

# RDFIA - Diffusion Models and Flow Matching (3-a)

Bourzag Mohamed Chakib, Missoum Youcef

January 2026

## 1 Introduction

This document contains the exercises and written solutions for the Diffusion Models and Flow Matching practical work (RDFIA TP3-a). We'll start with a brief summary of the subject before moving to the exercises:

## 2 Summary

### 2.1 Context and Objectives

In the landscape of generative modeling, **Flow Matching (FM)** and Diffusion Models have emerged as powerful alternatives to Generative Adversarial Networks (GANs). While GANs attempt to map noise to data in a single "long jump" (prone to instability and mode collapse), Flow Matching formulates the generation process as a continuous time evolution governed by an **Ordinary Differential Equation (ODE)**.

The objective of this practical work is to understand the geometry of these generative paths. Specifically, we investigate how the choice of the **interpolation schedule**—which defines how we mix noise and data over time—affects the learning dynamics and the geometric properties of the generated distribution.

### 2.2 Theoretical Framework: The Geometry of Flow

Flow Matching learns a time-dependent vector field  $v_t(x)$  that pushes probability mass from a simple prior distribution  $p_0$  (Gaussian noise) to the complex data distribution  $p_1$ . The core theoretical insight explored here is the distinction between two types of paths:

- **Linear Schedule (Optimal Transport):** Defines a straight path between noise and data ( $X_t = (1 - t)X_0 + tX_1$ ). While conceptually simple and minimizing transport cost, we demonstrated mathematically that it is **not variance preserving**. In high dimensions, this causes trajectories to pass through low-density regions near the origin (variance collapse), making learning inefficient.
- **Cosine Schedule (Variance Preserving):** Defines a curved path along the hypersphere ( $X_t = \cos(\frac{\pi}{2}t)X_0 + \sin(\frac{\pi}{2}t)X_1$ ). By satisfying  $\alpha_t^2 + \beta_t^2 = 1$ , this schedule maintains constant signal energy. Geometrically, this restricts the flow to the "habitable zone" of the latent space, avoiding the origin and simplifying the regression task for the neural network.

### 2.3 Practical Approach and Key Findings

The practical was divided into two main investigations:

**1. Dynamics on 2D Toy Data:** We implemented a Multi-Layer Perceptron (MLP) to learn the vector field for 2D Gaussian distributions. By visualizing the trajectories and monitoring the variance of the generated samples, we empirically confirmed our theoretical predictions: the linear schedule exhibited a variance dip at  $t = 0.5$ , whereas the cosine schedule maintained stable variance throughout the integration, validating its geometric superiority.

**2. Generative Modeling on MNIST:** We extended the approach to high-dimensional image generation. We contrasted Flow Matching with the GANs studied in previous sessions. Our analysis highlighted a fundamental trade-off:

- **GANs:** Offer rapid, single-step inference but suffer from training instability (minimax game) and mode collapse.
- **Flow Matching:** Provides stable, regression-based training and excellent mode coverage. Although inference is slower due to the iterative ODE solving (requiring multiple function evaluations), the decomposition of the complex generation task into many small, simple steps allows Flow Matching to model highly complex distributions even with limited network capacity.

### 3 Exercises

#### Exercise 1. Generative Adversarial Networks

1. A popular schedule of Diffusion Models/Flow Matching is the “cosine schedule”. This is of the form  $\alpha_t = \cos(at)$ ,  $\beta_t = \sin(bt)$ . Determine the smallest  $a$  and  $b$  such that the border conditions are respected.

**Solution.**

The boundary conditions for a generative path from noise  $X_0$  to data  $X_1$  are:

- At  $t = 0$  (Noise):  $X_0 = 1 \cdot X_0 + 0 \cdot X_1 \implies \alpha_0 = 1, \beta_0 = 0$ .
- At  $t = 1$  (Data):  $X_1 = 0 \cdot X_0 + 1 \cdot X_1 \implies \alpha_1 = 0, \beta_1 = 1$ .

Using the given forms:

- $\alpha_0 = \cos(0) = 1$  (OK).
- $\beta_0 = \sin(0) = 0$  (OK).
- $\alpha_1 = \cos(a) = 0 \implies a = \frac{\pi}{2}$  (smallest positive solution).
- $\beta_1 = \sin(b) = 1 \implies b = \frac{\pi}{2}$  (smallest positive solution).

Thus,  $\mathbf{a} = \frac{\pi}{2}$  and  $\mathbf{b} = \frac{\pi}{2}$ . □

2. Calculate  $dX_t/dt$  in the case of the cosine schedule.

**Solution.**

With  $a = b = \frac{\pi}{2}$ , the process is defined as:

$$X_t = \cos\left(\frac{\pi}{2}t\right) X_0 + \sin\left(\frac{\pi}{2}t\right) X_1$$

Differentiating with respect to  $t$ :

$$\begin{aligned} \frac{dX_t}{dt} &= \frac{d}{dt} \left[ \cos\left(\frac{\pi}{2}t\right) \right] X_0 + \frac{d}{dt} \left[ \sin\left(\frac{\pi}{2}t\right) \right] X_1 \\ \frac{dX_t}{dt} &= -\frac{\pi}{2} \sin\left(\frac{\pi}{2}t\right) X_0 + \frac{\pi}{2} \cos\left(\frac{\pi}{2}t\right) X_1 \\ \frac{dX_t}{dt} &= \frac{\pi}{2} \left( \cos\left(\frac{\pi}{2}t\right) X_1 - \sin\left(\frac{\pi}{2}t\right) X_0 \right) \end{aligned}$$

□

3. Consider the linear schedule between two Gaussian random variables:  $X_t = (1-t)X_0 + tX_1$ , with  $X_0 \sim \mathcal{N}(0, 1)$  and  $X_1 \sim \mathcal{N}(0, 1)$ . By looking at the case  $t = 1/2$ , show that the schedule is not variance preserving.

**Solution.**

Assuming independent  $X_0, X_1$ , the variance of the linear combination is:

$$\text{Var}(X_t) = (1-t)^2 \text{Var}(X_0) + t^2 \text{Var}(X_1)$$

Given  $\text{Var}(X_0) = \text{Var}(X_1) = 1$ :

$$\text{Var}(X_t) = (1-t)^2 + t^2$$

At  $t = 1/2$ :

$$\text{Var}(X_{0.5}) = (0.5)^2 + (0.5)^2 = 0.25 + 0.25 = \mathbf{0.5}$$

Since  $0.5 \neq 1$ , the variance is **not preserved** (it collapses in the middle of the trajectory). □

4. Show that the cosine schedule is variance preserving (using the same technique).

**Solution.**

Using the cosine schedule  $X_t = \cos(\frac{\pi}{2}t)X_0 + \sin(\frac{\pi}{2}t)X_1$ :

$$\text{Var}(X_t) = \cos^2\left(\frac{\pi}{2}t\right) \text{Var}(X_0) + \sin^2\left(\frac{\pi}{2}t\right) \text{Var}(X_1)$$

Assuming unit variance for inputs:

$$\text{Var}(X_t) = \cos^2\left(\frac{\pi}{2}t\right) + \sin^2\left(\frac{\pi}{2}t\right) = \mathbf{1}$$

Since the variance remains 1 for all  $t \in [0, 1]$ , the schedule is **variance preserving**. □

5. Why is it an advantage to be variance preserving (think about the geometry of the generation process)?

**Solution.**

In high-dimensional spaces, probability mass concentrates on a "spherical shell" at a specific radius (proportional to  $\sqrt{d}$ ) rather than near the origin.

- **Non-VP (Linear):** A linear interpolation acts as a chord cutting through the sphere. At  $t = 0.5$ , the norm (and variance) drops, creating samples in low-density regions that are out-of-distribution.
- **VP (Cosine/Arc):** This schedule traverses the surface of the hypersphere (a geodesic). This ensures that intermediate samples  $X_t$  maintain the correct signal magnitude/energy throughout the process, keeping the trajectory within the high-probability manifold. This simplifies learning as the network does not need to learn to "re-inflate" collapsed signals.

□

6. Do the experimental observations confirm your theoretical predictions concerning the variance preservation of the schedules? To be sure about this, you can calculate the variance of the generated data at each time step.

**Solution.**

Yes, the observations confirm the theory. This can be seen in the following trajectories:

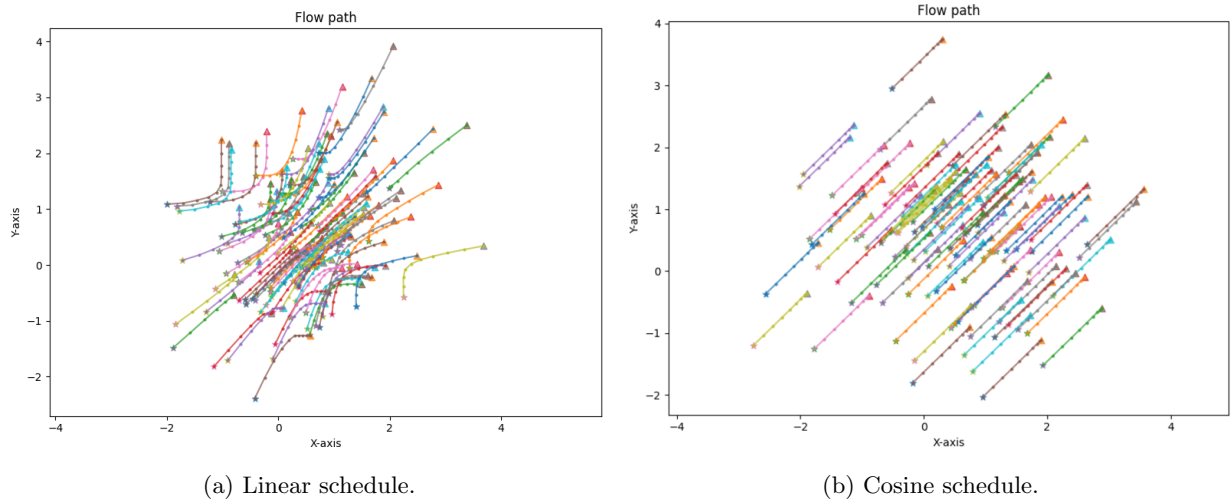


Figure 1: Flow matching on 2D Gaussian data with linear and cosine schedules.

- To verify this, we used the following snippet in the generation loop:

```
variances = traj.var(dim=1).mean(dim=1)

for t, var in enumerate(variances):
    if t % 10 == 0: # Print every 10 steps
        print(f"Time {t}: Variance = {var.item():.4f}")
```

- We observe the variance dropping to approximately 0.5 at the midpoint ( $t = 0.5$ ) when using **linear schedule**, confirming that linear paths traverse low-density regions (magnitude collapse) and are not variance preserving.

□

## Exercise 2. Images (MNIST)

- 7. What is the major advantage of Diffusion Models/Flow Matching over Generative Adversarial Networks (GANs) ?

**Solution.**

The major advantages are **training stability** and **mode coverage**.

Unlike GANs, which rely on an unstable adversarial minimax game (where the generator and discriminator must remain balanced to avoid vanishing gradients), Diffusion Models and Flow Matching use a stable regression objective (minimizing Mean Squared Error). Furthermore, they are likelihood-based (or approximate likelihood) models, which forces them to capture the entire data distribution, effectively solving the "Mode Collapse" problem common in GANs where the model ignores hard-to-generate samples.

□

- 8. What is the computational disadvantage of Diffusion Models/Flow Matching in comparison to GANs ?

**Solution.**

The main disadvantage is **slow inference speed** (sampling latency).

GANs generate an image in a single forward pass of the neural network ( $z \rightarrow x$ ). In contrast, Diffusion Models and Flow Matching require an **iterative process** (solving an ODE or iterative denoising) that involves calling the neural network many times (e.g., 10 to 1000 steps) to generate a single image. This makes real-time generation much more computationally expensive.  $\square$

- 9. GANs generate images in one “step”, ie with one neural network, whereas Flow Matching does this in several steps. In light of this, if we allow each method to have a fixed number of neural network parameters, why do you think the Flow Matching algorithm produces better results ?

**Solution.**

Flow Matching produces better results because it **decomposes a complex problem into many simple ones**. A GAN must learn the entire, highly non-linear mapping from noise to data in a single function call (approximating the integral directly), which requires massive capacity to be accurate. In contrast, Flow Matching only needs to learn the **local vector field** (the velocity or derivative) at a specific time  $t$ . This local function is much smoother and easier to learn. By applying this simpler function iteratively (integration), the model effectively constructs an incredibly complex transformation that a single-step network of the same size could never represent. The “computational depth” is effectively multiplied by the number of steps.  $\square$

- For the Image-to-image translation part, we achieved really interesting translation results for 200 epochs from one domain to another as shown in figure 2. Though it may need more epochs for better outcome, this capability is really interesting.

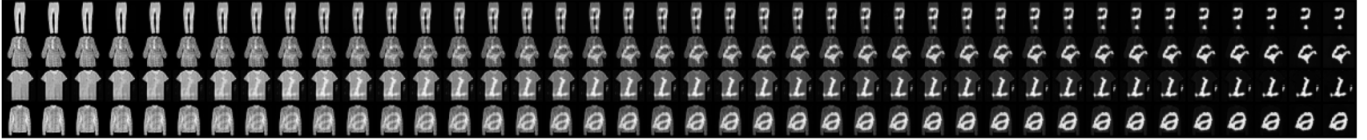


Figure 2: Image-to-image translation results at 200 epochs.

## 4 Conclusion

To conclude, this practical provided a deep dive into the mechanics of **Flow Matching**, illustrating why it has become a dominant paradigm in modern generative modeling.

Through our theoretical analysis and 2D experiments, we demonstrated that the **geometry of the interpolation path** is critical. We proved mathematically and verified empirically that while linear schedules (Optimal Transport) minimize distance, they fail to preserve variance, causing trajectories to collapse into low-density regions. In contrast, **Variance Preserving (VP)** schedules, like the cosine schedule, restrict the generative flow to the hypersphere surface, maintaining signal energy and simplifying the learning task for the neural network.

Furthermore, by comparing Flow Matching to the GANs studied in previous sessions, we identified a fundamental trade-off in generative deep learning:

- **GANs** offer extremely fast, single-step inference but are hindered by unstable adversarial training dynamics and mode collapse.
- **Flow Matching/Diffusion** offers robust, stable training (via simple regression objectives) and excellent mode coverage. The cost for this stability is inference latency, as the model must iteratively solve an ODE.

Ultimately, this work highlights that the superior performance of recent models stems from decomposing the complex generation task into many small, manageable steps—effectively trading computational time at inference for learning stability and generation quality. It also showed an interesting translation capability from one domain to another.