An Introduction to Flow Matching and Diffusion Models

Peter Holderrieth and Ezra Erives

Website: https://diffusion.csail.mit.edu/

1 Introduction

> Creating noise from data is easy; creating data from noise is generative modeling.
>
> Song et al. [10]

1.1 Overview

In recent years, we all have witnessed a tremendous revolution in artificial intelligence (AI). Image generators like *Nano Banana* or *Stable Diffusion 3* can generate photorealistic and artistic images across a diverse range of styles, video models like Meta's *VEO-3* can generate highly realistic movie clips, and large language models like *ChatGPT* can generate seemingly human-level responses to text prompts. At the heart of this revolution lies a new ability of AI systems: the ability to **generate** objects. While previous generations of AI systems were mainly used for **prediction**, these new AI system are creative: they dream or come up with new objects based on user-specified input. Such **generative AI** systems are at the core of this recent AI revolution.

The goal of this class is to teach you two of the most widely used generative AI algorithms: **denoising diffusion models** [11] and **flow matching** [5, 7, 1, 6]. These models are the backbone of the best image, audio, and video generation models (e.g., *Nano Banana*, *FLUX*, or *VEO-3*), and have most recently became the state-of-the-art in scientific applications such as protein structures (e.g., *AlphaFold3* is a diffusion model). Without a doubt, understanding these models is truly an extremely useful skill to have.

All of these generative models generate objects by iteratively converting **noise** into **data**. This evolution from noise to data is facilitated by the simulation of **ordinary or stochastic differential equations (ODEs/SDEs)**. Flow matching and denoising diffusion models are a family of techniques that allow us to construct, train, and simulate, such ODEs/SDEs at large scale with deep neural networks. While these models are rather simple to implement, the technical nature of SDEs can make these models difficult to understand. In this course, our goal is to provide a self-contained introduction to the necessary mathematical toolbox regarding differential equations to enable you to systematically understand these models. Beyond being widely applicable, we believe that the theory behind flow and diffusion models is elegant in its own right. Therefore, most importantly, we hope that this course will be a lot of fun to you.

Remark 1 (Additional Resources)
While these lecture notes are self-contained, there are two additional resources that we encourage you to use:

1. **Lecture recordings:** These guide you through each section in a lecture format.
2. **Labs:** These guide you in implementing your own diffusion model from scratch. We highly recommend that you "get your hands dirty" and code.

You can find these on our course website: https://diffusion.csail.mit.edu/.

1.2 Course Structure

We give a brief overview over of this document. Sections 1-6 represent the core "canonical" ingredients for diffusion models, while sections 7 and 8 are advanced topics and optional.

- **Section 1, Generative Modeling as Sampling:** We formalize what it means to "generate" an image, video, protein, etc. We will translate the problem of e.g., "how to generate an image of a dog?" into the more precise problem of sampling from a probability distribution.
- **Section 2, Flow and Diffusion Models:** We explain the machinery of generation. As you can guess by the name of this class, this machinery consists of simulating ordinary and stochastic differential equations. We provide an introduction to differential equations and explain how to use them to construct generative models.
- **Section 3, Flow Matching:** Next, we explain and derive *flow matching*, a simple and scalable algorithm lying at the core of all afore-mentioned large-scale generative models such as Stable Diffusion, Nano Banana, or SORA.
- **Section 4, Score Matching:** We study *score functions* and how they can be learnt via *score matching*. Not only is this the training algorithm for diffusion models, but it unlocks SDE sampling and guidance.
- **Section 5, Guidance:** We learn how to condition our samples on a prompt (e.g. "an image of a cat") and how we can enforce adherence to such a prompt.
- **Section 6, Large-Scale Image and Video Generators:** We discuss how one builds large-scale image and video generators such as *Nano Banana*. This includes common neural network architectures and how to build things in latent space. We also survey state-of-the-art models.
- **Section 7 (Optional), Distillation:** We learn how to accelerate diffusion models. Increasing the speed of these models is of interest for real-time interactive applications such as world models (e.g. *Genie 3*).
- **Section 8 (Optional), Discrete Diffusion Models:** We learn how to translate the principles of diffusion models from Euclidean space to discrete data such as language. This enables the construction of large language models using the principles of diffusion models.

Required background. Due to the technical nature of this subject, we recommend some base level of mathematical maturity, and in particular some familiarity with probability theory. For this reason, we included a brief reminder section on probability theory in Section A. Don't worry if some of the concepts there are unfamiliar to you.

1.3 Generative Modeling As Sampling

Let's begin by thinking about various data types, or **data modalities**, that we might encounter, and how we will go about representing them numerically:

1. **Image:** Consider images with $H \times W$ pixels where H describes the height and W the width of the image, each with three color channels (RGB). For every pixel and every color channel, we are given an intensity value in $\mathbb{R}$. Therefore, an image can be represented by an element $z \in \mathbb{R}^{H \times W \times 3}$.

2. **Video:** A video is simply a series of images in time. If we have T time points or **frames**, a video would therefore be represented by an element $z \in \mathbb{R}^{T \times H \times W \times 3}$.

3. **Molecular structure:** A naive way would be to represent the structure of a molecule by a matrix $z = (z^1, \dots, z^N) \in \mathbb{R}^{3 \times N}$ where N is the number of atoms in the molecule and each $z^i \in \mathbb{R}^3$ describes the location of that atom. Of course, there are other, more sophisticated ways of representing such a molecule.

In all of the above examples, the object that we want to generate can be mathematically represented as a vector (potentially after flattening). Therefore, throughout this document, we will have:

Key Idea 1 (Objects as Vectors)
We identify the objects being generated as vectors $z \in \mathbb{R}^d$.

A notable exception to the above is text data, which is typically modeled as a discrete object by language models (such as *ChatGPT*). While continuous data $z \in \mathbb{R}^d$ is our main focus, we also study text generation in Section 8.

Generation as Sampling. Let us define what it means to "generate" something. For example, let's say we want to generate an image of a dog. Naturally, there are *many* possible images of dogs that we would be happy with. In particular, there is no one single "best" image of a dog. Rather, there is a spectrum of images that fit better or worse. In machine learning, it is common to realize this diversity of possible images as a *probability distribution* over the *space of images*. We call such a distribution a **data distribution** and denote it as p_{data}. Mathematically, one can think of p_{data} as a **probability density**, i.e. a function $p_{\text{data}} : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ that assigns each possible object $z \in \mathbb{R}^d$ a *likelihood* $p_{\text{data}}(z) \geq 0$. In the example of dog images, this distribution would therefore give higher likelihood $p_{\text{data}}(z)$ to images z that look more like a dog. Therefore, how "good" an image/video/molecule fits - a rather subjective statement - is replaced by how "likely" it is under the data distribution p_{data}. With this, we can mathematically express the task of generation as sampling from the (unknown) distribution p_{data}:

Key Idea 2 (Generation as Sampling)
Generating an object z is modeled as sampling from the data distribution $z \sim p_{\text{data}}$.

A **generative model** is a machine learning model that allows us to generate samples from p_{data}. In machine learning, we require data to train models. In generative modeling, we usually assume access to a finite number of examples sampled independently from p_{data}, which together serve as a proxy for the true distribution.

Key Idea 3 (Dataset)
A dataset consists of a finite number of samples $z_1, \dots, z_N \sim p_{\text{data}}$.

For images, we might construct a dataset by compiling publicly available images from the internet. For videos, we might similarly look to use YouTube. For protein structures, sources like the RCSB Protein Data Bank (PDB) provide hundreds of thousands of experimentally resolved structures. As the size of our dataset grows very large, it becomes an increasingly better representation of the underlying distribution p_{data}.

Guided/Conditional Generation. In many cases, we want to generate an object **conditioned** on some data y. For example, we might want to generate an image conditioned on $y =$"a dog running down a hill covered with snow with mountains in the background". We can rephrase this as sampling from a **conditional distribution**:

Key Idea 4 (Guided Generation)
Guided generation involves sampling from $z \sim p_{\text{data}}(\cdot|y)$, where y is a conditioning variable.

We call $p_{\text{data}}(\cdot|y)$ the **guided data distribution**. The guided generative modeling task typically involves learning to condition on an arbitrary, rather than fixed, choice of y. Using our previous example, we might alternatively want to condition on a different text prompt, such as $y =$"a photorealistic image of a cat blowing out birthday candles". We therefore seek a single model which may be conditioned on any such choice of y. It turns out that techniques for unconditional generation are readily generalized to the conditional case. Therefore, for the first 3 sections, we will focus almost exclusively on the unconditional case (keeping in mind that conditional generation is what we're building towards).

Generative Models. Abstractly speaking, a generative model is an algorithm that returns samples from $z \sim p_{\text{data}}$ (or at least approximately). If p_{data} is the distribution of images of dogs, this algorithm would return random images of dogs. In this course, we will focus on the specific construction of generative models using flow or diffusion models as these represent the current state-of-the-art. However, it is important to keep in mind that many other generative models were developed (and maybe even more that will be discovered in the future).

Summary 2 (Generation as Sampling)
We summarize the findings of this section:

1. In this class, we mainly consider the task of generating objects that are represented as vectors $z \in \mathbb{R}^d$ such as images, videos, and molecular structures.
2. Generation is the task of generating samples from a probability distribution p_{data} having access to a dataset of samples $z_1, \dots, z_N \sim p_{\text{data}}$ during training.
3. Guided generation assumes that we condition the distribution on a label y and we want to sample from $p_{\text{data}}(\cdot|y)$ having access to data set of pairs $(z_1, y) \dots, (z_N, y)$ during training.
4. Our goal is to construct a generative model, i.e. a model that returns samples from p_{data} after training.

2 Flow and Diffusion Models

In the previous section, we formalized generative modeling as sampling from a data distribution p_{data}. Further, we formalized our goal: To construct a generative model, i.e. an algorithm that returns samples $z \sim p_{\text{data}}$. In this section, we describe how a generative model can be built as the simulation of a suitably constructed differential equation. For example, flow matching and diffusion models involve simulating **ordinary differential equations** (ODEs) and **stochastic differential equations** (SDEs), respectively. The goal of this section is therefore to define and construct these generative models as they will be used throughout the remainder of the notes. Specifically, we first define ODEs and SDEs, and discuss their simulation. Second, we describe how to parameterize an ODE/SDE using a deep neural network. This leads to the definition of a flow and diffusion model and the fundamental algorithms to sample from such models. In later sections, we then explore how to train these models.

2.1 Flow Models

We start by defining **ordinary differential equations (ODEs)**. A solution to an ODE is defined by a **trajectory**, i.e. a function of the form

$$X : [0,1] \to \mathbb{R}^d, \quad t \mapsto X_t,$$

that maps from time t to some location in space $\mathbb{R}^d$. Every ODE is defined by a **vector field** u, i.e. a function of the form

$$u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d, \quad (x,t) \mapsto u_t(x),$$

i.e. for every time t and location x we get a vector $u_t(x) \in \mathbb{R}^d$ specifying a velocity in space (see Figure 1). An ODE imposes a condition on a trajectory: we want a trajectory X that "follows along the lines" of the vector field u_t, starting at the point x_0. We may formalize such a trajectory as being the solution to the equation:

$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t) \qquad \blacktriangleright \text{ ODE} \tag{1a}$$

$$X_0 = x_0 \qquad \blacktriangleright \text{ initial conditions} \tag{1b}$$

Equation (1a) requires that the derivative of X_t is specified by the direction given by u_t. Equation (1b) requires that we start at x_0 at time $t = 0$. We may now ask: if we start at $X_0 = x_0$ at $t = 0$, where are we at time t (what is X_t)? This question is answered by a function called the **flow**, which is a solution to the ODE

$$\psi : \mathbb{R}^d \times [0,1] \mapsto \mathbb{R}^d, \quad (x_0,t) \mapsto \psi_t(x_0) \tag{2a}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} \psi_t(x_0) = u_t(\psi_t(x_0)) \qquad \blacktriangleright \text{ flow ODE} \tag{2b}$$

$$\psi_0(x_0) = x_0 \qquad \blacktriangleright \text{ flow initial conditions} \tag{2c}$$

For a given initial condition $X_0 = x_0$, a trajectory of the ODE is recovered via $X_t = \psi_t(X_0)$. Therefore, vector fields, ODEs, and flows are, intuitively, three descriptions of the same object: **vector fields define ODEs whose solutions are flows**. As with every equation, we should ask ourselves about an ODE: Does a solution exist and if

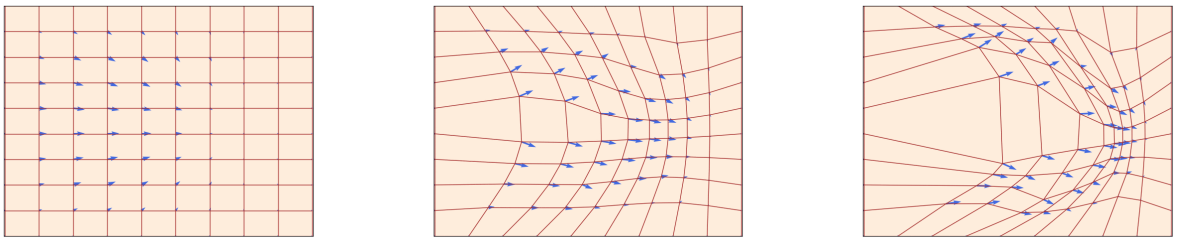

Figure 1: A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (red square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations (here, $d = 2$). We show three different times t. As one can see, a flow is a diffeomorphism that "warps" space. Figure from [6].

so, is it unique? A fundamental result in mathematics is "yes!" to both, as long we impose weak assumptions on u_t:

Theorem 3 (Flow existence and uniqueness)
If $u : \mathbb{R}^d \times [0, 1] \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative, then the ODE in (2) has a unique solution given by a flow ψ_t. In this case, ψ_t is a **diffeomorphism** for all t, i.e. ψ_t is continuously differentiable with a continuously differentiable inverse ψ_t^{-1}.

Note that the assumptions required for the existence and uniqueness of a flow are almost always fulfilled in machine learning, as we use neural networks to parameterize $u_t(x)$ and they always have bounded derivatives. Therefore, Theorem 3 should not be a concern for you but rather good news: **flows exist and are unique solutions to ODEs in our cases of interest.** A proof can be found in [9, 2].

Example 4 (Linear Vector Fields)
Let us consider a simple example of a vector field $u_t(x)$ that is a simple linear function in x, i.e. $u_t(x) = -\theta x$ for $\theta > 0$. Then the function

$$\psi_t(x_0) = \exp(-\theta t)\, x_0 \tag{3}$$

defines a flow ψ solving the ODE in Equation (2). You can check this yourself by checking that $\psi_0(x_0) = x_0$ and computing

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x_0) \overset{(3)}{=} \frac{\mathrm{d}}{\mathrm{d}t}\left(\exp(-\theta t)\, x_0\right) \overset{(i)}{=} -\theta \exp(-\theta t)\, x_0 \overset{(3)}{=} -\theta\psi_t(x_0) = u_t(\psi_t(x_0)),$$

where in (i) we used the chain rule. In Figure 3, we visualize a flow of this form converging to 0 exponentially.

Simulating an ODE. In general, it is not possible to compute the flow ψ_t explicitly if u_t is not as simple as in the previous example. In these cases, one uses **numerical methods** to simulate ODEs. Fortunately, this is a classical and well researched topic in numerical analysis, and a myriad of powerful methods exist [4]. One of the simplest

and most intuitive methods is the **Euler method**. In the Euler method, we initialize with $X_0 = x_0$ and update via

$$X_{t+h} = X_t + hu_t(X_t) \quad (t = 0, h, 2h, 3h, \dots, 1-h) \tag{4}$$

where $h = n^{-1} > 0$ is the **step size** and $n \in \mathbb{N}$ is the number of simulation steps. For this class, the Euler method will be good enough. To give you a taste of a more complex method, let us consider **Heun's method** defined via the update rule

$$X'_{t+h} = X_t + hu_t(X_t) \qquad \blacktriangleright \text{ initial guess of new state (same as Euler step)}$$
$$X_{t+h} = X_t + \frac{h}{2}(u_t(X_t) + u_{t+h}(X'_{t+h})) \qquad \blacktriangleright \text{ update with average } u \text{ at current and guessed state}$$

Intuitively, the Heun's method is as follows: it takes a first guess X'_{t+h} of what the next step could be but corrects the direction initially taken via an updated guess.

Flow models. We can now construct a generative model via an ODE by making the vector field a **neural network vector field** u_t^θ. For now, we simply mean that u_t^θ is a parameterized function $u_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ with parameters θ. Later, we will discuss particular choices of neural network architectures. Remember that our goal was to generate samples $z \sim p_{\text{data}}$ from a distribution p_{data}. In particular, these samples must be *random*. Note though that an ODE itself is not random but fully deterministic. To inject some randomness, we simple make the initial condition X_0 random. Specifically, we choose an **initial distribution** p_{init}. In most cases, we set $p_{\text{init}} = \mathcal{N}(0, I_d)$ to be a simple standard Gaussian. Most importantly, whatever distribution you choose, it must be one that we can easily sample from at inference-time. A **flow model** is then described by the ODE

$$X_0 \sim p_{\text{init}} \qquad \blacktriangleright \text{ random initialization}$$
$$\frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^\theta(X_t) \qquad \blacktriangleright \text{ ODE}$$

Our goal is to make the endpoint X_1 of the trajectory have distribution p_{data}, i.e.

$$X_1 \sim p_{\text{data}} \quad \Leftrightarrow \quad \psi_1^\theta(X_0) \sim p_{\text{data}}$$

where ψ_t^θ describes the flow induced by u_t^θ. Note however: although it is called *flow model*, **the neural network parameterizes the vector field, not the flow** (at least for now). In order to compute the flow, we need to simulate the ODE. In Algorithm 1, we summarize the procedure how to sample from a flow model.

Algorithm 1 Sampling from a Flow Model with Euler method

Require: Neural network vector field u_t^θ, number of steps n
1: Set $t = 0$
2: Set step size $h = \frac{1}{n}$
3: Draw a sample $X_0 \sim p_{\text{init}}$
4: **for** $i = 1, \dots, n$ **do**
5: $\quad X_{t+h} = X_t + hu_t^\theta(X_t)$
6: $\quad$ Update $t \leftarrow t + h$
7: **end for**
8: **return** X_1

2.2 Diffusion Models

Stochastic differential equations (SDEs) extend the deterministic trajectories from ODEs with **stochastic** trajectories. A stochastic trajectory is commonly called a **stochastic process** $(X_t)_{0\leq t\leq 1}$ and is given by

$$\begin{aligned} X_t \text{ is a random variable for every } 0 \leq t \leq 1 \\ X : [0,1] \to \mathbb{R}^d, \quad t \mapsto X_t \text{ is a random trajectory for every draw of } X \end{aligned}$$

In particular, when we simulate the same stochastic process twice, we might get different outcomes because the dynamics are designed to be random.

Brownian Motion. SDEs are constructed via a **Brownian motion** - a fundamental stochastic process that came out of the study physical diffusion processes. You can think of a Brownian motion as a continuous random walk. Let us define it: A **Brownian motion** $W = (W_t)_{0\leq t\leq 1}$ is a stochastic process such that $W_0 = 0$, the trajectories $t \mapsto W_t$ are continuous, and the following two conditions hold:

1. **Normal increments:** $W_t - W_s \sim \mathcal{N}(0, (t-s)I_d)$ for all $0 \leq s < t$, i.e. increments have a Gaussian distribution with variance increasing linearly in time (I_d is the identity matrix).
2. **Independent increments:** For any $0 \leq t_0 < t_1 < \cdots < t_n = 1$, the increments $W_{t_1} - W_{t_0}, \ldots, W_{t_n} - W_{t_{n-1}}$ are independent random variables.

Brownian motion is also called a **Wiener process**, which is why we denote it with a "W".[1] We can easily simulate a Brownian motion approximately with step size $h > 0$ by setting $W_0 = 0$ and updating

$$W_{t+h} = W_t + \sqrt{h}\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I_d) \quad (t = 0, h, 2h, \ldots, 1-h) \quad (5)$$

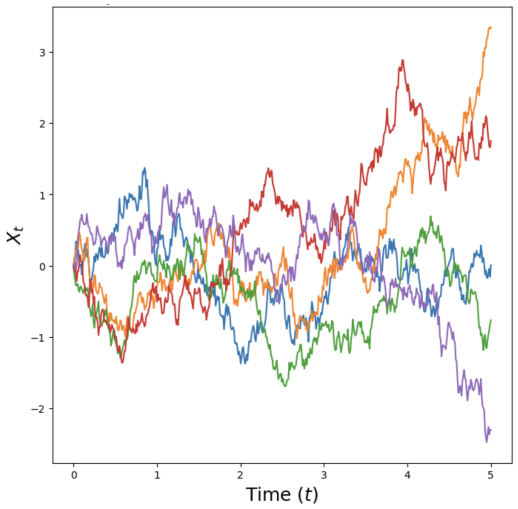

Figure 2: Sample trajectories of a Brownian motion W_t in dimension $d = 1$ simulated using Equation (5).

In Figure 2, we plot a few example trajectories of a Brownian motion. Brownian motion is as central to the study of stochastic processes as the Gaussian distribution is to the study of probability distributions. From finance to statistical physics to epidemiology, the study of Brownian motion has far reaching applications beyond machine learning. In finance, for example, Brownian motion is used to model the price of complex financial instruments. Also just as a mathematical construction, Brownian motion is fascinating: For example, while the paths of a Brownian motion are continuous (so that you could draw it without ever lifting a pen), they are infinitely long (so that you would never stop drawing).

From ODEs to SDEs. The idea of an SDE is to extend the deterministic dynamics of an ODE by adding stochastic dynamics driven by a Brownian motion. Because everything is stochastic, we may no longer take the derivative as

[1] Nobert Wiener was a famous mathematician who taught at MIT. You can still see his portraits hanging at the MIT math department.

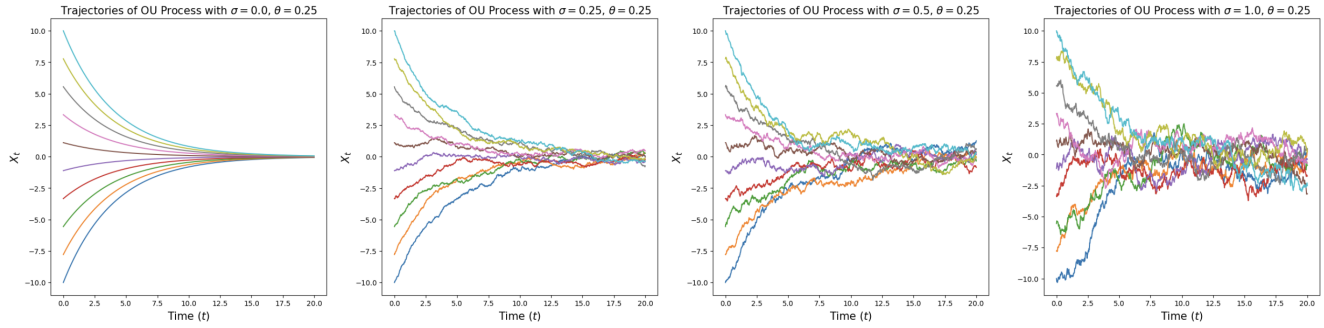

Figure 3: Illustration of Ornstein-Uhlenbeck processes (Equation (8)) in dimension $d = 1$ for $\theta = 0.25$ and various choices of σ (increasing from left to right). For $\sigma = 0$, we recover a flow (smooth, deterministic trajectories) that converges to the origin as $t \to \infty$. For $\sigma > 0$ we have random paths which converge towards the Gaussian $\mathcal{N}(0, \frac{\sigma^2}{2\theta})$ as $t \to \infty$.

in Equation (1a). Hence, we need to find an **equivalent formulation of ODEs that does not use derivatives**. For this, let us therefore rewrite trajectories $(X_t)_{0 \le t \le 1}$ of an ODE as follows:

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t(X_t) && \blacktriangleright \text{ expression via derivatives} \\ \overset{(i)}{\Leftrightarrow} \quad & \frac{1}{h}\left(X_{t+h} - X_t\right) = u_t(X_t) + R_t(h) \\ \Leftrightarrow \quad & X_{t+h} = X_t + h u_t(X_t) + h R_t(h) && \blacktriangleright \text{ expression via infinitesimal updates} \end{aligned}$$

where $R_t(h)$ describes a negligible function for small h, i.e. such that $\lim_{h \to 0} R_t(h) = 0$, and in (i) we simply use the definition of derivatives. The derivation above simply restates what we already know: A trajectory $(X_t)_{0 \le t \le 1}$ of an ODE takes, at every timestep, a small step in the direction $u_t(X_t)$. We may now amend the last equation to make it stochastic: A trajectory $(X_t)_{0 \le t \le 1}$ of an SDE takes, at every timestep, a small step in the direction $u_t(X_t)$ *plus* some contribution from a Brownian motion:

$$X_{t+h} = X_t + \underbrace{h u_t(X_t)}_{\text{deterministic}} + \sigma_t \underbrace{(W_{t+h} - W_t)}_{\text{stochastic}} + \underbrace{h R_t(h)}_{\text{error term}} \tag{6}$$

where $\sigma_t \ge 0$ describes the **diffusion coefficient** and $R_t(h)$ describes a stochastic error term such that the standard deviation $\mathbb{E}[\|R_t(h)\|^2]^{1/2} \to 0$ goes to zero for $h \to 0$. The above describes a **stochastic differential equation (SDE)**. It is common to denote it in the following symbolic notation:

$$\mathrm{d}X_t = u_t(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t \qquad \blacktriangleright \text{ SDE} \tag{7a}$$

$$X_0 = x_0 \qquad \blacktriangleright \text{ initial condition} \tag{7b}$$

However, always keep in mind that the "$\mathrm{d}X_t$"-notation above is a purely informal notation of Equation (6). Unfortunately, SDEs do not have a flow map ϕ_t anymore. This is because the value X_t is not fully determined by $X_0 \sim p_{\text{init}}$ anymore as the evolution itself is stochastic. Still, in the same way as for ODEs, we have:

Theorem 5 (SDE Solution Existence and Uniqueness)
If $u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ is continuously differentiable with a bounded derivative and σ_t is continuous, then the SDE in (7) has a solution given by the unique stochastic process $(X_t)_{0 \leq t \leq 1}$ satisfying Equation (6).

If this was a stochastic calculus class, we would spend several lectures proving this theorem and constructing SDEs with full mathematical rigor, i.e. constructing a Brownian motion from first principles and constructing the process X_t via **stochastic integration**. As we focus on machine learning in this class, we refer to [8] for a more technical treatment. Finally, note that every ODE is also an SDE - simply with a vanishing diffusion coefficient $\sigma_t = 0$. Therefore, for the remainder of this class, **when we speak about SDEs, we consider ODEs as a special case**.

Example 6 (Ornstein-Uhlenbeck Process)
Let us consider a constant diffusion coefficient $\sigma_t = \sigma \geq 0$ and a constant linear drift $u_t(x) = -\theta x$ for $\theta > 0$, yielding the SDE

$$\mathrm{d}X_t = -\theta X_t \mathrm{d}t + \sigma dW_t. \tag{8}$$

A solution $(X_t)_{0 \leq t \leq 1}$ to the above SDE is known as an **Ornstein-Uhlenbeck (OU) process**. We visualize it in Figure 3. The vector field $-\theta x$ pushes the process back to its center 0 (as I always go the inverse direction of where I am), while the diffusion coefficient σ always adds more noise. This process converges towards a Gaussian distribution $\mathcal{N}(0, \sigma^2/(2\theta))$ if we simulate it for $t \to \infty$. Note that for $\sigma = 0$, we have a flow with linear vector field that we have studied in Equation (3).

Simulating an SDE. If you struggle with the abstract definition of an SDE so far, then don't worry about it. A more intuitive way of thinking about SDEs is given by answering the question: How might we simulate an SDE? The simplest such scheme is known as the **Euler-Maruyama method**, and is essentially to SDEs what the Euler method is to ODEs. Using the Euler-Maruyama method, we initialize $X_0 = x_0$ and update iteratively via

$$X_{t+h} = X_t + h u_t(X_t) + \sqrt{h} \sigma_t \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(0, I_d) \tag{9}$$

where $h = n^{-1} > 0$ is a step size hyperparameter for $n \in \mathbb{N}$. In other words, to simulate using the Euler-Maruyama method, we take a small step in the direction of $u_t(X_t)$ as well as add a little bit of Gaussian noise scaled by $\sqrt{h}\sigma_t$. When simulating SDEs in this class (such as in the accompanying labs), we will usually stick to the Euler-Maruyama method.

Diffusion Models. We can now construct a generative model via an SDE in the same way as we did for ODEs. Remember that our goal was to convert a simple distribution p_{init} into a complex distribution p_{data}. Like for ODEs, the simulation of an SDE randomly initialized with $X_0 \sim p_{\text{init}}$ is a natural choice for this transformation. To parameterize this SDE, we can simply parameterize its central ingredient - the vector field u_t - a neural network

Algorithm 2 Sampling from a Diffusion Model (Euler-Maruyama method)

Require: Neural network u_t^θ, number of steps n, diffusion coefficient σ_t
Set $t = 0$
Set step size $h = \frac{1}{n}$
Draw a sample $X_0 \sim p_{\text{init}}$
for $i = 1, \dots, n$ **do**
  Draw a sample $\epsilon \sim \mathcal{N}(0, I_d)$
  $X_{t+h} = X_t + h u_t^\theta(X_t) + \sigma_t \sqrt{h} \epsilon$
  Update $t \leftarrow t + h$
end for
return X_1

u_t^θ. A **diffusion model** is thus given by

$$X_0 \sim p_{\text{init}} \quad \blacktriangleright \text{ random initialization}$$
$$\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t \quad \blacktriangleright \text{ SDE}$$

In Algorithm 2, we describe the procedure by which to sample from a diffusion model with the Euler-Maruyama method. We summarize the results of this section as follows.

Summary 7 (SDE generative model)
Throughout this document, a **diffusion model** consists of a neural network u_t^θ with parameters θ that parameterize a vector field and a fixed diffusion coefficient σ_t:

$$\textbf{Neural network: } u^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d,\ (x,t) \mapsto u_t^\theta(x) \text{ with parameters } \theta$$
$$\textbf{Fixed: } \sigma_t : [0,1] \to [0,\infty),\ t \mapsto \sigma_t$$

To obtain samples from our SDE model (i.e. generate objects), the procedure is as follows:

Initialization: $X_0 \sim p_{\text{init}}$ ▶ Initialize with simple distribution, e.g. a Gaussian
Simulation: $\mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t$ ▶ Simulate SDE from 0 to 1
Goal: $X_1 \sim p_{\text{data}}$ ▶ Goal is to make X_1 have distribution p_{data}

A diffusion model with $\sigma_t = 0$ is a **flow model**.

3 Flow Matching

In the previous section, we constructed flow and diffusion models as generative models parameterized by a neural network vector field u_t^θ. However, we have not yet discussed how to train them. i.e. how to optimize the parameters θ such that generative model returns something sensible, e.g. a nice-looking image or exciting video. Next, we discuss **flow matching** [5, 1, 7], a algorithm to train u_t^θ that is simple, scalable, and represents the current state-of-the-art.

In this section, we restrict ourselves to flow models, i.e. we have a neural network u_t^θ and obtain samples from the generative model by simulating the ODE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^\theta(X_t)\mathrm{d}t \qquad \text{(Flow model)} \tag{10}$$

and using the endpoints X_1 fro $t = 1$ as samples. As we discussed, our goal is that X_1 is distributed according to the data distribution p_{data}, i.e. $X_1 \sim p_{\text{data}}$. Therefore, the question "how to train" the neural network is really the following question: **How do we optimize θ such that simulating the flow model in Equation (10) results in samples from the data distribution $X_1 \sim p_{\text{data}}$?**

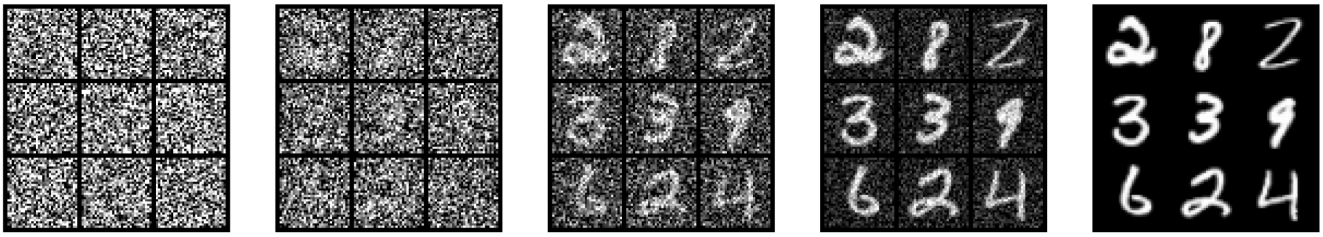

Figure 4: Gradual interpolation from noise to data via a Gaussian conditional probability path for a collection of images. Note that each image is a data point of dimension $d = 32 \times 32$, so we are plotting individual samples from the probability path, while in Figure 5 we plot the distribution as a 2d histogram.

3.1 Conditional and Marginal Probability Path

The first step of flow matching is to specify a **probability path**. Intuitively, a probability path specifies a gradual interpolation between noise p_{init} and data p_{data} (see Figure 4). But why would we want that? Remember that our desired ODE trajectory fulfills $X_0 \sim p_{\text{init}}$ for $t = 0$ and $X_1 \sim p_{\text{data}}$ for $t = 1$. But what about times $0 < t < 1$ in between start and end? It turns out that we have some freedom to choose what should happen in between and this is what is mathematically formalized in a probability path.

In the following, for a data point $z \in \mathbb{R}^d$, we denote with δ_z the **Dirac delta** "distribution". This is the simplest distribution that one can imagine: sampling from δ_z always returns z (i.e. it is deterministic). A **conditional (interpolating) probability path** is a set of distribution $p_t(x|z)$ over $\mathbb{R}^d$ such that:

$$p_0(\cdot|z) = p_{\text{init}}, \quad p_1(\cdot|z) = \delta_z \quad \text{for all } z \in \mathbb{R}^d. \tag{11}$$

In other words, a conditional probability path gradually converts the initial distribution p_{init} into a *single* data point (see e.g. Figure 4). You can think of a probability path as a trajectory in the space of distributions.

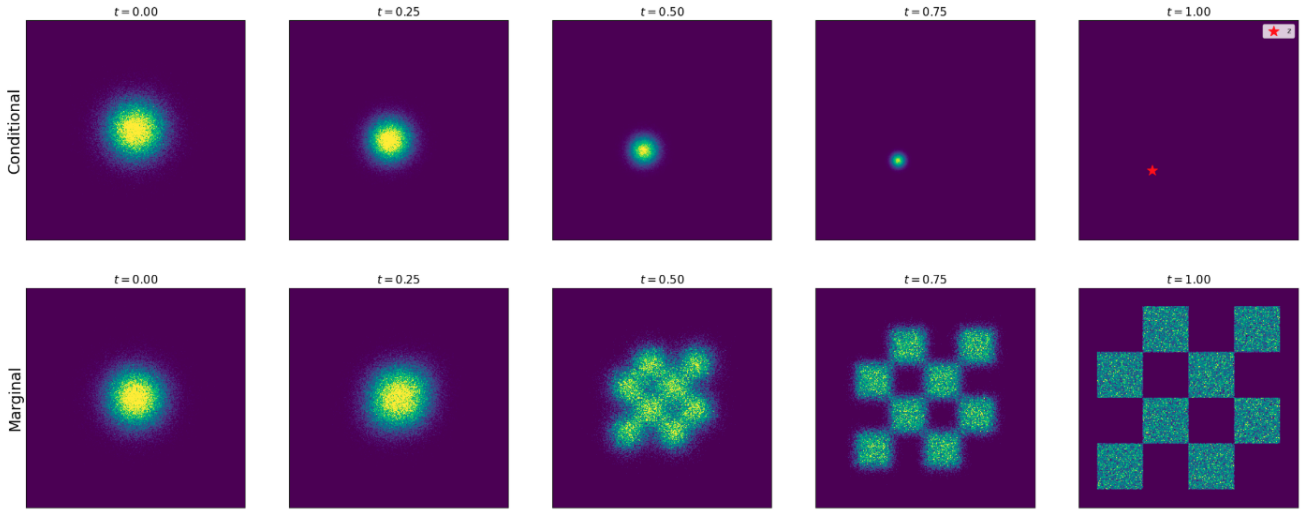

Figure 5: Illustration of a conditional (top) and marginal (bottom) probability path. Here, we plot a Gaussian probability path with $\alpha_t = t, \beta_t = 1 - t$. The conditional probability path interpolates a Gaussian $p_{\text{init}} = \mathcal{N}(0, I_d)$ and $p_{\text{data}} = \delta_z$ for single data point z. The marginal probability path interpolates a Gaussian and a data distribution p_{data} (Here, p_{data} is a toy distribution in dimension $d = 2$ represented by a chess board pattern.)

Every conditional probability path $p_t(x|z)$ induces a **marginal probability path** $p_t(x)$ defined as the distribution that we obtain by first sampling a data point $z \sim p_{\text{data}}$ from the data distribution and then sampling from $p_t(\cdot|z)$:

$$z \sim p_{\text{data}}, \quad x \sim p_t(\cdot|z) \quad \Rightarrow x \sim p_t \qquad \blacktriangleright \text{ sampling from marginal path} \tag{12}$$

$$p_t(x) = \int p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \qquad \blacktriangleright \text{ density of marginal path} \tag{13}$$

Note that we know how to sample from p_t but we don't know the density values $p_t(x)$ as the integral is intractable (i.e. we can actually compute Equation (12) but not Equation (13)). Check for yourself that because of the conditions on $p_t(\cdot|z)$ in Equation (11), the marginal probability path p_t interpolates between p_{init} and p_{data}:

$$p_0 = p_{\text{init}} \quad \text{and} \quad p_1 = p_{\text{data}}. \qquad \blacktriangleright \text{ noise-data interpolation} \tag{14}$$

The - by far - most important example of a probability path is the Gaussian probability path - hence, we strongly recommend reading the next example thoroughly.

Example 8 (Gaussian Conditional Probability Path)
One particularly popular probability path is the **Gaussian probability path**. This is the **probability path used by denoising diffusion models**. Let α_t, β_t be **noise schedulers**: two continuously differentiable, monotonic functions with $\alpha_0 = \beta_1 = 0$ and $\alpha_1 = \beta_0 = 1$. We then define the conditional probability path

$$p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d) \qquad \blacktriangleright \text{ Gaussian conditional path} \tag{15}$$

which, by the conditions we imposed on α_t and β_t, fulfills

$$p_0(\cdot|z) = \mathcal{N}(\alpha_0 z, \beta_0^2 I_d) = \mathcal{N}(0, I_d), \quad \text{and} \quad p_1(\cdot|z) = \mathcal{N}(\alpha_1 z, \beta_1^2 I_d) = \delta_z,$$

where we have used the fact that a normal distribution with zero variance and mean z is just δ_z. Therefore, this choice of $p_t(x|z)$ fulfills Equation (11) for $p_{\text{init}} = \mathcal{N}(0, I_d)$ and is therefore a valid conditional interpolating path. In Figure 4, we illustrate its application to an image. We can express sampling from the marginal path p_t as:

$$z \sim p_{\text{data}},\ \epsilon \sim p_{\text{init}} = \mathcal{N}(0, I_d) \Rightarrow x = \alpha_t z + \beta_t \epsilon \sim p_t \qquad \blacktriangleright \text{ sampling from marginal Gaussian path} \tag{16}$$

Intuitively, the above procedure adds more noise for lower t until time $t = 0$, at which point there is only noise. In Figure 5, we plot an example of such an interpolating path.

3.2 Conditional and Marginal Vector Fields

A probability path $(p_t)_{0 \le t \le 1}$ specified what distributions $X_t \sim p_t$ the points X_t along a trajectory *should* have. At this point, this is just we "wish" to be the case. But how can we find a vector field such that the trajectories X_t follow the probability path? Flow matching explicitly constructs such a vector field - the "marginal vector field" - which we explain in this section.

For every data point $z \in \mathbb{R}^d$, let $u_t^{\text{target}}(\cdot|z)$ denote a **conditional vector field**. This can be any vector field such that corresponding ODE yields the conditional probability path $p_t(\cdot|z)$, i.e. such that it holds

$$X_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^{\text{target}}(X_t|z) \quad \Rightarrow \quad X_t \sim p_t(\cdot|z) \quad (0 \le t \le 1). \tag{17}$$

We can often find a conditional vector field $u_t^{\text{target}}(\cdot|z)$ analytically by hand (i.e. by just doing some algebra ourselves). We illustrate this by deriving a conditional vector field $u_t(x|z)$ for our running example of a Gaussian probability path in Example 10.

At first sight, a conditional vector field seems useless because all endpoints of the ODE X_1 will collapse to $X_1 = z$, i.e. we are just re-generating known data points z. However, the conditional vector field serves as a building block for a vector field that generates actual samples from p_{data}:

Theorem 9 (Marginalization trick)
Let $u_t^{\text{target}}(x|z)$ be a conditional vector field (Equation (17)). Then the **marginal vector field** $u_t^{\text{target}}(x)$ defined as

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z, \tag{18}$$

follows the marginal probability path, i.e.

$$X_0 \sim p_{\text{init}}, \quad \frac{\mathrm{d}}{\mathrm{d}t} X_t = u_t^{\text{target}}(X_t) \quad \Rightarrow \quad X_t \sim p_t \quad (0 \le t \le 1). \tag{19}$$

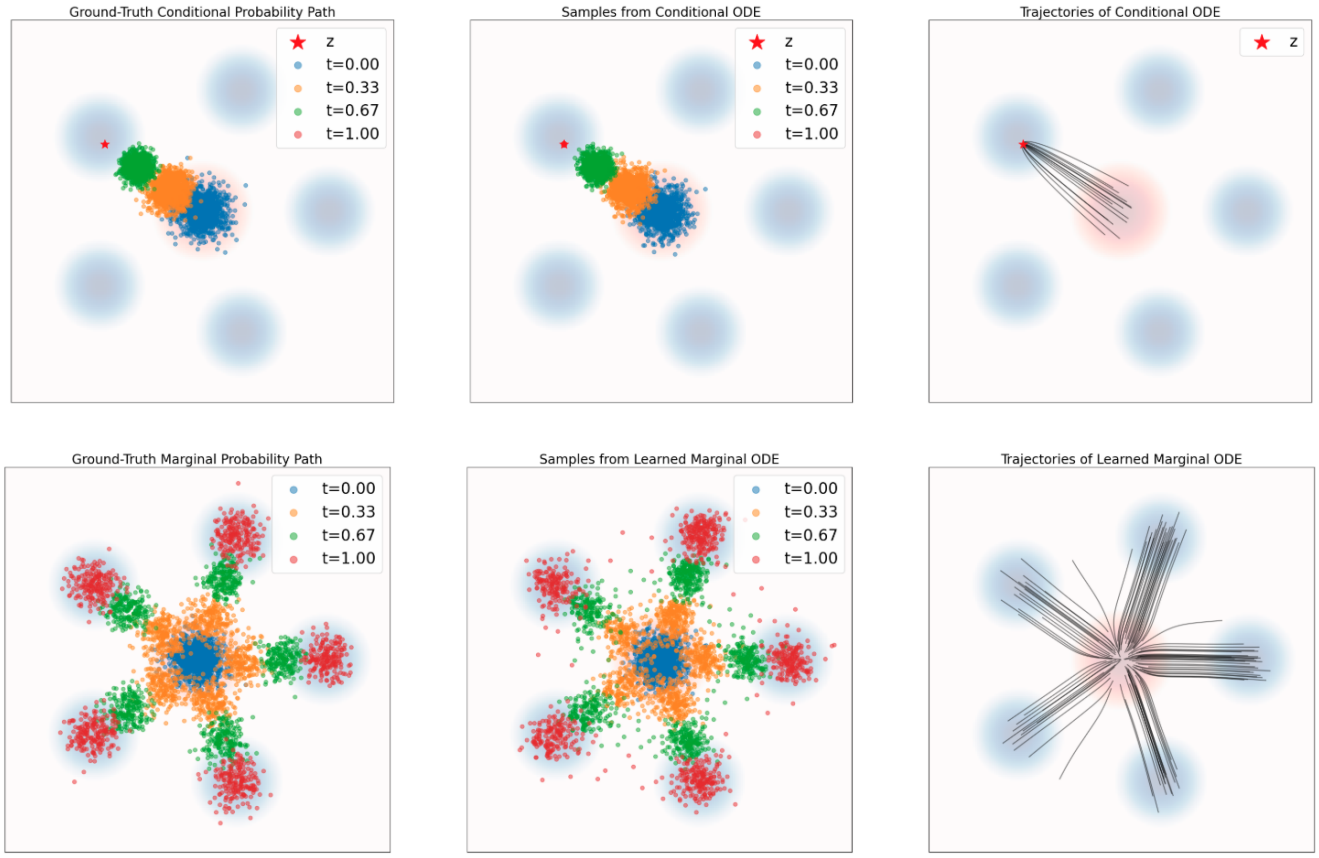

Figure 6: Illustration of Theorem 9. Simulating a probability path with ODEs. Data distribution p_{data} in blue background. Gaussian p_{init} in red background. Top row: Conditional probability path. Left: Ground truth samples from conditional path $p_t(\cdot|z)$. Middle: ODE samples over time. Right: Trajectories by simulating ODE with $u_t^{\text{target}}(x|z)$ in Equation (20). Bottom row: Simulating a marginal probability path. Left: Ground truth samples from p_t. Middle: ODE samples over time. Right: Trajectories by simulating ODE with marginal vector field $u_t^{\text{flow}}(x)$. As one can see, the conditional vector field follows the conditional probability path and the marginal vector field follows the marginal probability path.

In particular, $X_1 \sim p_{\text{data}}$ for this ODE, so that we might say "u_t^{target} converts noise p_{init} into data p_{data}".

Example 10 (Target ODE for Gaussian probability paths)
As before, let $p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$ for noise schedulers α_t, β_t (see Equation (15)). Let $\dot{\alpha}_t = \partial_t \alpha_t$ and $\dot{\beta}_t = \partial_t \beta_t$ denote respective time derivatives of α_t and β_t. Here, we want to show that the **conditional Gaussian vector field** given by

$$u_t^{\text{target}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x \tag{20}$$

is a valid conditional vector field model in the sense of Theorem 9: its ODE trajectories X_t satisfy $X_t \sim p_t(\cdot|z) = \mathcal{N}(\alpha_t z, \beta_t^2 I_d)$ if $X_0 \sim \mathcal{N}(0, I_d)$. In Figure 6, we confirm this visually by comparing samples from the conditional probability path (ground truth) to samples from simulated ODE trajectories of this flow. As you can see, the distribution match. We will now prove this.

Proof. Let us construct a conditional flow model $\psi_t^{\text{target}}(x|z)$ first by defining

$$\psi_t^{\text{target}}(x|z) = \alpha_t z + \beta_t x. \tag{21}$$

If X_t is the ODE trajectory of $\psi_t^{\text{target}}(\cdot|z)$ with $X_0 \sim p_{\text{init}} = \mathcal{N}(0, I_d)$, then by definition

$$X_t = \psi_t^{\text{target}}(X_0|z) = \alpha_t z + \beta_t X_0 \sim \mathcal{N}(\alpha_t z, \beta^2 I_d) = p_t(\cdot|z).$$

We conclude that the trajectories are distributed like the conditional probability path (i.e, Equation (17) is fulfilled). It remains to extract the vector field $u_t^{\text{target}}(x|z)$ from $\psi_t^{\text{target}}(x|z)$. By the definition of a flow (Equation (2b)), it holds

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\psi_t^{\text{target}}(x|z) = u_t^{\text{target}}(\psi_t^{\text{target}}(x|z)|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(i)}{\Leftrightarrow} \quad &\dot{\alpha}_t z + \dot{\beta}_t x = u_t^{\text{target}}(\alpha_t z + \beta_t x|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(ii)}{\Leftrightarrow} \quad &\dot{\alpha}_t z + \dot{\beta}_t \left(\frac{x - \alpha_t z}{\beta_t}\right) = u_t^{\text{target}}(x|z) \quad \text{for all } x, z \in \mathbb{R}^d \\
\overset{(iii)}{\Leftrightarrow} \quad &\left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x = u_t^{\text{target}}(x|z) \quad \text{for all } x, z \in \mathbb{R}^d
\end{aligned}$$

where in (i) we used the definition of $\psi_t^{\text{target}}(x|z)$ (Equation (21)), in (ii) we reparameterized $x \to (x - \alpha_t z)/\beta_t$, and in (iii) we just did some algebra. Note that the last equation is the conditional Gaussian vector field as we defined in Equation (20). This proves the statement.[a] □

[a] One can also double check this by plugging it into the continuity equation introduced later in this section.

See Figure 6 for an illustration of Theorem 9. Let's gain some intuition for the marginal vector field. **Bayes' rule** from statistics says that the following term describes a posterior distribution

$$\frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)} = \text{"posterior over data points } z \text{ given noisy data } x\text{"}$$

where $p_{\text{data}}(z)$ is the prior distribution. The marginal vector field then is simply a average: for every *possible* data point z it takes the velocity $u_t(x|z)$ - i.e. the direction that would bring us to z - and then weighs this velocity by how much we believe that x comes from z. Averaging over all data points, we obtain the marginal vector field.

The remainder of this section will make this intuition rigorous and prove Theorem 9. As the main mathematical tool, we will use the **continuity equation**, a fundamental equation in mathematics and physics. Define the **divergence** operator div as

$$\text{div}(v_t)(x) = \sum_{i=1}^{d} \frac{\partial}{\partial x_i} v_t^i(x) \tag{22}$$

where v_t^i is the i-th coordinate of v_t.

Theorem 11 (Continuity Equation)
Let us consider an flow model with vector field u_t^{target} with $X_0 \sim p_{\text{init}}$. Then $X_t \sim p_t$ for all $0 \leq t \leq 1$ if and only if

$$\partial_t p_t(x) = -\text{div}(p_t u_t^{\text{target}})(x) \quad \text{for all } x \in \mathbb{R}^d, 0 \leq t \leq 1, \tag{23}$$

where $\partial_t p_t(x) = \frac{\mathrm{d}}{\mathrm{d}t} p_t(x)$ denotes the time-derivative of $p_t(x)$. Equation 23 is known as the **continuity equation**.

For the mathematically-inclined reader, we present a self-contained proof of the Continuity Equation in Section B. Before we move on, let us try and understand intuitively the continuity equation. The left-hand side $\partial_t p_t(x)$ describes how much the probability $p_t(x)$ at x changes over time. Intuitively, the change should correspond to the net inflow of probability mass. For a flow model, a particle X_t follows along the vector field u_t^{target}. As you might recall from physics, the divergence measures a sort of net outflow from the vector field. Therefore, the negative divergence measures the net inflow. Scaling this by the total probability mass currently residing at x, we get that the net $-\text{div}(p_t u_t)$ measures the total inflow of probability mass. Since probability mass is conserved (always integrates to 1), the left-hand and right-hand side of the equation should be the same! We now proceed with a proof of the marginalization trick from Theorem 9.

Proof of Theorem 9. By Theorem 11, we have to show that the marginal vector field u_t^{target}, as defined as in Equation (18), satisfies the continuity equation. We can do this by direct calculation:

$$\begin{aligned}
\partial_t p_t(x) &\overset{(i)}{=} \partial_t \int p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \\
&= \int \partial_t p_t(x|z) p_{\text{data}}(z) \mathrm{d}z \\
&\overset{(ii)}{=} \int -\text{div}(p_t(\cdot|z) u_t^{\text{target}}(\cdot|z))(x) p_{\text{data}}(z) \mathrm{d}z \\
&\overset{(iii)}{=} -\text{div}\left(\int p_t(x|z) u_t^{\text{target}}(x|z) p_{\text{data}}(z) \mathrm{d}z \right) \\
&\overset{(iv)}{=} -\text{div}\left(p_t(x) \int u_t^{\text{target}}(x|z) \frac{p_t(x|z) p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z \right)(x) \\
&\overset{(v)}{=} -\text{div}\left(p_t u_t^{\text{target}} \right)(x),
\end{aligned}$$

where in (i) we used the definition of $p_t(x)$ in Equation (12), in (ii) we used the continuity equation for the conditional probability path $p_t(\cdot|z)$, in (iii) we swapped the integral and divergence operator using Equation (22), in (iv) we multiplied and divided by $p_t(x)$, and in (v) we used Equation (18). The beginning and end of the above chain of equations show that the continuity equation is fulfilled for u_t^{target}. By Theorem 11, this is enough to imply Equation (19), and we are done. □

3.3 Learning the Marginal Vector Field

Now, we are ready to describe the training algorithm. The goal of flow matching is to train the neural network u_t^θ such that it equals the marginal vector field u_t^{target}. If this holds, we know that the endpoints $X_1 \sim p_{\text{data}}$ have the desired distribution by Theorem 9. In the following, we denote by $\text{Unif} = \text{Unif}_{[0,1]}$ the uniform distribution on the interval $[0,1]$, and by $\mathbb{E}$ the expected value of a random variable. An intuitive way of obtaining $u_t^\theta \approx u_t^{\text{target}}$ is to use a mean-squared error, i.e. to use the **flow matching loss** defined as

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},x\sim p_t}[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2] \tag{24}$$

$$\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2], \tag{25}$$

where $p_t(x) = \int p_t(x|z)p_{\text{data}}(z)\text{d}z$ is the marginal probability path and in (i) we used the sampling procedure given by Equation (12). Intuitively, this loss says: First, draw a random time $t \in [0,1]$. Second, draw a random point z from our data set, sample from $p_t(\cdot|z)$ (e.g., by adding some noise), and compute $u_t^\theta(x)$. Finally, compute the mean-squared error between the output of our neural network and the marginal vector field $u_t^{\text{target}}(x)$. Unfortunately, we are *not* done here. While we do know the formula for u_t^{target} by Theorem 9, we cannot compute it efficiently. Instead, we will exploit the fact that the **conditional** velocity field $u_t^{\text{target}}(x|z)$ is tractable. To do so, let us define the **conditional flow matching loss**

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2]. \tag{26}$$

Note the difference to Equation (24): we use the conditional vector field $u_t^{\text{target}}(x|z)$ instead of the marginal vector $u_t^{\text{target}}(x)$. As we have an analytical formula for $u_t^{\text{target}}(x|z)$, we can minimize the above loss easily. But wait, what sense does it make to regress against the conditional vector field if it's the marginal vector field we care about? As it turns out, by *explicitly* regressing against the tractable, conditional vector field, we are *implicitly* regressing against the intractable, marginal vector field. The next result makes this intuition precise.

Theorem 12

The marginal flow matching loss equals the conditional flow matching loss up to a constant. That is,

$$\mathcal{L}_{\text{FM}}(\theta) = \mathcal{L}_{\text{CFM}}(\theta) + C,$$

where C is independent of θ. Therefore, their gradients coincide:

$$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta).$$

Hence, minimizing $\mathcal{L}_{\text{CFM}}(\theta)$ with e.g., stochastic gradient descent (SGD) is equivalent to minimizing $\mathcal{L}_{\text{FM}}(\theta)$ with in the same fashion. In particular, **for the minimizer θ^* of $\mathcal{L}_{\text{CFM}}(\theta)$, it will hold that $u_t^{\theta^*} = u_t^{\text{target}}$, i.e. the neural network will equal the marginal vector field** (assuming an infinitely expressive parameterization).

Proof. The proof works by expanding the mean-squared error into three components and removing constants:

$$\begin{aligned}
\mathcal{L}_{\text{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},x\sim p_t}[\|u_t^\theta(x) - u_t^{\text{target}}(x)\|^2] \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\text{Unif},x\sim p_t}[\|u_t^\theta(x)\|^2 - 2u_t^\theta(x)^T u_t^{\text{target}}(x) + \|u_t^{\text{target}}(x)\|^2] \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\text{Unif},x\sim p_t}\left[\|u_t^\theta(x)\|^2\right] - 2\mathbb{E}_{t\sim\text{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\text{target}}(x)] + \underbrace{\mathbb{E}_{t\sim\text{Unif}_{[0,1]},x\sim p_t}[\|u_t^{\text{target}}(x)\|^2]}_{=:C_1} \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2] - 2\mathbb{E}_{t\sim\text{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\text{target}}(x)] + C_1
\end{aligned}$$

where (i) holds by definition, in (ii) we used the formula $\|a-b\|^2 = \|a\|^2 - 2a^Tb + \|b\|^2$, in (iii) we define a constant C_1 and in (iv) we used the sampling procedure of p_t given by Equation (12). Let us reexpress the second summand:

$$\begin{aligned}
\mathbb{E}_{t\sim\text{Unif},x\sim p_t}[u_t^\theta(x)^T u_t^{\text{target}}(x)] &\overset{(i)}{=} \int_0^1 \int p_t(x) u_t^\theta(x)^T u_t^{\text{target}}(x)\,\mathrm{d}x\,\mathrm{d}t \\
&\overset{(ii)}{=} \int_0^1 \int p_t(x) u_t^\theta(x)^T \left[\int u_t^{\text{target}}(x|z)\frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)}\mathrm{d}z\right]\mathrm{d}x\,\mathrm{d}t \\
&\overset{(iii)}{=} \int_0^1 \int\int\int u_t^\theta(x)^T u_t^{\text{target}}(x|z) p_t(x|z) p_{\text{data}}(z)\,\mathrm{d}z\,\mathrm{d}x\,\mathrm{d}t \\
&\overset{(iv)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[u_t^\theta(x)^T u_t^{\text{target}}(x|z)]
\end{aligned}$$

where in (i) we expressed the expected value as an integral, in (ii) we use **??**, in (iii) we use the fact that integrals are linear, in (iv) we express the integral as an expected value. Note that this was really the crucial step of the proof: The beginning of the equality used the marginal vector field $u_t^{\text{target}}(x)$, while the end uses the conditional vector field $u_t^{\text{target}}(x|z)$. We plug is into the equation for $\mathcal{L}_{\text{FM}}$ to get:

$$\begin{aligned}
\mathcal{L}_{\text{FM}}(\theta) &\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2] - 2\mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[u_t^\theta(x)^T u_t^{\text{target}}(x|z)] + C_1 \\
&\overset{(ii)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x)\|^2 - 2u_t^\theta(x)^T u_t^{\text{target}}(x|z) + \|u_t^{\text{target}}(x|z)\|^2 - \|u_t^{\text{target}}(x|z)\|^2] + C_1 \\
&\overset{(iii)}{=} \mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2] + \underbrace{\mathbb{E}_{t\sim\text{Unif},z\sim p_{\text{data}},x\sim p_t(\cdot|z)}[-\|u_t^{\text{target}}(x|z)\|^2]}_{C_2} + C_1 \\
&\overset{(iv)}{=} \mathcal{L}_{\text{CFM}}(\theta) + \underbrace{C_2 + C_1}_{=:C}
\end{aligned}$$

where in (i) we plugged in the derived equation, in (ii) we added and subtracted the same value, in (iii) we used the formula $\|a-b\|^2 = \|a\|^2 - 2a^Tb + \|b\|^2$ again, and in (iv) we defined a constant in θ. This finishes the proof. □

Therefore, **flow matching training consists of minimizing the conditional flow matching loss.** The training procedure is summarized in Algorithm 3 and visualized in Figure 7. Note that there are several striking features about this algorithm: First, we never actually simulate any ODE during training. People call this feature of the algorithm **simulation-free**. This makes training extremely cheap as you don't have to roll out trajectories of the ODE during training (which takes a lot of steps). Second, the training is a simple regression objective - we are just

regressing against $u_t^{\text{target}}(x|z)$. So it is not too different from supervised learning after all. Finally, the algorithm is extremely simple - it is hard to think of a much simpler training objective. All of this makes flow matching an extremely appealing method for large-scale machine learning models. Once u_t^θ has been trained, we may simulate the flow model

$$\mathrm{d}X_t = u_t^\theta(X_t)\,\mathrm{d}t, \qquad X_0 \sim p_{\text{init}} \tag{27}$$

via e.g., Algorithm 1 to obtain samples $X_1 \sim p_{\text{data}}$. This whole pipeline is called **flow matching** in the literature [5, 7, 1, 6]. Let us now instantiate the conditional flow matching loss for the choice of Gaussian probability paths:

Example 13 (Flow Matching for Gaussian Conditional Probability Paths)
Let us return to the example of Gaussian probability paths $p_t(\cdot|z) = \mathcal{N}(\alpha_t z; \beta_t^2 I_d)$, where we may sample from the conditional path via

$$\epsilon \sim \mathcal{N}(0, I_d) \quad \Rightarrow \quad x_t = \alpha_t z + \beta_t \epsilon \sim \mathcal{N}(\alpha_t z, \beta_t^2 I_d) = p_t(\cdot|z). \tag{28}$$

As we derived in Equation (20), the conditional vector field $u_t^{\text{target}}(x|z)$ is given by

$$u_t^{\text{target}}(x|z) = \left(\dot\alpha_t - \frac{\dot\beta_t}{\beta_t}\alpha_t\right) z + \frac{\dot\beta_t}{\beta_t} x, \tag{29}$$

where $\dot\alpha_t = \partial_t \alpha_t$ and $\dot\beta_t = \partial_t \beta_t$ are the respective time derivatives. Plugging in this formula, the conditional flow matching loss reads

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim\mathcal{N}(\alpha_t z, \beta_t^2 I_d)}[\|u_t^\theta(x) - \left(\dot\alpha_t - \frac{\dot\beta_t}{\beta_t}\alpha_t\right) z - \frac{\dot\beta_t}{\beta_t} x\|^2] \tag{30}$$

$$\overset{(i)}{=} \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0, I_d)}[\|u_t^\theta(\alpha_t z + \beta_t \epsilon) - (\dot\alpha_t z + \dot\beta_t \epsilon)\|^2] \tag{31}$$

where in (i) we plugged in Equation (28) and replaced x by $\alpha_t z + \beta_t \epsilon$. Note the simplicity of $\mathcal{L}_{\text{CFM}}$: We sample a data point z, sample some noise ϵ and then we take a mean squared error. Let us make this even more concrete for the special case of $\alpha_t = t$, and $\beta_t = 1 - t$. The corresponding probability $p_t(x|z) = \mathcal{N}(tz, (1-t)^2)$ is sometimes referred to as the (Gaussian) **CondOT probability path**. Then we have $\dot\alpha_t = 1, \dot\beta_t = -1$, so that

$$\mathcal{L}_{\text{cfm}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0, I_d)}[\|u_t^\theta(tz + (1-t)\epsilon) - (z - \epsilon)\|^2]$$

Many famous state-of-the-art models have been trained using this simple yet effective procedure, e.g. *Stable Diffusion 3*, Meta's *Movie Gen Video*, and probably many more proprietary models. In Figure 7, we visualize it in a simple example and in Algorithm 3 we summarize the training procedure.

Let us summarize the results of this section.

Summary 14 (Flow Matching)
Flow matching training consists of learning the marginal vector field u_t^{target}. To construct it, we choose a

Algorithm 3 Flow Matching Training Procedure (for Gaussian CondOT path $p_t(x|z) = \mathcal{N}(tz, (1-t)^2)$)

Require: A dataset of samples $z \sim p_{\text{data}}$, neural network u_t^θ

1: **for** each mini-batch of data **do**
2: Sample a data example z from the dataset.
3: Sample a random time $t \sim \text{Unif}_{[0,1]}$.
4: Sample noise $\epsilon \sim \mathcal{N}(0, I_d)$
5: Set

$$x = tz + (1-t)\epsilon \qquad \text{(General case: } x \sim p_t(\cdot \mid z)\text{)}$$

6: Compute loss

$$\mathcal{L}(\theta) = \|u_t^\theta(x) - (z - \epsilon)\|^2 \qquad \text{(General case: } = \|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2\text{)}$$

7: Update $\theta \leftarrow \text{grad\_update}(\mathcal{L}(\theta))$.
8: **end for**

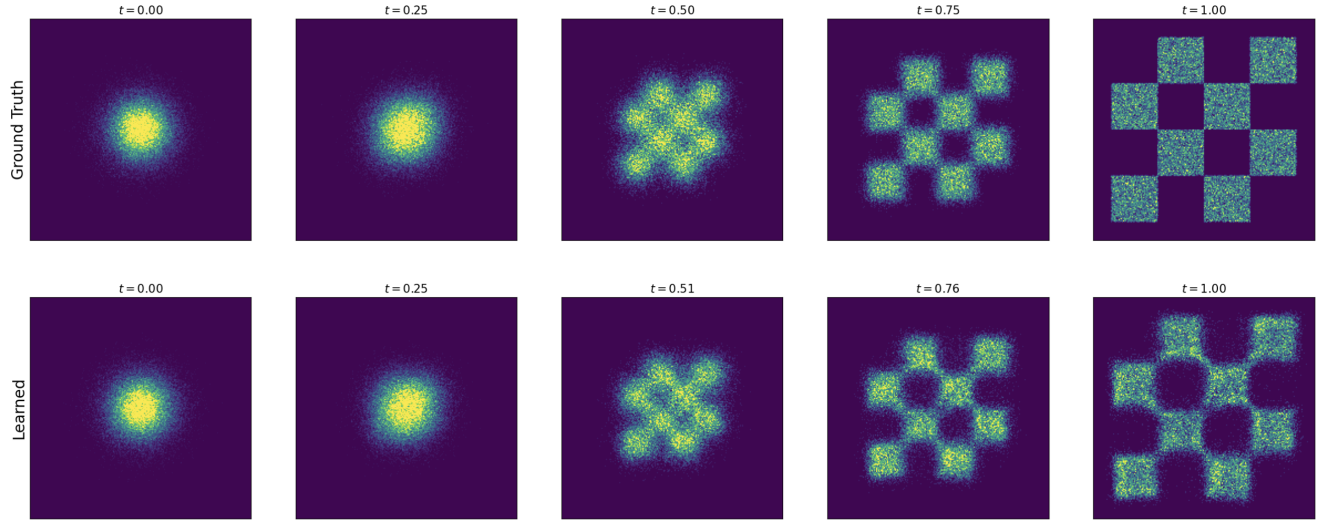

Figure 7: Illustration of Theorem 12 with a Gaussian CondOT probability path: simulating an ODE from a trained flow matching model. The data distribution is the chess board pattern (top right). Top row: Histogram from ground truth marginal probability path $p_t(x)$. Bottom row: Histogram of samples from flow matching model. As one can see, the top row and bottom row match after training (up to training error). The model was trained using Algorithm 3.

conditional probability path $p_t(x|z)$ that fulfils $p_0(\cdot|z) = p_{\text{init}}$, $p_1(\cdot|z) = \delta_z$. Next, we find a **conditional vector field** $u_t^{\text{target}}(x|z)$ such that its corresponding flow $\psi_t^{\text{target}}(x|z)$ fulfills

$$X_0 \sim p_{\text{init}} \quad \Rightarrow \quad X_t = \psi_t^{\text{target}}(X_0|z) \sim p_t(\cdot|z),$$

or, equivalently, that u_t^{target} satisfies the continuity equation. Then the **marginal vector field** defined by

$$u_t^{\text{target}}(x) = \int u_t^{\text{target}}(x|z) \frac{p_t(x|z)p_{\text{data}}(z)}{p_t(x)} \mathrm{d}z, \tag{32}$$

follows the marginal probability path, i.e.,

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t^{\text{target}}(X_t)\mathrm{d}t \Rightarrow X_t \sim p_t \quad (0 \leq t \leq 1). \tag{33}$$

In particular, $X_1 \sim p_{\text{data}}$ for this ODE, so that u_t^{target} "converts noise into data", as desired. To learn it, we minimize the **conditional flow matching loss**

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, x\sim p_t(\cdot|z)}[\|u_t^\theta(x) - u_t^{\text{target}}(x|z)\|^2]. \tag{34}$$

The most widely used example is the **Gaussian probability path**. For this case, the formulas become:

$$p_t(x|z) = \mathcal{N}(x; \alpha_t z, \beta_t^2 I_d) \tag{35}$$

$$u_t^{\text{flow}}(x|z) = \left(\dot{\alpha}_t - \frac{\dot{\beta}_t}{\beta_t}\alpha_t\right) z + \frac{\dot{\beta}_t}{\beta_t} x \tag{36}$$

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\text{Unif}, z\sim p_{\text{data}}, \epsilon\sim\mathcal{N}(0,I_d)}[\|u_t^\theta(\alpha_t z + \beta_t \epsilon) - (\dot{\alpha}_t z + \dot{\beta}_t \epsilon)\|^2] \tag{37}$$

for **noise schedulers** $\alpha_t, \beta_t \in \mathbb{R}$, i.e. continuously differentiable, monotonic functions that we choose such that $\alpha_0 = \beta_1 = 0$ $\alpha_1 = \beta_0 = 1$ (e.g. $\alpha_t = t, \beta_t = 1 - t$).

4 Score Functions and Score Matching

5 Guidance: How To Condition on a Prompt

6 Building Large-Scale Image or Video Generators

7 Need for Speed: Distillation, Flow Maps and Consistency Models

8 Discrete Diffusion Models: Building Language Models with Diffusion

9 References

[1] Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. "Stochastic interpolants: A unifying framework for flows and diffusions". In: *arXiv preprint arXiv:2303.08797* (2023).

[2] Earl A Coddington, Norman Levinson, and T Teichmann. *Theory of ordinary differential equations*. 1956.

[3] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[4] Arieh Iserles. *A first course in the numerical analysis of differential equations*. Cambridge university press, 2009.

[5] Yaron Lipman et al. "Flow matching for generative modeling". In: *arXiv preprint arXiv:2210.02747* (2022).

[6] Yaron Lipman et al. "Flow Matching Guide and Code". In: *arXiv preprint arXiv:2412.06264* (2024).

[7] Xingchao Liu, Chengyue Gong, and Qiang Liu. "Flow straight and fast: Learning to generate and transfer data with rectified flow". In: *arXiv preprint arXiv:2209.03003* (2022).

[8] Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

[9] Lawrence Perko. *Differential equations and dynamical systems*. Vol. 7. Springer Science & Business Media, 2013.

[10] Yang Song et al. *Score-Based Generative Modeling through Stochastic Differential Equations*. 2021. arXiv: 2011.13456 [cs.LG]. URL: https://arxiv.org/abs/2011.13456.

[11] Yang Song et al. "Score-based generative modeling through stochastic differential equations". In: *arXiv preprint arXiv:2011.13456* (2020).

A A Reminder on Probability Theory

We present a brief overview of basic concepts from probability theory. This section was partially taken from [6].

A.1 Random vectors

Consider data in the d-dimensional Euclidean space $x = (x^1, \dots, x^d) \in \mathbb{R}^d$ with the standard Euclidean inner product $\langle x, y \rangle = \sum_{i=1}^d x^i y^i$ and norm $\|x\| = \sqrt{\langle x, x \rangle}$. We will consider random variables (RVs) $X \in \mathbb{R}^d$ with continuous probability density function (PDF), defined as a *continuous* function $p_X : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ providing event A with probability

$$\mathbb{P}(X \in A) = \int_A p_X(x) \mathrm{d}x, \tag{38}$$

where $\int p_X(x)\mathrm{d}x = 1$. By convention, we omit the integration interval when integrating over the whole space ($\int \equiv \int_{\mathbb{R}^d}$). To keep notation concise, we will refer to the PDF p_{X_t} of RV X_t as simply p_t. We will use the notation $X \sim p$ or $X \sim p(X)$ to indicate that X is distributed according to p. One common PDF in generative modeling is the d-dimensional isotropic Gaussian:

$$\mathcal{N}(x; \mu, \sigma^2 I) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{\|x - \mu\|_2^2}{2\sigma^2}\right), \tag{39}$$

where $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}_{>0}$ stand for the mean and the standard deviation of the distribution, respectively.

The expectation of a RV is the constant vector closest to X in the least-squares sense:

$$\mathbb{E}[X] = \arg\min_{z \in \mathbb{R}^d} \int \|x - z\|^2 p_X(x)\mathrm{d}x = \int x p_X(x)\mathrm{d}x. \tag{40}$$

One useful tool to compute the expectation of *functions of RVs* is the *law of the unconscious statistician*:

$$\mathbb{E}[f(X)] = \int f(x) p_X(x)\mathrm{d}x. \tag{41}$$

When necessary, we will indicate the random variables under expectation as $\mathbb{E}_X f(X)$.

A.2 Conditional densities and expectations

Given two random variables $X, Y \in \mathbb{R}^d$, their joint PDF $p_{X,Y}(x, y)$ has marginals

$$\int p_{X,Y}(x, y)\mathrm{d}y = p_X(x) \text{ and } \int p_{X,Y}(x, y)\mathrm{d}x = p_Y(y). \tag{42}$$

See Figure 8 for an illustration of the joint PDF of two RVs in $\mathbb{R}$ ($d = 1$). The *conditional* PDF $p_{X|Y}$ describes the PDF of the random variable X when conditioned on an event $Y = y$ with density $p_Y(y) > 0$:

$$p_{X|Y}(x|y) \coloneqq \frac{p_{X,Y}(x, y)}{p_Y(y)}, \tag{43}$$

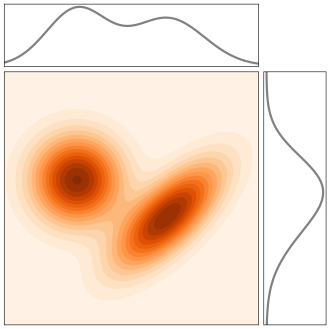

Figure 8: Joint PDF $p_{X,Y}$ (in shades) and its marginals p_X and p_Y (in black lines). Figure from [6]

.

and similarly for the conditional PDF $p_{Y|X}$. Bayes' rule expresses the conditional PDF $p_{Y|X}$ with $p_{X|Y}$ by

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)}, \tag{44}$$

for $p_X(x) > 0$.

The *conditional expectation* $\mathbb{E}[X|Y]$ is the best approximating *function* $g_\star(Y)$ to X in the least-squares sense:

$$\begin{aligned} g_\star &\coloneqq \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \mathbb{E}\left[\|X - g(Y)\|^2\right] = \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \|x - g(y)\|^2 p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \left[\int \|x - g(y)\|^2 p_{X|Y}(x|y)\mathrm{d}x\right] p_Y(y)\mathrm{d}y. \end{aligned} \tag{45}$$

For $y \in \mathbb{R}^d$ such that $p_Y(y) > 0$ the conditional expectation function is therefore

$$\mathbb{E}[X|Y = y] \coloneqq g_\star(y) = \int x p_{X|Y}(x|y)\mathrm{d}x, \tag{46}$$

where the second equality follows from taking the minimizer of the inner brackets in Equation (45) for $Y = y$, similarly to Equation (40). Composing $g_\star$ with the random variable Y, we get

$$\mathbb{E}[X|Y] \coloneqq g_\star(Y), \tag{47}$$

which is a random variable in $\mathbb{R}^d$. Rather confusingly, both $\mathbb{E}[X|Y = y]$ and $\mathbb{E}[X|Y]$ are often called *conditional expectation*, but these are different objects. In particular, $\mathbb{E}[X|Y = y]$ is a function $\mathbb{R}^d \to \mathbb{R}^d$, while $\mathbb{E}[X|Y]$ is a random variable assuming values in $\mathbb{R}^d$. To disambiguate these two terms, our discussions will employ the notations introduced here.

The *tower property* is an useful property that helps simplify derivations involving conditional expectations of two RVs X and Y:

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X] \tag{48}$$

Because $\mathbb{E}[X|Y]$ is a RV, itself a function of the RV Y, the outer expectation computes the expectation of $\mathbb{E}[X|Y]$. The tower property can be verified by using some of the definitions above:

$$\begin{aligned} \mathbb{E}[\mathbb{E}[X|Y]] &= \int \left(\int x p_{X|Y}(x|y)\mathrm{d}x\right) p_Y(y)\mathrm{d}y \\ &\overset{(43)}{=} \int\int x p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &\overset{(42)}{=} \int x p_X(x)\mathrm{d}x = \mathbb{E}[X]. \end{aligned}$$

Finally, consider a helpful property involving two RVs $f(X,Y)$ and Y, where X and Y are two arbitrary RVs. Then, by using the Law of the Unconscious Statistician with (46), we obtain the identity

$$\mathbb{E}[f(X,Y)|Y = y] = \int f(x,y) p_{X|Y}(x|y)\mathrm{d}x. \tag{49}$$

B A Proof of the Fokker-Planck equation

In this section, we give here a self-contained proof of the Fokker-Planck equation which includes the continuity equation as a special case (Theorem 11). We stress that **this section is not necessary to understand the remainder of this document** and is mathematically more advanced. If you desire to understand where the Fokker-Planck equation comes from, then this section is for you.

Theorem 15 (Fokker-Planck Equation)
Let p_t be a probability path with $p_0 = p_{\text{init}}$ and let us consider the SDE

$$X_0 \sim p_{\text{init}}, \quad \mathrm{d}X_t = u_t(X_t)\mathrm{d}t + \sigma_t \mathrm{d}W_t.$$

Then X_t has distribution p_t for all $0 \leq t \leq 1$ if and only if the **Fokker-Planck equation** holds:

$$\partial_t p_t(x) = -\mathrm{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2}\Delta p_t(x) \quad \text{for all } x \in \mathbb{R}^d, 0 \leq t \leq 1, \tag{50}$$

We start by showing that the Fokker-Planck is a necessary condition, i.e. if $X_t \sim p_t$, then the Fokker-Planck equation is fulfilled. The trick for the proof is to use **test functions** f, i.e. functions $f : \mathbb{R}^d \to \mathbb{R}$ that are infinitely differentiable ("smooth") and are only non-zero within a bounded domain (compact support). We use the fact that for arbitrary integrable functions $g_1, g_2 : \mathbb{R}^d \to \mathbb{R}$ it holds that

$$g_1(x) = g_2(x) \text{ for all } x \in \mathbb{R}^d \quad \Leftrightarrow \quad \int f(x) g_1(x) \mathrm{d}x = \int f(x) g_2(x) \mathrm{d}x \text{ for all test functions } f \tag{51}$$

In other words, we can express the pointwise equality as equality of taking integrals. The useful thing about test functions is that they are smooth, i.e. we can take gradients and higher-order derivatives. In particular, we can use **integration by parts** for arbitrary test functions f_1, f_2:

$$\int f_1(x) \frac{\partial}{\partial x_i} f_2(x) \mathrm{d}x = -\int f_2(x) \frac{\partial}{\partial x_i} f_1(x) \mathrm{d}x \tag{52}$$

under the condition that f_1, f_2 and their product $f_1 \cdot f_2$ is integrable. By using this together with the definition of the divergence and Laplacian (see Equation (22)), we get the identities:

$$\int \nabla f_1^T(x) f_2(x) \mathrm{d}x = -\int f_1(x) \mathrm{div}(f_2)(x) \mathrm{d}x \quad (f_1 : \mathbb{R}^d \to \mathbb{R}, f_2 : \mathbb{R}^d \to \mathbb{R}^d) \tag{53}$$

$$\int f_1(x) \Delta f_2(x) \mathrm{d}x = \int f_2(x) \Delta f_1(x) \mathrm{d}x \quad (f_1 : \mathbb{R}^d \to \mathbb{R}, f_2 : \mathbb{R}^d \to \mathbb{R}) \tag{54}$$

Now let's proceed to the proof. We use the stochastic update of SDE trajectories as in Equation (6):

$$X_{t+h} = X_t + h u_t(X_t) + \sigma_t(W_{t+h} - W_t) + h R_t(h) \tag{55}$$

$$\approx X_t + h u_t(X_t) + \sigma_t(W_{t+h} - W_t) \tag{56}$$

where for now we simply ignore the error term $R_t(h)$ for readability as we will take $h \to 0$ anyway. We can then

make the following calculation:

$$\begin{aligned}
f(X_{t+h}) - f(X_t) &\overset{(56)}{=} f(X_t + hu_t(X_t) + \sigma_t(W_{t+h} - W_t)) - f(X_t) \\
&\overset{(i)}{=} \nabla f(X_t)^T \left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right) \\
&\quad + \frac{1}{2}\left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right)^T \nabla^2 f(X_t) \left(hu_t(X_t) + \sigma_t(W_{t+h} - W_t)\right) \\
&\overset{(ii)}{=} h\nabla f(X_t)^T u_t(X_t) + \sigma_t \nabla f(X_t)^T (W_{t+h} - W_t) \\
&\quad + \frac{1}{2} h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + h\sigma_t u_t(X_t)^T \nabla^2 f(X_t)(W_{t+h} - W_t) + \\
&\quad + \frac{1}{2}\sigma_t^2 (W_{t+h} - W_t)^T \nabla^2 f(X_t)(W_{t+h} - W_t)
\end{aligned}$$

where in (i) we used a 2nd Taylor approximation of f around X_t and in (ii) we used the fact that the Hessian $\nabla^2 f$ is a symmetric matrix. Note that $\mathbb{E}[W_{t+h} - W_t | X_t] = 0$ and $W_{t+h} - W_t | X_t \sim \mathcal{N}(0, hI_d)$. Therefore

$$\begin{aligned}
&\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t] \\
=&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2} h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \mathbb{E}_{\epsilon_t \sim \mathcal{N}(0, I_d)}[\epsilon_t^T \nabla^2 f(X_t) \epsilon_t] \\
\overset{(i)}{=}&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2} h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \mathrm{trace}(\nabla^2 f(X_t)) \\
\overset{(ii)}{=}&h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2} h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \Delta f(X_t)
\end{aligned}$$

where in (i) we used the fact that $\mathbb{E}_{\epsilon_t \sim \mathcal{N}(0, I_d)}[\epsilon_t^T A \epsilon_t] = \mathrm{trace}(A)$ and in (ii) we used the definition of the Laplacian and the Hessian matrix. With this, we get that

$$\begin{aligned}
&\partial_t \mathbb{E}[f(X_t)] \\
=&\lim_{h \to 0} \frac{1}{h} \mathbb{E}[f(X_{t+h}) - f(X_t)] \\
=&\lim_{h \to 0} \frac{1}{h} \mathbb{E}[\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t]] \\
=&\mathbb{E}[\lim_{h \to 0} \frac{1}{h} \left(h\nabla f(X_t)^T u_t(X_t) + \frac{1}{2} h^2 u_t(X_t)^T \nabla^2 f(X_t) u_t(X_t) + \frac{h}{2}\sigma_t^2 \Delta f(X_t)\right)] \\
=&\mathbb{E}[\nabla f(X_t)^T u_t(X_t) + \frac{1}{2}\sigma_t^2 \Delta f(X_t)] \\
\overset{(i)}{=}&\int \nabla f(x)^T u_t(x) p_t(x) \mathrm{d}x + \int \frac{1}{2}\sigma_t^2 \Delta f(x) p_t(x) \mathrm{d}x \\
\overset{(ii)}{=}&-\int f(x) \mathrm{div}(u_t p_t)(x) \mathrm{d}x + \int \frac{1}{2}\sigma_t^2 f(x) \Delta p_t(x) \mathrm{d}x \\
=&\int f(x) \left(-\mathrm{div}(u_t p_t)(x) + \frac{1}{2}\sigma_t^2 \Delta p_t(x)\right) \mathrm{d}x
\end{aligned}$$

where in (i) we used the assumption that p_t as the distribution of X_t and in (ii) we used Equation (53) and Equation (54). Note that to use this, we require integrability of the product $p_t(x)u_t(x)$, i.e. such that

$$\int p_t(x) \|u_t(x)\| \mathrm{d}x < \infty$$

Note that this condition almost always holds in machine learning (bounded data and functions because of numerical precision limits). Therefore, it holds that

$$\partial_t \mathbb{E}[f(X_t)] = \int f(x) \left(-\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x) \right) \mathrm{d}x \quad \text{(for all } f \text{ and } 0 \leq t \leq 1) \tag{57}$$

$$\overset{(i)}{\Leftrightarrow} \quad \partial_t \int f(x) p_t(x) \mathrm{d}x = \int f(x) \left(-\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x) \right) \mathrm{d}x \quad \text{(for all } f \text{ and } 0 \leq t \leq 1) \tag{58}$$

$$\overset{(ii)}{\Leftrightarrow} \quad \int f(x) \partial_t p_t(x) \mathrm{d}x = \int f(x) \left(-\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x) \right) \mathrm{d}x \quad \text{(for all } f \text{ and } 0 \leq t \leq 1) \tag{59}$$

$$\overset{(iii)}{\Leftrightarrow} \quad \partial_t p_t(x) = -\operatorname{div}(p_t u_t)(x) + \frac{\sigma_t^2}{2} \Delta p_t(x) \quad \text{(for all } x \in \mathbb{R}^d, 0 \leq t \leq 1) \tag{60}$$

where in (i) we used the assumption that $X_t \sim p_t$, in (ii) we swapped the derivative with the integral and (iii) we used Equation (51) . This completes the proof that the Fokker-Planck equation is a necessary condition.

Finally, we explain why it is also a sufficient condition. The Fokker-Planck equation is a partial differential equation (PDE). More specifically, it is a so-called *parabolic partial differential equation*. Similar to Theorem 3, such differential equations have a unique solution given fixed initial conditions (see e.g. [3, Chapter 7]). Now, if Equation (50) holds for p_t, we just shown above that it must also hold for true distribution q_t of X_t (i.e. $X_t \sim q_t$) - in other words, both p_t and q_t are solutions to the parabolic PDE. Further, we know that the initial conditions are the same, i.e. $p_0 = q_0 = p_{\text{init}}$ by construction of an interpolating probability path. Hence, by uniqueness of the solution of the differential equation, we know that $p_t = q_t$ for all $0 \leq t \leq 1$ - which means $X_t \sim q_t = p_t$ and which is what we wanted to show.