Beyond Likelihoods: Bayesian Parameter Inference for Black-Box Simulators with sbi

A Hands-On Introduction to Simulation-Based Inference

EuroSciPy 2025 | Kraków, Poland | 90 minutes
Case Study: Ecological Monitoring with Limited Data

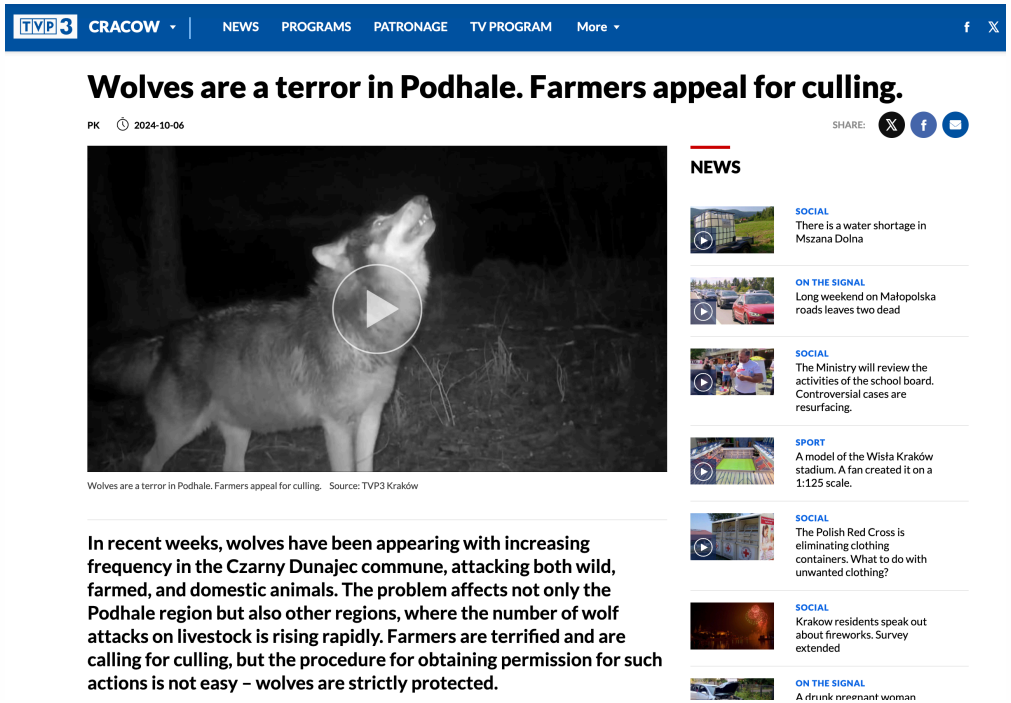
Jan Teusen (Boelts) | TransferLab, appliedAI Institute for Europe

📱 **Materials:** `github.com/janfb/euroscipy-2025-sbi-tutorial`

center

A Real Conservation Crisis in Poland

October 2024: Headlines from Southern Poland



TVP 3 CRACOW | NEWS | PROGRAMS | PATRONAGE | TV PROGRAM | More

Wolves are a terror in Podhale. Farmers appeal for culling.

PK 2024-10-06 SHARE:

Wolves are a terror in Podhale. Farmers appeal for culling. Source: TVP3 Kraków

In recent weeks, wolves have been appearing with increasing frequency in the Czarny Dunajec commune, attacking both wild, farmed, and domestic animals. The problem affects not only the Podhale region but also other regions, where the number of wolf attacks on livestock is rising rapidly. Farmers are terrified and are calling for culling, but the procedure for obtaining permission for such actions is not easy – wolves are strictly protected.

NEWS

SOCIAL
There is a water shortage in Mszana Dolna

ON THE SIGNAL
Long weekend on Małopolska roads leaves two dead

SOCIAL
The Ministry will review the activities of the school board. Controversial cases are resurfacing.

SPORT
A model of the Wisła Kraków stadium. A fan created it on a 1:125 scale.

SOCIAL
The Polish Red Cross is eliminating clothing containers. What to do with unwanted clothing?

SOCIAL
Krakow residents speak out about fireworks. Survey extended

ON THE SIGNAL
A drunk pregnant woman

TVP Kraków Reports:

"Wolves are the terror of Podhale. Farmers are calling for a cull"

The Crisis

- **Wolf attacks increasing** in Czarny Dunajec
- Targeting livestock and domestic animals
- **Farmers demanding action**
- **Wolves strictly protected** by law

> *"Problem spreading beyond Podhale to other regions"*

The Science: Escalating Wolf-Sheep Conflicts

Research Confirms the Growing Problem

ANIMAL SCIENCE AND GENETICS

ANIMAL SCIENCE AND GENETICS
Published by the Polish Society of Animal Production
vol. 20 (2024), no 4, 49-65
DOI: 10.5604/01.3001.0055.0385
Research Article *Open Access*

Preliminary report on wolf attacks on flocks of sheep of native breeds in Poland

Marta Pasternak#, Michał Puchała, Aldona Kawęcka

Department of Sheep Breeding, National Research Institute of Animal Production, 32-083 Balice n. Kraków, Poland

SUMMARY

The primary food source of wolves is wild ungulates, mainly deer, but farm animals, including sheep, may also fall prey to these predators. The aim of this study was to characterize the scale of wolf attacks on flocks of sheep of native breeds in Poland in 2015–2020, with particular emphasis on mountainous regions, considered to be the grazing areas most at risk of attacks by wolves. The study included an analysis of documentation submitted by breeders

Pasternak et al. (March 2025):

"Preliminary report on wolf attacks on flocks of sheep"

Key Findings (2015-2020)

- **76.9% of attacks** in southern Poland
- **Peak season:** July-August
- **Trend:** Increasing year over year
- **Most affected:** Podhale Zackel sheep

> *"Methods of protecting flocks should be improved"*

🎯 Your Mission: Inform Policy Decisions

You're consulting for the State Environmental Agency

The Dilemma

- **Conservation success:** Wolves recovering after near-extinction
- **Economic impact:** Farmers losing livestock
- **Policy question:** How much culling (if any)?

Your Task

- Model wolf-deer ecosystem dynamics
- Infer population parameters

Available Data

```
# Summary statistics from monitoring
observations = {
    "deer_mean": 45.2,
    "wolf_mean": 8.7,
    "deer_std": 12.1,
    "wolf_std": 2.4,
    "max_counts": [78, 15],
    "correlations": 0.82
}
```

Challenge: From limited data, infer ecosystem dynamics to guide policy

🔬 Our Tool: The Lotka-Volterra Model

Classic predator-prey dynamics

The Equations

$$\frac{dx}{dt} = \alpha x - \beta xy$$

$$\frac{dy}{dt} = \delta xy - \gamma y$$

Where:

- x = deer population
- y = wolf population
- α = deer birth rate
- β = predation rate
- δ = wolf efficiency

Why This Model?

- **Well-understood** ecological dynamics
- **Captures oscillations** seen in nature
- **Parameters map** to real processes
- **Fast to simulate** (enables SBI)

```
def lotka_volterra(params):
    α, β, δ, γ = params
    # Simulate populations
    return deer, wolves
```

The Traditional Approach: Optimization

Finding the "best" parameters

```
# Grid search or optimization
best_params = optimize(
    simulator,
    observed_data
)
```

✅ Gives an answer
❌ No uncertainty
❌ Misses alternatives

The result: A single point

```
α* = 0.52  # Birth rate
β* = 0.024 # Predation
δ* = 0.011 # Efficiency
γ* = 0.48  # Death rate
```

But how confident are we?

🎯 The Hidden Problem

Many parameters can explain your data!

Three different parameter sets, similar observations:

Parameters	α	β	δ	γ	Result
Set 1	0.52	0.024	0.011	0.48	✓ Matches
Set 2	0.48	0.026	0.009	0.51	✓ Matches
Set 3	0.55	0.022	0.012	0.45	✓ Matches

> ***Which one is correct?*** 🤔
> ***What about future predictions?*** 📈

What We Really Want: Distributions

❌ Point Estimate

- Single "best" value
- No uncertainty
- False confidence
- Poor predictions

✅ Posterior Distribution

- **Range of plausible values**
- **Quantified uncertainty**
- **Parameter correlations**
- **Robust predictions**

> ***Goal:*** `p(parameters | observation)`
>
> *The probability distribution of parameters given what we observed*

The Likelihood Problem

Why can't we just use Bayes' rule?

Bayes' Rule:

$$p(\theta|x) \propto p(x|\theta) \times p(\theta)$$

For complex simulators:

- 🎲 **Stochastic:** Different output each run
- 📦 **Black-box:** No analytical likelihood $p(x|\theta)$
- 🐌 **Slow:** Can't evaluate millions of times

Examples: Climate models, neural circuits, epidemics, cosmology...

🚀 Enter: Simulation-Based Inference

Let neural networks learn from simulations!

```
# The SBI workflow
1. parameters ~ prior()              # Sample parameters
2. data = simulator(parameters)      # Run simulation
3. train neural_network on (parameters, data) pairs
4. posterior = neural_network(observed_data)  # Inference!
```

Key insight: Turn inference into supervised learning!

- No likelihood needed ✓
- Works with any simulator ✓
- Learns from examples ✓

What You'll Learn Today

Three hands-on exercises, progressive difficulty

🔳 Exercise 1: Quick Win

15 minutes

- Load Lotka-Volterra simulator
- Run NPE in 5 lines
- Visualize posterior
- See uncertainty!

🔍 Exercise 2: Trust & Verify

20 minutes

- Posterior predictive checks
- Coverage diagnostics
- Warning signs
- "Can I trust this?"

🚀 Exercise 3: Your Problem

20 minutes

- Adapt template to your simulator
- OR use provided examples

Part 2: Core Intuition

Two Approaches to SBI

Classical vs Modern SBI

📚 Classical: Rejection Sampling

- Simple and intuitive
- No neural networks
- Inefficient in high-D
- Good for understanding

🧠 Modern: Neural Density Estimation

- Efficient and scalable
- Amortized inference
- Handles high-D
- Powers the `sbi` package

> *We'll see both for intuition, then use the modern approach*

Rejection Sampling in 5 Lines

```
# The simplest SBI algorithm
accepted_params = []

for _ in range(n_simulations):
    θ = prior.sample()                    # 1. Sample parameters
    x_sim = simulator(θ)                  # 2. Simulate data
    if distance(x_sim, x_obs) < ε:       # 3. Accept if close
        accepted_params.append(θ)         # 4. Store accepted

posterior_samples = accepted_params       # 5. These approximate p(θ|x)
```

Intuition: Keep parameters that produce data similar to observations

The Curse of Dimensionality

Acceptance rate drops exponentially! 📉

Dimensions	Acceptance Rate	Simulations for 1000 samples
2D	10%	10,000 ✅
5D	0.1%	1,000,000 😐
10D	0.00001%	10,000,000,000 ❌

Problem: In high dimensions, almost nothing is "close" to your observation

> ***Solution:*** *Learn the relationship instead of rejecting!*

Neural Posterior Estimation (NPE)

Learning to predict parameters from data

The Network

Input: Observed data `x`

Output: Distribution `p(θ|x)`

```
# Training
for θ, x in training_data:
    loss = -log q(θ|x)
    optimize(loss)

# Inference (instant!)
posterior = q(θ|x_observed)
```

Key Innovation

Transform inference into **supervised learning**

1. Generate training pairs
2. Train neural density estimator
3. Amortized inference

Result: Instant posterior for any observation!

How NPE Training Works

Three simple steps:

1 Generate Training Data

```
for i in range(n_simulations):
    θ[i] ~ prior()
    x[i] = simulator(θ[i])
```

2 Train Neural Network

```
neural_net = NeuralPosterior()
neural_net.train(parameters=θ, observations=x)
```

3 Get Posterior (instant!)

```
posterior = neural_net(x_observed)
samples = posterior.sample(10000)  # Milliseconds!
```

The Power of Amortization

Train once, infer many times! ⚡

Method	New observation	Computational Cost
MCMC	Re-run everything	Hours ⏰
Rejection	Re-run everything	Hours ⏰
NPE	Forward pass	**Milliseconds!** ⚡

Perfect for:

- Real-time applications
- Interactive exploration
- Multiple observations
- Experimental design

🚀 Let's Code!

Three exercises, increasing complexity

🔳 **Exercise 1:** First Inference (15 min)

🔍 **Exercise 2:** Diagnostics (20 min)

🎯 **Exercise 3:** Your Problem (20 min)

> ***Setup check:*** *Can everyone run this?*

```
import sbi
import torch
print("Ready for SBI! 🚀")
```

Exercise 1: Your First Inference

Wolf-Deer Dynamics from Summary Statistics!

```
# The entire SBI workflow
from sbi import inference as sbi_inference

# 1. Setup: simulator outputs summary stats
simulator_with_stats = lambda θ: compute_summary_stats(
    lotka_volterra(θ)
)

# 2. Train neural network on summary statistics
npe = sbi_inference.NPE(prior)
npe.train(simulator_with_stats, num_simulations=10000)

# 3. Infer parameters from observed summaries
posterior = npe.build_posterior(observed_stats)

# 4. Sample & visualize uncertainty!
samples = posterior.sample((1000,))
plot_posterior(samples)
```

📝 **Open notebook:** `01_first_inference.ipynb`

Exercise 2: Trust but Verify

Critical with Summary Statistics! 🔍

Why extra important? Summary stats lose information → Need validation!

Four key diagnostics:

1. Prior Predictive Check

- Can prior generate observations?
- Catch bad prior specification

2. Training Diagnostics

- Did neural network converge?
- Check for overfitting

3. Posterior Predictive Check

- Can posterior recreate data?
- Validates summary statistics choice

4. Simulation-Based Calibration

- Are credible intervals calibrated?
- 90% CI contains truth 90% of time?

📝 **Open notebook:** `02_diagnostics.ipynb`

Exercise 3: Your Own Problem

Three options:

🔬 Option A: Your Simulator

If you brought one, we'll adapt it!

🎾 Option B: Ball Throw Physics

Simple projectile motion with air resistance

🦠 Option C: SIR Epidemic Model

Disease spread dynamics

Template provides:

- Prior specification guide
- Simulator wrapper
- Diagnostic pipeline

Part 4: Next Steps

Where to go from here

Beyond NPE: The Full SBI Toolbox

Method	What it learns	Best for	Key advantage
NPE	`p(θ|x)`	Fast amortized inference	Instant posteriors
NLE	`p(x|θ)`	MCMC sampling	Exact inference
NRE	`p(θ,x)/p(θ)p(x)`	Model comparison	Hypothesis testing
Sequential	Iteratively	Sample efficiency	10x fewer simulations

All available in the `sbi` package with the same interface!

⚠️ Common Pitfalls & Solutions

Learn from our mistakes!

Pitfall	Consequence	Solution
Prior too wide	Wasted simulations	Start narrow, expand if needed
Too few simulations	Poor approximation	Use diagnostics!
Ignoring diagnostics	False confidence	Always verify
Poor summary stats	Information loss	Include diverse statistics
Assuming sufficiency	Missing key info	Test with diagnostics

> ***Golden rule:*** *Always validate your results!*

Advanced Topics

Where to dive deeper 🏊

Methods

- NLE+ `pyro` (**Talk Wed, 11:40, 1.38**)
- Multi-round inference (sequential)
- Flow matching, diffusion models
- Tabular Foundation Models for NPE

Applications

- Hierarchical Bayesian inference
- Expensive simulators
- High-dimensional problems
- Training-free SBI

> 📚 ***Resources:*** *Papers, tutorials, and examples at sbi-dev.github.io*

🌍 Real-World Applications

SBI in the wild:

Science

- 🧠 **Neuroscience:** Neural circuits
- 🦠 **Epidemiology:** COVID-19 models
- 🌍 **Climate:** Weather prediction
- 🔬 **Physics:** Gravitational waves
- 🧬 **Biology:** Gene regulation

Engineering

- 🚗 **Automotive:** Safety testing
- 💊 **Pharma:** Drug discovery

> ***Your application next?*** 🚀

Join the SBI Community!

SBI Hackathon 2025, Tübingen - Join us next time!

📦 The Package

- GitHub: github.com/sbi-dev/sbi
- 700+ stars, 82+ contributors
- Active development

📚 Resources

- Documentation
- JOSS paper
- Tutorial papers & notebooks

💬 Get Help & Connect

- GitHub Discussions
- **Annual hackathons** (next: 2026!)
- Discord Server

🤝 Contribute!

- Join the next hackathon
- Add your use case
- Help others get started

Thank You! 🙏

Questions?

📧 Contact

GitHub: Create an issue or discussion

Discord: Join via GitHub link

💬 Let's Talk!

Available after the session for discussions

📱 Materials & Feedback

center

[LINK_PLACEHOLDER]

Scan for:

- Tutorial notebooks
- Slides
- Feedback survey

What will you infer? 🚀

Backup Slides

Mathematical Details: NPE Loss

Training objective

The neural posterior estimator minimizes:

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

Where:

- $q_\phi(\theta|x)$ is the neural network approximation
- ϕ are the network parameters
- Expectation over joint distribution of parameters and data

Implementation: Normalizing flows for flexible distributions