Neural Network Algorithms for Stochastic Optimal Control

SIAM OP26 – MS38 “Learning-informed Optimization and Control”

Lars Ruthotto
Departments of Mathematics and Computer Science
Emory University
lruthotto@emory.edu
larsruthotto

Stochastic Optimal Control: Setup and Applications

Setup state $\mathbf{z} \in \mathbb{R}^d$, control $\mathbf{u} \in \mathbb{R}^m$:

$$\Phi(t, \mathbf{x}) = \min_{\mathbf{u}} \mathbb{E}\left[g(\mathbf{z}(T)) + \int_t^T L(s, \mathbf{z}, \mathbf{u})\, ds \right] \quad \text{s.t.} \quad d\mathbf{z}(s) = f(s, \mathbf{z}, \mathbf{u})\, ds + \sigma\, dW, \quad \mathbf{z}(t) = \mathbf{x}$$

- drift f and noise $\sigma\, dW$ govern the dynamics
- running cost L along the path, terminal cost g at $s = T$

Rare Event Analysis

- f: system dynamics
- L: trajectory penalty
- g: event indicator

e.g. chaotic Lorenz, extreme weather, Burgers–Huxley

Bayesian Inverse Problems

- f: score / flow prior
- L: control penalty
- g: neg. log-likelihood

e.g. inverse scattering, structural health, inverse design

Stochastic PDE Control

- f: discretized SPDE
- L: tracking + control
- g: terminal error

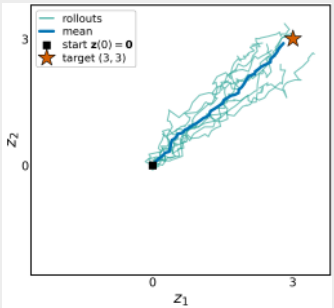
e.g. stochastic reaction-diffusion, FitzHugh–Nagumo

common challenge: high-dim, complex dynamics, reliable training

Two Running Examples

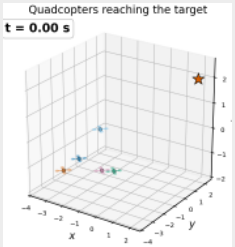
Ex 1: 100-D Shifted Target

- $d\mathbf{z} = 2\mathbf{u}\,ds + \sigma\,dW$, $\mathbf{z}(0) = \mathbf{0}$ Li et al. 2024
- $L = \|\mathbf{u}\|^2$; $g(\mathbf{x}) = \ln\left(\frac{1}{2}(1 + \|\mathbf{x} - \mathbf{x}_{\text{ref}}\|^2)\right)$
- $\mathbf{x}_{\text{ref}} = (3, \ldots, 3) \in \mathbb{R}^{100}$; high-$d$, target far from initial distribution



Ex 2: 12-D Quadcopter

- state (d=12): position, Euler angles Θ, linear velocity $\mathbf{v}$ & angular velocity $\boldsymbol{\omega}$
- drift $f = \left(\mathbf{v},\ \boldsymbol{\omega},\ \frac{F}{m}\mathbf{n}(\Theta) - g_0\mathbf{e}_3,\ \boldsymbol{\tau}\right)$: 1 thrust F + 3 torques $\boldsymbol{\tau}$
- $\mathbf{n}(\Theta) = \left(s_\psi s_\varphi + c_\psi s_\theta c_\varphi,\ -c_\psi s_\varphi + s_\psi s_\theta c_\varphi,\ c_\theta c_\varphi\right)$: attitude rotates the thrust (nonlinear)



The Hamilton–Jacobi–Bellman (HJB) Equation

The value function $\Phi(t, \mathbf{x})$ is the minimal expected cost-to-go from state $\mathbf{x}$ at time t.

Quick facts from optimal control theory (write $\nabla\Phi$ for the spatial gradient $\nabla_{\mathbf{x}}\Phi$)

1. feedback form (Pontryagin Maximum Principle, PMP) relates optimal control and value function

$$\mathbf{u}^*(s) \in \arg\min_{\mathbf{u}} \mathcal{H}(s, \mathbf{z}(s), \mathbf{p}(s), \mathbf{u}), \qquad \mathbf{p}(s) := \nabla\Phi(s, \mathbf{z}(s))$$

   with Hamiltonian $\mathcal{H}(s, \mathbf{z}, \mathbf{p}, \mathbf{u}) = \mathbf{p}^\top f(s, \mathbf{z}, \mathbf{u}) + L(s, \mathbf{z}, \mathbf{u})$; the costate $\mathbf{p}$ *is* the value gradient $\nabla\Phi$ evaluated along $\mathbf{z}(s)$

2. value function Φ satisfies HJB:

$$\partial_s\Phi(s, \mathbf{x}) + \tfrac{\sigma^2}{2}\Delta\Phi(s, \mathbf{x}) + \min_{\mathbf{u}} \mathcal{H}(s, \mathbf{x}, \nabla\Phi, \mathbf{u}) = 0$$

$$\Phi(T, \mathbf{x}) = g(\mathbf{x})$$

costate $\mathbf{p} = \nabla\Phi$ links control and value — every method below approximates it

Neural Network Solvers for High-Dimensional SOC

- **PINN-based HJB solvers** Sirignano and Spiliopoulos 2018; Raissi 2018
  learn Φ_θ on a fixed sampling distribution; loss $=$ HJB residual
- **Forward–Backward SDE (FB-SDE) / Deep BSDE** Han et al. 2018
  parametrize Φ_θ and $\sigma^\top \nabla \Phi_\theta$ along an uncontrolled forward rollout; loss $=$ backward-SDE (BSDE) terminal residual (Feynman–Kac)
- **Neural ODE/SDE (NeuralSOC)** Onken et al. 2022; Li et al. 2024
  learn Φ_θ; sample via PMP feedback; loss $=$ HJB residual $+$ terminal
- **Diffusion-Optimization (matching)** Domingo-Enrich et al. 2024; Domingo-Enrich et al. 2025; Blessing et al. 2025
  learn $\mathbf{u}_\theta$ directly; sample under current policy; loss $=$ matching / log-ratio

Forward–Backward SDE (FB-SDE) / Deep BSDE

Feynman–Kac semilinear representation Han et al. 2018 along an uncontrolled forward rollout $d\mathbf{z} = f(s, \mathbf{z}, \mathbf{0})\, ds + \sigma\, dW$, $\mathbf{z}(0) \sim \rho_0$, the value function satisfies the backward identity:

$$\Phi(s, \mathbf{z}(s)) \;=\; g(\mathbf{z}(T)) \;+\; \int_s^T \min_{\mathbf{u}} \big\{ (\nabla\Phi)^\top f(r, \mathbf{z}, \mathbf{u}) + L(r, \mathbf{z}, \mathbf{u}) \big\}\, dr \;-\; \int_s^T \big(\sigma^\top \nabla\Phi\big)^\top dW$$

equality holds pathwise (a.s. along the rollout) — Itô's lemma applied to $\Phi(s, \mathbf{z}(s))$

What is learned scalar $\Phi_{\boldsymbol{\theta}}(0, \mathbf{x})$ plus a network $\mathbf{v}_{\boldsymbol{\theta}}(s, \cdot) \approx \sigma^\top \nabla\Phi_{\boldsymbol{\theta}}(s, \cdot)$ at each Euler–Maruyama step $s_0, \ldots, s_N$ (so $\nabla\Phi_{\boldsymbol{\theta}} = \mathbf{v}_{\boldsymbol{\theta}}/\sigma$)

Training problem accumulate the value forward via the BSDE increment, then match $\widehat{\Phi}_N$ to the terminal g:

$$\widehat{\Phi}_{n+1} = \widehat{\Phi}_n - \min_{\mathbf{u}} \big\{ (\nabla\Phi_{\boldsymbol{\theta}})^\top f + L \big\} \Delta s + \mathbf{v}_{\boldsymbol{\theta}}(s_n, \mathbf{z}_n)^\top \Delta W_n, \quad \min_{\boldsymbol{\theta}} \mathbb{E} \big\| \widehat{\Phi}_N - g(\mathbf{z}(T)) \big\|^2$$

How it learns $\widehat{\Phi}_N$ depends on all the gradient networks $\mathbf{v}_{\boldsymbol{\theta}}(s_0, \cdot), \ldots, \mathbf{v}_{\boldsymbol{\theta}}(s_{N-1}, \cdot)$; the policy-free path is the convenience (mesh-free) *and* the later limitation (cannot probe a target it never visits)

mesh-free, scales to $d \approx 100$ — but only where the uncontrolled rollout visits

A 100-D Semilinear HJB Benchmark Han et al. 2018

Setup $d = 100$, $T = 1$, $\sigma = \sqrt{2}$

- $d\mathbf{z} = 2\mathbf{u}\,ds + \sigma\,dW$, $\mathbf{z}(0) = \mathbf{0}$
- $L = \|\mathbf{u}\|^2$
- terminal $g(\mathbf{x}) = \ln\big((1 + \|\mathbf{x}\|^2)/2\big)$

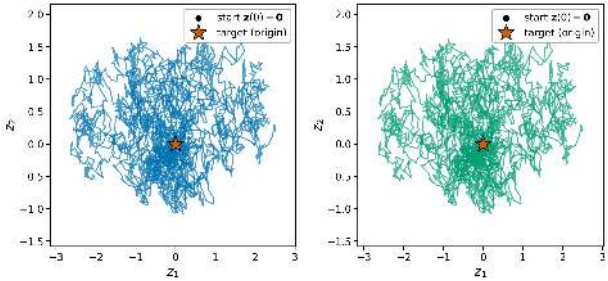
HJB
$$\partial_s \Phi + \Delta\Phi - \|\nabla\Phi\|^2 = 0$$

Reference Feynman–Kac, closed form

$$\Phi(s, \mathbf{x}) = -\ln \mathbb{E}\Big[\exp\big(-g(\mathbf{x} + \sqrt{2}\, W_{T-s})\big)\Big]$$

Baselines at $d = 100$ Deep BSDE Han et al. 2018 matches the reference via uncontrolled forward rollouts; DGM / PINN residual solvers Sirignano and Spiliopoulos 2018 via sampled collocation

2D projection of 100-D trajectories (NeuralSOC)



uncontrolled (left): $d\mathbf{z} = \sigma\,dW$

PMP controlled (right): $d\mathbf{z} = -\nabla\Phi_\theta\,ds + \sigma\,dW$

learned control tiny: mean $\|\mathbf{u}\| \approx 0.14$; rollout gap $< 1.2\%$ of the $\|\mathbf{z}_T\| \approx 14.8$ noise spread

the target sits at the origin where Brownian motion already goes — Deep BSDE / PINN learn near-zero control, so controlled and uncontrolled rollouts coincide

NeuralSOC: A Continuous-time Learning Problem

Three ingredients Onken et al. 2022; Li et al. 2024 extension of the method of characteristics for HJB (cf. Onken et al. deterministic NeuralOC):

- **PMP-driven sampling**: roll out under the feedback control $\mathbf{u}_\theta$
- **HJB** residual along the sampled trajectory penalizes the value function
- **discretize-then-optimize (DTO)** through the neural SDE by reverse-mode AD

Objective given dynamics f and costs L, g, learn Φ_θ (feedback $\mathbf{u}_\theta = \arg\min_{\mathbf{u}} \mathcal{H}$):

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\rho_0}\big[\,\alpha\big(\ell(T) + g(\mathbf{z}(T))\big) + \beta_1\, c_{\mathrm{HJ}}(T) + \beta_2\,\big|\Phi_{\boldsymbol{\theta}}(T,\mathbf{z}(T)) - g(\mathbf{z}(T))\big|\,\big]$$

$$\text{s.t. } d\begin{pmatrix}\mathbf{z}\\ \ell\\ c_{\mathrm{HJ}}\end{pmatrix} = \begin{pmatrix} f(\mathbf{z},\mathbf{u}_{\boldsymbol{\theta}})\\ L(s,\mathbf{z},\mathbf{u}_{\boldsymbol{\theta}})\\ \big|\partial_s\Phi_{\boldsymbol{\theta}} + \mathcal{H}^*(s,\mathbf{z},\nabla\Phi_{\boldsymbol{\theta}}) + \frac{\sigma^2}{2}\Delta\Phi_{\boldsymbol{\theta}}\big|\end{pmatrix} ds + \begin{pmatrix}\sigma\, dW\\ 0\\ 0\end{pmatrix},$$

$\mathbf{z}(0) = \mathbf{x}$, $\ell(0) = c_{\mathrm{HJ}}(0) = 0$

PMP feedback Both examples are control-affine with $L = \frac{1}{2}\|\mathbf{u}\|^2$, so the argmin is explicit: $\mathbf{u}_\theta = -B(\mathbf{z})^\top \nabla\Phi_\theta$ with $B(\mathbf{z}) = \partial_{\mathbf{u}} f$. Ex 1 ($f{=}2\mathbf{u}$): $\mathbf{u}_\theta = -\nabla\Phi_\theta$; quadcopter: closed-form thrust + torque channels.

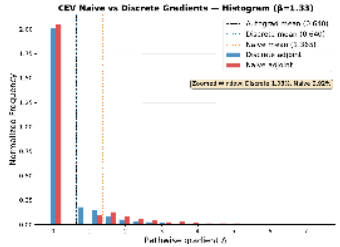
NeuralSOC = HJB residual + PMP-driven forward roll-out

Differentiating Through SDEs

Setup Leburu et al. 2026 gradient of $J(\mathbf{x}_0) = \mathbb{E}[\Phi(\mathbf{z}_T)]$ for an Itô SDE — the building block behind NeuralSOC's DTO adjoint.

Two tales **DTO**: AD through Euler–Maruyama (Itô) / Heun (Stratonovich) steps; **OTD**: continuous adjoint SDE, then discretize.

CEV model, $\beta = 1.33$



discrete adjoint $= 0.640$ (correct); naive $= 1.365$ (biased)

Leburu–Nurbekyan–R. 2026 Leburu et al. 2026 (tutorial leveraging Kidger 2022)

- DTO is *exact* for the discrete objective in both Itô and Stratonovich
- naive OTD on Itô is biased when $\partial_{\mathbf{x}} g$ depends on $\mathbf{x}$ — fix: Itô→Stratonovich (Øksendal)

DTO via AD is exact — in both Itô and Stratonovich

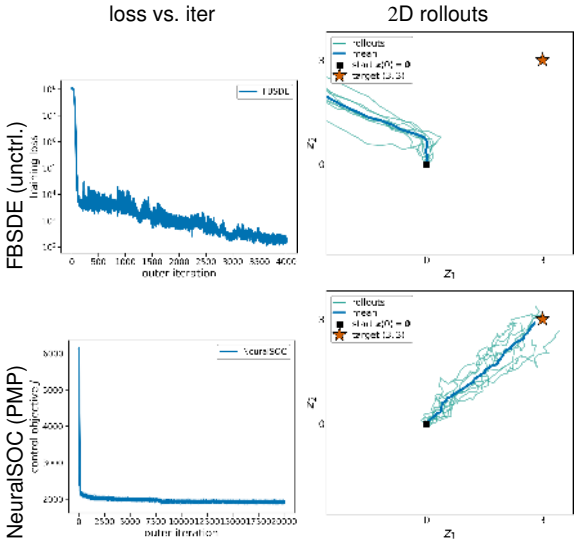
100-D Stress Test: Shifted Target

Same benchmark, two changes

$d = 100$, $T = 1$

- $d\mathbf{z} = 2\mathbf{u}\,ds + \sigma\,dW$, $\mathbf{z}(0) = \mathbf{0}$, $L = \|\mathbf{u}\|^2$
- $g(\mathbf{x}) = \ln\big((1 + \|\mathbf{x} - \mathbf{x}_{\text{ref}}\|^2)/2\big)$, $\mathbf{x}_{\text{ref}} = (3, \ldots, 3)$
  target moved far off the origin
- noise $\sigma = 2\sqrt{2}/5$: lowered so the shift dominates the Brownian spread (else $\sigma\sqrt{T}\sqrt{d} \approx 14$ washes it out)

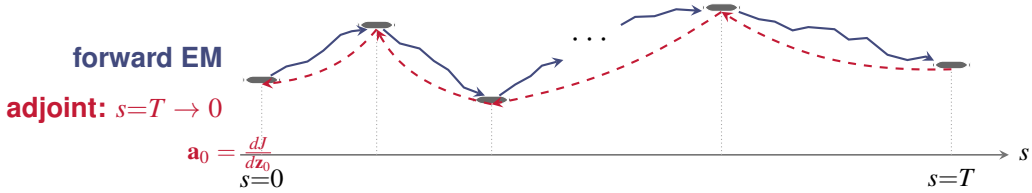
Why it is hard Brownian sampling never reaches $\mathbf{x}_{\text{ref}}$, so the BSDE loss cannot inform Φ along the optimal path Li et al. 2024.



uncontrolled FBSDE misses; PMP sampling reaches ⋆ (same MLP architecture)

loss decay ≠ success: FBSDE never reaches $\mathbf{x}_{\text{ref}}$; PMP sampling does

Computational Cost of NeuralSOC



Discrete pathwise adjoint differentiate $J = g(\mathbf{z}_N)$ w.r.t. $\mathbf{z}_0$ (drop $\mathbf{u}$; ΔW_n independent of $\mathbf{z}_0$):

$$\mathbf{z}_{n+1} = \mathbf{z}_n + f(\mathbf{z}_n)\,\Delta s + \sigma\,\Delta W_n, \qquad \mathbf{a}_N = \nabla g(\mathbf{z}_N), \qquad \mathbf{a}_n = \big(I + \Delta s\,\partial_{\mathbf{z}} f(\mathbf{z}_n)\big)^\top \mathbf{a}_{n+1}$$

$$\frac{dJ}{d\mathbf{z}_0} = \mathbf{a}_0 = M_0^\top M_1^\top \cdots M_{N-1}^\top \nabla g(\mathbf{z}_N), \qquad M_n := I + \Delta s\,\partial_{\mathbf{z}} f(\mathbf{z}_n)$$

- one backward sweep reuses the stored forward states — cost $\approx$ one forward pass, memory $O(N\,d_{\text{act}})$

classic DTO: store the forward rollout, sweep the pathwise adjoint backward

From Full Adjoint to Lean Adjoint

Setting adjoint method on NeuralSOC's controlled SDE $d\mathbf{z} = f(\mathbf{z}, \mathbf{u}_\theta)\, ds + \sigma\, dW$; which terms can we drop?

Full closed-loop adjoint (BSDE)

$$-d\mathbf{p} = \big[\partial_{\mathbf{z}} f^\top \mathbf{p} + \partial_{\mathbf{z}} L \;+\; \underbrace{\partial_{\mathbf{z}} \mathbf{u}_{\boldsymbol{\theta}}^\top\, \partial_{\mathbf{u}} \mathcal{H}}_{= D(s,\mathbf{z})}\big]\, ds - \mathbf{q}\, dW, \quad \mathbf{p}(T) = \nabla g(\mathbf{z}(T))$$

with PMP residual $\partial_{\mathbf{u}} \mathcal{H} = \partial_{\mathbf{u}} f^\top \mathbf{p} + \partial_{\mathbf{u}} L$.

Drop D $\partial_{\mathbf{z}} \mathbf{u}_{\boldsymbol{\theta}}$ is the policy input-Jacobian at every EM step. Dropping it leaves a plain backward sweep, with lean costate $\tilde{\mathbf{p}}$ and lean target $\tilde{\mathbf{a}}$:

$$-d\tilde{\mathbf{p}} = \big[\partial_{\mathbf{z}} f^\top \tilde{\mathbf{p}} + \partial_{\mathbf{z}} L\big]\, ds - \mathbf{q}\, dW, \;\; \tilde{\mathbf{p}}(T) = \nabla g(\mathbf{z}(T)), \;\; \tilde{\mathbf{a}} = -\sigma^\top \tilde{\mathbf{p}}$$

At the optimum, lean = full

- PMP: $\partial_{\mathbf{u}} \mathcal{H}^\star \equiv 0 \Rightarrow D(\mathbf{z}^\star) \equiv 0$; $\mathbf{u}^\star$ critical point of $\mathcal{L}_{\mathrm{AM}}$ Domingo-Enrich et al. 2025; off-policy $D \neq 0$, vanishes as $\mathbf{u}_{\boldsymbol{\theta}} \to \mathbf{u}^\star$

lean is biased off-policy; the bias vanishes as $\mathbf{u}_\theta \to \mathbf{u}^\star$

Adjoint Matching: The Algorithm

Adjoint Matching Domingo-Enrich et al. 2025; Domingo-Enrich and Han 2026
(one outer iteration; repeat to convergence)

(i) **Forward simulate** under frozen $\bar{\mathbf{u}}_{\boldsymbol{\theta}} = \mathrm{sg}(\mathbf{u}_{\boldsymbol{\theta}})$: $d\mathbf{z} = f(\mathbf{z}, \bar{\mathbf{u}}_{\boldsymbol{\theta}})\, ds + \sigma\, dW$, $\mathbf{z}(0) \sim \rho_0$

(ii) **Lean adjoint BSDE** backward: $-d\tilde{\mathbf{p}} = \big[\partial_{\mathbf{z}} f^\top \tilde{\mathbf{p}} + \partial_{\mathbf{z}} L\big] ds - \mathbf{q}\, dW$, $\tilde{\mathbf{p}}(T) = \nabla g$, $\tilde{\mathbf{a}} = -\sigma^\top \tilde{\mathbf{p}}$

(iii) **Regress** $\mathbf{u}_{\boldsymbol{\theta}}$ onto $\tilde{\mathbf{a}}$: $\min_{\boldsymbol{\theta}}\ \mathbb{E}\int_0^T \frac{1}{2}\, \|\mathbf{u}_{\boldsymbol{\theta}}(s, \mathbf{z}(s)) - \tilde{\mathbf{a}}(s)\|^2\, ds$

Why this loss? when the regression residual *is* the PMP residual ($L = \frac{1}{2}\|\mathbf{u}\|^2$, $\partial_{\mathbf{u}} f = \sigma$), its gradient *is* the policy gradient:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{AM}} = \mathbb{E}\int_0^T (\partial_{\boldsymbol{\theta}} \mathbf{u}_{\boldsymbol{\theta}})^\top \big(\mathbf{u}_{\boldsymbol{\theta}} - \tilde{\mathbf{a}}\big)\, ds = \mathbb{E}\int_0^T (\partial_{\boldsymbol{\theta}} \mathbf{u}_{\boldsymbol{\theta}})^\top \underbrace{\big(\partial_{\mathbf{u}} L + \sigma^\top \tilde{\mathbf{p}}\big)}_{=\partial_{\mathbf{u}} \mathcal{H}}\, ds = \nabla_{\boldsymbol{\theta}} J[\mathbf{u}_{\boldsymbol{\theta}}]$$

convex regression, **no backprop through the SDE**; **weight-free** = no importance / Girsanov reweighting (unlike SOCM log-ratio losses), a plain L_2 fit onto the Gauss–Newton / PMP-residual direction; minimized with Adam.

the regression gradient *is* the policy gradient — no backprop through the SDE

Lean AM on the 100-D Shifted Target ($\partial_{\mathbf{z}} f \equiv 0$)

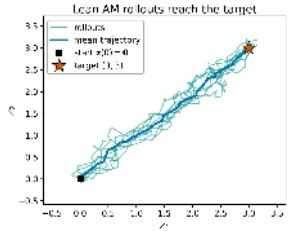
Controlled comparison (our code)

- Ex 1: $f = 2\mathbf{u} \Rightarrow \partial_{\mathbf{z}} f \equiv 0$ — dropping D costs nothing structurally
- NeuralSOC: value Φ-net (OT-Flow, $20{,}184$ p.) *or* MLP control net ($628{,}836$ p.); lean AM uses the same MLP
- matched 2×10^4 iters, batch 64 (representative run) — final J: lean AM $884 \approx$ NeuralSOC-MLP $908 <$ Φ-net 1956

Why this works $\partial_{\mathbf{z}} f \equiv 0$: lean AM and NeuralSOC on the same MLP converge together — the lean drop is free.

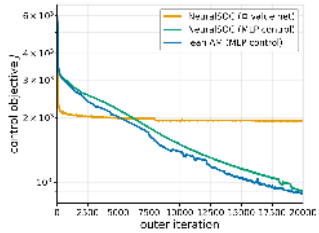
lean AM matches NeuralSOC where $\partial_{\mathbf{z}} f \equiv 0$ — matching's home regime

Lean AM trajectories



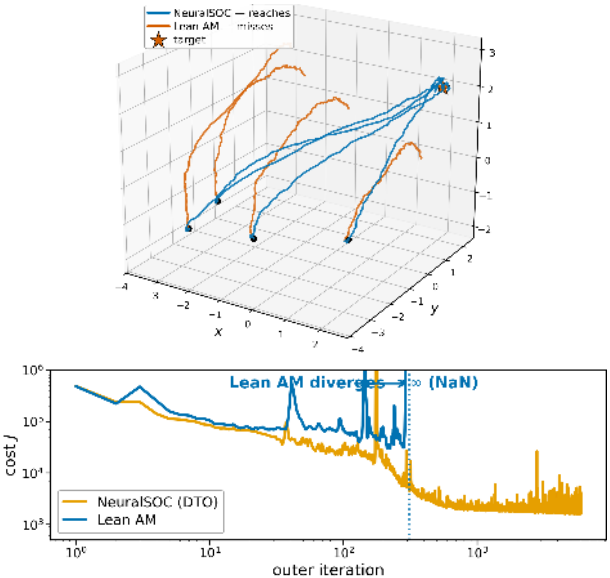
2D projection, start $\mathbf{0}$, target $\star$ at $(3,3)$

Control objective vs. outer iter



all converge (E=0); the two MLP variants track each other

Lean AM on the 12-D Quadcopter (expansive $\partial_{\mathbf{z}} f$)



Experiment control-affine drift f (see Two Examples), wind $\sigma{=}0.2$; attitude tilts $\mathbf{n}(\Theta) \Rightarrow \partial_{\mathbf{z}} f$ expansive Onken et al. 2022

- architecture controlled: both use an MLP policy — NeuralSOC (MLP control, DTO adjoint) *vs.* lean AM (lean adjoint); same T, noise, seeds

Interpretation

- NeuralSOC reaches across $\sigma \in \{0, 0.1, 0.3, 1.0\}$; lean AM $\to \infty$/NaN
- closed-loop $E > 0$; bias amplifies $\sim e^E$ — the lean-vs-full adjoint is the cause

DTO converges, lean AM diverges — same problem

The Lean-Drop Bias and the Expansion Integral E

Setting linearize around $(\mathbf{z}^\star, \mathbf{u}^\star)$ ($\sigma_W{=}0$, mean perturbation), $B := \partial_{\mathbf{u}} f$; perturb $\mathbf{u}_{\boldsymbol{\theta}} = \mathbf{u}^\star + \delta\mathbf{u}$ and track one outer iteration $\delta\mathbf{u} \mapsto \delta\mathbf{u}^+$.

1. Forward variational ODE along $\mathbf{z}^\star$:

$$\tfrac{d}{ds}\delta\mathbf{z} = \partial_{\mathbf{z}} f\, \delta\mathbf{z} + B\, \delta\mathbf{u}, \qquad \delta\mathbf{z}(0) = 0$$

2. Adjoint linearization linearizing the lean adjoint about $\tilde{\mathbf{p}}^\star$: lean generator $\partial_{\mathbf{z}} f^\top$ plus a curvature source ($\partial_{\mathbf{z}} L$, $\partial_{\mathbf{z}} f^\top \mathbf{p}$ differentiated):

$$-\tfrac{d}{ds}\delta\tilde{\mathbf{p}} = \partial_{\mathbf{z}} f^\top \delta\tilde{\mathbf{p}} + \underbrace{(\partial_{\mathbf{zz}} \mathcal{H}^\star)\, \delta\mathbf{z}}_{\text{curvature source}}, \quad \delta\tilde{\mathbf{p}}(T) = \nabla^2 g\, \delta\mathbf{z}(T)$$

3. Regression update $\mathbf{u}_{\boldsymbol{\theta}}^+$ regresses onto $\mathbf{a} = -B^\top \tilde{\mathbf{p}}$ at fixed $\mathbf{z} \Rightarrow \delta\mathbf{u}^+ \propto -B^\top \delta\tilde{\mathbf{p}}$.

Composing $\delta\mathbf{u} \to \delta\mathbf{z} \to \delta\tilde{\mathbf{p}} \to \delta\mathbf{u}^+$ is a linear map whose gain is the state-transition matrix (matrizant) of $\partial_{\mathbf{z}} f$ along $\mathbf{z}^\star$:

$$\|\delta\mathbf{u}^+\| \;\le\; \exp(E)\, \|\delta\mathbf{u}\|, \qquad \boxed{E \;=\; \int_0^T \partial_{\mathbf{z}} f\big(\mathbf{z}^\star(s), \mathbf{u}^\star(s)\big)\, ds}$$

$E > 0$: lean iteration diverges before the on-policy bias can vanish

Why Common Fixes Don't Help in This Example

Common diagnosis E depends on $\partial_{\mathbf{z}} f$ *on the optimal trajectory* — it is a property of the problem, not of the optimizer.

- **Noise** σ leaves $\partial_{\mathbf{z}} f$ on $\mathbf{z}^\star$ unchanged $\Rightarrow$ E unchanged; on the quadcopter $\hat{E} > 0$ for every σ we tried.
- **Trust region / regularizers** select *which* $\mathbf{z}^\star, \mathbf{u}^\star$ the iteration converges to — but not the sign of E along it.
- **More inner iterations** drive $\mathbf{u}_{\boldsymbol{\theta}}$ harder onto the *wrong* target (dropped-D adjoint); amplify error.

What is needed better matching algorithms with convergence theory *beyond* LQG: augmented-Hamiltonian / extended-MSA Li et al. 2017 and trust-region matching Blessing et al. 2025.

expansion is a problem property, not an optimizer one — better matching algorithms are needed

Implicit Controls: The Lean Idea, Jacobian-Free

When the feedback is implicit if L is non-quadratic or f is *not* control-affine, $\mathbf{u}^\star = \arg\min_{\mathbf{u}} \mathcal{H}$ has no closed form — $\mathbf{u}$ is pinned only by $\partial_{\mathbf{u}}\mathcal{H} = 0$.

Jacobian-Free Backpropagation Gelphman et al. 2025 keep the value net $\Phi_{\boldsymbol{\theta}}$; make $\mathbf{u}_{\boldsymbol{\theta}}$ the fixed point $\mathbf{u} = T(\mathbf{u})$ of Hamiltonian gradient ascent, and train end-to-end while *dropping* the inverse-Jacobian $(I - \partial_{\mathbf{u}} T)^{-1} \to I$ — still a descent direction under contractivity (Thm 1).

The connection

- same drop-the-Jacobian idea as the **lean adjoint** (which drops the policy Jacobian D) — here it buys *implicit* controls
- $\mathcal{O}(m^2)$ not $\mathcal{O}(m^3)$; reaches 80-D state / 40-D control where plain AD runs out of memory
- *deterministic* so far — the stochastic / matching extension is open

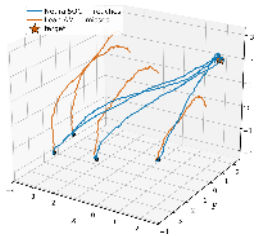
the Jacobian-free idea also unlocks implicit, high-dim controls — stochastic is next

Σ: Neural SOC — where matching wins, where it doesn't

- **NeuralSOC**: learn $\Phi_{\boldsymbol{\theta}}$; HJB residual + PMP sampling; DTO adjoint
- **Adjoint Matching**: drop $D \to$ **lean** adjoint; **weight-free** regression; gradient = policy gradient at $\mathbf{u}^\star$
- **Limits**: *empirically*, $E = \int_0^T \partial_{\mathbf{z}} f \, ds \leq 0$ is *sufficient* (not necessary) to contract; the expansive quadcopter ($E > 0$) diverges

What we learned

- *in our experiments*, matching works when f is contractive ($E \leq 0$) — the regime it was built for (incl. gen-AI fine-tuning)
- lean AM = continuous-time MSA Domingo-Enrich and Han 2026 (deterministic limit; deep-learning ext. Li et al. 2017)



better algorithms are needed to get the advantages of both

Joint with K. Keegan, R. Leburu, X. Li,

L. Nurbekyan, D. Onken, D. Verma, N. Yang

lruthotto@emory.edu

Questions?

References

- Blessing, D. et al. (2025). “Trust Region Constrained Measure Transport in Path Space for Stochastic Optimal Control and Inference”. In: *Advances in Neural Information Processing Systems (NeurIPS)*.
- Domingo-Enrich, C. and J. Han (2026). “Adjoint Matching through the Lens of the Stochastic Maximum Principle in Optimal Control”. In: *arXiv preprint arXiv:2604.08580*.
- Domingo-Enrich, C. et al. (2024). “Stochastic Optimal Control Matching”. In: *Advances in Neural Information Processing Systems 37 (NeurIPS)*.
- Domingo-Enrich, C. et al. (2025). “Adjoint Matching: Fine-tuning Flow and Diffusion Generative Models with Memoryless Stochastic Optimal Control”. In: *International Conference on Learning Representations (ICLR)*. Spotlight.
- Gelphman, E. et al. (2025). *End-to-End Training of High-Dimensional Optimal Control with Implicit Hamiltonians via Jacobian-Free Backpropagation*. arXiv: 2510.00359 [math.OC]. URL: https://arxiv.org/abs/2510.00359.
- Han, J. et al. (Aug. 2018). “Solving high-dimensional partial differential equations using deep learning”. In: *Proceedings of the National Academy of Sciences*, pp. 1 –6. DOI: 10.1073/pnas.1718942115.
- Kidger, P. (2022). “On Neural Differential Equations”. PhD thesis. University of Oxford.
- Leburu, R. et al. (2026). “Differentiating through Stochastic Differential Equations: A Primer”. arXiv:2601.08594.
- Li, Q. et al. (2017). “Maximum principle based algorithms for deep learning”. In: *The Journal of Machine Learning Research* 18.1, pp. 5998–6026.
- Li, X. et al. (2024). “A Neural Network Approach for Stochastic Optimal Control”. In: *SIAM SISC*, arXiv:2209.13104.
- Onken, D. et al. (2022). “A neural network approach for high-dimensional optimal control applied to multiagent path finding”. In: *IEEE TCST* 31.1, pp. 235–251.

References (cont.)


Raissi, M. (2018). “Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations”. In: *arXiv preprint arXiv:1804.07010*.


Sirignano, J. and K. Spiliopoulos (Dec. 2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* 375, pp. 1339–1364.