Stochastic Optimal Control Matching

Carles Domingo-Enrich, Jiequn Han, Brandon Amos, Joan Bruna, Ricky T. Q. Chen

Thomas Mousseau

August 27, 2025

Overview

1. Setup and Preliminaries
2. Stochastic Optimal Control Matching
3. Experiments and results
4. Conclusion

Evolution of Generative Models

2020 **DDPM:** Denoising Diffusion Probabilistic Models interpret generation as reversing a discrete noise-adding process, learning to denoise at each step. They produced high-quality samples but required thousands of slow sampling steps.

2021 **Score-based Models:** Score-based generative models extended diffusion to continuous-time SDEs, learning the score function $(\nabla_x \log p_t(x))$ to reverse a stochastic diffusion process. This unified diffusion with stochastic control, allowed probability flow ODEs, and sped up sampling.

2023 **Flow Matching:** Flow matching views generation as learning a deterministic ODE vector field that directly transports a simple distribution (e.g., Gaussian) to data. This removed stochasticity and significantly improved efficiency compared to diffusion/score methods.

SOC as the Foundation of Generative Models

The Core Challenge: Unnormalized Densities

Generative models must sample from complex distributions $p_{\text{data}}(x) = \frac{1}{Z}\tilde{p}_{\text{data}}(x)$ where the normalization constant $Z = \int \tilde{p}_{\text{data}}(x)dx$ is intractable to compute. This intractability arises from the curse of dimensionality when integrating over high-dimensional spaces.

SOC Connection

Key Insight:
Transform tractable distributions (Gaussian) to complex target distributions through optimal control policies.

This bridges the gap between:

- Simple sampling (easy)
- Complex data distributions (hard)

Modern Implementations

Diffusion Models:
$u_t = -\frac{1}{2}\nabla_x \log p_t(x)$ (denoising)

Score-based Models:
$u_t = \nabla_x \log p_t(x)$ (score function)

Flow Matching:
$u_t = \frac{x_1 - x_0}{T - t}$ (deterministic flow)
All learn optimal control policies to transport distributions!

SOC as the Foundation of Generative Models

The Core Challenge: Unnormalized Densities

Generative models must sample from complex distributions $p_{\text{data}}(x) = \frac{1}{Z}\tilde{p}_{\text{data}}(x)$ where the normalization constant $Z = \int \tilde{p}_{\text{data}}(x)dx$ is intractable to compute. This intractability arises from the curse of dimensionality when integrating over high-dimensional spaces.

SOC Connection

Key Insight:
Transform tractable distributions (Gaussian) to complex target distributions through optimal control policies.

This bridges the gap between:

- Simple sampling (easy)
- Complex data distributions (hard)

Modern Implementations

Diffusion Models:
$u_t = -\frac{1}{2}\nabla_x \log p_t(x)$ (denoising)

Score-based Models:
$u_t = \nabla_x \log p_t(x)$ (score function)

Flow Matching:
$u_t = \frac{x_1 - x_0}{T - t}$ (deterministic flow)
All learn optimal control policies to transport distributions!

Unnormalized Densities: The fundamental challenge in generative modeling is sampling from distributions $p(x) = \frac{1}{Z}e^{-E(x)}$ where Z is unknown. SOC provides the mathematical framework to construct sampling procedures.

Historical Context: From Langevin dynamics to modern diffusion models, all major breakthroughs in generative modeling can be understood through the lens of stochastic optimal control theory.

What is a Stochastic Control Problem?

Control-Affine Stochastic Differential Equation

The general form of a controlled stochastic process:

$$dX_t^u = \big(b(X_t^u, t) + \sigma(t)u(X_t^u, t)\big)dt + \sqrt{\lambda}\sigma(t)dB_t \tag{1}$$

State Process: $X_t^u \in \mathbb{R}^d$ (system state under control u at time t)

Drift Term: $b(X_t^u, t) \in \mathbb{R}^d$ (natural evolution of the system)

Control Term: $\sigma(t)u(X_t^u, t) \in \mathbb{R}^d$ (how control influences the system)

Diffusion Coefficient: $\sigma(t) \in \mathbb{R}^{d \times d}$ (volatility matrix)

Noise Process: $B_t \in \mathbb{R}^d$ (Brownian motion, external randomness)

Noise Intensity: $\lambda > 0$ (controls the strength of stochastic perturbations)

What is a Stochastic Control Problem?

Control-Affine Stochastic Differential Equation

The general form of a controlled stochastic process:

$$dX_t^u = \left(b(X_t^u, t) + \sigma(t)u(X_t^u, t)\right)dt + \sqrt{\lambda}\sigma(t)dB_t \quad (1)$$

State Process: $X_t^u \in \mathbb{R}^d$ (system state under control u at time t)

Drift Term: $b(X_t^u, t) \in \mathbb{R}^d$ (natural evolution of the system)

Control Term: $\sigma(t)u(X_t^u, t) \in \mathbb{R}^d$ (how control influences the system)

Diffusion Coefficient: $\sigma(t) \in \mathbb{R}^{d\times d}$ (volatility matrix)

Noise Process: $B_t \in \mathbb{R}^d$ (Brownian motion, external randomness)

Noise Intensity: $\lambda > 0$ (controls the strength of stochastic perturbations)

Control-Affine Structure: The ”control-affine” property means the control u enters linearly (affinely) in the drift term. This is the most general practical form for controlled SDEs, encompassing most applications in finance, robotics, and machine learning.

Je dois aussi expliquer l’importance que le steering input ainsi que le noise sont tous les 2 multipliés par sigma(t) ce qui signifie que ...

The Optimal Control Objective

Cost Function to Minimize

Find the optimal control policy $u^* \in \mathcal{U}$ that minimizes:

$$\min_{u \in \mathcal{U}} \mathbb{E}\left[\int_0^T \left(\frac{1}{2}\|u(X_t^u, t)\|^2 + f(X_t^u, t)\right) dt + g(X_T^u)\right] \tag{2}$$

Control Effort: $\frac{1}{2}\|u(X_t^u, t)\|^2$ (penalizes large control actions)

Running Cost: $f(X_t^u, t)$ (ongoing cost during the process evolution)

Terminal Cost: $g(X_T^u)$ (final cost based on end state at time T)

Control Space: $\mathcal{U}$ (set of admissible control policies)

2025-08-27

Stochastic Optimal Control Matching

└─Setup and Preliminaries

└─The Optimal Control Objective

The Optimal Control Objective

Cost Function to Minimize

Find the optimal control policy $u^* \in \mathcal{U}$ that minimizes:

$$\min_{u \in \mathcal{U}} \mathbb{E}\left[\int_0^T \left(\frac{1}{2}\|u(X_t^u, t)\|^2 + f(X_t^u, t)\right) dt + g(X_T^u)\right] \tag{2}$$

Control Effort: $\frac{1}{2}\|u(X_t^u, t)\|^2$ (penalizes large control actions)

Running Cost: $f(X_t^u, t)$ (ongoing cost during the process evolution)

Terminal Cost: $g(X_T^u)$ (final cost based on end state at time T)

Control Space: $\mathcal{U}$ (set of admissible control policies)

SOC in Practice

Aspect	Steering Actuator	Diffusion Models	Flow Models
State X_t	Vehicle position, velocity	Noisy data sample	Clean data sample
Control u_t	Steering angle, throttle	Denoising direction	Flow velocity field
Dynamics Source	Newtonian mechanics (vehicle dynamics)	Forward noising process	No natural drift (learnable)
Drift $b(X_t, t)$	Kinematic equations	Predetermined schedule	$b = 0$ (control learns drift)
Control Goal	Reach target safely	Reverse noise process	Transport distributions
Noise $\sqrt{\lambda}\sigma dB_t$	Road disturbances	Brownian motion	Optional stochasticity

SOC in Practice

Aspect	Steering Actuator	Diffusion Models	Flow Models
State X_t	Vehicle position, velocity	Noisy data sample	Clean data sample
Control u_t	Steering angle, throttle	Denoising direction	Flow velocity field
Dynamics Source	Newtonian mechanics (vehicle dynamics)	Forward noising process	No natural drift (learnable)
Drift $b(X_t, t)$	Kinematic equations	Predetermined schedule	$b = 0$ (control learns drift)
Control Goal	Reach target safely	Reverse noise process	Transport distributions
Noise $\sqrt{\lambda}\sigma dB_t$	Road disturbances	Brownian motion	Optional stochasticity

Steering Actuator: The dynamics come from well-established physics - Newton's laws, kinematics, and vehicle dynamics. The control optimizes safety and efficiency.

Diffusion Models: The drift is determined by the forward noising process (e.g., $\beta(t)$), and control learns to reverse this predetermined corruption.

Flow Models: No natural drift exists - the control u_t directly becomes the drift term, learning the entire velocity field that transports distributions.

Applications of SOC

Key ML Applications of SOC

- **Reward fine-tuning of diffusion and flow models:** Optimizing generation quality using reward signals
- **Conditional sampling on diffusion and flow models:** Steering generation towards specific conditions or constraints
- **Sampling from unnormalized densities:** Efficiently drawing samples from complex, intractable distributions
- **Importance sampling of rare events in SDEs:** Computing probabilities of low-probability but critical events

Key Insight

The prevalence of SOC formulations in modern ML motivates the need for more efficient and stable solving methods.

Applications of SOC

Key ML Applications of SOC

- **Reward fine-tuning of diffusion and flow models:** Optimizing generation quality using reward signals
- **Conditional sampling on diffusion and flow models:** Steering generation towards specific conditions or constraints
- **Sampling from unnormalized densities:** Efficiently drawing samples from complex, intractable distributions
- **Importance sampling of rare events in SDEs:** Computing probabilities of low-probability but critical events

Key Insight

The prevalence of SOC formulations in modern ML motivates the need for more efficient and stable solving methods.

Je dois etre capable de vraiment bien expliquer les applications du SOC dans le ML moderne pour pouvoir demontrer l'importance d'avoir pris ce papier

Solve SOC Problems in Low Dimension

Classical Approach: Hamilton-Jacobi-Bellman PDE

In small state spaces (1D, 2D, sometimes 3D), discretize the state space and solve the HJB PDE directly:

$$\frac{\partial V}{\partial t} + \min_u \left[\frac{1}{2}\|u\|^2 + f(x,t) + \nabla V \cdot (b + \sigma u) + \frac{1}{2}\lambda \mathrm{tr}(\sigma^T \nabla^2 V \sigma) \right] = 0 \tag{3}$$

What You Get

- **Value function:** $V(x,t)$
- **Optimal control:** $u^*(x,t) = -\sigma(t)^T \nabla V(x,t)$
- **Global optimum** guaranteed
- **Theoretical guarantees** (convergence, stability)

The Curse of Dimensionality

- **Grid size:** $\mathcal{O}(N^d)$ where d is dimension
- **Memory:** Exponential growth with d
- **Computation:** Infeasible for $d > 3$
- **Impossible** to discretize for $d \gg 1$

Solve SOC Problems in Low Dimension

Classical Approach: Hamilton-Jacobi-Bellman PDE

In small state spaces (1D, 2D, sometimes 3D), discretize the state space and solve the HJB PDE directly:

$$\frac{\partial V}{\partial t} + \min_u \left[\frac{1}{2}\|u\|^2 + f(x,t) + \nabla V \cdot (b + \sigma u) + \frac{1}{2}\lambda \mathrm{tr}(\sigma^T \nabla^2 V \sigma)\right] = 0 \tag{3}$$

What You Get

- **Value function:** $V(x,t)$
- **Optimal control:** $u^*(x,t) = -\sigma(t)^T \nabla V(x,t)$
- **Global optimum** guaranteed
- **Theoretical guarantees** (convergence, stability)

The Curse of Dimensionality

- **Grid size:** $\mathcal{O}(N^d)$ where d is dimension
- **Memory:** Exponential growth with d
- **Computation:** Infeasible for $d > 3$
- **Impossible** to discretize for $d \gg 1$

Works perfectly in low dimensions, but state space grows exponentially with dimension.
Classical HJB: In low dimensions, you can discretize the entire state space on a grid and solve the HJB PDE using finite difference methods. This gives you the exact solution but becomes computationally impossible as dimension increases.

Curse of Dimensionality: For a d-dimensional problem with N grid points per dimension, you need N^d total grid points. Even modest problems ($d = 10$, $N = 100$) require $100^{10} = 10^{20}$ grid points.

Solve SOC Problems in High Dimension

Modern Approach: Adjoint Methods

Cannot discretize high-dimensional spaces, so use **adjoint methods** to optimize control directly:

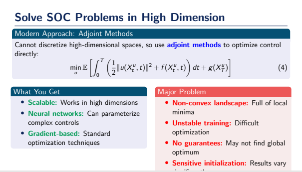
$$\min_u \mathbb{E}\left[\int_0^T \left(\frac{1}{2}\|u(X_t^u, t)\|^2 + f(X_t^u, t)\right) dt + g(X_T^u)\right] \tag{4}$$

What You Get

- **Scalable:** Works in high dimensions
- **Neural networks:** Can parameterize complex controls
- **Gradient-based:** Standard optimization techniques

Major Problem

- **Non-convex landscape:** Full of local minima
- **Unstable training:** Difficult optimization
- **No guarantees:** May not find global optimum
- **Sensitive initialization:** Results vary significantly



Highly non-convex functional landscape makes optimization extremely challenging. Traditional SOC methods struggle with local minima and unstable training dynamics.

Adjoint Methods: These methods solve the forward SDE and then use the adjoint equation to compute gradients efficiently. This allows gradient-based optimization of the control parameters without discretizing the state space.

Non-convex Optimization: The major challenge is that the resulting optimization landscape is highly non-convex, leading to difficult optimization with many local minima and unstable training dynamics.

Similar Trend in Generative Modeling

The Same Pattern: Non-convex $\longrightarrow$ Least-Squares

Generative modeling experienced the same transition from non-convex to convex optimization

Continuous Normalizing Flows

Method: Learn invertible transformations

$$\frac{dx}{dt} = f_\theta(x, t) \tag{5}$$

Problem:

- Use **adjoint methods** for gradients
- **Non-convex** optimization landscape
- Difficult training, unstable dynamics

Denoising Diffusion Models

Method: Learn to reverse noise process

$$\mathbb{E}[\|\epsilon_\theta(x_t, t) - \epsilon\|^2] \tag{6}$$

Advantage:

- Use **least-squares loss**
- **Convex** functional landscape
- Stable, reliable optimization

Similar Trend in Generative Modeling

The Same Pattern: Non-convex → Least-Squares

Generative modeling experienced the same transition from non-convex to convex optimization

Continuous Normalizing Flows

Method: Learn invertible transformations

$$\frac{dx}{dt} = f_\theta(x, t) \tag{5}$$

Problem:

- Use **adjoint methods** for gradients
- **Non-convex** optimization landscape
- Difficult training, unstable dynamics

Denoising Diffusion Models

Method: Learn to reverse noise process

$$\mathbb{E}[\|\epsilon_\theta(x_t, t) - \epsilon\|^2] \tag{6}$$

Advantage:

- Use **least-squares loss**
- **Convex** functional landscape
- Stable, reliable optimization

Insight: Apply the same principle to SOC! Replace non-convex adjoint methods with **least-squares matching**, creating a **convex optimization landscape** for stochastic optimal control.

Historical Parallel: Continuous Normalizing Flows (CNFs) suffered from the same optimization challenges as traditional SOC - they used adjoint methods and had non-convex landscapes. DDPMs revolutionized generative modeling by reformulating the problem as least-squares regression.

SOCM's Contribution: SOCM brings the same insight to stochastic optimal control, replacing direct optimization with least-squares matching to achieve stable, convex optimization.

SOCM in Context: Optimization Landscapes

Task	Non-convex	Least Squares
Generative Modeling	Maximum Likelihood CNFs	Diffusion models and Flow Matching
Stochastic Optimal Control	Adjoint Methods	**Stochastic Optimal Control Matching**

SOCM Framework: From Classical SOC to SOCM

Dynamics (Same for Both)

$$dX_t^v = (b(X_t^v, t) + \sigma(t)v(X_t^v, t))dt + \sqrt{\lambda}\sigma(t)dB_t, \text{ with } X_0^v \sim p_0 \tag{7}$$

Original SOC Cost Function

$$\min_{u \in \mathcal{U}} \mathcal{L}_{SOC} := \mathbb{E}\left[\int_0^T \left(\frac{1}{2}\|u(X_t^u, t)\|^2 + f(X_t^u, t)\right) dt + g(X_T^u)\right] \tag{8}$$

SOCM Cost Function

$$\min_{u,M} \mathcal{L}_{SOCM}(u, M) := \mathbb{E}\left[\frac{1}{T}\int_0^T \|u(X_t^v, t) - w(t, v, X^v, B, M_t)\|^2 dt \times \alpha(v, X^v, B)\right] \tag{9}$$

Key Point: Same underlying stochastic dynamics, but SOCM uses a least-squares matching approach instead of direct minimization of the original cost.

SOCM Parameters Definition

$$\min_{u,M} \mathcal{L}_{SOCM}(u, M) := \mathbb{E}\left[\frac{1}{T}\int_0^T \|u(X_t^v, t) - w(t, v, X^v, B, M_t)\|^2 dt \times \alpha(v, X^v, B)\right]$$

Where:

$u : \mathbb{R}^d \times [0,1] \to \mathbb{R}^d$ is the **control** (policy being learned)

v is a **fixed arbitrary control**, X^v is the solution of the SDE with control v

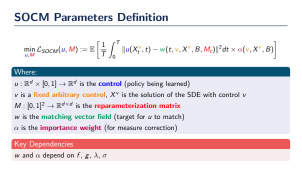
$M : [0,1]^2 \to \mathbb{R}^{d\times d}$ is the **reparameterization matrix**

w is the **matching vector field** (target for u to match)

α is the **importance weight** (for measure correction)

Key Dependencies

w and α depend on f, g, λ, σ



Control u: This is what we're trying to learn - the optimal policy that minimizes our cost function.

Arbitrary Control v: Used to generate sample trajectories for training. The choice of v affects the efficiency but not the correctness of SOCM.

Reparameterization Matrix M: Jointly optimized with u to reduce variance in the gradient estimation through path-wise reparameterization.

Matching Vector Field (1/2)

Reparameterization Function $w(t, v, X^v, B, M_t)$

The **matching vector field** computed via path-wise reparameterization:

$$
\begin{aligned}
w(t, v, X^v, B, M_t) = \sigma(t)^\top \Bigg(& - \int_t^T M_t(s) \nabla_x f(X_s^v, s)\, ds \; - \; M_t(T) \nabla g(X_T^v) \\
& + \int_t^T \big(M_t(s) \nabla_x b(X_s^v, s) - \partial_s M_t(s)\big)(\sigma^{-1}(s))^\top v(X_s^v, s)\, ds \\
& + \sqrt{\lambda} \int_t^T \big(M_t(s) \nabla_x b(X_s^v, s) - \partial_s M_t(s)\big)(\sigma^{-1}(s))^\top dB_s \Bigg)
\end{aligned} \tag{10}
$$

Where $M_t(s)$ is the **reparameterization matrix** (learned jointly with u)

Matching Vector Field (2/2) - Path Integral I

Step 1 — Path-integral form of u^* (Kappen 2005)

What to Show

We want to eliminate the value function V from the optimal control formula and express u^* directly in terms of path integrals.

The value function is defined as the infimum over all controls:

$$V(x,t) = \inf_u J(u;x,t) = \inf_u \mathbb{E}\left[\int_t^T \left(\frac{1}{2}\|u(X_s^u,s)\|^2 + f(X_s^u,s)\right) ds + g(X_T^u) \,\middle|\, X_t^u = x\right]$$

Matching Vector Field (2/2) - Path Integral II

Assume $\sigma(t)$ is invertible and enters both drift (via σu) and diffusion, and control cost is quadratic. This enables the Cole-Hopf linearization.
The **infinitesimal generator** L of the uncontrolled diffusion
$dX_t = b(x,t)\,dt + \sqrt{\lambda}\sigma(t)\,dB_t$ is:

$$L := \frac{\lambda}{2}\sum_{i,j=1}^{d}(\sigma\sigma^\top)_{ij}(t)\frac{\partial^2}{\partial x_i \partial x_j} + \sum_{i=1}^{d} b_i(x,t)\frac{\partial}{\partial x_i}$$

The Hamilton-Jacobi-Bellman equation becomes (minimize pointwise in (x,t)):

$$\frac{\partial V}{\partial t} + LV + \inf_u\left[\frac{1}{2}\|u\|^2 + \langle \sigma^\top \nabla V, u\rangle + f\right] = 0$$

Matching Vector Field (2/2) - Path Integral III

To find the infimum, we differentiate the quadratic expression with respect to u (complete the square):

$$\frac{\partial}{\partial u}\left[\frac{1}{2}\|u\|^2 + \langle \sigma^\top \nabla V, u\rangle\right] = u + \sigma^\top \nabla V = 0$$

This gives us the **optimal control**:

$$u^*(x,t) = -\sigma^\top(t)\nabla V(x,t)$$

Substituting back into the HJB equation:

$$\frac{\partial V}{\partial t} + LV - \frac{1}{2}\|\sigma^\top \nabla V\|^2 + f = 0, \quad V(x,T) = g(x)$$

Matching Vector Field (2/2) - Path Integral IV

Reminder (Linear Case)

If $\frac{\partial w}{\partial t} + Lw + q = 0$ with $w(x, T) = h(x)$, then Feynman-Kac gives:

$$w(x,t) = \mathbb{E}\left[h(X_T) - \int_t^T q(X_s, s)ds \,\middle|\, X_t = x \right]$$

The HJB for V is nonlinear; we apply Feynman-Kac to the linear PDE for ϕ, then set $V = -\lambda \log \phi$.

Cole-Hopf transformation: $V(x,t) = -\lambda \log \phi(x,t)$
Linearized PDE for ϕ:

$$\frac{\partial \phi}{\partial t} + L\phi + \frac{1}{\lambda} f\phi = 0, \quad \phi(x, T) = e^{-g(x)/\lambda}$$

Matching Vector Field (2/2) - Path Integral V

Feynman-Kac applied to ϕ:

$$\phi(x,t) = \mathbb{E}_0\left[\exp\left(-\frac{1}{\lambda}\int_t^T f(X_s,s)ds - \frac{1}{\lambda}g(X_T)\right)\middle| X_t = x\right] \text{ (uncontrolled SDE)}$$

Converting back: $V(x,t) = -\lambda \log \phi(x,t)$

Set $Z(x,t) := \phi(x,t)$, so the **partition function** is:

$$Z(x,t) = \mathbb{E}_0\left[\exp\left(-\frac{1}{\lambda}\int_t^T f(X_s,s)ds - \frac{1}{\lambda}g(X_T)\right)\middle| X_t = x\right]$$

Then the optimal control becomes:

Matching Vector Field (2/2) - Path Integral VI

Path-integral Optimal Control

$$u^*(x,t) = \lambda\,\sigma^\top(t)\,\nabla_x \log Z(x,t)$$

$$u^*(x,t) = \lambda\,\sigma^\top(t)\,\nabla_x \log \mathbb{E}_0\left[\exp\left(-\frac{1}{\lambda}\int_t^T f(X_s,s)ds - \frac{1}{\lambda}g(X_T)\right)\,\middle|\, X_t = x\right]$$

∇_x is the gradient w.r.t. the initial condition $X_t = x$, not a terminal-state score.

Matching Vector Field (2/2) - Path Integral VII

Why This Matters

Key Achievement: We eliminate V to avoid solving a nonlinear PDE and reframe the control as a **score function** $\nabla_x \log Z(x,t)$.

We avoid solving a nonlinear PDE and instead target a score that admits a pathwise estimator (next slide). This sets up a score we can approximate with a **low-variance pathwise estimator** in Step 2, which will become the matching vector field w. This is the foundation for SOCM's **least-squares matching** approach.

Matching Vector Field (2/3) — Optimal Control I

Step 2 — Rendering u^* Trackable using Pathwise reparameterization (under the uncontrolled law)

The Hypothetical Scenario

What if we could compute the optimal control $u^*(X_t, t)$ during training? We would simply minimize the L_2 distance between our learned control u and the true optimum u^*.

Matching Vector Field (2/3) — Optimal Control II

Ideal Loss Function

$$\tilde{\mathcal{L}}(u) = \mathbb{E}\left[\frac{1}{T}\int_0^T \|u(X_t,t) - u^*(X_t,t)\|^2 dt \exp\left(-\lambda^{-1}\int_0^T f(X_t,t)dt - \lambda^{-1}g(X_T)\right)\right] \tag{11}$$

Let's expand the squared term $\|u(X_t,t) - u^*(X_t,t)\|^2$:

$$\begin{aligned}\tilde{\mathcal{L}}(u) = \mathbb{E}\Bigg[&\frac{1}{T}\int_0^T \left(\|u(X_t,t)\|^2 - 2\langle u(X_t,t), u^*(X_t,t)\rangle + \|u^*(X_t,t)\|^2\right) dt \\ &\times \exp\left(-\lambda^{-1}\int_0^T f(X_t,t)dt - \lambda^{-1}g(X_T)\right)\Bigg]\end{aligned} \tag{12}$$

Matching Vector Field (2/3) — Optimal Control III

The Fundamental Challenge

We cannot compute $u^*(X_t, t)$ directly! This is exactly why we need SOCM's matching approach - to find a computable target w that serves as a proxy for u^*.

Why Focus on the Cross Term?

We concentrate on transforming the cross term $\langle u(X_t, t), u^*(X_t, t)\rangle$ because:

First term $\|u(X_t, t)\|^2$: Straightforward to compute - depends only on our learned control u

Third term $\|u^*(X_t, t)\|^2$: Constant with respect to u - doesn't affect the optimization

Cross term: The **only term** that requires the path-integral formulation since it involves both u and u^*

Matching Vector Field (2/3) — Optimal Control IV

Using the path-integral representation from Step 1, we can transform the problematic cross term:

$$
\begin{aligned}
&\mathbb{E}\left[\frac{1}{T}\int_0^T \langle u(X_t,t), u^*(X_t,t)\rangle dt \exp\left(-\lambda^{-1}\int_0^T f(X_t,t)dt - \lambda^{-1}g(X_T)\right)\right] \\
&= \lambda\mathbb{E}\left[\frac{1}{T}\int_0^T \left\langle u(X_t,t), \sigma(t)^T\nabla_x\mathbb{E}\left[\exp\left(-\lambda^{-1}\int_t^T f(X_s,s)ds - \lambda^{-1}g(X_T)\right)\middle| X_t = x\right]\right\rangle\right. \\
&\qquad\qquad \left.\times \exp\left(-\lambda^{-1}\int_0^t f(X_s,s)ds\right) dt\right]
\end{aligned}
\tag{13}
$$

Matching Vector Field (2/3) — Optimal Control V

Next Challenge

The transformed cross term still contains an intractable expectation. This is where path-wise reparameterization becomes essential to make this computable since Monte-Carlo "score-function" (REINFORCE) estimators for this expectation have prohibitively high variance.

The Challenge: Intractable Conditional Expectation

From Step 2, we need to make this computable for training:

$$\nabla_x \mathbb{E}\left[\exp\left(-\lambda^{-1}\int_t^T f(X_s,s)ds - \lambda^{-1}g(X_T)\right)\middle| X_t = x\right]$$

Standard approach: Monte-Carlo score estimators $\rightarrow$ **prohibitively high variance**

Matching Vector Field (2/3) — Optimal Control (continued) I

This transformation comes from a general theorem for Fréchet-differentiable functionals $F(X)$:

Proposition 4: Path-wise Reparameterization

$$\nabla_x \mathbb{E}[\exp(-F(X))|X_0 = x] = \mathbb{E}\Bigg[\Big(-\nabla_z F(X+\psi)\Big|_{z=0} + \frac{1}{\lambda}\int_0^T (\nabla_z \psi \nabla_x b - \nabla_z \partial_s \psi)(\sigma^{-1})^T dB_s\Big) \times e^{-F(X)} \Big| X_0 = x \Bigg] \tag{14}$$

Matching Vector Field (2/3) — Optimal Control (continued) II

where $\psi(z, s)$ is a "path shift" function with $\psi(z, 0) = z$ and $\psi(0, s) = 0$.

SOCM's Specific Choice

SOCM chooses a **linear path shift**: $\psi_t(z, s) = M_t(s)^T z$ with $M_t(t) = I$
This gives: $\nabla_z \psi_t(0, s) = M_t(s)$ and $\nabla_z \partial_s \psi_t(0, s) = \partial_s M_t(s)$
The reparameterization matrix $M_t(s)$ becomes a learnable parameter for variance reduction!

Matching Vector Field (2/3) — Optimal Control (continued) III

Path-wise Reparameterization Solution

Transform the intractable gradient into a computable expectation:

$$
\begin{aligned}
&\nabla_x \mathbb{E}\left[\exp\left(-\lambda^{-1}\int_t^T f(X_s,s)ds - \lambda^{-1}g(X_T)\right)\middle| X_t = x\right] \\
&= \mathbb{E}\Bigg[\Bigg(-\lambda^{-1}\int_t^T M_t(s)\nabla_x f(X_s,s)ds - \lambda^{-1}M_t(T)\nabla g(X_T) \\
&\qquad + \lambda^{-1/2}\int_t^T (M_t(s)\nabla_x b(X_s,s) - \partial_s M_t(s))(\sigma^{-1})^T(s)dB_s\Bigg) \\
&\qquad \times \exp\left(-\lambda^{-1}\int_t^T f(X_s,s)ds - \lambda^{-1}g(X_T)\right)\middle| X_t = x\Bigg]
\end{aligned}
\tag{15}
$$

Matching Vector Field (3/3) - Girsanov Theorem I

Step 3 — Girsanov Theorem

Matching Vector Field (3/3) - Girsanov Theorem II

SOCM Loss in Construction

From Steps 1-2, we have the theoretical loss function:

$$\tilde{\mathcal{L}}(u) = \mathbb{E}\Bigg[\frac{1}{T}\int_0^T \Bigg\| u(X_t,t) + \sigma(t)^T \Bigg(\int_t^T M_t(s)\nabla_x f(X_s,s)ds + M_t(T)\nabla g(X_T) \\ - \lambda^{1/2}\int_t^T (M_t(s)\nabla_x b(X_s,s) - \partial_s M_t(s))(\sigma^{-1})^T(s)dB_s\Bigg) \\ \times \exp\left(-\lambda^{-1}\int_0^T f(X_t,t)dt - \lambda^{-1}g(X_T)\right) + K\Bigg] \tag{16}$$

Problem: This expectation is under the **uncontrolled** measure, but we need to sample using an **arbitrary control** v for practical training.

Matching Vector Field (3/3) - Girsanov Theorem (continued) I

Girsanov Measure Change

To transform from uncontrolled to controlled sampling, Girsanov theorem:
Changes the drift: $dX_t = bdt + \sqrt{\lambda}\sigma dB_t \to dX_t^v = (b + \sigma v)dt + \sqrt{\lambda}\sigma dB_t$
Introduces importance weight:

$$\alpha(v, X^v, B) = \exp\left(-\frac{1}{\sqrt{\lambda}}\int_0^T \langle v, dB_t\rangle - \frac{1}{2\lambda}\int_0^T \|v\|^2 dt\right)$$

Modifies the exponential weight: The original $\exp(-\lambda^{-1}\int f - \lambda^{-1}g)$ becomes part of the full importance weight.

Matching Vector Field (3/3) - Girsanov Theorem (continued) II

Final SOCM Loss

After Girsanov transformation, we get the practical SOCM loss:

$$\boxed{\mathcal{L}_{SOCM}(u, M) = \mathbb{E}\left[\frac{1}{T}\int_0^T \|u(X_t^v, t) - w(t, v, X^v, B, M_t)\|^2 dt \times \alpha(v, X^v, B)\right]} \tag{17}$$

where:

- w is the matching vector field (from the reparameterization terms)
- α is the complete importance weight including costs and control effort
- Trajectories X^v are generated using the practical control v

Matching Vector Field (3/3) - Girsanov Theorem (continued) III

Key Achievement

Girsanov bridges theory and practice: We can train on controlled trajectories X^v while the importance weight α ensures we're still optimizing the original SOC objective.
The extra drift shift from v generates both the v-dependent terms in w and the corrective importance weight α.

SOCM Algorithm

Algorithm 2 Stochastic Optimal Control Matching (SOCM)

Input: State cost $f(x,t)$, terminal cost $g(x)$, diffusion coeff. $\sigma(t)$, base drift $b(x,t)$, noise level λ, number of iterations N, batch size m, number of time steps K, initial control parameters θ_0, initial matrix parameters ω_0, loss $\mathcal{L}_{\mathrm{SOCM}}$ in (125)

for $n \in \{0, \dots, N-1\}$ **do**

  Simulate m trajectories of the process X^v controlled by $v = u_{\theta_n}$, e.g., using Euler-Maruyama updates

  Detach the m trajectories from the computational graph, so that gradients do not backpropagate

  Using the m trajectories, compute an m-sample Monte-Carlo approximation $\hat{\mathcal{L}}_{\mathrm{SOCM}}(u_{\theta_n}, M_{\omega_n})$ of the loss $\mathcal{L}_{\mathrm{SOCM}}(u_{\theta_n}, M_{\omega_n})$ in (125)

  Compute the gradients $\nabla_{(\theta,\omega)}\hat{\mathcal{L}}_{\mathrm{SOCM}}(u_{\theta_n}, M_{\omega_n})$ of $\hat{\mathcal{L}}_{\mathrm{SOCM}}(u_{\theta_n}, M_{\omega_n})$ at (θ_n, ω_n)

  Obtain θ_{n+1}, ω_{n+1} with via an Adam update on θ_n, ω_n, resp.

end

Output: Learned control u_{θ_N}

Figure: Stochastic Optimal Control Matching (SOCM) Algorithm

Ornstein-Uhlenbeck Process Benchmarks I

What's an Ornstein-Uhlenbeck (OU) Process?

A **mean-reverting** stochastic differential equation with linear drift and Gaussian noise:

$$dX_t = \theta(\mu - X_t)dt + \sigma dB_t \quad \text{(1D case)}$$

For multi-dimensional benchmarks: $dX_t = Axdt + \sigma dB_t$ with linear drift $b(x,t) = Ax$

Double-Well (Challenging Nonlinear Case)

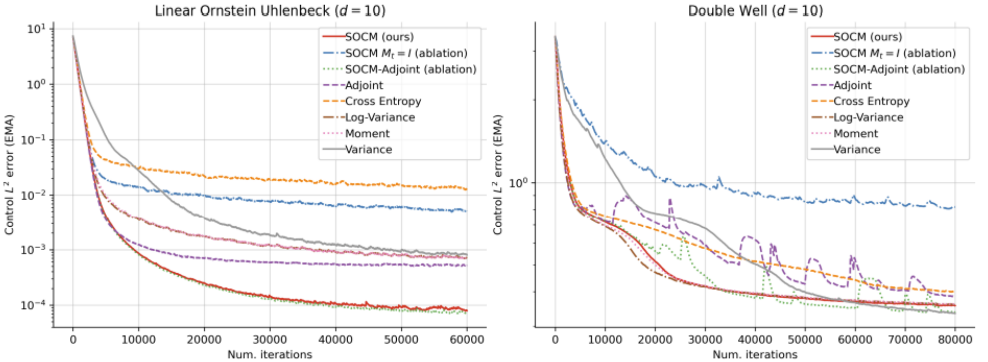
Nonlinear drift from quartic potential: $\Psi(x) = \sum_i \kappa_i (x_i^2 - 1)^2$
Multimodal terminal cost: $g(x)$ has 2^d modes (exponentially many!)
Highly challenging: Tests algorithm stability in complex, multimodal landscapes

Ornstein-Uhlenbeck Process Benchmarks II

Why OU-type Problems are Standard SOC Testbeds

- **Ground truth available:** Analytic u^* for LQR/linear cases $\rightarrow$ direct L_2 control error measurement
- **Controlled difficulty:** Can dial complexity via matrix scaling (A, P, Q parameters)
- **Stress test variance:** Reveals importance weight and gradient signal-to-noise issues
- **Multimodal landscapes:** Double-Well tests stability in challenging optimization scenarios

Experimental Results (1/2)



Experimental Results (2/2)

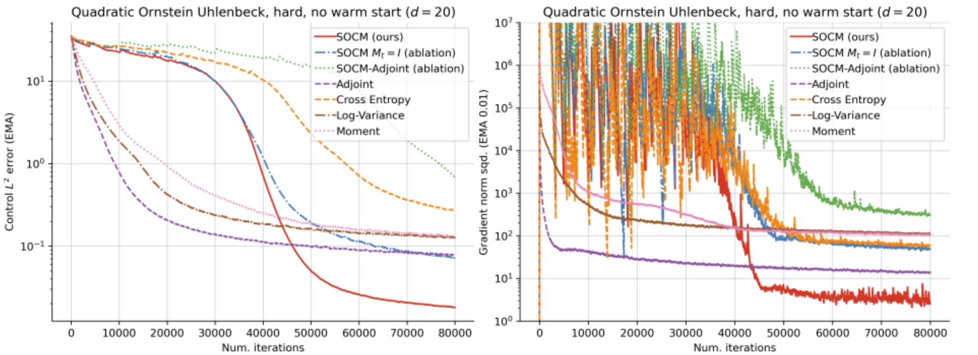


Figure 3 Plots of the L^2 error incurred by the learned control (*top*), and the norm squared of the gradient with respect to the parameters θ of the control (*bottom*), for the QUADRATIC ORNSTEIN UHLENBECK (HARD) setting and for each IDO loss. All the algorithms use a warm-started control (see Appendix D).

Experimental Results (2/2)

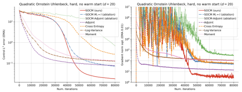


Figure 3 Plots of the L^2 error incurred by the learned control (*top*), and the norm squared of the gradient with respect to the parameters θ of the control (*bottom*), for the Quadratic Ornstein Uhlenbeck (hard) setting and for each IDO loss. All the algorithms use a warm-started control (see Appendix D).

At the end of training, SOCM obtains the lowest L2 error, improving over all existing methods by a factor of around ten. The two SOCM ablations come in second and third by a substantial difference, which underlines the importance of the path-wise reparameterization trick.

JE DOIS COMPRENDRE CE QUE EST UN ORNSTEIN UHLENBECK PROCESS

Conclusion

Paper's conclusion

peepee

Personal thoughts

poopoo

Conclusion

Paper's conclusion

peepee

Personal thoughts

poopoo

The main roadblock when we try to apply SOCM to more challenging problems is that the variance of the factor alpha(v, Xv, B) explodes when f and/or g are large, or when the dimension d is high. The control L2 error for the SOCM and cross-entropy losses remains high and fluctuates heavily due to the large variance of alpha The large variance of alpha is due to the mismatch between the probability measures induced by the learned control and the optimal control. Similar problems are encountered in out-of-distribution generalization for reinforcement learning, and some approaches may be carried over from that area (Munos et al., 2016).