mathrm{d}t \\ &\quad + \int_0^T \mathcal{F}(\rho^*(\cdot,t)) - \mathcal{F}(\rho_h(\cdot,t))\mathrm{d}t + h\int_{\mathbb{R}^d} m(x,T)\phi^*(x,T)\mathrm{d}x + \mathcal{G}(\rho^*(\cdot,T)) - \mathcal{G}(\rho_h(\cdot,T)) \end{aligned} \tag{C.1}$$

By our earlier argument, the integral over space and time and the $m(x,T)\phi^*(x,T)$ integral are finite. Moreover, as $\mathcal{F}$ and $\mathcal{G}$ are bounded, the integral over the $\mathcal{F}$ functions and the evaluation of the $\mathcal{G}$ functions are well-defined. As $\Psi(\rho^*, \phi^*)$ is finite by assumption, all the terms in the last expression are finite. Hence, $I_1(h)$ is finite for all $h \in \mathbb{R}$.

From the last two lines in equation (C.1), it becomes clear that $I_1(h)$ is differentiable at $h = 0$. The reason is that the integrals after the h values have a finite value and that by definition of the variational derivatives F and G, $\frac{\mathrm{d}}{\mathrm{d}h}\mathcal{F}(\rho^*(\cdot,t) + hm(\cdot,t))|_{h=0}$ and $\frac{\mathrm{d}}{\mathrm{d}h}\mathcal{G}(\rho^*(\cdot,T) + hm(\cdot,T))|_{h=0}$ exist. Hence, taking the derivative of the last equation in (C.1) yields:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}h}I_1(h) &= \int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)))m(x,t)\mathrm{d}x - \frac{\mathrm{d}}{\mathrm{d}h}(\mathcal{F}(\rho_h(\cdot,t))) \right\} \mathrm{d}t \\ &\quad + \int_{\mathbb{R}^d} m(x,T)\phi^*(x,T)\mathrm{d}x - \frac{\mathrm{d}}{\mathrm{d}h}(\mathcal{G}(\rho_h(\cdot,T)))\,. \end{aligned}$$

From the definition of (ρ^*, ϕ^*), $I_1(0) \geq I_1(h)$ for all $h \in \mathbb{R}$. So $I_1(0) = \min_{h\in\mathbb{R}} I_1(h)$. From this it can be concluded that $\frac{\mathrm{d}I_1}{\mathrm{d}h}(0) = 0$. Using the definition of F and G and using that $m \in M$ is arbitrary, one gets

$$\begin{aligned} \frac{\mathrm{d}I_1}{\mathrm{d}h}(0) &= \int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)) - F(x, \rho^*(\cdot,t)))\, m(x,t)\mathrm{d}x\mathrm{d}t \\ &\quad + \int_{\mathbb{R}^d} m(x,T)(\phi^*(x,T) - G(x, \rho^*(\cdot,t)))\mathrm{d}x = 0\,,\ \forall m \in M\,. \end{aligned}$$

Choosing m such that $m(x,T) = 0$, one gets

$$\begin{aligned} \frac{\mathrm{d}I_1}{\mathrm{d}h}(0) &= \int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)) - F(x, \rho^*(\cdot,t)))\, m(x,t)\mathrm{d}x\mathrm{d}t \\ &= 0\,. \end{aligned} \tag{C.2}$$

As $\phi^*, \rho^* \in C^2(\Omega_T)$, H is continuous in both its arguments and F is continuous in x given ρ^*:

$$(-\phi_t^* - \nu\Delta\phi^* + H(\cdot, \nabla\phi^*) - F(\cdot, \rho^*)) \in C(\Omega_T)\,.$$

Take any point $(x,t) \in \mathbb{R}^d \times (0,T)$. Assume that $(-\phi_t^* - \nu\Delta\phi^* + H(\cdot, \nabla\phi^*) - F(\cdot, \rho^*)) > 0$ at (x,t). By continuity, there exists an open ball $B((x,t),\epsilon)$ of radius ϵ such that at all $(\tilde{x}, \tilde{t}) \in$

$U := B((x,t),\epsilon)$, one has $(-\phi_t^* - \nu\Delta\phi^* + H(\cdot,\nabla\phi^*) - F(\cdot,\rho^*)) > 0$. Using the definition of $U_{\epsilon/3}$ as introduced in Appendix B, letting $I_{U_{\epsilon/3}}(x) = 1$ if $x \in U_{\epsilon/3}$ and 0 otherwise, define $m \in M$ as:

$$m(x,t) = (\eta_{\epsilon/3} * I_{U_{\epsilon/3}})(x,t)\,, \quad \text{spt}(m(\cdot,t)) \subset U\,,$$

where η_δ is the standard mollifier (for a proper definition, see Appendix B). For this choice of $m(x,t)$, one gets:

$$\frac{\mathrm{d}I_1}{\mathrm{d}h}(0) = \int_0^T \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x,\nabla\phi^*(x,t)) - F(x,\rho^*(\cdot,t)))\, m(x,t)\mathrm{d}x\mathrm{d}t > 0\,,$$

which contradicts equation (C.2). Hence, $(-\phi_t^* - \nu\Delta\phi^* + H(\cdot,\nabla\phi^*) - F(\cdot,\rho^*)) > 0$ at (x,t) can not happen. By employing the same strategy when $(-\phi_t^* - \nu\Delta\phi^* + H(x,\nabla\phi^*) - F(x,\rho^*)) < 0$ at (x,t), one can conclude that $(-\phi_t^* - \nu\Delta\phi^* + H(\cdot,\nabla\phi^*) - F(\cdot,\rho^*))) = 0$ at $(x,t) \in \mathbb{R}^d \times (0,T)$.

Relaxing the constraint that $m(x,T) = 0$, one gets that

$$\frac{\mathrm{d}I_1}{\mathrm{d}h}(0) = \int_{\mathbb{R}^d} m(x,T)(\phi^*(x,T) - G(x,\rho^*(\cdot,T)))\mathrm{d}x = 0\,,\ \forall m \in M\,.$$

As $m(\cdot,T) \in C_c^2(\mathbb{R}^d)$ is arbitrary and following the same reasoning as before, one gets $\phi^*(x,T) = G(x,\rho^*(\cdot,T))$. Hence, we proved that ϕ^* should satisfy the Hamilton-Jacobi-Bellmann equation and the corresponding condition at $t = T$.

Solution should satisfy the Fokker-Planck equation and initial condition for ρ

To see that ρ^* should satisfy a Fokker-Planck equation, define:

$$I_2(h) = \Psi(\rho^*, \phi^* + h\cdot u) = \Psi(\rho^*,\phi_h)\,,\ \text{for } u \in C^2(\Omega_T)\,,\ \bigcup_{t\in[0,T]} \text{spt}(u(\cdot,t)) \subset U_u\,.$$

Similar to when we dealt with I_1, we first show that I_2 has a finite value for all $h \in \mathbb{R}$ and $u \in M := \{u(x,t) \in C^2(\Omega_T) | \bigcup_{t\in[0,T]} \text{spt}(u(\cdot,t)) \subset U_u, U_u \subset \mathbb{R}^d \text{ any compact set}\}$. We proceed in the same way as when we dealt with I_1. Calculating $I_2(h)$ for $u \in M$ gives

$$\begin{aligned} I_2(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-(\phi_h)_t(x,t) - \nu\Delta\phi_h(x,t) + H(x,\nabla\phi_h(x,t)))\rho^*(x,t)\mathrm{d}x - \mathcal{F}(\rho^*(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)\phi_h(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi_h(x,0)\mathrm{d}x - \mathcal{G}(\rho^*(\cdot,T))\,. \end{aligned}$$

Using the definition of ϕ_h, several integrals can be split into two:

$$\begin{aligned} I_2(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x,\nabla\phi_h(x,t)))\rho^*(x,t)\mathrm{d}x - \mathcal{F}(\rho^*(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)\phi^*(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi^*(x,0)\mathrm{d}x - \mathcal{G}(\rho^*(\cdot,T)) \\ &+ h\int_0^T \int_{\mathbb{R}^d} (-u_t(x,t) - \nu\Delta u(x,t))\rho^*(x,t)\mathrm{d}x\mathrm{d}t \\ &+ h\int_{\mathbb{R}^d} \rho^*(x,T)u(x,T)\mathrm{d}x - h\int_{\mathbb{R}^d} \rho_0(x)u(x,0)\mathrm{d}x \end{aligned}$$

The integrals on the last two lines are finite by the assumption that $u \in M$, $\rho^* \in C^2(\Omega_T)$ and $\rho_0 \in C^2(\mathbb{R}^d)$. As when we dealt with $I_1(h)$, we add a clever zero by adding $H(x, \nabla\phi^*(x,t)) - H(x, \nabla\phi^*(x,t))$. Defining $D_H(h,x,t) = H(x, \nabla\phi_h(x,t)) - H(x, \nabla\phi^*(x,t))$, we get:

$$\begin{aligned} I_2(h) = &\int_0^T \left\{ \int_{\mathbb{R}^d} (-\phi_t^*(x,t) - \nu\Delta\phi^*(x,t) + H(x, \nabla\phi^*(x,t)) + D_H(h,x,t))\rho^*(x,t)\mathrm{d}x - \mathcal{F}(\rho^*(\cdot,t)) \right\} \mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)\phi^*(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)\phi^*(x,0)\mathrm{d}x - \mathcal{G}(\rho^*(\cdot,T)) \\ &+ h\int_0^T \int_{\mathbb{R}^d} (-u_t(x,t) - \nu\Delta u(x,t))\rho^*(x,t)\mathrm{d}x\mathrm{d}t \\ &+ h\int_{\mathbb{R}^d} \rho^*(x,T)u(x,T)\mathrm{d}x - h\int_{\mathbb{R}^d} \rho_0(x)u(x,0)\mathrm{d}x \end{aligned}$$

Noting that $D_H(h,x,t) = 0$ for $x \notin U_u$, where U_u is the compact set belonging to $u \in M$, the integral of D_H over $\mathbb{R}^d$ turns into an integral over the compact domain U_u. As $D_H(h,x,t)$ is continuous in (x,t), the integral over the compact set $U_u \times [0,T]$ is finite. Based on these observations, we can rewrite $I_2(h)$ to:

$$\begin{aligned} I_2(h) = &\Psi(\rho^*, \phi^*) + h\int_0^T \int_{\mathbb{R}^d} (-u_t(x,t) - \nu\Delta u(x,t))\rho^*(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_0^T \int_{\mathbb{R}^d} (H(x, \nabla\phi_h(x,t)) - H(x, \nabla\phi^*(x,t)))\rho^*(x,t)\mathrm{d}x\mathrm{d}t \\ &+ h\int_{\mathbb{R}^d} \rho^*(x,T)u(x,T)\mathrm{d}x - h\int_{\mathbb{R}^d} \rho_0(x)u(x,0)\mathrm{d}x \end{aligned} \tag{C.3}$$

By the earlier considerations, we can conclude that the integrals after the value of h are finite. As $\Psi(\rho^*, \phi^*)$ is assumed finite and the integral involving the Hamiltonian H is finite by our earlier considerations, $I_2(h)$ has a finite value for all $h \in \mathbb{R}$.

We have shown that $I_2(h)$ has a finite value for all $h \in \mathbb{R}$. Moreover, using equation (C.3), it becomes clear that the derivative with respect to h exists and equals:

$$\begin{aligned} \frac{\mathrm{d}I_2}{\mathrm{d}h}(h) = &\int_0^T \int_{\mathbb{R}^d} (-u_t(x,t) - \nu\Delta u(x,t))\rho^*(x,t) + \rho^*(x,t)\langle \nabla_p H(x, \nabla\phi_h(x,t)), \nabla_x u(x,t)\rangle_{l^2} \mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{R}^d} \rho^*(x,T)u(x,T)\mathrm{d}x - \int_{\mathbb{R}^d} \rho_0(x)u(x,0)\mathrm{d}x\,. \end{aligned}$$

Note that all the present integrals have a finite value as $u \in M$ and the integrands are continuous. As ρ^* and ϕ_h are sufficiently smooth, integration by parts can be applied. Applying integration by parts to rewrite the above equation, we get

$$\begin{aligned} \frac{\mathrm{d}I_2}{\mathrm{d}h}(h) = &\int_0^T \int_{\mathbb{R}^d} (\rho_t^*(x,t) - \nu\Delta\rho^*(x,t))u(x,t) + \rho^*(x,t)\langle \nabla_p H(x, \nabla\phi_h(x,t)), \nabla_x u(x,t)\rangle_{l^2} \mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{R}^d} u(x,0)(\rho^*(x,0) - \rho_0(x))\mathrm{d}x + \int_0^T \int_{\partial\mathbb{R}^d} (u\nabla\rho^* - \rho^*\nabla u)\cdot\hat{n}\mathrm{d}\Gamma\mathrm{d}t\,, \end{aligned}$$

where

$$\int_{\partial\mathbb{R}^d} (u\nabla\rho^* - \rho^*\nabla u)\cdot\hat{n}\mathrm{d}\Gamma = \lim_{r\to\infty} \int_{\partial B(0,r)} (u(x,t)\nabla\rho^*(x,t) - \rho^*(x,t)\nabla u(x,t))\cdot\hat{n}\mathrm{d}\Gamma$$

with $\hat{n}$ the outward normal of the ball $B(0,r)$. However, note that the boundary integral vanishes as $u \in M$.

Removing this boundary integral, we can again rewrite the equation. We do this by using $\nabla \cdot (uv) = \langle \nabla u, v \rangle_{l^2} + u \nabla \cdot v$ and Gauss' theorem. Using this, one gets:

$$\begin{aligned}\frac{\mathrm{d}I_2}{\mathrm{d}h}(h) = &\int_0^T \int_{\mathbb{R}^d} (\rho_t^*(x,t) - \nu\Delta\rho^*(x,t) - \nabla \cdot (\rho^*(x,t)\nabla_p H(x, \nabla\phi_h(x,t))))u(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{R}^d} u(x,0)(\rho^*(x,0) - \rho_0(x))\mathrm{d}x + \int_0^T \int_{\partial\mathbb{R}^d} (u\rho^*\nabla_p H(x, \nabla\phi_h)) \cdot \hat{n}\mathrm{d}\Gamma\mathrm{d}t\,,\end{aligned}$$

Again note that the boundary integral vanishes as $u \in M$. Hence, we are left with

$$\begin{aligned}\frac{\mathrm{d}I_2}{\mathrm{d}h}(h) = &\int_0^T \int_{\mathbb{R}^d} (\rho_t^*(x,t) - \nu\Delta\rho^*(x,t) - \nabla \cdot (\rho^*(x,t)\nabla_p H(x, \nabla\phi_h(x,t))))u(x,t)\mathrm{d}x\mathrm{d}t \\ &+ \int_{\mathbb{R}^d} u(x,0)(\rho^*(x,0) - \rho_0(x))\mathrm{d}x\,,\end{aligned}$$

Subsequently, noting that $u(x,t)$ is arbitrary, evaluating $\frac{\mathrm{d}I_2}{\mathrm{d}h}$ at $h = 0$ and employing the same considerations as at $I_1(h)$, one concludes that the following must hold:

$$\begin{aligned}\rho_t^*(x,t) - \nu\Delta\rho^*(x,t) - \nabla \cdot (\rho^*(x,t)\nabla_p H(x, \nabla\phi^*(x,t))) = 0\,,\\ \rho^*(x,0) = \rho_0(x), \quad (x,t) \in \mathbb{R}^d \times [0,T]\,.\end{aligned}$$

Hence, ρ^* should satisfy the Fokker-Planck equation. As earlier it is shown that ϕ^* should satisfy the Hamilton-Jacobi-Bellmann equations, the proof is finished. ■

Appendix D

Proof of Theorem 3.4.1

Before presenting and proving this theorem, we introduce a lemma:

Lemma D.1 (From [Bellman, 1997])**.** Let $X(t)$ be a matrix-valued function on $t \in [0, T]$ satisfying:

$$\frac{\mathrm{d}X}{\mathrm{d}t} = A(t)X, \quad X(0) = I\,,$$

where $A(t)$ is some matrix and I is the identity matrix. Moreover, let $|X(t)| := \det(X(t))$. Then:

$$|X(t)| = \exp\left(\int_0^t \mathrm{Tr}(A(s))\mathrm{d}s\right)\,.$$

Using this lemma, we can prove Theorem 3.4.1. For convenience, the theorem is restated:

Theorem (From [Ruthotto et al., 2020])**.** Let ρ satisfy the following continuity equation

$$\frac{\partial}{\partial t}\rho(x,t) + \nabla \cdot (\rho(x,t)v(x,t)) = 0$$

with initial condition $\rho(x,0) = \rho_0(x)$. Moreover, let $z(x,t)$ be the solution to

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}z(x,t) &= v(z(x,t),t)\,,\\ z(x,0) &= x\,.\end{aligned}$$

Then the following identity holds

$$\rho(z(x,t),t)\det(\nabla_x z(x,t)) = \rho_0(x)\,.$$

Proof. The proof is divided into two proofs/steps:

1. Solutions to $\frac{\mathrm{d}}{\mathrm{d}t}z(x,t) = v(z(x,t),t)$ are characteristics of the Fokker-Planck equation $\frac{\partial}{\partial t}\rho(x,t) + \nabla \cdot (\rho(x,t)v(x,t)) = 0$. This gives us $\rho(z(x,t),t) = \rho(x,0)\exp\left(-\int_0^t \tilde{A}(z(x,s),s)\mathrm{d}s\right)$ for some function $\tilde{A}$.

2. It is shown that $\det(\nabla_x z(x,t)) = \exp\left(\int_0^t \tilde{A}(z(x,s),s)\mathrm{d}s\right)$. Combining this result with the earlier result finishes the proof.

ODE solutions are characteristics of the Fokker-Planck equation:
The method of characteristics is a technique that solves a PDE by converting the PDE into a system of ODEs. The idea is to find a characteristic curve along which the value of the PDE solution can be calculated. For more information on the method of characteristics, see [Evans, 2010].

For our purposes, the method of characteristics for quasilinear PDE suffices. This is presented first. Assume one has the following PDE.

$$\sum_{i=1}^{n} a_i(x,u)\frac{\partial}{\partial x_i}u = c(x,u)\,,$$

where x contains all independent variables and a_i and c are some functions. Note that x can contain variables describing the spatial location as well as an independent time variable. The method of characteristics uses the following ODE system to calculate the solution u to the quasilinear PDE:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}s}x_i(s) &= a_i(x(s),u(s))\,,\\ \frac{\mathrm{d}}{\mathrm{d}s}u(s) &= c(x(s),u(s))\,,\end{aligned}$$

where s is an independent variable used for indicating the position along the characteristic, $x(s)$ is the characteristic curve and u is the solution to the PDE along the characteristic curve.

In our case, letting $x \in \mathbb{R}^d$, the Fokker-Planck equation is a quasilinear PDE as it can be rewritten to:

$$\begin{aligned}&\frac{\partial}{\partial t}\rho(x,t) + \sum_{i=1}^{d}\frac{\partial}{\partial x_i}(\rho(x,t))v_i(x,t)\\ &\qquad = -\rho(x,t)\sum_{i=1}^{d}\frac{\partial}{\partial x_i}v_i(x,t)\,.\end{aligned}$$

Now define:

$$\begin{aligned}a_0(t,x,\rho) &= 1\,,\\ a_i(t,x,\rho) &= v_i(x,t) \quad (i=1,\dots,d)\,,\\ c(t,x,\rho) &= -\rho\sum_{i=1}^{d}\frac{\partial}{\partial x_i}v_i(x,t)\,.\end{aligned} \tag{D.1}$$

These definitions nicely illustrate how to create the system of ODEs used in the method of characteristics. This system is:

$$\begin{aligned}\frac{\mathrm{d}t}{\mathrm{d}s} &= 1\,,\\ \frac{\mathrm{d}x_i}{\mathrm{d}s} &= a_i(t,x,\rho) = v_i(x,t) \quad (i=1,\dots,d)\,,\\ \frac{\mathrm{d}\rho}{\mathrm{d}s} &= c(t,x,\rho) = -\rho\sum_{i=1}^{d}\frac{\partial}{\partial x_i}v_i(x,t)\,.\end{aligned}$$

From the first equation, it becomes apparent that one can replace s with t and ignore the first equation afterwards. Moreover, note that the second equation is precisely the agent dynamics. Hence, the characteristic curves are precisely the agent dynamics.

Define $\tilde{A}$ as

$$\begin{aligned}\tilde{A}(x,t) &= \sum_{i=1}^{d} \frac{\partial}{\partial x_i} v_i(x,t) \\ &= \nabla_x \cdot v(x,t)\,.\end{aligned}$$

Moreover, define $z(x,t)$ as the solution to

$$\begin{aligned}\frac{\mathrm{d}z(x,t)}{\mathrm{d}t} &= v(z(x,t),t)\,, \\ z(x,0) &= x\,.\end{aligned} \tag{D.2}$$

Replacing x in equation (D.1) by $z(x,t)$ and using the last equation of equation (D.1), one gets

$$\begin{aligned}\rho(z(x,t),t) &= \rho(z(x,0),0) \exp\left(-\int_0^t \tilde{A}(z(x,s),s)\mathrm{d}s\right) \\ &= \rho_0(x) \exp\left(-\int_0^t \tilde{A}(z(x,s),s)\mathrm{d}s\right),\end{aligned}$$

where $\rho(x,0) = \rho_0(x)$ is the initial condition.

Proving specific formula Jacobian determinant

In this part, we want to show:

$$\det(\nabla_x z(x,t)) = \exp\left(\int_0^t \tilde{A}(z(x,s),s)\mathrm{d}s\right), \tag{D.3}$$

where

$$\tilde{A}(x,t) = \nabla_x \cdot v(x,t)\,. \tag{D.4}$$

This result is proven via Lemma D.1. To use this lemma, a matrix ODE should be formed for $\nabla_x z(x,t)$:

$$\begin{aligned}\frac{\partial \nabla_x z(x,t)}{\partial t} &= \nabla_x \frac{\partial z(x,t)}{\partial t} \\ &= \nabla_x v(z(x,t),t)\,,\end{aligned}$$

where the last equation follows from the fact that $z(x,t)$ is a solution to the agent dynamics (see equation (D.2)). Using the chain rule for matrices, one gets:

$$\frac{\partial \nabla_x z(x,t)}{\partial t} = \nabla_z v(z(x,t),t) \nabla_x z(x,t)\,.$$

Now define:

$$A(t) := \nabla_z v(z(x,t),t)\,.$$

Given a specific value of x and the calculated $z(x,t)$, this is a function of time only. Using this notation and noting that $\nabla_x z(x,0) = \nabla_x x = I$, the following ODE is obtained:

$$\frac{\partial \nabla_x z(x,t)}{\partial t} = A(t) \nabla_x z(x,t), \quad \nabla_x z(x,0) = I\,.$$

Note that this is the type of matrix ODE that is used in Lemma D.1. Applying this lemma gives:

$$|\nabla_x z(x,t)| = \exp\left(\int_0^t \operatorname{Tr}\left(A(s)\right) \mathrm{d}s\right).$$

As equation (D.3) needs to be shown, it suffices to prove $\operatorname{Tr}(A(t)) = \tilde{A}(z(x,t),t)$. Calculating the trace of $A(t)$ gives

$$\begin{aligned}\operatorname{Tr}(A(t)) &= \operatorname{Tr}\left(\nabla_z v(z(x,t),t)\right)\\ &= \nabla_z \cdot v(z(x,t),t)\\ &= \tilde{A}(z(x,t),t)\,,\end{aligned}$$

where the last equality follows from equation (D.4). Hence, one sees that $\operatorname{Tr}(A(t)) = \tilde{A}(z(x,t),t)$ holds. This finishes the proof. ■