Generative AI With Stochastic Differential Equations: Course Notes

Peter Holderrieth and Ezra Erives

1 Introduction

> Creating noise from data is easy; creating data from noise is generative modeling.
>
> Song et al. [6]

1.1 Overview

In recent years, we all have witnessed a tremendous revolution in artificial intelligence (AI). Image generators like *Stable Diffusion* can generate photorealistic and artistic images across a diverse range of styles, video models like *Sora* 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 to *predict* things, these new AI system are creative: they create, 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** and **flow matching**. These models are the backbone of the best image, audio, or video generation models (e.g. Stable Diffusion, Sora) - and 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 constructed via a **ordinary or stochastic differential equation (SDEs)**. Flow matching and denoising diffusion models are a family of techniques that allow us to (1) construct, (2) train, and (3) simulate, such 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 class, our goal is to equip you with the necessary mathematical toolbox about differential equations to allow you to systematically understand these models and provide a self-contained introduction to these algorithms. While this will provide you with a systematic understanding of generative models, it is the instructors' firm belief that true learning requires *getting your hands dirty*, i.e. to code. Accordingly, this course involves a hands-on coding component, which will allow you to implement and experiment with concepts taught in class, and which will culminate in the design of your very own generative model built from scratch. Therefore, the goal of this class is to give a systematic and principled understanding of *why* and *how* flow and diffusion models work. Finally, we believe that flow and diffusion models are not only very useful but also very beautiful concepts. Therefore, most importantly, we hope that this class will be a lot of fun to you.

1.2 Course Overview

We give a brief overview over of this document.

1. **Section, From generation to sampling:** We will conclude this section by briefly formalizing 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.

2. **Section, Flow and Diffusion Models:** Next, 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 construct them with neural networks.
3. **Section, Finding a training target:** To train our generative model, we must first pin down precisely what it is that our model is supposed to approximate. In other words, what's the ground truth? We will introduce the celebrated *Fokker-Planck equation*, a fundamental result from stochastic calculus which governs the evolution of a distribution under the flow of an ODE or SDE, and which will allow us to formalize the notion of ground truth.
4. **Section, Training algorithm:** This section formulates a *training objective*, allowing us to approximate the training target, or ground truth, of the previous section. With this, we are ready to provide a minimal implementation of flow matching and denoising diffusion models.
5. **Section, Building an image generator:** We next focus on using our algorithms to build a conditional image generator. In order to do so, we need to discuss two crucial design choices: (1) neural network architectures and (2) how to condition on a specific prompt (e.g. "generate an image of a cat").
6. **Section, Advanced topics:** Finally, we discuss some advanced topics. We discuss distillation (making diffusion models faster), reward-finetuning (aligning with human feedback) as well as building diffusion-like models for other data types such as data on manifolds.

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 appendix A. Don't worry if some of the concepts there are unfamiliar to you. We are always happy to explain them after class.

1.3 Generative Modeling As Sampling

Let's begin by thinking about various data types, or *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 $z \in \mathbb{R}^{N \times 3}$ 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 via autoregressive language models (such as *ChatGPT*). While flow and diffusion models for discrete data have been developed, this course focuses exclusively on applications to continuous data.

Generation as Sampling. Let us think for a second 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 think of this diversity of possible images as a *probability distribution*. We call it the **data distribution** and denote it as p_{data}. In the example of dog images, this distribution would therefore give higher likelihood to images that look more like a dog. Therefore, how "good" an image/video/molecule fits 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 use YouTube as a database. For protein structures, we can use experimental data bases from sources such as the Protein Data Bank (PDB) that collected scientific measurements over decades. As the size of our dataset grows very large, it becomes an increasingly better representation of the underlying distribution p_{data}.

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 (Conditional Generation)
Conditional 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 **conditional data distribution**. The conditional 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).

From Noise to Data. So far, we have discussed the *what* of generative modeling: generating samples from p_{data}. Here, we will briefly discuss the *how*. For this, we assume that we have access to some **initial distribution** p_{init} that we can easily sample from, such as the Gaussian $p_{\text{init}} = \mathcal{N}(0, I_d)$. The goal of a generative modeling is then to transform samples from $x \sim p_{\text{init}}$ into samples from p_{data}. We note that p_{init} does not have to be so simple as a Gaussian. As we shall see, there are interesting use cases for leveraging this flexibility. Despite this, we will stick with the name p_{init}, because in the majority of applications we take it to be a simple Gaussian.

Summary We summarize our discussion so far as follows:

Summary 1 (Generation as Sampling)
We summarize the findings of this section:

1. In this class, we consider the task of generating objects that are represented as vectors $z \in \mathbb{R}^d$ such as images, videos, or 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. Conditional 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 train a generative model to transform samples from a simple distribution p_{init} (e.g. a Gaussian) into samples from p_{data}.

2 Flow and Diffusion Models

In the previous section, we formalized generative modeling as sampling from a data distribution p_{data}. Further, we saw that sampling could be achieved via the transformation of samples from a simple distribution p_{init}, such as the Gaussian $\mathcal{N}(0, I_d)$, to samples from the target distribution p_{data}. In this section, we describe how the desired transformation can be obtained 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. Therefore, the goal of this section to construct the generative model used throughout this document: (1) Define what an ODE/SDE is and (2) Define how to parameterize an ODE/SDE in a 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 explain 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 fig. 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:

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

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

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

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

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

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

For a given initial condition 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

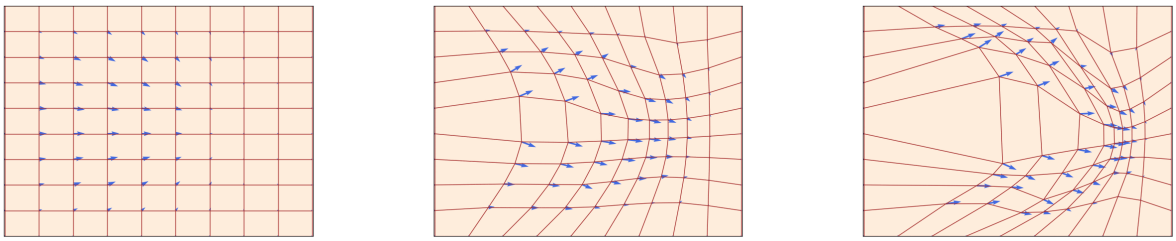

Figure 1: A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (red square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations (here, $d = 2$). We show three different times t. As one can see, a flow is a diffeomorphism that "warps" space. Figure from [3].

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

Theorem 2 (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 2 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 [5, 1].

Example 3 (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

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

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

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

where in (i) we used the chain rule. In fig. 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 a linear function. 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 [2]. 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

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

where $h = n^{-1} > 0$ is a step size hyperparameter with $n \in \mathbb{N}$. For most of this class, 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

$$\begin{aligned} X'_{t+h} &= X_t + hu_t(X_t) && \blacktriangleright \text{ initial guess of new state} \\ X_{t+h} &= X_t + \frac{h}{2}(u_t(X_t) + u_{t+h}(X'_{t+h})) && \blacktriangleright \text{ update with average } u \text{ at current and guessed state} \end{aligned}$$

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. We will mainly use the Euler method in this class.

Flow models. We can now construct a generative model via an ODE. Remember that our goal was to convert a simple distribution p_{init} into a complex distribution p_{data}. The simulation of an ODE is thus a natural choice for this transformation. A **flow model** is described by the ODE

$$\begin{aligned} X_0 &\sim p_{\text{init}} && \text{(random initialization)} \\ \frac{\mathrm{d}}{\mathrm{d}t} X_t &= u_t^\theta(X_t) && \text{(ODE)} \end{aligned}$$

where the vector field u_t^θ is a neural network u_t^θ with parameters θ. For now, we will speak of u_t^θ as being a generic neural network; i.e. a continuous 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. 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 is the vector field (not the flow)**. 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** $i = 1, \ldots, n-1$ **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 Stochastic Differential Equations (SDEs)

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

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

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 first define what a Brownian motion is. A **Brownian motion** $W = (W_t)_{0\le t\le 1}$ 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 $0 \le s < t$, i.e. increments have a Gaussian distribution with variance increasing linearly in time (I_d is the identity matrix).
2. **Independent increments:** For any $0 \le t_0 < t_1 < \cdots < t_n = 1$, 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

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

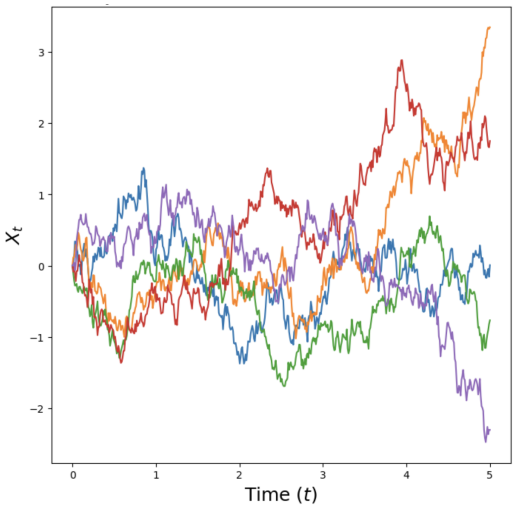

Figure 2: Sample trajectories of a Brownian motion W_t in dimension $d = 1$ simulated using eq. (5).

In fig. 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 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, there is *no derivative* anymore. Hence,

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

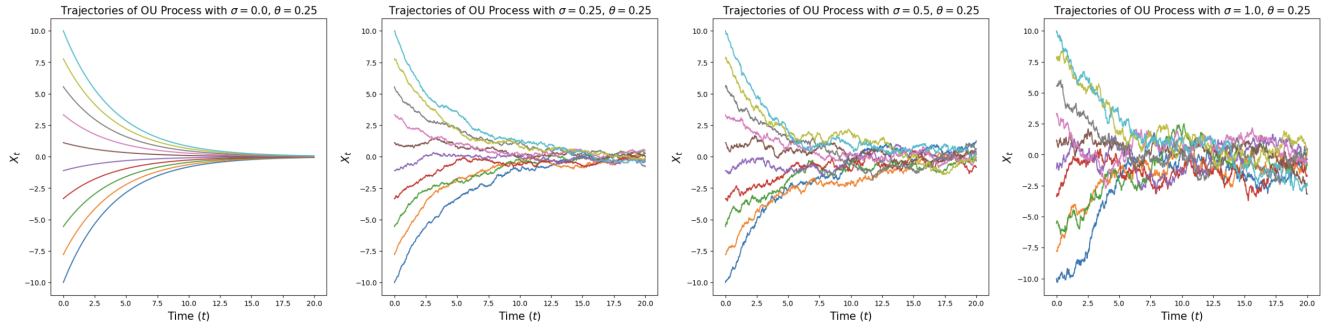

Figure 3: Illustration of Ornstein-Uhlenbeck processes (eq. (8)) in dimension $d = 1$ for $\theta = 0.25$ and various choices of σ (increasing from left to right). For $\sigma = 0$, we recover a flow (smooth, deterministic trajectories) that converges to the origin as $t \to \infty$. For $\sigma > 0$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

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:

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

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

However, always keep in mind that the "$\mathrm{d}X_t$"-notation above is a purely informal notation of eq. (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 4 (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 eq. (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 integrals*. As we focus on machine learning in this class, we refer to a proof to [4]. 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 always include ODEs**.

Example 5 (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 $\theta > 0$ resulting in 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 process**. We visualize it in fig. 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 $\mathcal{N}(0, \sigma^2)$ 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 eq. (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

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

where $h = n^{-1} > 0$ is a step size hyper-parameter with $n \in \mathbb{N}$. 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 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 make its central ingredient - the vector field u_t - a neural network u_t^θ. A

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

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

diffusion model is given by

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

In algorithm 2, we describe the algorithm to sample from a diffusion model with the Euler-Maruyama method. We can summarize the results of this section as:

Summary 6 (SDE generative model)
Throughout this document, an **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:

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

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 References

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

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

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

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

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

[6] 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.

A A Reminder on Probability Theory

We give here a brief reminder on basic concepts in probability theory. This section was partially taken from [3].

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

$$\mathbb{P}(X \in A) = \int_A p_X(x) \mathrm{d}x, \tag{10}$$

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:

$$\mathcal{N}(x; \mu, \sigma^2 I) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{\|x - \mu\|_2^2}{2\sigma^2}\right), \tag{11}$$

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:

$$\mathbb{E}[X] = \arg\min_{z \in \mathbb{R}^d} \int \|x - z\|^2 p_X(x)\mathrm{d}x = \int x p_X(x)\mathrm{d}x. \tag{12}$$

One useful tool to compute the expectation of *functions of RVs* is the *Law of the Unconscious Statistician*:

$$\mathbb{E}[f(X)] = \int f(x) p_X(x)\mathrm{d}x. \tag{13}$$

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

$$\int p_{X,Y}(x, y)\mathrm{d}y = p_X(x) \text{ and } \int p_{X,Y}(x, y)\mathrm{d}x = p_Y(y). \tag{14}$$

See fig. 4 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 $p_Y(y) > 0$:

$$p_{X|Y}(x|y) \coloneqq \frac{p_{X,Y}(x, y)}{p_Y(y)}, \tag{15}$$

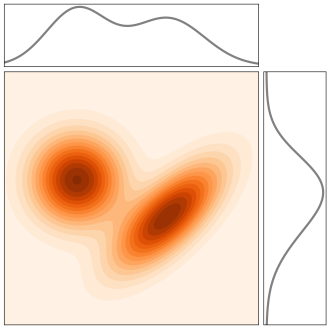

Figure 4: Joint PDF $p_{X,Y}$ (in shades) and its marginals p_X and p_Y (in black lines). Figure from [3]

.

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

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)}, \tag{16}$$

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:

$$\begin{aligned} g_\star &\coloneqq \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \mathbb{E}\left[\|X - g(Y)\|^2\right] = \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \|x - g(y)\|^2 p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \left[\int \|x - g(y)\|^2 p_{X|Y}(x|y)\mathrm{d}x\right] p_Y(y)\mathrm{d}y. \end{aligned} \tag{17}$$

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

$$\mathbb{E}[X|Y = y] \coloneqq g_\star(y) = \int x p_{X|Y}(x|y)\mathrm{d}x, \tag{18}$$

where the second equality follows from taking the minimizer of the inner brackets in eq. (17) for $Y = y$, similarly to eq. (12). Composing $g_\star$ with the random variable Y, we get

$$\mathbb{E}[X|Y] \coloneqq g_\star(Y), \tag{19}$$

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:

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X] \tag{20}$$

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:

$$\begin{aligned} \mathbb{E}[\mathbb{E}[X|Y]] &= \int \left(\int x p_{X|Y}(x|y)\mathrm{d}x\right) p_Y(y)\mathrm{d}y \\ &\overset{(15)}{=} \int\int x p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &\overset{(14)}{=} \int x p_X(x)\mathrm{d}x \\ &= \mathbb{E}[X]. \end{aligned}$$

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 (18), we obtain the identity

$$\mathbb{E}[f(X,Y)|Y = y] = \int f(x,y) p_{X|Y}(x|y)\mathrm{d}x. \tag{21}$$