AdjointDEIS: Efficient Gradients for Diffusion Models

On the use of the continuous adjoint equations with diffusion models

Zander W. Blasingame

Clarkson University
Potsdam, NY, USA

2025.01.15

Outline

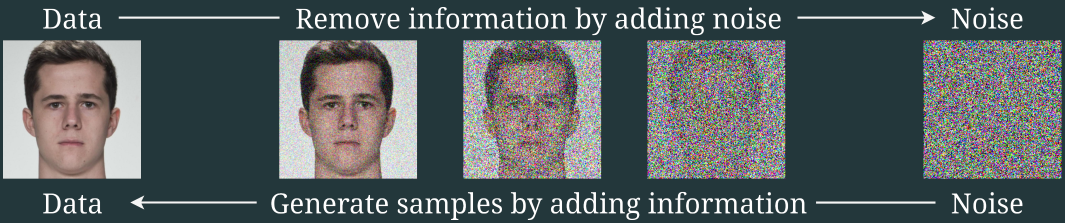
Introduction

AdjointDEIS

Remarks about using the continuous adjoint equations with diffusion models

Introduction

Diffusion models



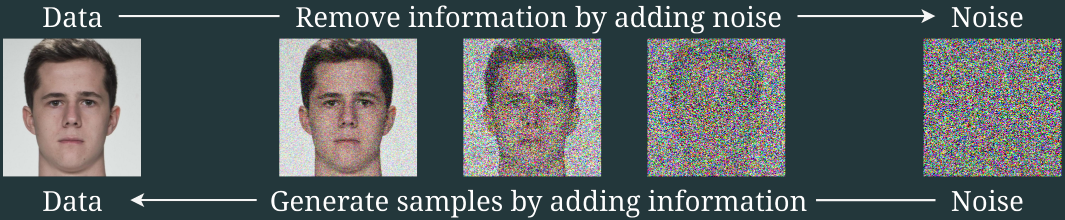
- Forward diffusion process is governed by the Itô SDE

$$d\mathbf{x}_t = f(t)\mathbf{x}_t \, dt + g(t) \, d\mathbf{w}_t, \tag{1}$$

where $\{\mathbf{w}_t\}_{t \in [0,T]}$ is the standard Wiener process on $[0, T]$.

[1] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

Diffusion models



- The diffusion equation can be reversed with

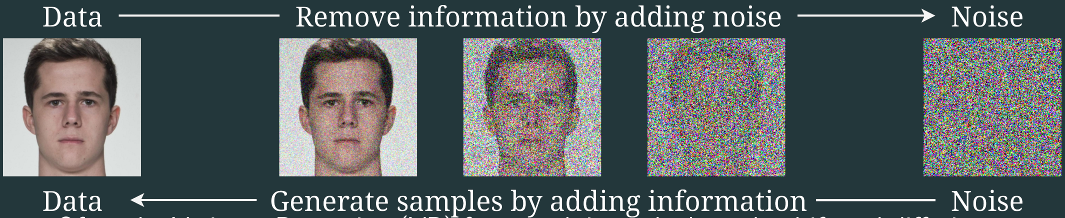
$$d\mathbf{x}_t = [f(t)\mathbf{x}_t - g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)]\, dt + g(t)\, d\bar{\mathbf{w}}_t, \tag{2}$$

  where $\bar{\mathbf{w}}_t$ is the *reverse* Wiener process and 'dt' is a *negative* timestep.
- The marginal distributions $p_t(\mathbf{x})$ follow the *probability flow* ODE[1]

$$\frac{d\mathbf{x}_t}{dt} = f(t)\mathbf{x}_t - \frac{1}{2} g^2(t)\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t). \tag{3}$$

[1] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

Diffusion models



- Often the Variance Preserving (VP) framework is used where the drift and diffusion coefficients are

$$f(t) = \frac{\mathrm{d}\log \alpha_t}{\mathrm{d}t}, \qquad g^2(t) = \frac{\mathrm{d}\sigma_t^2}{\mathrm{d}t} - 2\frac{\mathrm{d}\log \alpha_t}{\mathrm{d}t}\sigma_t^2, \tag{4}$$

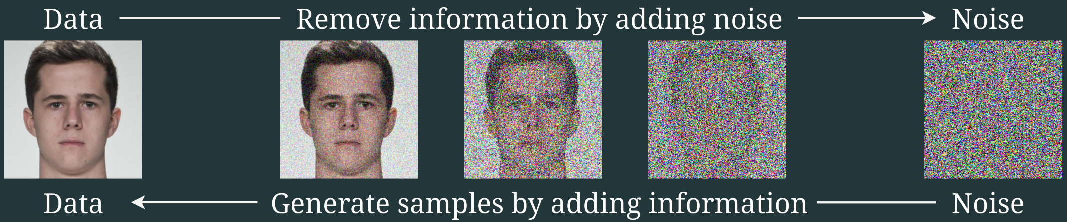
  for some noise schedule α_t, σ_t

- Sampling the forward trajectory then simplifies to

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \boldsymbol{\epsilon}_t \qquad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5}$$

[1] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

Diffusion models



- Train the model via score-matching to learn $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t)$.
- This is similar to learning the noise ϵ, *i.e.*,

$$\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \approx -\sigma_t \nabla_{\mathbf{x}} \log p_t(\mathbf{x}_t). \tag{6}$$

[1] Yang Song et al. “Score-Based Generative Modeling through Stochastic Differential Equations”. In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

Problem statement

- Solve the following optimization problem:

$$\arg\min_{\mathbf{x}_T,\mathbf{z},\theta} \mathcal{L}\left(\mathbf{x}_T + \int_T^0 f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,\mathbf{z},t)\;\mathrm{d}t\right). \tag{7}$$

- Or in the SDE case:

$$\arg\min_{\mathbf{x}_T,\mathbf{z},\theta} \mathcal{L}\left(\mathbf{x}_T + \int_T^0 f(t)\mathbf{x}_t + \frac{g^2(t)}{\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,\mathbf{z},t)\;\mathrm{d}t + \int_T^0 g(t)\;\mathrm{d}\bar{\mathbf{w}}_t\right). \tag{8}$$

- To backpropagate through an ODE/SDE solve we solve the continuous adjoint equations.

Continuous adjoint equations

- Let $\boldsymbol{f}_\theta$ describe a parameterized neural field of the probability flow ODE, defined as

$$\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t) = f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t). \tag{9}$$

Continuous adjoint equations

- Let $\boldsymbol{f}_\theta$ describe a parameterized neural field of the probability flow ODE, defined as

$$\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t) = f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t). \tag{9}$$

- Then $\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)$ describes a neural ODE which admits an adjoint state, $\mathbf{a}_\mathbf{x} := \partial\mathcal{L}/\partial\mathbf{x}_t$ (and likewise for $\mathbf{a}_\mathbf{z}(t)$ and $\mathbf{a}_\theta(t)$), which solve the continuous adjoint equations [7, Theorem 5.2] in the form of the following Initial Value Problem (IVP):

$$\begin{aligned}
\mathbf{a}_\mathbf{x}(0) &= \frac{\partial\mathcal{L}}{\partial\mathbf{x}_0}, & \frac{\mathrm{d}\mathbf{a}_\mathbf{x}}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top\frac{\partial\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial\mathbf{x}_t},\\
\mathbf{a}_\mathbf{z}(0) &= \mathbf{0}, & \frac{\mathrm{d}\mathbf{a}_\mathbf{z}}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top\frac{\partial\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial\mathbf{z}},\\
\mathbf{a}_\theta(0) &= \mathbf{0}, & \frac{\mathrm{d}\mathbf{a}_\theta}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top\frac{\partial\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial\theta}.
\end{aligned} \tag{10}$$

Continuous adjoint equations

- Let $\boldsymbol{f}_\theta$ describe a parameterized neural field of the probability flow ODE, defined as

$$\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t) = \underbrace{f(t)\mathbf{x}_t + \frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t).}_{\text{Black box model } \boldsymbol{f}_\theta(\mathbf{x}_t,\mathbf{z},t) \text{ loses known information of } f(t) \text{ and } g(t).} \tag{9}$$

- Then $\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)$ describes a neural ODE which admits an adjoint state, $\mathbf{a}_\mathbf{x} := \partial \mathcal{L} / \partial \mathbf{x}_t$ (and likewise for $\mathbf{a}_\mathbf{z}(t)$ and $\mathbf{a}_\theta(t)$), which solve the continuous adjoint equations [7, Theorem 5.2] in the form of the following Initial Value Problem (IVP):

$$\begin{aligned}
\mathbf{a}_\mathbf{x}(0) &= \frac{\partial \mathcal{L}}{\partial \mathbf{x}_0}, & \frac{\mathrm{d}\mathbf{a}_\mathbf{x}}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \mathbf{x}_t},\\
\mathbf{a}_\mathbf{z}(0) &= \mathbf{0}, & \frac{\mathrm{d}\mathbf{a}_\mathbf{z}}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \mathbf{z}},\\
\mathbf{a}_\theta(0) &= \mathbf{0}, & \frac{\mathrm{d}\mathbf{a}_\theta}{\mathrm{d}t}(t) &= -\mathbf{a}_\mathbf{x}(t)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \theta}.
\end{aligned} \tag{10}$$

AdjointDEIS

The continuous adjoint equations are also semi-linear

- Like diffusion ODEs the adjoint diffusion ODE is also semi-linear

$$\frac{\mathrm{d}\mathbf{a}_{\mathbf{x}}}{\mathrm{d}t}(t) = -\underbrace{f(t)\mathbf{a}_{\mathbf{x}}(t)}_{\text{Linear}} - \frac{g^2(t)}{2\sigma_t}\mathbf{a}_{\mathbf{x}}(t)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \mathbf{x}_t}. \tag{11}$$

[2] Cheng Lu et al. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 5775–5787. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/260a14acce2a89dad36adc8eefe7c59e-Paper-Conference.pdf.

The continuous adjoint equations are also semi-linear

- Like diffusion ODEs the adjoint diffusion ODE is also semi-linear

$$\frac{\mathrm{d}\mathbf{a}_{\mathbf{x}}}{\mathrm{d}t}(t) = -f(t)\mathbf{a}_{\mathbf{x}}(t) - \frac{g^2(t)}{2\sigma_t}\mathbf{a}_{\mathbf{x}}(t)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \mathbf{x}_t}. \tag{11}$$

- Then, the exact solution at time s given time $t < s$ is found to be

$$\mathbf{a}_{\mathbf{x}}(s) = \underbrace{e^{\int_s^t f(\tau)\,\mathrm{d}\tau}\mathbf{a}_{\mathbf{x}}(t)}_{\text{linear}} - \underbrace{\int_t^s e^{\int_s^u f(\tau)\,\mathrm{d}\tau}\frac{g^2(u)}{2\sigma_u}\mathbf{a}_{\mathbf{x}}(u)^\top \frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_u, \mathbf{z}, u)}{\partial \mathbf{x}_u}\,\mathrm{d}u\,.}_{\text{non-linear}} \tag{12}$$

[2] Cheng Lu et al. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 5775–5787. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/260a14acce2a89dad36adc8eefe7c59e-Paper-Conference.pdf.

The continuous adjoint equations are also semi-linear

- Like diffusion ODEs the adjoint diffusion ODE is also semi-linear

$$\frac{\mathrm{d}\mathbf{a_x}}{\mathrm{d}t}(t) = -f(t)\mathbf{a_x}(t) - \frac{g^2(t)}{2\sigma_t}\mathbf{a_x}(t)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)}{\partial \mathbf{x}_t}. \tag{11}$$

- Then, the exact solution at time s given time $t < s$ is found to be

$$\mathbf{a_x}(s) = \underbrace{e^{\int_s^t f(\tau)\,\mathrm{d}\tau}\mathbf{a_x}(t)}_{\text{linear}} - \underbrace{\int_t^s e^{\int_s^u f(\tau)\,\mathrm{d}\tau} \frac{g^2(u)}{2\sigma_u}\mathbf{a_x}(u)^\top \frac{\boldsymbol{\epsilon}_\theta(\mathbf{x}_u, \mathbf{z}, u)}{\partial \mathbf{x}_u}\,\mathrm{d}u}_{\text{non-linear}}. \tag{12}$$

- Use the log-SNR trick[2] to further simplify the exact solution with $\lambda_t \coloneqq \log(\alpha_t/\sigma_t)$.

[2] Cheng Lu et al. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps". In: *Advances in Neural Information Processing Systems.* Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 5775–5787. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/260a14acce2a89dad36adc8eefe7c59e-Paper-Conference.pdf.

Simplified exact solutions

Proposition 1 (Exact solution of adjoint diffusion ODEs[3])

Given initial values $[\mathbf{a}_{\mathbf{x}}(t), \mathbf{a}_{\mathbf{z}}(t), \mathbf{a}_{\theta}(t)]$ *at time* $t \in (0, T)$*, the solution* $[\mathbf{a}_{\mathbf{x}}(s), \mathbf{a}_{\mathbf{z}}(s), \mathbf{a}_{\theta}(s)]$ *at time* $s \in (t, T]$ *of adjoint diffusion ODEs in Eq.* (10) *is*

$$\mathbf{a}_{\mathbf{x}}(s) = \frac{\alpha_t}{\alpha_s}\mathbf{a}_{\mathbf{x}}(t) + \frac{1}{\alpha_s}\int_{\lambda_t}^{\lambda_s} \alpha_\lambda^2 e^{-\lambda}\mathbf{a}_{\mathbf{x}}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{x}_\lambda}\,\mathrm{d}\lambda, \tag{13}$$

$$\mathbf{a}_{\mathbf{z}}(s) = \mathbf{a}_{\mathbf{z}}(t) + \int_{\lambda_t}^{\lambda_s} \alpha_\lambda e^{-\lambda}\mathbf{a}_{\mathbf{x}}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{z}}\,\mathrm{d}\lambda, \tag{14}$$

$$\mathbf{a}_{\theta}(s) = \mathbf{a}_{\theta}(t) + \int_{\lambda_t}^{\lambda_s} \alpha_\lambda e^{-\lambda}\mathbf{a}_{\mathbf{x}}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \theta}\,\mathrm{d}\lambda. \tag{15}$$

[3] Zander W. Blasingame and Chen Liu. "AdjointDEIS: Efficient Gradients for Diffusion Models". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=fAlcxvrOEX.

Designing bespoke ODE solvers

- We denote the n-th derivative of the *scaled* vector-Jacobian product by

$$\mathbf{V}^{(n)}(\mathbf{x};\lambda_t) = \frac{\mathrm{d}^n}{\mathrm{d}\lambda^n}\left[\alpha_\lambda^2 \mathbf{a}_\mathbf{x}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{x}_\lambda}\right]_{\lambda=\lambda_t}. \tag{16}$$

- Use Taylor Expansion on Eq. (13) to obtain and letting $h = \lambda_s - \lambda_t$ yields

$$\mathbf{a}_\mathbf{x}(s) = \underbrace{\frac{\alpha_t}{\alpha_s}\mathbf{a}_\mathbf{x}(t)}_{\substack{\text{Linear term} \\ \textbf{Exactly computed}}} + \frac{1}{\alpha_s}\sum_{n=0}^{k-1} \mathbf{V}^{(n)}(\mathbf{x};\lambda_t) \int_{\lambda_t}^{\lambda_s} \frac{(\lambda-\lambda_t)^n}{n!} e^{-\lambda}\,\mathrm{d}\lambda + \mathcal{O}(h^{k+1}). \tag{17}$$

Designing bespoke ODE solvers

- We denote the n-th derivative of the *scaled* vector-Jacobian product by

$$\mathbf{V}^{(n)}(\mathbf{x};\lambda_t) = \frac{\mathrm{d}^n}{\mathrm{d}\lambda^n}\left[\alpha_\lambda^2 \mathbf{a}_{\mathbf{x}}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{x}_\lambda}\right]_{\lambda=\lambda_t}. \tag{16}$$

- Use Taylor Expansion on Eq. (13) to obtain and letting $h = \lambda_s - \lambda_t$ yields

$$\mathbf{a}_{\mathbf{x}}(s) = \underbrace{\frac{\alpha_t}{\alpha_s}\mathbf{a}_{\mathbf{x}}(t)}_{\substack{\text{Linear term}\\ \textbf{Exactly computed}}} + \frac{1}{\alpha_s}\sum_{n=0}^{k-1} \underbrace{\mathbf{V}^{(n)}(\mathbf{x};\lambda_t)}_{\substack{\text{Derivatives}\\ \textbf{Approximated}}} \int_{\lambda_t}^{\lambda_s} \frac{(\lambda-\lambda_t)^n}{n!} e^{-\lambda}\,\mathrm{d}\lambda + \mathcal{O}(h^{k+1}). \tag{17}$$

Designing bespoke ODE solvers

- We denote the n-th derivative of the *scaled* vector-Jacobian product by

$$\mathbf{V}^{(n)}(\mathbf{x};\lambda_t) = \frac{\mathrm{d}^n}{\mathrm{d}\lambda^n}\left[\alpha_\lambda^2 \mathbf{a}_{\mathbf{x}}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{x}_\lambda}\right]_{\lambda=\lambda_t}. \tag{16}$$

- Use Taylor Expansion on Eq. (13) to obtain and letting $h = \lambda_s - \lambda_t$ yields

$$\mathbf{a}_{\mathbf{x}}(s) = \underbrace{\frac{\alpha_t}{\alpha_s}\mathbf{a}_{\mathbf{x}}(t)}_{\substack{\text{Linear term} \\ \textbf{Exactly computed}}} + \frac{1}{\alpha_s}\sum_{n=0}^{k-1} \underbrace{\mathbf{V}^{(n)}(\mathbf{x};\lambda_t)}_{\substack{\text{Derivatives} \\ \textbf{Approximated}}} \underbrace{\int_{\lambda_t}^{\lambda_s} \frac{(\lambda-\lambda_t)^n}{n!} e^{-\lambda}\,\mathrm{d}\lambda}_{\substack{\text{Coefficients} \\ \textbf{Analytically computed}}} + \mathcal{O}(h^{k+1}). \tag{17}$$

Designing bespoke ODE solvers

- We denote the n-th derivative of the *scaled* vector-Jacobian product by

$$\mathbf{V}^{(n)}(\mathbf{x};\lambda_t) = \frac{\mathrm{d}^n}{\mathrm{d}\lambda^n}\left[\alpha_\lambda^2 \mathbf{a}_\mathbf{x}(\lambda)^\top \frac{\partial \boldsymbol{\epsilon}_\theta(\mathbf{x}_\lambda, \mathbf{z}, \lambda)}{\partial \mathbf{x}_\lambda}\right]_{\lambda=\lambda_t}. \tag{16}$$

- Use Taylor Expansion on Eq. (13) to obtain and letting $h = \lambda_s - \lambda_t$ yields

$$\mathbf{a}_\mathbf{x}(s) = \underbrace{\frac{\alpha_t}{\alpha_s}\mathbf{a}_\mathbf{x}(t)}_{\substack{\text{Linear term}\\ \textbf{Exactly computed}}} + \frac{1}{\alpha_s}\sum_{n=0}^{k-1} \underbrace{\mathbf{V}^{(n)}(\mathbf{x};\lambda_t)}_{\substack{\text{Derivatives}\\ \textbf{Approximated}}} \underbrace{\int_{\lambda_t}^{\lambda_s} \frac{(\lambda-\lambda_t)^n}{n!} e^{-\lambda}\,\mathrm{d}\lambda}_{\substack{\text{Coefficients}\\ \textbf{Analytically computed}}} + \underbrace{\mathcal{O}(h^{k+1})}_{\substack{\text{Higher-order errors}\\ \textbf{Omitted}}}. \tag{17}$$

- And analogously for $\mathbf{a}_\mathbf{z}(t)$ and $\mathbf{a}_\theta(t)$.

AdjointDEIS

Theorem 1 (AdjointDEIS-k as a k-th order solver)

Assume the function $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)$ and its associated vector-Jacobian products follow the regularity conditions detailed in Appendix B of the main paper, then for $k = 1, 2$, AdjointDEIS-k is a k-th order solver for adjoint diffusion ODEs, i.e., for the sequence $\{\tilde{\mathbf{a}}_\mathbf{x}(t_i)\}_{i=1}^M$ computed by AdjointDEIS-k, the global truncation error at time T satisfies $\tilde{\mathbf{a}}_\mathbf{x}(t_M) - \mathbf{a}_\mathbf{x}(T) = \mathcal{O}(h_{max}^2)$, where $h_{max} = \max_{1 \leq j \leq M}(\lambda_{t_i} - \lambda_{t_{i-1}})$. Likewise, AdjointDEIS-$k$ is a k-th order solver for the estimated gradients w.r.t. $\mathbf{z}$ and θ.

Certain adjoint SDEs are actually ODEs

Theorem 2

Let $\boldsymbol{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ *be in* $\mathcal{C}_b^{\infty,1}$ *and* $\boldsymbol{g} : \mathbb{R} \to \mathbb{R}^{d\times w}$ *be in* $\mathcal{C}_b^1$*. Let* $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ *be a scalar-valued differentiable function. Let* $\mathbf{w}_t : [0,T] \to \mathbb{R}^w$ *be a* w*-dimensional Wiener process. Let* $\mathbf{x} : [0,T] \to \mathbb{R}^d$ *solve the Stratonovich SDE*

$$\mathrm{d}\mathbf{x}_t = \boldsymbol{f}(\mathbf{x}_t, t)\,\mathrm{d}t + \boldsymbol{g}(t) \circ \mathrm{d}\mathbf{w}_t,$$

with initial condition $\mathbf{x}_0$*. Then the adjoint process* $\mathbf{a}_{\mathbf{x}}(t) \coloneqq \partial\mathcal{L}(\mathbf{x}_T)/\partial\mathbf{x}_t$ *is a strong solution to the backwards-in-time ODE*

$$\mathrm{d}\mathbf{a}_{\mathbf{x}}(t) = -\mathbf{a}_{\mathbf{x}}(t)^\top \frac{\partial \boldsymbol{f}}{\partial \mathbf{x}_t}(\mathbf{x}_t, t)\,\mathrm{d}t. \tag{18}$$

ODE solvers for the adjoint diffusion SDE

- Probability Flow ODEs are related to diffusion SDEs by the manipulations of the Kolmogorov equations[4].
- The drift term is identical to the vector field of the ODE, sans a factor of two:

$$\underbrace{\mathrm{d}\mathbf{x}_t = f(t)\mathbf{x}_t + 2\frac{g^2(t)}{2\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)\;\mathrm{d}t}_{\text{Probabilty Flow ODE}} + g(t)\;\mathrm{d}\bar{\mathbf{w}}_t. \tag{19}$$

[4] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

ODE solvers for the adjoint diffusion SDE

- Probability Flow ODEs are related to diffusion SDEs by the manipulations of the Kolmogorov equations[4].
- The drift term is identical to the vector field of the ODE, sans a factor of two:

$$\mathrm{d}\mathbf{x}_t = \underbrace{f(t)\mathbf{x}_t + \frac{g^2(t)}{\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)}_{=\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)}\,\mathrm{d}t + g(t)\,\mathrm{d}\bar{\mathbf{w}}_t. \tag{19}$$

- By Theorem 2 the adjoint SDE evolves with an ODE with vector field $-\mathbf{a}_\mathbf{x}(t)^\top \partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)/\partial \mathbf{x}_t$.

[4] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

ODE solvers for the adjoint diffusion SDE

- Probability Flow ODEs are related to diffusion SDEs by the manipulations of the Kolmogorov equations[4].
- The drift term is identical to the vector field of the ODE, sans a factor of two:

$$\mathrm{d}\mathbf{x}_t = \underbrace{f(t)\mathbf{x}_t + \frac{g^2(t)}{\sigma_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \mathbf{z}, t)}_{=\boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)} \;\mathrm{d}t + g(t)\;\mathrm{d}\bar{\mathbf{w}}_t. \tag{19}$$

- By Theorem 2 the adjoint SDE evolves with an ODE with vector field $-\mathbf{a}_\mathbf{x}(t)^\top \partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}, t)/\partial \mathbf{x}_t$.
- Therefore, we can use the *same* bespoke ODE solvers for adjoint diffusion ODEs with the added factor of 2!

[4] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=PxTIG12RRHS.

Experiment - face morphing

(a) Identity a **(b)** Face morphing with AdjointDEIS **(c)** Identity b

Figure 1: Create a morphed face which causes a Face Recognition (FR) system to accept it with **both** identities.

Experiment - face morphing

- Goal is to adversarially attack an FR system by finding the $\mathbf{x}_T, \mathbf{z}$ which creates the optimal morph.
- Optimality is defined with respect to an identity loss which is simply the average distance between the embeddings in the FR space.
- Using AdjointDEIS massively improves the performance of Diffusion Morphs (DiM).

Table 1: Vulnerability of different FR systems across different morphing attacks on the SYN-MAD 2022 dataset. FMR $= 0.1\%$.

		MMPMR [17](↑)		
Morphing Attack	**NFE(↓)**	**AdaFace** [10]	**ArcFace** [5]	**ElasticFace** [4]
Webmorph [6]	-	97.96	96.93	98.36
MIPGAN-I [19]	-	72.19	77.51	66.46
MIPGAN-II [19]	-	70.55	72.19	65.24
DiM-A [3]	350	92.23	90.18	93.05
Fast-DiM [2]	300	92.02	90.18	93.05
Morph-PIPE [20]	2350	95.91	92.84	95.5
DiM + AdjointDEIS-1 (ODE)	2250	**99.8**	**98.77**	**99.39**
DiM + AdjointDEIS-1 (SDE)	2250	98.57	97.96	97.75

Continuous adjoint equations for scheduled conditional information

- Currently, we consider constant conditional information $\mathbf{z}$.

Continuous adjoint equations for scheduled conditional information

- Currently, we consider constant conditional information $\mathbf{z}$.
- What if the conditioning signal changed with the timestep, *i.e.*, $\mathbf{z}_t$?

Continuous adjoint equations for scheduled conditional information

- Currently, we consider constant conditional information $\mathbf{z}$.
- What if the conditioning signal changed with the timestep, *i.e.*, $\mathbf{z}_t$?
- Question is then how do we find $\partial \mathcal{L} / \partial \mathbf{z}_t := \mathbf{a_z}(t)$?

Continuous adjoint equations for scheduled conditional information

- Currently, we consider constant conditional information $\mathbf{z}$.
- What if the conditioning signal changed with the timestep, *i.e.*, $\mathbf{z}_t$?
- Question is then how do we find $\partial\mathcal{L}/\partial\mathbf{z}_t \coloneqq \mathbf{a}_\mathbf{z}(t)$?
- Fortunately, it reduces to a simple integral.

Gradients of scheduled conditional information

Theorem 3

Suppose there exists a function $\mathbf{z} : [0, T] \to \mathbb{R}^z$ *which can be defined as a càdlàg piecewise function where* $\mathbf{z}$ *is continuous on each partition of* $[0, T]$ *given by* $\Pi = \{0 = t_0 < t_1 < \cdots < t_n = T\}$ *and whose right derivatives exist for all* $t \in [0, T]$*. Let* $\boldsymbol{f}_\theta : \mathbb{R}^d \times \mathbb{R}^z \times \mathbb{R} \to \mathbb{R}^d$ *be continuous in* t*, uniformly Lipschitz in* $\mathbf{x}$*, and continuously differentiable in* $\mathbf{x}$*. Let* $\mathbf{x} : \mathbb{R} \to \mathbb{R}^d$ *be the unique solution for the ODE*

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} = \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}_t, t),$$

with initial condition $\mathbf{x}_0$*. Let* $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ *be a scalar-valued loss function defined on the output of the neural ODE. Then* $\partial\mathcal{L}/\partial\mathbf{z}_t := \mathbf{a}_\mathbf{z}(t)$ *and there exists a unique solution* $\mathbf{a}_\mathbf{z} : \mathbb{R} \to \mathbb{R}^z$ *to the following IVP:*

$$\mathbf{a}_\mathbf{z}(T) = \mathbf{0}, \qquad \frac{\mathrm{d}\mathbf{a}_\mathbf{z}}{\mathrm{d}t}(t) = -\mathbf{a}_\mathbf{x}(t)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_t, \mathbf{z}_t, t)}{\partial \mathbf{z}_t}.$$

Gradients of the conditional information are still a mere integral

- As the vector fields of the ODE are independent of $\mathbf{a}_{\mathbf{z}}$ we have a mere integral,

$$\mathbf{a}_{\mathbf{z}}(t) = -\int_T^t \mathbf{a}_{\mathbf{x}}(\tau)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_\tau, \mathbf{z}_\tau, \tau)}{\partial \mathbf{z}_\tau} \, \mathrm{d}\tau. \tag{20}$$

Gradients of the conditional information are still a mere integral

- As the vector fields of the ODE are independent of $\mathbf{a}_{\mathbf{z}}$ we have a mere integral,

$$\mathbf{a}_{\mathbf{z}}(t) = -\int_T^t \mathbf{a}_{\mathbf{x}}(\tau)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_\tau, \mathbf{z}_\tau, \tau)}{\partial \mathbf{z}_\tau} \, \mathrm{d}\tau. \tag{20}$$

- We can simply replace our $\mathbf{z}$ with $\mathbf{z}_t$ when performing guided generation.
- Enables us to update time-dependent conditioning signal.

Gradients of the conditional information are still a mere integral

- As the vector fields of the ODE are independent of $\mathbf{a_z}$ we have a mere integral,

$$\mathbf{a_z}(t) = -\int_T^t \mathbf{a_x}(\tau)^\top \frac{\partial \boldsymbol{f}_\theta(\mathbf{x}_\tau, \mathbf{z}_\tau, \tau)}{\partial \mathbf{z}_\tau} \,\mathrm{d}\tau. \tag{20}$$

- We can simply replace our $\mathbf{z}$ with $\mathbf{z}_t$ when performing guided generation.
- Enables us to update time-dependent conditioning signal.
- We have the flexibility to only update back to a certain $t \in [0, T)$.

What about neural CDEs?

- Kidger [9, Theorem C.1] showed that any equation of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{h}_\theta(\mathbf{x}_s, \mathbf{z}_s, s)\,\mathrm{d}s, \tag{21}$$

can be rewritten as a neural *controlled differential equation* (CDE) of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{f}_\theta(\mathbf{x}_s, s)\,\mathrm{d}\mathbf{z}_s, \tag{22}$$

where $\int \,\mathrm{d}\mathbf{z}_s$ is the Riemann-Stieltjes integral.

What about neural CDEs?

- Kidger [9, Theorem C.1] showed that any equation of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{h}_\theta(\mathbf{x}_s, \mathbf{z}_s, s)\,\mathrm{d}s, \tag{21}$$

can be rewritten as a neural *controlled differential equation* (CDE) of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{f}_\theta(\mathbf{x}_s, s)\,\mathrm{d}\mathbf{z}_s, \tag{22}$$

where $\int \,\mathrm{d}\mathbf{z}_s$ is the Riemann-Stieltjes integral.

- Note, the converse is not true.

What about neural CDEs?

- Kidger [9, Theorem C.1] showed that any equation of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{h}_\theta(\mathbf{x}_s, \mathbf{z}_s, s)\,\mathrm{d}s, \tag{21}$$

  can be rewritten as a neural *controlled differential equation* (CDE) of the form

$$\mathbf{x}_t = \mathbf{x}_0 + \int_0^t \boldsymbol{f}_\theta(\mathbf{x}_s, s)\,\mathrm{d}\mathbf{z}_s, \tag{22}$$

  where $\int \,\mathrm{d}\mathbf{z}_s$ is the Riemann-Stieltjes integral.
- Note, the converse is not true.
- Neural CDEs used $\mathbf{z}_t$ as an additional control signal, but were not interested in **updating** $\mathbf{z}_t$.

Time scheduled conditional information in practice

- Suppose we have a fully observed, but irregularly sampled time series $\{\mathbf{z}_{t_i}\}_{i=1}^{N}$ with $0 = t_0 < \cdots < t_n = T$.
- Define $\mathbf{Z} : [0, T] \to \mathbb{R}^z$ as the natural cubic spline with knots at $t_0, \ldots, t_n$ such that $\mathbf{Z}(t_i) = \mathbf{z}_{t_i}$.
- Can use this with adaptive step size solvers for $\mathbf{a}_{\mathbf{x}}(t)$.

Remarks about using the continuous adjoint equations with diffusion models

Approaches for guided generation

- We will broadly categorize approaches into two categories:
  - **Solution Optimization** - Only cares about finding the optimal output, $\mathbf{x}_0$,
  - **End-to-End Optimization** - Cares about finding the optimal *entire* solution trajectory and associated variables, $(\{\mathbf{x}_{t_i}\}_{i=1}^{n}, \mathbf{z}, \theta)$.

Approaches for guided generation

- We will broadly categorize approaches into two categories:
  - **Solution Optimization** - Only cares about finding the optimal output, $\mathbf{x}_0$,
  - **End-to-End Optimization** - Cares about finding the optimal *entire* solution trajectory and associated variables, $(\{\mathbf{x}_{t_i}\}_{i=1}^{n}, \mathbf{z}, \theta)$. **Focus on this category.**

Approaches for guided generation

- We will broadly categorize approaches into two categories:
  - **Solution Optimization** - Only cares about finding the optimal output, $\mathbf{x}_0$,
  - **End-to-End Optimization** - Cares about finding the optimal *entire* solution trajectory and associated variables, $(\{\mathbf{x}_{t_i}\}_{i=1}^n, \mathbf{z}, \theta)$.
- For this later category we need to backpropogate through an ODE/SDE solve of the diffusion model

Backpropagation through neural differential equations

- **Discretize-then-optimize** (DTO)
  - Simplest approach, just backprop through the solver.
  - Pros: Accuracy of gradients, fast, and easy to implement.
  - Cons: Memory intensive, optimization w.r.t. discretization and not continuous ideal.

[5] Patrick Kidger. "On Neural Differential Equations". PhD thesis. Oxford University, 2022.

Backpropagation through neural differential equations

- **Discretize-then-optimize** (DTO)
  - Simplest approach, just backprop through the solver.
  - Pros: Accuracy of gradients, fast, and easy to implement.
  - Cons: Memory intensive, optimization w.r.t. discretization and not continuous ideal.
- **Optimize-then-discretize** (OTD)
  - The adjoint method, numerically solve adjoint equations for gradients.
  - Pros: Memory efficiency, flexibility.
  - Cons: Computational cost, truncation errors.

[5] Patrick Kidger. "On Neural Differential Equations". PhD thesis. Oxford University, 2022.

Backpropagation through neural differential equations

- **Discretize-then-optimize** (DTO)
  - Simplest approach, just backprop through the solver.
  - Pros: Accuracy of gradients, fast, and easy to implement.
  - Cons: Memory intensive, optimization w.r.t. discretization and not continuous ideal.
- **Optimize-then-discretize** (OTD)
  - The adjoint method, numerically solve adjoint equations for gradients.
  - Pros: Memory efficiency, flexibility.
  - Cons: Computational cost, truncation errors.
- **Reversible solvers**
  - Possible best of both worlds.
  - Pros: Memory efficient and accurate gradients.
  - Cons: Low-order and poor stability (recent work has started to address this).
- For more details we refer to Patrick Kidger's monograph on neural differential equations[5].

[5] Patrick Kidger. "On Neural Differential Equations". PhD thesis. Oxford University, 2022.

Techniques for OTD optimization of diffusion models

Table 2: Overview of different OTD methods for diffusion models.

Method	**ODE**	**SDE**	**Key Idea**
DiffPure [15]	✗	✓	First to consider OTD for diffusion models
AdjointDPM [16]	✓	✗	Exponential integrators with OTD
Implicit Diffusion [12]	✓	✓	Time parallelization of OTD
AdjointDEIS [1]	✓	✓	Bespoke solvers for ODE/SDE

The flexibility of OTD

- Beyond memory efficiency OTD provides a very flexible framework.

The flexibility of OTD

- Beyond memory efficiency OTD provides a very flexible framework.
- We can have a different number of discretization steps for the forward and backward solve (more relevant for models with a large number of sampling steps).

The flexibility of OTD

- Beyond memory efficiency OTD provides a very flexible framework.
- We can have a different number of discretization steps for the forward and backward solve (more relevant for models with a large number of sampling steps).
- Area of future research, can we get away with less “accurate” gradients in practice?

The flexibility of OTD

- Beyond memory efficiency OTD provides a very flexible framework.
- We can have a different number of discretization steps for the forward and backward solve (more relevant for models with a large number of sampling steps).
- Area of future research, can we get away with less “accurate” gradients in practice?
- Different solvers for the underlying state $\{\mathbf{x}_t\}$ and gradients $\{\mathbf{a}_{\mathbf{x}}(t)\}$.

Truncation errors

- Potentially inaccurate gradients due to different estimates of $\{\mathbf{x}_t\}$ in the forward and backwards solve.

Truncation errors

- Potentially inaccurate gradients due to different estimates of $\{\mathbf{x}_t\}$ in the forward and backwards solve.
- Consider a first-order solver (DDIM, DPM-Solver-1) then given

$$\hat{\mathbf{x}}_0 = \frac{\alpha_0}{\alpha_t}\mathbf{x}_t - \sigma_0(e^h - 1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t), \quad h = \lambda_0 - \lambda_t, \tag{23}$$

$$\hat{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_0}\hat{\mathbf{x}}_0 - \sigma_t(e^{-h} - 1)\boldsymbol{\epsilon}_\theta(\hat{\mathbf{x}}_0, 0), \tag{24}$$

it is not necessary that for all $t \in [0, T]$ and $\mathbf{x}_t \in \mathbb{R}^d$ that

$$\hat{\mathbf{x}}_t = \mathbf{x}_t, \tag{25}$$

holds.

Truncation errors

- Potentially inaccurate gradients due to different estimates of $\{\mathbf{x}_t\}$ in the forward and backwards solve.
- Consider a first-order solver (DDIM, DPM-Solver-1) then given

$$\hat{\mathbf{x}}_0 = \frac{\alpha_0}{\alpha_t}\mathbf{x}_t - \sigma_0(e^h - 1)\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t), \quad h = \lambda_0 - \lambda_t, \tag{23}$$

$$\hat{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_0}\hat{\mathbf{x}}_0 - \sigma_t(e^{-h} - 1)\boldsymbol{\epsilon}_\theta(\hat{\mathbf{x}}_0, 0), \tag{24}$$

it is not necessary that for all $t \in [0, T]$ and $\mathbf{x}_t \in \mathbb{R}^d$ that

$$\hat{\mathbf{x}}_t = \mathbf{x}_t, \tag{25}$$

holds.

- This can be mitigated with small step sizes at the cost of increased compute.

Stability of OTD

- Consider a simple ODE on $t \in [0, T]$ given by

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = \lambda y(t), \quad y(0) = y_0, \quad \lambda < 0. \tag{26}$$

Stability of OTD

- Consider a simple ODE on $t \in [0, T]$ given by

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = \lambda y(t), \quad y(0) = y_0, \quad \lambda < 0. \tag{26}$$

- Most ODE solvers with a non-trivial region of stability will solve the ODE without issue.
- As $\lambda < 0$, the errors decay exponentially.

Stability of OTD

- Consider a simple ODE on $t \in [0, T]$ given by

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = \lambda y(t), \quad y(0) = y_0, \quad \lambda < 0. \tag{26}$$

- Most ODE solvers with a non-trivial region of stability will solve the ODE without issue.
- As $\lambda < 0$, the errors decay exponentially.
- **However**, for backwards in time solve from $y(T)$ the errors will **grow** exponentially.

Stability of OTD

- Consider a simple ODE on $t \in [0, T]$ given by

$$\frac{\mathrm{d}y}{\mathrm{d}t}(t) = \lambda y(t), \quad y(0) = y_0, \quad \lambda < 0. \tag{26}$$

- Most ODE solvers with a non-trivial region of stability will solve the ODE without issue.
- As $\lambda < 0$, the errors decay exponentially.
- However, for backwards in time solve from $y(T)$ the errors will grow exponentially.
- The adjoint ODE has the same stability properties as y.

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.
- Φ is said to be *algebraically reversible* if we can compute $(\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}}) \mapsto (\mathbf{x}_{t_i}, \alpha_{t_i})$.

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.
- Φ is said to be *algebraically reversible* if we can compute $(\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}}) \mapsto (\mathbf{x}_{t_i}, \alpha_{t_i})$.
- Since the solver is algebraically reversible there is no truncation error.

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.
- Φ is said to be *algebraically reversible* if we can compute $(\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}}) \mapsto (\mathbf{x}_{t_i}, \alpha_{t_i})$.
- Since the solver is algebraically reversible there is no truncation error.
- Reversible solvers **may** have better stability.

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.
- Φ is said to be *algebraically reversible* if we can compute $(\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}}) \mapsto (\mathbf{x}_{t_i}, \alpha_{t_i})$.
- Since the solver is algebraically reversible there is no truncation error.
- Reversible solvers **may** have better stability.
- We will review several non-symplectic[6] reversible solvers.

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Reversible solvers

- Let Φ be a numerical scheme which iteratively computes $(\mathbf{x}_{t_i}, \alpha_{t_i}) \mapsto (\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}})$ where α_{t_i} is extra auxillary information.
- Φ is said to be *algebraically reversible* if we can compute $(\mathbf{x}_{t_{i+1}}, \alpha_{t_{i+1}}) \mapsto (\mathbf{x}_{t_i}, \alpha_{t_i})$.
- Since the solver is algebraically reversible there is no truncation error.
- Reversible solvers **may** have better stability.
- We will review several non-symplectic[6] reversible solvers.
- Consider a neural ODE on the time interval $[0, T]$ with definition

$$\mathbf{x}(0) = \mathbf{x}_0, \qquad \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t}(t) = \boldsymbol{f}_\theta(\mathbf{x}(t), t). \tag{27}$$

[6] Many symplectic solvers are algebraically reversible. For more details we refer to [7].

Asynchronous leapfrog method (ICLR 2021)

- Initially proposed by Mutze[7] and popularized by Zhuang *et al.*[8]
- Is a second-order method.

Forward pass

With $h := t_{i+1} - t_i$ the forward pass is defined as

$$\begin{aligned} \mathbf{x}_{t_i+\frac{h}{2}} &= \mathbf{x}_{t_i} + \frac{1}{2}\mathbf{v}_{t_i}h, \\ \mathbf{v}_{t_{i+1}} &= 2\boldsymbol{f}_\theta(\mathbf{x}_{t_i+\frac{h}{2}}, t_i + h/2) - \mathbf{v}_{t_i}, \\ \mathbf{x}_{t_{i+1}} &= \mathbf{x}_{t_i} + \boldsymbol{f}_\theta(\mathbf{x}_{t_i+h/2}, t_i + \frac{h}{2})h. \end{aligned}$$

[7] Ulrich Mutze. "An asynchronous leapfrog method II". In: *arXiv preprint arXiv:1311.6602* (2013).

[8] Juntang Zhuang et al. "MALI: A memory efficient and reverse accurate integrator for Neural ODEs". In: *International Conference on Learning Representations* (2021).

Asynchronous leapfrog method (ICLR 2021)

- Initially proposed by Mutze[7] and popularized by Zhuang *et al.*[8]
- Is a second-order method.

Backward pass

With $h := t_{i+1} - t_i$ the backward pass is defined as

$$\begin{aligned} \mathbf{x}_{t_i+\frac{h}{2}} &= \mathbf{x}_{t_{i+1}} - \frac{1}{2}\mathbf{v}_{t_{i+1}} h, \\ \mathbf{v}_{t_i} &= 2\boldsymbol{f}_\theta(\mathbf{x}_{t_i+\frac{h}{2}}, t_i + h/2) - \mathbf{v}_{t_{i+1}}, \\ \mathbf{x}_{t_i} &= \mathbf{x}_{t_{i+1}} - \boldsymbol{f}_\theta(\mathbf{x}_{t_i+\frac{h}{2}}, t_i + h/2)h. \end{aligned}$$

[7] Ulrich Mutze. "An asynchronous leapfrog method II". In: *arXiv preprint arXiv:1311.6602* (2013).

[8] Juntang Zhuang et al. "MALI: A memory efficient and reverse accurate integrator for Neural ODEs". In: *International Conference on Learning Representations* (2021).

Reversible Heun (NeurIPS 2021)

- Proposed by Kidger *et al.*[9]
- Works for neural ODEs, CDEs, and SDEs.
- ODE solver is a second-order method and exhibits a strong convergence of order 1 if the noise is additive.

Forward pass

With $h := t_{i+1} - t_i$ the forward pass is defined as

$$\widehat{\mathbf{x}}_{t_{i+1}} = 2\mathbf{x}_{t_i} - \widehat{\mathbf{x}}_{t_i} + \boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_i}, t_i)h,$$
$$\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \frac{1}{2}\big(\boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_{i+1}}, t_{i+1}), \boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_i}, t_i)\big)h.$$

[9] Patrick Kidger et al. "Efficient and Accurate Gradients for Neural SDEs". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 18747–18761. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/9ba196c7a6e89eafd0954de80fc1b224-Paper.pdf.

Reversible Heun (NeurIPS 2021)

- Proposed by Kidger *et al.*[9]
- Works for neural ODEs, CDEs, and SDEs.
- ODE solver is a second-order method and exhibits a strong convergence of order 1 if the noise is additive.

Backward pass

With $h := t_{i+1} - t_i$ the backward pass is defined as

$$\begin{aligned}\widehat{\mathbf{x}}_{t_i} &= 2\mathbf{x}_{t_{i+1}} - \widehat{\mathbf{x}}_{t_{i+1}} - \boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_{i+1}}, t_{i+1})h,\\ \mathbf{x}_{t_i} &= \mathbf{x}_{t_{i+1}} - \frac{1}{2}\big(\boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_{i+1}}, t_{i+1}), \boldsymbol{f}_\theta(\widehat{\mathbf{x}}_{t_i}, t_i)\big)h.\end{aligned}$$

Note, this method is also *symmetric*.

[9] Patrick Kidger et al. "Efficient and Accurate Gradients for Neural SDEs". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 18747–18761. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/9ba196c7a6e89eafd0954de80fc1b224-Paper.pdf.

Stability of the reversible solvers

- Consider the test equation $\dot{y} = \lambda y$ with $y(0) = 1$ defined on $[0, \infty)$ with $\lambda \in \mathbb{C}$.

Stability of the reversible solvers

- Consider the test equation $\dot{y} = \lambda y$ with $y(0) = 1$ defined on $[0, \infty)$ with $\lambda \in \mathbb{C}$.
- The region of stability of a numerical solver is the values of λ which ensures the numerical solver converges for a fixed step size.

Stability of the reversible solvers

- Consider the test equation $\dot{y} = \lambda y$ with $y(0) = 1$ defined on $[0, \infty)$ with $\lambda \in \mathbb{C}$.
- The region of stability of a numerical solver is the values of λ which ensures the numerical solver converges for a fixed step size.
- The region of stability for the asynchronous leapfrog method and reversible Heun are both the complex interval $[-i, i]$.

Stability of the reversible solvers

- Consider the test equation $\dot{y} = \lambda y$ with $y(0) = 1$ defined on $[0, \infty)$ with $\lambda \in \mathbb{C}$.
- The region of stability of a numerical solver is the values of λ which ensures the numerical solver converges for a fixed step size.
- The region of stability for the asynchronous leapfrog method and reversible Heun are both the complex interval $[-i, i]$.
- Could be due to an unstable step of form $2A - B$, instability is amplified when
  - $\mathbf{v}_{t_i}$ and $\boldsymbol{f}_\theta(\mathbf{x}_{t_i}, t_i)$ drift apart (asynchronous leapfrog),
  - $\mathbf{x}_{t_i}$ and $\widehat{\mathbf{x}}_{t_i}$ drift apart (reversible Heun).

McCallum-Foster method

- Recently, McCallum and Foster[10] showed that it is possible to construct an algebraically reversible solver from **any** explicit numerical ODE solver $\Phi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$.
- Suppose the explicit solver can be expressed as $\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \Phi_h(\mathbf{x}_{t_i}, t_i)$ with step size h.

[10] Sam McCallum and James Foster. "Efficient, Accurate and Stable Gradients for Neural ODEs". In: *arXiv preprint arXiv:2410.11648* (2024).

McCallum-Foster method

- Recently, McCallum and Foster[10] showed that it is possible to construct an algebraically reversible solver from **any** explicit numerical ODE solver $\Phi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$.
- Suppose the explicit solver can be expressed as $\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \Phi_h(\mathbf{x}_{t_i}, t_i)$ with step size h.
- If Φ has k-th order convergence then *reversible* solver will also have k-th order convergence.

[10] Sam McCallum and James Foster. "Efficient, Accurate and Stable Gradients for Neural ODEs". In: *arXiv preprint arXiv:2410.11648* (2024).

McCallum-Foster method

- Recently, McCallum and Foster[10] showed that it is possible to construct an algebraically reversible solver from **any** explicit numerical ODE solver $\Phi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$.
- Suppose the explicit solver can be expressed as $\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \Phi_h(\mathbf{x}_{t_i}, t_i)$ with step size h.
- If Φ has k-th order convergence then *reversible* solver will also have k-th order convergence.

Forward pass

With $h := t_{i+1} - t_i$ the forward pass is defined as

$$\begin{aligned} \mathbf{x}_{t_{i+1}} &= \lambda \mathbf{x}_{t_i} + (1-\lambda)\widehat{\mathbf{x}}_{t_i} + \Phi_h(\widehat{\mathbf{x}}_{t_i}, t_i) \\ \widehat{\mathbf{x}}_{t_{i+1}} &= \widehat{\mathbf{x}}_{t_i} - \Phi_{-h}(\mathbf{x}_{t_{i+1}}, t_{i+1}), \end{aligned}$$

where $\lambda \in (0, 1]$ is a coupling parameter.

[10] Sam McCallum and James Foster. "Efficient, Accurate and Stable Gradients for Neural ODEs". In: *arXiv preprint arXiv:2410.11648* (2024).

McCallum-Foster method

- Recently, McCallum and Foster[10] showed that it is possible to construct an algebraically reversible solver from **any** explicit numerical ODE solver $\Phi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$.
- Suppose the explicit solver can be expressed as $\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + \Phi_h(\mathbf{x}_{t_i}, t_i)$ with step size h.
- If Φ has k-th order convergence then *reversible* solver will also have k-th order convergence.

Backward pass

With $h := t_{i+1} - t_i$ the backward pass is defined as

$$\begin{aligned}
\widehat{\mathbf{x}}_{t_i} &= \widehat{\mathbf{x}}_{t_{i+1}} + \Phi_{-h}(\mathbf{x}_{t_{i+1}}, t_{i+1}),\\
\mathbf{x}_{t_i} &= \lambda^{-1}\mathbf{x}_{t_{i+1}} + (1 - \lambda^{-1})\widehat{\mathbf{x}}_{t_i} - \lambda^{-1}\Phi_h(\widehat{\mathbf{x}}_{t_i}, t_i).
\end{aligned}$$

[10] Sam McCallum and James Foster. "Efficient, Accurate and Stable Gradients for Neural ODEs". In: *arXiv preprint arXiv:2410.11648* (2024).

Non-trivial stability

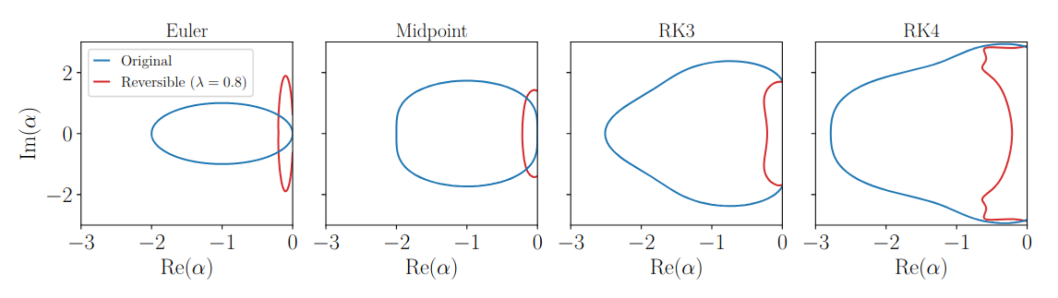


Figure 2: Stability plots from McCallum and Foster [13].

Record the solution trajectory

- A simple fix for the numerical unstability of OTD is to simply cache $\{\mathbf{x}_{t_i}\}_{i=1}^{n}$ during the forward pass.
- Note, we are not storing the internal operations of the solver and network.

Record the solution trajectory

- A simple fix for the numerical unstability of OTD is to simply cache $\{\mathbf{x}_{t_i}\}_{i=1}^{n}$ during the forward pass.
- Note, we are not storing the internal operations of the solver and network.
- The stability is much improved using this method at the cost of increased memory usage.

Record the solution trajectory

- A simple fix for the numerical unstability of OTD is to simply cache $\{\mathbf{x}_{t_i}\}_{i=1}^{n}$ during the forward pass.
- Note, we are not storing the internal operations of the solver and network.
- The stability is much improved using this method at the cost of increased memory usage.
- For diffusion models with a small number of sampling steps this may be acceptable, however.

Record the solution trajectory

- A simple fix for the numerical unstability of OTD is to simply cache $\{\mathbf{x}_{t_i}\}_{i=1}^{n}$ during the forward pass.
- Note, we are not storing the internal operations of the solver and network.
- The stability is much improved using this method at the cost of increased memory usage.
- For diffusion models with a small number of sampling steps this may be acceptable, however.
- A similar approach is that of interpolated adjoints.
- Record $\{x_{\tau_j}\}_{j=1}^{m}$ and $\{\tau_j\}_{j=1}^{m} \subseteq \{t_i\}_{i=1}^{n}$.

Record the solution trajectory

- A simple fix for the numerical unstability of OTD is to simply cache $\{\mathbf{x}_{t_i}\}_{i=1}^{n}$ during the forward pass.
- Note, we are not storing the internal operations of the solver and network.
- The stability is much improved using this method at the cost of increased memory usage.
- For diffusion models with a small number of sampling steps this may be acceptable, however.
- A similar approach is that of interpolated adjoints.
- Record $\{x_{\tau_j}\}_{j=1}^{m}$ and $\{\tau_j\}_{j=1}^{m} \subseteq \{t_i\}_{i=1}^{n}$.
- Then for any $\tau_j > t > \tau_{j+1}$ interpolate between the two samples $\mathbf{x}_{\tau_j}$ and $\mathbf{x}_{\tau_{j+1}}$ to find $\mathbf{x}_t$.

Parallel sampling and optimization

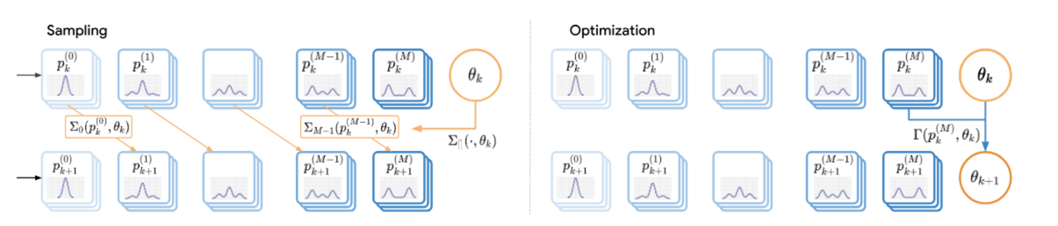


Figure 3: Implicit Diffusion by Marion *et al.*[11]

- Fill up shift register with initial sample trajectory $\{\mathbf{x}_{t_i}\}_{i=1}^n$.
- Can now sample and backpropagate in parallel.
- Because it's a shift register it still takes m steps to propagate an update to $\mathbf{x}_T$.

[11]Pierre Marion et al. "Implicit Diffusion: Efficient Optimization through Stochastic Sampling". In: *arXiv preprint arXiv:2402.05468* (2024).

Thoughts on when and how to use OTD for diffusion models

- Does DTO work? Most accurate in terms of model gradients.

Thoughts on when and how to use OTD for diffusion models

- Does DTO work? Most accurate in terms of model gradients.
- If not, can we record the solution states?

Thoughts on when and how to use OTD for diffusion models

- Does DTO work? Most accurate in terms of model gradients.
- If not, can we record the solution states?
- If not, consider a reversible solver for the backward pass.

Thoughts on when and how to use OTD for diffusion models

- Does DTO work? Most accurate in terms of model gradients.
- If not, can we record the solution states?
- If not, consider a reversible solver for the backward pass.
- Note, in practice OTD seems to work well enough and the gradient inaccuracy might not be a big deal in certain applications.

?

References

[1] Zander W. Blasingame and Chen Liu. “AdjointDEIS: Efficient Gradients for Diffusion Models”. In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024. URL: https://openreview.net/forum?id=fAlcxvrOEX.

[2] Zander W. Blasingame and Chen Liu. “Fast-DiM: Towards Fast Diffusion Morphs”. In: *IEEE Security & Privacy* 22.4 (June 2024), pp. 103–114. DOI: 10.1109/MSEC.2024.3410112.

[3] Zander W. Blasingame and Chen Liu. “Leveraging Diffusion for Strong and High Quality Face Morphing Attacks”. In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 6.1 (2024), pp. 118–131. DOI: 10.1109/TBIOM.2024.3349857.

References ii

[4] Fadi Boutros et al. "ElasticFace: Elastic Margin Loss for Deep Face Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. June 2022, pp. 1578–1587.

[5] Jiankang Deng et al. "Arcface: Additive angular margin loss for deep face recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4690–4699.

[6] Marco Huber et al. "SYN-MAD 2022: Competition on Face Morphing Attack Detection Based on Privacy-aware Synthetic Training Data". In: *2022 IEEE International Joint Conference on Biometrics (IJCB)*. 2022, pp. 1–10. DOI: 10.1109/IJCB54206.2022.10007950.

[7] Patrick Kidger. "On Neural Differential Equations". PhD thesis. Oxford University, 2022.

[8] Patrick Kidger et al. "Efficient and Accurate Gradients for Neural SDEs". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 18747–18761. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/9ba196c7a6e89eafd0954de80fc1b224-Paper.pdf.

[9] Patrick Kidger et al. "Neural controlled differential equations for irregular time series". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6696–6707.

[10] Minchul Kim, Anil K Jain, and Xiaoming Liu. "AdaFace: Quality Adaptive Margin for Face Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022.

[11] Cheng Lu et al. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo et al. Vol. 35. Curran Associates, Inc., 2022, pp. 5775–5787. URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/260a14acce2a89dad36adc8eefe7c59e-Paper-Conference.pdf.

[12] Pierre Marion et al. "Implicit Diffusion: Efficient Optimization through Stochastic Sampling". In: *arXiv preprint arXiv:2402.05468* (2024).

[13] Sam McCallum and James Foster. "Efficient, Accurate and Stable Gradients for Neural ODEs". In: *arXiv preprint arXiv:2410.11648* (2024).

[14] Ulrich Mutze. "An asynchronous leapfrog method II". In: *arXiv preprint arXiv:1311.6602* (2013).

[15] Weili Nie et al. "Diffusion Models for Adversarial Purification". In: *International Conference on Machine Learning (ICML)*. 2022.

[16] Jiachun Pan et al. "AdjointDPM: Adjoint Sensitivity Method for Gradient Backpropagation of Diffusion Probabilistic Models". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: `https://openreview.net/forum?id=y33lDRBgWI`.

[17] Ulrich Scherhag et al. "Biometric Systems under Morphing Attacks: Assessment of Morphing Techniques and Vulnerability Reporting". In: *2017 International Conference of the Biometrics Special Interest Group (BIOSIG)*. 2017, pp. 1–7. DOI: `10.23919/BIOSIG.2017.8053499`.

[18] Yang Song et al. "Score-Based Generative Modeling through Stochastic Differential Equations". In: *International Conference on Learning Representations*. 2021. URL: `https://openreview.net/forum?id=PxTIG12RRHS`.

References v

[19] Haoyu Zhang et al. "MIPGAN—Generating Strong and High Quality Morphing Attacks Using Identity Prior Driven GAN". In: *IEEE Transactions on Biometrics, Behavior, and Identity Science* 3.3 (2021), pp. 365–383. DOI: 10.1109/TBIOM.2021.3072349.

[20] Haoyu Zhang et al. "Morph-PIPE: Plugging in Identity Prior to Enhance Face Morphing Attack Based on Diffusion Model". In: *Norwegian Information Security Conference (NISK)*. 2023.

[21] Juntang Zhuang et al. "MALI: A memory efficient and reverse accurate integrator for Neural ODEs". In: *International Conference on Learning Representations* (2021).