

Flow Matching Corrected Posterior Estimation (FMCPE)

Pierre-Louis Ruhlmann

Pedro L. Rodrigues, Michael Arbel, Florence Forbes

EDF workshop on Metamodels

About me

- PhD student at Inria Grenoble (3rd year)
- Statify/Thoth team
- Research focus: Simulation-based inference
- Supervisors: Pedro L. Rodrigues, Michael Arbel, Florence Forbes

Inria



Statify

About me



Inria



Statify

Introduction

Statistical Inference

Given observations x and unknown parameters θ , we use **Bayes' rule**:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

- $p(\theta|x)$ - **Posterior**: what we want to compute
- $p(x|\theta)$ - **Likelihood**: probability of data given parameters
- $p(\theta)$ - **Prior**: initial beliefs about parameters
- $p(x)$ - **Evidence**: normalizing constant

Statistical Inference

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

Bayesian Inference

Input: Model likelihood $p(x|\theta)$

Methods: MCMC, Variational Inference

Output: Posterior $p(\theta|x)$

Simulation-Based Inference

Input: Simulator $S : \theta \mapsto x$

Methods: *(See next section)*

Output: Posterior $p(\theta|x)$

Key difference: SBI works when likelihood is intractable

Approximate Bayesian Computation (ABC)

Algorithm

Inputs: Simulator S , \mathbf{y}_{obs} , prior $p(\theta)$, distance ρ , threshold ϵ

For $i = 1, \dots, n$:

1. Sample $\theta_i \sim p(\theta)$
2. Simulate $\mathbf{x}_i = S(\theta_i)$
3. Keep θ_i if $\rho(\mathbf{x}_i, \mathbf{y}_{\text{obs}}) < \epsilon$

Python Code

```
# Inputs
S = simulator
y_obs = observed_data
prior = p_theta
rho = distance_function
epsilon = threshold

samples = []
for i in range(n):
    # 1. Sample from prior
    theta_i = prior.sample()
    # 2. Simulate
    x_i = S(theta_i)
    # 3. Keep if distance < epsilon
    if rho(x_i, y_obs) < epsilon:
        samples.append(theta_i)
```

Limitations:

- Poor scaling with dimension
- Requires summary statistics
- Not amortized

References: Beaumont et al. (2002), Sisson et al. (2018)

Neural Posterior Estimation (NPE)

Algorithm

Inputs: Simulator S , budget N_{train} , N_{epochs} , neural approximator q_ϕ , summary net h_ω

Generate training data:

Sample $\{(\theta_i, \mathbf{x}_i)\}_{i=1}^{N_{\text{train}}}$

For $e = 1, \dots, N_{\text{epochs}}$:

Minimize KL divergence:

$$\mathcal{L}(\phi, \omega) = -\mathbb{E}[\log q_\phi(\theta | h_\omega(\mathbf{x}))]$$

Python Code

```
# Inputs
S = simulator
N_train = budget
N_epochs = num_epochs
q_phi = neural_approximator
h_psi = summary_net

# Generate training data
thetas = prior.sample(N_train)
xs = S(thetas)

# Training loop
for epoch in range(N_epochs):
    # Minimize KL divergence
    loss = -q_phi.log_prob(thetas, h_psi(xs)).mean()
    loss.backward()
    optimizer.step()
```

Benefits: Amortized inference, no hand-crafted statistics needed

Limitations: Prone to overfitting, sensitive to distribution shift

References: Papamakarios & Murray (2016), Greenberg et al. (2019)

Challenges in SBI

- **Expensive simulations** - High computational cost
- **Exploit the simulator** - In certain cases where the simulator is not a black box, we can leverage information (e.g. its gradient) to improve inference.
- **Model misspecification** - Main focus of this work.

Model Misspecification

Types of Misspecification and Objective

Types of misspecification:

- **Simulator misspecification:** Model assumptions are wrong or incomplete (e.g., missing physics)
- **Prior misspecification:** Prior distribution does not reflect true parameter ranges
- **Observation noise:** Noise model in likelihood is misspecified

Approaches to address misspecification:

- **Robustness:** Ensemble methods or conservative posteriors
- **Detection:** Identify presence of misspecification
- **Correction:** Build new posterior accounting for discrepancy

→ In this work, we focus on correcting simulator misspecification.

Misspecification of the simulator

The data generating process $\mathbf{y} \sim p^*$

$$p^* \notin \{p(\cdot|\theta) \mid \theta \in \Theta\}$$

Example: Damped Pendulum

High-fidelity model (ground truth):

$$\mathbf{y}(t) = e^{-\alpha t} A \cos(\omega_0 t) + \epsilon$$

Parameters: $\theta = (\omega_0, A)$, damping $\alpha \sim \mathcal{U}(0, 1)$

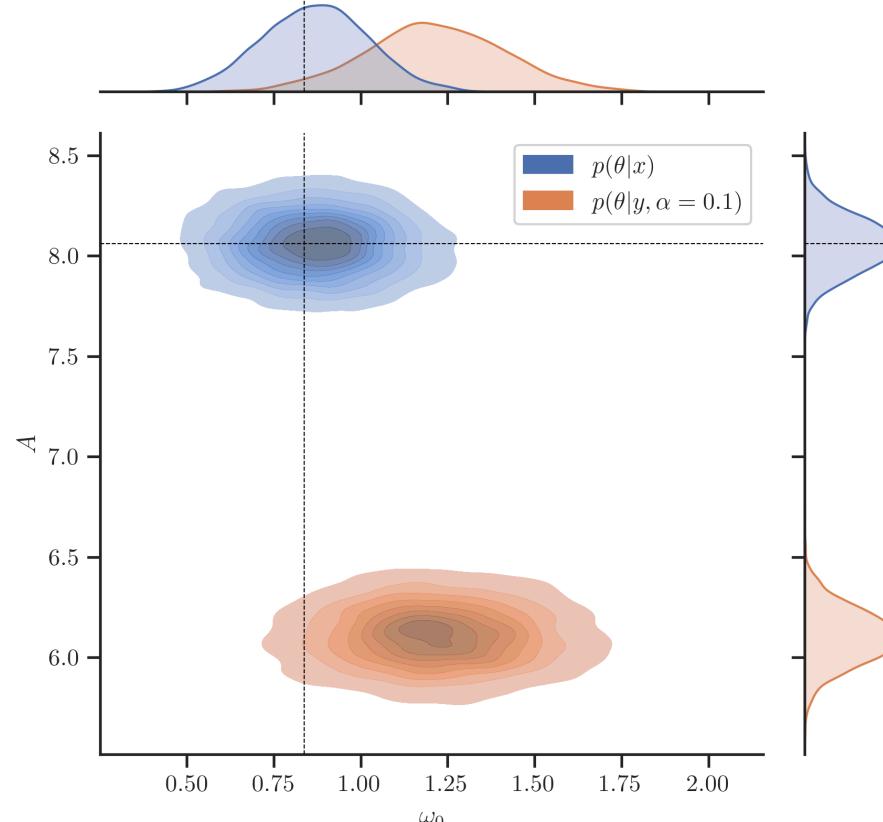
Low-fidelity simulator (misspecified):

$$\mathbf{x}(t) = A \cos(\omega_0 t) + \epsilon$$

Missing damping term! $\alpha = 0$

→ Even a small amount of misspecification can lead to highly biased posteriors !

NPE: $p(\theta|x)$ vs $p(\theta|y, \alpha = 0.1)$



Data setting

There are two main ways to approach model misspecification in SBI:

Case 1

1. Simulation data:

Access to simulated pairs $\{\theta_i, \mathbf{x}_i\}_{1:m}$

2. Observations:

Independent observations $\{\mathbf{y}_j\}_{1:n}$
(\mathbf{y} does not depend on θ)

3. Examples:

Radio propagation model (Turin), Recurrent Spiking Network
(neuroscience)

Case 2

1. Simulation data:

Access to simulated pairs $\{\theta_i, \mathbf{x}_i\}_{1:m}$

2. Calibration set:

Few paired observations $\{\theta_j, \mathbf{y}_j\}_{1:n}$
(*Multi-fidelity simulation, expensive ground truth*)

3. Examples:

CAMELS dataset (Astrophysics), Causal Chambers

→ In this work, we will focus on Case 2.

Existing Methods

MF-NPE

A first approach to correct the misspecification is to fine-tune the posterior estimator on the calibration set.

Assuming we have data from a low-fidelity simulator $(\theta, \mathbf{x}) \sim p(\theta)p(\mathbf{x} \mid \theta)$, we train a neural density estimator using:

$$\mathcal{L}_{NPE}(\phi) = \mathbb{E}_{\theta, \mathbf{x}} [-\log q_\phi(\theta | \mathbf{x})]$$

Then, using our small but precise calibration set $(\theta_i, \mathbf{y}_i) \sim p(\theta, \mathbf{y})$, we fine-tune the posterior estimator using the same loss:

$$\mathcal{L}_{NPE}(\phi) = \mathbb{E}_{\theta, \mathbf{y}} [-\log q_\phi(\theta | \mathbf{y})]$$

Reference: Krouglova et al. (2025). Multifidelity Simulation-based Inference for Computationally Expensive Simulators. *arXiv:2502.08416*.

ROPE: Transporting \mathbf{y} to \mathbf{x}

The ROPE method proposes to use optimal transport to match $p^*(\mathbf{y})$ and $p(\mathbf{x})$.

Input: Simulator $S(\theta, \epsilon)$, calibration data $\{(\theta^i, \mathbf{y}^i)\}_{i=1}^{N_c}$ and test set $\{\mathbf{y}^i\}_{i=1}^{N_o}$

Step 1: Train n.n. h_ω and density estimator q_ϕ

$$\mathcal{L}_{NPE}(\omega, \phi) = \mathbb{E}_{\theta, \mathbf{x}} [-\log q_\phi(\theta | h_\omega(\mathbf{x}))]$$

Step 2: Fine-tune h_ω on calibration data $\{(\theta^i, \mathbf{y}^i)\}$ (Initialize $g = h$)

$$\mathcal{L}(\varphi) = \sum_{i=1}^{N_c} \|g_\varphi(\mathbf{y}^i) - \mathbb{E}_\epsilon[h_\omega(S(\theta^i, \epsilon))]\|^2$$

Step 3: Solve OT problem between \mathbf{x}^j and \mathbf{y}^i with cost:

$$C(\mathbf{x}^j, \mathbf{y}^i) = \|h_\omega(\mathbf{x}^j) - g_\psi(\mathbf{y}^i)\|^2$$

Output: Corrected posterior $\tilde{p}(\theta | \mathbf{y}) = \sum_j^{N_s} N_o P_{ij}^* q_\phi(\theta | h_\omega(\mathbf{x}^j))$

Key takeaways of ROPE

- Two main components: The alignment of summary statistics using fine tuning and the transport between \mathbf{x} and \mathbf{y}
- The correspondence between θ_i and \mathbf{y}^i is only used to define the OT cost
- The "test" set is used for the computation of the optimal transport. The method can't be applied to new data
- The hypothesis $\mathbf{y} \perp \theta \mid \mathbf{x}$ is required to compute the posterior

Our Approach

Flow Matching Corrected Posterior Estimation

Main Idea: Learn to Correct the Posterior

The Challenge:

- We have a posterior $q_\phi(\theta|\mathbf{x})$ trained on cheap simulations \mathbf{x}
- We have a small **calibration set** $\mathcal{D}_{\text{cal}} = (\theta_i, \mathbf{y}_i)$ from the true process
- Goal: Estimate $p(\theta|\mathbf{y})$ by leveraging the calibration set and $q_\phi(\theta|\mathbf{x})$

Two step approach:

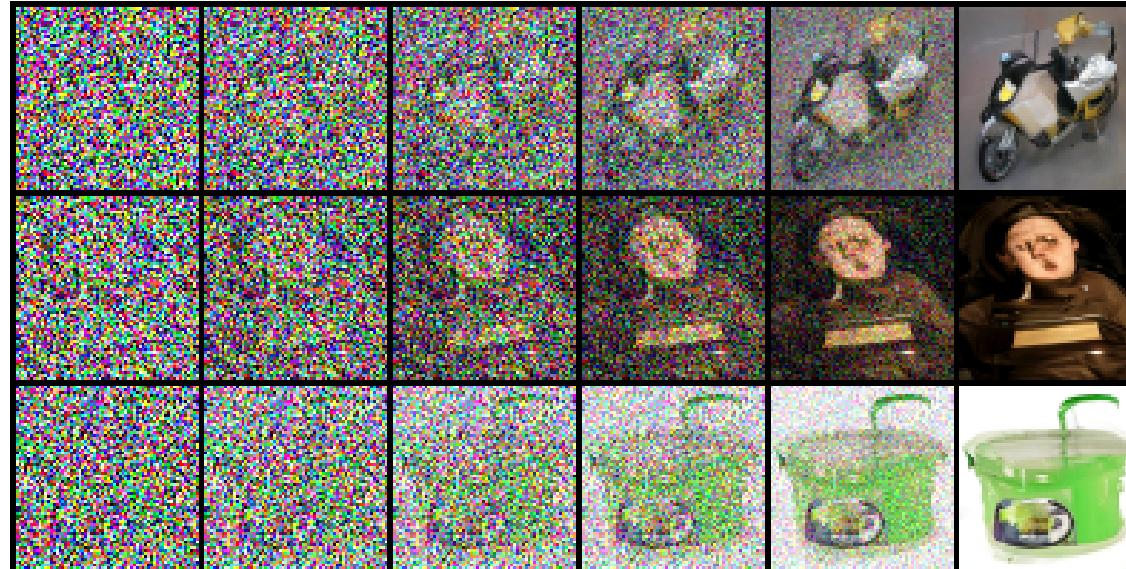
1. Design a **source distribution** $\pi(\theta|\mathbf{y})$ which acts as a first guess for the target $p(\theta|\mathbf{y})$
2. Use **Flow Matching** to move samples from the source $\pi(\theta|\mathbf{y})$ to $p(\theta|\mathbf{y})$

Key Advantages:

- Leverages abundant simulations AND expensive calibration data efficiently
- No independence assumption $\mathbf{y} \perp \theta|\mathbf{x}$ required (as opposed to RoPE).
- Can be plugged-in on top of any SBI methods that provides $\pi(\theta|\mathbf{x})$

Flow Matching: Key Idea

Flow Matching learns to transport samples between distributions



Three key elements:

1. **Base distribution** we can sample from (e.g., Gaussian) $\rightarrow q_0(x)$
2. **Target distribution** (arbitrary, potentially complex) $\rightarrow q_1(x)$
3. **Transport mechanism** to move samples $q_0(x)$ from $q_1(x) \rightarrow p_t(x)$

References:

Lipman et al. (2023). Flow Matching for Generative Modeling. *ICLR'23*.

Liu et al. (2023). Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. *ICLR'23*.

Flow Matching

Objective: Match 2 distributions q_0, q_1 via an ODE, i.e. learn a probability path $p_t(\mathbf{x})$ with $p_0 = q_0$ and $p_1 = q_1$

$$\frac{d\psi_t(\mathbf{x})}{dt} = u_t(\psi_t(\mathbf{x})) \quad \psi_0(\mathbf{x}) = \mathbf{x}$$

Relationship between flow and probability path:

$$\mathbf{x}_t = \psi_t(\mathbf{x}_0) \text{ with } \mathbf{x}_0 \sim q_0 \quad \Rightarrow \quad \mathbf{x}_t \sim p_t$$

Boundary conditions:

$$\psi_0 = \text{Id} \quad \text{and} \quad \psi_1 \# q_0 = q_1$$

How: We can regress the vector field with a neural network

$$\mathcal{L}_{FM}(\phi) = \mathbb{E}_{t \sim \mathcal{U}[0,1]} \mathbb{E}_{\mathbf{x} \sim p_t} [\|u_\phi(t, \mathbf{x}) - u(t, \mathbf{x})\|^2]$$

→ We don't know $u_t(\mathbf{x})$!

Conditional Flow Matching

Key Idea: Since we have access to samples from q_1 , we can condition the vector field on the target point $\mathbf{x}_1 \sim q_1$

$$p_t(\mathbf{x}_t) = \int q_1(\mathbf{x}_1) p_{t|1}(\mathbf{x}_t | \mathbf{x}_1) d\mathbf{x}_1 \quad p_{0|1}(\mathbf{x} | \mathbf{x}_1) = \mathcal{N}(0, I)$$
$$\Rightarrow p_{t|1}(\mathbf{x}_t | \mathbf{x}_1) = \mathcal{N}(t\mathbf{x}_1, (1-t)^2 I)$$

The induced $u_t(\mathbf{x} | \mathbf{x}_1)$ is available in closed form e.g.

$$u_t(\mathbf{x} | \mathbf{x}_1) = \frac{\mathbf{x}_1 - \mathbf{x}}{1 - t}$$

The loss becomes

$$\mathcal{L}_{CFM}(\phi) = \mathbb{E}_{t \sim \mathcal{U}[0,1], \mathbf{x}_1 \sim q_1, \mathbf{x}_t \sim p_t(\mathbf{x} | \mathbf{x}_1)} [\|u_\phi(t, \mathbf{x}) - u_t(\mathbf{x} | \mathbf{x}_1)\|^2]$$

→ Same gradient as \mathcal{L}_{FM}

FMCPE: Three Main Components

1. **Parameter space flow:** Transport samples from source $\pi(\theta|y)$ to target $p(\theta|y)$
2. **Source distribution construction:** Design $\pi(\theta|y)$ using NPE and learnable kernel

$$\pi(\theta|y) = \int p(\theta|x)q(x|y)dx$$

3. **Joint optimization:** Learn both flows together for improved robustness

Component 1: Parameter Space Flow

Goal: Map samples from source $\pi(\theta|y)$ to corrected posterior $p(\theta|y)$

Flow definition:

$$\frac{d\theta_t}{dt} = u_\Theta(t, \theta_t, y)$$

Training objective: Conditional Flow Matching loss

$$\mathcal{L}_\Theta = \mathbb{E} [\|u_\Theta(t, \theta_t, y) - (\theta_1 - \theta_0)\|^2]$$

where $\theta_1 \sim p(\theta|y)$ (from calibration) and $\theta_0 \sim \pi(\theta|y)$ (from source)

Component 2: Source Distribution

Key insight: Use simulation-trained NPE $p(\theta|x)$ with learned correction

Source construction:

$$\pi(\theta|y) = \int p(\theta|x)q(x|y)dx$$

- $p(\theta|x)$: Pretrained NPE on cheap simulations (frozen)
- $q(x|y)$: Learnable kernel mapping y to simulator space

Why this works: Bridges the gap between calibration data y and simulations x

Component 3: Learning $q(x|y)$

Approach: Use Flow Matching to learn $q(x|y)$

Flow in simulation space:

$$\frac{dx_t}{dt} = u_X(t, x_t, y)$$

Training data generation: For each calibration pair (θ_i, y_i) :

1. Generate $x_1 = S(\theta_i)$ using the simulator
2. Sample base $x_0 \sim \mathcal{N}(y, \sigma^2 I)$
3. Train flow to match $x_0 \rightarrow x_1$

Result: $q(x|y)$ learns to transport from observed space to simulator space

Joint Training Algorithm

Key idea: Train both flows u_X and u_Θ simultaneously

Training loop:

For each minibatch:

1. Sample (θ_1, y) from calibration set \mathcal{D}_{cal}
2. Generate $x_1 = S(\theta_1)$ using simulator
3. Sample base points: $x_0 \sim \mathcal{N}(y, \sigma^2 I)$
4. Map x_0 to simulator space: $\tilde{x} = T_X(x_0; y)$ using current u_X
5. Sample from source: $\theta_0 \sim p(\theta|\tilde{x})$ using pretrained NPE
6. Compute flow matching losses for both flows:

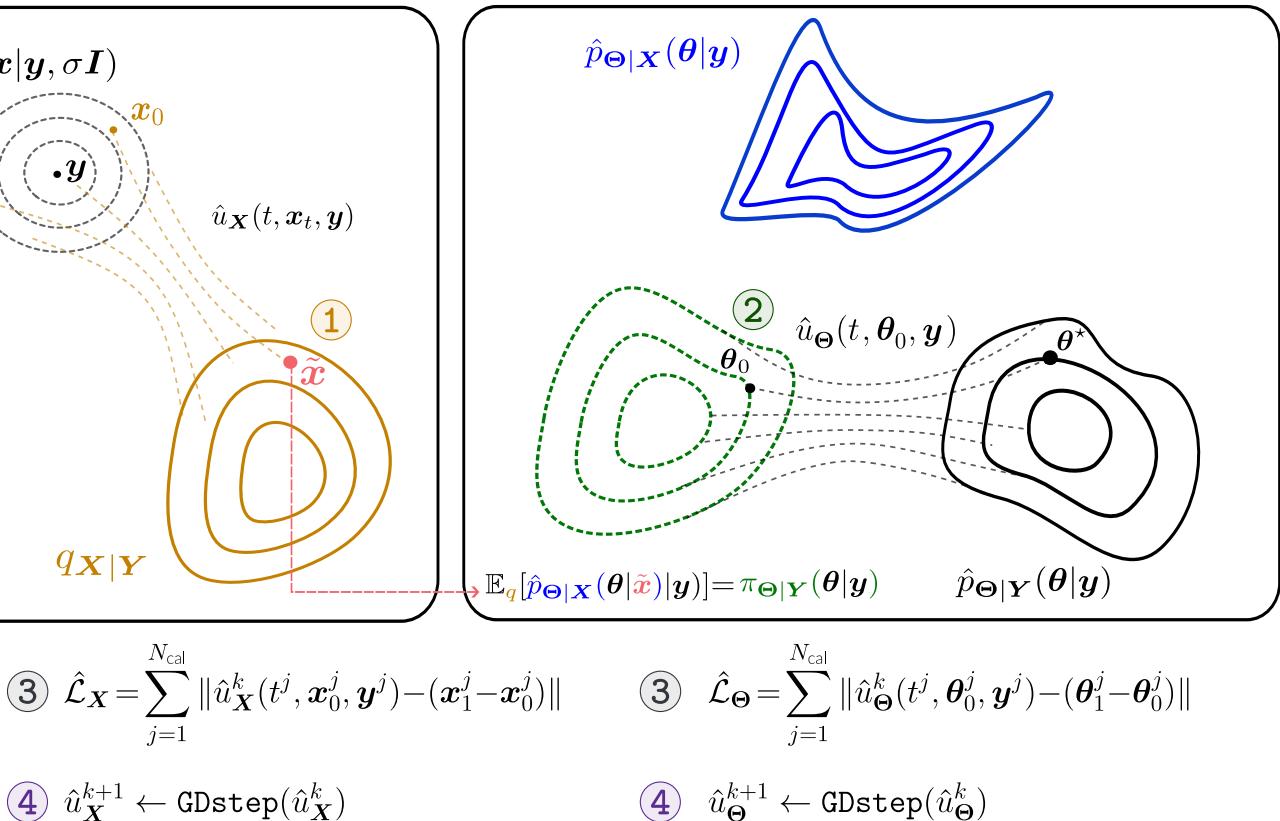
$$\ell_X = \|u_X(t, x_t, y) - (x_1 - x_0)\|^2$$

$$\ell_\Theta = \|u_\Theta(\tau, \theta_\tau, y) - (\theta_1 - \theta_0)\|^2$$

7. Update both u_X and u_Θ using $\mathcal{L} = \ell_X + \ell_\Theta$

→ Joint training ensures consistency between source and parameter flows

FMCPE Overview



Algorithm 1: Iterative update of \hat{u}_{Θ} and \hat{u}_X

Input: Calibration set $\mathcal{D}_{\text{cal}} = \{(\theta^j, \mathbf{y}^j)\}_{1 \leq j \leq N_{\text{cal}}}$, pre-trained $p_{\Theta|\mathbf{X}}$
for $k = 1$ **to** K **do**

/* $\textcircled{1}$ Sample from noise model
 Sample $\tilde{\mathbf{x}}$ from $q_{\mathbf{X}|\mathbf{Y}}(\mathbf{x}|\mathbf{y})$:

$$\mathbf{x}_0 \sim \mathcal{N}(\mathbf{y}, \sigma^2 \mathbf{I}), \quad \tilde{\mathbf{x}} = \mathbf{x}_0 + \int \underbrace{\hat{u}_X^k(t, \mathbf{x}_0, \mathbf{y})}_{\text{Weights at step } k} dt$$

/* $\textcircled{2}$ Sample from proposal
 Sample θ_0 from $\pi_{\Theta|\mathbf{Y}}^k$:

$$\theta_0 \sim \pi_{\Theta|\mathbf{Y}}^k(\theta|\mathbf{y}) = \mathbb{E}_{q_{\mathbf{X}|\mathbf{Y}}^k} [\hat{p}_{\Theta|\mathbf{X}}(\theta|\tilde{\mathbf{x}})]$$

/* $\textcircled{3}$ Compute losses with $\mathbf{x}_1^j = S(\theta^j)$ $\theta_1^j = \theta^j$

$$\hat{\mathcal{L}}_{\Theta} = \sum_{j=1}^{N_{\text{cal}}} \|\hat{u}_{\Theta}^k(t^j, \theta_0^j, \mathbf{y}^j) - (\theta_1^j - \theta_0^j)\| \quad \hat{\mathcal{L}}_X = \sum_{j=1}^{N_{\text{cal}}} \|\hat{u}_X^k(t^j, \mathbf{x}_0^j, \mathbf{y}^j) - (\mathbf{x}_1^j - \mathbf{x}_0^j)\|$$

/* $\textcircled{4}$ Gradient update

$$\hat{u}_{\Theta}^{k+1} \leftarrow \text{GDstep}(\hat{u}_{\Theta}^k), \quad \hat{u}_X^{k+1} \leftarrow \text{GDstep}(\hat{u}_X^k)$$

Experimental Setup

We evaluated FMCPE on **4 datasets**:

Toy Examples:

- Gaussian
- Pendulum

Real-world (Causal Chambers):

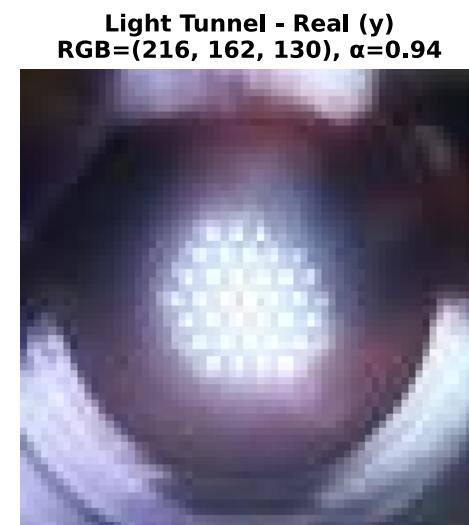
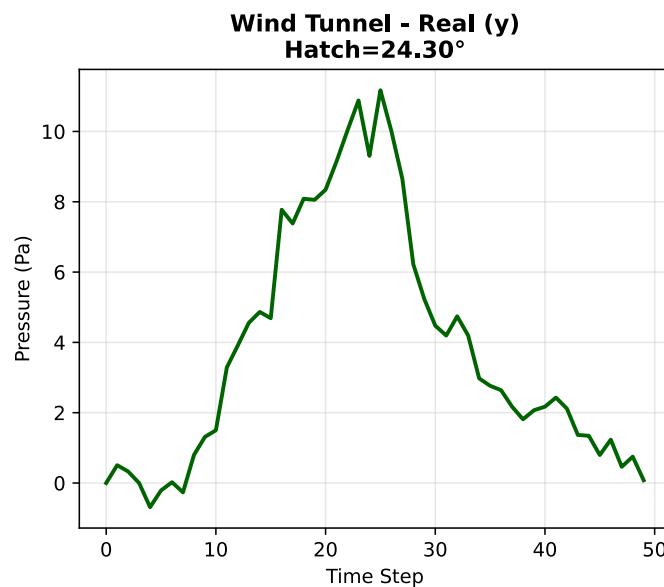
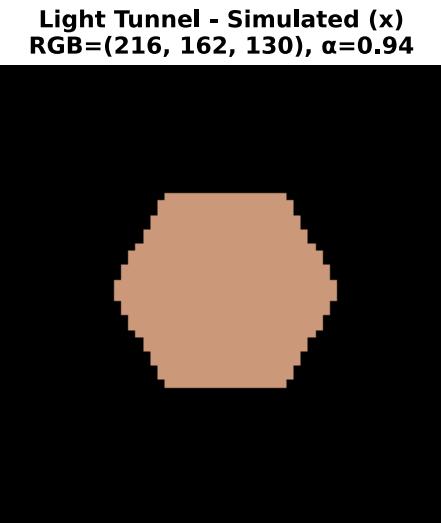
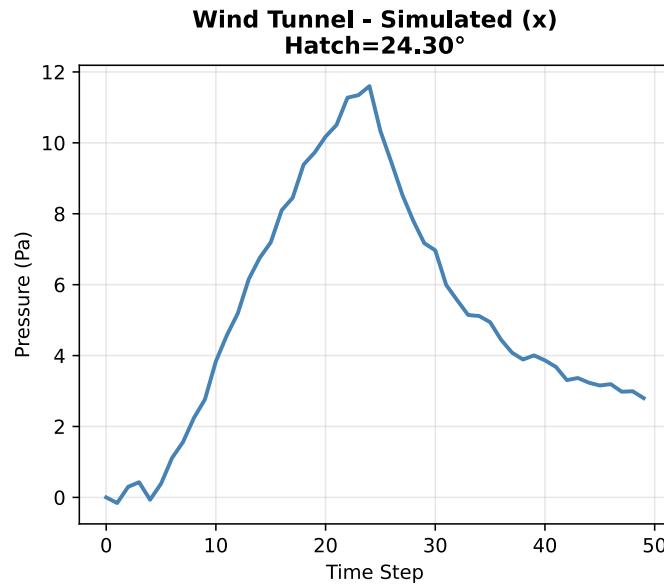
- Wind Tunnel
- Light Tunnel

Causal Chambers Dataset: Real-world benchmark for causal inference

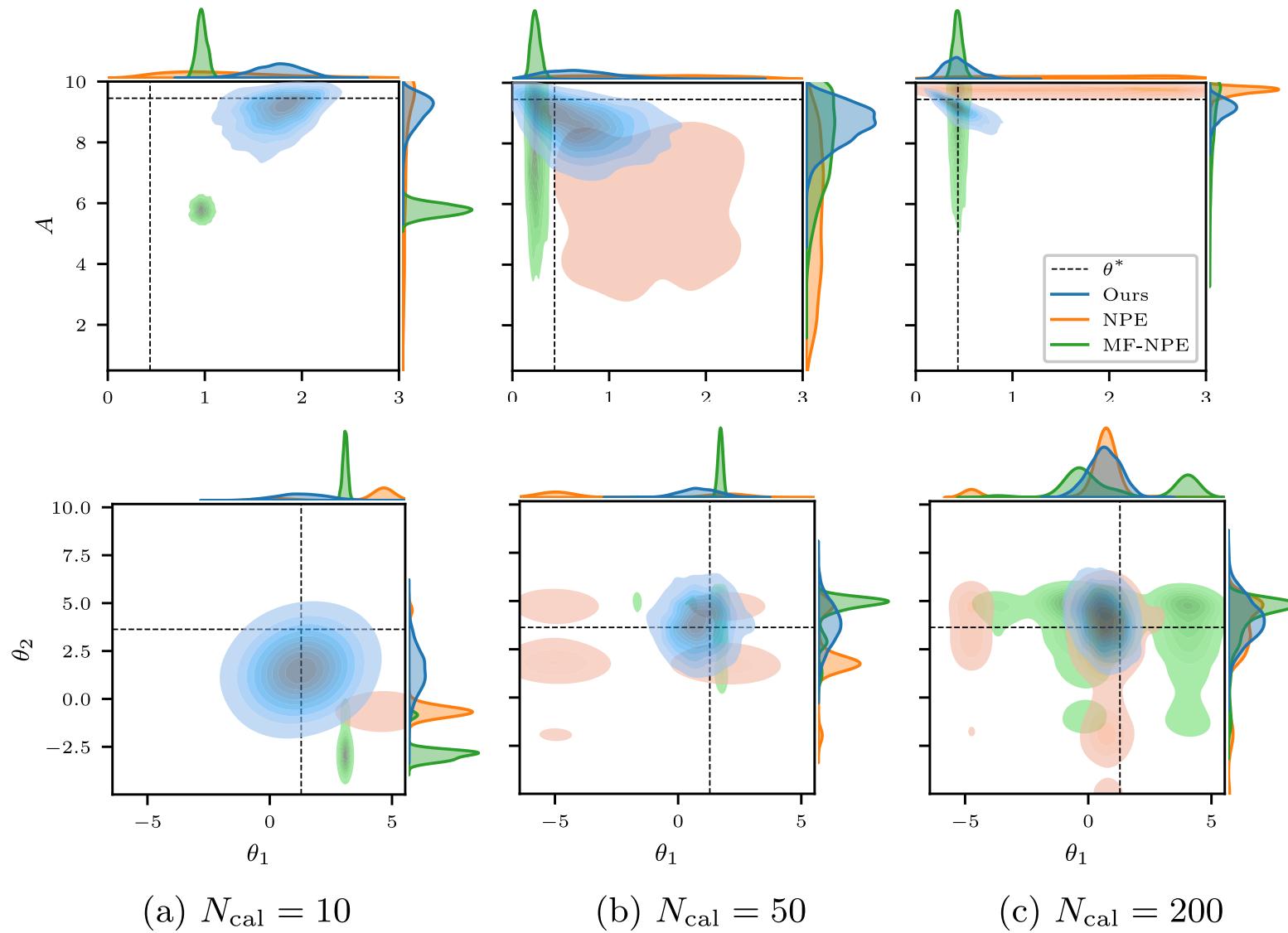
 <https://causalchamber.ai/>



Causal Chambers Dataset



Posterior Estimation Results



SBI Hackathon - Grenoble

January 21-23, 2026 | Grenoble

Join us for a hands-on hackathon on the `sbi` package - a simple and powerful tool for simulation-based inference!



<https://github.com/SBI-Grenoble/sbi-hackathon-2026>

Example: Training an NPE model in just a few lines:

```
import torch
from sbi.inference import NPE

# define shifted Gaussian simulator
def simulator(theta): return theta + torch.randn_like(theta)

# draw parameters from Gaussian prior
theta = torch.randn(1000, 2)
# simulate data
x = simulator(theta)

# choose sbi method and train
inference = NPE()
inference.append_simulations(theta, x).train()

# do inference given observed data
x_o = torch.ones(2)
posterior = inference.build_posterior()
samples = posterior.sample((1000,), x=x_o)
```

Thank You!

Questions?

Appendix

Common hypothesis

In most settings, the error is assumed to be independent of θ :

$$\mathbf{y} \perp \theta \mid \mathbf{x}$$

This is a necessary condition to express $p(\theta|\mathbf{y})$ using $p(\theta|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{y})$:

$$p(\theta|\mathbf{y}) = \int_{\mathcal{X}} p(\theta|\mathbf{x})p(\mathbf{x}|\mathbf{y})d\mathbf{x}$$

In practice, this is not always true for various physical systems. The low-fidelity simulator might not depend on some of the parameters, using a lower dimensional θ .

NPE Loss Derivation from KL Divergence

We want to minimize the KL divergence for all \mathbf{x} :

$$\begin{aligned} & \mathbb{E}_{p(\mathbf{x})} [\text{KL}(p(\theta|\mathbf{x}) || q_\phi(\theta|\mathbf{x}))] \\ &= \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p(\theta|\mathbf{x})} \left[\log \frac{p(\theta|\mathbf{x})}{q_\phi(\theta|\mathbf{x})} \right] \end{aligned}$$

Expanding the logarithm:

$$\begin{aligned} &= \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p(\theta|\mathbf{x})} [\log p(\theta|\mathbf{x})] \\ &\quad - \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p(\theta|\mathbf{x})} [\log q_\phi(\theta|\mathbf{x})] \end{aligned}$$

First term does not depend on ϕ :

$$\min_{\phi} \mathbb{E}_{p(\mathbf{x})} [\text{KL}(p || q_\phi)]$$

Equivalent to maximizing:

$$\max_{\phi} \mathbb{E}_{p(\mathbf{x})} \mathbb{E}_{p(\theta|\mathbf{x})} [\log q_\phi(\theta|\mathbf{x})]$$

Using joint distribution $p(\theta, \mathbf{x}) = p(\mathbf{x}|\theta)p(\theta)$:

$$= \max_{\phi} \mathbb{E}_{p(\theta, \mathbf{x})} [\log q_\phi(\theta|\mathbf{x})]$$

Approximate with samples:

$$\mathcal{L}(\phi, \omega) = -\frac{1}{N} \sum_{i=1}^N \log q_\phi(\theta_i | h_\omega(\mathbf{x}_i))$$

where $(\theta_i, \mathbf{x}_i) \sim p(\theta)p(\mathbf{x}|\theta)$