A Brief Introduction to Probabilistic Programming

Martin Jankowiak
November 23rd, 2019

Uber AI



Outline

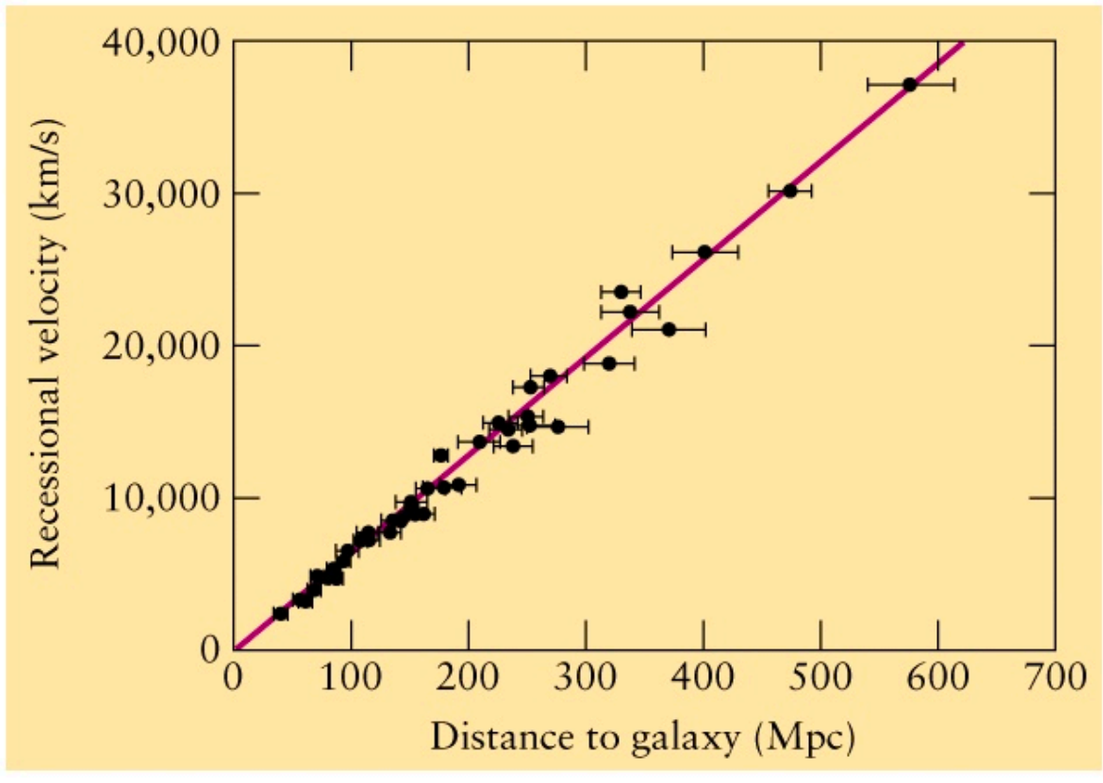
- Introduction
- Timeseries
- Experimental Design
- Gravitational Lensing

Model Based Reasoning

A basic model: linear regression

$$v = H_0 D$$

Hubble's Constant



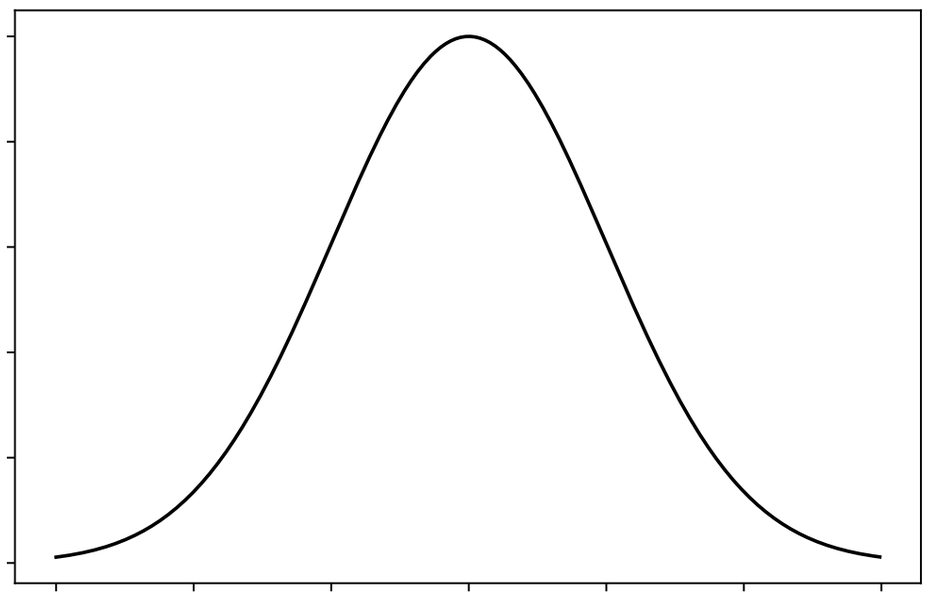
Models connect hypotheses to observed data

Probabilistic Modeling

We want to incorporate **uncertainty.**
So make unknown quantities **random variables.**

Prior distribution:
represents prior beliefs

$$p(H_0)$$



Probabilistic Modeling

We want to incorporate **uncertainty**;
So make unknown quantities **random variables**

Likelihood:
connects observations to latent random variables

$$p(v|H_0, D)$$

$$p(v|H_0, D) = \mathcal{N}(v|H_0 D, \sigma^2_{\text{obs}})$$

normal (i.e. gaussian) distribution

$$v = H_0 D + \sigma_{\text{obs}} \epsilon$$

independent gaussian noise

Bayesian Inference

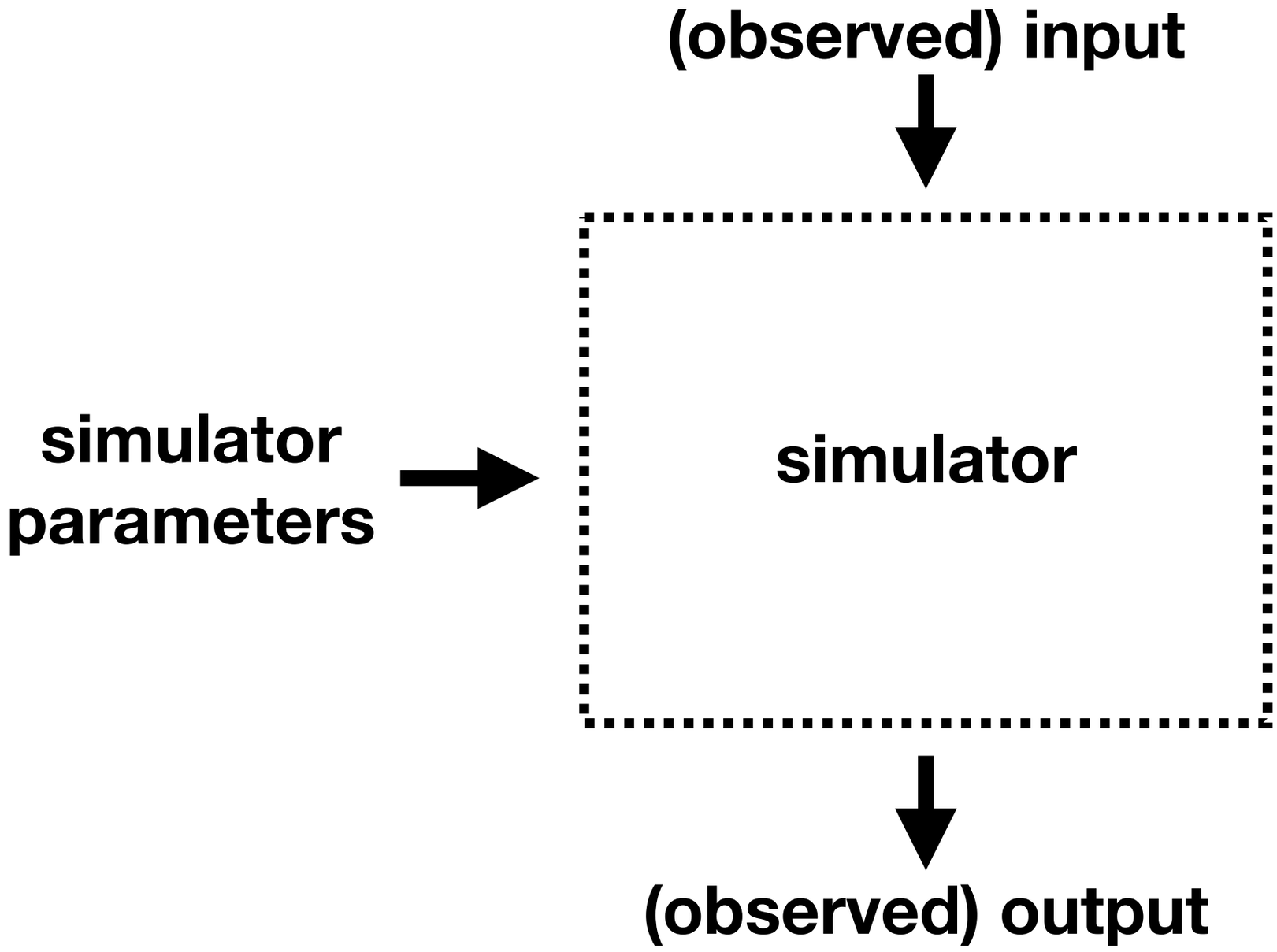
After **observing data** we want to **update our beliefs** about the latent random variables.

Posterior distribution:

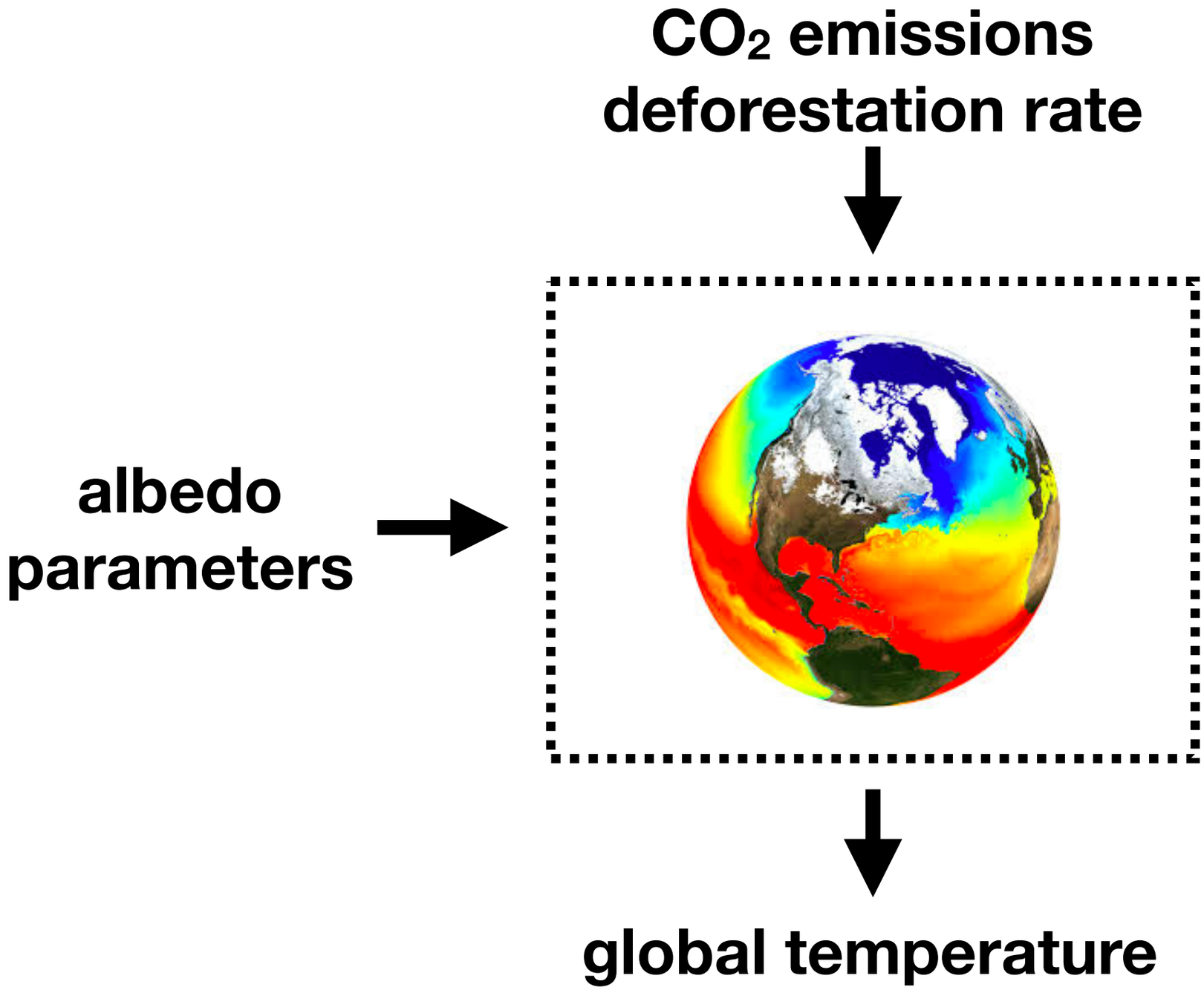
$$p(H_0|\{v_i, D_i\}) = \frac{p(H_0)\prod_i p(v_i|H_0, D_i)}{\int dH_0' p(H_0')\prod_i p(v_i|H_0', D_i)}$$

The posterior distribution represents a **balance** between the values of H_0 preferred by the **observed data** and the values of H_0 preferred by the **prior**.

Modeling as Simulation

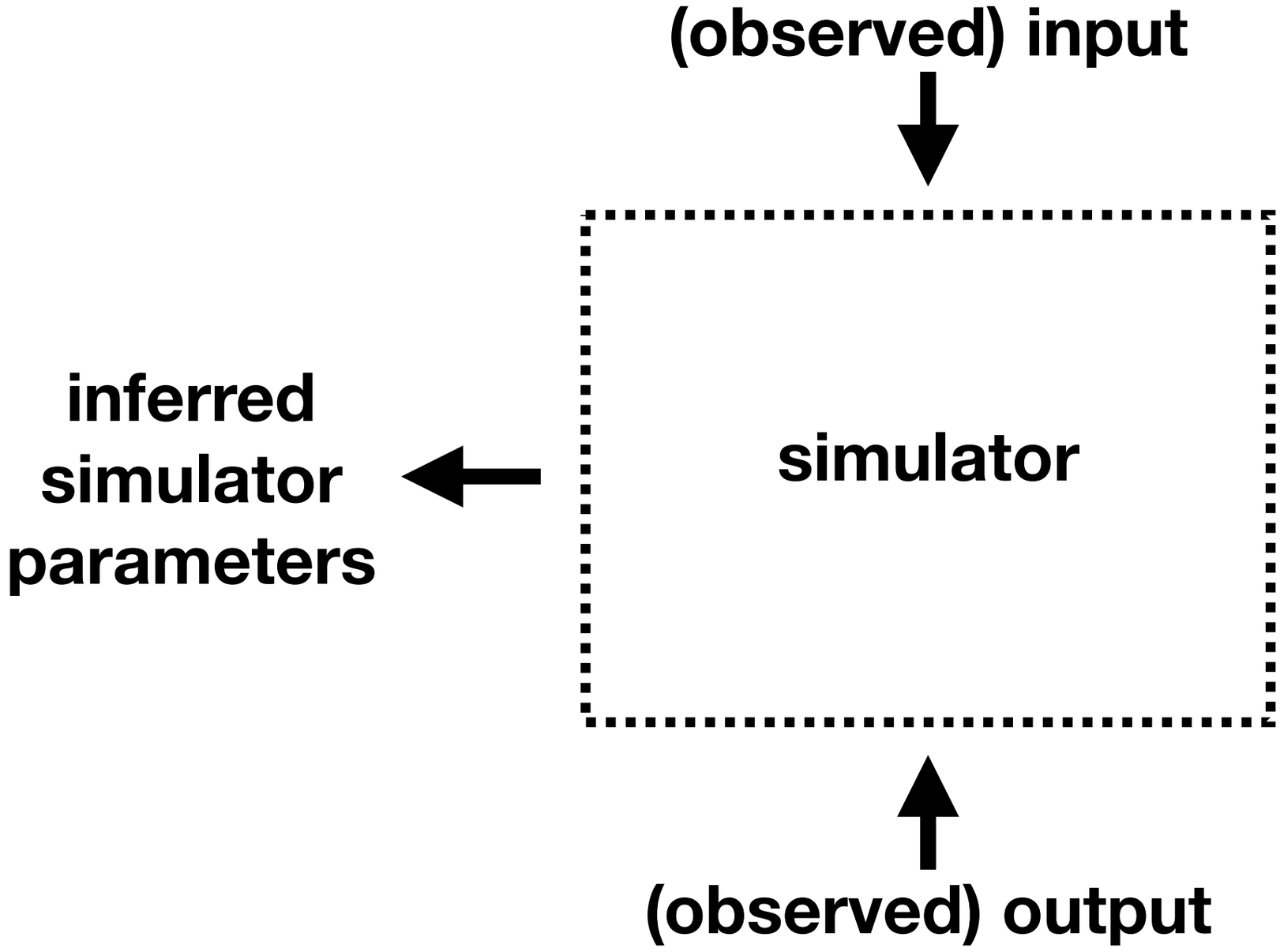


Modeling as Simulation



Inference as an Inverse Problem

How do we setup our simulator to obtain given observed inputs and outputs?



Programming Languages

Most modern programming languages are **Turing complete**. They are powerful tools for representing an infinite variety of different types of computations with the help of flexible coding paradigms like function composition, recursion, polymorphism, higher order functions…

```
def fibonacci(n):
    # Base case
    if n == 0:
        return 0
    elif n == 1:
        return 1

    # Recursion
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

Probabilistic Programming Languages

Probabilistic modeling + programming languages = PPL = **?**

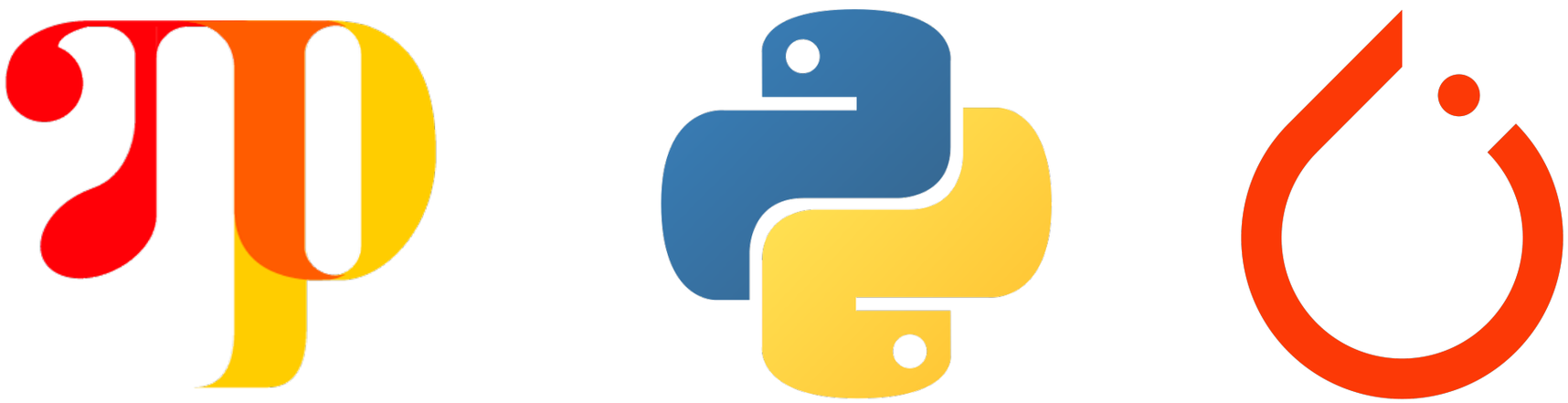
Different views:

- Adding stochastic primitives to programming languages
- As formal models for the representation of and reasoning about uncertainty
- Tools for automating model specification and inference

  PPL :: probabilistic modeling ◀▬▶ TF/PyTorch :: neural networks
- …

Pyro

Pyro is a PPL built on **Python** and **PyTorch**.



https://pyro.ai

There are **many more** PPLs out there!

Hakaru, Augur, R2, Figaro, IBAL, PSI, Church, Anglican,
BLOG, Edward, Infer.NET, PyMC3, Factorie, Turing.jl,
BayesDB, Venture, Probabilistic-C, ProbTorch, webPPL,
CPProb, Biips, LibBi, Birch, STAN, JAGS, BUGS…

A Mostly Deterministic Climate Simulator

```
1 def simulator(albedo):
2     mean_temp = simulate_temp(albedo)
3     return mean_temp + add_noise()
```

A Pyro Model

prior distribution

```
1 def model(observed_temp):
2     albedo = pyro.sample("albedo", LogNormal(5.0, 1.5))
3     mean_temp = simulate_temp(albedo)
4     return pyro.sample("temp", Normal(mean_temp, 1.0),
5                        obs=observed_temp)
```

- Pyro models are **stochastic functions**
- we use the **sample** primitive to declare stochastic program values
- sample sites are **named**
- samples are governed by (prior/likelihood) **distributions**

Pyro Interface

initial value

```
theta = pyro.param("theta", torch.ones(100),
                           constraint=positive)
```

Primitives

- sample
- param
- plate
- …

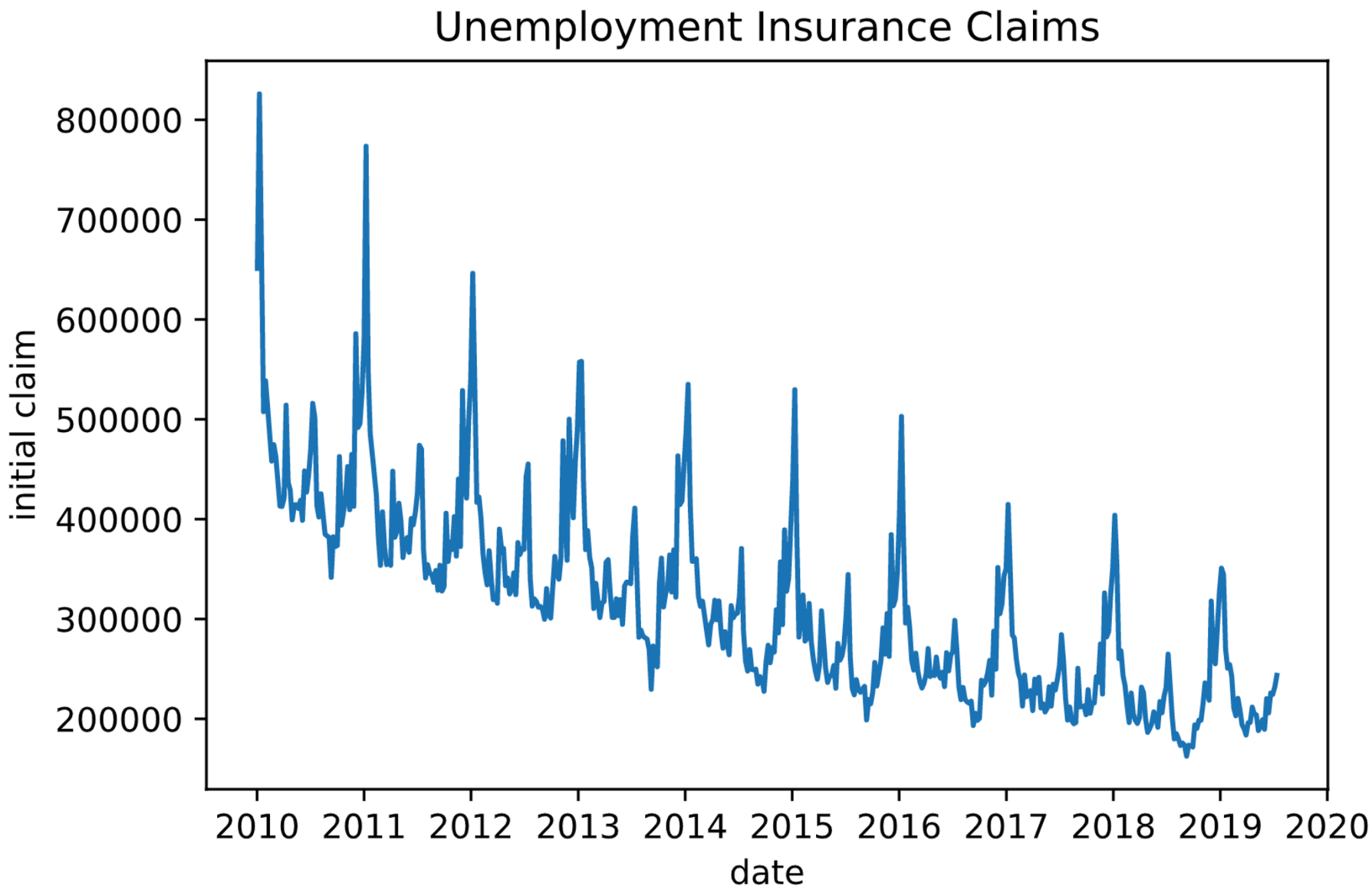
Effects

- trace
- replay
- markov
- …

Inference

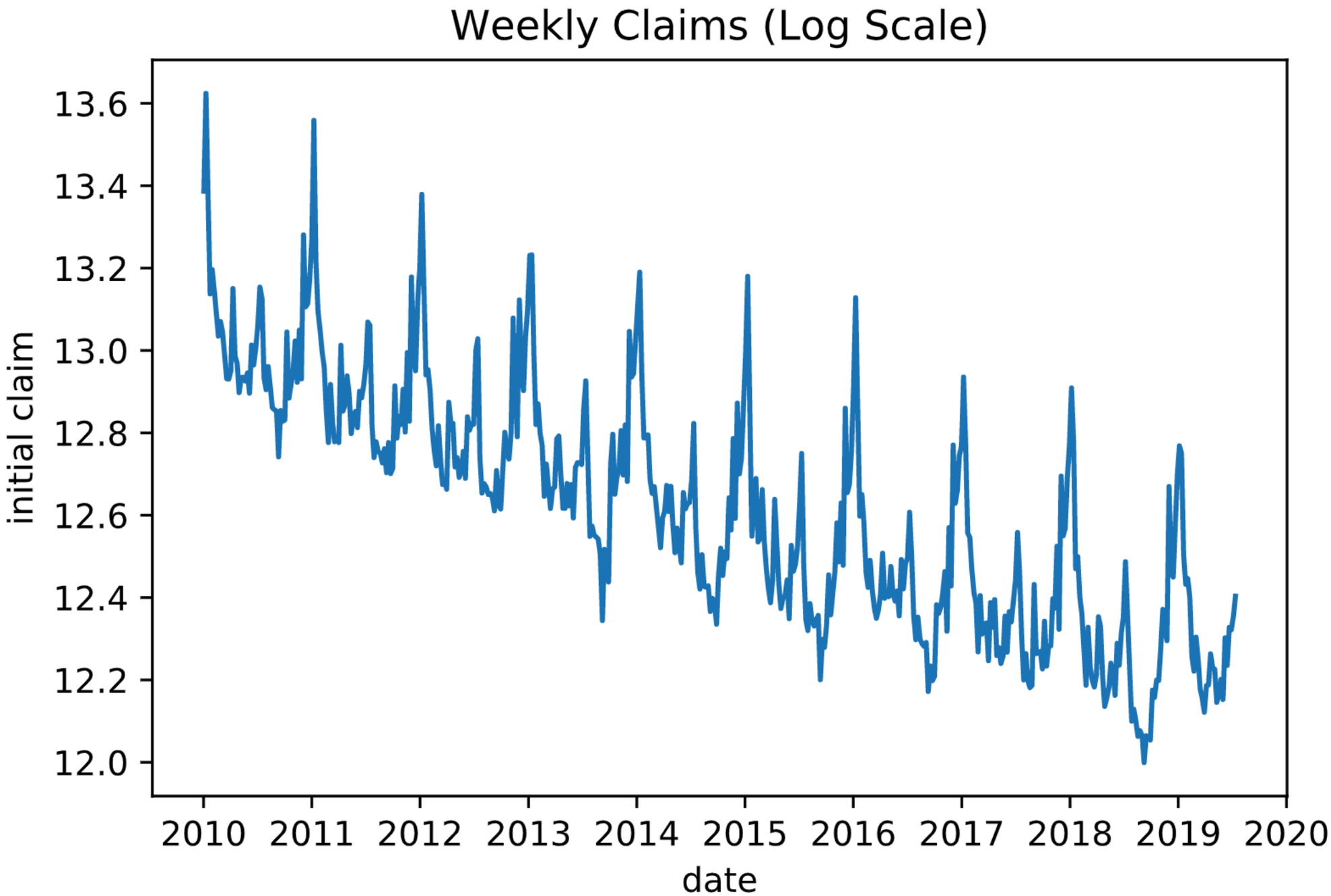
- Variational Inference
- HMC
- …

Timeseries Modeling



Follow along at *https://pyro.ai/time*

Timeseries Modeling



Global Trend Model

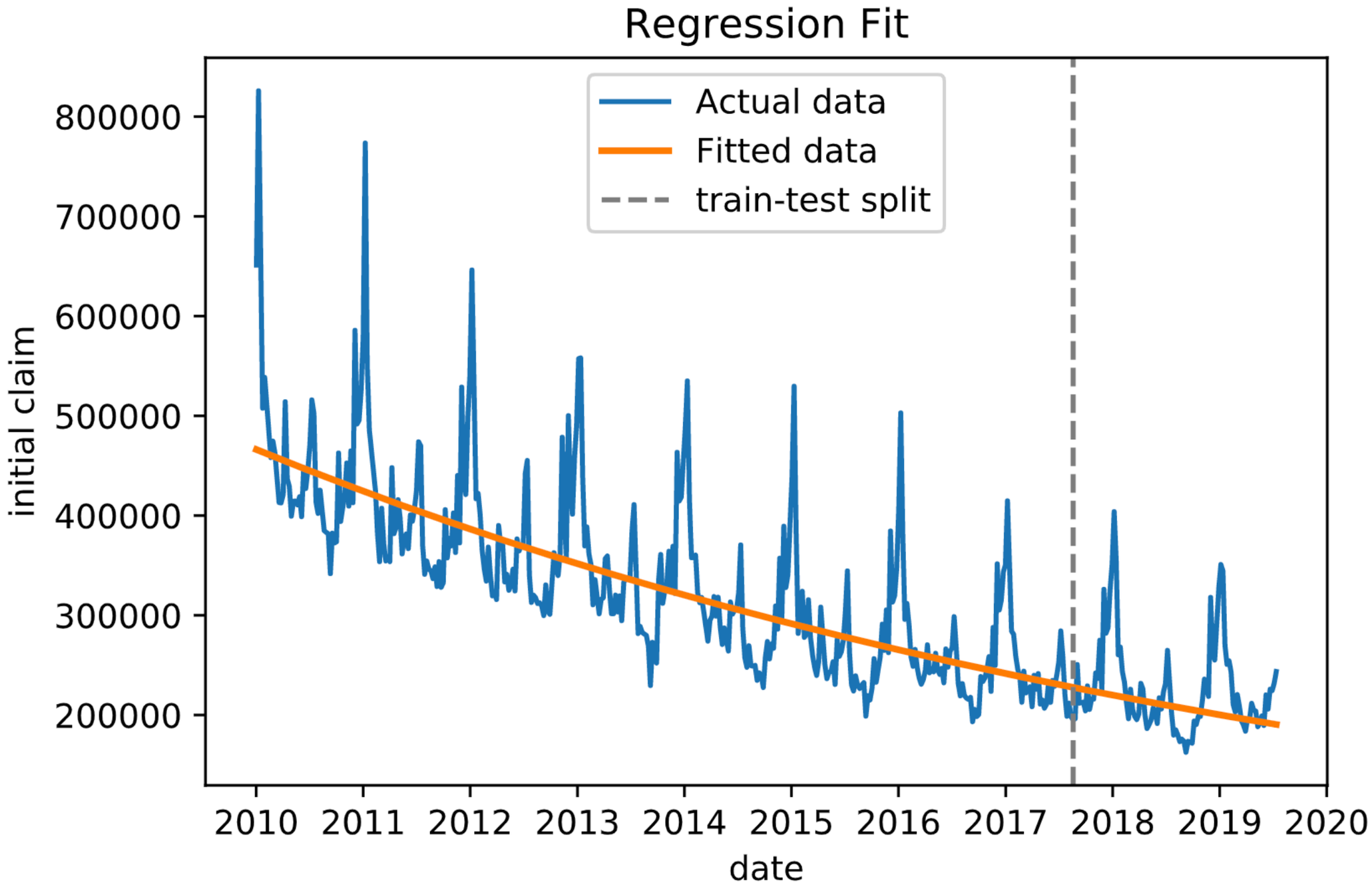
$$\mu_t = \text{slope} \times t + \text{intercept}$$

$$\log y_t \sim \mathcal{N}(\mu_t, \sigma)$$

```
1 def model_global_trend(y):
2     slope = pyro.param('slope', torch.tensor(0.))
3     intercept = pyro.param('intercept', torch.tensor(13.))
4     for t in range(len(y)):
5         log_y_hat = slope * t + intercept
6         pyro.sample('y_{}'.format(t),
7                     dist.LogNormal(log_y_hat, 1.), obs=y[t])
```

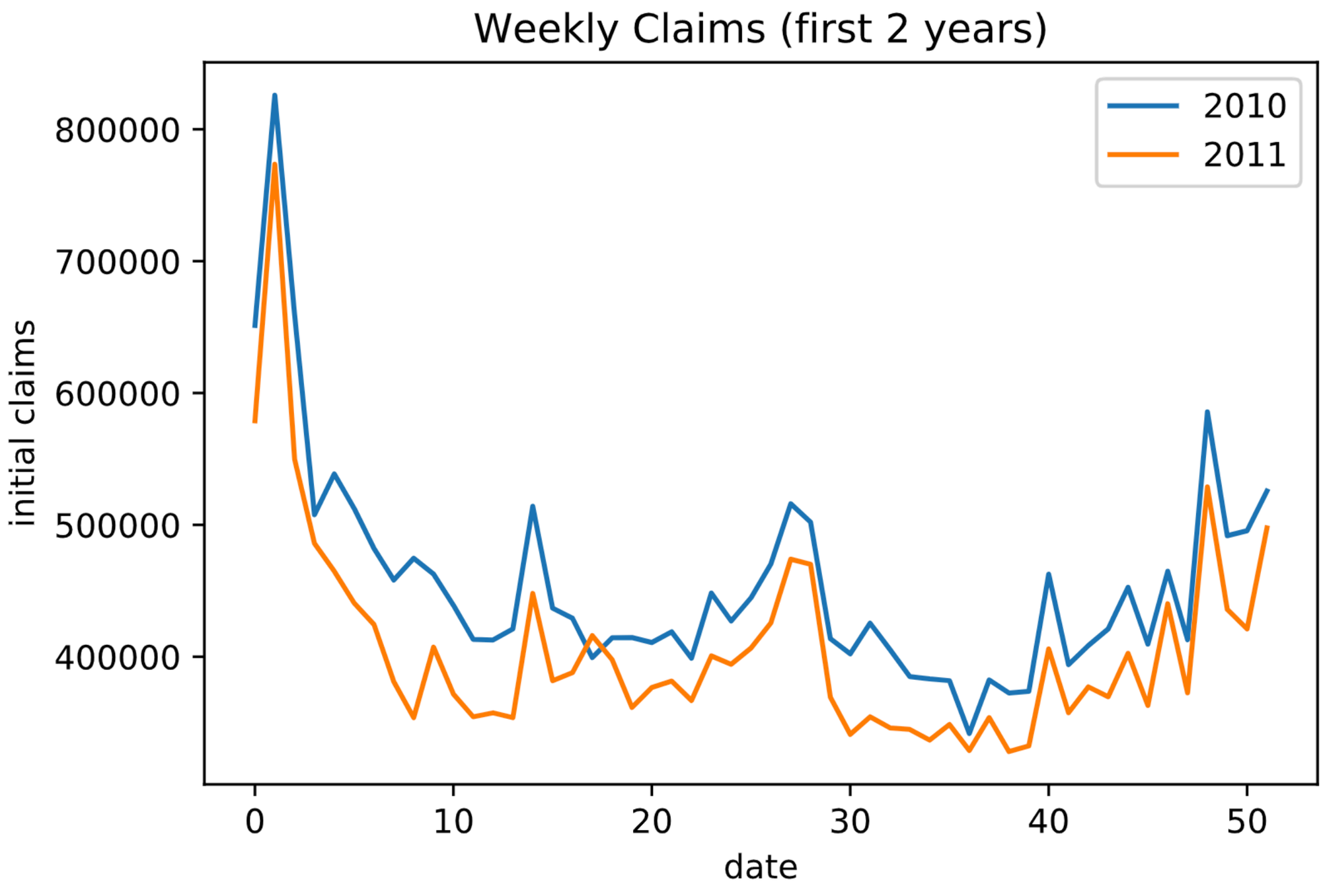
observed data

Global Trend Model



not bad but we're clearly missing **seasonality** effects!

Seasonal Global Trend Model



Seasonal Global Trend Model

```
def prob_sgt(y):
    N = y.size(0)
    slope = pyro.sample('slope', Normal(0.0, 1.0))
    obs_sd = pyro.sample('obs_sd', HalfNormal(1.0))
    with pyro.plate('S', 52):
        seasonality = pyro.sample('seasonality', Normal(0.0, 1.0))
        seasonality = seasonality - seasonality.mean()
    with pyro.plate('N', N) as time:
        seasonality = seasonality.repeat(math.ceil(N / 52))[:N]
        log_y_hat = slope * time + seasonality
        return pyro.sample('y', LogNormal(log_y_hat, obs_sd),
                           obs=y)
```

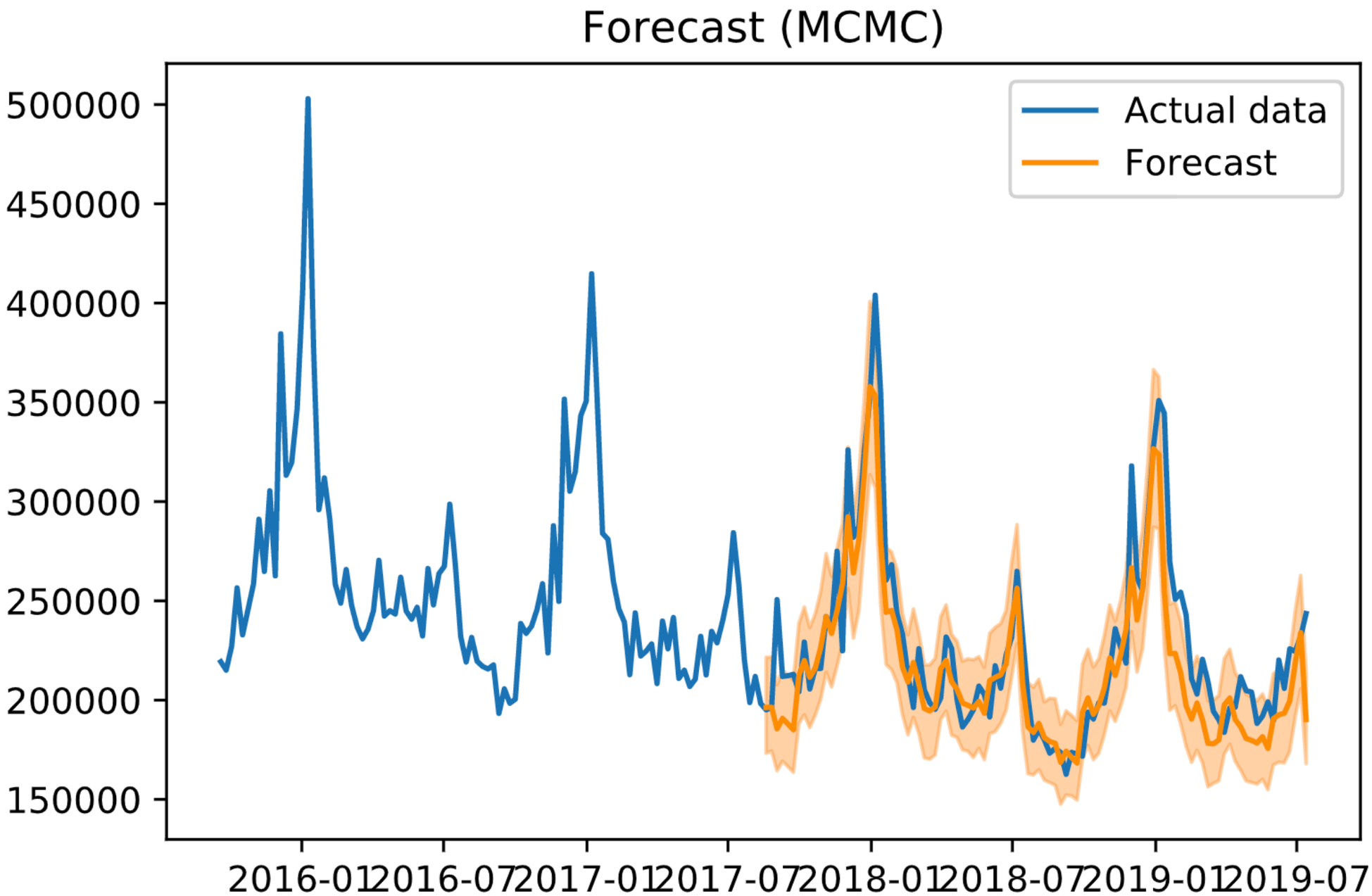
- we introduced priors for our latent parameters (slope, intercept, …)
- we now explicitly account for **seasonality**
- we use **pyro.plate** to define vectors of conditionally independent random variables

Seasonal Global Trend Model

Inference with Hamiltonian Monte Carlo:

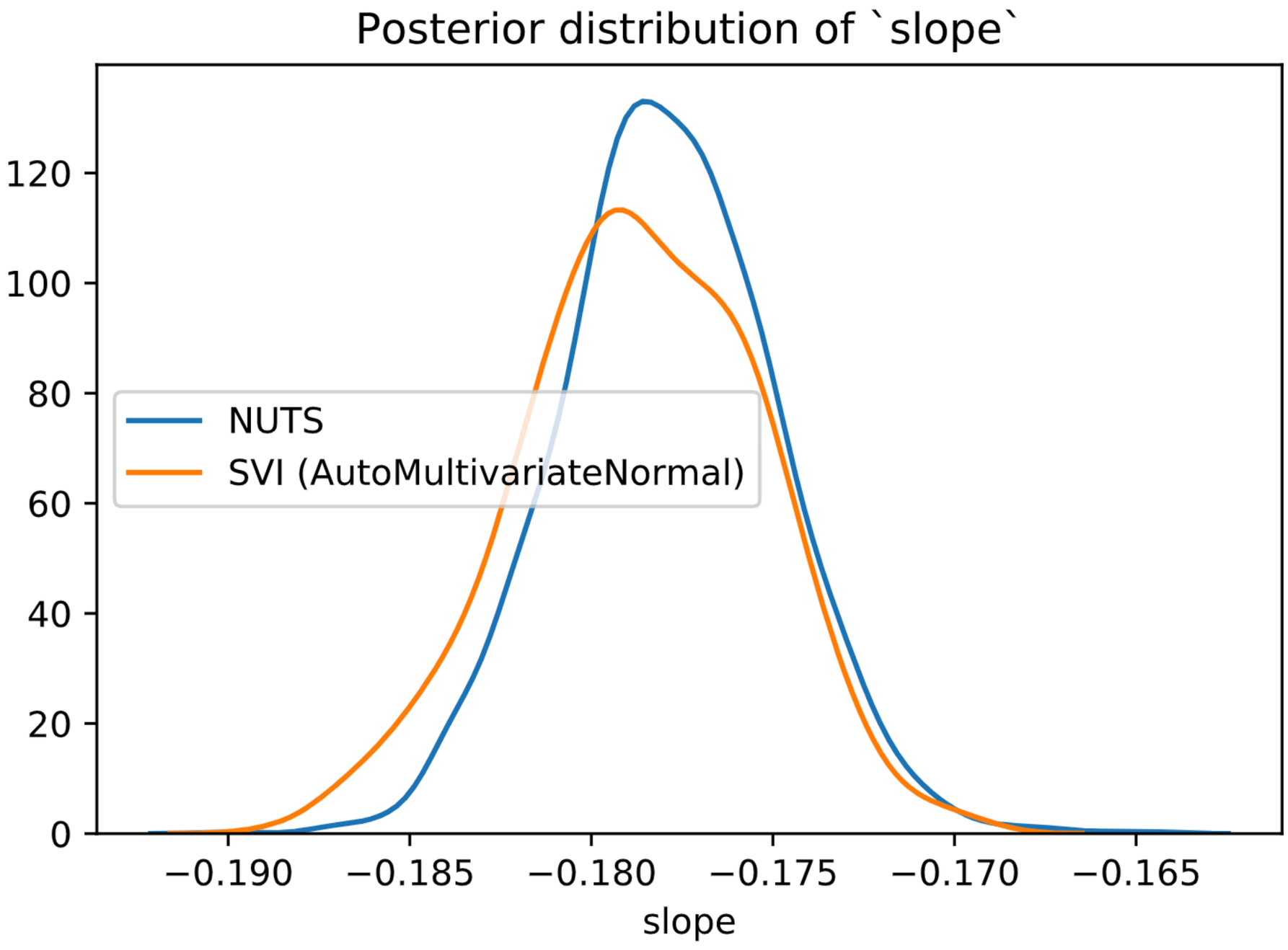
```
nuts_kernel = pyro.infer.mcmc.NUTS(model)
mcmc = pyro.infer.mcmc.api.MCMC(nuts_kernel,
                                num_samples=2000,
                                warmup_steps=1000)
mcmc.run(y)
```

Seasonal Global Trend Model



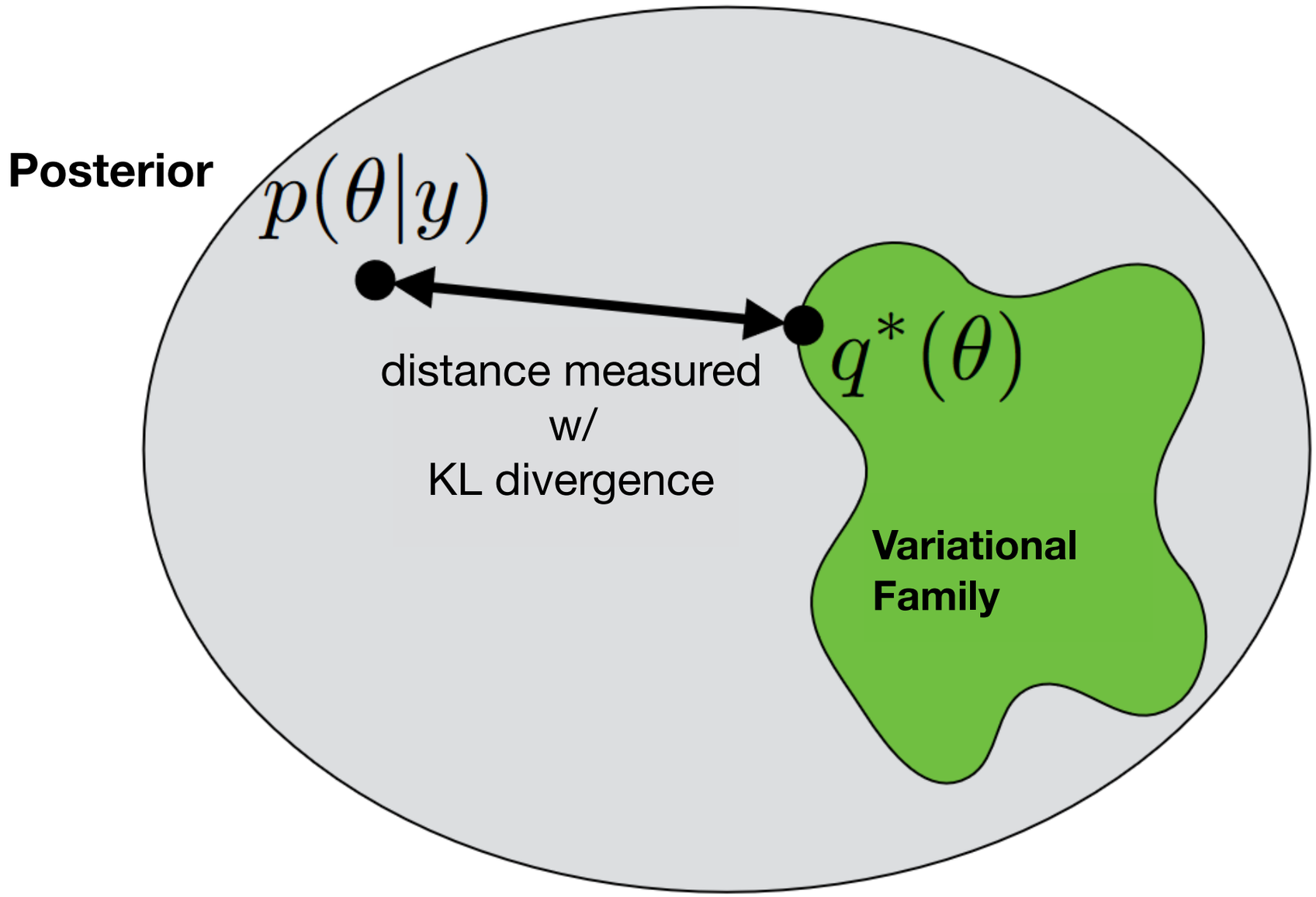
Seasonal Global Trend Model

Inspecting the posterior over the slope



Aside: Variational Inference

Bayesian Inference as optimization



Amortized Variational Inference

```
def model(y):
    pyro.sample("theta", prior_distribution)
    pyro.sample("y", likelihood_distribution,
                obs=y)

def guide(y):
    pyro.module("nn", nn)
    variational_params = nn(y)
    variational_dist = Distribution(variational_params)
    pyro.sample("theta", variational_dist)
```

after parameter training, the guide is
our approximate posterior distribution

Amortized Variational Inference

Learn the guide program by maximizing the ELBO objective

$$\mathcal{L}_{\mathrm{ELBO}} \equiv \mathbb{E}_{q(\theta)}\left[\log \frac{p(y,\theta)}{q(\theta)}\right] \leq \log p(y)$$

1. run the guide program forward
2. run the model program forward,
   replaying stochastic choices from the guide
3. compute the ELBO
4. take a gradient step
5. repeat until convergence

Bayesian Optimal Experimental Design

Variational Estimators for Bayesian Optimal Experimental Design

Adam Foster[†*] **Martin Jankowiak**[‡] **Eli Bingham**[‡] **Paul Horsfall**[‡]
Yee Whye Teh[†] **Tom Rainforth**[†] **Noah Goodman**[‡§]
[†]Department of Statistics, University of Oxford, Oxford, UK
[‡]Uber AI Labs, Uber Technologies Inc., San Francisco, CA, USA
[§]Stanford University, Stanford, CA, USA
adam.foster@stats.ox.ac.uk

Abstract

Bayesian optimal experimental design (BOED) is a principled framework for making efficient use of limited experimental resources. Unfortunately, its applicability is

bioinformatics (Vanlier et al., 2012), and neuroscience (Shababo et al., 2013).

Suppose we have parameters θ that we wish to learn about and for which we have some prior beliefs $p(\theta)$. These parameters might, for example, encapsulate characteristics of a participant in a psychology trial.

Bayesian data analysis

observations y

parameter θ

prior $p(\theta)$

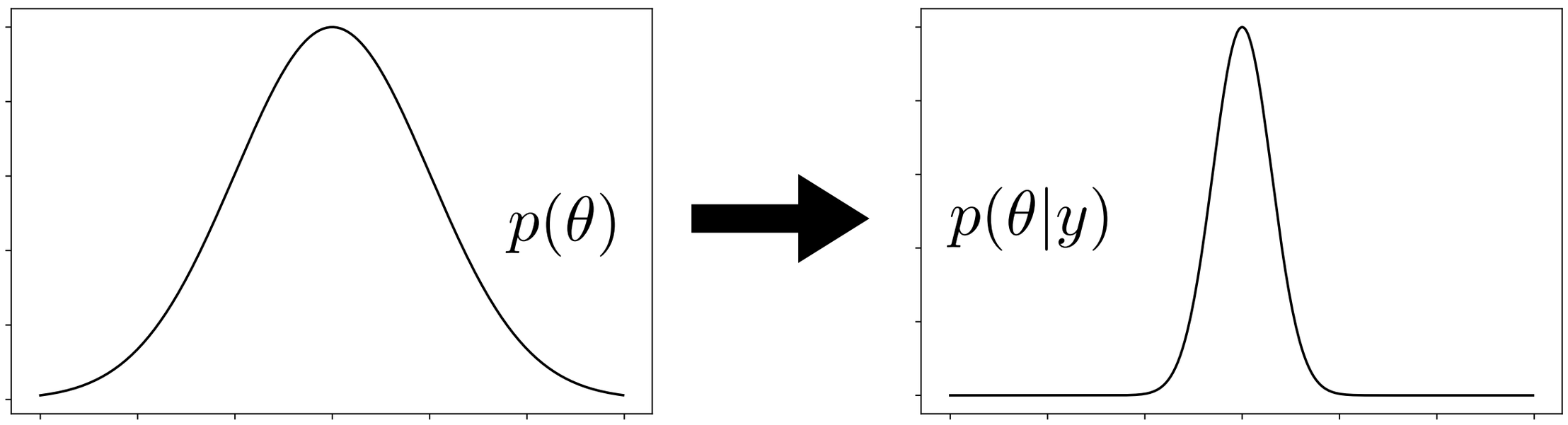
likelihood $p(y|\theta)$

joint $p(y, \theta) = p(y|\theta)p(\theta)$

posterior $$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{\int d\theta' p(y|\theta')p(\theta')}$$

Bayesian data analysis

1. do experiment
2. observe outcome y
3. compute posterior
4. posterior becomes new prior
5. rinse and repeat



Bayesian optimal experimental design

1. do experiment
2. observe outcome y
3. compute posterior
4. posterior becomes new prior
5. rinse and repeat

what experiment should we do?

Bayesian optimal experimental design

the design is the part of the experiment we control

1. **which questions do we ask in a survey?**
2. **what dosage of drug do we give a patient?**
3. **at what times do we measure a population for the flu?**
4. **. . .**

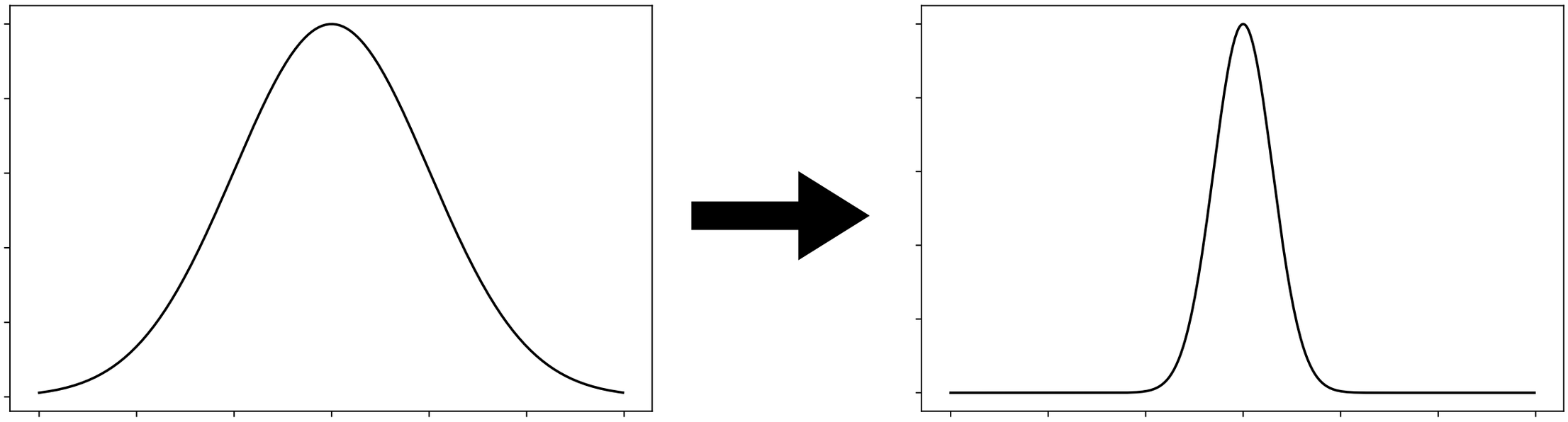
non-random design d → d

non-random design d → θ

$$p(y, \theta|d) = p(y|\theta, d)p(\theta)$$

Bayesian optimal experimental design

the natural bayesian formulation of this problem uses an ***information theoretic*** criterion to choose the design.



we want to use designs that reduce the ***posterior entropy***.

Bayesian optimal experimental design

to formalize this notion we define the

information gain

$$\mathrm{IG}(y, d) = H[p(\theta)] - H[p(\theta|y, d)]$$

prior entropy posterior entropy

Bayesian optimal experimental design

our criterion for choosing the design will be the

expected information gain

$$\mathrm{EIG}(d) \triangleq \mathbb{E}_{p(y|d)}\left[H[p(\theta)] - H[p(\theta|y,d)]\right]$$

$$= \mathbb{E}_{p(y,\theta|d)}\left[\log \frac{p(\theta|y,d)}{p(\theta)}\right]$$

$$= \mathbb{E}_{p(y,\theta|d)}\left[\log \frac{p(y,\theta|d)}{p(\theta)p(y|d)}\right]$$

mutual information

Bayesian optimal experimental design

the **optimal** design is then given by

$$d^* \in \underset{d \in \mathcal{D}}{\arg\max}\ \mathrm{EIG}(d)$$

Bayesian optimal experimental design

estimating the EIG is hard…



$$\mathbb{E}_{p(y,\theta|d)}\left[\log \frac{p(\theta|y,d)}{p(\theta)}\right]$$

posterior distribution

Variational bayesian OED

upper bound

$$\mathrm{EIG}(d) \leq \mathbb{E}_{p(y,\theta|d)}\left[\log \frac{p(y|\theta,d)}{q_m(y|d)}\right]$$

Variational distribution

A concrete example

We want to understand human's working memory capacity



Let's ask people to memorize d-digit numbers

design

A concrete example

Hypothesis: longer numbers are harder to remember

y = 1 => correctly remembered
y = 0 => misremembered

parameter of interest

$$p(y|\theta, d) = \text{Bern}(y|\sigma(\text{offset} - \theta \cdot d))$$

sigmoid function

number of digits in number

$$p(\theta) = \mathcal{N}(\theta|\mu = 0.5, \sigma = 0.1)$$

normal prior

A concrete example

Define $p(y, \theta | d)$

```
1 def model(design, prior_mean=0.5, prior_scale=0.1, offset=3.0):
2         theta = sample("theta", Normal(prior_mean, prior_scale))
3         guess_prob = sigmoid(offset - theta * design)
4         sample("y", Bernoulli(guess_prob))
```

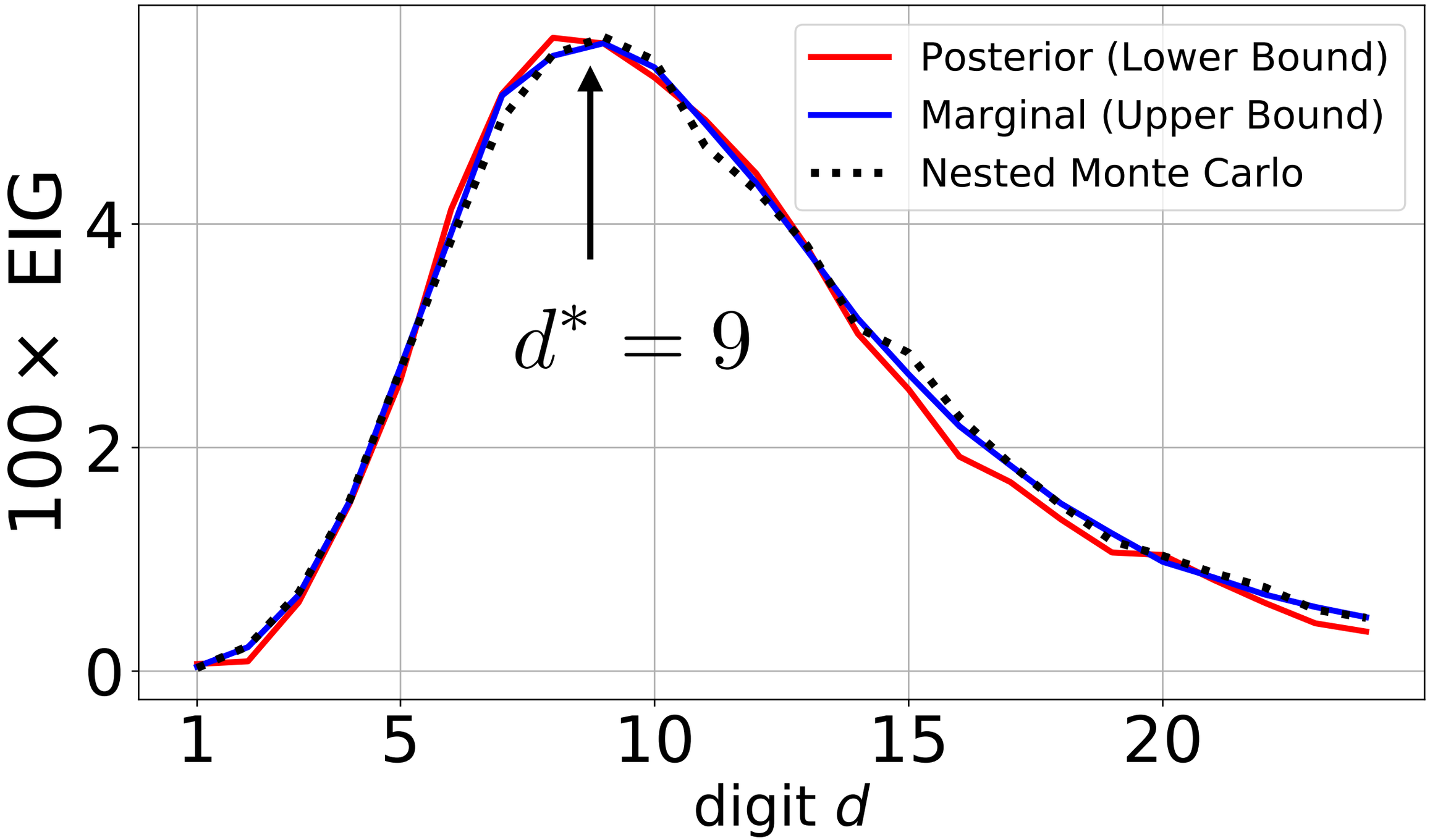
Note that this is a generative model.

Define $q_m(y|d)$

```
1 from pyro.contrib.oed.eig import marginal_eig
2
3 def marginal_guide(design, observation_labels, target_labels):
4     logits = param("logits", zeros(design.shape[-1]))
5     sample("y", Bernoulli(logits=logits))
6
7 eig = marginal_eig(model, design, "y", "theta", num_samples=10,
8                     num_steps=1000, guide=marginal_guide,
9                     optim=Adam({"lr": 0.001}), final_num_samples=10000)
```

Compute $\mathbb{E}_{p(y,\theta|d)}\left[\log \frac{p(y|\theta,d)}{q_m(y|d)}\right]$

EIG estimates



Gravitational Lensing

Differentiable Strong Lensing: Uniting Gravity and Neural Nets through Differentiable Probabilistic Programming

Marco Chianese,[1,*] Adam Coogan,[1,†] Paul Hofma,[1,‡] Sydney Otten,[1,2,§] and Christoph Weniger[1,¶]

[1] *Gravitation Astroparticle Physics Amsterdam (GRAPPA), Institute for Theoretical Physics Amsterdam and Delta Institute for Theoretical Physics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands*

[2] *Institute for Mathematics, Astrophysics and Particle Physics (IMAPP), Radboud University, Heyendaalseweg 135, 6525 AJ Nijmegen, The Netherlands*

(Dated: October 15, 2019)

Gravitational Lensing



Figure 2: Examples of the most common configurations of galaxy-scale gravitational lens systems. A background source (top left) can produce four visible images (a "quad"; top right), an (incomplete) Einstein ring (bottom left), or two visible images (a "double"; bottom right), depending on the ellipticity of the projected mass distribution of the deflector and on the relative alignment between source and deflector (data from Moustakas et al. 2007, Image courtesy of P. Marshall).

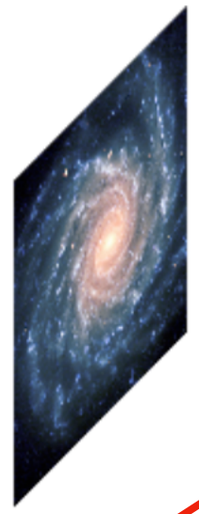
a tool for understanding the spatial distribution of mass in the universe!

e.g. useful for probing dark matter halos

Gravitational Lensing

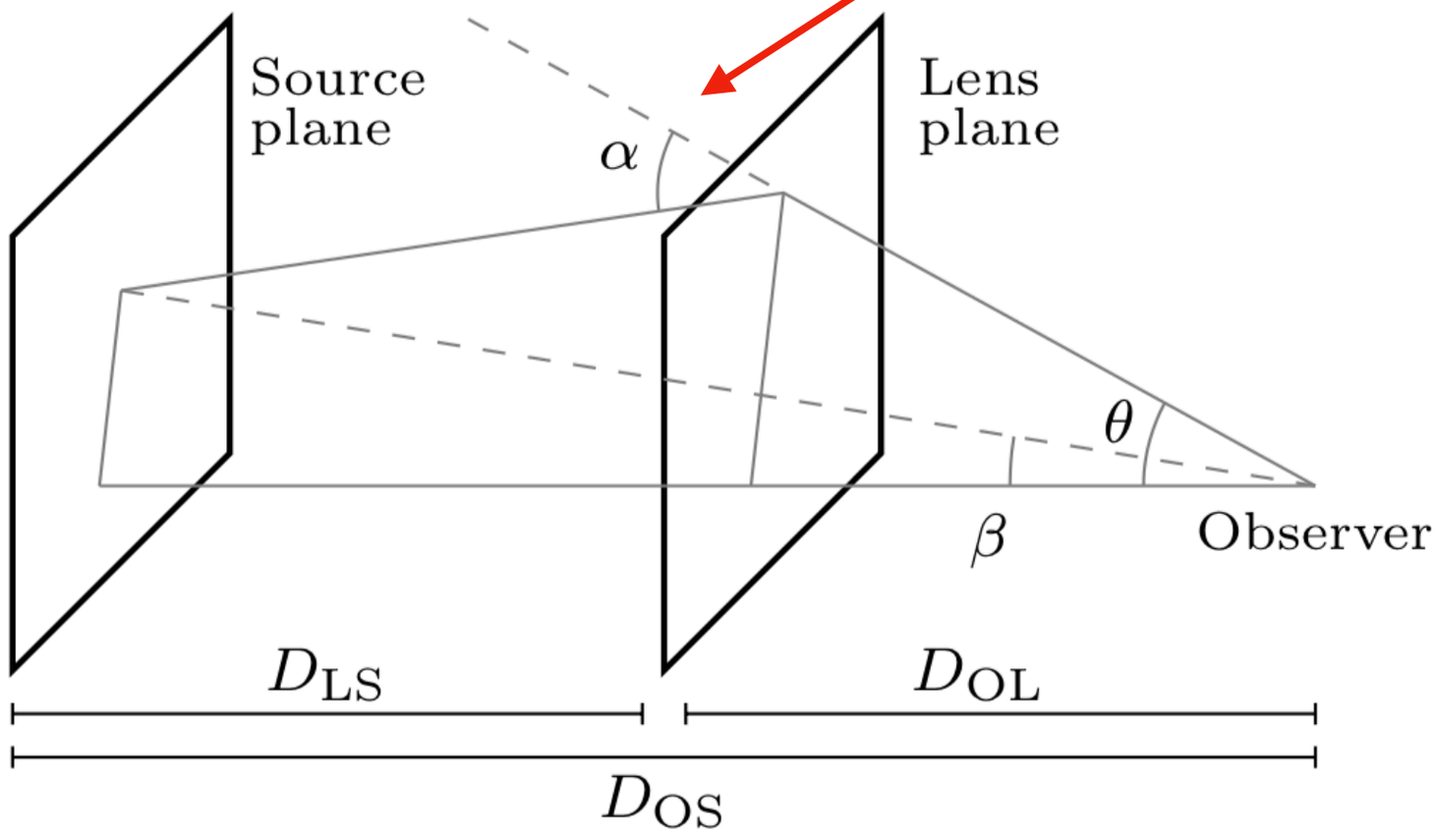
source galaxy





lens galaxy

displacement field $\boldsymbol{\alpha}$



Gravitational Lensing

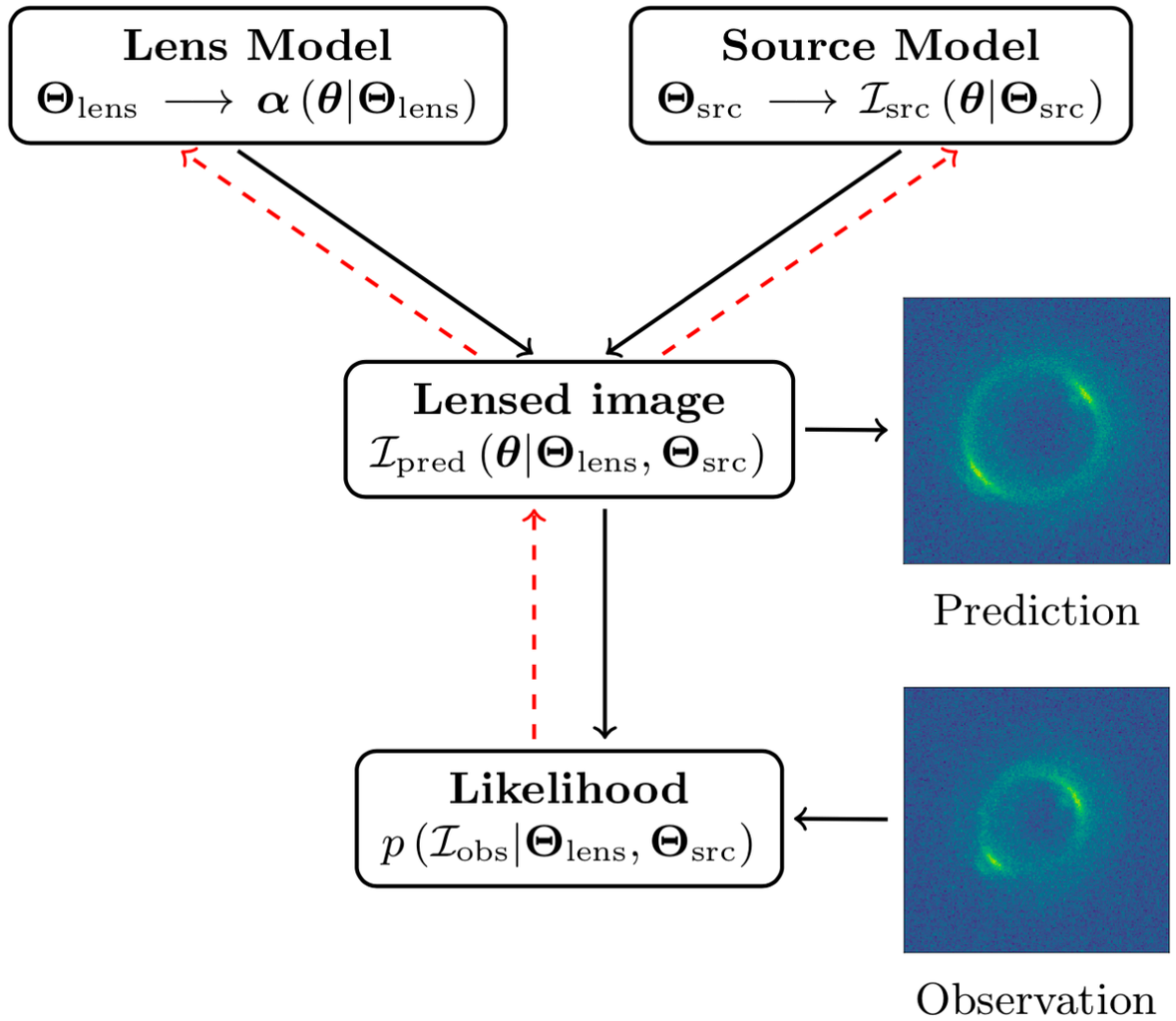


FIG. 5. **Lensing Pipeline.** The forward flow (black solid lines) estimates the likelihood by comparing the observation with the predicted lensed image. This is obtained through the Lens Model and the Source Model. The backward flow (red dashed lines) computes the derivatives of the likelihood with respect to all the parameters of the models, $\boldsymbol{\Theta}_{\rm lens}$ and $\boldsymbol{\Theta}_{\rm src}$. Each box represents an independent module.

Lens Model

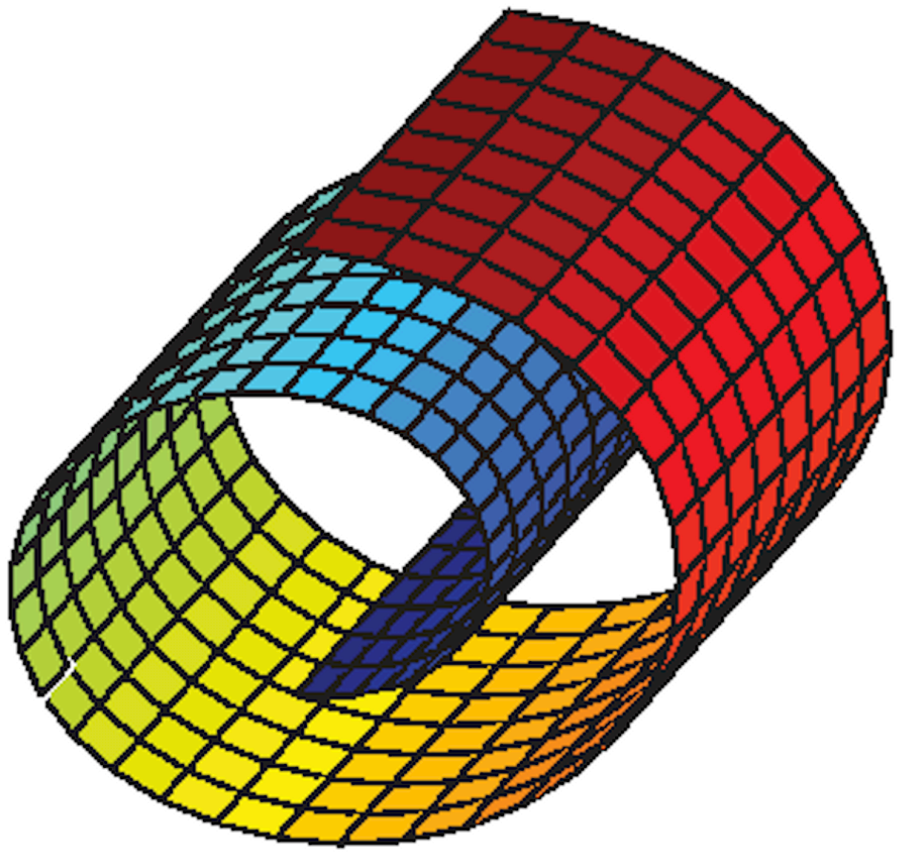
The displacement field α is a differentiable function of its inputs.

$$\boldsymbol{\alpha} = \frac{4\,G}{c^2}\frac{D_{\mathrm{OL}}D_{\mathrm{LS}}}{D_{\mathrm{OS}}}\int \Sigma(\boldsymbol{\theta}')\frac{\boldsymbol{\theta}-\boldsymbol{\theta}'}{|\boldsymbol{\theta}-\boldsymbol{\theta}'|^2}d^2\theta'$$

lens surface mass density

Source Model

Idea: While images of galaxies live in some pixel space of large (say width x height) dimensions, the *effective dimension* is much lower.



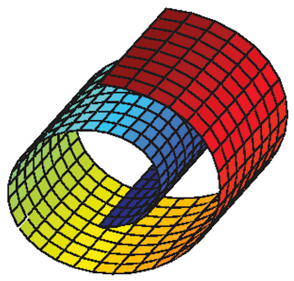
Source Model

So we use a **latent variable model** defined by the following generative process:

Sample a (low-dimensional) latent code $\mathbf{z} \sim p(\mathbf{z})$

Sample a (high-dimensional) image given the latent code $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z})$

$$p(\mathbf{x}) = \int d\mathbf{z}\; p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$



Variational Autoencoders

```
1 decoder = nn.Sequential(nn.Linear(20, 50), ...)
2
3 def model():
4     z = pyro.sample("z", Normal(torch.zeros(20), 1.0))
5     pyro.module("decoder", decoder)
6     mu_x, sigma_x = decoder(z)
7     return pyro.sample("image", Normal(mu_x, sigma_x))
```

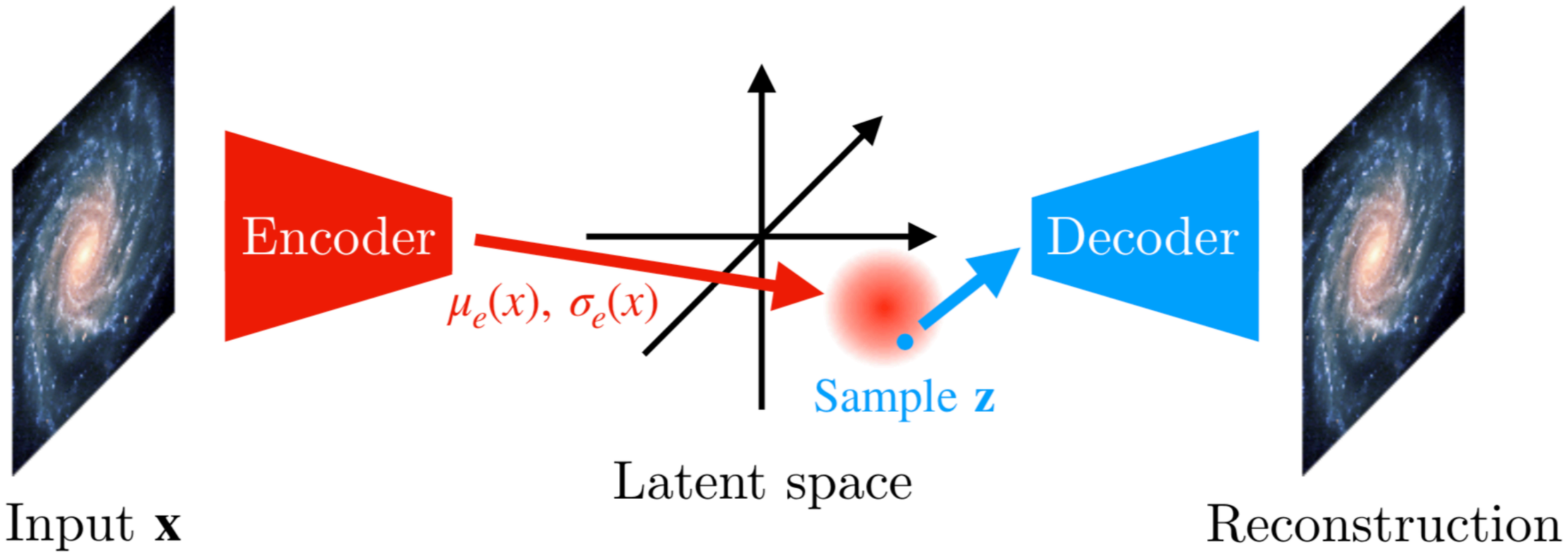
Line 1: define a decoder network
Line 4: sample latent code from a Normal prior
Line 5: register decoder network with Pyro
Line 6: pass sampled z through decoder network
Line 7: sample image given distribution specified by mean mu_x and standard deviation sigma_x

Variational Autoencoders

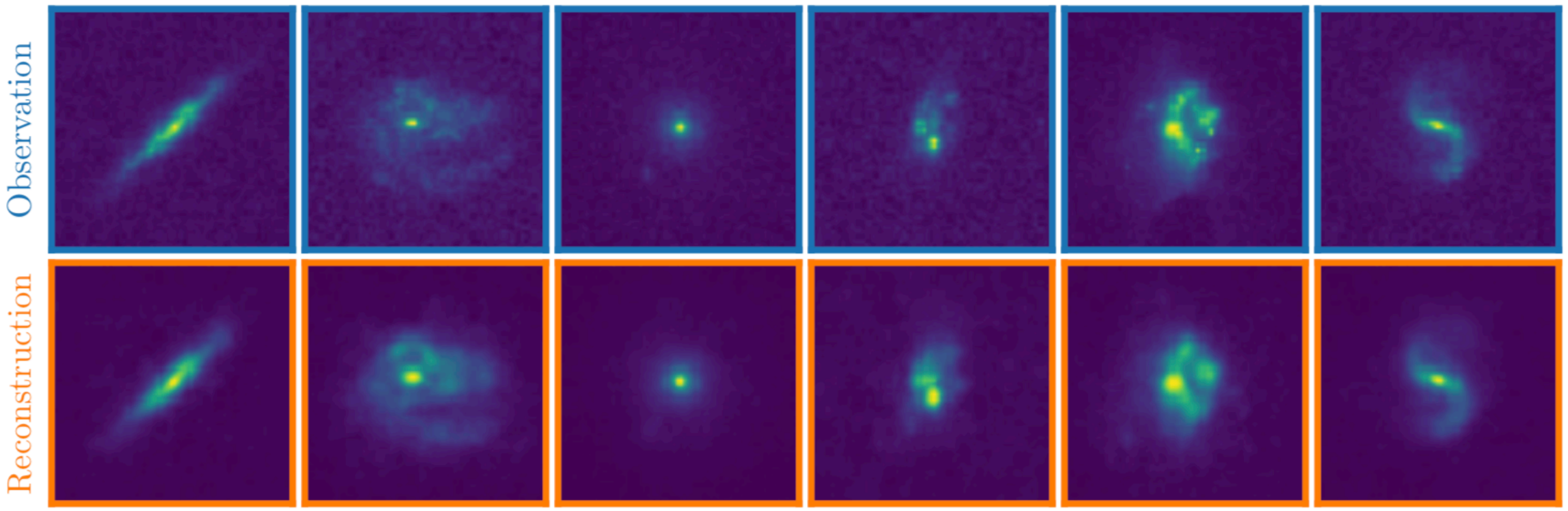
```
1 encoder = nn.Sequential(nn.Linear(64 * 64, 50), ...)
2
3 def guide(observed_image):
4     pyro.module("encoder", encoder)
5     mu_z, sigma_z = encoder(observed_image)
6     return pyro.sample("z", Normal(mu_z, sigma_z))
```

Line 1: define an encoder network
Line 4: register encoder network with Pyro
Line 5: pass observed image through encoder network
Line 6: sample latent code given distribution specified by mu_z and sigma_z

Variational Autoencoders

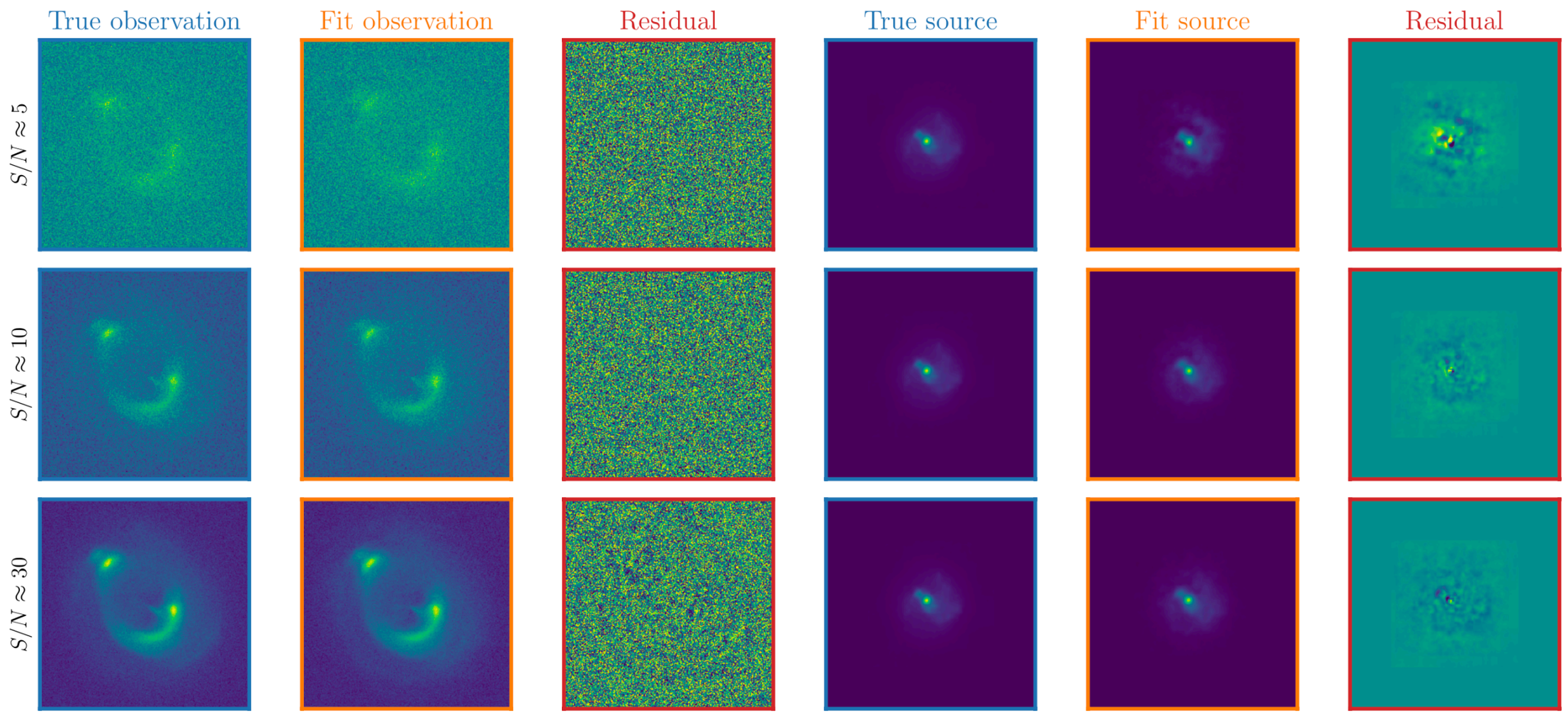


Source Model

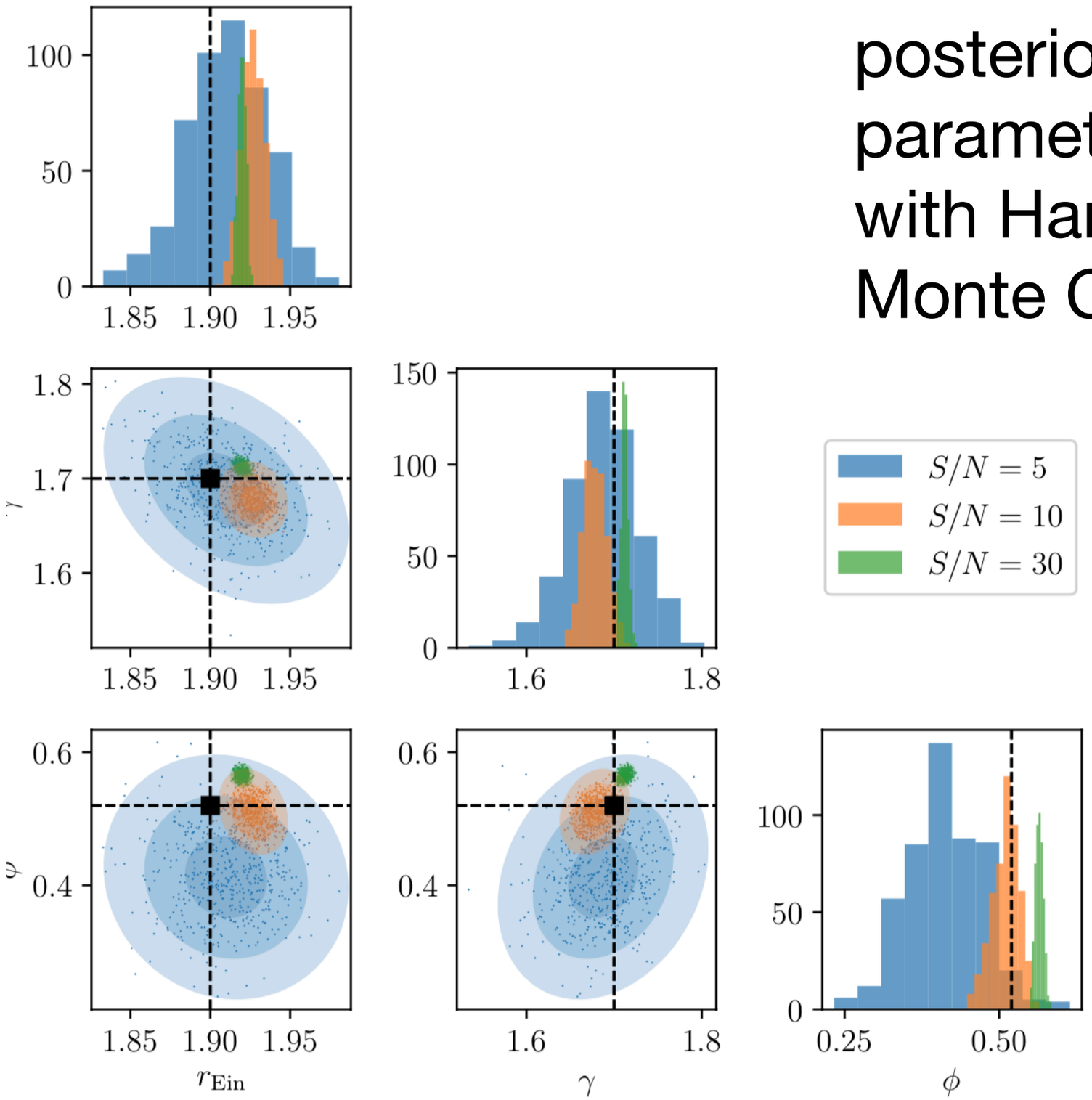


trained on (unsupervised) dataset
of observed images of galaxies taken
with the Hubble telescope

Mock Data Analysis



Mock Data Analysis



posterior on lens parameters obtained with Hamiltonian Monte Carlo

Conclusions

- PPLs are a powerful tool for specifying complex probabilistic models
- As PPLs mature, we might expect them to play an ever more important role in scientific modeling and data analysis
- PPLs are an exciting area of research, combining aspects of programming languages, Bayesian inference, deep learning, …

References

- **“The Design and Implementation of Probabilistic Programming Languages,”** Noah D. Goodman and Andreas Stuhlmüller. http://dippl.org
- **“An Introduction to Probabilistic Programming,”** Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, Frank Wood
- **“Probabilistic machine learning and artificial intelligence,”** Zoubin Ghahramani.

For **Pyro** tutorials and docs visit **https://pyro.ai**

