Flow Matching Guide and Code

Yaron Lipman[1], **Marton Havasi**[1], **Peter Holderrieth**[2], **Neta Shaul**[3], **Matt Le**[1], **Brian Karrer**[1], **Ricky T. Q. Chen**[1], **David Lopez-Paz**[1], **Heli Ben-Hamu**[3], **Itai Gat**[1]

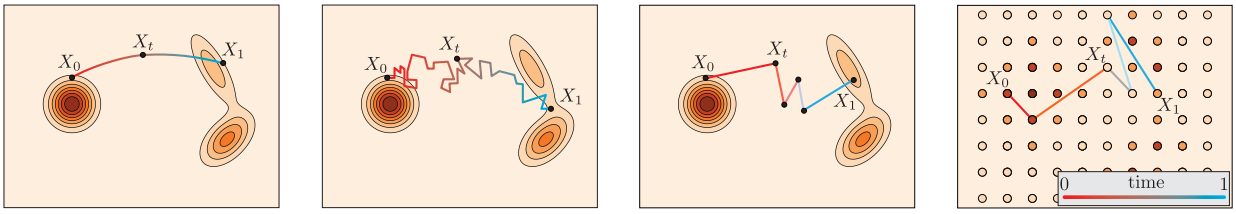
[1]FAIR at Meta, [2]MIT CSAIL, [3]Weizmann Institute of Science

Flow Matching (FM) is a recent framework for generative modeling that has achieved state-of-the-art performance across various domains, including image, video, audio, speech, and biological structures. This guide offers a comprehensive and self-contained review of FM, covering its mathematical foundations, design choices, and extensions. By also providing a PyTorch package featuring relevant examples (*e.g.*, image and text generation), this work aims to serve as a resource for both novice and experienced researchers interested in understanding, applying and further developing FM.

Date: December 9, 2024
Code: `flow_matching` library at https://github.com/facebookresearch/flow_matching

Meta

Contents



(a) Flow **(b)** Diffusion **(c)** Jump **(d)** CTMC

Figure 1 Four time-continuous processes $(X_t)_{0 \le t \le 1}$ taking source sample X_0 to a target sample X_1. These are a flow in a continuous state space, a diffusion in continuous state space, a jump process in continuous state space (densities visualized with contours), and a jump process in discrete state space (states as disks, probabilities visualized with colors).

1 Introduction

Flow matching (FM) (Lipman et al., 2022; Albergo and Vanden-Eijnden, 2022; Liu et al., 2022) is a simple framework for generative modeling framework that has pushed the state-of-the-art in various fields and large-scale applications including generation of images (Esser et al., 2024), videos (Polyak et al., 2024), speech (Le et al., 2024), audio (Vyas et al., 2023), proteins (Huguet et al., 2024), and robotics (Black et al., 2024). This manuscript and its accompanying codebase have two primary objectives. First, to serve as a comprehensive and self-contained reference to Flow Matching, detailing its design choices and numerous extensions developed by the research community. Second, to enable newcomers to quickly adopt and build upon Flow Matching for their own applications.

The framework of Flow Matching is based on learning a velocity field (also called vector field). Each velocity field defines a **flow** ψ_t by solving an ordinary differential equation (ODE) in a process called simulation. A flow is a determinstic, time-continuous bijective transformation of the d-dimensional Euclidean space, $\mathbb{R}^d$. The goal of Flow Matching is to build a flow that transforms a sample $X_0 \sim p$ drawn from a source distribution p into a target sample $X_1 := \psi_1(X_0)$ such that $X_1 \sim q$ has a desired distribution q, see figure 1a. Flow models were introduced to the machine learning community by (Chen et al., 2018; Grathwohl et al., 2018) as Continuous Normalizing Flows (CNFs). Originally, flows were trained by maximizing the likelihood $p(X_1)$ of training examples X_1, resulting in the need of simulation and its differentiation during training. Due to the resulting computational burdens, later works attempted to learn CNFs without simulation (Rozen et al., 2021; Ben-Hamu et al., 2022), evolving into modern-day Flow Matching algorithms (Lipman et al., 2022; Liu et al., 2022; Albergo and Vanden-Eijnden, 2022; Neklyudov et al., 2023; Heitz et al., 2023; Tong et al., 2023). The resulting framework is a recipe comprising two steps (Lipman et al., 2022), see figure 2: First, choose a probability path p_t interpolating between the source p and target q distributions. Second, train a velocity field (neural network) that defines the flow transformation ψ_t implementing p_t.

The principles of FM can be extended to state spaces $\mathcal{S}$ other than $\mathbb{R}^d$ and even evolution processes that are not flows. Recently, **Discrete Flow Matching** (Campbell et al., 2024; Gat et al., 2024) develops a Flow Matching algorithm for time-continuous Markov processes on discrete state spaces, also known as Continuous Time Markov Chains (CTMC), see figure 1c. This advancement opens up the exciting possibility of using Flow Matching in discrete generative tasks such as language modeling. **Riemannian Flow Matching** (Chen and Lipman, 2024) extends Flow Matching to flows on Riemannian manifolds $\mathcal{S} = \mathcal{M}$ that now became the state-of-the-art models for a wide variety of applications of machine learning in chemistry such as protein folding (Yim et al., 2023; Bose et al., 2023). Even more generally, **Generator Matching** (Holderrieth et al., 2024) shows that the Flow Matching framework works for any modality and for general Continuous Time Markov Processes (**CTMPs**) including, as illustrated in figure 1, **flows**, **diffusions**, and **jump processes** in continuous spaces, in addition to CTMC in discrete spaces. Remarkably, for any such CTMP, the Flow Matching recipe remains the same, namely: First, choose a path p_t interpolating source p and target q on the relevant state space $\mathcal{S}$. Second, train a *generator*, which plays a similar role to velocities for flows, and defines a CTMP process implementing p_t. This generalization of Flow Matching allows us to see many existing generative models in a unified light and develop new generative models for any modality with a generative Markov process of choice.

Chronologically, **Diffusion Models** were the first to develop simulation-free training of a CTMP process, namely a diffusion process, figure 1b. Diffusion Models were originally introduced as discrete time Gaussian processes (Sohl-Dickstein et al., 2015; Ho et al., 2020) and later formulated in terms of continuous time Stochastic Differential Equations (SDEs) (Song et al., 2021). In the lens of Flow Matching, Diffusion Models build the probability path p_t interpolating source and target distributions in a particular way via *forward noising processes* modeled by particular SDEs. These SDEs are chosen to have closed form marginal probabilities that are in turn used to parametrize the generator of the diffusion process (*i.e.*, drift and diffusion coefficient) via the *score function* (Song and Ermon, 2019). This parameterization is based on a reversal process to the forward noising process (Anderson, 1982). Consequently, Diffusion Models learn the score function of the marginal probabilities. Diffusion Models' literature suggested also other parametrizations of the generator besides the score, including *noise prediction*, *denoisers* (Kingma et al., 2021), or *v-prediction* (Salimans and Ho, 2022)—where the latter coincides with velocity prediction for a particular choice of probability path p_t. **Diffusion bridges** (Peluchetti, 2023) offers another approach to design p_t and generators for diffusion process that extends diffusion models to arbitrary source-target couplings. In particular these constructions are build again on SDEs with marginals known in closed form, and again use the score to formulate the generator (using Doob's h-transform). Shi et al. (2023); Liu et al. (2023) show that the linear version of Flow Matching can be seen as a certain limiting case of bridge matching.

The rest of this manuscript is organized as follows. Section 2 offers a self-contained "cheat-sheet" to understand and implement vanilla Flow Matching in PyTorch. Section 3 offers a rigorous treatment of flow models, arguably the simplest of all CTMPs, for continuous state spaces. In section 4 we introduce the Flow Matching framework in $\mathbb{R}^d$ and its various design choices and extensions. We show that flows can be constructed by considering the significantly simpler conditional setting, offering great deal of flexibility in their design, *e.g.*, by readily extending to Riemannian geometries, described in section 5. Section 6 provides an introduction to Continuous Time Markov Chains (CTMCs) and the usage as generative models on discrete state spaces. Then, section 7 discusses the extension of Flow Matching to CTMC processes. In section 8, we provide an introduction to using Continuous Time Markov Process (CTMPs) as generative models for arbitrary state spaces. In section 9, we describe Generator Matching (GM) - a generative modeling framework for arbitrary modalities that describes a scalable way of training CTMPs. GM also unifies all models in previous sections into a common framework. Finally, due to their wide-spread use, we discuss in section 10 denoising diffusion models as a specific instance of the FM family of models.

2 Quick tour and key concepts

Given access to a training dataset of samples from some target distribution q over $\mathbb{R}^d$, our goal is to build a model capable of generating new samples from q. To address this task, **Flow Matching (FM)** builds a **probability path** $(p_t)_{0\leq t\leq 1}$, from a known source distribution $p_0 = p$ to the data target distribution $p_1 = q$, where each p_t is a distribution over $\mathbb{R}^d$. Specifically, FM is a simple regression objective to train the **velocity field** neural network describing the instantaneous velocities of samples—later used to convert the source distribution p_0 into the target distribution p_1, along the probability path p_t. After training, we generate a novel sample from the target distribution $X_1 \sim q$ by (i) drawing a novel sample from the source distribution $X_0 \sim p$, and (ii) solving the Ordinary Differential Equation (ODE) determined by the velocity field.

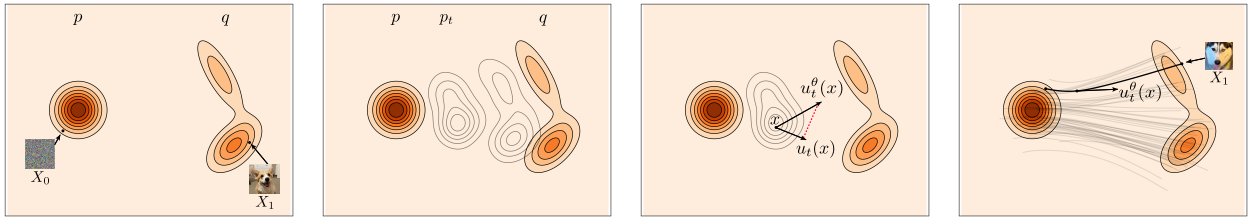
More formally, an ODE is defined via a time-dependent vector field $u : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ which, in our case, is the velocity field modeled in terms of a neural network. This velocity field determines a time-dependent **flow** $\psi : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$, defined as

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x) = u_t(\psi_t(x)),$$

where $\psi_t := \psi(t, x)$ and $\psi_0(x) = x$. The velocity field u_t *generates* the probability path p_t if its flow ψ_t satisfies

$$X_t := \psi_t(X_0) \sim p_t \text{ for } X_0 \sim p_0. \tag{2.1}$$

According to the equation above, the velocity field u_t is the only tool necessary to sample from p_t by solving the ODE above. As illustrated in figure 2d, solving the ODE until $t = 1$ provides us with samples $X_1 = \psi_1(X_0)$, resembling the target distribution q. Therefore, and in sum, the goal of Flow Matching is to learn a vector field u_t^θ such that its flow ψ_t generates a probability path p_t with $p_0 = p$ and $p_1 = q$.



(a) Data. **(b)** Path design. **(c)** Training. **(d)** Sampling.

Figure 2 *The Flow Matching blueprint.* (a) The goal is to find a flow mapping samples X_0 from a known source or noise distribution q into samples X_1 from an unknown target or data distribution q. (b) To do so, design a time-continuous probability path $(p_t)_{0\leq t\leq 1}$ interpolating between $p := p_0$ and $q := p_1$. (c) During training, use regression to estimate the velocity field u_t known to generate p_t. (d) To draw a novel target sample $X_1 \sim q$, integrate the estimated velocity field $u_t^\theta(X_t)$ from $t = 0$ to $t = 1$, where $X_0 \sim p$ is a novel source sample.

Using the notations above, the goal of Flow Matching is to learn the parameters θ of a velocity field u_t^θ implemented in terms of a neural network. As anticipated in the introduction, we do this in two steps: design a probability path p_t interpolating between p and q (see figure 2b), and train a velocity field u_t^θ generating p_t by means of regression (see figure 2c).

Therefore, let us proceed with the first step of the recipe: designing the probability path p_t. In this example, let the source distribution $p := p_0 = \mathcal{N}(x|0, I)$, and construct the probability path p_t as the aggregation of the conditional probability paths $p_{t|1}(x|x_1)$, each conditioned on one of the data examples $X_1 = x_1$ comprising the training dataset. (One of such conditional paths is illustrated in figure 3a.) The probability path p_t therefore follows the expression:

$$p_t(x) = \int p_{t|1}(x|x_1)q(x_1)\mathrm{d}x_1, \text{ where } p_{t|1}(x|x_1) = \mathcal{N}(x|tx_1, (1-t)^2 I). \tag{2.2}$$

This path, also known as the *conditional optimal-transport* or *linear* path, enjoys some desirable properties that we will study later in this manuscript. Using this probability path, we may define the random variable $X_t \sim p_t$ by drawing $X_0 \sim p$, drawing $X_1 \sim q$, and taking their linear combination:

$$X_t = tX_1 + (1-t)X_0 \sim p_t. \tag{2.3}$$

We now continue with the second step in the Flow Matching recipe: regressing our velocity field u_t^θ (usually implemented in terms of a neural network) to a target velocity field u_t known to generate the desired probability path p_t. To this end, the **Flow Matching loss** reads:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t,X_t}\left\|u_t^\theta(X_t) - u_t(X_t)\right\|^2, \text{ where } t \sim \mathcal{U}[0,1] \text{ and } X_t \sim p_t. \tag{2.4}$$

In practice, one can rarely implement the objective above, because u_t is a complicated object governing the *joint* transformation between two high-dimensional distributions. Fortunately, the objective simplifies drastically by conditioning the loss on a single target example $X_1 = x_1$ picked at random from the training set. To see how, borrowing equation (2.3) to realize the conditional random variables

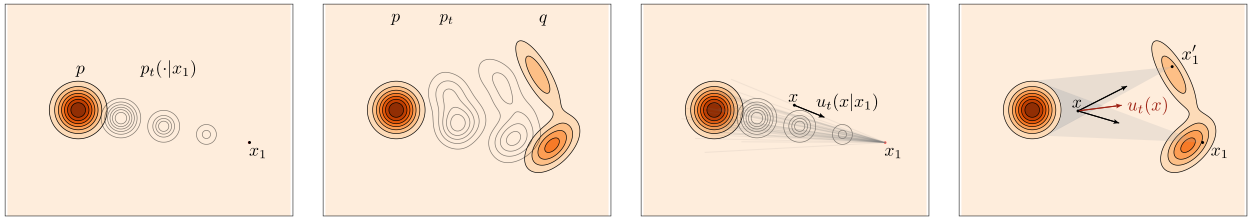
$$X_{t|1} = tx_1 + (1-t)X_0 \quad \sim \quad p_{t|1}(\cdot|x_1) = \mathcal{N}(\cdot \mid tx_1, (1-t)^2 I). \tag{2.5}$$

Using these variables, solving $\frac{\mathrm{d}}{\mathrm{d}t}X_{t|1} = u_t(X_{t|1}|x_1)$ leads to the **conditional velocity field**

$$u_t(x|x_1) = \frac{x_1 - x}{1-t}, \tag{2.6}$$

which generates the conditional probability path $p_{t|1}(\cdot|x_1)$. (For an illustration on these two conditional objects, see figure 3c.) Equipped with the simple equation (2.6) for the conditional velocity fields generating the designed conditional probability paths, we can formulate a tractable version of the Flow Matching loss in (2.4). This is the **conditional Flow Matching loss**:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,X_t,X_1}\|u_t^\theta(X_t) - u_t(X_t|X_1)\|^2, \text{ where } t \sim U[0,1],\ X_0 \sim p,\ X_1 \sim q, \tag{2.7}$$



(a) Conditional probability path $p_t(x|x_1)$. **(b)** (Marginal) Probability path $p_t(x)$. **(c)** Conditional velocity field $u_t(x|x_1)$. **(d)** (Marginal) Velocity field $u_t(x)$.

Figure 3 *Path design in Flow Matching.* Given a fixed target sample $X = x_1$, its conditional velocity field $u_t(x|x_1)$ generates the conditional probability path $p_t(x|x_1)$. The (marginal) velocity field $u_t(x)$ results from the aggregation of all conditional velocity fields—and similarly for the probability path $p_t(x)$.

and $X_t = (1-t)X_0 + tX_1$. Remarkably, the objectives in (2.4) and (2.7) provide the same gradients to learn u_t^θ, *i.e.*,

$$\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) = \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta). \tag{2.8}$$

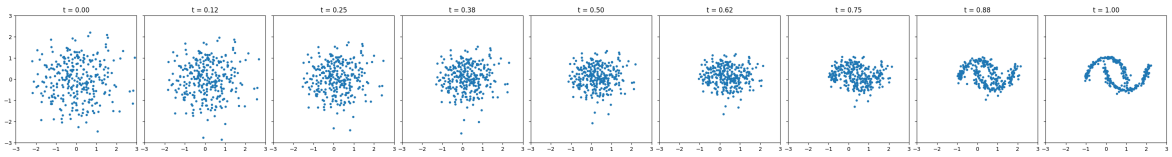
Finally, by plugging $u_t(x|x_1)$ from (2.6) into equation (2.7), we get the simplest implementation of Flow Matching:

$$\mathcal{L}_{\text{CFM}}^{\text{OT,Gauss}}(\theta) = \mathbb{E}_{t,X_0,X_1} \| u_t^\theta(X_t) - (X_1 - X_0) \|^2, \text{ where } t \sim U[0,1],\ X_0 \sim \mathcal{N}(0, I),\ X_1 \sim q. \tag{2.9}$$

A standalone implementation of this quick tour in pure PyTorch is provided in code 1. Later in the manuscript, we will cover more sophisticated variants and design choices, all of them implemented in the accompanying `flow_matching` library.

Code 1: Standalone Flow Matching code
flow_matching/examples/standalone_flow_matching.ipynb

```python
import torch
from torch import nn, Tensor
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons

class Flow(nn.Module):
  def __init__(self, dim: int = 2, h: int = 64):
    super().__init__()
    self.net = nn.Sequential(
      nn.Linear(dim + 1, h), nn.ELU(),
      nn.Linear(h, h), nn.ELU(),
      nn.Linear(h, h), nn.ELU(),
      nn.Linear(h, dim))

  def forward(self, x_t: Tensor, t: Tensor) -> Tensor:
    return self.net(torch.cat((t, x_t), -1))

  def step(self, x_t: Tensor, t_start: Tensor, t_end: Tensor) -> Tensor:
    t_start = t_start.view(1, 1).expand(x_t.shape[0], 1)
    # For simplicity, using midpoint ODE solver in this example
    return x_t + (t_end - t_start) * self(x_t + self(x_t, t_start) * (t_end - t_start) / 2,
                                          t_start + (t_end - t_start) / 2)

# training
flow = Flow()
optimizer = torch.optim.Adam(flow.parameters(), 1e-2)
loss_fn = nn.MSELoss()

for _ in range(10000):
  x_1  = Tensor(make_moons(256, noise=0.05)[0])
  x_0  = torch.randn_like(x_1)
  t    = torch.rand(len(x_1), 1)
  x_t  = (1 - t) * x_0 + t * x_1
  dx_t = x_1 - x_0
  optimizer.zero_grad()
  loss_fn(flow(x_t, t), dx_t).backward()
  optimizer.step()

# sampling
x = torch.randn(300, 2)
n_steps = 8
fig, axes = plt.subplots(1, n_steps + 1, figsize=(30, 4), sharex=True, sharey=True)
time_steps = torch.linspace(0, 1.0, n_steps + 1)

axes[0].scatter(x.detach()[:, 0], x.detach()[:, 1], s=10)
axes[0].set_title(f't = {time_steps[0]:.2f}')
axes[0].set_xlim(-3.0, 3.0)
axes[0].set_ylim(-3.0, 3.0)

for i in range(n_steps):
  x = flow.step(x, time_steps[i], time_steps[i + 1])
  axes[i + 1].scatter(x.detach()[:, 0], x.detach()[:, 1], s=10)
  axes[i + 1].set_title(f't = {time_steps[i + 1]:.2f}')

plt.tight_layout()
plt.show()
```



3 Flow models

This section introduces **flows**, the mathematical object powering the simplest forms of Flow Matching. Later parts in the manuscript will discuss Markov processes more general than flows, leading to more sophisticated generative learning paradigms introducing many more design choices to the Flow Matching framework. The reason we start with flows is three-fold: First, flows are arguably the simplest of all CTMPs—being deterministic and having a compact parametrization via velocities—these models can transform any source distribution p into any target distribution q, as long as these two have densities. Second, flows can be sampled rather efficiently by approximating the solution of ODEs, compared, *e.g.*, to the harder-to-simulate SDEs for diffusion processes. Third, the deterministic nature of flows allows an unbiased model likelihood estimation, while more general stochastic processes require working with lower bounds. To understand flows, we must first review some background notions in probability and differential equations theory, which we do next.

3.1 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

$$\mathbb{P}(X \in A) = \int_A p_X(x) \mathrm{d}x, \tag{3.1}$$

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:

$$\mathcal{N}(x|\mu, \sigma^2 I) = (2\pi\sigma^2)^{-\frac{d}{2}} \exp\left(-\frac{\|x-\mu\|_2^2}{2\sigma^2}\right), \tag{3.2}$$

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:

$$\mathbb{E}[X] = \operatorname*{arg\,min}_{z \in \mathbb{R}^d} \int \|x - z\|^2 p_X(x) \mathrm{d}x = \int x p_X(x) \mathrm{d}x. \tag{3.3}$$

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

$$\mathbb{E}[f(X)] = \int f(x) p_X(x) \mathrm{d}x. \tag{3.4}$$

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

3.2 Conditional densities and expectations

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

$$\int p_{X,Y}(x, y) \mathrm{d}y = p_X(x) \text{ and } \int p_{X,Y}(x, y) \mathrm{d}x = p_Y(y). \tag{3.5}$$

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 $p_Y(y) > 0$:

$$p_{X|Y}(x|y) \coloneqq \frac{p_{X,Y}(x, y)}{p_Y(y)}, \tag{3.6}$$

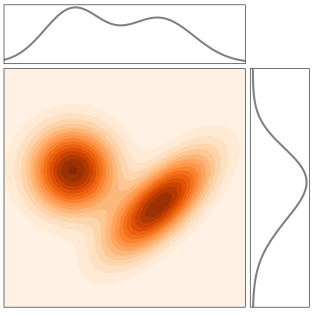


Figure 4 Joint PDF $p_{X,Y}$ (in shades) and its marginals p_X and p_Y (in black lines).

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

$$p_{Y|X}(y|x) = \frac{p_{X|Y}(x|y)p_Y(y)}{p_X(x)}, \tag{3.7}$$

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:

$$\begin{aligned} g_\star &\coloneqq \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \mathbb{E}\left[\|X - g(Y)\|^2\right] = \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \|x - g(y)\|^2 p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \operatorname*{arg\,min}_{g:\mathbb{R}^d\to\mathbb{R}^d} \int \left[\int \|x - g(y)\|^2 p_{X|Y}(x|y)\mathrm{d}x\right] p_Y(y)\mathrm{d}y. \end{aligned} \tag{3.8}$$

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

$$\mathbb{E}[X|Y = y] \coloneqq g_\star(y) = \int x p_{X|Y}(x|y)\mathrm{d}x, \tag{3.9}$$

where the second equality follows from taking the minimizer of the inner brackets in equation (3.8) for $Y = y$, similarly to equation (3.3). Composing $g_\star$ with the random variable Y, we get

$$\mathbb{E}[X|Y] \coloneqq g_\star(Y), \tag{3.10}$$

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:

$$\mathbb{E}[\mathbb{E}[X|Y]] = \mathbb{E}[X] \tag{3.11}$$

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:

$$\begin{aligned} \mathbb{E}[\mathbb{E}[X|Y]] &= \int \left(\int x p_{X|Y}(x|y)\mathrm{d}x\right) p_Y(y)\mathrm{d}y \\ &\overset{(3.6)}{=} \int\int x p_{X,Y}(x,y)\mathrm{d}x\mathrm{d}y \\ &\overset{(3.5)}{=} \int x p_X(x)\mathrm{d}x \\ &= \mathbb{E}[X]. \end{aligned}$$

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 (3.9), we obtain the identity

$$\mathbb{E}[f(X,Y)|Y = y] = \int f(x,y) p_{X|Y}(x|y)\mathrm{d}x. \tag{3.12}$$

3.3 Diffeomorphisms and push-forward maps

We denote by $C^r(\mathbb{R}^m, \mathbb{R}^n)$ the collection of functions $f : \mathbb{R}^m \to \mathbb{R}^n$ with continuous partial derivatives of order r:

$$\frac{\partial^r f^k}{\partial x^{i_1} \cdots \partial x^{i_r}}, \qquad k \in [n], i_j \in [m], \tag{3.13}$$

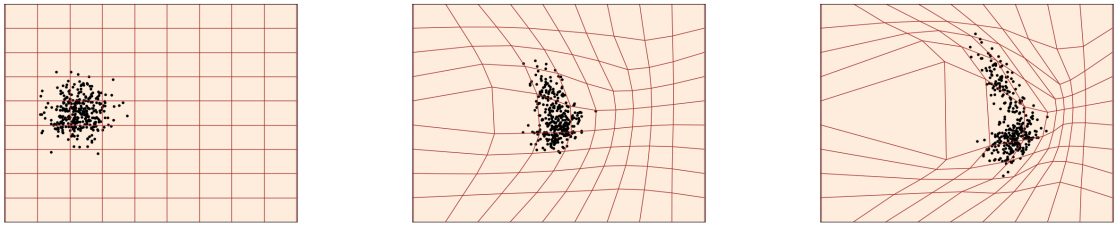


Figure 5 A flow model $X_t = \psi_t(X_0)$ is defined by a diffeomorphism $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with a brown square grid) pushing samples from a source RV X_0 (left, black points) toward some target distribution q (right). We show three different times t.

where $[n] \coloneqq \{1, 2, \ldots, n\}$. To keep notation concise, define also $C^r(\mathbb{R}^n) \coloneqq C^r(\mathbb{R}^m, \mathbb{R})$ so, for example, $C^1(\mathbb{R}^m)$ denotes the continuously differentiable scalar functions. An important class of functions are the C^r **diffeomorphism**; these are invertible functions $\psi \in C^r(\mathbb{R}^n, \mathbb{R}^n)$ with $\psi^{-1} \in C^r(\mathbb{R}^n, \mathbb{R}^n)$.

Then, given a RV $X \sim p_X$ with density p_X, let us consider a RV $Y = \psi(X)$, where $\psi : \mathbb{R}^d \to \mathbb{R}^d$ is a C^1 diffeomorphism. The PDF of Y, denoted p_Y, is also called the *push-forward* of p_X. Then, the PDF p_Y can be computed via a change of variables:

$$\mathbb{E}\left[f(Y)\right] = \mathbb{E}\left[f(\psi(X))\right] = \int f(\psi(x))p_X(x)\mathrm{d}x = \int f(y)p_X(\psi^{-1}(y))\left|\det \partial_y \psi^{-1}(y)\right|\mathrm{d}y,$$

where the third equality is due the change of variables $x = \psi^{-1}(y)$, $\partial_y \phi(y)$ denotes the Jacobian matrix (of first order partial derivatives), *i.e.*,

$$\left[\partial_y \phi(y)\right]_{i,j} = \frac{\partial \phi^i}{\partial x^j}, \;\; i, j \in [d],$$

and $\det A$ denotes the determinant of a square matrix $A \in \mathbb{R}^{d \times d}$. Thus, we conclude that the PDF p_Y is

$$p_Y(y) = p_X(\psi^{-1}(y))\left|\det \partial_y \psi^{-1}(y)\right|. \tag{3.14}$$

We will denote the push-forward operator with the symbol $\sharp$, that is

$$\left[\psi_\sharp p_X\right](y) \coloneqq p_X(\psi^{-1}(y))\left|\det \partial_y \psi^{-1}(y)\right|. \tag{3.15}$$

3.4 Flows as generative models

As mentioned in section 2, the goal of generative modeling is to transform samples $X_0 = x_0$ from a **source distribution** p into samples $X_1 = x_1$ from a **target distribution** q. In this section, we start building the tools necessary to address this problem by means of a flow mapping ψ_t. More formally, a C^r **flow** is a time-dependent mapping $\psi : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ implementing $\psi : (t, x) \mapsto \psi_t(x)$. Such flow is also a $C^r([0, 1] \times \mathbb{R}^d, \mathbb{R}^d)$ function, such that the function $\psi_t(x)$ is a C^r diffeomorphism in x for all $t \in [0, 1]$. A **flow model** is a *continuous-time Markov process* $(X_t)_{0 \le t \le 1}$ defined by applying a flow ψ_t to the RV X_0:

$$X_t = \psi_t(X_0), \quad t \in [0, 1], \text{ where } X_0 \sim p. \tag{3.16}$$

See Figure 5 for an illustration of a flow model. To see why X_t is Markov, note that, for any choice of $0 \le t < s \le 1$, we have

$$X_s = \psi_s(X_0) = \psi_s(\psi_t^{-1}(\psi_t(X_0))) = \psi_{s|t}(X_t), \tag{3.17}$$

where the last equality follows from using equation (3.16) to set $X_t = \psi_t(X_0)$, and defining $\psi_{s|t} \coloneqq \psi_s \circ \psi_t^{-1}$, which is also a diffeomorphism. $X_s = \psi_{s|t}(X_t)$ implies that states later than X_t depend only on X_t, so X_t is Markov. In fact, for flow models, this dependence is *deterministic*.

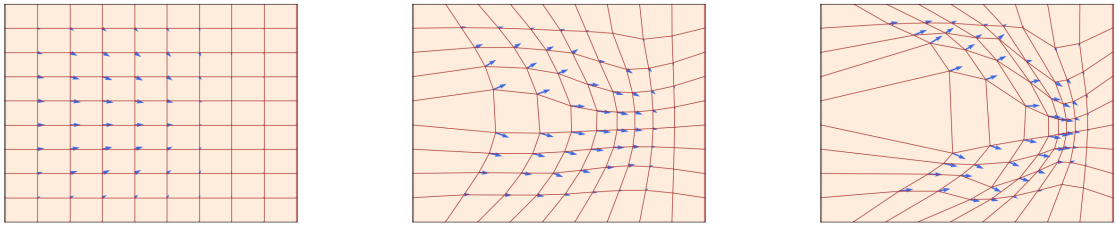


Figure 6 A flow $\psi_t : \mathbb{R}^d \to \mathbb{R}^d$ (square grid) is defined by a velocity field $u_t : \mathbb{R}^d \to \mathbb{R}^d$ (visualized with blue arrows) that prescribes its instantaneous movements at all locations. We show three different times t.

In summary, the goal **generative flow modeling** is to find a flow ψ_t such that

$$X_1 = \psi_1(X_0) \sim q. \tag{3.18}$$

3.4.1 Equivalence between flows and velocity fields

A C^r flow ψ can be defined in terms of a $C^r([0,1] \times \mathbb{R}^d, \mathbb{R}^d)$ *velocity field* $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ implementing $u : (t, x) \mapsto u_t(x)$ via the following ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x) = u_t(\psi_t(x)) \qquad \text{(flow ODE)} \tag{3.19a}$$

$$\psi_0(x) = x \qquad \text{(flow initial conditions)} \tag{3.19b}$$

See figure 6 for an illustration of a flow together with its velocity field.

A standard result regarding the existence and uniqueness of solutions $\psi_t(x)$ to equation (3.19) is (see *e.g.*, Perko (2013); Coddington et al. (1956)):

Theorem 1 (Flow local existence and uniqueness)**.** *If u is $C^r([0,1] \times \mathbb{R}^d, \mathbb{R}^d)$, $r \geq 1$ (in particular, locally Lipschitz), then the ODE in (3.19) has a unique solution which is a $C^r(\Omega, \mathbb{R}^d)$ diffeomorphism $\psi_t(x)$ defined over an open set Ω which is super-set of $\{0\} \times \mathbb{R}^d$.*

This theorem guarantees only the *local* existence and uniqueness of a C^r flow moving each point $x \in \mathbb{R}^d$ by $\psi_t(x)$ during a potentially limited amount of time $t \in [0, t_x)$. To guarantee a solution until $t = 1$ for all $x \in \mathbb{R}^d$, one must place additional assumptions beyond local Lipschitzness. For instance, one could consider global Lipschitness, guaranteed by bounded first derivatives in the C^1 case. However, we will later rely on a different condition—namely, integrability—to guarantee the existence of the flow almost everywhere, and until time $t = 1$.

So far, we have shown that a velocity field uniquely defines a flow. Conversely, given a C^1 flow ψ_t, one can extract its defining velocity field $u_t(x)$ for arbitrary $x \in \mathbb{R}^d$ by considering the equation $\frac{\mathrm{d}}{\mathrm{d}t}\psi_t(x') = u_t(\psi_t(x'))$, and using the fact that ψ_t is an invertible diffeomorphism for every $t \in [0,1]$ to let $x' = \psi_t^{-1}(x)$. Therefore, the unique velocity field u_t determining the flow ψ_t is

$$u_t(x) = \dot{\psi}_t(\psi_t^{-1}(x)), \tag{3.20}$$

where $\dot{\psi}_t \coloneqq \frac{\mathrm{d}}{\mathrm{d}t}\psi_t$. In conclusion, we have shown the equivalence between C^r flows ψ_t and C^r velocity fields u_t.

3.4.2 Computing target samples from source samples

Computing a target sample X_1—or, in general, any sample X_t—entails approximating the solution of the ODE in equation (3.19) starting from some initial condition $X_0 = x_0$. Numerical methods for ODEs is a classical and well researched topic in numerical analysis, and a myriad of powerful methods exist (Iserles, 2009). One of the simplest methods is the *Euler method*, implementing the update rule

$$X_{t+h} = X_t + h u_t(X_t) \tag{3.21}$$

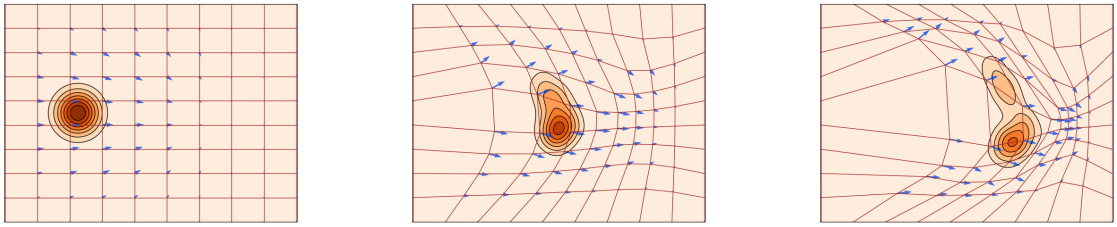


Figure 7 A velocity field u_t (in blue) *generates* a probability path p_t (PDFs shown as contours) if the flow defined by u_t (square grid) reshapes p (left) to p_t at all times $t \in [0, 1)$.

where $h = n^{-1} > 0$ is a step size hyper-parameter with $n \in \mathbb{N}$. To draw a sample X_1 from the target distribution, apply the Euler method starting at some $X_0 \sim p$ to produce the sequence $X_h, X_{2h}, \ldots, X_1$. The Euler method coincides with first-order Taylor expansion of X_t:

$$X_{t+h} = X_t + h\dot{X}_t + o(h) = X_t + hu_t(X_t) + o(h),$$

where $o(h)$ stands for a function growing slower than h, that is, $o(h)/h \to 0$ as $h \to 0$. Therefore, the Euler method accumulates $o(h)$ error per step, and can be shown to accumulate $o(1)$ error after $n = 1/h$ steps. Therefore, the error of the Euler method vanishes as we consider smaller step sizes $h \to 0$. The Euler method is just one example among many ODE solvers. Code 2 exemplifies another alternative, the second-order *midpoint method*, which often outperforms the Euler method in practice.

Code 2: Computing X_1 with Midpoint solver

```python
from flow_matching.solver import ODESolver
from flow_matching.utils import ModelWrapper

class Flow(ModelWrapper):
    def __init__(self, dim=2, h=64):
        super().__init__()
        self.net = torch.nn.Sequential(
            torch.nn.Linear(dim + 1, h), torch.nn.ELU(),
            torch.nn.Linear(h, dim))

    def forward(self, x, t):
        t = t.view(-1, 1).expand(*x.shape[:-1], -1)
        return self.net(torch.cat((t, x), -1))

velocity_model = Flow()

... # Optimize the model parameters s.t. model(x_t, t) = u_t(X_t)

x_0 = torch.randn(batch_size, *data_dim)  # Specify the initial condition

solver = ODESolver(velocity_model=velocity_model)
num_steps = 100
x_1 = solver.sample(x_init=x_0, method='midpoint', step_size=1.0 / num_steps)
```

3.5 Probability paths and the Continuity Equation

We call a time-dependent probability $(p_t)_{0\leq t\leq 1}$ a **probability path**. For our purposes, one important probability path is the marginal PDF of a flow model $X_t = \psi_t(X_0)$ at time t:

$$X_t \sim p_t. \tag{3.22}$$

For each time $t \in [0, 1]$, these marginal PDFs are obtained via the push-forward formula in equation (3.15), that is,

$$p_t(x) = [\psi_{t\sharp}p](x). \tag{3.23}$$

Given some arbitrary probability path p_t we define

$$u_t \textbf{ generates } p_t \text{ if } X_t = \psi_t(X_0) \sim p_t \text{ for all } t \in [0, 1). \tag{3.24}$$

In this way, we establish a close relationship between velocity fields, their flows, and the generated probability paths, see Figure 7 for an illustration. Note that we use the time interval $[0, 1)$, open from the right, to allow dealing with target distributions q with compact support where the velocity is not defined precisely at $t = 1$.

To verify that a velocity field u_t generates a probability path p_t, one can verify if the pair (u_t, p_t) satisfies a partial differential equation (PDE) known as the *Continuity Equation*:

$$\frac{\mathrm{d}}{\mathrm{d}t}p_t(x) + \operatorname{div}(p_t u_t)(x) = 0, \tag{3.25}$$

where $\operatorname{div}(v)(x) = \sum_{i=1}^{d} \partial_{x^i} v^i(x)$, and $v(x) = (v^1(x), \dots, v^d(x))$. The following theorem, a rephrased version of the *Mass Conservation Formula* (Villani et al., 2009), states that a solution u_t to the Continuity Equation generates the probability path p_t:

Theorem 2 (Mass Conservation)**.** *Let p_t be a probability path and u_t a locally Lipchitz integrable vector field. Then, the following two statements are equivalent:*

1. *The Continuity Equation (3.25) holds for $t \in [0, 1)$.*
2. *u_t generates p_t, in the sense of (3.24).*

In the previous theorem, local Lipschitzness assumes that there exists a local neighbourhood over which $u_t(x)$ is Lipschitz, for all (t, x). Assuming that u is integrable means that:

$$\int_0^1 \int \|u_t(x)\| \, p_t(x)\mathrm{d}x\mathrm{d}t < \infty. \tag{3.26}$$

Specifically, integrating a solution to the flow ODE (3.19a) across times $[0, t]$ leads to the integral equation

$$\psi_t(x) = x + \int_0^t u_s(\psi_s(x))\mathrm{d}s. \tag{3.27}$$

Therefore, integrability implies

$$\begin{aligned}
\mathbb{E}\,\|X_t\| &\overset{(3.16)}{=} \int \|\psi_t(x)\| \, p(x)\mathrm{d}x \\
&= \int \left\| x + \int_0^t u_s(\psi_s(x))ds \right\| p(x)\mathrm{d}x \\
&\overset{(i)}{\leq} \mathbb{E}\,\|X_0\| + \int_0^1 \int \|u_s(x)\| \, p_t(x)\mathrm{d}t \\
&\overset{(ii)}{<} \infty,
\end{aligned}$$

where (i) follows from the triangle inequality, and (ii) assumes the integrability condition (3.26) and $\mathbb{E}\,\|X_0\| < \infty$. In sum, integrability allows assuming that X_t has bounded expected norm, if X_0 also does.

To gain further insights about the meaning of the Continuity Equation, we may write it in *integral form* by means of the Divergence Theorem—see Matthews (2012) for an intuitive exposition, and Loomis and Sternberg (1968) for a rigorous treatment. This result states that, considering some domain $\mathcal{D}$ and some smooth vector field $u : \mathbb{R}^d \to \mathbb{R}^d$, accumulating the divergences of u inside $\mathcal{D}$ equals the *flux* leaving $\mathcal{D}$ by orthogonally crossing its boundary $\partial\mathcal{D}$:

$$\int_{\mathcal{D}} \operatorname{div}(u)(x)\mathrm{d}x = \int_{\partial\mathcal{D}} \langle u(y), n(y)\rangle \, \mathrm{d}s_y, \tag{3.28}$$

where $n(y)$ is a unit-norm normal field pointing outward to the domain's boundary $\partial\mathcal{D}$, and $\mathrm{d}s_y$ is the boundary's area element. To apply these insights to the Continuity Equation, let us integrate (3.25) over a small domain $\mathcal{D} \subset \mathbb{R}^d$ (for instance, a cube) and apply the Divergence Theorem to obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_{\mathcal{D}} p_t(x)\mathrm{d}x = -\int_{\mathcal{D}} \operatorname{div}(p_t u_t)(x)\mathrm{d}x = -\int_{\partial\mathcal{D}} \langle p_t(y)u_t(y), n(y)\rangle \, \mathrm{d}s_y. \tag{3.29}$$

This equation expresses the rate of change of total probability mass in the volume $\mathcal{D}$ (left-hand side) as the negative probability *flux* leaving the domain (right-hand side). The probability flux, defined as $j_t(y) = p_t(y)u_t(y)$, is the probability mass flowing through the hyperplane orthogonal to $n(y)$ per unit of time and per unit of (possibly high-dimensional) area. See figure 8 for an illustration.

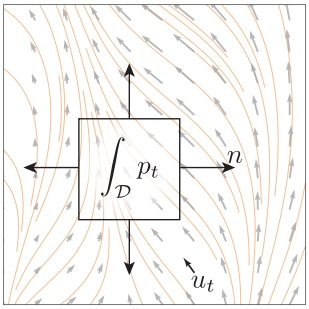


Figure 8 The continuity equation asserts that the local change in probability equals minus the net outgoing probability flux.

3.6 Instantaneous Change of Variables

One important benefit of using flows as generative models is that they allow the tractable computation of *exact* likelihoods $\log p_1(x)$, for all $x \in \mathbb{R}^d$. This feature is a consequence of the Continuity Equation called the *Instantaneous Change of Variables* (Chen et al., 2018):

$$\frac{\mathrm{d}}{\mathrm{d}t}\log p_t(\psi_t(x)) = -\operatorname{div}(u_t)(\psi_t(x)). \tag{3.30}$$

This is the ODE governing the change in log-likelihood, $\log p_t(\psi_t(x))$, along a sampling trajectory $\psi_t(x)$ defined by the flow ODE (3.19a). To derive (3.30), differentiate $\log p_t(\psi_t(x))$ with respect to time, and apply both the Continuity Equation (3.25) and the flow ODE (3.19a). Integrating (3.30) from $t = 0$ to $t = 1$ and rearranging, we obtain

$$\log p_1(\psi_1(x)) = \log p_0(\psi_0(x)) - \int_0^1 \operatorname{div}(u_t)(\psi_t(x))\mathrm{d}t. \tag{3.31}$$

In practice, computing $\operatorname{div}(u_t)$, which equals the trace of the Jacobian matrix $\partial_x u_t(x) \in \mathbb{R}^{d\times d}$, is increasingly challenge as the dimensionality d grows. Because of this reason, previous works employ unbiased estimators such as Hutchinson's trace estimator (Grathwohl et al., 2018):

$$\operatorname{div}(u_t)(x) = \operatorname{tr}\left[\partial_x u_t(x)\right] = \mathbb{E}_Z \operatorname{tr}\left[Z^T \partial_x u_t(x) Z\right], \tag{3.32}$$

where $Z \in \mathbb{R}^{d\times d}$ is any random variable with $\mathbb{E}[Z] = 0$ and $\operatorname{Cov}(Z, Z) = I$, (for example, $Z \sim \mathcal{N}(0, I)$), and $\operatorname{tr}[Z] = \sum_{i=1}^d Z_{i,i}$. By plugging the equation above into (3.31) and switching the order of integral and expectation, we obtain the following unbiased log-likelihood estimator:

$$\log p_1(\psi_1(x)) = \log p_0(\psi_0(x)) - \mathbb{E}_Z \int_0^1 \operatorname{tr}\left[Z^T \partial_x u_t(\psi_t(x)) Z\right] \mathrm{d}t. \tag{3.33}$$

In contrast to $\operatorname{div}(u_t)(\psi_t(x))$ in (3.30), computing $\operatorname{tr}\left[Z^T \partial_x u_t(\psi_t(x)) Z\right]$ for a fixed sample Z in the equation above can be done with a single backward pass via a vector-Jacobian product (JVP)[1].

[1] E.g., see https://pytorch.org/docs/stable/generated/torch.autograd.functional.vjp.html.

In summary, computing an unbiased estimate of $\log p_1(x)$ entails simulating the ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} f(t) \\ g(t) \end{bmatrix} = \begin{bmatrix} u_t(f(t)) \\ -\mathrm{tr}\left[Z^T \partial_x u_t(f(t)) Z\right] \end{bmatrix}, \tag{3.34a}$$

$$\begin{bmatrix} f(1) \\ g(1) \end{bmatrix} = \begin{bmatrix} x \\ 0 \end{bmatrix}, \tag{3.34b}$$

backwards in time, from $t = 1$ to $t = 0$, and setting:

$$\widehat{\log p_1}(x) = \log p_0(f(0)) - g(0). \tag{3.35}$$

See code 3 for an example on how to obtain log-likelihood estimates from a flow model using the `flow_matching` library.

Code 3: Computing the likelihood

```python
from flow_matching.solver import ODESolver
from flow_matching.utils import ModelWrapper
from torch.distributions.normal import Normal

velocity_model: ModelWrapper = ...  # Train the model parameters s.t. model(x_t, t) = u_t(x_t)

x_1 = torch.randn(batch_size, *data_dim)  # Point X_1 where we wish to compute log p_1(x)

# Define log p_0(x)
gaussian_log_density = Normal(torch.zeros(size=data_dim), torch.ones(size=data_dim)).log_prob

solver = ODESolver(velocity_model=velocity_model)
num_steps = 100
x_0, log_p1 = solver.compute_likelihood(
    x_1=x_1,
    method='midpoint',
    step_size=1.0 / num_steps,
    log_p0=gaussian_log_density
)
```

3.7 Training flow models with simulation

The Instantaneous Change of Variables, and the resulting ODE system (3.34), allows training a flow model by maximizing the log-likelihood of training data (Chen et al., 2018; Grathwohl et al., 2018). Specifically, let u_t^θ be a velocity field with learnable parameters $\theta \in \mathbb{R}^p$, and consider the problem of learning θ such that

$$p_1^\theta \approx q. \tag{3.36}$$

We can pursue this goal, for instance, by minimizing the KL-divergence of p_1^θ and q:

$$\mathcal{L}(\theta) = D_{\mathrm{KL}}(q, p_1^\theta) = -\mathbb{E}_{Y\sim q} \log p_1^\theta(Y) + \text{constant}, \tag{3.37}$$

where p_1^θ is the distribution of $X_1 = \psi_1^\theta(X_0)$, ψ_t^θ is defined by u_t^θ, and we can obtain an unbiased estimate of $\log p_1^\theta(Y)$ via the solution to the ODE system (3.34). However, computing this loss—as well as its gradients—requires precise ODE simulations during training, where only errorless solutions constitute unbiased gradients. In contrast, *Flow Matching*, presented next, is a simulation-free framework to train flow generative models without the need of solving ODEs during training.

4 Flow Matching

Given a source distribution p and a target distribution q, Flow Matching (FM) (Lipman et al., 2022; Liu et al., 2022; Albergo and Vanden-Eijnden, 2022) is a scalable approach for training a flow model, defined by a learnable velocity u_t^θ, and solving the **Flow Matching Problem**:

$$\text{Find } u_t^\theta \text{ generating } p_t, \text{ with } p_0 = p \text{ and } p_1 = q. \tag{4.1}$$

In the equation above, "generating" is in the sense of equation (3.24). Revisiting the Flow Matching *blueprint* from figure 2, the FM framework (a) identifies a known source distribution p and an unknown data target distribution q, (b) prescribes a probability path p_t interpolating from $p_0 = p$ to $p_1 = q$, (c) learns a velocity field u_t^θ implemented in terms of a neural network and generating the path p_t, and (d) samples from the learned model by solving an ODE with u_t^θ. To learn the velocity field u_t^θ in step (c), FM minimizes the regression loss:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{X_t \sim p_t} D\left(u_t(X_t), u_t^\theta(X_t)\right), \tag{4.2}$$

where D is a dissimilarity measure between vectors, such as the squared ℓ_2-norm $D(u, v) = \|u - v\|^2$. Intuitively, the FM loss encourages our learnable velocity field u_t^θ to match the ground truth velocity field u_t known to generate the desired probability path p_t. Figure 9 depicts the main objects in the Flow Matching framework and their dependencies. Let us start our exposition of Flow Matching by describing how to build p_t and u_t, as well as a practical implementation of the loss (4.2).

4.1 Data

To reiterate, let source samples be a RV $X_0 \sim p$ and target samples a RV $X_1 \sim q$. Commonly, source samples follow a known distribution that is easy to sample, and target samples are given to us in terms of a dataset of finite size. Depending on the application, target samples may constitute images, videos, audio segments, or other types of high-dimensional, richly structured data. Source and target samples can be independent, or originate from a general joint distribution known as the **coupling**

$$(X_0, X_1) \sim \pi_{0,1}(X_0, X_1), \tag{4.3}$$

where, if no coupling is known, the source-target samples are following the independent coupling $\pi_{0,1}(X_0, X_1) = p(X_0)q(X_1)$. One common example of independent source-target distributions is to consider the generation of images X_1 from random Gaussian noise vectors $X_0 \sim \mathcal{N}(0, I)$. As an example of a dependent coupling, consider the case of producing high-resolution images X_1 from their low resolution versions X_0, or producing colorized videos X_1 from their gray-scale counterparts X_0.

4.2 Building probability paths

Flow Matching drastically simplifies the problem of designing a probability path p_t—together with its corresponding velocity field u_t—by adopting a *conditional* strategy. As a first example, consider conditioning the design of p_t on a single target example $X_1 = x_1$, yielding the **conditional probability path** $p_{t|1}(x|x_1)$ illustrated in figure 3a. Then, we may construct the overall, **marginal probability path** p_t by aggregating such conditional probability paths $p_{t|1}$:

$$p_t(x) = \int p_{t|1}(x|x_1) q(x_1) \mathrm{d}x_1, \tag{4.4}$$

as illustrated in figure 3b. To solve the Flow Matching Problem, we would like p_t to satisfy the following boundary conditions:

$$p_0 = p, \quad p_1 = q, \tag{4.5}$$

that is, the marginal probability path p_t interpolates from the source distribution p at time $t = 0$ to the target distribution q at time $t = 1$. These boundary conditions can be enforced by requiring the conditional probability paths to satisfy

$$p_{0|1}(x|x_1) = \pi_{0|1}(x|x_1), \text{ and } p_{1|1}(x|x_1) = \delta_{x_1}(x), \tag{4.6}$$

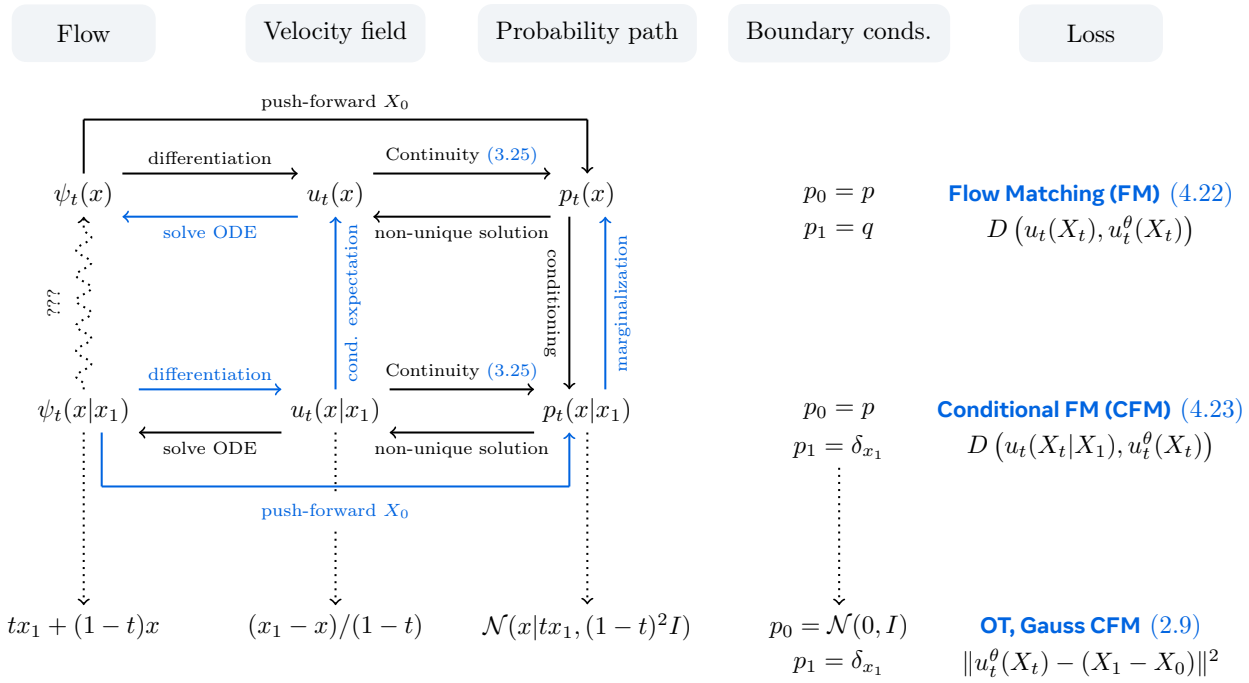


Figure 9 Main objects of the Flow Matching framework and their relationships. A Flow is represented with a Velocity field defining a random process generating a Probability path. The main idea of Flow Matching is to break down the construction of a complex flow satisfying the desired Boundary conditions (top row) to conditional flows (middle row) satisfying simpler Boundary conditions and consequently easier to solve. The arrows indicate dependencies between different objects; **Blue arrows** signify relationships employed by the Flow Matching framework. The Loss column lists the losses for learning the Velocity field, where the CFM loss (middle and bottom row) is what used in practice. The bottom row lists the simplest FM algorithm instantiation as described in section 2.

where the conditional coupling $\pi_{0|1}(x_0|x_1) = \pi_{0,1}(x_0, x_1)/q(x_1)$ and δ_{x_1} is the delta measure centered at x_1. For the independent coupling $\pi_{0,1}(x_0, x_1) = p(x_0)q(x_1)$, the first constraint above reduces to $p_{0|1}(x|x_1) = p(x)$. Because the delta measure does not have a density, the second constraint should be read as $\int p_{t|1}(x|y)f(y)\mathrm{d}y \to f(x)$ as $t \to 1$ for continuous functions f. Note that the boundary conditions (4.5) can be verified plugging (4.6) into (4.4).

A popular example of a conditional probability path satisfying the conditions in (4.6) was given in (2.2):

$$\mathcal{N}(\cdot \,|\, tx_1, (1-t)^2 I) \to \delta_{x_1}(\cdot) \text{ as } t \to 1.$$

4.3 Deriving generating velocity fields

Equipped with a marginal probability path p_t, we now build a velocity field u_t generating p_t. The generating velocity field u_t is an average of multiple **conditional velocity fields** $u_t(x|x_1)$, illustrated in figure 3c, and satisfying:

$$u_t(\cdot|x_1) \text{ generates } p_{t|1}(\cdot|x_1). \tag{4.7}$$

Then, the **marginal velocity field** $u_t(x)$, generating the marginal path $p_t(x)$, illustrated in figure 3d, is given by averaging the conditional velocity fields $u_t(x|x_1)$ across target examples:

$$u_t(x) = \int u_t(x|x_1)p_{1|t}(x_1|x)\mathrm{d}x_1. \tag{4.8}$$

To express the equation above using known terms, recall Bayes' rule

$$p_{1|t}(x_1|x) = \frac{p_{t|1}(x|x_1)q(x_1)}{p_t(x)}, \tag{4.9}$$

defined for all x with $p_t(x) > 0$. Equation (4.8) can be interpreted as the weighted average of the conditional velocities $u_t(x|x_1)$, with weights $p_{1|t}(x_1|x)$ representing the posterior probability of target samples x_1 given the current sample x. Another interpretation of (4.8) can be given with conditional expectations (see section 3.2). Namely, if X_t is *any* RV such that $X_t \sim p_{t|1}(\cdot|X_1)$, or equivalently, the joint distribution of (X_t, X_1) has density $p_{t,1}(x, x_1) = p_{t|1}(x|x_1)q(x_1)$ then using (3.12) to write (4.8) as a conditional expectation, we obtain

$$u_t(x) = \mathbb{E}\left[u_t(X_t|X_1) \,|\, X_t = x\right], \tag{4.10}$$

which yields the useful interpretation of $u_t(x)$ as the least-squares approximation to $u_t(X_t|X_1)$ given $X_t = x$, see section 3.2. Note, that the X_t in (4.10) is in general a different RV that X_t defined by the final flow model (3.16), although they share the same marginal probability $p_t(x)$.

4.4 General conditioning and the Marginalization Trick

To justify the constructions above, we need to show that the marginal velocity field u_t from equations (4.8) and (4.10) generates the marginal probability path p_t from equation (4.4) under mild assumptions. The mathematical tool to prove this is the Mass Conservation Theorem (theorem 2). To proceed, let us consider a slightly more general setting that will be useful later in the manuscript. In particular, there is nothing special about building conditional probability paths and velocity fields by conditioning on $X_1 = x_1$. As noted in Tong et al. (2023), the analysis from the previous section carries through to conditioning on any arbitrary RV $Z \in \mathbb{R}^m$ with PDF p_Z. This yields the **marginal probability path**

$$p_t(x) = \int p_{t|Z}(x|z)p_Z(z)dz, \tag{4.11}$$

which in turn is generated by the **marginal velocity field**

$$u_t(x) = \int u_t(x|z)p_{Z|t}(z|x)dz = \mathbb{E}\left[u_t(X_t|Z) \mid X_t = x\right], \tag{4.12}$$

where $u_t(\cdot|z)$ generates $p_{t|Z}(\cdot|z)$, $p_{Z|t}(z|x) = \frac{p_{t|Z}(x|z)p_Z(z)}{p_t(x)}$ follows from Bayes' rule given $p_t(x) > 0$, and $X_t \sim p_{t|Z}(\cdot|Z)$. Naturally, we can recover the constructions in previous sections by setting $Z = X_1$. Before we prove the main result, we need some regularity assumptions, encapsulated as follows.

Assumption 1. $p_{t|Z}(x|z)$ is $C^1([0,1) \times \mathbb{R}^d)$ and $u_t(x|z)$ is $C^1([0,1) \times \mathbb{R}^d, \mathbb{R}^d)$ as a function of (t, x). Furthermore, p_Z has bounded support, that is, $p_Z(x) = 0$ outside some bounded set in $\mathbb{R}^m$. Finally, $p_t(x) > 0$ for all $x \in \mathbb{R}^d$ and $t \in [0, 1)$.

These are mild assumptions. For example, one can show that $p_t(x) > 0$ by finding a condition z such that $p_Z(z) > 0$ and $p_{t|Z}(\cdot|z) > 0$. In practice, one can satisfy this by considering $(1-(1-t)\epsilon)p_{t|Z} + (1-t)\epsilon\mathcal{N}(0, I)$ for an arbitrarily small $\epsilon > 0$. One example of $p_{t|Z}(\cdot|z)$ satisfying this assumption is the path in (2.2), where we let $Z = X_1$. We are now ready to state the main result:

Theorem 3 (Marginalization Trick)**.** *Under assumption 1, if $u_t(x|z)$ is conditionally integrable and generates the conditional probability path $p_t(\cdot|z)$, then the marginal velocity field u_t generates the marginal probability path p_t, for all $t \in [0, 1)$.*

In the theorem above, *conditionally integrable* refers to a conditional version of the integrability condition from the Mass Conservation Theorem (3.26), namely:

$$\int_0^1 \int \int \|u_t(x|z)\| \, p_{t|Z}(x|z)p_Z(x)dzdxdt < \infty. \tag{4.13}$$

Proof. The result follows from verifying the two conditions of the Mass Conservation in theorem 2. First, let us check that the pair (u_t, p_t) satisfies the Continuity Equation (3.25). Because $u_t(\cdot|x_1)$ generates $p_t(\cdot|x_1)$, we

have that

$$\frac{\mathrm{d}}{\mathrm{d}t}p_t(x) \overset{(i)}{=} \int \frac{\mathrm{d}}{\mathrm{d}t}p_{t|Z}(x|z)p_Z(x)dz \tag{4.14}$$

$$\overset{(ii)}{=} -\int \mathrm{div}_x\left[u_t(x|z)p_{t|Z}(x|z)\right]p_Z(z)dz \tag{4.15}$$

$$\overset{(i)}{=} -\mathrm{div}_x \int u_t(x|z)p_{t|Z}(x|z)p_Z(z)dz \tag{4.16}$$

$$\overset{(iii)}{=} -\mathrm{div}_x\left[u_t(x)p_t(x)\right]. \tag{4.17}$$

Equalities (i) follows from switching differentiation ($\frac{\mathrm{d}}{\mathrm{d}t}$ and div_x, respectively) and integration, as justified by Leibniz's rule, the fact that $p_{t|Z}(x|z)$ and $u_t(x|z)$ are C^1 in t, x, and the fact that p_Z has bounded support (so all the integrands are integrable as continuous functions over bounded sets). Equality (ii) follows from the fact that $u_t(\cdot|z)$ generates $p_{t|Z}(\cdot|z)$ and theorem 2. Equality (iii) follows from multiplying and dividing by $p_t(x)$ (strictly positive by assumption) and using the formula (4.12) for u_t.

To verify the second and last condition from theorem 2, we shall prove that u_t is integrable and locally Lipschitz. Because C^1 functions are locally Lipschitz, it suffices to check that $u_t(x)$ is C^1 for all (t, x). This would follow from verifying that $u_t(x|z)$ and $p_{t|Z}(x|z)$ are C^1 and $p_t(x) > 0$, which hold by assumption. Furthermore, $u_t(x)$ is integrable because $u_t(x|z)$ is conditionally integrable:

$$\int_0^1 \int \|u_t(x)\|\, p_t(x)\mathrm{d}x\mathrm{d}t \leq \int_0^1 \int\int \|u_t(x|z)\|\, p_{t|Z}(x|z)p_Z(z)dz\mathrm{d}x\mathrm{d}t < \infty, \tag{4.18}$$

where the first inequality follows from vector Jensen's inequality. □

4.5 Flow Matching loss

After having established that the target velocity field u_t generates the prescribed probability path p_t from p to q, the missing ingredient is a tractable loss function to learn a velocity field model u_t^θ as close as possible to the target u_t. One major roadblock towards stating this loss function directly is that computing the target u_t is infeasible, as it requires marginalizing over the entire training set (that is, integrating with respect to x_1 in equation (4.8) or with respect to z in equation (4.12)). Fortunately, a family of loss functions known as **Bregman divergences** provides unbiased gradients to learn $u_t^\theta(x)$ in terms of *conditional* velocities $u_t(x|z)$ alone.

Bregman divergences measure dissimilarity between two vectors $u, v \in \mathbb{R}^d$ as

$$D(u, v) \coloneqq \Phi(u) - \left[\Phi(v) + \langle u - v, \nabla\Phi(v)\rangle\right], \tag{4.19}$$

where $\Phi : \mathbb{R}^d \to \mathbb{R}$ is a strictly convex function defined over some convex set $\Omega \subset \mathbb{R}^d$. As illustrated in figure 10, the Bregman divergence measures the difference between $\Phi(u)$ and the linear approximation to Φ developed around v and evaluated at u. Because linear approximations are global lower bounds for convex functions, it holds that $D(u, v) \geq 0$. Further, as Φ is strictly convex, it follows that $D(u, v) = 0$ if and only if $u = v$. The most basic Bregman divergence is the squared Euclidean distance $D(u, v) = \|u - v\|^2$, esulting from choosing $\Phi(u) = \|u\|^2$. The key property that makes Bregman divergences useful for Flow Matching is that their gradient with respect to the second argument is *affine invariant* (Holderrieth et al., 2024):

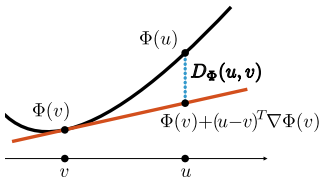


Figure 10 Bregman divergence.

$$\nabla_v D(au_1 + bu_2, v) = a\nabla_v D(u_1, v) + b\nabla_v D(u_2, v), \text{ for any } a + b = 1, \tag{4.20}$$

as it can be verified from equation (4.19). Affine invariance allows us to swap expected values with gradients as follows:

$$\nabla_v D(\mathbb{E}[Y], v) = \mathbb{E}[\nabla_v D(Y, v)] \text{ for any RV } Y \in \mathbb{R}^d. \tag{4.21}$$

The **Flow Matching loss** employs a Bregman divergence to regress our learnable velocity $u_t^\theta(x)$ onto the target velocity $u_t(x)$ along the probability path p_t:

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{t, X_t \sim p_t} D(u_t(X_t), u_t^\theta(X_t)), \tag{4.22}$$

where time $t \sim U[0,1]$. As mentioned above, however, the target velocity u_t is not tractable, so the loss above cannot be computed as is. Instead, we consider the simpler and tractable **Conditional Flow Matching (CFM) loss**:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t, Z, X_t \sim p_{t|Z}(\cdot|Z)} D(u_t(X_t|Z), u_t^\theta(X_t)). \tag{4.23}$$

The two losses are equivalent for learning purposes, since their gradients coincide (Holderrieth et al., 2024):

Theorem 4. *The gradients of the Flow Matching loss and the Conditional Flow Matching loss coincide:*

$$\nabla_\theta \mathcal{L}_{FM}(\theta) = \nabla_\theta \mathcal{L}_{CFM}(\theta). \tag{4.24}$$

In particular, the minimizer of the Conditional Flow Matching loss is the marginal velocity $u_t(x)$.

Proof. The proof follows a direct computation:

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\text{FM}}(\theta) &= \nabla_\theta \mathbb{E}_{t, X_t \sim p_t} D(u_t(X_t), u_t^\theta(X_t)) \\
&= \mathbb{E}_{t, X_t \sim p_t} \nabla_\theta D(u_t(X_t), u_t^\theta(X_t)) \\
&\overset{(i)}{=} \mathbb{E}_{t, X_t \sim p_t} \nabla_v D(u_t(X_t), u_t^\theta(X_t)) \nabla_\theta u_t^\theta(X_t) \\
&\overset{(4.12)}{=} \mathbb{E}_{t, X_t \sim p_t} \nabla_v D(\mathbb{E}_{Z \sim p_{Z|t}(\cdot|X_t)}[u_t(X_t|Z)], u_t^\theta(X_t)) \nabla_\theta u_t^\theta(X_t) \\
&\overset{(ii)}{=} \mathbb{E}_{t, X_t \sim p_t} \mathbb{E}_{Z \sim p_{Z|t}(\cdot|X_t)} \left[\nabla_v D(u_t(X_t|Z), u_t^\theta(X_t)) \nabla_\theta u_t^\theta(X_t)\right] \\
&\overset{(iii)}{=} \mathbb{E}_{t, X_t \sim p_t} \mathbb{E}_{Z \sim p_{Z|t}(\cdot|X_t)} [\nabla_\theta D(u_t(X_t|Z), u_t^\theta(X_t))] \\
&\overset{(iv)}{=} \nabla_\theta \mathbb{E}_{t, Z \sim q, X_t \sim p_{t|Z}(\cdot|Z)} [D(u_t(X_t|Z), u_t^\theta(X_t))] \\
&= \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta)
\end{aligned}$$

where in (i),(iii) we used the chain rule; (ii) follows from equation (4.21) applied conditionally on X_t; and in (iv) we use Bayes' rule. □

Bregman divergences for learning conditional expectations. Theorem 4 is a particular instance of a more general result utilizing Bregman divergences for learning conditional expectations described next. It will be used throughout this manuscript and provide the basis for all scalable losses behind Flow Matching:

Proposition 1 (Bregman divergence for learning conditional expectations)**.** *Let $X \in \mathcal{S}_X, Y \in \mathcal{S}_Y$ be RVs over state spaces $\mathcal{S}_X, \mathcal{S}_Y$ and $g : \mathbb{R}^p \times \mathcal{S}_X \to \mathbb{R}^n$, $(\theta, x) \mapsto g^\theta(x)$, where $\theta \in \mathbb{R}^p$ denotes learnable parameters. Let $D_x(u, v)$, $x \in \mathcal{S}_X$ be a Bregman divergence over a convex set $\Omega \subset \mathbb{R}^n$ that contains the image of f. Then,*

$$\nabla_\theta \mathbb{E}_{X,Y} D_X\left(Y, g^\theta(X)\right) = \nabla_\theta \mathbb{E}_X D_X\left(\mathbb{E}\left[Y \mid X\right], g^\theta(X)\right). \tag{4.25}$$

In particular, for all x with $p_X(x) > 0$, the global minimum of $g^\theta(x)$ w.r.t. θ satisfies

$$g^\theta(x) = \mathbb{E}\left[Y \mid X = x\right]. \tag{4.26}$$

Proof. We assume g^θ is differentiable w.r.t. θ and that the distributions of X and Y, as well as D_x, and g allow switching differentiation and integration, develop:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{X,Y} D_X\left(Y, g^\theta(X)\right) &\overset{(i)}{=} \mathbb{E}_X\left[\mathbb{E}\left[\nabla_v D_X\left(Y, g^\theta(X)\right) \nabla_\theta g^\theta(X) \mid X\right]\right] \\ &\overset{(ii)}{=} \mathbb{E}_X\left[\nabla_v D_X\left(\mathbb{E}\left[Y \mid X\right], g^\theta(X)\right) \nabla_\theta g^\theta(X)\right] \\ &\overset{(iii)}{=} \mathbb{E}_X\left[\nabla_\theta D_X\left(\mathbb{E}\left[Y \mid X\right], g^\theta(X)\right)\right] \\ &= \nabla_\theta \mathbb{E}_X D_X\left(\mathbb{E}\left[Y \mid X\right], g^\theta(X)\right),\end{aligned}$$

where (i) follows from the chain rule and the tower property of expectations (3.11). Equality (ii) follows from (4.21). Equality (iii) uses the chain rule again. Lastly, for every $x \in \mathcal{S}_X$ with $p_X(x) > 0$ we can choose $g^\theta(x) = \mathbb{E}\left[Y|X = x\right]$, obtaining $\mathbb{E}_X D_X\left(\mathbb{E}\left[Y \mid X\right], g^\theta(X)\right) = 0$, which must be the global minimum with respect to θ. □

Theorem 4 is readily shown from proposition 1 by making the choices $X = X_t$, $Y = u_t(X_t|Z)$, $g^\theta(x) = u_t^\theta(x)$, and taking the expectation with respect to $t \sim U[0,1]$.

General time distributions One useful variation of the FM loss is to sample times t from a distribution other than Uniform. Specifically, consider $t \sim \omega(t)$, where ω is a PDF over $[0,1]$. This leads to the following weighted objective:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t\sim\omega, Z, X_t} D(u_t(X_t|Z), u_t^\theta(X_t)) = \mathbb{E}_{t\sim U, Z, X_t} \omega(t) D(u_t(X_t|Z), u_t^\theta(X_t)). \tag{4.27}$$

Although mathematically equivalent, sampling $t \sim \omega$ leads to better performance than using weights $\omega(t)$ in large scale image generation tasks (Esser et al., 2024).

4.6 Solving conditional generation with conditional flows

So far, we have reduced the problem of training a flow model u_t^θ to: (i) Find conditional probability paths $p_{t|Z}(x|z)$ yielding a marginal probability path $p_t(x)$ satisfying the boundary conditions in (4.5). (ii) Find conditional velocity fields $u_t(x|z)$ generating the conditional probability path. (iii) Train using the Conditional Flow Matching loss (see equation (4.23)). We now discuss a concrete options on how to do step (i) and (ii), *i.e.*, design such conditional probability paths and velocity fields.

We will now propose a flexible method to design such conditional probability paths and velocity fields using a specific construction via **conditional flows**. The idea is as follows: *Define* a flow model $X_{t|1}$ (similarly to (3.16)) satisfying the boundary conditions (4.6), and extract the velocity field from $X_{t|1}$ by differentiation (3.20). This process defines both $p_{t|1}(x|x_1)$ and $u_t(x|x_1)$. In more detail, define the **conditional flow model**

$$X_{t|1} = \psi_t(X_0|x_1), \qquad \text{where } X_0 \sim \pi_{0|1}(\cdot \,|\, x_1), \tag{4.28}$$

where $\psi : [0,1) \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a **conditional flow** defined by

$$\psi_t(x|x_1) = \begin{cases} x & t = 0 \\ x_1 & t = 1 \end{cases}, \tag{4.29}$$

smooth in (t, x), and a diffeomorphism in x. (Smooth here means that all derivatives of $\psi_t(x|x_1)$ with respect to t and x exist and are continuous: $C^\infty([0,1) \times \mathbb{R}^d, \mathbb{R}^d)$). These conditions could be further relaxed to $C^2([0,1) \times \mathbb{R}^d, \mathbb{R}^d)$ at the expense of simplicity.) The push-forward formula (3.15) defines the probability density of $X_{t|1}$ as

$$p_{t|1}(x|x_1) \coloneqq \left[\psi_t(\cdot|x_1)_\sharp \pi_{0|1}(\cdot|x_1)\right](x), \tag{4.30}$$

although we will not need this expression in practical optimization of the CFM loss it is used theoretically to show that $p_{t|1}$ satisfies the two boundary conditions (4.6). First, and according to (4.29), $\psi_0(\cdot|x_1)$ is the identity map, keeping $\pi_{0|1}(\cdot|x_1)$ intact at time $t = 0$. Second, $\psi_1(\cdot|x_1) = x_1$ is the constant map, concentrating all probability mass at x_1 as $t \to 1$. Furthermore, note that $\psi_t(\cdot|x_1)$ is a smooth diffeomorphism

for $t \in [0, 1)$. Therefore, by the equivalence of flows and velocity fields (section 3.4.1), there exists a unique smooth conditional velocity field (see equation (3.20)) taking form:

$$u_t(x|x_1) = \dot{\psi}_t(\psi_t^{-1}(x|x_1)|x_1). \tag{4.31}$$

To summarize: we have further reduced the task of finding the conditional path and a corresponding generating velocity to simply building a conditional flow $\psi_t(\cdot|x_1)$ satisfying (4.29). In section 4.7 we will pick a particularly simple $\psi_t(x|x_1)$ with some desirable properties (conditional Optimal Transport flow) that leads to the standard Flow Matching algorithm as seen in section 1, and in section 4.8 we will discuss a particular and well-known family of conditional flows, namely affine flows that include some known examples from the diffusion models' literature. In section 5 we will use conditional flows to define Flow Matching on manifold which showcase the flexibility of this approach.

4.6.1 The Conditional Flow Matching loss, revisited

Let us revisit the CFM loss (4.23) by setting $Z = X_1$ and using the conditional flows way of defining the conditional probability path and velocity,

$$\begin{aligned} \mathcal{L}_{\text{CFM}}(\theta) &= \mathbb{E}_{t, X_1, X_t \sim p_t(\cdot|X_1)} D\left(u_t(X_t|X_1), u_t^\theta(X_t)\right) \\ &\overset{(3.4)}{=} \mathbb{E}_{t,(X_0,X_1)\sim\pi_{0,1}} D\left(\dot{\psi}_t(X_0|X_1), u_t^\theta(X_t)\right) \end{aligned} \tag{4.32}$$

where in the second equality we used the Law of Unconscious Statistician with $X_t = \psi_t(X_0|X_1)$ and

$$u_t(X_t|X_1) \overset{(4.31)}{=} \dot{\psi}_t\left(\psi_t^{-1}\left(\psi_t\left(X_0|X_1\right)|X_1\right)\middle|X_1\right) = \dot{\psi}_t(X_0|X_1). \tag{4.33}$$

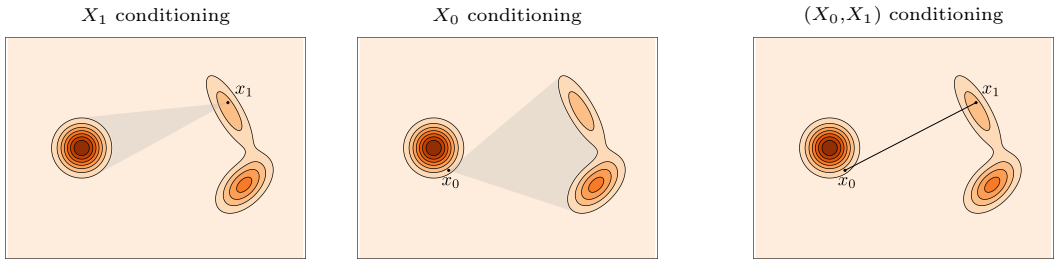
The minimizer of the loss (4.32) according to proposition 1 takes the form as in (Liu et al., 2022),

$$u_t(x) = \mathbb{E}\left[\dot{\psi}_t(X_0|X_1)\,\middle|\,X_t = x\right]. \tag{4.34}$$

In the `flow_matching` library the `ProbPath` object defines a probability path. This probability path can be sampled at (t, X_0, X_1) to obtain X_t and $\dot{\psi}_t(X_0|X_1)$. Then, one can compute a Monte Carlo estimate of the CFM loss $\mathcal{L}_{\text{CFM}}(\theta)$. An example training loop with the CFM objective is shown in code 4.

Code 4: Training with the conditional flow matching (CFM) loss

```python
import torch
from flow_matching.path import ProbPath
from flow_matching.path.path_sample import PathSample

path: ProbPath = ...  # The flow_matching library implements the most common probability paths
velocity_model: torch.nn.Module = ...  # Initialize the velocity model
optimizer = torch.optim.Adam(velocity_model.parameters())

for x_0, x_1 in dataloader:  # Samples from π_{0,1} of shape [batch_size, *data_dim]
    t = torch.rand(batch_size)  # Randomize time t ~ U[0,1]
    sample: PathSample = path.sample(t=t, x_0=x_0, x_1=x_1)
    x_t = sample.x_t
    dx_t = sample.dx_t  # dX_t is ψ̇_t(X_0|X_1).
    # If D is the Euclidean distance, the CFM objective corresponds to the mean-squared error
    cfm_loss = torch.pow(velocity_model(x_t, t) - dx_t, 2).mean()  # Monte Carlo estimate
    optimizer.zero_grad()
    cfm_loss.backward()
    optimizer.step()
```



$$
\begin{array}{rccccc}
 & \psi_t(X_0|x_1) \sim p_{t|1}(\cdot|x_1) & & \psi_t(X_1|x_0) \sim p_{t|0}(\cdot|x_0) & & \psi_t(x_0,x_1) \sim p_{t|0,1}(\cdot|x_0,x_1) \\
p_t(x) = & \int p_{t|1}(x|x_1) q(x_1) \mathrm{d}x_1 & = & \int p_{t|0}(x|x_0) p(x_0) \mathrm{d}x_1 & = & \int p_{t|0,1}(x|x_0,x_1) \pi_{0,1}(x_0,x_1) \mathrm{d}x_0 \mathrm{d}x_1 \\
u_t(x) = & \mathbb{E}\left[u_t(X_t|X_1)|X_t = x\right] & = & \mathbb{E}\left[u_t(X_t|X_0)|X_t = x\right] & = & \mathbb{E}\left[u_t(X_t|X_0,X_1)|X_t = x\right]
\end{array}
$$

Figure 11 Different forms of conditioning in Flow Matching and path design with corresponding conditional flows. When the conditional flows are a diffeomorphism, all constructions are equivalent. When, they are not, extra conditions are required to validate that the marginal velocity generates the marginal path, see text for more details.

4.6.2 The Marginalization Trick for probability paths built from conditional flows

Next, we introduce a version of the Marginalization trick for probability paths that are built from conditional flows. To this end, note that if $\pi_{0|1}(\cdot|x_1)$ is C^1, then $p_t(x|x_1)$ is also C^1 by construction; moreover, $u_t(x|x_1)$ is conditionally integrable if

$$\mathbb{E}_{t,(X_0,X_1)\sim\pi_{0,1}} \left\| \dot{\psi}_t(X_0|X_1) \right\| < \infty. \tag{4.35}$$

Therefore, by setting $Z = X_1$, the following corollary to theorem 3 is obtained.

Corollary 1. *Assume that q has bounded support, $\pi_{0|1}(\cdot|x_1)$ is $C^1(\mathbb{R}^d)$ and strictly positive for some x_1 with $q(x_1) > 0$, and $\psi_t(x|x_1)$ is a conditional flow satisfying equations (4.29) and (4.35). Then $p_{t|1}(x|x_1)$ and $u_t(x|x_1)$, defined in (4.30) and (4.31), respectively, define a marginal velocity field $u_t(x)$ generating the marginal probability path $p_t(x)$ interpolating p and q.*

Proof. If $\pi_{0|1}(\cdot|x_1) > 0$ for some $x_1 \in \mathbb{R}^d$ such that $q(x_1) > 0$, it follows that $p_{t|1}(x|x_1) > 0$ for all $x \in \mathbb{R}^d$ and is $C^1([0,1) \times \mathbb{R}^d)$ (see (4.30) and (3.15) for definitions). Furthermore, $u_t(x|x_1)$ (defined in (4.31)) is smooth and satisfies

$$
\begin{aligned}
\int_0^1 \int \|u_t(x|x_1)\| \, p_{t|1}(x|x_1) q(x_1) \mathrm{d}x_1 \mathrm{d}x \mathrm{d}t \; &= \mathbb{E}_{t,X_1\sim q, X_t\sim p_{t|1}(\cdot|X_1)} \|u_t(X_t|X_1)\| \\
&\overset{(3.4)}{=} \mathbb{E}_{t,X_1\sim q, X_0\sim\pi_{0|1}(\cdot|X_1)} \|u_t(\psi_t(X_0|X_1)|X_1)\| \\
&\overset{(4.33)}{=} \mathbb{E}_{t,(X_0,X_1)\sim\pi_{0,1}} \left\| \dot{\psi}_t(X_0|X_1) \right\| \\
&< \infty.
\end{aligned}
$$

Therefore, $u_t(x|x_1)$ is conditionally integrable (see (4.13)). By theorem 3, the marginal u_t generates p_t. Because $p_{t|1}(x|x_1)$ as defined by (4.30) satisfies (4.6), it follows that p_t interpolates p and q. □

This theorem will be used as a tool to show that particular choices of conditional flows lead to marginal velocity $u_t(x)$ generating the marginal probability path $p_t(x)$.

4.6.3 Conditional flows with other conditions

Different conditioning choices Z exist but are essentially all equivalent. As illustrated in figure 11, main options include fixing target samples $Z = X_1$ (Lipman et al., 2022), source samples $Z = X_0$ (Esser et al., 2024), or two-sided $Z = (X_0, X_1)$ (Albergo and Vanden-Eijnden, 2022; Liu et al., 2022; Pooladian et al., 2023; Tong et al., 2023).

Let us focus on the two-sided condition $Z = (X_0, X_1)$. Following the FM blueprint described above, we are now looking to build a conditional probability path $p_{t|0,1}(x|x_0, x_1)$ and a corresponding generating velocity $u_t(x|x_0, x_1)$ such that

$$p_{0|0,1}(x|x_0, x_1) = \delta_{x_0}(x), \text{ and } p_{1|0,1}(x|x_0, x_1) = \delta_{x_1}(x). \tag{4.36}$$

We will keep this discussion formal as it requires usage of delta functions δ and our existing derivations so far only deals with probability densities (and not general distributions). To build such a path we can consider an **interpolant** (Albergo and Vanden-Eijnden, 2022) defined by $X_{t|0,1} = \psi_t(x_0, x_1)$ for a function $\psi : [0, 1] \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ satisfying conditions similar to (4.29),

$$\psi_t(x_0, x_1) = \begin{cases} x_0 & t = 0 \\ x_1 & t = 1. \end{cases} \tag{4.37}$$

Therefore, $\psi_t(\cdot, x_1)$ pushes $\delta_{x_0}(x)$ to $\delta_{x_1}(x)$. We now, similarly to before, define the conditional probability path to be

$$p_{t|0,1}(\cdot|x_0, x_1) \coloneqq \psi_t(\cdot, x_1)_\sharp \delta_{x_0}(\cdot) \tag{4.38}$$

which satisfies the boundary constraints in (4.36). Albergo and Vanden-Eijnden (2022)'s **stochastic interpolant** is defined by

$$X_t = \psi_t(X_0, X_1) \sim p_t(\cdot) = \int p_{t|0,1}(\cdot|x_0, x_1)\pi_{0,1}(x_0, x_1)\mathrm{d}x_0\mathrm{d}x_1. \tag{4.39}$$

Next, the conditional velocity along this path can also be computed with (3.20) giving

$$u_t(x|x_0, x_1) = \dot{\psi}_t(x_0, x_1) \tag{4.40}$$

which is defined only for $x = \psi_t(x_0, x_1)$. Ignoring for a second the extra conditions, Theorem 3 now presumably implies that the marginal velocity generating $p_t(x)$ is

$$\begin{aligned} u_t(x) &= \mathbb{E}\left[u_t(X_t|X_0, X_1) \,|\, X_t = x\right] \\ &= \mathbb{E}\left[\dot{\psi}_t(X_0, X_1) \,|\, X_t = x\right], \end{aligned}$$

which leads to the same marginal formula as the X_1-conditioned case (4.34), but with a seemingly more permissive conditional flow $\psi_t(x_0, x_1)$ which is only required to be an interpolant now, weakening the more stringent diffeomorphism condition. However, A more careful look reveals that some extra conditions are still required to make $u_t(x)$ a generating velocity for $p_t(x)$ and simple interpolation (as defined in (4.37)) is not enough to guarantee this, not even with extra smoothness conditions, as required in Theorem 3. To see this, consider

$$\psi_t(x_0, x_1) = (1 - 2t)_+^\tau x_0 + (2t - 1)_+^\tau x_1, \text{ where } (s)_+ = \mathrm{ReLU}(s), \tau > 2,$$

a $C^2([0, 1])$ interpolant (in time) concentrating all probability mass at location 0 at time $t = 0.5$ for all x_0, x_1. That is $\mathbb{P}(X_{\frac{1}{2}} = 0) = 1$. Therefore, assuming $u_t(x)$ indeed generates $p_t(x)$ its marginal at $t = \frac{1}{2}$ is δ_0 and since a flow is both Markovian (as shown in (3.17)) and deterministic its marginal has to be a delta function for all $t > 0.5$ leading to a contradiction since $X_1 = \psi_1(X_0, X_1) \sim q$, which is generally not a delta function. Albergo and Vanden-Eijnden (2022) and Liu et al. (2022) provide some extra conditions that guarantee that $u_t(x)$ indeed geenrates $p_t(x)$ but these are somewhat harder to verify compared to the conditions of Theorem 3. Below we will show how to practically check the conditions of Theorem 3 to validate that particular paths of interest are guaranteed to be generated by the respective marginal velocities.

Nevertheless, when $\psi_t(x_0, x_1)$ is in addition a diffeomorphism in x_0 for a fixed x_1, and in x_1 for a fixed x_0, the three constructions leads to the same marginal velocity, defined by (4.34), and same marginal probability path p_t, defined by $X_t = \psi_t(X_0, X_1) = \psi_t(X_0|X_1) = \psi_t(X_1|X_0)$, see Figure 11.

4.7 Optimal Transport and linear conditional flow

We now ask: how to find a useful conditional flow $\psi_t(x|x_1)$? One approach is to choose it as a minimizer of a natural cost functional, ideally with some desirable properties. One popular example of such cost functional is the dynamic Optimal Transport problem with quadratic cost (Villani et al., 2009; Villani, 2021; Peyré et al., 2019), formalized as

$$(p_t^\star, u_t^\star) = \arg\min_{p_t, u_t} \int_0^1 \int \|u_t(x)\|^2 p_t(x) \mathrm{d}x\mathrm{d}t \qquad \text{(Kinetic Energy)} \tag{4.41a}$$

$$\text{s.t. } p_0 = p, p_1 = q \qquad \text{(interpolation)} \tag{4.41b}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t + \operatorname{div}(p_t u_t) = 0. \qquad \text{(continuity equation)} \tag{4.41c}$$

The $(p_t^\star, u_t^\star)$ above defines a flow (via equation (3.19)) with the form

$$\psi_t^\star(x) = t\phi(x) + (1-t)x, \tag{4.42}$$

called the **OT displacement interpolant** (McCann, 1997), where $\phi : \mathbb{R}^d \to \mathbb{R}^d$ is the Optimal Transport map. The OT displacement interpolant also solves the Flow Matching Problem (4.1) by defining the random variable

$$X_t = \psi_t^\star(X_0) \sim p_t^\star \quad \text{when} \quad X_0 \sim p. \tag{4.43}$$

The Optimal Transport formulation promotes straight sample trajectories

$$X_t = \psi_t^\star(X_0) = X_0 + t(\phi(X_0) - X_0),$$

with a constant velocity $\phi(X_0) - X_0$, which are in general easier to sample with ODE solvers—in particular, the target sample X_1 is here perfectly solvable with a single step of the Euler Method (3.21).

We can now try to plug our marginal velocity formula (equation (4.34)) into the Optimal Transport problem (4.41) and search for an optimal $\psi_t(x|x_1)$. While this seems like a challenge, we can instead find a bound for the Kinetic Energy for which such a minimizer is readily found (Liu et al., 2022):

$$\int_0^1 \mathbb{E}_{X_t \sim p_t} \|u_t(X_t)\|^2 \mathrm{d}t = \int_0^1 \mathbb{E}_{X_t \sim p_t} \left\| \mathbb{E}\left[\dot{\psi}_t(X_0|X_1) \mid X_t \right] \right\|^2 \mathrm{d}t \tag{4.44}$$

$$\overset{(i)}{\leq} \int_0^1 \mathbb{E}_{X_t \sim p_t} \mathbb{E}\left[\left\| \dot{\psi}_t(X_0|X_1) \right\|^2 \mid X_t \right] \mathrm{d}t \tag{4.45}$$

$$\overset{(ii)}{=} \mathbb{E}_{(X_0, X_1) \sim \pi_{0,1}} \int_0^1 \left\| \dot{\psi}_t(X_0|X_1) \right\|^2 \mathrm{d}t, \tag{4.46}$$

where in the (i) we used Jensen's inequality, and in (ii) we used the tower property of conditional expectations (see equation (3.11)) and switch integration of t and expectation. Now the integrand in (4.46) can be minimized individually for each (X_0, X_1) — this leads to the following variational problem for $\gamma_t = \psi_t(x|x_1)$:

$$\min_{\gamma : [0,1] \to \mathbb{R}^d} \quad \int_0^1 \|\dot{\gamma}_t\|^2 \mathrm{d}t \tag{4.47a}$$

$$\text{s.t.} \quad \gamma_0 = x, \gamma_1 = x_1. \tag{4.47b}$$

This problem can be solved using Euler-Lagrange equations (Gelfand et al., 2000), which in this case take the form $\frac{d^2}{\mathrm{d}t^2}\gamma_t = 0$. By incorporating the boundary conditions, we obtain the minimizer:

$$\psi_t(x|x_1) = tx_1 + (1-t)x. \tag{4.48}$$

Note that although not constrained to be, this choice of $\psi_t(x|x_1)$ is a diffeomorphism in x for $t \in [0, 1)$ and smooth in t, x, as required from conditional flows.

Several conclusions can be drawn:

1. The linear conditional flow minimizes a bound of the Kinetic Energy among *all* conditional flows.
2. In case the target q consists of a *single* data point $q(x) = \delta_{x_1}(\cdot)$ we have that the linear conditional flow in (4.48) is the Optimal Transport (Lipman et al., 2022). Indeed, in this case $X_t = \psi_t(X_0|x_1) \sim p_t$ and $X_0 = \psi^{-1}(X_t|x_1)$ is a function of X_t which makes $\mathbb{E}\left[\dot{\psi}_t(X_0|x_1) \,\middle|\, X_t\right] = \dot{\psi}_t(X_0|x_1)$ and therefore (ii) becomes an equality.

   Theorem 5. *If $q = \delta_{x_1}$, then the dynamic OT problem (4.41) has an analytic solution given by the OT displacement interpolant in (4.48).*

3. Plugging the linear conditional flow in (4.46) we get

$$\int_0^1 \mathbb{E}_{X_t \sim p_t} \|u_t(X_t)\|^2 \, \mathrm{d}t \leq \mathbb{E}_{(X_0, X_1) \sim \pi_{0,1}} \int_0^1 \|X_1 - X_0\|^2 \, \mathrm{d}t \tag{4.49}$$

   showing that the Kinetic Energy of the marginal velocity $u_t(x)$ is not bigger than that of the original coupling $\pi_{0,1}$ (Liu et al., 2022).

The conditional flow in (4.48) is in particular affine and consequently motivates investigating the family of *affine* conditional flows, discussed next.

4.8 Affine conditional flows

In the previous section we discovered the linear (Conditional-OT) flows as a minimizer to a bound of the Kinetic Energy among *all* conditional flows. The linear conditional flow is a particular instance the wider family of *affine conditional flows*, explored in this section.

$$\psi_t(x|x_1) = \alpha_t x_1 + \sigma_t x, \tag{4.50}$$

where $\alpha_t, \sigma_t : [0,1] \to [0,1]$ are smooth functions satisfying

$$\alpha_0 = 0 = \sigma_1, \;\; \alpha_1 = 1 = \sigma_0, \;\; \text{and } \dot{\alpha}_t, -\dot{\sigma}_t > 0 \text{ for } t \in (0,1). \tag{4.51}$$

We call the pair (α_t, σ_t) a *scheduler*. The derivative condition above ensures that α_t is strictly monotonically increasing, while σ_t is strictly monotonically decreasing. The conditional flow (4.50) is a simple affine map in x for each $t \in [0,1)$, which satisfies the conditions (4.29). The associated marginal velocity field (4.34) is

$$u_t(x) = \mathbb{E}\left[\dot{\alpha}_t X_1 + \dot{\sigma}_t X_0 | X_t = x\right]. \tag{4.52}$$

By virtue of corollary 1, we can prove that, if using the independent coupling and a smooth and strictly positive source density p with finite second moments—for instance, a Gaussian $p = \mathcal{N}(\cdot|0, I)$—then u_t generates a probability path p_t interpolating p and q. We formally state this result, significant for Flow Matching applications, as the following theorem.

Theorem 6. *Assume that q has bounded support, p is $C^1(\mathbb{R}^d)$ with strictly positive density with finite second moments, and these two relate by the independent coupling $\pi_{0,1}(x_0, x_1) = p(x_0)q(x_1)$. Let $p_t(x) = \int p_{t|1}(x|x_1)q(x_1)\mathrm{d}x_1$ be defined by equation (4.30), with ψ_t defined by equation (4.50). Then, the marginal velocity (4.52) generates p_t interpolating p and q.*

Proof. We apply corollary 1. First, note that $\pi_{0|1}(\cdot|x_1) = p(\cdot)$ is C^1 and positive everywhere by assumption. Second, ψ_t, defined in (4.50), satisfies (4.29). Third, we are left with checking (4.35):

$$\begin{aligned}
\mathbb{E}_{t,(X_0,X_1)} \left\|\dot{\psi}(X_0|X_1)\right\| &= \mathbb{E}_{t,(X_0,X_1)} \|\dot{\alpha}_t X_1 + \dot{\sigma}_t X_0\| \\
&\leq \mathbb{E}_t|\dot{\alpha}_t| \, \mathbb{E}_{X_1} \|X_1\| + \mathbb{E}_t|\dot{\sigma}_t| \, \mathbb{E}_{X_0} \|X_0\| \\
&= \mathbb{E}_{X_1} \|X_1\| + \mathbb{E}_{X_0} \|X_0\| \\
&< \infty,
\end{aligned}$$

where the last inequality follows from the fact that $X_1 \sim q$ has bounded support and $X_0 \sim p$ has bounded second moments. □

In this affine case, the CFM loss (4.32) takes the form

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,(X_0,X_1)\sim\pi_{0,1}} D(\dot{\alpha}_t X_1 + \dot{\sigma}_t X_0, u_t^\theta(X_t)) \,. \tag{4.53}$$

Code 5: Examples of affine probability paths in the `flow_matching` library

```python
from flow_matching.path import AffineProbPath, CondOTPath
from flow_matching.path.scheduler import (
    CondOTScheduler, PolynomialConvexScheduler, LinearVPScheduler, CosineScheduler)

# Conditional Optimal Transport schedule with α_t = t, σ_t = 1 − t
path = AffineProbPath(scheduler=CondOTScheduler())
path = CondOTPath()  # Shorthand for the affine path with the CondOTScheduler above

# Polynomial schedule with α_t = t^n, σ_t = 1 − t^n
path = AffineProbPath(scheduler=PolynomialConvexScheduler(n=1.0))

# Linear variance preserving schedule with α_t = t, σ_t = √(1 − t^2)
path = AffineProbPath(scheduler=LinearVPScheduler())

# Cosine schedule with α_t = sin(0.5tπ), σ_t = cos(0.5tπ)
path = AffineProbPath(scheduler=CosineScheduler())
```

4.8.1 Velocity parameterizations

In the affine case, the marginal velocity field u_t admits multiple parametrizations, each of them learnable using the Flow Matching losses introduced in section 4.5. To derive these parametrizations, use the equivalent formulations of the affine paths

$$X_t = \alpha_t X_1 + \sigma_t X_0 \quad \Leftrightarrow \quad X_1 = \frac{X_t - \sigma_t X_0}{\alpha_t} \quad \Leftrightarrow \quad X_0 = \frac{X_t - \alpha_t X_1}{\sigma_t}, \tag{4.54}$$

in the marginal velocity formula (4.52), obtaining

$$u_t(x) = \dot{\alpha}_t \mathbb{E}\left[X_1 | X_t = x\right] + \dot{\sigma}_t \mathbb{E}\left[X_0 | X_t = x\right] \tag{4.55}$$

$$= \frac{\dot{\sigma}_t}{\sigma_t} x + \left[\dot{\alpha}_t - \alpha_t \frac{\dot{\sigma}_t}{\sigma_t}\right] \mathbb{E}\left[X_1 | X_t = x\right] \tag{4.56}$$

$$= \frac{\dot{\alpha}_t}{\alpha_t} x + \left[\dot{\sigma}_t - \sigma_t \frac{\dot{\alpha}_t}{\alpha_t}\right] \mathbb{E}\left[X_0 | X_t = x\right], \tag{4.57}$$

where we have used the fact that $\mathbb{E}\left[Z|Z=z\right] = z$. Then, denote the deterministic functions:

$$x_{1|t}(x) = \mathbb{E}\left[X_1 | X_t = x\right] \text{ as the } x_1\text{-prediction (target)}, \tag{4.58}$$

$$x_{0|t}(x) = \mathbb{E}\left[X_0 | X_t = x\right] \text{ as the } x_0\text{-prediction (source)}. \tag{4.59}$$

These provides two more opportunities to parameterize u_t: via the x_1-prediction $x_{1|t}$ (4.56) and via the x_0-prediction $x_{0|t}$ (4.57). Table 1 offers conversion formulas between the parameterizations. These parameterizations can also be learned using a Conditional Matching loss, similar to (4.23). In particular, any function

$$g_t(x) \coloneqq \mathbb{E}\left[f_t(X_0, X_1) | X_t = x\right], \tag{4.60}$$

where $f_t(X_0, X_1)$ is a RV defined as a time-dependent function of X_0 and X_1, can be learned by minimizing a Matching loss of the form

$$\mathcal{L}_{\text{M}}(\theta) = \mathbb{E}_{t,X_t\sim p_t} D\left(g_t(X_t), g_t^\theta(X_t)\right). \tag{4.61}$$

This loss has the same gradients as the Conditional Matching loss

$$\mathcal{L}_{\text{CM}}(\theta) = \mathbb{E}_{t,(X_0,X_1)\sim\pi_{0,1}} D\left(f_t(X_0,X_1), g_t^\theta(X_t)\right). \tag{4.62}$$

To learn $x_{1|t}$, the Conditional Matching loss employs $f_t(x_0,x_1) = x_1$, and similarly for $x_{0|t}$. This procedure is justified by theorem 7, which is an immediate result from proposition 1 when letting $X = X_t$, $Y = f_t(X_0,X_1)$, and integrating with respect to $t \sim U[0,1]$.

Theorem 7. *The gradients of the Matching loss and the Conditional Matching loss coincide for arbitrary functions $f_t(X_0,X_1)$ of X_0, X_1:*

$$\nabla_\theta \mathcal{L}_M(\theta) = \nabla_\theta \mathcal{L}_{CM}(\theta). \tag{4.63}$$

In particular, the minimizer of the Conditional Matching loss is the conditional expectation

$$g_t^\theta(x) = \mathbb{E}\left[f_t(X_0,X_1)|X_t = x\right]. \tag{4.64}$$

Code 6 shows how to train with x_1-prediction using the `flow_matching` library.

Code 6: Training an X_1-prediction model using the Conditional Matching (CM) objective

```python
import torch
from flow_matching.path import AffineProbPath
from flow_matching.solver import ODESolver
from flow_matching.utils import ModelWrapper

path: AffineProbPath = ...
denoiser_model: torch.nn.Module = ...  # Initialize the denoiser
optimizer = torch.optim.Adam(velocity_model.parameters())

for x_0, x_1 in dataloader:  # Samples from π_{0,1} of shape [batch_size, *data_dim]
    t = torch.rand(batch_size)  # Randomize time t ~ U[0, 1]
    sample = path.sample(t=t, x_0=x_0, x_1=x_1)  # Sample the conditional path
    cm_loss = torch.pow(model(sample.x_t, t) - sample.x_1, 2).mean()  # CM loss
    optimizer.zero_grad()
    cm_loss.backward()
    optimizer.step()

# Convert from denoiser to velocity prediction
class VelocityModel(ModelWrapper):
    def __init__(self, denoiser: nn.Module, path: AffineProbPath):
        super().__init__(model=denoiser)
        self.path=path

    def forward(self, x: torch.Tensor, t: torch.Tensor, **extras) -> torch.Tensor:
        x_1_prediction = super().forward(x, t, **extras)
        return self.path.target_to_velocity(x_1=x_1_prediction, x_t=x, t=t)

# Sample X_1
velocity_model = VelocityModel(denoiser=denoiser_model, path=path)
x_0 = torch.randn(batch_size, *data_dim)  # Specify the initial condition
solver = ODESolver(velocity_model=velocity_model)
num_steps = 100
x_1 = solver.sample(x_init=x_0, method='midpoint', step_size=1.0 / num_steps)
```

Singularities in the velocity parameterizations. Seemingly, the coefficients of (4.56) would blow up as $t \to 1$, and similarly for (4.57) as $t \to 0$. If $\mathbb{E}\left[X_1|X_0 = x\right]$ and $\mathbb{E}\left[X_0|X_1 = x\right]$ exist, which is the case for $p(x) > 0$ and $q(x) > 0$, these are not essential singularities *in theory*, meaning that the singularities in $x_{1|t}$ and $x_{0|t}$ would cancel with the singularities of the coefficients of the parameterization. However, these singularities could be still problematic *in practice* when the learnable $x_{1|t}^\theta$ and $x_{0|t}^\theta$ are by construction continuous and therefore do not perfectly regress their targets $x_{1|t}$ and $x_{0|t}$. To understand how to fix these potential issues, recall (4.55) and consider $u_0(x) = \dot{\alpha}_0 \mathbb{E}\left[X_1|X_0 = x\right] + \dot{\sigma}_0 x$ as $t \to 0$, and $u_1(x) = \dot{\alpha}_1 x + \dot{\sigma}_1 \mathbb{E}\left[X_0|X_1 = x\right]$ as $t \to 1$. These can be computed in many cases of interest. Returning to our example $\pi_{0,1}(x_0,x_1) = \mathcal{N}(x_0|0,I)q(x_1)$

and assuming $\mathbb{E}_{X_1} X_1 = 0$, it follows that $u_0(x) = \dot{\sigma}_0 x$ and $u_1(x) = \dot{\alpha}_1 x$. These expressions can be used to fix singularities when converting from $x_{1|t}$ and $x_{0|t}$ to $u_t(x)$ as $t \to 1$ or $t \to 0$, respectively.

4.8.2 Post-training velocity scheduler change

Affine conditional flows admit a closed-form transformation from a marginal velocity field $u_t(x)$, based on a scheduler (α_t, σ_t) and an arbitrary data coupling $\pi_{0,1}$, to a marginal velocity field $\bar{u}_r(x)$, based on a different scheduler $(\bar{\alpha}_r, \bar{\sigma}_r)$ and the same data coupling $\pi_{0,1}$. Such a transformation is useful to adapt a trained velocity field to a different scheduler, potentially improving sample efficiency and quality generation (Karras et al., 2022; Shaul et al., 2023b; Pokle et al., 2023). To proceed, define the *scale-time (ST) transformation* (s_r, t_r) between the two conditional flows:

$$\bar{\psi}_r(x_0|x_1) = s_r \psi_{t_r}(x_0|x_1), \tag{4.65}$$

where $\psi_t(x_0|x_1) = \alpha_t x_1 + \sigma_t x_0$, $\bar{\psi}_r(x_0|x_1) = \bar{\alpha}_r x_1 + \bar{\sigma}_r x_0$, and $s, t : [0,1] \to \mathbb{R}_{\geq 0}$ are time-scale reparametrizations. Solving (4.65) yields

$$\begin{aligned} t_r &= \rho^{-1}(\bar{\rho}(r)) \\ s_r &= \bar{\sigma}_r / \sigma_{t_r}, \end{aligned} \tag{4.66}$$

where we define the signal-to-noise ratio by

$$\begin{aligned} \rho(t) &= \frac{\alpha_t}{\sigma_t} \\ \bar{\rho}(t) &= \frac{\bar{\alpha}_t}{\bar{\sigma}_t}, \end{aligned} \tag{4.67}$$

assumed to be an invertible function. The marginal velocity $\bar{u}_r(x)$ for the new scheduler $(\bar{\alpha}_r, \bar{\sigma}_r)$ follows the expression

$$\begin{aligned} \bar{u}_r(x) &= \mathbb{E}\left[\dot{\bar{X}}_r \middle| \bar{X}_r = x\right] \\ &\overset{(4.65)}{=} \mathbb{E}\left[\dot{s}_r X_{t_r} + s_r \dot{X}_{t_r} \dot{t}_r \middle| s_r X_{t_r} = x\right] \\ &= \dot{s}_r \mathbb{E}\left[X_{t_r} \middle| X_{t_r} = \frac{x}{s_r}\right] + s_r \dot{t}_r \mathbb{E}\left[\dot{X}_{t_r} \middle| X_{t_r} = \frac{x}{s_r}\right] \\ &= \frac{\dot{s}_r}{s_r} x + s_r \dot{t}_r u_{t_r}\left(\frac{x}{s_r}\right), \end{aligned}$$

where as before $\bar{X}_r = \bar{\psi}_r(X_0|X_1)$ and $X_t = \psi_t(X_0|X_1)$. This last term can be used to change a schedular post-training. Code 7 shows how to change the scheduler of a velocity field trained with a variance preserving schedule to the conditional Optimal Transport schedule using the `flow_matching` library.

Equivalence of schedulers. One additional important consequence of the above formula is that all schedulers *theoretically* lead to the *same sampling* at time $t = 1$ (Shaul et al., 2023a). That is,

$$\bar{\psi}_1(x_0) = \psi_1(x_0), \text{ for all } x_0 \in \mathbb{R}^d. \tag{4.68}$$

To see that, denote $\bar{\psi}_r(x)$ the flow defined by $\bar{u}_t(x)$, and differentiate $\tilde{\psi}_r(x) \coloneqq s_r \psi_{t_r}(x)$ w.r.t. r and note that it also satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \tilde{\psi}_r(x) = \bar{u}_r(\tilde{\psi}_r(x)). \tag{4.69}$$

Therefore, from uniqueness of ODE solutions we have that $\bar{\psi}_r(x) = \tilde{\psi}_r(x) = s_r \psi_{t_r}(x)$. Now, to avoid dealing with infinite signal-to-noise ratio assume the schedulers satisfy $\sigma_1 = \epsilon = \bar{\sigma}_1$ for arbitrary $\epsilon > 0$ (in addition to (4.51)), then for $r = 1$ we have $t_1 = 1$ and $s_1 = 1$ and therefore equation (4.68) holds.

Code 7: Post-training scheduler change

```python
import torch
from flow_matching.path import AffineProbPath
from flow_matching.path.scheduler import ScheduleTransformedModel, CondOTScheduler, VPScheduler
from flow_matching.solver import ODESolver
from flow_matching.utils import ModelWrapper

training_scheduler = VPScheduler()  # Variance preserving schedule
path = AffineProbPath(scheduler=training_scheduler)
velocity_model: ModelWrapper = ...  # Train a velocity model with the variance preserving schedule

# Change the scheduler from variance preserving to conditional OT schedule
sampling_scheduler = CondOTScheduler()
transformed_model = ScheduleTransformedModel(
    velocity_model=velocity_model,
    original_scheduler=training_scheduler,
    new_scheduler=sampling_scheduler
)

# Sample the transformed model with the conditional OT schedule
solver = ODESolver(velocity_model=transformed_model)
x_0 = torch.randn(batch_size, *data_dim)  # Specify the initial condition
solver = ODESolver(velocity_model=velocity_model)
num_steps = 100
x_1 = solver.sample(x_init=x_0, method='midpoint', step_size=1.0 / num_steps)
```

4.8.3 Gaussian paths

At the time of writing, the most popular class of affine probability paths is instantiated by the independent coupling $\pi_{0,1}(x_0, x_1) = p(x_0)q(x_1)$ and a Gaussian source distribution $p(x) = \mathcal{N}(x|0, \sigma^2 I)$. Because Gaussians are invariant to affine transformations, the resulting conditional probability paths take form

$$p_{t|1}(x|x_1) = \mathcal{N}(x|\alpha_t x_1, \sigma_t^2 I). \tag{4.70}$$

This case subsumes probability paths generated by standard diffusion models (although in diffusion the generation is stochastic and follows an SDE, it has the same marginal probabilities). Two examples are the **Variance Preserving (VP)** and **Variance Exploding (VE)** paths (Song et al., 2021), defined by choosing the following schedulers:

$$\alpha_t \equiv 1, \sigma_0 \gg 1, \sigma_1 = 0; \tag{VP}$$

$$\alpha_t = e^{-\frac{1}{2}\beta_t}, \sigma_t = \sqrt{1 - e^{-\beta_t}}, \beta_0 \gg 1, \beta_1 = 0. \tag{VE}$$

In the previous equations, "$\gg 1$" requires a sufficiently large scalar such that $p_0(x) = \int p_{0|1}(x|x_1)q(x_1)\mathrm{d}x_1$ is close to a known Gaussian distribution for $t = 0$—that is, the Gaussian $\mathcal{N}(\cdot|0, \sigma_0^2 I)$ for VE, and $\mathcal{N}(\cdot|0, I)$ for VP. Note that in both cases, $p_t(x)$ does not exactly reproduce p at $t = 0$, in contrast to the FM paths in (4.51).

One useful quantity admitting a simple form in the Gaussian case is the **score**, defined as the gradient of the log probability. Specifically, the score of the conditional path in (4.70) follows the expression

$$\nabla \log p_{t|1}(x|x_1) = -\frac{1}{\sigma_t^2}\left(x - \alpha_t x_1\right). \tag{4.71}$$

B \ A	velocity	x_1-prediction	x_0-prediction	score
velocity	$0, 1$	$\frac{\dot\sigma_t}{\sigma_t}, \frac{\dot\alpha_t\sigma_t - \dot\sigma_t\alpha_t}{\sigma_t}$	$\frac{\dot\alpha_t}{\alpha_t}, \frac{\dot\sigma_t\alpha_t - \dot\alpha_t\sigma_t}{\alpha_t}$	$\frac{\dot\alpha_t}{\alpha_t}, -\frac{\dot\sigma_t\sigma_t\alpha_t - \dot\alpha_t\sigma_t^2}{\alpha_t}$
x_1-prediction		$0, 1$	$\frac{1}{\alpha_t}, -\frac{\sigma_t}{\alpha_t}$	$\frac{1}{\alpha_t}, \frac{\sigma_t^2}{\alpha_t}$
x_0-prediction			$0, 1$	$0, -\sigma_t$
score				$0, 1$

Table 1 Conversion between different model parameterizations: (a_t, b_t) corresponds to $f_t^B(x) = a_t x + b_t f_t^A(x)$. The colors indicate the tranformation is relevant for all paths, Affine paths and Gaussian paths. The lower diagonal is computed from the upper diagonal using the inverse transformation: $f_t^A(x) = \frac{1}{b_t}\left(-a_t x + f_t^B(x)\right)$ which can be expressed as the pair $-\frac{a_t}{b_t}, \frac{1}{b_t}$. Note some of these conversions have singularities, as discussed at the end of Section 4.8.1.

The score of the corresponding *marginal* probability path (4.4) is

$$\nabla \log p_t(x) = \int \frac{\nabla p_{t|1}(x|x_1) q(x_1)}{p_t(x)} \mathrm{d}x_1 \tag{4.72}$$

$$= \int \nabla \log p_{t|1}(x|x_1) \frac{p_{t|1}(x|x_1) q(x_1)}{p_t(x)} \mathrm{d}x_1 \tag{4.73}$$

$$= \mathbb{E}\left[\nabla \log p_{t|1}(X_t|X_1) \,|\, X_t = x\right] \tag{4.74}$$

$$\overset{(4.71)}{=} \mathbb{E}\left[-\frac{1}{\sigma_t^2}\left(X_t - \alpha_t X_1\right) \,|\, X_t = x\right] \tag{4.75}$$

$$\overset{(4.54)}{=} \mathbb{E}\left[-\frac{1}{\sigma_t} X_0 \,\middle|\, X_t = x\right] \tag{4.76}$$

$$\overset{(4.59)}{=} -\frac{1}{\sigma_t} x_{0|t}(x), \tag{4.77}$$

where we borrow the notation $x_{0|t}$ from (4.59). The literature on diffusion refers to x_0-prediction ($x_{0|t}$) as *noise*-prediction, or ϵ-prediction. The formula above shows that the score is proportional to the x_0-prediction, and provides a conversion rule—for the Gaussian path case—from score to other parametrizations, as shown in table 1.

Kinetic optimality of marginal velocity. A consequence of the conversion formulas developed above (table 1) is that the marginal velocity for Gaussian paths can be written in the form

$$u_t(x) = \frac{\dot\alpha_t}{\alpha_t} x - \frac{\dot\sigma_t\sigma_t\alpha_t - \dot\alpha_t\sigma_t^2}{\alpha_t} \nabla \log p_t(x) \tag{4.78}$$

$$= \nabla\left[\frac{\dot\alpha_t}{2\alpha_t}\|x\|^2 - \frac{\dot\sigma_t\sigma_t\alpha_t - \dot\alpha_t\sigma_t^2}{\alpha_t} \log p_t(x)\right] \tag{4.79}$$

that shows $u_t(x)$ is a gradient and therefore Kinetic Optimal for the fixed marginalized Gaussian probability path $p_t(x)$ defined by $p_{t|1}(x|x_1) = \mathcal{N}(x|\alpha_t x_1, \sigma_t^2 I)$ (see *e.g.*, Villani (2021) Section 8.1.2, or Neklyudov et al. (2023) Theorem 2.1).

4.9 Data couplings

In developing the Flow Matching training algorithm, we have assumed we can draw samples $(X_0, X_1) \sim \pi_{0,1}(X_0, X_1)$ from some coupling $\pi_{0,1}$ (x_0, x_1) of the source p and target q distributions. For example, independent samples $\pi_{0,1}(x_0, x_1) = p(x_0)q(x_1)$, the simplest coupling preserving the marginal distributions p and q, or paired samples $(X_0, X_1) \sim \pi_{0,1}$ provided as part of the dataset. This section explores two examples of concrete couplings that can be used to train Flow Matching models.

4.9.1 Paired data

Dependent couplings arise naturally in learning tasks on paired data. Consider, for instance, the task of image in-painting, where q is a distribution of natural images, and p is the distribution of those same images with a square region masked-out. Rather than transforming noise into data, our goal here is to learn a mapping from masked-out images x_0 to their filled counterparts x_1. As this is an ill-defined problem—many filled images x_1 are compatible with each masked-out image x_0—solving this task can casted as learning to sample from the unknown, data-dependent coupling $\pi_{1|0}(x_1|x_0)$.

Based on these insights, Liu et al. (2023); Albergo et al. (2024) propose learning a *bridge* or flow model with data-dependent couplings, a simple modification enabling a new regime of applications. While the object of interest $\pi_{1|0}(x_1|x_0)$ is unavailable to sample, it is often the case that one can sample from the reverse dependency, $\pi_{0|1}(x_0|x_1)$. Returning to the example of image in-painting, it is easy to mask out a filled image $X_1 \sim q$ (target sample) to produce a source sample $X_0 \sim p$. To this end, specify

$$\pi_{0,1}(x_0, x_1) = \pi_{0|1}(x_0|x_1)q(x_1). \tag{4.80}$$

Thus, we can obtain a pair (X_0, X_1) by (i) drawing $X_1 \sim q$, and (ii) applying a predefined randomized transformation to obtain X_0 from X_1. To satisfy the conditions of corollary 1 (making sure the source is a density) and to encourage diversity, we add noise when sampling from $\pi_{0|1}(x_0|x_1)$. (Liu et al., 2023; Albergo et al., 2024) demonstrated the capability of this approach on various applications, such as image super-resolution, in-painting, and de-blurring, outperforming methods based on guided diffusion (Saharia et al., 2022).

4.9.2 Multisample couplings

As discussed in section 4.7, straight probability paths yield ODEs simulations with smaller errors. Therefore, it is natural ask: how could we change the training algorithm, so the learned velocity field induces straight(er) trajectories?

As hinted above, straight trajectories are related to the Optimal Transport (OT) problem. Specifically, consider a convex cost functional $c : \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ and the conditional OT flow $\psi_t(x_0|x_1) = tx_1 + (1-t)x_0$. Then, the transport cost of the coupling admits an upper bound on the marginal transport cost (Liu et al., 2022; Pooladian et al., 2023), that is:

$$\mathbb{E}\left[c(\psi_1(X_0) - X_0)\right] \leq \mathbb{E}\left[c(X_1 - X_0)\right], \tag{4.81}$$

where the case of $c(x) = \|x\|^2$ can be understood from the bound in (4.49) after plugging the OT solution in the l.h.s. that satisfies $u_t(X_t) = u_t(\psi_t(x)) = \phi(x) - x$, where ϕ is the OT map. Therefore, one could construct low-cost marginal transport maps by reducing the coupling cost. To this end, Pooladian et al. (2023) propose **multisample couplings**, a process to implicitly construct non-trivial joints $\pi_{0,1}(x_0, x_1)$ introducing dependencies between source and target distributions:

1. Sample $X_0^{(i)} \sim p$ and $X_1^{(i)} \sim q$, $i \in [k]$ independently.
2. Construct $\pi^k \in B_k$ by $\pi^k \coloneqq \arg\min_{\pi \in B_k} \mathbb{E}_\pi \left[c(X_0^{(i)} - X_1^{(j)})\right]$.
3. Sample a pair $(X_0^{(i)}, X_0^{(j)})$ uniformly at random from $(X_0^{(i)}, X_1^{(j)})$ for which $\pi^k(i,j) = 1$.

where B_k is the polytope of $k \times k$ doubly stochastic matrices.

The process above implicitly defines a joint distribution $\pi_{0,1}^k(x_0, x_1)$ by means of sampling. This implicit joint preserves the marginals and obeys an optimality constraint (step 2) (Pooladian et al., 2023). For $k = 1$, the method reduces to independent couplings. For $k > 1$, Pooladian et al. (2023) show that the transport cost is reduced compared to independent couplings, that is, $\mathbb{E}_{(x_0,x_1)\sim\pi_{0,1}^k(x_0,x_1)}\left[c(x_1 - x_0)\right] \leq \mathbb{E}_{X_0\sim p, X_1 \sim q}\left[c(X_1 - X_0)\right]$. Furthermore, for the quadratic cost function, multisample couplings approach the Optimal Transport cost and induces straight trajectories as $k \to \infty$ (Pooladian et al., 2023; Tong et al., 2023).

4.10 Conditional generation and guidance

We now consider training a generative model under a guiding signal to further control the produced samples. This technique has proved valuable in numerous practical applications, such as image-to-image translation (Saharia et al., 2022) and text-to-image generation (Nichol et al., 2022; Esser et al., 2024). In this subsection, we assume access to *labeled* target samples (x_1, y), where $y \in \mathcal{Y} \subseteq \mathbb{R}^k$ is a label or guidance variable.

4.10.1 Conditional models

One natural way to train a generative model under guidance is to learn to sample from the conditional distribution $q(x_1|y)$, as demonstrated by both diffusion and FM models (Zheng et al., 2023). Following the FM blueprint in figure 2, consider samples from the conditional target distribution $q(x_1|y)$ and prescribe a simple—typically but not necessarily Gaussian—source distribution p. Next, design a **guided probability path** as the aggregation of conditional probability paths:

$$p_{t|Y}(x|y) = \int p_{t|1}(x|x_1)q(x_1|y)\mathrm{d}x_1. \tag{4.82}$$

where we assume $p_{t,1|Y}(x, x_1|y) = p_{t|1}(x|x_1)q(x_1|y)$, meaning that the conditional path does not depend on Y. The resulting guided probability path is conditioned on the guidance variable $Y \sim p_Y$, and satisfies the marginal endpoints

$$p_{0|Y}(\cdot|y) = p(\cdot), \quad p_{1|Y}(\cdot|y) = q(\cdot|y). \tag{4.83}$$

The **guided velocity field** takes form

$$u_t(x|y) = \int u_t(x|x_1)p_{1|t,Y}(x_1|x, y)\mathrm{d}x_1, \tag{4.84}$$

where, by Bayes' Rule, it follows

$$p_{1|t,Y}(x_1|x, y) = \frac{p_{t|1}(x|x_1)q(x_1|y)}{p_{t|Y}(x|y)}. \tag{4.85}$$

To show that $u_t(x|y)$ generates $p_{t|Y}(x|y)$, plug (4.82) and (4.84) into (4.14), and realize that the FM/CFM losses remain unchanged for the guided case, and enable the same steps appearing in the proof of theorem 4. In practice, we train a single neural network $u_t^\theta : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}^d$ to model the guided marginal velocity field for all values of y. Then, the guided version of the CFM loss (4.32) follows the expression

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,(X_0,X_1,Y)\sim\pi_{0,1,Y}} D\left(\dot{\psi}_t(X_0|X_1), u_t^\theta(X_t|Y)\right). \tag{4.86}$$

In practice, the literature in diffusion models shows that guidance is most effective in applications where a large amount of target samples X_1 share the same guiding signal Y, such as in class guidance (Nichol and Dhariwal, 2021). However, guiding is more challenging in settings where the guidance variable Y is non-repeating and complex, such as image captions.

4.10.2 Classifier guidance and classifier-free guidance

For flows trained with Gaussian paths, classifier guidance (Song et al., 2021; Dhariwal and Nichol, 2021) and classifier-free guidance (Ho and Salimans, 2021) can be applied utilizing the transformations between velocity fields and score functions for conditional distributions shown in table 1 (Zheng et al., 2023):

$$u_t(x|y) = a_t x + b_t \nabla \log p_{t|Y}(x|y). \tag{4.87}$$

Using Bayes' rule over the guided probability path yields

$$p_{t|Y}(x|y) = \frac{p_{Y|t}(y|x)p_t(x)}{p_Y(y)}. \tag{4.88}$$

Taking logarithms and gradients with respect to x, $\nabla = \nabla_x$, we arrive at the fundamental relation between the scores of the probability path $p_t(x)$ and its guided counterpart $p_{t|Y}(x|y)$:

$$\overbrace{\nabla \log p_{t|Y}(x|y)}^{\text{conditional score}} = \nabla \overbrace{\log p_{Y|t}(y|x)}^{\text{classifier}} + \overbrace{\nabla \log p_t(x)}^{\text{unconditional score}}. \tag{4.89}$$

Namely, the two are related by means of the score of a classifier model $p_{Y|t}(y|x)$ attempting to predict the guidance variable y given a sample x.

Based on this relation, Song et al. (2021) propose **classifier guidance**, that is sampling from the conditional model $q(x_1|y)$ by guiding an unconditional model (parameterized with $\nabla \log p_t(x)$) with a time-dependent classifier (predicting the guiding variable y given $x \sim p_t(x)$). The corresponding velocity field then translates to:

$$\tilde{u}_t^{\theta,\phi}(x|y) = a_t x + b_t \left(\nabla \log p_{Y|t}^{\phi}(y|x) + \nabla \log p_t^{\theta}(x) \right) = u_t^{\theta}(x) + b_t \nabla \log p_{Y|t}^{\phi}(y|x), \tag{4.90}$$

where $u_t^{\theta}(x)$ is a velocity field trained on the unconditional target $q(x)$, and $\log p_{Y|t}^{\phi}(y|x)$ is a time-dependent classifier with parameters $\phi \in \mathbb{R}^m$. Dhariwal and Nichol (2021) show that this approach outperforms the conditional model from section 4.10.1 for both class- and text-conditioning (Nichol et al., 2022). In practice, because the classifier and the unconditional score are learned separately, it is often necessary to calibrate the classifier guidance as

$$\tilde{u}_t^{\theta,\phi}(x|y) = u_t^{\theta}(x) + b_t w \nabla \log p_{Y|t}^{\phi}(y|x), \tag{4.91}$$

where $w \in \mathbb{R}$ is the classifier scale, typically chosen to be $w > 1$ (Dhariwal and Nichol, 2021).

In a later work, (Ho and Salimans, 2021) propose a pure generative approach called **classifier-free guidance**. By simply re-arranging (4.89), we obtain

$$\nabla \overbrace{\log p_{Y|t}(y|x)}^{\text{classifier}} = \overbrace{\nabla \log p_{t|Y}(x|y)}^{\text{conditional score}} - \overbrace{\nabla \log p_t(x)}^{\text{unconditional score}}, \tag{4.92}$$

revealing that the score of the classifier can be implicitly approximated by the difference between the scores of the vanilla and guided probability paths. Then, the authors propose to learn the conditional and unconditional scores simultaneously using the same model. In terms of velocities, Zheng et al. (2023) show one can also plug 4.92 into 4.91 and use the conversion from scores to velocities as in table 1 to get:

$$\tilde{u}_t^{\theta}(x|y) = (1-w) u_t^{\theta}(x|\emptyset) + w u_t^{\theta}(x|y), \tag{4.93}$$

where w is once again the guidance calibration scale. Now, only a single model is trained, $u_t^{\theta}(x|y)$, where $y \in \{\mathcal{Y}, \emptyset\}$, $\emptyset$ is a place-holder value denoting the null-condition, and $u_t^{\theta}(x|\emptyset)$ is the velocity field generating the unconditional probability path $p_t(x)$. The resulting loss reads:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,\xi,(X_0,X_1,Y)\sim\pi_{0,1,Y}} \left[D\left(\dot{\psi}_t(X_0|X_1), u_t^{\theta}(X_t|(1-\xi)\cdot Y + \xi \cdot \emptyset) \right) \right], \tag{4.94}$$

where $\xi \sim \text{Bernoulli}(p_{\text{uncond}})$, and p_{uncond} is the probability of drawing the null condition $\emptyset$ during training. The exact distribution which CFG samples from is unknown, with some works proposing different intuitive or theoretical justifications for CFG sampling (Dieleman, 2022; Guo et al., 2024; Chidambaram et al., 2024; Bradley and Nakkiran, 2024). Despite this, at the time of writing, CFG is the most popular approach to training a conditional model. Esser et al. (2024); Polyak et al. (2024) show the application of classifier-free guidance to train large-scale guided FM models.

5 Non-Euclidean Flow Matching

This section extends Flow Matching from Euclidean spaces $\mathbb{R}^d$ to general *Riemannian manifolds* $\mathcal{M}$. Informally, Riemannian manifolds are spaces behaving locally like Euclidean spaces, and are equipped with a generalized notion of distances and angles. Riemannian manifolds are useful to model various types of data. For example, probabilities of natural phenomena on Earth can be modeled on the sphere Mathieu and Nickel (2020), and protein backbones are often parameterized inn terms of matrix Lie groups Jumper et al. (2021). The extension of flows to Riemannian manifolds is due to Mathieu and Nickel (2020); Lou et al. (2020). However, their original training algorithms required expensive ODE simulations. Following Chen and Lipman (2024), the Flow Matching solutions in this section provide a scalable, simulation-free training algorithm to learn generative models on Riemannian manifolds.

5.1 Riemannian manifolds

We consider complete connected, smooth Riemannian manifolds $\mathcal{M}$ with a metric g. The tangent space at point $x \in \mathcal{M}$, a vector space containing all tangent vectors to $\mathcal{M}$ at x, is denoted with $T_x\mathcal{M}$. The Riemannian metric g defines an inner product over $T_x\mathcal{M}$ denoted by $\langle u, v \rangle_g$, for $u, v \in T_x\mathcal{M}$. Let $T\mathcal{M} = \cup_{x\in\mathcal{M}} \{x\} \times T_x\mathcal{M}$ be the tangent bundle that collects all the tangent planes of the manifold. In the following, vector fields defined on tangent spaces are important objects to build flows on manifolds with velocity fields. We denote by $\mathcal{U} = \{u_t\}$ the space of time-dependent smooth vector fields (VFs) $u_t : [0, 1] \times \mathcal{M} \to T\mathcal{M}$, where $u_t(x) \in T_x\mathcal{M}$ for all $x \in \mathcal{M}$. Also, $\mathrm{div}_g(u_t)$ is the Riemannian divergence with respect to the spatial (x) argument. Finally, we denote by dvol_x the volume element over $\mathcal{M}$, and integration of a function $f : \mathcal{M} \to \mathbb{R}$ over $\mathcal{M}$ is denoted $\int f(x)\mathrm{dvol}_x$.

5.2 Probabilities, flows and velocities on manifolds

Probability density functions over a manifold $\mathcal{M}$ are continuous non-negative functions $p : \mathcal{M} \to \mathbb{R}_{\geq 0}$ integrating to 1, namely $\int_{\mathcal{M}} p(x)\mathrm{dvol}_x = 1$. We define a probability path in time p_t as a time-dependent curve in probability space $\mathcal{P}$, namely $p_t : [0, 1] \to \mathcal{P}$. A time-dependent flow, $\psi : [0, 1] \times \mathcal{M} \to \mathcal{M}$, similar to the Euclidean space, defines a global diffeomorphism on $\mathcal{M}$ for every t.

Remarkably, constructing flow-based models via velocity fields naturally applies to general Riemannian manifolds. Formally, and rephrasing Proposition 1 from Mathieu and Nickel (2020):

Theorem 8 (Flow local existence and uniqueness)**.** *Let $\mathcal{M}$ a smooth complete manifold and a velocity field $u_t \in \mathcal{U}$. If u is $C^\infty([0, 1] \times \mathcal{M}, T\mathcal{M})$ (in particular, locally Lipschitz), then the ODE in (3.19) has a unique solution which is a $C^\infty(\Omega, \mathcal{M})$ diffeomorphism $\psi_t(x)$ defined over the open set $\Omega \supset \{0\} \times \mathcal{M}$.*

Similar to theorem 1, flow ODEs generally only define a local diffeomorphism on the manifold, meaning that $\psi_t(x)$ may be defined on a maximal interval in time $[0, t_x])$ for different values of $x \in \mathcal{M}$. Similar to the Euclidean case we will work with the semi-open time interval $t \in [0, 1)$ to allow q to have compact support (for which u_t is not everywhere defined). To ensure existence for the desired time interval, $[0, 1)$, we add the integrability constraint (see theorem 2) and rely on the Mass Conservation theorem once again. For a Riemannian manifold with metric g, the **Riemannian continuity equation** reads

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) + \mathrm{div}_g(p_t u_t)(x) = 0, \tag{5.1}$$

and the corresponding Manifold Mass Conservation theorem (Villani et al., 2009) is stated as follows.

Theorem 9 (Manifold Mass Conservation)**.** *Let p_t be a probability path and $u_t \in \mathcal{U}$ a locally Lipchitz integrable vector field over a Riemannian manifold $\mathcal{M}$ with metric g. Then the following are equivalent*

1. *The Continuity Equation (5.1) holds for $t \in [0, 1)$.*
2. *u_t generates p_t in the sense of 3.24.*

In the previous result, by integrable u we mean

$$\int_0^1 \int_{\mathcal{M}} \|u_t(x)\| \, p_t(x) \mathrm{dvol}_x \mathrm{d}t < \infty. \tag{5.2}$$

Note that the assumptions of theorem 9 yield a global diffeomorphism on $\mathcal{M}$, giving rise to the **Riemannian instantaneous change of variables** formula:

$$\frac{\mathrm{d}}{\mathrm{d}t} \log p_t(\psi_t(x)) = -\mathrm{div}_g(u_t)(\psi_t(x)). \tag{5.3}$$

Finally, we say that u_t generates p_t from p if

$$X_t = \psi_t(X_0) \sim p_t \text{ for } X_0 \sim p. \tag{5.4}$$

Having positioned flows as valid generative models on manifolds, it stands to reason that the FM principles can be transferred to this domain as well. In the Riemannian version of FM we aim to find a velocity field $u_t^\theta \in \mathcal{U}$ generating a target probability path $p_t : [0,1] \to \mathcal{P}$ with marginal constraints $p_0 = p$ and $p_1 = q$, where p, q denote the source and target distributions over the manifold $\mathcal{M}$. As the velocity field lies on the tangent spaces of the manifold, the **Riemannian Flow Matching loss** compares velocities using a Bregman divergence defined over the individual tangent planes of the manifold,

$$\mathcal{L}_{\mathrm{RFM}}(\theta) = \mathbb{E}_{t, X_t \sim p_t} D_{X_t} \left(u_t(X_t), u_t^\theta(X_t) \right). \tag{5.5}$$

In the equation above, the expectation is now an integral over the manifold, that is $E[f(X)] = \int_{\mathcal{M}} f(x) p_X(x) \mathrm{dvol}_x$, for a smooth function $f : \mathcal{M} \to \mathcal{M}$ and a random variable $X \sim p_X$. The Bregman divergences, D_x, $x \in \mathcal{M}$, are potentially defined with the Riemannian inner product and a strictly convex function assigned to each tangent space $\Phi_x : T_x\mathcal{M} \to T_x\mathcal{M}$, that is, $D_x(u, v) \coloneqq \Phi_x(u) - [\Phi_x(v) + \langle u - v, \nabla_v \Phi_x(v) \rangle_g]$. For example, choosing the Riemannian metric $\Phi_x = \|\cdot\|_g^2$ then $D_x(u, v) = \|u - v\|_g^2$ for $u, v \in T_x\mathcal{M}$.

5.3 Probability paths on manifolds

Marginal probability paths are built as in the Euclidean case (4.4):

$$p_t(x) = \int_{\mathcal{M}} p_t(x|x_1) q(x_1) \mathrm{dvol}_{x_1}, \tag{5.6}$$

where $p_{t|1}(x|x_1)$ is the **conditional probability path** defined on the manifold. We also require the boundary constraints

$$p_0 = p, \quad p_1 = q. \tag{5.7}$$

For instance, these constraints can be implemented by requiring the conditional path $p_{t|1}(x|x_1)$ to satisfy

$$p_{0|1}(x|x_1) = \pi_{0|1}(x|x_1), \text{ and } p_{1|1}(x|x_1) = \delta_{x_1}(x), \tag{5.8}$$

where $\pi_{0|1}$ is the conditional coupling, $\pi_{0|1}(x_0|x_1) = \pi_{0,1}(x_0, x_1)/q(x_1)$.

5.4 The Marginalization Trick for manifolds

The Marginalization Trick for the marginal velocity field (theorem 3) readily applies to the Riemannian case. Consider the **conditional velocity field** $u_t(x|x_1) \in \mathcal{U}$ such that

$$u_t(\cdot|x_1) \text{ generates } p_{t|1}(\cdot|x_1). \tag{5.9}$$

Then, the **marginal velocity** field $u_t(x)$ is given by the following averaging of the conditional velocities,

$$u_t(x) = \int_{\mathcal{M}} u_t(x|x_1) p_{1|t}(x_1|x) \mathrm{dvol}_{x_1}, \tag{5.10}$$

where, by Bayes' Rule for PDFs, we obtain

$$p_{1|t}(x_1|x) = \frac{p_{t|1}(x|x_1)q(x_1)}{p_t(x)}, \tag{5.11}$$

which is defined for all $x \in \mathcal{M}$ for which $p_t(x) > 0$.

The Marginalization Trick (theorem 3) for the Riemannian case requires adjusting assumption 1 as follows:

Assumption 2. $p_{t|1}(x|x_1)$ is $C^\infty([0,1) \times \mathcal{M})$ and $u_t(x|x_1)$ is $C^\infty([0,1) \times \mathcal{M}, \mathcal{M})$ as function of (t, x). Furthermore, we assume either q has bounded support, *i.e.*, $q(x_1) = 0$ outside some bounded set or $\mathcal{M}$ is compact; and $p_t(x) > 0$ for all $x \in \mathcal{M}$ and $t \in [0, 1)$.

We are now ready to state the **Manifold Marginalization Trick** theorem:

Theorem 10 (Manifold Marginalization Trick)**.** *Under Assumption 2, if $u_t(x|x_1)$ is conditionally integrable and generates the conditional probability path $p_t(\cdot|x_1)$ then the marginal velocity field $u_t(\cdot)$ generates the marginal probability path $p_t(\cdot)$.*

By **conditionally integrable**, we mean a conditioned version of the integrability condition from the Mass Conservation Theorem (5.2):

$$\int_0^1 \int_{\mathcal{M}} \int_{\mathcal{M}} \|u_t(x|x_1)\|_g \, p_{t|1}(x|x_1)q(x_1)\mathrm{dvol}_{x_1}\mathrm{dvol}_x\mathrm{d}t < \infty \tag{5.12}$$

The proof of theorem 10 is repeating the arguments of theorem 3 and is given in appendix A.2.

5.5 Riemannian Flow Matching loss

The **Riemannian Conditional Flow Matching (RCFM) loss** reads

$$\mathcal{L}_{\text{RCFM}}(\theta) = \mathbb{E}_{t, X_1, X_t \sim p_{t|1}(\cdot|X_1)} D_{X_t}\left(u_t(X_t|X_1), u_t^\theta(X_t)\right). \tag{5.13}$$

Once again, we have the equivalence:

Theorem 11. *The gradients of the Riemannian Flow Matching loss and the Riemannian Conditional Flow Matching loss coincide:*

$$\nabla_\theta \mathcal{L}_{RFM}(\theta) = \nabla_\theta \mathcal{L}_{RCFM}(\theta). \tag{5.14}$$

The above theorem can be proved using proposition 1 with $X = X_t$, $Y = u_t(X_t|X_1)$, $g^\theta(x) = u_t^\theta(x)$, and integrating w.r.t. $t \in [0, 1]$.

5.6 Conditional flows through premetrics

Having established how to learn a flow model with the RCFM loss, we are left with specifying the conditional probability path and its generating velocity field. Similar to section 4.6, we begin by stating the requirements for the corresponding conditional flow $\psi : [0, 1) \times \mathcal{M} \times \mathcal{M} \to \mathcal{M}$, such that $p_{t|1}(\cdot|x_1)$ satisfies the boundary conditions (5.8). The **conditional flow model** is

$$X_{t|1} = \psi_t(X_0|x_1), \qquad \text{where } X_0 \sim \pi_{0|1}(\cdot|x_1), \tag{5.15}$$

where the **conditional flow** is

$$\psi_t(x|x_1) = \begin{cases} x & t = 0 \\ x_1 & t = 1 \end{cases}, \quad \text{is smooth in } t, x \text{ and diffeomorphism in } x \text{ on } \mathcal{M}. \tag{5.16}$$

Our analysis in Euclidean space focused on affine conditional flows, as these served as a rich class of easily computable (simulation-free) conditional flows. Unfortunately, combinations $\alpha_t x_1 + \sigma_t x_0$ for $\alpha_t + \sigma_t \neq 1$ are

not naturally defined on manifolds. The manifold analog for the case $\alpha_t + \sigma_t = 1$ would be using geodesic interpolation. Indeed, Chen and Lipman (2024) proposed building conditional flows by moving along geodesic curves, in particular, generalizing the conditional OT paths moving along straight lines in Euclidean space (see theorem 5). Geodesics represent the shortest paths between two points on a manifold, reducing to straight lines in Euclidean spaces. For manifolds, we define the **geodesic conditional flow** as

$$\psi_t(x_0|x_1) = \exp_{x_0}(\kappa(t)\log_{x_0}(x_1)), \quad t \in [0,1], \tag{5.17}$$

where $\kappa(t) : [0,1] \to [0,1]$ is a monotonically increasing scheduler satisfying $\kappa(0) = 0$ and $\kappa(1) = 1$, making sure all x_0 are pushed to x_1 at $t = 1$. The exponential map, evaluated at $x \in \mathcal{M}$, $\exp_x : T_x\mathcal{M} \to \mathcal{M}$, $v \mapsto \exp_x(v)$, returns the endpoint at time $t = 1$ of the unique geodesic starting at x with initial speed v. The logarithmic map $\log_x : \mathcal{M} \to T_xM$, $y \mapsto \log_x(y)$, is the inverse of the exponential map. In Euclidean space, the exponential map is simply vector addition, and the logarithmic map is vector subtraction. Now, if we plug these in (5.17), we get $\psi_t(x_0|x_1) = x_0 + \kappa(t)(x_1 - x_0)$, and by choosing $\kappa(t) = t$ we recover the conditional OT flow.

For simple manifolds with closed-form exponential and logarithmic maps, this construction allows a simulation-free recipe for training flows on manifolds, an arguably clear advantage compared to diffusion models approaches built on manifolds (De Bortoli et al., 2022; Huang et al., 2022b; Lou et al., 2023). In particular, manifold diffusion models require in-training simulation to sample from p_t, and have to resort to approximations of the score function on the manifold.

Nevertheless, while building geodesic conditional flows is a natural construction, geodesics may be hard to compute for general manifolds that do not have closed-form exponential and logarithmic maps and/or introduce undesired bias such as concentrating probability at boundary points. To overcome the difficulty in computing geodesics and/or inject a desired implicit bias, one may seek an alternative notion of smooth distance function, $d(\cdot,\cdot) : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_{\geq 0}$, and require that the conditional flow satisfies

$$d(\psi_t(x_0|x_1), x_1) = \bar{\kappa}(t)d(x_0, x_1), \tag{5.18}$$

where $\bar{\kappa}(t) = 1 - \kappa(t)$. This will assure that the conditional flow concentrates all the probability at x_1 at time $t = 1$ if the following conditions hold:

1. *Non-negative*: $d(x, y) \geq 0$ for all $x, y \in \mathcal{M}$.
2. *Positive*: $d(x, y) = 0$ if and only if $x = y$.
3. *Non-degenerate*: $\nabla d(x, y) \neq 0$ if and only if $x \neq y$.

Chen and Lipman (2024) showed that the minimal norm conditional velocity field corresponding to a flow that satisfies (5.18) has the form:

$$u_t(x|x_1) = \frac{d\log\bar{\kappa}(t)}{\mathrm{d}t} d(x, x_1) \frac{\nabla d(x, x_1)}{\|\nabla d(x, x_1)\|_g^2}, \tag{5.19}$$

where the non-degeneracy requirement of the premetric ensures that the velocity field has no discontinuities, since $u_t(x|x_1) \propto 1/\|\nabla d(x, x_1)\|_g$. In particular, note that the geodesic conditional flow in (5.17) satisfies (5.18) for the choice $d = d_g$, where d_g is the geodesic distance. An example of a choice of alternative premetrics is using spectral distances on general geometries (Chen and Lipman, 2024), where the conditional velocity offers a way to sample from $p_t(x|x_1)$ by simulation. Importantly, although conditional flows with premetrics require in-training simulation—like diffusion models on manifolds—the velocity field can still be accurately recovered compared to approximations of the score function.

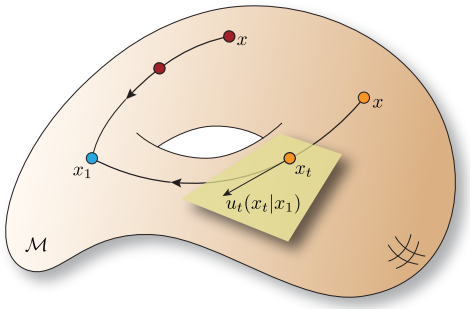


Figure 12 Conditional flows on the manifold $\mathcal{M}$.

Another issue, is that both conditional flows defined via geodesic interpolation and premetric can suffer from singularities, *e.g.*, for compact manifolds. For example on the 2-sphere the geodesic function $d(x, x_1)$ is not differentiable at the antipodal point $x = -x_1$. Furthermore, any smooth function such as $x \mapsto d(x, x_1)$ will showcase at-least two critical points (maximum and minimum) where the velocity in (5.19) is not-defined. However, the set of such problematic points is generally very small (in fact of zero volume usually). Therefore, this issue does not cause problems in practice, at-least in use cases we are aware of.

In any case, to deal with this issue, we can include an augmented scheduler in the geodesic conditional flow. That is, use $\bar{\kappa}(t, x, x_1)$, that depends also on x, x_1 to make (5.17) globally smooth. To deal with the zero gradient issue of the premetric conditional flow we can relax the non-degeneracy requirement as follows:

3. *Non-degenerate (relaxed)*: The volume of the set $\mathcal{A}_y = \{x \in \mathcal{M} \,|\, \nabla d(x, y) = 0 \text{ and } x \neq y\}$ is 0 for all $y \in \mathcal{M}$.

Code 8: Training with geodesic flows on a Sphere using the CFM objective

```python
import torch
from flow_matching.path import GeodesicProbPath, PathSample
from flow_matching.path.scheduler import CondOTScheduler
from flow_matching.utils.manifolds import Sphere

model = ...  # Define a trainable velocity model
optimizer = torch.optim.Adam(model.parameters())
loss_fn = torch.nn.MSELoss()  # Any Bregman divergence

manifold = Sphere()
scheduler = CondOTScheduler()
path = GeodesicProbPath(scheduler=scheduler, manifold=manifold)

for x_0, x_1 in dataloader:  # Samples from π_{0,1} of shape [batch_size, *data_dim]
    t = torch.rand(batch_size)  # Randomize time t ~ U[0, 1]
    sample: PathSample = path.sample(t=t, x_0=x_0, x_1=x_1)  # Sample the conditional path

    model_output = model(sample.x_t, sample.t)
    projected_model_output = manifold.proju(sample.x_t, model_output)  # Project to tangent space

    loss = loss_fn(projected_model_output, sample.dx_t)  # CFM loss

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

6 Continuous Time Markov Chain Models

This section presents the **Continuous Time Markov Chains (CTMCs)** as an alternative generative model to flow, with the use-case of generating discrete data, *i.e.*, data residing in a discrete (and finite) state space. CTMC are Markov processes that form the building blocks behind the generative model paradigm of Discrete Flow Matching (DFM) Campbell et al. (2024); Gat et al. (2024), later discussed in section 7. Therefore, this section is analogous to section 3, where we presented flows as the building blocks behind the generative model paradigm of Flow Matching (FM).

6.1 Discrete state spaces and random variables

Consider a finite version of $\mathbb{R}^d$ as our state space $\mathcal{S} = \mathcal{T}^d$, where $\mathcal{T} = [K] = \{1, 2, \dots, K\}$, sometimes called **vocabulary**. Samples and states are denoted by $x = (x^1, \dots, x^d) \in \mathcal{S}$, where $x^i \in \mathcal{T}$ is single coordinate or a **token**. We will similarly use states $y, z \in \mathcal{S}$. Next, X denotes a random variable taking values in the state space $\mathcal{S}$, with probabilities governed by the **probability mass function (PMF)** $p_X : \mathcal{S} \to \mathbb{R}_{\geq 0}$, such that $\sum_{x \in \mathcal{S}} p_X(x) = 1$, and the probability of an event $A \subset \mathcal{S}$ being

$$\mathbb{P}(X \in A) = \sum_{x \in A} p_X(x). \tag{6.1}$$

The notations $X \sim p_X$ or $X \sim p_X(X)$ indicate that X has the PMF p_X. The δ PMF in the discrete case is defined by

$$\delta(x, z) = \begin{cases} 1 & x = z, \\ 0 & \text{else.} \end{cases} \tag{6.2}$$

where we sometimes also define δ PMFs on tokens, such as in $\delta(x^i, y^i)$, for some $x^i, y^i \in \mathcal{T}$.

6.2 The CTMC generative model

The **CTMC model** is an $\mathcal{S}$-valued time-dependent family of random variables $(X_t)_{0 \leq t \leq 1}$ that a form a Markov chain characterized by the **probability transition kernel** $p_{t+h|t}$ defined via

$$p_{t+h|t}(y|x) \coloneqq \mathbb{P}(X_{t+h} = y | X_t = x) = \delta(y, x) + h u_t(y, x) + o(h), \text{ and } \mathbb{P}(X_0 = x) = p(x), \tag{6.3}$$

where the PMF p indicates the initial distribution of the process at time $t = 0$, and $o(h)$ is an arbitrary function satisfying $o(h)/h \to 0$ as $t \to 0$. The values $u_t(y, x)$, called **rates** or **velocities**, indicate the speed at which the probability transitions between states as a function of time. By fully characterized, we mean that all the joints $\mathbb{P}(X_{t_1} = x_1, \dots, X_{t_n} = x_n)$, for arbitrary $0 \leq t_1 < \cdots < t_n \leq 1$ and $x_i \in \mathcal{S}$, $i \in [n]$, are defined this way.

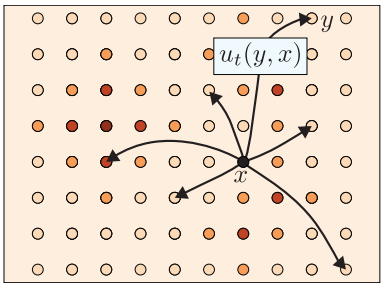


Figure 13 The CTMC model is defined by prescribing rates (velocities) of probability between states.

To make sure the transition probabilities $p_{t+h|t}(y|x)$ are defined via (6.3), velocities needs to satisfy the following **rate conditions**:

$$u_t(y, x) \geq 0 \text{ for all } y \neq x, \text{ and } \sum_y u_t(y, x) = 0. \tag{6.4}$$

If one of these conditions were to fail, then the transition probabilities $p_{t+h|t}(\cdot|x)$ would become negative or sum to $c \neq 1$ for arbitrary small $h > 0$. Equation (6.3) plays he same role as equation (3.16) and equation (3.19) when we were defining the flow generative modeling. The *marginal probability* of the process X_t is denoted by the PMF $p_t(x)$ for time $t \in [0, 1]$. Then, similarly to equation (3.24) for the case of flows, we say that

$$u_t \textbf{ generates } p_t \text{ if there exists } p_{t+h|t} \text{ satisfying (6.3) with marginals } p_t. \tag{6.5}$$

Simulating CTMC. To sample X_t, sample $X_0 \sim p$ and take steps using the **(naive) Euler method**:

$$\mathbb{P}(X_{t+h} = y \mid X_t) = \delta(y, X_t) + hu_t(y, X_t). \tag{6.6}$$

According to (6.3), these steps introduce $o(h)$ errors to the update probabilities. In practice, this means that we would need a sufficiently small $h > 0$ to ensure that the right-hand side in (6.6) remains a valid PMF. One possible remedy to assure that any choice of $h > 0$ results in a valid PMF, and maintains the $o(h)$ local error in probabilities is the following **Euler method**:

$$\mathbb{P}(X_{t+h} = y \mid X_t) = \begin{cases} \exp\left[hu_t(X_t, X_t)\right] & y = X_t \\ \frac{u_t(y, X_t)}{|u_t(X_t, X_t)|}\left(1 - \exp\left[hu_t(X_t, X_t)\right]\right) & y \neq X_t \end{cases}. \tag{6.7}$$

6.3 Probability paths and Kolmogorov Equation

Similarly to Continuity Equation in the continuous case, the marginal probabilities p_t of the CTMC model $(X_t)_{0 \leq t \leq 1}$ are characterized by the **Kolmogorov Equation**

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(y) = \sum_x u_t(y, x) p_t(x). \tag{6.8}$$

The following classical theorem (see also Theorems 5.1 and 5.2 in Coddington et al. (1956)) describes the existence of unique solutions for this linear homogeneous system of ODEs.

Theorem 12 (Linear ODE existence and uniqueness)**.** *If $u_t(y, x)$ are in $C([0, 1))$ (continuous with respect to time), then there exists a unique solution $p_t(x)$ to the Kolmogorov Equation (6.8), for $t \in [0, 1)$ and satisfying $p_0(x) = p(x)$.*

For the CTMC, the solution is guaranteed to exist for all times $t \in [0, 1)$ and no extra conditions are required (unlike the non-linear case in theorem 1). The Kolmogorov Equation has an intimate connection with the Continuity Equation (3.25). Rearranging the right-hand side of (6.8) by means of the rate conditions yields

$$\begin{aligned} \sum_x u_t(y, x) p_t(x) &\overset{(6.4)}{=} \overbrace{\sum_{x \neq y} u_t(y, x) p_t(x)}^{\text{incoming flux}} - \overbrace{\sum_{x \neq y} u_t(x, y) p_t(y)}^{\text{outgoing flux}} \\ &= -\sum_{x \neq y} \left[j_t(x, y) - j_t(y, x)\right], \end{aligned}$$

where $j_t(y, x) \coloneqq u_t(y, x) p_t(x)$ is the *probability flux* describing the probability of moving from state x to state y per unit of time. The excess of outgoing flux is defined as the **divergence**, giving the Kolmogorov Equation the same structure as the one described in section 3.5 for the Continuity Equation (Gat et al., 2024).

The following result is the main tool to build probability paths and velocities in the CTMC framework:

Theorem 13 (Discrete Mass Conservation)**.** *Let $u_t(y, x)$ be in $C([0, 1))$ and $p_t(x)$ a PMF in $C^1([0, 1))$ in time t. Then, the following are equivalent:*

1. *p_t, u_t satisfy the Kolmogorov Equation (6.8) for $t \in [0, 1)$, and u_t satisfies the rate conditions (6.4).*
2. *u_t generates p_t in the sense of 6.5 for $t \in [0, 1)$.*

The proof of theorem 13 is given in appendix A.1.

6.3.1 Probability preserving velocities

As a consequence of the Discrete Mass Conservation (theorem 13), if velocity $u_t(y, x)$ generates the probability path $p_t(x)$, then

$$\tilde{u}_t(y, x) = u_t(y, x) + v_t(y, x) \text{ generates } p_t(x), \tag{6.9}$$

as long as $v_t(y, x)$ satisfies the rate conditions (6.4) and solves the **divergence-free velocity** equation

$$\sum_x v_t(y, x)p_t(x) = 0. \tag{6.10}$$

In fact, $\tilde{u}_t(y, x)$ solves the Kolmogorov Equation:

$$\sum_x \tilde{u}_t(y, x)p_t(x) = \sum_x u_t(y, x)p_t(x) = \dot{p}_t(y),$$

showing that one may add divergence-free velocities during sampling without changing the marginal probability. This will be a useful fact when sampling from discrete Flow Matching models, described next.

7 Discrete Flow Matching

Remarkably, the Flow Matching blueprint in figure 2 carries out seamlessly from the continuous case to the discrete case, yielding the **Discrete Flow Matching (DFM)** framework (Campbell et al., 2024; Gat et al., 2024). In analogy to the continuous case, start by defining a probability path p_t interpolating between a source PMF p and a target PMF q. Second, we would like to find a CTMC model $(X_t)_{0\leq t\leq 1}$, defined by a learnable velocity u_t^θ, that generates the probability path p_t. Finally, we train u_t^θ by minimizing a Bregman divergence that defines the Discrete Flow Matching loss. In sum, this is to solve the discrete version of the Flow Matching problem (4.1).

7.1 Data and coupling

Our goal is to transfer samples $X_0 \sim p$ from a source PMF p to samples $X_1 \in q$ from a target PMF q, where $X_0, X_1 \in \mathcal{S}$ are two RVs each taking values in the state space $\mathcal{S}$. Source and target samples can be related by means of the independent coupling $(X_0, X_1) \sim p(X_0)q(X_1)$, or associate by means of a general PMF coupling $\pi_{0,1}(x_0, x_1)$. For example, text translation data considers coupled data (x_0, x_1) representing the same document written in two different languages. Another application, such as text generation, concerns independent pairing where $p(x_0)$ is either the uniform probability over $\mathcal{S}$ giving all states equal probability, or adding a special token $\mathtt{m}$ to the vocabulary $\mathcal{T}$, *i.e.*, $\mathcal{T} \cup \{\mathtt{m}\}$, and considering $\pi_{0,1}(x_0, x_1) = \delta(x_0, \mathtt{m})q(x_1)$. Any RV $X_0 \sim \delta(X_0, \mathtt{m})$ is the constant RV $X_0 = (\mathtt{m}, \dots, \mathtt{m})$.

7.2 Discrete probability paths

The next step in the FM recipe is, as usual, to prescribe a probability path p_t interpolating p and q. Following section 4.4, we condition these objects on a general conditioning RV $Z \sim p_Z$ taking values in some arbitrary space $\mathcal{Z}$. The **marginal probability path** takes form

$$p_t(x) = \sum_{z\in\mathcal{Z}} p_{t|Z}(x|z)p_Z(z), \tag{7.1}$$

where $p_{t|Z}(\cdot|z)$ is a conditional PMF, and the marginal probability path satisfies the boundary constraints $p_0 = p$ and $p_1 = q$.

7.3 The Marginalization Trick

The Marginalization Trick (see section 4.4) transfers to the discrete case as-is (Campbell et al., 2024; Gat et al., 2024). Assuming that the **conditional velocity field** $u_t(\cdot, \cdot|z)$ generates $p_t(\cdot|z)$ in the sense of (6.5), we obtain the **marginal velocity field**

$$u_t(y, x) = \sum_z u_t(y, x|z)p_{Z|t}(z|x) = \mathbb{E}\left[u_t(y, X_t|Z) \mid X_t = x\right], \tag{7.2}$$

defined for all $x, y \in \mathcal{S}$ where $p_t(x) > 0$, and RV $X_t \sim p_{t|Z}(\cdot|Z)$. By using Bayes' rule, we get

$$p_{Z|t}(z|x) = \frac{p_{t|Z}(x|z)p_Z(z)}{p_t(x)}. \tag{7.3}$$

To prove the discrete version of the Marginalization Trick Theorem (theorem 3), assume:

Assumption 3. $p_{t|Z}(x|z) \in C^1([0,1))$, $u_t(y,x|z) \in C([0,1))$, and $p_t(x) > 0$ for all $x \in \mathcal{S}$ and $t \in [0,1)$.

As it happened in the continuous case, the assumption $p_t > 0$ is in practice mild, because we can always use $(1-(1-t)\epsilon)\cdot p_{Z|t} + (1-t)\epsilon \cdot p_{\text{uni}}$, where p_{uni} is the uniform distribution over $\mathcal{S}$, and $\epsilon > 0$ is arbitrary small. We are no ready to state and prove the result.

Theorem 14 (Discrete Marginalization Trick)**.** *Under Assumption 3, if $u_t(y,x|z)$ generates $p_{t|Z}(x|z)$ then the marginal velocity $u_t(y,x)$ in (7.2) generates $p_t(x)$ in (7.1) for $t \in [0,1)$.*

Proof. The proof is conceptually similar to the continuous case. Start by computing:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}p_t(y) &= \sum_z \frac{\mathrm{d}}{\mathrm{d}t}p_{t|Z}(y|z)p_Z(z) \\
&\overset{(i)}{=} \sum_z \left[\sum_x u_t(y,x|z)p_{t|Z}(x|z)\right] p_Z(z) \\
&\overset{(ii)}{=} \sum_x \left[\sum_z u_t(y,x|z)\frac{p_{t|Z}(x|z)p_Z(z)}{p_t(x)}\right] p_t(x) \\
&\overset{(Bayes)}{=} \sum_x \overbrace{\sum_z u_t(y,x|z)p_{Z|t}(z|x)}^{u_t(y,x)} p_t(x),
\end{aligned}$$

Equality (i) follows from theorem 13 and the fact that $u_t(y,x|z)$ generates $p_{t|Z}(y|z)$. Equality (ii) follows from multiplying and dividing by $p_t(x)$ which is assumed positive. Therefore, $u_t(y,x)$ satisfies the Kolmogorov Equation with p_t. Also, $u_t(y,x)$ satisfies the rate conditions (6.4), because each $u_t(y,x|z)$ satisfies them. Lastly, $u_t(y,x) \in C([0,1))$ because both $u_t(y,x|z)$ and $p_{Z|t}(z|x)$ are in $C([0,1))$. In particular, $p_{Z|t}(z|x) \in C([0,1))$ follows from assuming $p_t(x) > 0$ for $t \in [0,1)$. By theorem 13, and because $u_t(x,y)$ satisfies the Kolmogorov Equation with p_t and the rate conditions, it generates p_t in the sense of (6.5). □

7.4 Discrete Flow Matching loss

To construct a CTMC generative model $(X_t)_{0\le t\le 1}$ we parameterize a velocity field $u_t^\theta(y,x)$ with parameters θ, *e.g.*, using a neural network. One would construct the neural network to satisfy the rate conditions equation (6.4). The **Discrete Flow Matching loss** to train the CTMC model is defined as:

$$\mathcal{L}_{\text{DFM}}(\theta) = \mathbb{E}_{t, X_t\sim p_t} D_{X_t}(u_t(\cdot, X_t), u_t^\theta(\cdot, X_t)), \tag{7.4}$$

for $t \sim U[0,1]$ and $u_t(\cdot,x) \in \mathbb{R}^{\mathcal{S}}$ satisfying the rate conditions. This means that $u_t(\cdot,x) \in \Omega_x$, where

$$\Omega_x = \left\{ v \in \mathbb{R}^{\mathcal{S}} \;\middle|\; v(y) \ge 0 \;\forall y \ne x, \text{ and } v(x) = -\sum_{y\ne x} v(y) \right\} \subset \mathbb{R}^{\mathcal{S}}, \tag{7.5}$$

is a convex set, and $D_x(u,v)$ is a Bregman divergence defined using a convex function $\Phi_x : \Omega_x \to \mathbb{R}$. The **Conditional Discrete Flow Matching loss** takes form

$$\mathcal{L}_{\text{CDFM}}(\theta) = \mathbb{E}_{t, Z, X_t\sim p_{t|Z}} D_{X_t}(u_t(\cdot, X_t|Z), u_t^\theta(\cdot, X_t)). \tag{7.6}$$

Once again, the two losses (7.4) and (7.6) both provide the same learning gradients.

Theorem 15. *The gradients of the Discrete Flow Matching loss and the Conditional Discrete Flow Matching loss coincide:*

$$\nabla_\theta \mathcal{L}_{DFM}(\theta) = \nabla_\theta \mathcal{L}_{CDFM}(\theta). \tag{7.7}$$

In particular, the minimizer of the Conditional Discrete Matching loss is the marginal velocity

$$u_t^\theta(y, x) = \mathbb{E}\left[u_t(y, X_t|Z) \mid X_t = x\right]. \tag{7.8}$$

The proof follows by applying proposition 1 when setting $X = X_t$, $Y = (X_t, Z)$, defining $f : \mathcal{S}^2 \to \mathbb{R}^{\mathcal{S}}$ as $(x, z) \mapsto u_t(\cdot, x|z) \in \mathbb{R}^{\mathcal{S}}$, and integrating with respect to $t \in [0, 1]$.

7.5 Factorized paths and velocities

If implementing DFM as presented, we would require a learnable model $u_t^\theta(y, x)$—for instance, a neural network—that outputs a rate for *all* possible states $y \in \mathcal{S} = \mathcal{T}^d$. This would result in a huge output dimension K^d, infeasible for common sequence lengths d and vocabulary sizes K. One remedy to this issue is to consider **factorized velocities** (Campbell et al., 2022),

$$u_t(y, x) = \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x), \tag{7.9}$$

where $\bar{i} = (1, \dots, i-1, i+1, \dots, d)$ denotes all indices excluding i. Therefore, the factorized velocity above connects state x to state y only if these differ in at most one single token. When using factorized velocities, we only require to model $u_t^i(y^i, x)$, as these fully define $u_t(y, x)$. In turn, each $u_t^i(y^i, x)$ is a learnable model accepting $x \in \mathcal{S}$ and returning a scalar $u_t^i(y^i, x) \in \mathbb{R}$, for all $i \in [d] = \{1, 2, \dots, d\}$ and $y^i \in \mathcal{T}$. Therefore, the output of the model has a tractable dimension $d \cdot K$. The rate conditions for factorized velocities $u_t^i(y, x)$ are now required per dimension $i \in [d]$:

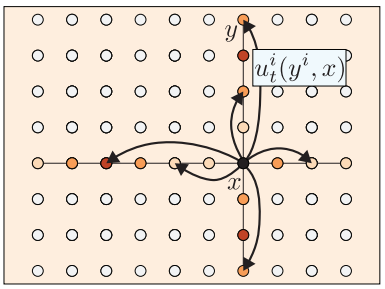


Figure 14 Factorized CTMC model allows non-zero rates (velocities) only between states that differ in at-most one coordinate (token).

$$u_t^i(y^i, x) \geq 0 \text{ for all } y^i \neq x^i, \text{ and } \sum_{y^i \in \mathcal{T}} u_t^i(y^i, x) = 0 \quad \text{for all } x \in \mathcal{S}. \tag{7.10}$$

7.5.1 Simulating CTMC with factorized velocities

When using factorized velocities, we can sample CTMC models coordinate-wise (Campbell et al., 2024):

$$\begin{aligned}\mathbb{P}(X_{t+h} = y \mid X_t = x) &= \delta(y, x) + h \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x) + o(h) \\ &= \prod_i \left[\delta(y^i, x^i) + h u_t^i(y^i, x) + o(h)\right],\end{aligned}$$

where the second equality follows from $\delta(y, x) = \prod_i \delta(y^i, x^i)$ and the identity

$$\prod_i \left[a^i + h b^i\right] = \prod_i a^i + h \sum_i \left(\prod_{j \neq i} a^j\right) b^i + o(h).$$

Therefore, and up to an $o(h)$ order, the transition kernel factorizes to coordinate-wise independent transitions

$$\mathbb{P}(X_{t+h}^i = y^i \mid X_t = x) = \delta(y^i, x^i) + h u^i(y^i, x) + o(h). \tag{7.11}$$

These can be sampled with the Euler method (6.7) per coordinate. Interestingly, continuous Flow Matching also enjoys a similar factorization $u_t(x) = [u_t^1(x), \dots, u_t^d(x)] \in \mathbb{R}^d$, where $\dot{X}_t^i(x) = u_t^i(X_t)$ determines the change for coordinate i, and can be sampled independently (the "samples" in continuous FM are just deterministic).

7.5.2 Building probability paths with factorized velocities

If we construct probability paths in a certain way, it turns out to have factorized velocities (equation (7.9)) by construction. We explain this construction next. For this, we define a **factorized probability path** as a probability path of the form:

$$q_t(x) = \prod_i q_t^i(x^i). \tag{7.12}$$

Then, the following result shows that these factorized probability paths have factorized velocities.

Proposition 2. *Let $q_t(x)$ be a factorized probability path as in (7.12), where $u_t^i(y^i, x^i) \in C([0,1))$ generates $q_t^i(x^i)$. Then q_t has a factorized generating velocity of the form*

$$u_t(y, x) = \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x^i). \tag{7.13}$$

To proceed with the proof, let us denote the marginal distributions of a PMF $q(x)$ by

$$q^i(x^i) \coloneqq \sum_{x^{\bar{i}}} q(x) \quad q^{\bar{i}}(x^{\bar{i}}) \coloneqq \sum_{x^i} q(x) \tag{7.14}$$

Proof. Let q_t be a factorized probability path (7.12). Let $u_t^i(y^i, x^i)$ be the generating velocity of $q_t^i(x^i)$. Differentiating with respect to t yields

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} q_t(y) &= \sum_i q_t^{\bar{i}}(y^{\bar{i}}) \frac{\mathrm{d}}{\mathrm{d}t} q_t^i(y^i) \\
&\overset{(i)}{=} \sum_i \left[\sum_{x^{\bar{i}}} \delta(y^{\bar{i}}, x^{\bar{i}}) q_t^{\bar{i}}(x^{\bar{i}}) \right] \left[\sum_{x^i} u_t^i(y^i, x^i) q_t^i(x^i) \right] \\
&\overset{(ii)}{=} \sum_x \left[\sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x^i) \right] q_t(x),
\end{aligned}$$

Equality (i) follows from $q_t^{\bar{i}}(y^{\bar{i}}) = \sum_{x^{\bar{i}}} \delta(y^{\bar{i}}, x^{\bar{i}}) q_t^{\bar{i}}(x^{\bar{i}})$ and the Kolmogorov Equation (6.8). Equality (ii) follows from changing the summation order and noting that, by the definition of q_t, we have $q_t^{\bar{i}}(x^{\bar{i}}) q_t^i(x^i) = q_t(x)$ and $\sum_{x^{\bar{i}}} \sum_{x^i} = \sum_x$. □

We are now ready to show the main tool for constructing paths p_t with factorized velocities that interpolate between arbitrary p and q (Campbell et al., 2024; Gat et al., 2024).

Theorem 16 (Discrete Factorized Marginalization Trick)**.** *Consider a marginal probability path constructed via*

$$p_t(x) = \sum_z p_{t|Z}(x|z) p_Z(z), \text{ with } p_{t|Z}(x|z) = \prod_i p_{t|Z}^i(x^i|z), \tag{7.15}$$

i.e., where the conditional path factorizes in the sense of equation (7.12). Further, assume that $u_t^i(y^i, x^i|z)$ is $C([0,1))$ generates $p_{t|Z}^i(x^i|z)$ in $C^1([0,1))$, and $p_t(x) > 0$ for all $x \in \mathcal{S}$ and $t \in [0,1)$. Then, the marginal velocity is

$$u_t(y, x) = \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x) \tag{7.16}$$

with

$$u_t^i(y^i, x) = \sum_z u_t^i(y^i, x^i|z) p_{Z|t}(z|x) = \mathbb{E}\left[u_t^i(y^i, X_t^i|Z) | X_t = x \right] \tag{7.17}$$

generates $p_t(x)$.

Proof. According to proposition 2, the factorized conditional paths $p_{t|Z}(x|z)$ have factorized generating velocities $u_t(y, x|z) = \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x^i|z)$. Therefore,

$$\begin{aligned} u_t(y, x) &\overset{(i)}{=} \sum_z u_t(y, x|z) p_{Z|t}(z|x) \\ &\overset{(ii)}{=} \sum_z \left[\sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) u_t^i(y^i, x^i|z) \right] p_{Z|t}(x|z) \\ &\overset{(iii)}{=} \sum_i \delta(y^{\bar{i}}, x^{\bar{i}}) \left[\sum_z u_t^i(y^i, x^i|z) p_{Z|t}(z|x) \right]. \end{aligned}$$

Equality (i) follows from (7.2). Equality (ii) follows from assuming that $p_{t|Z}$ has factorized velocity. Equality (iii) follows from changing summation order. Because $p_{t|Z}^i(x^i|z) \in C^1([0,1))$ and $p_t(x) > 0$, it follows that $p_{t|Z}(x|z) \in C^1([0,1))$. Similarly, because $u_t^i(y^i, x^i|z) \in C([0,1))$, it follows that $u_t(y, x|z) \in C([0,1))$. Therefore, theorem 14 implies that $u_t(y, x)$ generates $p_t(x)$, as required. □

By using theorem 16, we can design a probability path p_t with factorized velocities interpolating between a source PMF p and a target PMF q as follows.

1. Find factorized probability conditional paths $p_{t|Z}(x|z) = \prod_i p_{t|Z}^i(x^i|z)$ such that the marginal $p_t(x)$ satisfies $p_0 = p$ and $p_1 = q$.
2. Find generating velocities $u_t^i(y^i, x^i|z)$ to $p_{t|Z}^i(x^i|z)$. This can be done by finding solution $u_t^i(y^i, x^i|z)$ to the Kolmogorov Equation:
$$\sum_{x^i} u_t^i(y^i, x^i|z) p_{t|Z}^i(x^i|z) = \frac{\mathrm{d}}{\mathrm{d}t} p_{t|Z}^i(y^i|z), \tag{7.18}$$
for all $y^i \in \mathcal{T}$, fixed values of $i \in [d]$, $z \in \mathcal{Z}$, and $t \in [0,1)$. As a remark, (7.18) is an under-determined linear system of equations with $|\mathcal{T}|$ unknowns (significantly less unknowns than the entire state space $|\mathcal{S}|$).

7.5.3 Conditional Discrete Flow Matching loss for factorized velocities

Representing the marginal velocity u_t^θ in terms of factorized velocities $u_t^{\theta,i}$ enables the following Conditional Flow Matching loss

$$\mathcal{L}_{\text{CDFM}}(\theta) = \mathbb{E}_{t, Z, X_t \sim p_{t|Z}} \sum_i D_{X_t}^i \left(u_t^i(\cdot, X_t|Z), u_t^{\theta,i}(\cdot, X_t) \right), \tag{7.19}$$

where $t \sim U[0,1]$, and $u_t^i(\cdot, x|z), u_t^{\theta,i}(\cdot, x) \in \mathbb{R}^{\mathcal{T}}$ satisfy the rate conditions. This means that $u_t^i(\cdot, x|z), u_t^{\theta,i}(\cdot, x) \in \Omega_{x^i}$ where, for $\alpha \in \mathcal{T}$, we define

$$\Omega_\alpha = \left\{ v \in \mathbb{R}^{\mathcal{T}} \;\middle|\; v(\beta) \geq 0 \; \forall \beta \in \mathcal{T} \setminus \{\alpha\}, \text{ and } v(\alpha) = -\sum_{\beta \neq \alpha} v(\beta) \right\} \subset \mathbb{R}^{\mathcal{T}}. \tag{7.20}$$

This is a convex set, and $D_x^i(u, v)$ is a Bregman divergence defined by a convex function $\Phi_x^i : \Omega_{x^i} \to \mathbb{R}$. As before, we justify this loss using proposition 1 and setting $X = X_t$, $Y = u_t^i(\cdot, X_t, Z) \in \mathbb{R}^{\mathcal{T}}$, letting $D_x^i(u, v)$ be a Bregman divergence over $\Omega_{x^i} \subset \mathbb{R}^{\mathcal{T}}$, and integrating with respect to $t \in [0,1]$.

7.5.4 Mixture paths

It is time to implement section 7.5.2 to build practical probability paths and their corresponding conditional velocities. Following Gat et al. (2024), we condition on $Z = (X_0, X_1)$ to accommodate arbitrary data couplings $(X_0, X_1) \sim \pi_{0,1}(X_0, X_1)$. Then, we build the factorized conditional paths

$$p_{t|0,1}(x|x_0, x_1) = \prod_i p_{t|0,1}^i(x^i|x_0, x_1) \tag{7.21}$$

as mixtures

$$p^i_{t|0,1}(x^i|x_0,x_1) = \kappa_t \delta(x^i,x^i_1) + (1-\kappa_t)\delta(x^i,x^i_0), \tag{7.22}$$

where $\kappa : [0,1] \to [0,1]$ is a $C^1([0,1])$ scheduler. Note that a RV $X^i_t \sim p^i_{t|0,1}(\cdot|x_0,x_1)$ follows

$$X^i_t = \begin{cases} x^i_1 & \text{with prob } \kappa_t \\ x^i_0 & \text{with prob } (1-\kappa_t) \end{cases}, \tag{7.23}$$

i.e. it assumes either the source or the target states with a probability depending on the time t.

If $\kappa_0 = 0$ and $\kappa_1 = 1$, then the marginal $p_t(x)$ in (7.1) satisfies the boundary constraints. We also need generating velocities $u^i_t(y^i,x^i|x_0,x_1)$ for $p^i_{t|0,1}(x^i|x_0,x_1)$, which are solutions to (7.18). We derive these as follows:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t} p^i_{t|Z}(y^i|z) &\overset{(7.22)}{=} \dot\kappa_t \left[\delta(y^i,x^i_1) - \delta(y^i,x^i_0)\right] \\
&\overset{(7.22)}{=} \dot\kappa_t \left[\delta(y^i,x^i_1) - \frac{p^i_{t|Z}(y^i|z) - \kappa_t \delta(y^i,x^i_1)}{1-\kappa_t}\right] \\
&= \frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i,x^i_1) - p^i_{t|Z}(y^i|z)\right] \\
&= \sum_{x^i} \frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i,x^i_1) - \delta(y^i,x^i)\right] p^i_{t|Z}(x^i|z),
\end{aligned}$$

where we have used $z = (x_0,x_1)$ and $Z = (X_0,X_1)$ interchangeably to keep notation concise. In conclusion, we have found a conditional velocity generating the path in (7.22), namely

$$u^i_t(y^i,x^i|x_0,x_1) = \frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i,x^i_1) - \delta(y^i,x^i)\right]. \tag{7.24}$$

Code 9 shows how mixture paths are defined in the library `flow_matching` library.

Code 9: Discrete probability path

```python
import torch
from flow_matching.path import MixtureDiscreteProbPath
from flow_matching.path.path_sample import DiscretePathSample

# Create a discrete probability path object
path = MixtureDiscreteProbPath(scheduler=PolynomialConvexScheduler(n=1.0))

# Sample the conditional path
# batch_size = 2 for t, X_0 and X_1
t = torch.tensor([0.25, 0.5])
x_0 = torch.tensor([0, 0])
x_1 = torch.tensor([1, 2])
sample: DiscretePathSample = path.sample(t=t, x_0=x_0, x_1=x_1)
sample.x_0  # X_0 is [0, 0]
sample.x_1  # X_1 is [1, 2]
# X_t is
#   [0 with probability 0.75 and 1 with probability 0.25,
#    0 with probability 0.5 and 2 with probability 0.5]
sample.x_t
sample.t  # t is [0.25, 0.5]
```

Velocity posterior parameterization. Similar to the continuous case (*e.g.*, section 4.8.1), we can choose to parameterize our velocity $u_t^i(y^i, x)$ in different ways. The first approach is to parameterize it directly, akin to velocities in flows. Another way, which we take here, is motivated by the following computation of the **mixture marginal velocity** following (7.17):

$$\begin{aligned} u_t^i(y^i, x) &= \sum_{x_0, x_1} \frac{\dot{\kappa}_t}{1-\kappa_t} \left[\delta(y^i, x_1^i) - \delta(y^i, x^i)\right] p_{0,1|t}(x_0, x_1|x) \\ &= \sum_{x_1^i} \frac{\dot{\kappa}_t}{1-\kappa_t} \left[\delta(y^i, x_1^i) - \delta(y^i, x^i)\right] p_{1|t}^i(x_1^i|x), \end{aligned} \tag{7.25}$$

where for the second equality we denote the marginal of the posterior $p_{0,1|t}$

$$p_{1|t}^i(x_1^i|x) = \sum_{x_0, x_1^{\bar{i}}} p_{0,1|t}(x_0, x_1|x) \tag{7.26}$$

$$= \mathbb{E}\left[\delta(x_1^i, X_1^i) \mid X_t = x\right]. \tag{7.27}$$

This derivation represents the marginal $u_t^i(y^i, x)$ using a learnable posterior $p_{1|t}^{\theta,i}(x_1^i|x)$, which can be understood as a discrete version of x_1-prediction (section 4.8.1). Next, we explore loss functions to learn this posterior.

CDFM losses for mixture paths We present two options for learning $p_{1|t}^{\theta,i}(x_1^i|x)$, both justified by proposition 1. First, the marginal posterior (7.26) and (7.27) can be learned by the conditional matching loss

$$\mathcal{L}_{\text{CM}}(\theta) = \mathbb{E}_{t, X_0, X_1, X_t} D_{X_t}\left(\delta(\cdot, X_1^i), p_{1|t}^{\theta,i}(\cdot|X_t)\right) \tag{7.28}$$

Since $\delta(\cdot, X_1^i), p_{1|t}^{\theta,i}(\cdot|X_t)$ are PMFs. Therefore, we can set the Bregman divergence to be the KL-divergence $D(p, q) = \sum_{\alpha \in \mathcal{T}} p(\alpha) \log \frac{p(\alpha)}{q(\alpha)}$ comparing PMFs, obtaining

$$\mathcal{L}_{\text{CM}}(\theta) = -\mathbb{E}_{t, X_0, X_1, X_t} \log p_{1|t}^{\theta,i}(X_1^i|X_t) + \text{const.} \tag{7.29}$$

Alternatively, we may follow section 7.5.3 and use the factorized loss in (7.19) with $u_t^{\theta,i}$ parametrized by $p_{1|t}^{\theta,i}$. In this case, we can set the Bregman divergence to be the *generalized* KL comparing general (not necessarily probability) vectors $u, v \in \mathbb{R}_{\geq 0}^m$:

$$D(u, v) = \sum_j u^j \log \frac{u^j}{v^j} - \sum_j u_j + \sum_j v_j. \tag{7.30}$$

For this choice of D, we get

$$D\left(u_t^i(\cdot, x^i|x_0, x_1), u_t^{\theta,i}(\cdot, x)\right) = \frac{\dot{\kappa}_t}{1-\kappa_t}\left[(\delta(x_1^i, x^i) - 1) \log p_{1|t}^{\theta,i}(x_1^i|x) + \delta(x_1^i, x^i) - p_{1|t}^{\theta,i}(x^i|x)\right] \tag{7.31}$$

which implements the loss (7.19) when conditioning on $Z = (X_0, X_1)$. The generalized KL loss (7.31) also provides an evidence lower bound (ELBO) on the likelihood of the target distribution (Shaul et al., 2024),

$$-\log p_1^\theta(x_1) \leq \mathbb{E}_{t, X_0, X_t \sim p_{t|0,1}} \sum_i D\left(u_t^i(\cdot, X_t^i|X_0, x_1), u_t^{\theta,i}(\cdot, X_t)\right), \tag{7.32}$$

where p_1^θ is the marginal generated by the model at time $t = 1$. Hence, in addition to training, the generalized KL loss is commonly used for evaluation.

Sampling mixture paths The parametrization based on the posterior $p^{\theta,i}_{1|t}$ leads to the following sampling algorithm. As indicated in section 7.5.1 working with factorized velocities enables the coordinate-wise sampling (7.11). According to (7.17) and (7.25),

$$\mathbb{P}(X^i_{t+h} = y^i \mid X_t = x) = \delta(y^i, x^i) + hu^i(y^i, x) + o(h) \tag{7.33}$$

$$= \sum_{x^i_1} \left[\delta(y^i, x^i) + h\frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i, x^i_1) - \delta(y^i, x^i)\right] + o(h)\right] p^i_{1|t}(x^i_1|x). \tag{7.34}$$

Consequently, and given $X_t = x$, we may perform one step of sampling by performing the next two steps for each $i \in [d]$: (i) draw $X^i_1 \sim p^i_{1|t}(X^i_1|x)$; and (ii) update X^i_{t+h} according to the Euler step in (6.7) with the velocity $\frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i, X^i_1) - \delta(y^i, x^i)\right]$. Intuitively, (ii) decides whether to set $X^i_{t+h} = X^i_1$ or remain at $X^i_{t+h} = X^i_t$.

One-sided mixture paths and probability preserving velocities It is often useful to extend the design space of the sampling algorithm by adding some divergence-free component, as described in section 6.3.1. For factorized paths, the divergence-free velocity v^i_t needs to satisfy (7.18), namely,

$$\sum_{x^i} v^i_t(y^i, x^i|z)p^i_{t|Z}(x^i|z) = 0. \tag{7.35}$$

In general it could challenging to find such probability-preserving velocities without learning additional quantities, *e.g.*, $p^i_{0|t}$. However, one useful case where a probability preserving velocity can be found in closed form is when assuming iid source distribution *i.e.*, $p(x) = \prod_i p(x^i)$ and independent coupling $\pi_{0,1}(x_0, x_1) = p(x_0)q(x_1)$. In this case the marginal mixture path takes the form

$$p_t(x) = \sum_{x_1} p_{t|1}(x|x_1)q(x_1), \text{ where } p_{t|1}(x|x_1) = \prod_i p^i_{t|1}(x^i|x_1),$$

where $p^i_{t|1}(x^i|x_1) = \left[\kappa_t\delta(x^i, x^i_1) + (1-\kappa_t)p(x^i)\right]$. The conditional velocity in (7.24), *i.e.*,

$$u^i_t(y^i, x^i|x_1) = \frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i, x^i_1) - \delta(y^i, x^i)\right] \tag{7.36}$$

also generates $p^i_{t|1}(x^i|x_1)$. To find a divergence-free velocity we can, for example, subtract from this velocity a *backward-time velocity* $\tilde u^i_t$ for $p^i_{t|1}(y^i, x^i|x_1)$ (Gat et al., 2024), in the sense that it satisfies the Kolmogorov equation with $p^i_{t|1}(y^i, x^i|x_1)$ and $-\tilde u^i_t$ satisfies the rate conditions. Such a velocity can be found in a fashion to equation (7.24),

$$\tilde u^i_t(y^i, x^i|x_1) = \frac{\dot\kappa_t}{\kappa_t}\left[\delta(y^i, x^i) - p(x^i)\right]. \tag{7.37}$$

Therefore, a divergence-free velocity for $p^i_{t|1}(x^i|x_1)$ conditional path can be defined via

$$v^i_t(y^i, x^i|x_1) = u^i_t(y^i, x^i|x_1) - \tilde u^i_t(y^i, x^i|x_1). \tag{7.38}$$

According to section 6.3.1, if we add a divergence-free field $v^i_t(y^i, x^i|x_1)$ to the velocity $u^i_t(y^i, x^i|x_1)$, the latter still generates the same probability path $p^i_{t|1}(x^i|x_1)$. Consequently, theorem 16 implies that the marginal velocity $u^i_t(y^i, x)$ defined by

$$u^i_t(y^i, x) = \sum_{x_1}\left[u^i_t(y^i, x^i|x_1) + c_t v^i_t(y^i, x^i|x_1)\right] p_{1|t}(x_1|x)$$

$$= \sum_{x^i_1}\left[u^i_t(y^i, x^i|x^i_1) + c_t v^i_t(y^i, x^i|x^i_1)\right] p^i_{1|t}(x^i|x),$$

still generates the same marginal path $p_t(x)$, where the second equality follows from $u^i_t(y^i, x^i|x_1) = u^i_t(y^i, x^i|x^i_1)$ for mixture paths, and similarly for $v^i_t(y^i, x^i|x^i_1)$. In conclusion, and given $X_t = x$, a single step of the

generalized sampling algorithm consists in (i) drawing $X_1^i \sim p^i_{1|t}(X_1^i|x)$ and (ii) taking an Euler step (6.7) with the velocity

$$u_t^i(y^i, x^i|x_1) = \frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i, X_1^i) - \delta(y^i, x^i)\right] + c_t\left[\frac{\dot\kappa_t}{1-\kappa_t}\left[\delta(y^i, x_1^i) - \delta(y^i, x^i)\right] - \frac{\dot\kappa_t}{\kappa_t}\left[\delta(y^i, x^i) - p(x^i)\right]\right],$$

where $c_t > 0$ is a time dependent constant.

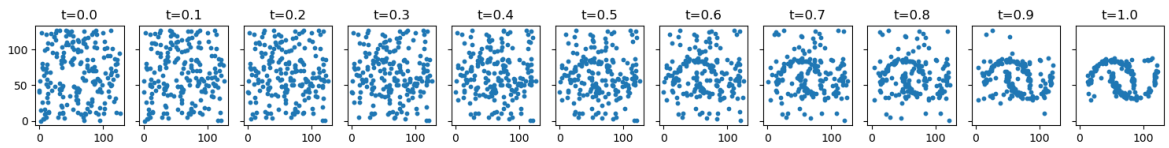
Similar to the continuous flow matching example in code 1, we provide a standalone implementation of discrete flow matching in pure PyTorch in code 11. Code 10 illustrates how to train a discrete flow with arbitrary data coupling using the `flow_matching` library.

Code 10: Training and sampling DFM with mixture paths and arbitrary data coupling.

```python
import torch

from flow_matching.path import MixtureDiscreteProbPath, DiscretePathSample
from flow_matching.path.scheduler import PolynomialConvexScheduler
from flow_matching.loss import MixturePathGeneralizedKL
from flow_matching.solver import MixtureDiscreteEulerSolver
from flow_matching.utils import ModelWrapper


model = ...  # Define a trainable velocity model
optimizer = torch.optim.Adam(model.parameters())

scheduler = PolynomialConvexScheduler(n=1.0)
path = MixtureDiscreteProbPath(scheduler=scheduler)
loss_fn = MixturePathGeneralizedKL(path=path)  # Generalized KL Bregman divergence

for x_0, x_1 in dataloader:  # Samples from π_{0,1} of shape [batch_size, *data_dim]
    t = torch.rand(batch_size) * (1.0 - 1e-3)  # Randomize time t ~ U[0, 1 − 10^{−3}]
    sample: DiscretePathSample = path.sample(t=t, x_0=x_0, x_1=x_1)  # Sample the conditional path
    model_output = model(sample.x_t, sample.t)

    loss = loss_fn(logits=model_output, x_1=sample.x_1, x_t=sample.x_t, t=sample.t) # CDFM loss

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

class ProbabilityDenoiser(ModelWrapper):
    def forward(self, x: torch.Tensor, t: torch.Tensor, **extras) -> torch.Tensor:
        logits = self.model(x, t, **extras)
        return torch.nn.functional.softmax(logits.float(), dim=-1)

# Sample X_1
probability_denoiser = ProbabilityDenoiser(model=model)
x_0 = torch.randint(size=[batch_size, *data_dim])  # Specify the initial condition
solver = MixtureDiscreteEulerSolver(
    model=probability_denoiser,
    path=path,
    vocabulary_size=vocabulary_size
)

step_size = 1 / 100
x_1 = solver.sample(x_init=x_0, step_size=step_size, time_grid=torch.tensor([0.0, 1.0-1e-3]))
```

Code 11: Standalone Discrete Flow Matching code
flow_matching/examples/standalone_discrete_flow_matching.ipynb

```python
import torch
import matplotlib.pyplot as plt
from torch import nn, Tensor
from sklearn.datasets import make_moons

class DiscreteFlow(nn.Module):
    def __init__(self, dim: int = 2, h: int = 128, v: int = 128):
        super().__init__()
        self.v = v
        self.embed = nn.Embedding(v, h)
        self.net = nn.Sequential(
            nn.Linear(dim * h + 1, h), nn.ELU(),
            nn.Linear(h, h), nn.ELU(),
            nn.Linear(h, h), nn.ELU(),
            nn.Linear(h, dim * v))

    def forward(self, x_t: Tensor, t: Tensor) -> Tensor:
        return self.net(torch.cat((t[:, None], self.embed(x_t).flatten(1, 2)),
        ↪ -1)).reshape(list(x_t.shape) + [self.v])

batch_size = 256
vocab_size = 128

model = DiscreteFlow(v=vocab_size)
optim = torch.optim.Adam(model.parameters(), lr=0.001)

for _ in range(10000):
    x_1 = Tensor(make_moons(batch_size, noise=0.05)[0])
    x_1 = torch.round(torch.clip(x_1 * 35 + 50, min=0.0, max=vocab_size - 1)).long()
    x_0 = torch.randint(low=0, high=vocab_size, size=(batch_size, 2))

    t = torch.rand(batch_size)
    x_t = torch.where(torch.rand(batch_size, 2) <  t[:, None], x_1, x_0)

    logits = model(x_t, t)
    loss = nn.functional.cross_entropy(logits.flatten(0, 1), x_1.flatten(0, 1)).mean()
    optim.zero_grad()
    loss.backward()
    optim.step()

x_t = torch.randint(low=0, high=vocab_size, size=(200, 2))
t = 0.0
results = [(x_t, t)]
while t < 1.0 - 1e-3:
    p1 = torch.softmax(model(x_t, torch.ones(200) * t), dim=-1)
    h = min(0.1, 1.0 - t)
    one_hot_x_t = nn.functional.one_hot(x_t, vocab_size).float()
    u = (p1 - one_hot_x_t) / (1.0 - t)
    x_t = torch.distributions.Categorical(probs=one_hot_x_t + h * u).sample()
    t += h
    results.append((x_t, t))
fig, axes = plt.subplots(1, len(results), figsize=(15, 2), sharex=True, sharey=True)
for (x_t, t), ax in zip(results, axes):
    ax.scatter(x_t.detach()[:, 0], x_t.detach()[:, 1], s=10)
    ax.set_title(f't={t:.1f}')
plt.tight_layout()
plt.show()
```



8 Continuous Time Markov Process Models

In the previous sections, we have developed a flow model on $\mathbb{R}^d$ and Riemannian manifolds (see section 3 and section 5) and a CTMC model for discrete data (see section 6). In this section, we want to unify and extend these models to a generative model that works for (1) general state spaces and (2) general Markov processes. This generative model will allow us to extend the principles of Flow Matching to a wide variety of generative models for a variety of modalities in section 9.

8.1 General state spaces and random variables

Working with general modalities. Our explicit goal is not specify the modality we use. Hence, throughout this section, let $\mathcal{S}$ be a general **state space**. Important examples are $\mathcal{S} = \mathbb{R}^d$ (*e.g.*, images, vectors), $\mathcal{S}$ discrete (*e.g.*, language), $\mathcal{S}$ a Riemannian manifold (*e.g.*, geometric data) or their products for generation of multiple data modalities jointly (multimodal models). For all modalities, we can define a metric (or distance function) $d : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, (x, y) \mapsto d(x, y)$ on $\mathcal{S}$. For example, for $\mathcal{S}$ discrete, the metric is simply $d(x, y) = 1$ if $y \neq x$ and $d(x, x) = 0$ for all $x \in \mathcal{S}$. For $\mathcal{S} = \mathbb{R}^d$, we use $d(x, y) = \|x - y\|$. We need to make a technical assumption that $(\mathcal{S}, d)$ is a Polish metric space, *i.e.*, it is complete (*i.e.*, any Cauchy sequence converges) and separable (*i.e.*, it has a countable dense subset). Any modality of interest for machine learning has that property.

Densities over general state spaces. So far, in this work we assumed that a probability distribution p over $\mathcal{S}$ is represented by a density $p : \mathcal{S} \to \mathbb{R}_{\geq 0}$. For general state spaces, we use a general **reference measure** ν and the density becomes the Radon-Nikodym derivative $\frac{\mathrm{d}p}{\mathrm{d}\nu}$. In other words, probabilities can be expressed as integrals with respect to ν

$$\mathbb{P}(A) = \int_A p(x)\nu(\mathrm{d}x) \text{ for all measurable } A \subset \mathcal{S}$$

For $\mathcal{S}$ discrete, ν was the counting measure (so the integrals are just sums) and $p(x)$ is just the probability mass function (PMF). For $\mathcal{S} = \mathbb{R}^d$, ν was the Lebesgue measure (so the integrals are just the "usual" integral) and $p(x)$ is just the probability density function (PDF). The above generalizes that to arbitrary state spaces.

(Optional) Working with arbitrary distributions. It is important to note that not every probability distribution admits a density with respect to a reference measure. For the reader unfamiliar with general measure theory, it is safe ignore this possibility as a technical remark as long one works with distributions of interest that have a density $p(x)$. However, note that these are not just pathological examples but there are real cases of interest for machine learning applications where this matters: A simple example is a probability path of the form $p_t = \delta_{(1-t)x+ty}$ on $\mathcal{S} = \mathbb{R}^d$ connecting two points $x, y \in \mathbb{R}^d$ in a straight line - this cannot be represented by a density. Another example would be probability distributions over $\mathcal{S} = C([0, 1], \mathbb{R})$, *e.g.*, for trajectory modeling, that often do not have a density with respect to a common reference measure. To mathematically handle such cases, we develop our framework for general **probability measures** p over $\mathcal{S}$. For this, we use the notation $p(\mathrm{d}x)$ where "$\mathrm{d}x$" is a *symbolic* expression denoting integration with respect to p in a variable x. For example, for a bounded measurable function $f : \mathcal{S} \to \mathbb{R}$ we write

$$\mathbb{E}_{X\sim p}[f(X)] = \int f(x)p(\mathrm{d}x)$$

for the Lebesgue integral or the expected value of f under p. As before, we use (with a slight abuse of notation) the same notation $p(x)$ to denote the density of the measure $p(\mathrm{d}x)$.

8.2 The CTMP generative model

Similarly, our explicit goal is build a model that works for arbitrary evolution process - regardless of whether we use a flow, diffusion, a CTMC, a combination, or something else. Therefore, we define a evolution process in this section that is general but satisfies the necessary regularity assumptions to build a generative model. For $t \in [0, 1]$, let $X_t \in \mathcal{S}$ be a random variable. We call $(X_t)_{0\leq t\leq 1}$ a **Continuous-time Markov process (CTMP)** if it fulfills the following condition:

Name	Flow	Diffusion	Jump process	Continous-time Markov chain
Space $\mathcal{S}$	$\mathcal{S} = \mathbb{R}^d$	$\mathcal{S} = \mathbb{R}^d$	$\mathcal{S}$ arbitrary	$\lvert\mathcal{S}\rvert < \infty$
Parameters	Velocity field: $u_t(x) \in \mathbb{R}^d$	Diffusion coefficient: $\sigma_t(x) \in \mathbb{R}^{d\times d}$ (p.s.d.)	Jump measure $Q_t(\mathrm{d}y, x)$	$u_t \in \mathbb{R}^{\mathcal{S}\times\mathcal{S}}$, $1^T u_t = 0$ $u_t(x'; x) \geq 0$ $(x' \neq x)$
Sampling	$X_{t+h} = X_t + h u_t(X_t)$	$X_{t+h} = X_t + \sqrt{h}\sigma_t(X_t)\epsilon_t$ $\epsilon_t \sim \mathcal{N}(0, I)$	$X_{t+h} = X_t$ with prob. $1 - h\int Q_t(\mathrm{d}y, x)$ $X_{t+h} \sim \frac{Q_t(\mathrm{d}y,x)}{\int Q_t(\mathrm{d}y,x)}$ with prob. $h\int Q_t(\mathrm{d}y, x)$	$X_{t+h} \sim (I + h u_t)(\cdot; X_t)$
Generator $\mathcal{L}_t$	$\nabla f^T u_t$	$\frac{1}{2}\nabla^2 f \cdot \sigma_t^2$	$\int [f(y) - f(x)] Q_t(\mathrm{d}y, x)$	$f^T u_t^T$
KFE (Adjoint)	Continuity Equation: $\partial_t p_t = -\mathrm{div}(p_t u_t)$	Fokker-Planck Equation: $\partial_t p_t = \frac{1}{2}\nabla^2 \cdot (p_t \sigma_t^2)$	Jump Continuity Equation: $\partial_t p_t(x) =$ $\int Q_t(x, y) p_t(y) - Q_t(y, x) p_t(x) \nu(\mathrm{d}y)$	Mass preservation: $\partial_t p_t = u_t p_t$
Marginal	$\mathbb{E}_{Z\sim p_{Z\lvert t}(\cdot\lvert x)}[u_t(x\lvert Z)]$	$\mathbb{E}_{Z\sim p_{Z\lvert t}(\cdot\lvert x)}[\sigma_t^2(x\lvert Z)]$	$\mathbb{E}_{Z\sim p_{Z\lvert t}(\cdot\lvert x)}[Q_t(y, x\lvert Z)]$	$\mathbb{E}_{Z\sim p_{Z\lvert t}(\cdot\lvert x)}[u_t(y; x\lvert Z)]$
CGM Loss (Example)	$\lVert u_t(X\lvert Z) - u_t^\theta(X)\rVert^2$	$\lVert\sigma_t^2(X\lvert Z) - (\sigma_t^\theta)^2(X)\rVert_2^2$	$(\int Q_t^\theta(y; X)\nu(\mathrm{d}y)$ $-Q_t(y, X\lvert Z)\log Q_t^\theta(y; X)\nu(\mathrm{d}y))$	$(\sum_{y\neq x} u_t^\theta(y; X)$ $-u_t(y; X\lvert Z)\log u_t^\theta(y; X))$

Table 2 Some examples of CTMP generative models and how they can be learnt with Generator Matching. This list is not exhaustive. Derivations are in section 8. For diffusion, we assume zero drift as this is covered by the "Flow" column. KFE is listed in its adjoint version, *i.e.*, assumes jump kernel $Q_t(y, x)$ and density $p_t(x)$ exists with respect to reference measure ν. "p.s.d.": Positive semi-definite.

$$\mathbb{P}[X_{t_{n+1}} \in A | X_{t_1}, X_{t_2}, \ldots, X_{t_n}] = \mathbb{P}[X_{t_{n+1}} \in A | X_{t_n}] \quad (0 \leq t_1 < \cdots < t_{n+1} \leq 1, A \subseteq \mathcal{S}) \tag{8.1}$$

Informally, the above condition says that the process has no memory. If we know the present, knowing the past will not influence our prediction of the future. In table 2, we give an overview over important classes of Markov processes. For example, a flow on a manifold is a Markov process with deterministic transitions, a diffusion is a Markov process with transitions driven by Brownian motion, and CTMCs are Markov processes determined by rates (we will explain this in detail in section 8.2.2). Each Markov process has a **transition kernel** $(p_{t+h|t})_{0\leq t<t+h\leq 1}$ that assigns every $x \in \mathcal{S}$ a probability distribution $p_{t+h|t}(\cdot|x)$ such that

$$\mathbb{P}[X_{t+h} \in A | X_t = x] = p_{t+h|t}(A|x) \quad \text{for all } t, h \geq 0, A \subset \mathcal{S} \text{ measurable} \tag{8.2}$$

Due to the Markov assumption, a Markov process is uniquely determined by the transition kernel and the distribution of X_0. Conversely, any transition kernel and initial distribution defines a Markov process. Therefore, there is a 1:1 correspondence.

Our next goal is to define a corresponding generalization of a velocity field for CTMPs. Informally, it would be the 1st-order derivative of the transition kernel in t:

$$\mathcal{L}_t \coloneqq \frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} p_{t+h|t} \tag{8.3}$$

We call the 1st-order derivative $\mathcal{L}_t$ the **generator** of $p_{t+h|t}$ (Ethier and Kurtz, 2009; Rüschendorf et al., 2016). Similar to derivatives, generators are first-order *linear* approximations and easier to parameterize than $p_{t+h|t}$. As we will see, diffusion, flows, and other generative models can all be seen as algorithms to learn the generator of a Markov process (see table 2). This leads to the general form of the **CTMP generative model** given by

$$\textbf{CTMP model (informal):} \quad p_{t+h|t}(\cdot|x) \coloneqq \delta_x + h\mathcal{L}_t(x) + o(h), \text{ and } X_0 \sim p. \tag{8.4}$$

However, as a transition kernel $p_{t+h|t}$ is not a real function, equation 8.3 and 8.4 are only heuristic and not well-defined yet. Therefore the first goal of this section is to provide a *formal* definition of the generator and the CTMP generative model.

8.2.1 Formal definition of generator

The first problem in equation (8.3) is that derivatives are usually defined with respect to functions mapping to vector spaces but $p_{t+h|t}$ maps to a distribution. However, this can be alleviated by using **test functions**. Test functions are a way to "probe" a probability distribution. They serve as a theoretical tool to handle distributions as if they were real-valued functions. Specifically, a set of test functions is a family $\mathcal{T}$ of bounded, measurable functions $f : S \to \mathbb{R}$ that *characterize* probability distributions fully, *i.e.*, for two probability distributions μ_1, μ_2 on $\mathcal{S}$ it holds

$$\mu_1 = \mu_2 \quad \Leftrightarrow \quad \mathbb{E}_{X\sim\mu_1}[f(X)] = \mathbb{E}_{X\sim\mu_2}[f(X)] \quad \text{for all } f \in \mathcal{T} \tag{8.5}$$

Generally speaking, one chooses $\mathcal{T}$ to be as "nice" (or regular) as possible. For example, if $\mathcal{S} = \mathbb{R}^d$, the space $\mathcal{T} = C_c^\infty(\mathbb{R}^d)$ of infinitely differentiable functions with compact support fulfills that property. For $\mathcal{S}$ discrete, $\mathcal{T} = \mathbb{R}^{\mathcal{S}}$ simply consists of all functions (which are just vectors in this case). Let $X_t \sim p_t$. We define the **marginal action** and **transition action** as

$$\langle p_t, f\rangle \coloneqq \int f(x)p_t(\mathrm{d}x) = \mathbb{E}_{X\sim p_t}\left[f(X)\right] \tag{8.6}$$

$$\left\langle p_{t+h|t}, f\right\rangle(x) \coloneqq \left\langle p_{t+h|t}(\cdot|x), f\right\rangle = \mathbb{E}\left[f(X_{t+h})|X_t = x\right] \tag{8.7}$$

where the marginal action maps each test function f to a scalar $\langle p_t, f\rangle \in \mathbb{R}$, while the transition action maps a real-valued function $x \mapsto f(x)$ to a another real-valued function $x \mapsto \left\langle p_{t+h|t}, f\right\rangle(x)$. The tower property implies that $\left\langle p_t, \left\langle p_{t+h|t}, f\right\rangle\right\rangle = \langle p_{t+h}, f\rangle$. We note that the above is only a "symbolic" dot product but becomes a "proper" dot product if a density $p_t(x)$ exists, *i.e.*, $\langle p_t, f\rangle = \int f(x)p_t(x)\nu(\mathrm{d}x)$.

The second step to formally define a derivative as in equation (8.3) is that we need to impose some notion of "smoothness" on a Markov process that we define now. Let $C_0(\mathcal{S})$ be the space of continuous functions $f : \mathcal{S} \to \mathbb{R}$ that vanish at infinity, *i.e.*, for all $\epsilon > 0$ there exists a compact set $K \subset \mathcal{S}$ such that $|f(x)| < \epsilon$ for all $x \in \mathcal{S} \setminus K$. We use the supremum norm $\|\cdot\|_\infty$ on $C_0(\mathcal{S})$. A CTMP X_t is called a **Feller process** if it fulfils the following two conditions (Feller, 1955; Rüschendorf et al., 2016):

1. **Strong continuity:** The action of $p_{t+h|t}$ is continuous in time:

$$\lim_{h'\to h, t'\to t} \| \left\langle p_{t'+h'|t'}, f\right\rangle - \left\langle p_{t+h|t}, f\right\rangle \|_\infty = 0 \quad \text{for all } h, t \geq 0, f \in C_0(\mathcal{S})$$

2. **No return from infinity:** The action of $p_{t+h|t}$ preserves functions that vanish at infinity:

$$\left\langle p_{t+h|t}, f\right\rangle \in C_0(\mathcal{S}) \quad \text{for all } h, t \geq 0, f \in C_0(\mathcal{S})$$

Assumption 4. The CTMP $(X_t)_{0\leq t\leq 1}$ is a Feller process.

This is a reasonable assumption given that we want to use X_t in a machine learning model: We define probability paths where the distribution of the generative process X_t vary smoothly, and all our data usually lies in some bounded (compact) set.

Let us now revisit (8.3) and try define the derivative of $p_{t+h|t}$. With the test function perspective in mind, we can take derivatives of $\left\langle p_{t+h|t}, f\right\rangle(x)$ per $x \in \mathcal{S}$ and define

$$\frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} \left\langle p_{t+h|t}, f\right\rangle(x) = \lim_{h\to 0} \frac{\left\langle p_{t+h|t}, f\right\rangle(x) - f(x)}{h} \coloneqq [\mathcal{L}_t f](x). \tag{8.8}$$

We call this action the **generator** $\mathcal{L}_t$ and define it for all f for which the above limit exists uniformly, *i.e.*, in the norm $\|\cdot\|_\infty$. Intuitively, a generator is defined as an *operator of test functions*. In table 2, there are several examples of generators that we derive in section 8.2.2. There is a 1:1 correspondence between a generator and a Feller process (Rogers and Williams, 2000; Ethier and Kurtz, 2009; Pazy, 2012) - in the same way as there is a correspondence between a flow and a vector field (see theorem 1). This will later allows us to parameterize a Feller process via a generator in a neural network.

With this definition, the **CTMP model** in (8.4) has the, now well-defined, form as

$$\langle p_{t+h|t}, f \rangle = f + h\mathcal{L}_t f + o(h) \text{ (for all } f \in \mathcal{T}) \quad \text{and } X_0 \sim p \tag{8.9}$$

where $o(h)$ describes an error terms $E(h) \in C_0(\mathcal{S})$ such that $\lim_{h\to 0} \frac{1}{h}\|E(h)\|_\infty = 0$. Similarly to equation (3.24) for the case of flows and to equation (6.5) for the case of CTMCs, we say that

$$\mathcal{L}_t \textbf{ generates } p_t \text{ if there exists a } p_{t+h|t} \text{ satisfying (8.9) with CTMP } X_t \text{ such that } X_t \sim p_t \tag{8.10}$$

In other words, a generator $\mathcal{L}_t$ generates the probability path p_t if a Markov process that is (1) initialized with $p = p_0$ and (2) simulated with $\mathcal{L}_t$ has marginals p_t.

8.2.2 Examples of CTMP models

We go through several examples to illustrate how to compute a generator of a Markov process. The results from this section are summarized in table 2.

Flows. Let $\mathcal{S} = \mathbb{R}^d$ and $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d, (t,x) \mapsto u_t(x)$ be a time-dependent velocity field defining a flow ψ_t (see section 3). Let $\mathcal{T} = C_c^\infty(\mathbb{R}^d)$ be the space of infinitely differentiable and smooth functions with compact support. Then we can compute the generator via

$$[\mathcal{L}_t f](x) = \lim_{h\to 0} \frac{\mathbb{E}\left[f(X_{t+h})|X_t = x\right] - f(x)}{h} \tag{8.11}$$

$$\overset{(i)}{=} \lim_{h\to 0} \frac{\mathbb{E}\left[f(X_t + hu_t(X_t) + o(h))|X_t = x\right] - f(x)}{h} \tag{8.12}$$

$$\overset{(ii)}{=} \lim_{h\to 0} \frac{\mathbb{E}\left[f(X_t) + h\nabla f(X_t)^T u_t(X_t) + o(h)|X_t = x\right] - f(x)}{h} \tag{8.13}$$

$$= \lim_{h\to 0} \frac{f(x) + h\nabla f(x)^T u_t(x) + o(h) - f(x)}{h} \tag{8.14}$$

$$\tag{8.15}$$

$$= \nabla f(x)^T u_t(x), \tag{8.16}$$

where (i) follows from a Euler approximation of the flow and (ii) follows from a first-order Taylor approximation of f around X_t. Therefore, the **flow generator** is given by

$$\mathcal{L}_t f(x) = \nabla f(x)^T u_t(x). \tag{8.17}$$

Diffusion. Let $\mathcal{S} = \mathbb{R}^d$ and $\sigma_t : [0,1] \times \mathbb{R}^d \to \mathbb{R}^{d\times d}, (t,x) \mapsto \sigma_t(x)$ be a time-dependent function mapping to symmetric positive semi-definite matrices σ_t in a continuous fashion. A diffusion process with diffusion coefficient σ_t is defined via the SDE $dX_t = \sigma_t(X_t)dW_t$ for a Wiener process W_t (Øksendal, 2003). This process can be approximated via the infinitesimal sampling procedure:

$$X_{t+h} = X_t + \sqrt{h}\sigma_t(X_t)\epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, I) \tag{8.18}$$

Let again be $\mathcal{T} = C_c^\infty(\mathbb{R}^d)$. Then we can compute the generator via

$$[\mathcal{L}_t f](x) = \lim_{h\to 0} \frac{\mathbb{E}\left[f(X_t + \sqrt{h}\sigma_t(X_t)\epsilon_t + o(h))|X_t = x\right] - f(x)}{h} \tag{8.19}$$

$$\overset{(i)}{=} \lim_{h\to 0} \frac{\mathbb{E}_{\epsilon_t}[f(x) + \nabla f(x)^T\sqrt{h}\sigma_t(x)\epsilon_t + \frac{1}{2}h[\sigma_t(x)\epsilon_t]^T\nabla^2 f(x)[\sigma_t(x)\epsilon_t] + o(h) - f(x)]}{h} \tag{8.20}$$

$$= \lim_{h\to 0} \frac{\nabla f(x)^T\sqrt{h}\sigma_t(x)\mathbb{E}_{\epsilon_t}[\epsilon_t] + \mathbb{E}_{\epsilon_t}[\frac{1}{2}h[\sigma_t(x)\epsilon_t]^T\nabla^2 f(x)[\sigma_t(x)\epsilon_t]]}{h} \tag{8.21}$$

$$= \frac{1}{2}\mathbb{E}_{\epsilon_t}[\epsilon_t^T[\sigma_t(x)]^T\nabla^2 f(x)[\sigma_t(x)]\epsilon_t]] \tag{8.22}$$

$$\overset{(ii)}{=} \frac{1}{2}\mathrm{tr}\left(\sigma_t(x)^T\nabla^2 f(x)\sigma_t(x)\right) \tag{8.23}$$

$$\overset{(iii)}{=} \frac{1}{2}\mathrm{tr}(\sigma_t(x)\sigma_t(x)^T\nabla^2 f(x)) \tag{8.24}$$

$$\overset{(iv)}{=} \frac{1}{2}\sigma_t^2(x)\cdot\nabla^2 f(x) \tag{8.25}$$

where in (i) we use a 2nd order Taylor approximation (2nd order because $\mathbb{E}[\|\sqrt{h}\epsilon_t\|^2] \propto h$), in (ii) the identity $\mathrm{tr}(A) = \mathbb{E}_{\epsilon_t}[\epsilon_t^T A\epsilon_t]$ for $A \in \mathbb{R}^{d\times d}$, in (iii) the cyclic property of the trace, and in (iv) the symmetry of σ_t. Further, we use $A \cdot B \coloneqq \mathrm{tr}(A^T B)$ to denote the matrix inner product for matrices $A, B \in \mathbb{R}^{d\times d}$. Therefore, the **diffusion generator** is given by

$$\mathcal{L}_t f(x) = \frac{1}{2}\sigma_t^2(x)\cdot\nabla^2 f(x). \tag{8.26}$$

Jumps. Next, let $\mathcal{S}$ be arbitrary and let us consider a **jump process**. A jump process is defined by a time-dependent kernel $Q_t(\mathrm{d}y, x)$, *i.e.*, for every $0 \le t \le 1$ and every $x \in \mathcal{S}$, $Q_t(\mathrm{d}y, x)$ is a positive measure over $S \setminus \{x\}$. The idea of a jump process is that the total volume assigned to $S \setminus \{x\}$

$$\lambda_t(x) = \int Q_t(\mathrm{d}y, x) \tag{8.27}$$

gives the **jump intensity**, *i.e.*, the infinitesimal likelihood of jumping. Further, if $\lambda_t(x) > 0$, we can assign a **jump distribution** by normalizing Q_t to a probability kernel

$$J_t(\mathrm{d}y, x) = \frac{Q_t(\mathrm{d}y, x)}{\lambda_t(x)}. \tag{8.28}$$

A jump process can be approximated via the infinitesimal sampling procedure as follows:

$$X_{t+h} = \begin{cases} X_t & \text{with probability } 1 - h\lambda_t(X_t) + o(h) \\ \sim J_t(\mathrm{d}y, X_t) & \text{with probability } h\lambda_t(X_t) + o(h) \end{cases} \tag{8.29}$$

For a rigorous treatment of jump processes, see for example (Davis, 1984). The generator is then given by

$$\mathcal{L}_t f(x) = \lim_{h\to 0} \frac{\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t = x]}{h} \tag{8.30}$$

$$= \lim_{h\to 0} \frac{\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t = x, \text{Jump in } [t, t+h)]\mathbb{P}[\text{Jump in } [t, t+h)|X_t = x]}{h} \tag{8.31}$$

$$+ \lim_{h\to 0} \underbrace{\frac{\mathbb{E}[f(X_{t+h}) - f(X_t)|X_t = x, \text{No jump in } [t, t+h)]\mathbb{P}[\text{No jump in } [t, t+h)|X_t = x]}{h}}_{=0} \tag{8.32}$$

$$= \lim_{h\to 0} \frac{\mathbb{E}_{Y\sim J_t(\mathrm{d}y,x)}\left[f(Y) - f(x)\right] h\lambda_t(x)}{h} \tag{8.33}$$

$$= \mathbb{E}_{Y\sim J_t(\mathrm{d}y,x)}\left[f(Y) - f(x)\right]\lambda_t(x) \tag{8.34}$$

$$= \int (f(y) - f(x))Q_t(\mathrm{d}y, x) \tag{8.35}$$

where we have used that if X_t does not jump in $[t, t+h]$, then $X_{t+h} = X_t$. Therefore, the **jump generator** is given by

$$\mathcal{L}_t f(x) = \int (f(y) - f(x)) Q_t(\mathrm{d}y, x) = \lambda_t(x) \mathbb{E}_{Y \sim J_t(\mathrm{d}y, x)}[f(Y) - f(x)]. \tag{8.36}$$

Continuous-time Markov chain (CTMC). Let us consider a continuous-time Markov chain X_t on a discrete space $\mathcal{S}$ with $|\mathcal{S}| < \infty$. In fact, this is simply a jump process on a discrete state space with a specific parameterization. To see this, consider a vanilla jump kernel on a discrete state space $\mathcal{S}$ given by a matrix $Q_t \in \mathbb{R}_{\geq 0}^{\mathcal{S} \times \mathcal{S}}$ and using equation (8.36), the generator is given by

$$\mathcal{L}_t f(x) = \sum_{y \in \mathcal{S}} [f(y) - f(x)] Q_t(y, x) = \sum_{y \neq x} [f(y) - f(x)] Q_t(y, x) \quad \text{for all } x \in \mathcal{S}, f \in \mathbb{R}^{\mathcal{S}} \tag{8.37}$$

i.e., the value of $Q_t(x, x)$ does not matter and is underdetermined. Therefore, a natural convention is to reparameterize the jump kernel on discrete state spaces by rates:

$$u_t(y, x) = \begin{cases} Q_t(y, x) & \text{if } y \neq x \\ -\sum_{z \neq x} Q_t(z; x) & \text{if } y = x \end{cases}$$

With this, we recover the rates $u_t(y, x)$ from section 6 fulfilling the rate conditions in 6.4 by construction. Therefore, this shows that a jump model on a discrete space coincides with the CTMC model (section 6). Applying this on equation (8.37), we get that the **CTMC generator** is given by

$$\mathcal{L}_t f(x) = \sum_{y \in \mathcal{S}} f(y) u_t(y, x) = f^T u_t \tag{8.38}$$

where we consider $f = (f(x))_{x \in \mathcal{S}}$ as a column vector and $u_t \in \mathbb{R}^{\mathcal{S} \times \mathcal{S}}$ as a matrix. Therefore, the generator function is simply vector multiplication from the left.

Flows on manifolds. Next, we consider flows on Riemannian manifolds $\mathcal{S} = \mathcal{M}$ as in section 5. A flow $\psi : [0, 1] \times \mathcal{M} \to \mathcal{M}$ is defined via a vector field $u : [0, 1] \times \mathcal{M} \to T\mathcal{M}$ via the ODE in (3.19). Let us denote the transition from time s to t via $\psi_{t|s}(x) = \psi_t(\psi_s^{-1}(x))$ (as in (3.17)). Then, for a smooth function $f : \mathcal{M} \to \mathbb{R}$ we have that the **Riemannian flow generator** is given via

$$\mathcal{L}_t f(x) = \lim_{h \to 0} \frac{f(\psi_{t+h|t}(x)) - f(x)}{h} = \left\langle \nabla f(x), \frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} \psi_{t+h|t}(x) \right\rangle_g = \langle \nabla f(x), u_t(x) \rangle_g \tag{8.39}$$

where $\langle \cdot, \cdot \rangle_g$ describes the dot product defining the Riemannian metric g and ∇f describes the gradient of f with respect to g. In fact, the generator coincides with the *Lie derivative* of a function (Jost, 2008), a fundamental concept in differential geometry.

8.3 Probability paths and Kolmogorov Equation

For Flow Matching on $\mathcal{S} = \mathbb{R}^d$, the Continuity Equation (see 3.25) is the central mathematical equation that allows us to construct velocity fields that generate a desired probability path (see section 3.5). In this section, we derive a corresponding - more general - equation for CTMPs. Let X_t be a CTMP with generator $\mathcal{L}_t$ and let $X_t \sim p_t$, then we know that:

$$\frac{\mathrm{d}}{\mathrm{d}t} \langle p_t, f \rangle = \frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} \langle p_{t+h}, f \rangle = \frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} \langle p_t, \langle p_{t+h|t}, f \rangle \rangle = \left\langle p_t, \frac{\mathrm{d}}{\mathrm{d}h}\Big|_{h=0} \langle p_{t+h|t}, f \rangle \right\rangle = \langle p_t, \mathcal{L}_t f \rangle$$

where we used that the $\langle p_t, \cdot \rangle$ operation is linear to swap the derivative, and that by the tower property $\langle p_t, \langle p_{t+h|t}, f \rangle \rangle = \langle p_{t+h}, f \rangle$. This shows that given a generator $\mathcal{L}_t$ of a Markov process X_t we can recover its marginal probabilities via their infinitesimal change, *i.e.*, we arrive at the **Kolmogorov Forward Equation (KFE)**

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle p_t, f\rangle = \langle p_t, \mathcal{L}_t f\rangle \text{ for all } f \in \mathcal{T} \tag{8.40}$$

The version of the KFE in equation (8.40) determines the evolution of expectations of test functions f. This is necessary if we use probability distributions that do not have a density. If a density exists, a more familiar version of the KFE can be used that directly prescribes the change of the probability densities. To present it, we introduce the **adjoint generator** $\mathcal{L}_t^*$, which acts on probability densities $p_t(x)$ with respect to a reference measure ν, namely $\mathcal{L}_t^* p_t(x)$ is (implicitly) defined by the identity

$$\int p_t(x)\mathcal{L}_t f(x)\nu(\mathrm{d}x) = \int \mathcal{L}_t^* p_t(x) f(x)\nu(\mathrm{d}x) \qquad \forall f \in \mathcal{T} \tag{8.41}$$

Further, we need to assume that p_t is differentiable in t. Now, (8.41) applied to the KFE (8.40) we get

$$\begin{aligned}
\int \frac{\mathrm{d}}{\mathrm{d}t} p_t(x) f(x)\nu(\mathrm{d}x) &= \frac{\mathrm{d}}{\mathrm{d}t}\int p_t(x) f(x)\nu(\mathrm{d}x) && (8.42)\\
&= \frac{\mathrm{d}}{\mathrm{d}t}\langle p_t, f\rangle && (8.43)\\
&= \langle p_t, \mathcal{L}_t f\rangle && (8.44)\\
&= \int p_t(x)\mathcal{L}_t f(x)\nu(\mathrm{d}x) && (8.45)\\
&= \int \mathcal{L}_t^* p_t(x) f(x)\nu(\mathrm{d}x) && (8.46)
\end{aligned}$$

As this holds for all test functions f, we can conclude using equation (8.5) that this is equivalent to the **adjoint KFE**

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = \mathcal{L}_t^* p_t(x) \text{ for all } x \in \mathcal{S} \tag{8.47}$$

As we will derive in the following examples, the adjoint KFE generalizes many famous equations used to develop generative models such as the Continuity Equation or the Fokker-Planck Equation (Song et al., 2021; Lipman et al., 2022) (see table 2). Whenever a probability density exists, we use the adjoint KFE - to avoid using test functions and work with probability densities directly. We summarize our findings in the following

Theorem 17 (General Mass Conservation)**.** *Let $\mathcal{L}_t$ be a generator of $(X_t)_{0\le t\le 1}$. Informally, the following conditions are equivalent:*

1. *$p_t, \mathcal{L}_t$ satisfies the KFE (8.40).*
2. *$\frac{\mathrm{d}p_t}{\mathrm{d}\nu}(x), \mathcal{L}_t$ satisfy the adjoint KFE (8.47).*
3. *$\mathcal{L}_t$ generates p_t in the sense of equation (8.10).*

Formally, (1) and (2) are equivalent whenever $\frac{\mathrm{d}p_t}{\mathrm{d}\nu}$ exists and is continuously differentiable in t. Further, (3) implies (1) for arbitrary state spaces. There are weak regularity assumptions that ensure that (1) implies (3) (see appendix A.3 for a list). In this work, we assume that these hold, i.e., **we assume that (3) implies (1)**.

To the best of our knowledge, there is no known result for abstract general state spaces that ensures that in theorem 17 condition (3) implies (1). This is why we simply assume it here. For the machine learning researcher, this assumption holds for any state space of interest and should therefore be of no concern (see appendix A.3).

8.3.1 Examples of KFEs

Adjoint KFE for Flows. Let us set $\mathcal{S} = \mathbb{R}^d$ and assume that p_t has a density $p_t(x)$ with respect to the Lebesgue measure that is bounded and continuously differentiable. Then we can compute the adjoint generator $\mathcal{L}_t^*$ via

$$\begin{aligned}
\langle p_t, \mathcal{L}_t f \rangle =& \mathbb{E}_{x\sim p_t}[\mathcal{L}_t f(x)] && (8.48)\\
=& \int \mathcal{L}_t f(x) p_t(x) \mathrm{d}x && (8.49)\\
\overset{(i)}{=}& \int \nabla f(x)^T u_t(x) p_t(x) \mathrm{d}x && (8.50)\\
\overset{(ii)}{=}& \int f(x) \underbrace{[-\mathrm{div}(p_t u_t)(x)]}_{=:\mathcal{L}_t^* p_t(x)} \mathrm{d}x && (8.51)\\
=& \int f(x) \mathcal{L}_t^* p_t(x) \mathrm{d}x && (8.52)
\end{aligned}$$

where (i) follows by equation (8.15) and (ii) by integration by parts. The above derivation shows that the adjoint generator is given by $\mathcal{L}_t^* p_t = -\mathrm{div}(p_t u_t)(x)$ (because it fulfils the condition in equation (8.41)). Using the adjoint KFE, we recover the **Continuity Equation** (see equation (3.25))

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = -\mathrm{div}(p_t u_t)(x), \tag{8.53}$$

an equation that we extensively studied in section 3.4.

Adjoint for diffusion. Let's set $\mathcal{S} = \mathbb{R}^d$ and assume that p_t has a density $p_t(x)$ with respect to the Lebesgue measure that is bounded and continuously differentiable. We can compute the adjoint generator $\mathcal{L}_t^*$ via

$$\begin{aligned}
\langle p_t, \mathcal{L}_t f \rangle =& \mathbb{E}_{x\sim p_t}[\mathcal{L}_t f(x)] && (8.54)\\
=& \int \mathcal{L}_t f(x) p_t(x) \mathrm{d}x && (8.55)\\
\overset{(i)}{=}& \frac{1}{2} \int \sigma_t^2(x) \cdot \nabla^2 f(x) p_t(x) \mathrm{d}x && (8.56)\\
\overset{(ii)}{=}& \int f(x) \underbrace{\frac{1}{2} \nabla^2 \cdot (p_t \sigma_t^2)(x)}_{=:\mathcal{L}_t^* p_t(x)} \mathrm{d}x && (8.57)\\
=& \int f(x) \mathcal{L}_t^* p_t(x) \mathrm{d}x && (8.58)
\end{aligned}$$

by (i) follows by equation (8.26) and (ii) follows by applying integration by parts twice. The above derivation shows that the adjoint generator is given by $\mathcal{L}_t^* p_t = \frac{1}{2}\nabla^2 \cdot (p_t \sigma_t^2)(x)$ (because it fulfils the condition in equation (8.41)). The adjoint KFE then recovers the well-known **Fokker-Planck equation**

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = \frac{1}{2} \nabla^2 \cdot (p_t \sigma_t^2)(x) \tag{8.59}$$

Adjoint KFE for jumps. Let's assume that p_t has a density $p_t(x)$ with respect to the Lebesgue measure that is bounded and continuously differentiable. Let's assume that the jump measures $Q_t(\mathrm{d}y, x)$ is given via a kernel $Q_t : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}, (y, x) \mapsto Q_t(y, x)$ such that

$$\int f(y) Q_t(\mathrm{d}y, x) = \int f(y) Q_t(y, x) \nu(\mathrm{d}y) \text{ for all integrable } f : \mathcal{S} \to \mathbb{R}. \tag{8.60}$$

Then we can derive the adjoint generator as follows:

$$\langle p_t, \mathcal{L}_t f \rangle (x) = \int \int (f(y) - f(x)) Q_t(y, x) \nu(\mathrm{d}y) p_t(x) \nu(\mathrm{d}x) \tag{8.61}$$

$$= \int \int f(y) Q_t(y, x) p_t(x) \nu(\mathrm{d}y) \nu(\mathrm{d}x) - \int \int f(x) Q_t(y, x) p_t(x) \nu(\mathrm{d}y) \nu(\mathrm{d}x) \tag{8.62}$$

$$\overset{(i)}{=} \int \int f(x) Q_t(x, y) p_t(y) \nu(\mathrm{d}y) \nu(\mathrm{d}x) - \int \int f(x) Q_t(y, x) p_t(x) \nu(\mathrm{d}y) \nu(\mathrm{d}x) \tag{8.63}$$

$$= \int f(x) \underbrace{\left[\int Q_t(x, y) p_t(y) - Q_t(y, x) p_t(x) \nu(\mathrm{d}y) \right]}_{=: \mathcal{L}_t^* p_t} \nu(\mathrm{d}x) \tag{8.64}$$

$$= \int f(x) \mathcal{L}_t^* p_t(x) \nu(\mathrm{d}x) \tag{8.65}$$

where in (i) we simply swap the variables x and y. The above derivation shows that $\mathcal{L}_t^*$ as defined above fulfils the condition in equation (8.41) and indeed describes the adjoint generator for jumps. With this, the adjoint KFE becomes the **Jump Continuity equation**:

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = \int \left[Q_t(x, y) p_t(y) - Q_t(y, x) p_t(x) \right] \nu(\mathrm{d}y) = \int \lambda_t(y) J_t(x, y) p_t(y) \nu(\mathrm{d}y) - \lambda_t(x) p_t(x) \tag{8.66}$$

where we use the decomposition $Q_t(y, x) = \lambda_t(x) J_t(y, x)$ into a jump intensity λ_t and a jump distribution J_t (see equation (8.28)).

Adjoint KFE for CTMCs. For $\mathcal{S}$ discrete and generator given by $f^T u_t$ as in equation (8.38), we get that

$$\begin{aligned} \langle p_t, \mathcal{L}_t f \rangle = \int p_t(x) \mathcal{L}_t f(x) \nu(\mathrm{d}x) &= \sum_{x \in \mathcal{S}} p_t(x) \sum_{y \in \mathcal{S}} u_t(y, x) f(y) \\ &= \sum_{y \in \mathcal{S}} \underbrace{\left[\sum_{x \in \mathcal{S}} p_t(x) u_t(y, x) \right]}_{=: \mathcal{L}_t^* p_t(x)} f(y) \\ &= \int \mathcal{L}_t^* p_t(y) f(y) \nu(\mathrm{d}y) \end{aligned}$$

where ν here just denotes the counting measure. Therefore, the adjoint is KFE simply given by

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) = \sum_{y \in \mathcal{S}} u_t(x, y) p_t(y) \tag{8.67}$$

This recovers the KFE for CTMCs as derived in equation (6.8) (with x and y switched to keep consistency with the derivations in this section).

8.4 Universal representation theorem

Generators allow us to characterize the space of possible Markov processes. Specifically, the following result allows us to not only characterize a wide class of CTMP generative models but to characterize the design space *exhaustively* for $\mathcal{S} = \mathbb{R}^d$ or $\mathcal{S}$ discrete.

Theorem 18 (Universal characterization of generators)**.** *Under weak regularity assumptions, the generators of a Feller processes X_t $(0 \leq t \leq 1)$ take the form:*

1. **Discrete $|\mathcal{S}| < \infty$:** *The generator is given by a rate transition matrix u_t and the Markov process corresponds to a continuous-time Markov chain (CTMC).*
2. **Euclidean space $\mathcal{S} = \mathbb{R}^d$:** *The generator has a representation as a sum of components described in table 2, i.e.,*

$$\mathcal{L}_t f(x) = \underbrace{\nabla f(x)^T u_t(x)}_{\text{flow}} + \underbrace{\frac{1}{2}\nabla^2 f(x) \cdot \sigma_t^2(x)}_{\text{diffusion}} + \underbrace{\int [f(y) - f(x)]\, Q_t(\mathrm{d}y, x)}_{\text{jump}} \tag{8.68}$$

where $u : [0,1] \times \mathbb{R}^d \to \mathbb{R}^d$ is a velocity field, $\sigma : [0,1] \times \mathbb{R}^d \to S_d^{++}$ the diffusion coefficient (S_d^{++} denotes the positive semi-definite matrices), and $Q_t(dy; x)$ a jump measure; $\nabla^2 f(x)$ describes the Hessian of f and $\nabla^2 f(x) \cdot \sigma_t^2(x)$ describes the Frobenius inner product.

The proof adapts a known result in the mathematical literature (Courrege, 1965; von Waldenfels, 1965) and can be found in (Holderrieth et al., 2024).

9 Generator Matching

In this section, we describe *Generator Matching* (GM) (Holderrieth et al., 2024), a generative modeling framework for (1) arbitrary data modalities and (2) general Markov processes. GM unifies the vast majority of generative models developed in recent years, including diffusion models, "discrete diffusion" models, and the FM variants described in previous sections. To introduce GM, we defined the CTMP generative model in section 8 that is constructed via a *generator* of a Markov process. GM describes a scalable algorithm to train generators - giving the method its name. Beyond providing a unifying framework, GM gives rise to a variety of new models, allows us to combine models of different classes, as well as allows to build models for arbitrary modalities including models across multiple data modalities.

9.1 Data and coupling

As before, our goal is to transfer samples $X_0 \sim p$ from a distribution p to samples $X_1 \sim q$ from a target distribution q, where $X_0, X_1 \in \mathcal{S}$ are two RVs each taking values in the state space $\mathcal{S}$. Source and target samples can be related by means of the independent coupling $(X_0, X_1) \sim p \otimes q$ (product distribution), or associated by means of a general PMF coupling $\pi_{0,1}$, *i.e.*, distribution over $\mathcal{S} \times \mathcal{S}$ with marginal $\pi_0 = p$ and $\pi_1 = q$. The only difference to before is that $\mathcal{S}$ is a general state space and that p, q can be arbitrary probability measures.

9.2 General probability paths

The next step in the GM recipe is, as before, to prescribe a probability path p_t interpolating p and q. Following section 4.4, we use a **conditional probability path** $p_{t|Z}(\mathrm{d}x|z)$, *i.e.*, a set of time-varying probability measures dependent on a latent state $z \in \mathcal{Z}$. Given a distribution p_Z over $\mathcal{Z}$, we consider the corresponding **marginal probability path** $p_t(\mathrm{d}x)$ defined via the hierarchical sampling procedure:

$$Z \sim p_Z, X_t \sim p_{t|Z}(\mathrm{d}x|z) \quad \Rightarrow \quad X_t \sim p_t(\mathrm{d}x)$$

i.e., we obtain a sample from p_t by first sampling Z from p_Z and then sampling X_t from $p_{t|Z}(\mathrm{d}x|z)$. As before, the marginal probability path is constructed to satisfy the boundary constraints $p_0 = p$ and $p_1 = q$.

We have seen already two common constructions for $\mathcal{Z} = \mathcal{S}$ and $p_Z = q$: First, affine conditional flows for $\mathcal{S} = \mathbb{R}^d$ (as used in continuous FM; section 4) defined via

$$Z \sim q, X_0 \sim p, X_t = \sigma_t X_0 + \alpha_t Z \quad \Rightarrow \quad X_t \sim p_t(\mathrm{d}x) \tag{9.1}$$

where $\alpha_t, \sigma_t \in \mathbb{R}_{\geq 0}$ are differentiable functions satisfying $\alpha_0 = \sigma_1 = 0$ and $\alpha_1 = \sigma_0 = 1$. Second, for arbitrary $\mathcal{S}$, we can use mixtures as used in discrete FM for discrete state spaces (equation (7.22)):

$$Z \sim q, X_0 \sim p, X_t \sim \begin{cases} Z & \text{with prob } \kappa_t \\ X_0 & \text{with prob } (1-\kappa_t) \end{cases} \quad \Rightarrow \quad X_t \sim p_t(\mathrm{d}x) \tag{9.2}$$

where $\kappa_t \in \mathbb{R}_{\geq 0}$ is a differentiable functions satisfying $\kappa_0 = 0$ and $\kappa_1 = 1$ and $0 \leq \kappa_t \leq 1$. One can easily see that the affine conditional and mixture probability paths interpolate p and q, *i.e.*, $p_0 = p$ and $p_1 = q$.

9.3 Parameterizing a generator via a neural network

Given a probability path p_t, our goal is construct a CTMP model specified by a generator $\mathcal{L}_t$ that generates this probability path (see equation (8.10)). To train a neural network for that, we first need to explain how to parameterize a generator $\mathcal{L}_t$ with a neural network $\mathcal{L}_t^\theta$ with parameters θ. We will do this in this section.

Let $\mathcal{T}$ again be a family of test functions (see section 8.2.1). A **linear parameterization** of $\mathcal{L}_t$ is defined as follows: for every $x \in \mathcal{S}$ there is (1) a convex closed set $\Omega_x \subset V_x$ that is a subset of a vector space V_x with an inner product $\langle \cdot, \cdot \rangle_x$ and (2) a linear operator $\mathcal{K} : \mathcal{T} \to C(\mathcal{S}; V_x)$ such that every considered generator $\mathcal{L}_t$ can be written as

$$\mathcal{L}_t f(x) = \langle \mathcal{K} f(x), F_t(x) \rangle_x \tag{9.3}$$

for a function F_t such that $F_t(x) \in \Omega_x$ for every $x \in \mathcal{S}$. Crucially, the operator $\mathcal{K}$ cannot depend on $\mathcal{L}_t$, *i.e.*, only F_t has to be learned. This leads to the

$$\textbf{Parameterized generator: } \mathcal{L}_t^\theta f(x) = \left\langle \mathcal{K} f(x), F_t^\theta(x) \right\rangle_x \text{ with neural network } F_t^\theta \text{ and parameters } \theta, \tag{9.4}$$

where again F_t^θ maps an element $x \in \mathcal{S}$ to $F_t^\theta(x) \in \Omega_x$. We list several examples to make this definition more concrete.

Linear parameterization of flows. Let $\mathcal{S} = \mathbb{R}^d$ and $\Omega_x = \mathbb{R}^d = V_x$. Let's consider all flows, *i.e.*, the family of generators is given by (see equation (8.15)):

$$\mathcal{L}_t f = \nabla f^T u_t, \quad u_t : \mathbb{R}^d \to \mathbb{R}^d. \tag{9.5}$$

Setting $\mathcal{K} f = \nabla f$ and $F_t = u_t$ we recover the shape of equation (9.3). This gives a natural linear parameterization of flow generators via their vector fields.

Linear parameterization of diffusion. Let $\mathcal{S} = \mathbb{R}^d$ and $\Omega_x = S_d^{++} \subset \mathbb{R}^{d \times d} = V_x$, where S_d^{++} denotes the set of all positive semi-definite matrices. Then a diffusion generator is given by (see equation (8.26)):

$$\mathcal{L}_t f = \nabla^2 f \cdot \sigma_t^2, \quad \sigma_t : \mathbb{R}^d \to S_d^{++} \tag{9.6}$$

Setting $\mathcal{K} f = \nabla^2 f$ and $F_t = \sigma_t^2$ we recover the shape of equation (9.3). This gives a natural linear parameterization of diffusion generators.

Linear parameterization of jumps. Let $\Omega_x = \{a : \mathcal{S} \setminus \{x\} \to \mathbb{R}_{\geq 0} \,|\, a \text{ integrable}\} \subset L^2(\mathcal{S} \setminus \{x\}) = V_x$ with dot product $\langle a, b \rangle_x = \int_{\mathcal{S} \setminus \{x\}} a(x) b(x) \nu(dx)$. Then the jump generator is given by (see equation (8.36)):

$$\mathcal{L}_t f(x) = \int [f(y) - f(x)] Q_t(y, x) \nu(\mathrm{d}y) = \langle \mathcal{K} f(x), Q_t(\cdot; x) \rangle_x \tag{9.7}$$

where we set $\mathcal{K} f(x)$ as the function $y \mapsto f(y) - f(x)$. Setting $F_t = Q_t$ we recover the shape of equation (9.3) - giving a linear parameterization of jump generators. We note that the above only parameterizes jumps with a jump kernel $Q_t(y, x)$, which does not necessarily include all jump measures.

Linear parameterization of CTMCs. Let $\mathcal{S}$ be discrete and $u_t \in \mathbb{R}^{\mathcal{S} \times \mathcal{S}}$ be a rate matrix of a continuous-time Markov chain. As for discrete FM (see equation (7.5)), we define

$$\Omega_x = \left\{ v \in \mathbb{R}^{\mathcal{S}} \;\middle|\; v(y) \geq 0 \; \forall y \neq x, \text{ and } v(x) = -\sum_{y \neq x} v(y) \right\} \subset V_x = \mathbb{R}^{\mathcal{S}}. \tag{9.8}$$

Then by equation (8.38), the generator is given for $f \in \mathbb{R}^{\mathcal{S}}$ by

$$\mathcal{L}_t f(x) = f^T u_t(\cdot, x) = \langle f, u_t(\cdot, x) \rangle_x \tag{9.9}$$

where $V_x = \mathbb{R}^S$ and $\mathcal{K} f = f$ and $\langle \cdot, \cdot \rangle_x$ is the standard Euclidean dot product. With this, we recover the shape of equation (9.3). Therefore, this gives a natural linear parameterization of CTMCs via their rates u_t.

Linear parameterization of flows on manifolds. Let $\mathcal{S} = \mathcal{M}$ be a Riemannian manifold and as in section 5, let us consider flows on Riemannian manifolds. By equation (8.39), the generator is given by

$$\mathcal{L}_t f(x) = \langle \nabla f(x), u_t(x) \rangle_g \tag{9.10}$$

with u_t being a time-dependent smooth vector field $u_t : [0, 1] \times \mathcal{M} \to T\mathcal{M}$, and $u_t(x) \in T_x \mathcal{M}$ for all $x \in \mathcal{M}$. Setting $\Omega_x = V_x = T_x \mathcal{M}$ and $\mathcal{K} = \nabla f$ the gradient operator, we recover the shape of equation (9.3). Therefore, this gives a natural linear parameterization of Riemannian flow generators.

9.4 Marginal and conditional generators

In this section, we show how to find generators for marginal probability paths. The recipe is as follows: We can find generators for conditional probability paths $p_{t|Z}(\mathrm{d}x|z)$, often analytically, and use these to construct generators for the marginal path. Specifically, let us assume that for every $z \in \mathcal{Z}$ we found a (conditional) generator $\mathcal{L}^z$ that generates $p_{t|Z}(\mathrm{d}x|z)$, *i.e.*, by theorem 17 this equivalent to the KFE (equation (8.40)):

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\langle p_{t|Z}(\cdot|z), f\right\rangle = \left\langle p_{t|Z}(\cdot|z), \mathcal{L}_t^z f\right\rangle \qquad \text{for all } f \in \mathcal{T}. \tag{9.11}$$

Further, let us assume that we found a linear parameterization (see equation (9.3)) as follows:

$$\mathcal{L}_t^z f(x) = \left\langle \mathcal{K} f(x), F_t(x|z)\right\rangle_x \quad z \in \mathcal{Z} \tag{9.12}$$

for functions $F_t(x|z) \in \Omega_x \subset V_x$. For example, $F_t(x|z)$ could be conditional velocity field in continuous FM (see section 4.3) or the conditional rates in discrete FM (see equation (7.2)). This allows us to find a formula for a generator that generates the marginal path:

Theorem 19 (General Marginalization Trick)**.** *The marginal probability path $(p_t)_{0\le t\le 1}$ is generated by a Markov process X_t with generator*

$$\mathcal{L}_t f(x) = \mathbb{E}_{Z\sim p_{Z|t}(\cdot|x)}[\mathcal{L}_t^Z f(x)] \tag{9.13}$$

where $p_{Z|t}(\mathrm{d}z|x)$ is the posterior distribution (i.e., the conditional distribution of z given x). The generator $\mathcal{L}_t$ has a linear parameterization given by

$$F_t(x) = \mathbb{E}_{Z\sim p_{Z|t}(\cdot|x)}[F_t(x|Z)]. \tag{9.14}$$

The above theorem gives us our training target: to approximate $\mathcal{L}_t$ in equation (9.13) with a neural network. The marginalization tricks seen in previous chapters (theorem 3, theorem 10, theorem 14) are special cases of this theorem. We give a proof here and then show a few examples of novel instantiations.

Proof. To prove that $\mathcal{L}_t$ generates p_t, we need to show by theorem 17 that the KFE is fulfilled. Let $p_{t+h|t}(\cdot|x,z)$ be the transition kernel of $\mathcal{L}_t^z$. Then:

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\left\langle p_t, f\right\rangle &= \lim_{h\to 0}\frac{1}{h}\left(\left\langle p_{t+h}, f\right\rangle - \left\langle p_t, f\right\rangle\right)\\
&= \lim_{h\to 0}\frac{1}{h}(\mathbb{E}_{Z\sim p_Z, X'\sim p_{t+h|Z}(\cdot|Z)}(f(X')) - \mathbb{E}_{Z\sim p_Z, X\sim p_{t|Z}(\cdot|z)}(f(X)))\\
&= \lim_{h\to 0}\frac{1}{h}(\mathbb{E}_{Z\sim p_Z, X\sim p_{t|Z}(\cdot|Z), X'\sim p_{t+h|t}(\cdot|X,Z)}(f(X')) - \mathbb{E}_{Z\sim p_Z, X\sim p_{t|Z}(\cdot|z)}(f(X)))\\
&= \lim_{h\to 0}\frac{1}{h}(\mathbb{E}_{Z\sim p_Z, X\sim p_{t|Z}(\cdot|Z), X'\sim p_{t+h|t}(\cdot|X,Z)}(f(X') - f(X)))\\
&= \lim_{h\to 0}\frac{1}{h}\mathbb{E}_{X\sim p_t}\left(\mathbb{E}_{Z\sim p_{t|Z}(\cdot|X)}\left(\left(\mathbb{E}_{(X'\sim p_{t+h|t}(\cdot|X,Z)}(f(X')) - f(X)\right)\right)\right)\\
&= \mathbb{E}_{X\sim p_t}\left(\mathbb{E}_{Z\sim p_{Z|t}(\cdot|X)}\left(\lim_{h\to 0}\frac{1}{h}\left(\mathbb{E}_{X'\sim p_{t+h|t}(\cdot|X,Z)}(f(X')) - f(X)\right)\right)\right)\\
&= \mathbb{E}_{X\sim p_t}(\underbrace{\mathbb{E}_{Z\sim p_{Z|t}(\cdot|X)}(\mathcal{L}_t^z f(X))}_{=:\mathcal{L}_t f(X)})\\
&= \left\langle p_t, \mathcal{L}_t f\right\rangle
\end{aligned}$$

The proof for the form of F_t follows then by

$$\begin{aligned}
\mathbb{E}_{Z\sim p_{Z|t}(\cdot|X)}(\mathcal{L}_t^z f(x)) \overset{(9.12)}{=} \mathbb{E}_{Z\sim p_{Z|t}(\cdot|X)}(\left\langle \mathcal{K} f(x), F_t(x|z)\right\rangle_x) &= \left\langle \mathcal{K} f(x), \mathbb{E}_{Z\sim p_{Z|t}(\cdot|X)}(F_t(x|z))\right\rangle_x\\
&= \left\langle \mathcal{K} f(x), F_t(x)\right\rangle_x
\end{aligned}$$

where we use the linearity of the dot product to swap it with the expected value. This shows that F_t is a linear parameterization (see equation (9.3)) of the marginal generator. □

Example - Jumps. Let $\mathcal{S}$ be arbitrary and $Q_t(y, x|z)$ be a conditional jump kernel on $\mathcal{S}$ for $y, x \in \mathcal{S}, z \in \mathcal{Z}$ generating the conditional probability path $p_{t|Z}(\mathrm{d}x|z)$. Using the linear parameterization of the jump kernel (see equation (9.7)), we get that the **marginal jump kernel**

$$Q_t(y, x) = \mathbb{E}_{Z \sim p_{Z|t}(\cdot|x)}[Q_t(y, x|z)].$$

generates the marginal probability $p_t(\mathrm{d}x)$.

Example - Marginal diffusion coefficient. Let $\mathcal{S} = \mathbb{R}^d$ and $\sigma_t^2(x|z)$ be a diffusion coefficient generating the conditional probability path $p_{t|Z}(\mathrm{d}x|z)$. Using the linear parameterization of the diffusion coefficient (see equation (9.6)), we get that the **marginal diffusion coefficient**

$$\sigma_t^2(x) = \mathbb{E}_{Z \sim p_{Z|t}(\cdot|x)}[\sigma_t^2(x|Z)]$$

generates the marginal probability path $p_t(\mathrm{d}x)$.

9.5 Generator Matching loss

Our next goal is to develop a training objective for learning a CTMP model. Let us assume that we have a neural network F_t^θ that gives us a generator parameterization $\mathcal{L}_t^\theta$ as in equation (9.4). As derived in theorem 19, our goal is to approximate the true marginal linear parameterization F_t given by equation (9.14).

As before, let us assume that for every $x \in \mathcal{S}$ we have a Bregman divergence $D_x : \Omega_x \times \Omega_x \to \mathbb{R}$ defined via

$$D_x(a, b) = \Phi_x(a) - [\Phi_x(b) + \langle a - b, \nabla\Phi_x(b)\rangle], \quad a, b \in \Omega_x \tag{9.15}$$

for a strictly convex function $\Phi_x : \Omega_x \to \mathbb{R}$ (see figure 10). The **Generator Matching loss** to train the CTMP model is defined as

$$\mathcal{L}_{\text{GM}}(\theta) = \mathbb{E}_{t, X_t \sim p_t} D_{X_t}(F_t(X_t), F_t^\theta(X_t)), \tag{9.16}$$

for $t \sim U[0, 1]$. Unfortunately, the above training objective is intractable as we do not know the marginal generator $\mathcal{L}_t$ and also not the parameterization F_t thereof (we only know the intractable formula in equation (9.14)). Hence, we introduce the **Conditional Generator Matching loss** as a tractable alternative that takes the form

$$\mathcal{L}_{\text{CGM}}(\theta) = \mathbb{E}_{t, Z, X_t \sim p_{t|Z}} D_{X_t}(F_t(X_t|Z), F_t^\theta(X_t)). \tag{9.17}$$

This objective is tractable as we can derive $F_t(x|z)$ analytically in many cases (see section 9.6). As the next theorem shows, the two losses (9.16) and (9.17) both provide the same learning gradients.

Theorem 20. *The gradients of the Generator Matching loss and the Conditional Generator Matching loss coincide:*

$$\nabla_\theta \mathcal{L}_{GM}(\theta) = \nabla_\theta \mathcal{L}_{CGM}(\theta). \tag{9.18}$$

In particular, the minimizer of the Conditional Generator Matching loss is the linear parameterization of the marginal generator (equation (9.14)):

$$F_t^\theta(x) = \mathbb{E}_{Z \sim p_{Z|t}(\cdot|x)}[F_t(x|Z)]. \tag{9.19}$$

Furthermore, for these properties to hold, D_x must **necessarily** *be a Bregman divergence.*

The above theorem generalizes theorem 4, theorem 11, and theorem 15 derived in previous sections to general CTMP models. It allows us to easily train any CTMP model parameterized by a neural network F_t^θ in a scalable fashion by minimizing the Conditional Generator Matching loss. In addition, it universally characterizes the space of loss functions. The proof of theorem 20 is identical as the proof of theorem 4 with u_t replaced by F_t. For the proof of the necessity of D being a Bregman divergence, we refer to (Holderrieth et al., 2024).

Example - Training a diffusion coefficient. We illustrate how theorem 20 allows us to train a diffusion coefficient of an SDE. Let $\mathcal{S} = \mathbb{R}^d$ and $\sigma_t^2(x|z)$ be a diffusion coefficient generating the conditional probability path $p_{t|Z}(\mathrm{d}x|z)$. We can parameterize the diffusion coefficient in a neural network $(\sigma_t^2)^\theta(x) \in \mathbb{R}^{d\times d}$. The Conditional Generator Matching loss then becomes

$$\mathcal{L}_{\mathrm{CGM}}(\theta) = \mathbb{E}_{t,Z,X_t\sim p_{t|Z}} \|\sigma_t^2(X_t|Z) - (\sigma_t^2)^\theta(X_t)\|^2$$

where we used the mean squared error as a Bregman divergence (many other options are possible). In (Holderrieth et al., 2024), examples are shown of models trained in this manner.

9.6 Finding conditional generators as solutions to the KFE

To enable scalable training with the Conditional Generator Matching loss (see theorem 20), we need to be able to find a conditional generator $\mathcal{L}_t^z$ that solves the KFE

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle p_{t|Z}(\cdot|z), f\rangle = \langle p_{t|Z}(\cdot|z), \mathcal{L}_t^z f\rangle \qquad \text{for all } f \in \mathcal{T}, z \in \mathcal{Z}. \tag{9.20}$$

If $p_{t|Z}(\mathrm{d}x|z)$ admits a density $p_{t|Z}(x|z)$ with respect to ν. In this case, we can equivalently solve the adjoint KFE

$$\frac{\mathrm{d}}{\mathrm{d}t}p_{t|Z}(x|z) = [(\mathcal{L}_t^z)^* p_{t|Z}(\cdot|z)](x) \qquad \text{for all } x \in \mathcal{S}, z \in \mathcal{Z}. \tag{9.21}$$

In general, equation (9.20) and equation (9.21) are challenging equations to solve analytically and there is no general formula to solve it for arbitrary generators. Therefore, we give 2 examples on how this can be done as illustration.

We illustrate it with jump models here, as these work for arbitrary state spaces. As explained in section 8.2.2, they are specified by a jump measure Q_t that can be decomposed into

$$Q_t(\mathrm{d}y, x) = \lambda_t(x) J_t(\mathrm{d}y, x) \qquad \text{for all } x \in \mathcal{S} \tag{9.22}$$

$$\lambda_t(x) \geq 0 \qquad \text{for all } x \in \mathcal{S} \tag{9.23}$$

$$\int J_t(\mathrm{d}y, x) = 1 \qquad \text{for all } x \in \mathcal{S} \tag{9.24}$$

where $\lambda_t(x)$ describes the jump intensity and J_t describes a probability kernel specifying the jump distribution. Note that we drop the dependency on $z \in \mathcal{Z}$ to simplify notation (we keep it for $p_{t|Z}(\mathrm{d}x|z)$ to avoid confusion with the marginal probability path).

Jump models for convex mixtures. Let us consider the mixture probability path given by (see equation (7.22)):

$$p_{t|Z}(\mathrm{d}x|z) = \kappa_t \delta_z(\mathrm{d}x) + (1-\kappa_t)p(\mathrm{d}x), \quad z \in \mathcal{S}. \tag{9.25}$$

Using the form of the generator for jump processes (see equation (8.36)), the KFE becomes:

$$\frac{\mathrm{d}}{\mathrm{d}t}\langle p_{t|Z}(\mathrm{d}x|z), f\rangle = \mathbb{E}_{X\sim p_{t|Z}(\cdot|z)}\lambda_t(X)\mathbb{E}_{Y\sim J_t(\mathrm{d}y,x)}[f(Y) - f(X)] \quad \text{for all } f \in \mathcal{T}, x \in \mathcal{S} \tag{9.26}$$

for λ_t, J_t satisfying the constraints in equation (9.23) and equation (9.24). We make the claim that this is satisfied for a jump model with

$$Q_t(\mathrm{d}y, x) = \lambda_t(x) J_t(\mathrm{d}y, x), \quad \lambda_t(x) = \frac{\dot{\kappa}_t}{1-\kappa_t}, \quad J_t(\mathrm{d}y, x) = \delta_z(\mathrm{d}y)$$

i.e., the jump intensity is given by λ_t and once we decided to jump, we jump straight to $z \in \mathcal{S}$. To show this, we show that the above jump process fulfils the KFE. We can derive:

$$\begin{aligned}
\mathbb{E}_{X\sim p_{t|Z}(\cdot|z)}\left[\lambda_t(X)\mathbb{E}_{Y\sim J_t(\cdot,X)}[f(Y)-f(X)]\right] &= \frac{\dot\kappa_t}{1-\kappa_t}\mathbb{E}_{X\sim p_{t|Z}(\cdot|z)}[f(z)-f(X)] \\
&= \frac{\dot\kappa_t}{1-\kappa_t}[f(z)-\mathbb{E}_{X\sim p_{t|Z}(\cdot|z)}[f(X)]] \\
&= \frac{\dot\kappa_t}{1-\kappa_t}[f(z)-[\kappa_t f(z)+(1-\kappa_t)\mathbb{E}_{X\sim p}f(X)] \\
&= \dot\kappa_t f(z) - \dot\kappa_t \mathbb{E}_{X\sim p}f(X) \\
&= \frac{\mathrm{d}}{\mathrm{d}t}\left[\kappa_t f(z)+(1-\kappa_t)\mathbb{E}_{x\sim p}[f(x)]\right] \\
&= \frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}_{X\sim p_{t|Z}(\cdot|z)}[f(X)] \\
&= \frac{\mathrm{d}}{\mathrm{d}t}\left\langle p_{t|Z}(\cdot|z), f\right\rangle .
\end{aligned}$$

Therefore, we see that the process fulfills the jump KFE (equation (9.26)). Therefore, we have established a jump model. We have seen a special example of that model for discrete state spaces in equation (7.24). Here, we have shown that one could also build a similar jump model for Euclidean space $\mathbb{R}^d$, for example.

Jump models for arbitrary paths with densities. Let us assume that we have a probability $p_{t|Z}(\mathrm{d}x|z)$ that admits a density $p_{t|Z}(x|z)$ with respect to a reference measure ν on $\mathcal{S}$ and that is differentiable in t (note that the mixture path in equation (9.25) would not fulfill that for $\mathcal{S}=\mathbb{R}^d$). Further, we restrict ourselves to jump kernels $J_t(y,x)$ that admit a density. With this, the adjoint KFE becomes the Jump Continuity Equation (equation (8.66)):

$$\frac{\mathrm{d}}{\mathrm{d}t}p_{t|Z}(x|z) = \int \lambda_t(y)J_t(x,y)p_{t|Z}(y|z)dy - p_{t|Z}(x|z)\lambda_t(x) \tag{9.27}$$

$$\Leftrightarrow \quad p_{t|Z}(x|z)\left[\frac{\mathrm{d}}{\mathrm{d}t}\log p_{t|Z}(x|z)+\lambda_t(x)\right] = \int \lambda_t(y)J_t(x,y)p_{t|Z}(y|z)dy \tag{9.28}$$

Making $J_t(x,y)=J_t(x)$ ("target-state-independent") and using $\partial_t = \frac{\mathrm{d}}{\mathrm{d}t}$, we get that this equivalent to:

$$p_{t|Z}(x|z)\left[\partial_t \log p_{t|Z}(x|z)+\lambda_t(x)\right] = J_t(x)\int \lambda_t(y)p_{t|Z}(y|z)\nu(dy) \tag{9.29}$$

$$\Leftrightarrow \quad \frac{p_{t|Z}(x|z)\left[\partial_t \log p_{t|Z}(x|z)+\lambda_t(x)\right]}{\int \lambda_t(y)p_{t|Z}(y|z)\nu(dy)} = J_t(x) \tag{9.30}$$

In order to define a valid jump process, we require λ_t, J_t to satisfy $\lambda_t(x)\geq 0$, $J_t(x)\geq 0$. Therefore, we get:

$$\lambda_t(x)\geq 0,\ J_t(x)\geq 0 \quad \Leftrightarrow \quad \lambda_t(x) \geq [-\partial_t \log p_t(x|z)]_+ \tag{9.31}$$

where $[x]_+ = \max(x,0)$ describes the ReLU operation. Further, we require J_t to define a valid jump distribution, *i.e.*, integrate to 1. This can be seen to hold:

$$\begin{aligned}
1 &= \int J_t(x)dx \\
\Leftrightarrow \quad \int \lambda_t(x)p_{t|Z}(x|z)\nu(\mathrm{d}x) &= \int p_{t|Z}(x|z)\left[\partial_t \log p_{t|Z}(x|z)+\lambda_t(x)\right]\nu(\mathrm{d}x) \\
\Leftrightarrow \quad 0 &= \int \partial_t p_{t|Z}(x|z)\nu(dx) \\
\Leftrightarrow \quad 0 &= \partial_t \int p_{t|Z}(x|z)\nu(dx) \\
\Leftrightarrow \quad 0 &= 0
\end{aligned}$$

i.e., J_t indeed integrates to 1. Choosing the minimal $\lambda_t(x)$, we get that a jump model defined via

$$\lambda_t(x) = \left[-\partial_t \log p_{t|Z}(x|z)\right]_+ ,$$
$$J_t(x) = \frac{p_{t|Z}(x|z)[\partial_t \log p_{t|Z}(x|z)]_+}{\int p_{t|Z}(y|z)[\partial_t \log p_{t|Z}(y|z)]_+ \nu(\mathrm{d}y)} = \frac{[\partial_t p_{t|Z}(x|z)]_+}{\int [\partial_t p_{t|Z}(y|z)]_+ \nu(\mathrm{d}y)}$$

is a solution to the jump continuity equation and therefore generates the conditional probability path $p_{t|Z}(x|z)$. At first, it seems dissatisfying that jump distribution is independent of the location. However, if we extend this model to multiple dimensions, the jump distribution will depend on the location and leads to a powerful generation model (Holderrieth et al., 2024).

9.7 Combining Models

In this section, we explain how GM allows us to combine generative models for the same state space $\mathcal{S}$ in different ways. The underlying principle is simple: the generator is a linear operator and the KFE $\partial_t \langle p_t, f\rangle = \langle p_t, \mathcal{L}_t f\rangle$ is a linear equation - so essentially, we can combine solutions for this equation like we do for matrix equations in linear algebra. Specifically, let $\mathcal{L}_t, \mathcal{L}'_t$ be two generators of two Markov processes that solve the KFE for a probability path p_t. Then for $\alpha_t^1, \alpha_t^2 \in \mathbb{R}$ with $\alpha_t^1 + \alpha_t^2 = 1$ it holds that:

$$\left\langle p_t, (\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \mathcal{L}'_t) f \right\rangle \tag{9.32}$$
$$= \alpha_t^1 \left\langle p_t, \mathcal{L}_t f\right\rangle + \alpha_t^2 \left\langle p_t, \mathcal{L}'_t f\right\rangle \tag{9.33}$$
$$= \alpha_t^1 \partial_t \left\langle p_t, f\right\rangle + \alpha_t^2 \partial_t \left\langle p_t, f\right\rangle \tag{9.34}$$
$$= (\alpha_t^1 + \alpha_t^2)\partial_t \left\langle p_t, f\right\rangle \tag{9.35}$$
$$= \partial_t \left\langle p_t, f\right\rangle , \tag{9.36}$$

i.e., $\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \mathcal{L}'_t$ is again a solution of the KFE. A small but important detail is whether α_t^1, α_t^2 are positive or negative and whether $\mathcal{L}_t, \mathcal{L}'_t$ correspond to Markov processes in forward or backward time. This leads to the following

Proposition 3 (Combining models)**.** *Let p_t be a marginal probability path, then the following generators solve the KFE for p_t and consequently define a generative model with p_t as marginal:*

1. **Markov superposition:** *$\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \mathcal{L}'_t$, where $\mathcal{L}_t, \mathcal{L}'_t$ are two generators of Markov processes solving the KFE for p_t, and $\alpha_t^1, \alpha_t^2 \geq 0$ satisfy $\alpha_t^1 + \alpha_t^2 = 1$.*
2. **Divergence-free components:** *$\mathcal{L}_t + \beta_t \mathcal{L}_t^{\mathrm{div}}$, where $\mathcal{L}_t^{\mathrm{div}}$ is a generator such that $\left\langle p_t, \mathcal{L}_t^{\mathrm{div}} f\right\rangle = 0$ for all $f \in \mathcal{T}$, and $\beta_t \geq 0$. We call such $\mathcal{L}_t^{\mathrm{div}}$* **divergence-free**.
3. **Predictor-corrector:** *$\alpha_t^1 \mathcal{L}_t + \alpha_t^2 \bar{\mathcal{L}}_t$, where $\mathcal{L}_t$ is a generator solving the KFE for p_t in forward-time and $\bar{\mathcal{L}}_t$ is a generator solving the KFE in backward time, and $\alpha_t^1, \alpha_t^2 \geq 0$ with $\alpha_t^1 - \alpha_t^2 = 1$.*

We give examples for a Markov superposition and a divergenc-free component here to illustrate proposition 3. An example of the power of a predictor-corrector scheme can be found in (Gat et al., 2024).

Markov superposition example - combining jump and flow. Markov superpositions can be used to combine generative models of different classes. These can be 2 networks trained separately or we can train two GM models in one network simultaneously (Holderrieth et al., 2024). We illustrate this here by combining a jump model and a flow model on $\mathcal{S} = \mathbb{R}^d$. Let us assume that we have two models where each of them generates the probability path p_t: (1) a flow model u_t and (2) a jump model with jump intensity λ_t and jump distribution J_t. By proposition 3, for $\alpha_t^1, \alpha_t^2 \geq 0$ with $\alpha_t^1 + \alpha_t^2 = 1$, it holds that the following generator defines a valid GM model generating p_t:

$$\mathcal{L}_t f(x) = \alpha_t^1 \mathcal{L}_t^{\mathrm{jump}} f(x) + \alpha_t^2 \mathcal{L}_t^{\mathrm{flow}} f(x)$$
$$= (\alpha_t^1 \lambda_t(x)) \mathbb{E}_{Y \sim J_t(\cdot, x)}[f(Y) - f(x)] + \nabla f^T(x)(\alpha_t^2 u_t(x))$$

where we have used equation (8.17) and equation (8.36). In fact, the above generator describes a **piecewise-deterministic Markov process**, a combined ODE and jump model (Davis, 1984). As the equation above shows,

we have to scale the jump intensity by α_t^1 and the vector field by α_t^2. We can sample from the resulting GM model with the following sampling procedure:

$$X_0 \sim p_0 = p$$
$$X_{t+h} = \begin{cases} \sim J_t(\mathrm{d}y, X_t) & \text{with probability } h\alpha_t^1 \lambda_t(X_t) \\ X_t + h\alpha_t^2 u_t(X_t) & \text{with probability } 1 - h\alpha_t^1 \lambda_t(X_t) \end{cases}$$

In (Holderrieth et al., 2024), several examples of Markov superpositions of jump and flow are given and shown to lead to performance improvements.

Divergence-free example - MCMC algorithms. To find divergence-free components, one can use existing Markov-Chain Monte-Carlo (MCMC) algorithms - all of these algorithms describe a general recipe to find a divergence-free component. We illustrate this with 2 famous examples. Let us assume that we are given a general probability path p_t with density $p_t(x)$. Then for a generator $\mathcal{L}_t^{\text{div}}$ to be divergence-free is equivalent to that its adjoint maps p_t to zero:

$$\left\langle p_t, \mathcal{L}_t^{\text{div}} f \right\rangle = 0 \text{ for all } f \in \mathcal{T} \quad \Leftrightarrow \quad [\mathcal{L}_t^{\text{div}}]^* p_t(x) = 0 \text{ for all } x \in \mathcal{S} \tag{9.37}$$

First, let us consider $\mathcal{S} = \mathbb{R}^d$. **Langevin dynamics** correspond to an SDE with velocity field $\frac{1}{2}\beta_t^2 \nabla \log p_t(x)$ and diffusion coefficient β_t, *i.e.*, the dynamics are given via

$$dX_t = \frac{1}{2}\beta_t^2 \nabla \log p_t(x) dt + \beta_t dW_t \tag{9.38}$$

The adjoint generator of this SDE is given by

$$\begin{aligned} [\mathcal{L}_t^{\text{div}}]^* p_t &\overset{(i)}{=} -\operatorname{div}(p_t \frac{1}{2}\beta_t^2 \nabla \log p_t)(x) + \frac{1}{2}\beta_t^2 \Delta p_t(x) \\ &\overset{(ii)}{=} -\frac{1}{2}\operatorname{div}(\beta_t^2 \nabla p_t)(x) + \frac{1}{2}\beta_t^2 \Delta p_t(x) \\ &\overset{(iii)}{=} -\frac{1}{2}\beta_t^2 \Delta p_t(x) + \frac{1}{2}\beta_t^2 \Delta p_t(x) = 0 \end{aligned}$$

where (i) holds by shape of flow and diffusion adjoint derived in section 8.3.1, (ii) holds because $\nabla \log p_t = \nabla p_t / p_t$, and (iii) holds by the identity $\operatorname{div}\nabla = \Delta$. The above shows that the generator of Langevin dynamics fulfils equation (9.37) and is therefore divergence-free in sense of proposition 3. This fact is widely applied in statistical physics and Markov chain Monte Carlo (Roberts and Tweedie, 1996). Proposition 3 shows that we can add these dynamics for arbitrary $\beta_t \geq 0$ to any generative model. In section 10, we use this to derive stochastic sampling for diffusion models.

Second, let $\mathcal{S}$ be a general state space again. The **Metropolis-Hastings** algorithm (Hastings, 1970) wdescribes the construction of a jump process with jump kernel $Q_t(y, x)$ that satisfies the **detailed balance condition**:

$$\begin{aligned} & Q_t(y, x) p_t(x) = Q_t(x, y) p_t(y) \quad \text{for all } x, y \in \mathcal{S} \\ \Rightarrow \quad & [\mathcal{L}_t^{\text{div}}]^* p_t(x) \overset{(i)}{=} \int Q_t(y, x) p_t(x) - Q_t(x, y) p_t(y) = 0 \end{aligned}$$

where in (i) we used equation (8.66). This shows that equation (9.37) is fulfilled and Q_t is divergence-free. Proposition 3 shows that one can arbitrarily add such a Metropolis-scheme to any GM model following the probability path p_t.

9.8 Multimodal models

We finally comment on how GM enables the construction of generative models over multiple data modalities jointly. For example, this could be a model that generates images and corresponding text descriptions at the same time. Two modalities are represented as two state spaces $\mathcal{S}_1, \mathcal{S}_2$ (*e.g.*, $\mathcal{S}_1$ are images and $\mathcal{S}_2$ is text) and a multimodal model would be a generative model over the product space $\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$. As $\mathcal{S}$ is just another state space and GM works for arbitrary state spaces, we could simply go about it like constructing any other GM model. However, there is a specific construction of probability paths that allows us to reuse or "recycle" GM models built for individual modalities. For example, we could build a joint text-image model by combining a discrete and continuous FM model. This specific construction relies on **factorized conditional probability paths**. We have seen a simple case of this already in section 7.5.2 for discrete FM where factorized probability paths lead to factorized velocities. This holds more generally for arbitrary modalities. While the construction is rather simple and intuitive, it is rather technical to express in full generality. We refer to (Holderrieth et al., 2024) for a rigorous treatment. A specific instantiation of this was also realized in (Campbell et al., 2024) for multimodal protein generation. This highlights that GM enables the construction of multimodal models in a principled and rigorous manner.

10 Relation to Diffusion and other Denoising Models

In this section, we finally discuss the relation to denoising diffusion models (DDMs) and related models on non-Euclidean spaces. We mainly focus here on the construction of diffusion models by (Song et al., 2021) via SDEs and explain how it can be placed with the FM/GM family of models. At the end of the section, we also discuss models in other modalities that took inspiration from DDMs ("denoising models") and how they can be framed as a GM model.

10.1 Time convention

The first simple difference between denoising diffusion models and flow matching is a difference how time is parameterized. This is just a convention but is important to avoid confusion. Different to FM, time is inverted in diffusion models and ranges from 0 to ∞. To differentiate the two time parameterizations, let us use r for the time convention of diffusion models and t for the time convention of FM. Then we have:

$$\textbf{FM time } t\text{: Noise} \equiv \text{“}t = 0\text{”}, \text{Data} \equiv \text{“}t = 1\text{”} \tag{10.1}$$
$$\textbf{Diffusion time } r\text{: Noise} \equiv \text{“}r = +\infty\text{”}, \text{Data} \equiv \text{“}r = 0\text{”} \tag{10.2}$$
$$\textbf{Reparameterization: } r = k(t), \quad t = k^{-1}(r) \tag{10.3}$$

where $k : (0, 1] \to [0, +\infty)$ is some strictly monotonically decreasing mapping with $k(1) = 0$ and $\lim_{t\to 0} k(t) = +\infty$.

10.2 Forward process vs. probability paths

The underlying idea of denoising diffusion models is to construct a *forward process* that corrupts the data distribution. We will explain how this corresponds to a specific construction of a probability path as used in FM. The forward process X_r is defined via the SDE

$$\mathrm{d}X_r = a_r(X_r)\mathrm{d}r + g_r\mathrm{d}W_r, \quad X_0 \sim q \tag{10.4}$$

where q is the data distribution, W_r is a Brownian motion and $a : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^d$ a velocity field, also called *drift* in the context of SDEs, and $g : \mathbb{R} \to \mathbb{R}_{\geq 0}$ a diffusion coefficient (see section 8.2.2). Every such SDE defines a conditional probability path and marginal probability path as

$$\tilde{p}_{r|0}(x|z) = \mathbb{P}[X_t = x|X_0 = z], \qquad \tilde{p}_r(x) = \mathbb{P}[X_t = x] \tag{10.5}$$
$$p_{t|1}(x|z) = \tilde{p}_{k(t)|0}(x|z), \qquad p_t(x) = \tilde{p}_{k(t)}(x) \tag{10.6}$$

where in the second line we reparameterized time into the FM time parameterization. We see that $p_{t|1}(x|z)$ gives a conditional probability path. Further, the forward process is constructed such that for $R \gg 0$, the distribution of X_R is approximately a Gaussian. Therefore, we get that:

Every forward process in diffusion defines a "valid" conditional probability path as used in FM, *i.e.*, the corresponding marginal path interpolates between a data distribution q and (roughly) a Gaussian p. Specifically:
(1) **Deterministic initialization:** The conditional probability path $p_{t|1}(x|z)$ corresponds to the distribution of the forward process SDE when initialized with $X_0 = z$.
(2) **Data initialization:** The marginal probability path $p_t(x)$ corresponds to the distribution of the forward process when initialized with $X_0 \sim q$ where q is the data distribution.

An important requirement of diffusion models is that one can compute the conditional probability $\tilde{p}_{r|0}(x|z)$ in closed form. This enforces working with SDEs that have analytic solution to their respective KFE (*i.e.*, Fokker-Planck equation). Throughout most of the literature, the forward process is therefore assumed to have **affine drift coefficients** *i.e.*, $a_r(x) = a_r x$ for some continuous function $a : \mathbb{R} \to \mathbb{R}$ (Song et al., 2021; Karras

et al., 2022). This assumption allows us to express the conditional distribution of X_r given $X_0 = z \in \mathbb{R}^d$ as a Gaussian distribution (Särkkä and Solin, 2019; Song et al., 2021; Karras et al., 2022):

$$\tilde{p}_{r|0}(x|z) = \mathcal{N}\left(\tilde{\alpha}_r z, \tilde{\sigma}_r^2 I\right), \quad \tilde{\alpha}_r = \exp\left(\int_0^r a_w \mathrm{d}w\right), \quad \tilde{\sigma}_r^2 = \tilde{\alpha}_r^2 \int_0^r \frac{g_w^2}{\tilde{\alpha}_w^2} dw \tag{10.7}$$

$$\Rightarrow \quad p_{t|1}(x|z) = \mathcal{N}\left(\alpha_t z, \sigma_t^2 I\right) \qquad \alpha_t = \tilde{\alpha}_{k(t)}, \qquad \sigma_t = \tilde{\sigma}_{k(t)} \tag{10.8}$$

Note that we have discussed such probability paths extensively in section 4.8.3 as **affine Gaussian probability paths**, *i.e.*, they are constructed via the affine conditional flow (see section 4.8):

$$\psi_t(x|x_1) = \alpha_t z + \sigma_t x, \quad z \sim q, \; x \sim \mathcal{N}(\alpha_0, \sigma_0^2 I). \tag{10.9}$$

Therefore, we can see that:

> Forward processes with affine drift coefficients correspond to using **Gaussian probability paths** (see section 4.8.3) defined by equation (10.8).

Note that for diffusion models in the above time parameterization, there is no finite times $r < +\infty$ at which the marginal $\tilde{p}_r(x)$ is an exact Gaussian.

10.3 Training a diffusion model

We now discuss how we can recover the training algorithm of diffusion models as a special case of FM training. In section 4.8, we discussed several options of how to parameterize and train a FM model with Gaussian probability paths (see theorem 7). One particular option was **x_0-prediction** where the neural network $x_{0|t}^\theta$ is trained to approximate

$$x_{0|t}^\theta \approx \mathbb{E}[X_0|X_t = x]$$

via the following training algorithm

$$\begin{aligned}
\mathcal{L}_{\text{CM}}(\theta) &= \mathbb{E}_{t, Z\sim q, X_0 \sim p} \| x_{0|t}^\theta(\underbrace{\alpha_t X_0 + \sigma_t Z}_{=X_t}) - X_0 \|^2 \\
&\overset{(i)}{=} \mathbb{E}_{t, Z\sim q, X_t \sim p_{t|1}(\cdot|Z)} \sigma_t^2 \| s_t^\theta(X_t) - [-\frac{1}{\sigma_t^2}(X_t - \alpha_t Z)] \|^2 \\
&\overset{(ii)}{=} \mathbb{E}_{t, Z\sim q, X_t \sim p_{t|1}(\cdot|Z)} \sigma_t^2 \| s_t^\theta(X_t) - \nabla \log p_{t|1}(X_t|Z) \|^2,
\end{aligned}$$

where in (i) we reparameterized the neural network via $s_t^\theta = -x_{0|t}^\theta / \sigma_t$ and in (ii) we used equation (4.71). The above loss is also called the **Denoising Score Matching** (Vincent, 2011) loss and is the fundamental loss function with which diffusion models are trained. By theorem 7, we get that the minimizer θ^* fulfills that

$$s_t^{\theta^*}(x) = -\frac{1}{\sigma_t} \mathbb{E}[X_0|X_t = x] \overset{(4.77)}{=} \nabla \log p_t(x), \tag{10.10}$$

i.e., at minimal loss, s_t^θ equals the score function $\nabla \log p_t(x)$ of the marginal probability path. Therefore, the network s_t^θ is also called the score network. We can therefore summarize:

> The training algorithm for diffusion models is **equivalent** to training a specific FM model with x_0-prediction. Specifically, in addition to reparameterizing time, it is the same as training a FM model with:
> (1) **Probability path:** Using a Gaussian probability path with independent coupling constructed via an SDE with affine drift coefficients (α_t, σ_t defined via (10.7)).
> (2) **Score parameterization:** Reparameterizing the marginal velocity field via the score function.

In table 1, we list how one can easily convert a score network to other velocity parameterizations. Therefore, different parameterizations are theoretically equivalent and one can even swap parameterizations post-training (see section 4.8.1). Note however, that the score and x_0 prediction parameterizations introduce a singularity at time $t = 0$ (near noise).

10.4 Sampling

Next, we discuss sampling from a diffusion model and how it relates to sampling from FM or GM model.

Deterministic sampling with ODE. If we consider the diffusion model a FM model, we would sample by sampling from the marginal vector field. In (4.78) we expressed the marginal vector field via the score function (for Gaussian paths):

$$u_t(x) = \frac{\dot{\alpha}_t}{\alpha_t} x - \left[\dot{\sigma}_t \sigma_t - \sigma_t^2 \frac{\dot{\alpha}_t}{\alpha_t}\right] \nabla \log p_t(x). \tag{10.11}$$

Using the specific form of equation (10.7) for α_t, σ_t, one derive the equivalent identity:

$$u_t(x) = \dot{k}(t) \left[a_t x - \frac{g_t^2}{2} \nabla \log p_t(x)\right]. \tag{10.12}$$

Alternatively, one insert the above $u_t(x)$ into the Continuity Equation to validate this directly. The corresponding ODE to u_t is also called the **Probability Flow ODE** (Song et al., 2021):

$$\mathrm{d}X_t = \dot{k}(t) \left[a_t X_t - \frac{g_t^2}{2} s_t^\theta(X_t)\right] \mathrm{d}t, \tag{10.13}$$

where we set $s_t^\theta(x) = \nabla \log p_t(x)$ as the learned score function. Note that we use here the notation for ODEs that is common for SDEs for a reason that becomes clear next. We also note that in equation (10.11) we add the term $\dot{k}(t)$ compared to (Song et al., 2021) because of the time reparameterization.

Stochastic sampling with SDE. In section 9.7, we have derived that we can add the Langevin dynamics

$$\frac{1}{2}\beta_t^2 \nabla \log p_t(x) dt + \beta_t dW_t \tag{10.14}$$

to any CTMP generative model and we will obtain a generative model following the same probability path. We can apply this to the Probability Flow ODE to get a whole family of SDEs that generate the probability path p_t:

$$dX_t = \left(\dot{k}(t)\alpha_t X_t + \frac{\beta_t^2 - \dot{k}(t) g_t^2}{2} \nabla \log p_t(X_t)\right) dt + \beta_t dW_t. \tag{10.15}$$

The above results in **stochastic sampling** of a diffusion model. In theory, all models above lead to the same marginals for every $\beta_t \geq 0$. This is a mathematical fact about the ground truth underlying stochastic process. In practice, we need to simulate the SDE

$$dX_t = \left(\dot{k}(t)\alpha_t X_t + \frac{\beta_t^2 - \dot{k}(t) g_t^2}{2} s_t^\theta(X_t)\right) dt + \beta_t dW_t \tag{10.16}$$

with a trained network s_t^θ. We have both estimation errors (*i.e.*, imperfect training of s_t^θ) as well as simulation errors (*i.e.*, imperfect sampling of the underlying SDE). Therefore, there is an optimal unknown noise level β_t (see *e.g.*, (Albergo and Vanden-Eijnden, 2022, equation (2.45))) that can be determined empirically (Karras et al., 2022) and theoretically (Ma et al., 2024).

Therefore, we get:

1. ODE sampling: For a Gaussian source, independent coupling, fixing α_t, σ_t according to (10.7), and score parameterization, sampling from a diffusion model with the Probability Flow ODE is the same as sampling from a FM model.
2. SDE sampling: Under the same conditions, sampling from a diffusion model with stochastic SDE sampling is equivalent to sampling from GM model defined via equation (10.15).

10.5 The role of time-reversal and the backward process

To finish our discussion of diffusion models, we discuss the role of time-reversal and the backward process in the context of diffusion models. Given how central the idea of time-reversal is for diffusion models, it might seem surprising to some readers that it is not needed for FM. We explain this, therefore, here in more detail. Specifically, to generate data, diffusion models frame training as learning a *backward process* $\bar{X}_r$ going from 0 to $R > 0$ such that

$$\bar{X}_r \overset{d}{=} X_{R-r} \text{ for all } r \in [0, R] \tag{10.17}$$

where $\overset{d}{=}$ denotes equality in distribution. Once we found such a process, we can initialize $\bar{X}_0 \overset{d}{=} X_R$ with a Gaussian and simulate it to obtain $\bar{X}_R \overset{d}{=} X_0 \sim q$, *i.e.*, a sample the data distribution q. If $X_r \sim \tilde{p}_r$ (*i.e.*, it is the marginal probability path as in equation (10.5)), then equation (10.17) is equivalent to:

$$\bar{X}_r \sim \bar{p}_r := \tilde{p}_{R-r} \text{ for all } r \in [0, R].$$

In other words, we want the stochastic process $\bar{X}_r$ to generate the probability path $\bar{p}_r$. But this is exactly what we are trying to do in Generator Matching for general Markov processes (see equation (8.10)), and in particular in Flow Matching with flows. Therefore, we get:

The following problems are equivalent:
(1) **Time-reverse marginals:** Finding an SDE (resp. ODE) for the backward process that has the same marginals as the forward process - as done for diffusion models.
(2) **Generate probability path:** Finding an SDE (resp. ODE) that generates the probability path defined by the forward process.
(3) **Solve KFE:** Finding an SDE (resp. ODE) that solves the Fokker-Planck Equation (resp. Continuity Equation).

The original description of diffusion models included the "full" time-reversal of an SDE (Anderson, 1982). This is a notion that is stronger than the one we use, *i.e.*, it requires that the joint distribution across time points are the same

$$\mathbb{P}[\bar{X}_{r_1} \in A_1, \dots, \bar{X}_{r_n} \in A_n] = \mathbb{P}[X_{R-r_1} \in A_1, \dots, X_{R-r_n} \in A_n] \tag{10.18}$$

$$\text{for all } 0 \leq r_1, \dots, r_n, \text{ and } A_1, \dots, A_n \subset S \text{ measurable.} \tag{10.19}$$

As shown in Anderson (1982), one can obtain a time-reversal satisfying the above condition with a backward process for a specific choice of β_t in equation (10.15) . However, for the purposes of generative modeling, we often only use the final point X_1 of the Markov process (*e.g.*, as a generated image) and discard earlier time points. Therefore, whether a Markov process is a "true" time-reversal or only has the same marginals (as in equation (10.17)) does not matter for many applications. A famous example is the probability flow ODE. The probability flow ODE does *not* constitute a time-reversal of a diffusion process in the sense of (Anderson, 1982) but it follows the same marginals. This illustrates that finding "true" time-reversals is a harder mathematical problem to solve but (often) not necessary for the purposes of generative modeling. The probability flow ODE is the current state-of-the-art for sampling with a low number of neural network evaluations (Karras et al., 2022). This shows that the "true" time-reversal might even give suboptimal results compared to other solutions of the Fokker-Planck equation (Ma et al., 2024).

10.6 Relation to Other Denoising Model

Inspired by the success of diffusion models, there were significant advances in translating the methods from diffusion models to other state spaces (Campbell et al., 2022, 2024; De Bortoli et al., 2022; Huang et al., 2022a; Benton et al., 2022). Similarly to diffusion models, they construct a forward Markov process that noises data and then time-reverse it (*i.e.*, "de-noises" it). We therefore informally refer to such models as "denoising models". Naturally, one can ask how these models relate to GM (see section 9). In brief, the relation of these "denoising models" to GM models is the same as the relation between diffusion models and FM:

Generally speaking, denoising models are Generator Matching models with
(1) **Time convention:** They use the diffusion time convention where time 0 corresponds to data.
(2) **Probability path construction:** Probability paths are constructed via a "forward" or "noising" process.
(3) **Solving the KFE:** A particular solution to the KFE is found via a time-reversal of the forward process.

We acknowledge that this an *informal* rule and that there might be exceptions to that rule. Therefore, we refer to an extended and detailed discussion of such related work (including a more complete list of references) to (Holderrieth et al., 2024).

References

Michael S Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

Michael Samuel Albergo, Mark Goldstein, Nicholas Matthew Boffi, Rajesh Ranganath, and Eric Vanden-Eijnden. Stochastic interpolants with data-dependent couplings. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

Luigi Ambrosio. Transport equation and cauchy problem forbvvector fields. *Inventiones mathematicae*, 158(2), 2004.

Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3): 313–326, 1982.

Heli Ben-Hamu, Samuel Cohen, Joey Bose, Brandon Amos, Maximillian Nickel, Aditya Grover, Ricky T. Q. Chen, and Yaron Lipman. Matching normalizing flows and probability paths on manifolds. *Proceedings of the 39th International Conference on Machine Learning*, 162, 2022.

Joe Benton, Yuyang Shi, Valentin De Bortoli, George Deligiannidis, and Arnaud Doucet. From denoising diffusions to denoising markov models. *arXiv preprint arXiv:2211.03595*, 2022.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: Vision-language-action flow model for general robot control. *arXiv*, 2024. https://www.physicalintelligence.company/download/pi0.pdf.

Vladimir I Bogachev, Nicolai V Krylov, Michael Röckner, and Stanislav V Shaposhnikov. *Fokker–Planck–Kolmogorov Equations*, volume 207. American Mathematical Society, 2022.

Avishek Joey Bose, Tara Akhound-Sadegh, Guillaume Huguet, Kilian Fatras, Jarrid Rector-Brooks, Cheng-Hao Liu, Andrei Cristian Nica, Maksym Korablyov, Michael Bronstein, and Alexander Tong. Se(3)-stochastic flow matching for protein backbone generation. *arXiv preprint arXiv:2310.02391*, 2023.

Arwen Bradley and Preetum Nakkiran. Classifier-free guidance is a predictor-corrector. *arXiv preprint arXiv:2408.09000*, 2024.

Andrew Campbell, Joe Benton, Valentin De Bortoli, Thomas Rainforth, George Deligiannidis, and Arnaud Doucet. A continuous time framework for discrete denoising models. *Advances in Neural Information Processing Systems*, 35: 28266–28279, 2022.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. *arXiv preprint arXiv:2402.04997*, 2024.

Ricky T. Q. Chen and Yaron Lipman. Flow matching on general geometries. In *The Twelfth International Conference on Learning Representations*, 2024.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Muthu Chidambaram, Khashayar Gatmiry, Sitan Chen, Holden Lee, and Jianfeng Lu. What does guidance do? a fine-grained analysis in a simple setting. *arXiv preprint arXiv:2409.13074*, 2024.

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

Philippe Courrege. Sur la forme intégro-différentielle des opérateurs de c_k^∞ dans c satisfaisant au principe du maximum. *Séminaire Brelot-Choquet-Deny. Théorie du Potentiel*, 10(1):1–38, 1965.

Mark HA Davis. Piecewise-deterministic markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 46(3):353–376, 1984.

Valentin De Bortoli, Emile Mathieu, Michael Hutchinson, James Thornton, Yee Whye Teh, and Arnaud Doucet. Riemannian score-based generative modelling. *Advances in Neural Information Processing Systems*, 35:2406–2422, 2022.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems*, volume 34, pages 8780–8794. Curran Associates, Inc., 2021.

Sander Dieleman. Guidance: a cheat code for diffusion models, 2022. https://benanne.github.io/2022/05/26/guidance.html.

Ronald J DiPerna and Pierre-Louis Lions. Ordinary differential equations, transport theory and sobolev spaces. *Inventiones mathematicae*, 98(3):511–547, 1989.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

Stewart N Ethier and Thomas G Kurtz. *Markov processes: characterization and convergence*. John Wiley & Sons, 2009.

William Feller. On second order differential operators. *Annals of Mathematics*, 61(1):90–105, 1955.

Alessio Figalli. Existence and uniqueness of martingale solutions for sdes with rough or degenerate coefficients. *Journal of Functional Analysis*, 254(1):109–153, 2008.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *arXiv preprint arXiv:2407.15595*, 2024.

Izrail Moiseevitch Gelfand, Richard A Silverman, et al. *Calculus of variations*. Courier Corporation, 2000.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Yingqing Guo, Hui Yuan, Yukang Yang, Minshuo Chen, and Mengdi Wang. Gradient guidance for diffusion models: An optimization perspective. *arXiv preprint arXiv:2404.14743*, 2024.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

Eric Heitz, Laurent Belcour, and Thomas Chambon. Iterative α-(de) blending: A minimalist deterministic diffusion model. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–8, 2023.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Peter Holderrieth, Marton Havasi, Jason Yim, Neta Shaul, Itai Gat, Tommi Jaakkola, Brian Karrer, Ricky Chen, and Yaron Lipman. Generator matching: Generative modeling with arbitrary markov processes. *Preprint*, 2024. http://arxiv.org/abs/2410.20587.

Chin-Wei Huang, Milad Aghajohari, Joey Bose, Prakash Panangaden, and Aaron C Courville. Riemannian diffusion models. *Advances in Neural Information Processing Systems*, 35:2750–2761, 2022a.

Chin-Wei Huang, Milad Aghajohari, Joey Bose, Prakash Panangaden, and Aaron C Courville. Riemannian diffusion models. In *Advances in Neural Information Processing Systems*, 2022b.

Guillaume Huguet, James Vuckovic, Kilian Fatras, Eric Thibodeau-Laufer, Pablo Lemos, Riashat Islam, Cheng-Hao Liu, Jarrid Rector-Brooks, Tara Akhound-Sadegh, Michael Bronstein, et al. Sequence-augmented se (3)-flow matching for conditional protein backbone generation. *arXiv preprint arXiv:2405.20313*, 2024.

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

Jürgen Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer, 2008.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in Neural Information Processing Systems*, 35:26565–26577, 2022.

Diederik Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Thomas G Kurtz. Equivalence of stochastic equations and martingale problems. *Stochastic analysis 2010*, pages 113–130, 2011.

Matthew Le, Apoorv Vyas, Bowen Shi, Brian Karrer, Leda Sari, Rashel Moritz, Mary Williamson, Vimal Manohar, Yossi Adi, Jay Mahadeokar, et al. Voicebox: Text-guided multilingual universal speech generation at scale. *Advances in neural information processing systems*, 36, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

Guan-Horng Liu, Arash Vahdat, De-An Huang, Evangelos A. Theodorou, Weili Nie, and Anima Anandkumar. I2sb: image-to-image schrödinger bridge. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Lynn Harold Loomis and Shlomo Sternberg. *Advanced calculus*. World Scientific, 1968.

Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser-Nam Lim, and Christopher De Sa. Neural manifold ordinary differential equations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

Aaron Lou, Minkai Xu, Adam Farris, and Stefano Ermon. Scaling riemannian diffusion models. *Advances in Neural Information Processing Systems*, 36:80291–80305, 2023.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. In *Advances in Neural Information Processing Systems*, 2020.

Paul C Matthews. *Vector calculus*. Springer Science & Business Media, 2012.

Robert J McCann. A convexity principle for interacting gases. *Advances in mathematics*, 128(1):153–179, 1997.

Kirill Neklyudov, Rob Brekelmans, Daniel Severo, and Alireza Makhzani. Action matching: Learning stochastic dynamics from samples. In *International conference on machine learning*, pages 25858–25889. PMLR, 2023.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171. PMLR, 18–24 Jul 2021.

Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 16784–16804. PMLR, 17–23 Jul 2022.

Bernt Øksendal. *Stochastic differential equations*. Springer, 2003.

Amnon Pazy. *Semigroups of linear operators and applications to partial differential equations*, volume 44. Springer Science & Business Media, 2012.

Stefano Peluchetti. Non-denoising forward-time diffusions. *arXiv preprint arXiv:2312.14589*, 2023.

Lawrence Perko. *Differential equations and dynamical systems*, volume 7. Springer Science & Business Media, 2013.

Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

Ashwini Pokle, Matthew J Muckley, Ricky T. Q. Chen, and Brian Karrer. Training-free linear image inversion via flows. *arXiv preprint arXiv:2310.04432*, 2023.

Adam Polyak, Amit Zohar, Andrew Brown, Andros Tjandra, Animesh Sinha, Ann Lee, Apoorv Vyas, Bowen Shi, Chih-Yao Ma, Ching-Yao Chuang, David Yan, Dhruv Choudhary, Dingkang Wang, Geet Sethi, Guan Pang, Haoyu Ma, Ishan Misra, Ji Hou, Jialiang Wang, Kiran Jagadeesh, Kunpeng Li, Luxin Zhang, Mannat Singh, Mary Williamson, Matt Le, Matthew Yu, Mitesh Kumar Singh, Peizhao Zhang, Peter Vajda, Quentin Duval, Rohit Girdhar, Roshan Sumbaly, Sai Saketh Rambhatla, Sam Tsai, Samaneh Azadi, Samyak Datta, Sanyuan Chen, Sean Bell, Sharadh Ramaswamy, Shelly Sheynin, Siddharth Bhattacharya, Simran Motwani, Tao Xu, Tianhe Li, Tingbo Hou, Wei-Ning Hsu, Xi Yin, Xiaoliang Dai, Yaniv Taigman, Yaqiao Luo, Yen-Cheng Liu, Yi-Chiao Wu, Yue Zhao, Yuval Kirstain, Zecheng He, Zijian He, Albert Pumarola, Ali Thabet, Artsiom Sanakoyeu, Arun Mallya, Baishan

Guo, Boris Araya, Breena Kerr, Carleigh Wood, Ce Liu, Cen Peng, Dimitry Vengertsev, Edgar Schonfeld, Elliot Blanchard, Felix Juefei-Xu, Fraylie Nord, Jeff Liang, John Hoffman, Jonas Kohler, Kaolin Fire, Karthik Sivakumar, Lawrence Chen, Licheng Yu, Luya Gao, Markos Georgopoulos, Rashel Moritz, Sara K. Sampson, Shikai Li, Simone Parmeggiani, Steve Fine, Tara Fowler, Vladan Petrovic, and Yuming Du. Movie gen: A cast of media foundation models, 2024. https://arxiv.org/abs/2410.13720.

Aram-Alexandre Pooladian, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. Multisample flow matching: Straightening flows with minibatch couplings. In *International Conference on Machine Learning*, 2023.

Jan Prüss, Mathias Wilke, and Mathias Wilke. *Ordinary differential equations and dynamic systems*. Springer, 2010.

Gareth O Roberts and Richard L Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli 2(4): 341-363 (December 1996)*, 1996.

Leonard CG Rogers and David Williams. *Diffusions, markov processes, and martingales: Volume 1, foundations*. Cambridge university press, 2000.

Noam Rozen, Aditya Grover, Maximilian Nickel, and Yaron Lipman. Moser flow: Divergence-based generative modeling on manifolds. *Advances in Neural Information Processing Systems*, 34:17669–17680, 2021.

Ludger Rüschendorf, Alexander Schnurr, and Viktor Wolf. Comparison of time-inhomogeneous markov processes. *Advances in Applied Probability*, 48(4):1015–1044, 2016.

Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 Conference Proceedings*, SIGGRAPH '22. Association for Computing Machinery, 2022.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

Simo Särkkä and Arno Solin. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

Neta Shaul, Ricky T. Q. Chen, Maximilian Nickel, Matthew Le, and Yaron Lipman. On kinetic optimal probability paths for generative models. In *International Conference on Machine Learning*, pages 30883–30907. PMLR, 2023a.

Neta Shaul, Juan Perez, Ricky T. Q. Chen, Ali Thabet, Albert Pumarola, and Yaron Lipman. Bespoke solvers for generative flow models. *arXiv preprint arXiv:2310.19075*, 2023b.

Neta Shaul, Itai Gat, Marton Havasi, Daniel Severo, Anuroop Sriram, Peter Holderrieth, Brian Karrer, Yaron Lipman, and Ricky T. Q. Chen. Flow matching with general discrete paths: A kinetic-optimal perspective, 2024. https://arxiv.org/abs/2412.03487.

Yuyang Shi, Valentin De Bortoli, Andrew Campbell, and Arnaud Doucet. Diffusion schrödinger bridge matching. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, pages 11895–11907, 2019.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

Alexander Tong, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Kilian Fatras, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *arXiv preprint arXiv:2302.00482*, 2023.

Cédric Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021.

Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7): 1661–1674, 2011.

Wilhelm von Waldenfels. Fast positive operatoren. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 4:159–174, 1965.

Apoorv Vyas, Bowen Shi, Matthew Le, Andros Tjandra, Yi-Chiao Wu, Baishan Guo, Jiemin Zhang, Xinyue Zhang, Robert Adkins, William Ngan, et al. Audiobox: Unified audio generation with natural language prompts. *arXiv preprint arXiv:2312.15821*, 2023.

Jason Yim, Andrew Campbell, Andrew YK Foong, Michael Gastegger, José Jiménez-Luna, Sarah Lewis, Victor Garcia Satorras, Bastiaan S Veeling, Regina Barzilay, Tommi Jaakkola, et al. Fast protein backbone generation with se (3) flow matching. *arXiv preprint arXiv:2310.05297*, 2023.

Qinqing Zheng, Matt Le, Neta Shaul, Yaron Lipman, Aditya Grover, and Ricky T. Q. Chen. Guided flows for generative modeling and decision making, 2023. https://arxiv.org/abs/2311.13443.

Appendix

A Additional proofs

A.1 Discrete Mass Conservation

Lemma 1 (PMF solutions to Kolmogoroc with rate conditions)**.** *Consider a solution $f_t(x)$ to Kolmogorov Equation (6.8) with initial condition $f_0(x) = p(x)$, where p is a PMF, $u_t(y, x)$ is $C([0, 1])$ in time t and satisfies the rate conditions (6.4). Then $f_t(x)$ is a probability mass function (PMF) for all $t \in [0, 1]$.*

Proof. Let $f_t(x)$, $t \in [0, 1]$, be the solution to the Kolmogorov Equation, the existence and uniqueness of which is guarateed by theorem 12. Now, $f_t(x)$ is a PMF if and only if it satisfies

$$f_t(x) \geq 0, \text{ and } \sum_x f_t(x) = 1. \tag{A.1}$$

The latter condition is shown to hold by summing both sides of the Kolmogorov Equation to get that the solution satisfies

$$\frac{\mathrm{d}}{\mathrm{d}t} \sum_x f_t(x) = \sum_x \sum_z u_t(x, z) p_t(z) = 0$$

where the second equality is due to $\sum_y u_t(y, x) = 0$ in the rate conditions. Since $\sum_x f_0(x) = \sum_x p(x) = 1$ we hae that $\sum_x f_t(x) \equiv 1$ for all $t \in [0, 1]$.

To prove that $f_t(x) \geq 0$ for all $x \in \mathcal{S}$ we will use a result on convex invariant sets of dynamical systems. In particular, Theorem 7.3.4 in Prüss et al. (2010) asserts that as long as $f_0 = p$ satisfies this condition (which it does) and whenever $w(z)$ is on the boundary of this constraint, *i.e.*, w is a PMF and $w(z) = 0$ for some $z \in \mathcal{S}$, then a non-positive inner-product with the outer-normal to the constraint, *i.e.*, $\sum_{x,y} u_t(y, x) w(x) \delta(y, z) \geq 0$ implies that the solution $f_t(x) \geq 0$ for all $t \in [0, 1]$ and $x \in \mathcal{S}$. Let us check this condition:

$$\sum_{x,y} u_t(y, x) w(x) \delta(y, z) = \sum_x u_t(z, x) w(x) = \sum_{x \neq z} u_t(z, x) w(x) \geq 0,$$

where in the second equality we use the fact that $w(z) = 0$ and in the last inequality we used the rate condition (6.4) that $u_t(z, x) \geq 0$ for $z \neq x$ and $w(y) \geq 0$ for all y.

□

Theorem *13* (Discrete Mass Conservation). Let $u_t(y, x)$ be in $C([0, 1))$ and $p_t(x)$ a PMF in $C^1([0, 1))$ in time t. Then, the following are equivalent:

1. p_t, u_t satisfy the Kolmogorov Equation (6.8) for $t \in [0, 1)$, and u_t satisfies the rate conditions (6.4).
2. u_t generates p_t in the sense of 6.5 for $t \in [0, 1)$.

Proof. Let us start by assuming 2. In this case, the probability transition kernel $p_{t+h|t}(y|x)$ satisfies (6.3),

$$p_{t+h|t}(y|x) = \delta(y, x) + h u_t(y, x) + o(h). \tag{A.2}$$

By expressing the marginal $p_t(y)$ using the Law of Total Probability, we obtain

$$p_{t+h}(y) = \sum_x p_{t+h|t}(y|x)\, p_t(x). \tag{A.3}$$

By plugging (A.2) into (A.3) and rearranging, we get

$$\frac{p_{t+h}(y) - p_t(y)}{h} = \sum_x u_t(y, x) p_t(x) + o(1),$$

where now $o(1) = o(h)/h \to 0$ as $h \to 0$, as per the definition of $o(h)$. Taking the limit $h \to 0$, we get that the pair (p_t, u_t) satisfies the Kolmogorov Equation (6.8). Next, let us prove that u_t satisfies the rate conditions (6.4). If $u_t(y, x) < 0$ for some $y \neq x$, it follows from (A.2) that $p_{t+h|t}(y|x) < 0$ for small $h > 0$, and this contradicts $p_{t+h|t}$ being a probability kernel. If $\sum_y u_t(y, x) = c \neq 0$, it follows from A.2 that $1 = \sum_x p_{t+h|t}(y|x) = 1 + hc + o(h)$, leading to a contradiction for small $h > 0$.

Conversely, assume now condition 1. That is, the pair (u_t, p_t) satisfies the Kolmogorov Equation (6.8) with initial condition $p_0 = p$. By theorem 12, let $p_{s|t}(y|x)$ be the unique solution to the Kolomogorov Equation

$$\frac{\mathrm{d}}{\mathrm{d}s} p_{s|t}(y|x) = \sum_z u_t(y, z) p_{s|t}(z|x), \tag{A.4}$$

with initial condition $p_{t|t}(y, x) = \delta(y, x)$, where $0 \leq t \leq s < 1$ and t and y are held constant. By lemma 1, $p_{s|t}(\cdot|x)$ is a PMF. The term $\sum_x p_{s|t}(y|x)p(x)$ also satisfies the Kolmogorov Equation, since

$$\frac{\mathrm{d}}{\mathrm{d}t} \sum_x p_{s|t}(y|x)p(x) = \sum_x \left[\sum_z u_t(y, z) p_{s|t}(z|x) \right] p(x) = \sum_z u_t(y, z) \left[\sum_x p_{s|t}(z|x)p(x) \right], \tag{A.5}$$

now with initial conditions $\sum_x p_{t|t}(y|x)p(x) = p(y)$. Due to the uniqueness of solutions to the Kolmogorov Equation (theorem 12), it follows that $\sum_x p_{s|t}(y|x)p(x) = p_s(y)$, as required. Lastly, the semi-group property of the transition kernel, $\sum_z p_{s|r}(y|z)p_{r|t}(z|x) = p_{s|t}(y|x)$ for $0 \leq t \leq r \leq s < 1$, can be shown by repeating the argument in A.5 with $p_{r|t}$ as initial condition at time r. In conclusion, we found a transition kernel $p_{t+h|t}$ that generates p_t, as required. □

A.2 Manifold Marginalization Trick

Theorem *10* (Manifold Marginalization Trick). Under Assumption 2, if $u_t(x|x_1)$ is conditionally integrable and generates the conditional probability path $p_t(\cdot|x_1)$ then the marginal velocity field $u_t(\cdot)$ generates the marginal probability path $p_t(\cdot)$.

Proof. To prove that $u_t(\cdot)$ generates $p_t(\cdot)$, we will again show they satisfy the conditions in the Mass Conservation Theorem. First, we check that $u_t(x)$ and $p_t(x)$ satisfy the Continuity Equation (5.1):

$$\frac{\mathrm{d}}{\mathrm{d}t} p_t(x) \overset{(i)}{=} \int_{\mathcal{M}} \frac{\mathrm{d}}{\mathrm{d}t} p_{t|1}(x|x_1) q(x_1) \mathrm{dvol}_{x_1} \tag{A.6}$$

$$\overset{(ii)}{=} -\int_{\mathcal{M}} \mathrm{div}_g \left[u_t(x|x_1) p_{t|1}(x|x_1) \right] q(x_1) \mathrm{dvol}_{x_1} \tag{A.7}$$

$$\overset{(i)}{=} -\mathrm{div}_g \int_{\mathcal{M}} u_t(x|x_1) p_{t|1}(x|x_1) q(x_1) \mathrm{dvol}_{x_1} \tag{A.8}$$

$$\overset{(iii)}{=} -\mathrm{div}_g \left[u_t(x) p_t(x) \right], \tag{A.9}$$

where in (i) we switched differentiation ($\frac{\mathrm{d}}{\mathrm{d}t}$ and div_g) and integration justified by Leibniz rule, and the fact that $p_{t|1}(x|x_1)$ and $u_t(x|x_1)$ are C^1 in t, x and q has bounded support or $\mathcal{M}$ is compact. In (ii), we used the fact that $u_t(\cdot|x_1)$ generates $p_{t|1}(\cdot|x_1)$ and theorem 9. In (iii), we multiplied and divided by $p_t(x)$ (which is strictly positive by assumption) and used the formula for u_t in (5.10). Lastly, u_t is integrable and locally Lipschitz by employing the same arguments as in the proof of theorem 3. □

A.3 Regularity assumptions for KFE

We note that assumption 5 is true under relatively weak assumptions and there is a diversity of mathematical literature on showing uniqueness of the solution of the KFE in p_t for different settings. However, to the best of our knowledge, there is no known result that states regularity assumptions for general state spaces and Markov processes, which is why simply state it here as an assumption. For the machine learning practitioner, this assumption holds for any state space of interest. To illustrate this, we point the rich sources in the mathematical literature that show uniqueness that list the regularity assumptions for specific spaces and classes of Markov processes:

1. Flows in $\mathbb{R}^d$ and manifolds: (Villani et al., 2009, Mass conservation formula, page 15), (DiPerna and Lions, 1989), (Ambrosio, 2004)
2. Diffusion in $\mathbb{R}^d$ and manifolds: (Villani et al., 2009, Diffusion theorem, page 16)
3. General Ito-SDEs in $\mathbb{R}^d$: (Figalli, 2008, Theorem 1.3 and 1.4), (Kurtz, 2011, Corollary 1.3), (Bogachev et al., 2022)
4. Discrete state spaces: Here, the KFE is a linear ODE, which has a unique solution under the assumption that the coefficients are continuous (see theorem 13).