CS 6501: Constrained-Aware Generative AI

Lecture Notes 8
Flow Matching and Rectified Flows

Learning vector fields for generative transport, and why ODE form is ideal for constraint injection

Prof. Ferdinando Fioretto
Department of Computer Science, University of Virginia

Thursday, February 5, 2026

Abstract

Diffusion models (Lecture 7) can be read as controlled stochastic dynamics whose reverse-time updates admit natural insertion points for guidance and feasibility restoration. Flow-based alternatives sharpen this view by switching from an SDE-style sampler to an ODE: generation becomes the numerical integration of a learned time-dependent vector field. This lecture introduces *flow matching* as a direct method to learn such vector fields without likelihood training, and *rectified flows* as a simplified transport construction that straightens trajectories to enable fast sampling.

The conceptual point for the course is algorithmic. When sampling is an ODE, constraint injection looks like adding a control term to the drift, and the standard optimization operators from Lecture 6 (projection, proximal maps, and splitting) become immediate building blocks for constrained sampling. We formalize flow matching objectives, derive practical training losses from path interpolants, discuss rectified flow training and "reflow" iterations, connect the framework to Neural ODEs, and end with concrete constraint-injection templates motivated by later splitting methods.

1 From diffusion updates to ODE transport

Lecture 7 emphasized that diffusion sampling is a long sequence of local updates. In continuous time, the score-based view of diffusion yields an SDE whose drift includes the score (and hence admits guidance as an additive drift term). Flow-based methods retain the "many small updates" interface, but replace stochasticity by a deterministic ODE

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t), \qquad t \in [0, 1], \tag{1}$$

where the learned vector field $\mathbf{v}_\theta$ transports a simple base distribution (often Gaussian noise) into the data distribution.

Diffusion-to-ODE intuition. A useful way to connect Lectures 7 and 8 is to view both samplers as defining a *time-indexed family of distributions* $(p_t)_{t\in[0,1]}$ and a rule for moving particles so that their empirical distribution tracks p_t. In diffusion, particles follow an SDE that combines a drift (often expressed using the score) and injected noise; in flow matching, particles follow an ODE with no injected noise. In particular, if one runs many particles in parallel and looks only at their evolving empirical density, both approaches can be interpreted as solving a transport PDE, with the primary difference being whether the microscopic dynamics include diffusion.

Algorithmic interface. Equation (1) should be read as "a simulator with a knob". The knob is the vector field $\mathbf{v}_\theta(\cdot, t)$, and numerical integration turns continuous time into a long sequence of discrete updates, analogous to reverse diffusion steps. This analogy is important for constrained-aware generation because it explains why constraint mechanisms from optimization (projection, proximal maps, penalties) can be inserted at the same granularity as the integrator steps.

Concrete toy example (2D transport). Consider a simple setting where p_{base} is $\mathcal{N}(0, I_2)$ in $\mathbb{R}^2$ and p_{data} is a ring-shaped distribution (points concentrated near $\|\mathbf{x}\|_2 \approx 1$). An unconstrained learned $\mathbf{v}_\theta$ can be interpreted as "pushing mass outward" while also redistributing angularly. If we further impose the hard constraint $\|\mathbf{x}\|_2 \leq 1$ (a closed unit disk), then projecting after each step will keep trajectories inside the disk; if instead we impose the soft penalty $\phi(\mathbf{x}) = (\|\mathbf{x}\|_2 - 1)_+^2$, adding $-\eta\nabla\phi$ produces a continuous force that discourages leaving the disk without a hard cut.

The ODE form is useful for constrained-aware generation because the insertion points are explicit. If a constraint is expressed as a penalty potential $\phi(\mathbf{x})$, the most direct injection is to modify the drift by adding a control force $-\eta\,\nabla\phi(\mathbf{x})$ (a continuous-time analogue of penalty methods). If feasibility is expressed as a set $\mathcal{C}$, then projection or proximal restoration can be applied after each numerical integration step (a splitting analogue). In other words, ODE sampling makes constrained generation look like *control plus numerical integration*, rather than "one-shot" sampling.

Control perspective (preview). It is helpful to interpret the constrained sampler as

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t) + \mathbf{u}_t(\mathbf{x}_t), \tag{2}$$

where the "control" $\mathbf{u}_t$ encodes constraint correction. Penalty guidance corresponds to $\mathbf{u}_t(\mathbf{x}) = -\eta(t)\nabla\phi(\mathbf{x})$, while projection or proximal restoration corresponds to an *impulsive* or *split* correction applied at discrete times. This will align directly with later splitting methods, where one alternates between (i) integrating the learned flow and (ii) applying an optimization operator.

2 Neural ODE preliminaries and the continuity equation

Neural ODEs (Chen et al., 2018) treat $\mathbf{v}_\theta(\cdot, t)$ as a neural network and integrate (1) to define a mapping from an initial condition $\mathbf{x}_0$ to a terminal state $\mathbf{x}_1$. For generative modeling, we interpret the initial state as drawn from a base distribution p_0 (noise) and the terminal state as a sample from a learned distribution p_1 (data). The induced time-marginal densities $(p_t)_{t\in[0,1]}$ satisfy the continuity equation

$$\partial_t p_t(\mathbf{x}) + \nabla \cdot \big(p_t(\mathbf{x})\,\mathbf{v}_\theta(\mathbf{x}, t)\big) = 0. \tag{3}$$

Equation (3) says that learning $\mathbf{v}_\theta$ is equivalent to learning how probability mass flows. Continuous normalizing flows exploit this by training via exact likelihood using an ODE for the log-density; flow matching instead learns $\mathbf{v}_\theta$ by local regression against target velocities, avoiding likelihood computation.

Mass conservation and geometric meaning. Equation (3) is the statement that probability mass is neither created nor destroyed under the dynamics (1). A helpful interpretation is: the divergence term $\nabla \cdot (p_t\mathbf{v}_\theta)$ accounts for inflow and outflow of probability through an infinitesimal volume around $\mathbf{x}$. If $\nabla \cdot \mathbf{v}_\theta$ is negative in a region, trajectories locally contract volume (mass concentrates); if it is positive, trajectories locally expand volume (mass spreads). This is why a learned vector field can "morph" a Gaussian into a complex distribution: it can bend trajectories (redistribute mass) and also compress or expand volume.

Connection to log-density dynamics (context). Although we will not rely on likelihood training here, it is useful to remember the standard identity used in continuous normalizing flows: along a trajectory $t \mapsto \mathbf{x}_t$, the log-density evolves as

$$\frac{d}{dt} \log p_t(\mathbf{x}_t) = -\nabla \cdot \mathbf{v}_\theta(\mathbf{x}_t, t), \tag{4}$$

so computing likelihood reduces to tracking the divergence of the vector field. Flow matching avoids this computation by not requiring p_t or its divergence during training.

Example (why divergence matters). Suppose p_{base} is isotropic Gaussian in $\mathbb{R}^2$ and the data lie near a 1D curve (for instance, a noisy spiral). To map a full-dimensional Gaussian to a distribution concentrated near a curve, the flow must contract volume dramatically toward that curve. This is precisely encoded by negative divergence in regions that funnel trajectories inward. For constrained-aware generation, this observation matters because a hard projection step can also introduce abrupt volume contraction, and stability depends on how this interacts with the learned drift.

For the purposes of this course, (1) is the key: any discretization yields a sequence of small steps, and every step can be augmented with the optimization primitives of Lecture 6.

Numerical note (stiffness). When $\mathbf{v}_\theta(\mathbf{x}, t)$ varies rapidly in $\mathbf{x}$ or t, the ODE may become stiff, making large step sizes unstable. This interacts with constraints: strong projection or strong penalty forces can effectively add high-curvature components to the dynamics. One practical takeaway is that "how hard we enforce constraints" and "how coarse we discretize" cannot be chosen independently.

3 Flow matching as learning a vector field

We follow Lipman et al. (2022).

3.1 Path construction via an interpolant

Let p_{data} denote the data distribution and p_{base} a simple base distribution (e.g., $\mathcal{N}(0, \mathbf{I})$). Choose a coupling between a data sample $\mathbf{x}_1 \sim p_{\text{data}}$ and a base sample $\mathbf{x}_0 \sim p_{\text{base}}$. Given a smooth scalar schedule $(\alpha(t), \sigma(t))$ with $\alpha(0) = 1$, $\alpha(1) = 0$, $\sigma(0) = 0$, and $\sigma(1) = 1$, define a stochastic interpolant

$$\mathbf{x}_t \;=\; \alpha(t)\,\mathbf{x}_1 + \sigma(t)\,\mathbf{x}_0, \qquad t \in [0, 1]. \tag{5}$$

This induces a family of time-marginals p_t and a conditional distribution over endpoints given $\mathbf{x}_t$. Differentiating (5) yields a pathwise velocity

$$\frac{d\mathbf{x}_t}{dt} \;=\; \dot{\alpha}(t)\,\mathbf{x}_1 + \dot{\sigma}(t)\,\mathbf{x}_0. \tag{6}$$

However, the sampler only has access to $(\mathbf{x}_t, t)$, not the latent endpoints $(\mathbf{x}_0, \mathbf{x}_1)$. The "correct" deterministic velocity field compatible with the interpolant is the conditional expectation

$$\mathbf{u}^\star(\mathbf{x}, t) \;\triangleq\; \mathbb{E}\big[\dot{\alpha}(t)\,\mathbf{x}_1 + \dot{\sigma}(t)\,\mathbf{x}_0 \;\big|\; \mathbf{x}_t = \mathbf{x}\big]. \tag{7}$$

This vector field transports p_{base} to p_{data} when used in (1) with appropriate boundary conditions.

Why the coupling matters. The definition of $\mathbf{x}_t$ in (5) uses a *paired* draw $(\mathbf{x}_1, \mathbf{x}_0)$. If $\mathbf{x}_0$ is sampled independently of $\mathbf{x}_1$, the coupling is the product coupling, and the resulting intermediate marginals p_t can be very broad. If instead the coupling aligns $\mathbf{x}_0$ to $\mathbf{x}_1$ in some informed way (for example by using a learned encoder, a nearest-neighbor match, or an iterative rectification procedure), then the resulting paths can be shorter and less curved. This is one of the main practical levers for making sampling fast.

Example (image-like data, conceptual). Think of $\mathbf{x}_1$ as an image and $\mathbf{x}_0$ as Gaussian noise. With $\alpha(t)$ decreasing and $\sigma(t)$ increasing, (5) looks like "progressively destroying the image and replacing it with noise" as t moves from 0 to 1. Flow matching learns the reverse-time velocity field that deterministically transports noise back into the data manifold. If we impose a constraint such as "preserve a known region of pixels" (an inpainting mask), then the interpolant can be modified to keep those pixels fixed, or the learned dynamics can be corrected after each step by projecting onto the set of images consistent with the mask.

3.2 The flow matching objective

Flow matching learns $\mathbf{v}_\theta$ by regression to the conditional velocity.

Flow matching objective (conceptual form)

$$\min_\theta \; \mathbb{E}_{t\sim\mathrm{Unif}[0,1]} \, \mathbb{E}_{\mathbf{x}\sim p_t}\Big[\|\mathbf{v}_\theta(\mathbf{x},t) - \mathbf{u}^\star(\mathbf{x},t)\|_2^2 \Big]. \tag{8}$$

In practice, we do not compute (7) exactly; instead, we use a tractable surrogate that is unbiased for the conditional expectation. A standard training recipe samples $(\mathbf{x}_1, \mathbf{x}_0, t)$, constructs $\mathbf{x}_t$ using (5), and then uses the pathwise velocity (6) as a target. Because $\mathbb{E}[\dot\alpha\,\mathbf{x}_1 + \dot\sigma\,\mathbf{x}_0 \mid \mathbf{x}_t] = \mathbf{u}^\star(\mathbf{x}_t, t)$, this yields a valid objective:

$$\min_\theta \; \mathbb{E}\Big[\|\mathbf{v}_\theta(\mathbf{x}_t,t) - (\dot\alpha(t)\,\mathbf{x}_1 + \dot\sigma(t)\,\mathbf{x}_0)\|_2^2 \Big]. \tag{9}$$

What is being regressed, operationally? At training time, the target $(\dot\alpha(t)\mathbf{x}_1 + \dot\sigma(t)\mathbf{x}_0)$ is a *velocity* attached to the point $\mathbf{x}_t$. Thus the model is trained as a time-conditioned vector regressor: given a partially-interpolated sample $\mathbf{x}_t$ and the scalar t, output the direction in $\mathbb{R}^d$ that moves $\mathbf{x}_t$ along the transport path. This framing makes it clear why the method is compatible with later constraint injection: any time you can evaluate a constraint signal at $\mathbf{x}_t$, you can add a correction direction to this velocity.

Example (choice of schedule). If $\alpha(t) = 1 - t$ and $\sigma(t) = t$, then (5) becomes linear interpolation and (6) becomes constant in t given endpoints. If instead $\sigma(t)$ grows very slowly near $t = 0$ and quickly near $t = 1$, then the early portion of the path stays closer to data and the late portion transitions rapidly to noise. This affects both learning difficulty (where the regression targets have higher variance) and sampling stability (where the vector field changes more abruptly in time).

Remark 1. *The freedom to choose the interpolant* (5) *is a major modeling knob. Different schedules change the geometry of the transport, numerical stiffness of* (1)*, and the effectiveness of later constraint injection. Rectified flows can be read as a specific choice aimed at simplifying the geometry.*

Constraint-aware interpolants (preview). In some applications, we may bake constraints directly into the interpolant. For instance, if $\mathcal{C}$ encodes a linear equality $A\mathbf{x} = b$, one can define an interpolant that always stays in the affine subspace by replacing $\mathbf{x}_0$ and $\mathbf{x}_1$ with their projections onto $\{\mathbf{x} : A\mathbf{x} = b\}$, or by adding an explicit correction term that cancels constraint-violating components along the path. This idea parallels "architecture as feasibility bias" from earlier lectures, now expressed as "path design as feasibility bias".

3.3 Sampling: integrating the learned ODE

Once trained, generation is performed by drawing $\mathbf{x}_0 \sim p_{\text{base}}$ and integrating (1) from $t = 0$ to $t = 1$. A simple Euler discretization with step size h yields

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + h\,\mathbf{v}_\theta(\mathbf{x}_k, t_k), \qquad t_k = kh,\ k = 0, \dots, \lfloor 1/h \rfloor - 1. \tag{10}$$

More accurate integrators (Runge–Kutta, adaptive solvers) are possible, but for constrained-aware generation Euler is often preferable because it exposes explicit "after-step" hooks for projection and repair.

Example (Euler with a repair operator). If $R(\cdot)$ is a discrete repair routine (possibly non-differentiable, as in Lecture 4 constrained decoding), then Euler sampling with repair becomes

$$\tilde{\mathbf{x}}_{k+1} \leftarrow \mathbf{x}_k + h\,\mathbf{v}_\theta(\mathbf{x}_k, t_k), \qquad \mathbf{x}_{k+1} \leftarrow R(\tilde{\mathbf{x}}_{k+1}), \tag{11}$$

which is a direct ODE analogue of inserting symbolic correction into a generative procedure. This is precisely the "natural insertion point" that motivates later splitting methods.

4 Rectified flows: simplified transport and fast sampling

Rectified flow methods (Liu et al., 2022) aim to learn a flow whose trajectories are close to straight lines, making coarse discretizations effective.

4.1 Straight-line interpolants and constant pathwise velocity

Consider the simplest interpolant between an endpoint pair $(\mathbf{x}_0, \mathbf{x}_1)$:

$$\mathbf{x}_t = (1-t)\,\mathbf{x}_0 + t\,\mathbf{x}_1. \tag{12}$$

The pathwise velocity is constant:

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{x}_1 - \mathbf{x}_0. \tag{13}$$

If we could condition on endpoints, transporting along (12) would be trivial. But as in flow matching, the sampler conditions only on $(\mathbf{x}_t, t)$, so the correct drift is the conditional expectation $\mathbb{E}[\mathbf{x}_1 - \mathbf{x}_0 \mid \mathbf{x}_t]$, which is generally nontrivial.

Geometric interpretation. Rectified flows attempt to select (or learn) couplings and dynamics so that the effective transport resembles linear interpolation in the ambient space. When this succeeds, fewer integration steps are needed because the vector field varies slowly along the path. This also tends to reduce the accumulation of discretization error, which is a primary limiter of fast ODE sampling.

Rectified flow training uses the flow matching recipe with the straight-line interpolant. The practical loss is therefore

$$\min_\theta\ \mathbb{E}\Big[\,\|\mathbf{v}_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1 - \mathbf{x}_0)\|_2^2\,\Big], \tag{14}$$

where $(\mathbf{x}_0, \mathbf{x}_1)$ are sampled from a chosen coupling between base and data.

Example (why conditioning is hard). Even though (13) is constant given endpoints, the conditional expectation $\mathbb{E}[\mathbf{x}_1 - \mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$ can vary significantly with $\mathbf{x}$. Intuitively, the same intermediate point $\mathbf{x}_t = \mathbf{x}$ may be compatible with many different endpoint pairs. The model $\mathbf{v}_\theta(\mathbf{x}, t)$ learns a *single best average direction* that, when integrated across time, yields the correct marginal transport.

4.2 Why "rectification" matters

The goal is not only to fit (14), but also to make the learned conditional velocity field as aligned as possible across time so that numerical integration with few steps introduces limited error. Liu et al. (2022) propose iterative "reflow" procedures that update the coupling by sampling with the current model, then re-train, progressively straightening paths.

What reflow is doing, conceptually. If the current model generates samples that are already "closer" to data than pure noise, then coupling those generated samples with real data points can produce shorter, less curved transports for the next training round. Thus reflow gradually replaces an initially poor coupling (noise paired with data independently) with a coupling aligned to the learned geometry of the model. You can think of this as repeatedly refining the training distribution of paths so that the regression target becomes easier and the induced ODE becomes simpler to integrate.

For constrained-aware generation, straighter paths mean fewer steps and less opportunity for constraints to drift. This is a double-edged sword: fewer steps reduces compute and accumulation of numerical error, but also reduces the number of correction opportunities. Later lectures on splitting methods will formalize when fewer, larger steps are stable under projection or proximal correction.

Example 1 (Fast sampling and constraint opportunities)**.** *Suppose $\mathcal{C}$ encodes a simple hard feasibility set (e.g., a box constraint or a convex cone). Under a many-step sampler, projecting after every step can keep the trajectory near $\mathcal{C}$. Under a very coarse rectified sampler, projection may introduce large discontinuities that the remaining steps cannot repair. This is one reason why splitting (model drift vs constraint correction) and step-size schedules matter.*

Example (discrete structure via continuous embedding). Suppose the underlying object is discrete (a sequence or a graph) but we model it in a continuous embedding space, as in earlier lectures when using relaxed representations. Rectified flows may allow very few steps in embedding space, but the final rounding or decoding step can reintroduce constraint violations (syntax, valency, graph connectivity). This is where a "projection-like" operator can mean a symbolic repair routine, not a Euclidean projection. The same splitting logic applies, but the correction operator is now combinatorial.

5 Constraint injection in ODE form: control and splitting

We now make explicit the algorithmic point stated in the lecture title: ODE sampling turns constrained generation into controlled dynamics and motivates later splitting methods.

5.1 Soft constraints as control forces

Recall the constrained target template from Lecture 1,

$$\pi(\mathbf{x}) \propto p_{\text{data}}(\mathbf{x})\, \exp(-\lambda\, \phi(\mathbf{x})), \tag{15}$$

where ϕ is a differentiable penalty. In ODE sampling, the most direct insertion is

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t, t) - \eta(t)\, \nabla\phi(\mathbf{x}_t). \tag{16}$$

Equation (16) is the continuous-time analogue of "guidance" from diffusion (Lecture 7), but now the control term is simply added to the drift. When ϕ is not differentiable, $\nabla\phi$ can be replaced by a subgradient or, more robustly, by a proximal correction in a split scheme (below).

Example (quadratic penalties recover familiar dynamics). If $\phi(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - b\|_2^2$, then $-\nabla\phi(\mathbf{x}) = -A^\top(A\mathbf{x} - b)$ is a linear restoring force toward feasibility. Inserting this force into (16) yields a drift that combines the learned generative transport with a least-squares feasibility correction, directly mirroring the penalty-method viewpoint from Lecture 6. Time-dependent weights $\eta(t)$ can emphasize constraint satisfaction later in the trajectory (when samples are closer to the data manifold) or earlier (to prevent leaving feasible regions).

Example (classifier-style guidance analogue). If $\phi(\mathbf{x})$ is a differentiable surrogate for "constraint violation" computed by a predictor (for example, a property predictor in molecules or a differentiable validity proxy in sequences), then $-\nabla\phi$ is exactly the direction that most rapidly improves the predicted property under a first-order approximation. Thus (16) can be read as *property guidance* in an ODE sampler. A practical issue is calibration: if the predictor gradients are unreliable far from the data manifold, one often schedules $\eta(t)$ to be small at early times and larger later.

5.2 Hard constraints via projection and proximal restoration

Let $\mathcal{C} \subseteq \mathbb{R}^d$ be a feasibility set. A simple constrained sampler applies a projection after each numerical step:

$$\tilde{\mathbf{x}}_{k+1} \leftarrow \mathbf{x}_k + h\,\mathbf{v}_\theta(\mathbf{x}_k, t_k), \qquad \mathbf{x}_{k+1} \leftarrow \mathrm{Proj}_{\mathcal{C}}(\tilde{\mathbf{x}}_{k+1}). \tag{17}$$

If feasibility is soft or nonsmooth (e.g., sparsity, total variation, or piecewise-linear penalties), replace projection by a proximal operator:

$$\mathbf{x}_{k+1} \leftarrow \mathrm{prox}_{h\lambda\phi}(\tilde{\mathbf{x}}_{k+1}). \tag{18}$$

This mirrors the penalty and proximal methods from Lecture 6, with the difference that the "descent" direction is provided by the learned transport field, not by the gradient of a known objective.

Example (simplex or box constraints). If $\mathcal{C}$ is a box $\{\mathbf{x} : \ell \leq \mathbf{x} \leq u\}$, then $\mathrm{Proj}_{\mathcal{C}}$ is coordinate-wise clipping, and (17) becomes extremely cheap. If $\mathcal{C}$ is a probability simplex (as in discrete relaxations), then $\mathrm{Proj}_{\mathcal{C}}$ is the Euclidean simplex projection from Lecture 6, and (17) becomes "integrate in logit space, then re-normalize by projection". These two examples show why flows are attractive for constrained generation: the correction step can be selected to match the constraint geometry.

Example (prox for sparsity). If $\phi(\mathbf{x}) = \|\mathbf{x}\|_1$, then $\mathrm{prox}_{h\lambda\phi}$ is soft-thresholding, and (18) enforces sparse structure after each drift step. This is a canonical instance where the correction is non-smooth but computationally simple, and it illustrates why proximal restoration is often more robust than subgradients in practice.

5.3 Splitting view and why it will matter later

Write a constrained sampler abstractly as a composition of two operators per step: a model-driven drift update and a constraint correction. When the correction is projection or a proximal map, this is a first-order splitting method. A Strang-type symmetric splitting would apply half a correction, then a drift step, then another half correction, improving stability when the constraint correction is strong.

Why splitting is the right abstraction. From the numerical analysis viewpoint, (16) mixes two vector fields (generative drift and constraint force) and then discretizes. Splitting instead discretizes each component separately and composes the resulting maps. This distinction matters when the correction operator is not naturally expressed as a smooth vector field (projection, repair, proximal steps): in those cases, splitting is the only faithful way to represent the correction.

We defer the formal treatment to later lectures, but the high-level message is: *flow samplers are numerical integrators, and constraint injection is operator splitting.* This connection will be central when we discuss stability, step size, and tradeoffs between fast sampling and robust feasibility.

6 Riemannian flow matching and structured domains

Many scientific domains live on manifolds or quotient spaces (periodic crystals, rotations, shape spaces). Euclidean vector fields can introduce artifacts when the state has geometric constraints. Miller et al. (2024) extend flow matching to Riemannian settings, learning vector fields on manifolds using tools such as the exponential map and logarithm map.

Example (rotations). If the state contains orientations, representing them in $\mathbb{R}^3$ as unconstrained vectors and transporting with an Euclidean ODE can drift off the manifold of valid rotations. A Riemannian approach instead learns tangent vectors on SO(3) (or a related representation) and updates via retraction or the exponential map so that every step stays on-manifold. This is analogous in spirit to hard constraints: the manifold structure is a feasibility set built into the state space.

Example (periodic crystals). For crystalline materials, lattice parameters can have periodic or quotient structure, and naive Euclidean interpolation between lattice representations can pass through physically meaningless intermediate states. Riemannian flow matching treats these variables in a geometry-aware way, which can reduce artifacts and improve sample validity even before any additional constraints are injected.

At a high level, the modification is conceptual: the ODE (1) is replaced by a manifold-valued ODE, and the regression targets in (9) are interpreted as tangent vectors. Algorithmically, this is aligned with the course theme: geometric structure acts as an architectural feasibility bias, and explicit constraints can then be injected as additional tangent-space control terms or as projection back to the manifold after a numerical step.

Constraint injection on manifolds (preview). In a manifold setting, "projection" may mean projecting an ambient-space update back onto the manifold (via a retraction), while "gradient penalty" may mean using the Riemannian gradient of ϕ. The same operator-splitting viewpoint applies, but all corrections are interpreted in the tangent space.

7 Summary and looking ahead

Flow matching learns a time-dependent vector field by local regression against target velocities induced by a chosen interpolant, and sampling is performed by integrating an ODE. Rectified flows specialize this construction toward straight trajectories, enabling fast sampling with coarse discretization.

Practical course takeaway. From the constrained-generation perspective, flow methods are attractive not because they are "more deterministic" than diffusion, but because they expose a clean numerical interface: every generation run is an integration loop. That loop is exactly where we can attach constraint checks, penalties, repairs, and splitting operators, and it is where we can reason about stability and compute budgets.

For constrained-aware generation, the ODE form is the key interface. Soft constraints can be injected as additive control forces in the drift, and hard constraints can be enforced by projection or proximal restoration after each integrator step. This naturally motivates operator splitting viewpoints, which will be formalized in later lectures when we study stability and the design of constraint correction schedules.

Example (preview of later splitting design). If constraint corrections are expensive (e.g., running a simulator or solving a combinatorial repair), one may apply them only every m steps, or use a coarse-to-fine schedule where corrections become more frequent as $t \to 1$. These design choices will be treated as splitting schedules, and later lectures will connect them to stability tradeoffs and feasibility guarantees.

References

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.

X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

R. Miller, S. Madireddy, A. Ziabari, P. Balachandran, and F. Fioretto. FlowMM: Generating materials with Riemannian flow matching. In *International Conference on Machine Learning (ICML)*, 2024.