MIT Class 6.S184: *Generative AI With Stochastic Differential Equations*, 2026

# 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. [9]

## 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** [10] and **flow matching** [4, 6, 1, 5]. 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 $z = (z^1, \ldots, z^N) \in \mathbb{R}^{3\times N}$ 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 $p_{\text{data}}$. Mathematically, one can think of $p_{\text{data}}$ 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 $p_{\text{data}}$. With this, we can mathematically express the task of generation as sampling from the (unknown) distribution $p_{\text{data}}$:

**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 $p_{\text{data}}$. 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 $p_{\text{data}}$, which together serve as a proxy for the true distribution.

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

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 $p_{\text{data}}$.

**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 $p_{\text{data}}$ 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 $p_{\text{data}}$ having access to a dataset of samples $z_1, \ldots, z_N \sim p_{\text{data}}$ 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 $(z_1, y) \ldots, (z_N, y)$ during training.
4. Our goal is to construct a generative model, i.e. a model that returns samples from $p_{\text{data}}$ after training.

# 2 Flow and Diffusion Models

In the previous section, we formalized generative modeling as sampling from a data distribution $p_{\text{data}}$. 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

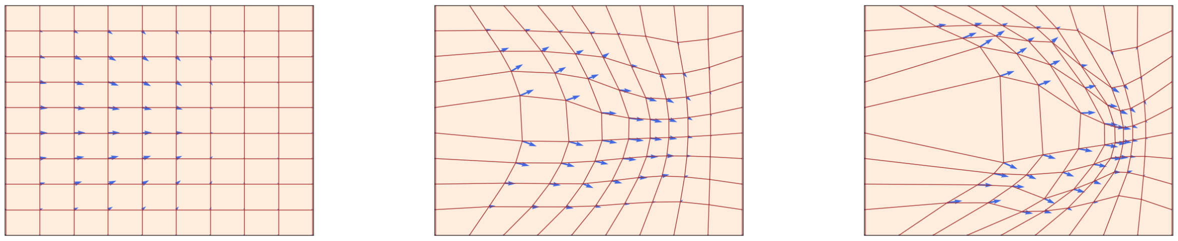
Figure 1: A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (red square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations (here, $d = 2$). We show three different times $t$. As one can see, a flow is a diffeomorphism that "warps" space. Figure from [5].

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 $\psi_t$. In this case, $\psi_t$ is a **diffeomorphism** for all $t$, i.e. $\psi_t$ is continuously differentiable with a continuously differentiable inverse $\psi_t^{-1}$.

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 [8, 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

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

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

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

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 $\psi_t$ 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 [3]. 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

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

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.

**Flow models.** We can now construct a generative model via an ODE by making the vector field a **neural network vector field** $u_t^\theta$. For now, we simply mean that $u_t^\theta$ is a parameterized function $u_t^\theta : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ with parameters $\theta$. 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 $p_{\text{data}}$. 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** $p_{\text{init}}$. 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

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

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

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

where $\psi_t^\theta$ describes the flow induced by $u_t^\theta$. 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 $u_t^\theta$, 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:     $X_{t+h} = X_t + hu_t^\theta(X_t)$
6:     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

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

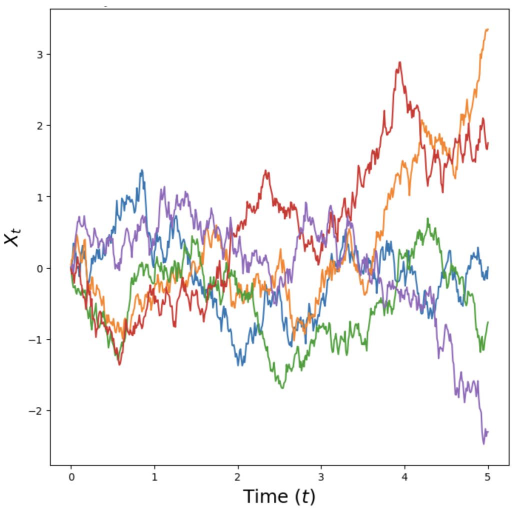

Figure 2: Sample trajectories of a Brownian motion $W_t$ in dimension $d=1$ simulated using Equation (5).

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.

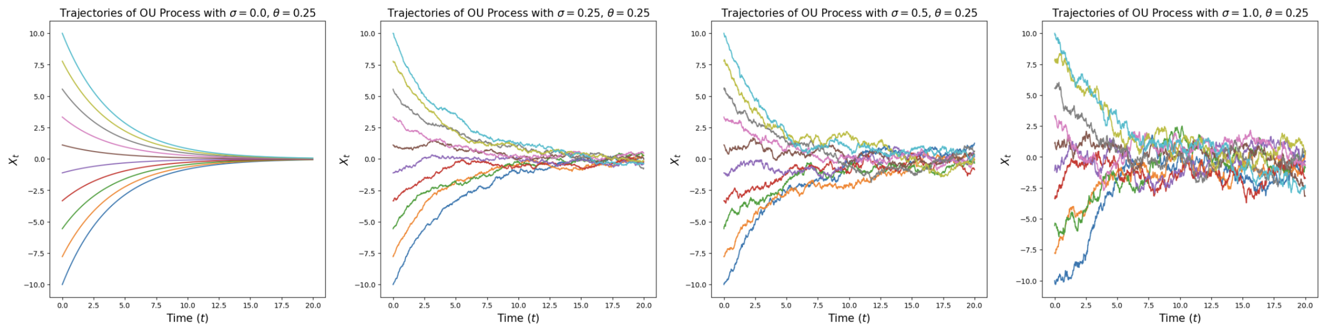

Figure 3: Illustration of Ornstein-Uhlenbeck processes (Equation (8)) in dimension $d = 1$ for $\theta = 0.25$ and various choices of $\sigma$ (increasing from left to right). For $\sigma = 0$, we recover a flow (smooth, deterministic trajectories) that converges to the origin as $t \to \infty$. For $\sigma > 0$ 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 $\phi_t$ 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 $\sigma_t$ 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 [7] 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 $\sigma$ 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

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

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 $p_{\text{init}}$ into a complex distribution $p_{\text{data}}$. 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 $u_t^\theta$, number of steps $n$, diffusion coefficient $\sigma_t$
1: Set $t = 0$
2: Set step size $h = \frac{1}{n}$
3: Draw a sample $X_0 \sim p_{\text{init}}$
4: for $i = 1, \dots, n$ 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 $X_1$
```

$u_t^\theta$. A **diffusion model** is thus given by

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

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 $u_t^\theta$ with parameters $\theta$ that parameterize a vector field and a fixed diffusion coefficient $\sigma_t$:

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

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

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

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

## 3 Flow Matching

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

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

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

[6] 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).

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

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

[9] 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.

[10] 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 [5].

### 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:

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

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 Figure 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 $p_Y(y) > 0$:

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

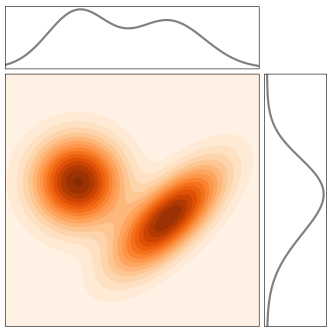
Figure 4: Joint PDF $p_{X,Y}$ (in shades) and its marginals $p_X$ and $p_Y$ (in black lines). Figure from [5]

.

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:

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

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 Equation (17) for $Y = y$, similarly to Equation (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

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