Advances in Probabilistic Programming with Python

PyData San Luis 2017

Christopher Fonnesbeck
Department of Biostatistics
Vanderbilt University Medical Center

Probabilistic Programming

Stochastic language "primitives"

Distribution over values:

```
X ~ Normal(μ, σ)
x = X.random(n=100)
```

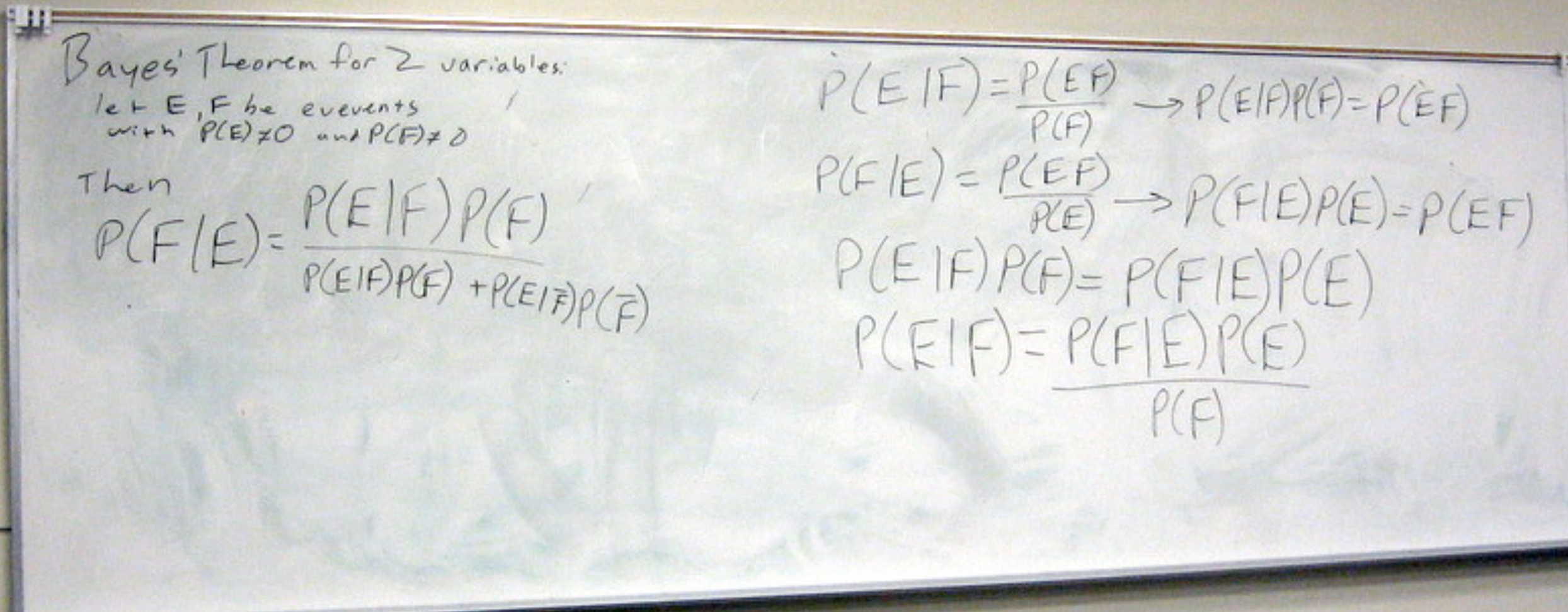
Distribution over functions:

```
Y ~ GaussianProcess(mean_func(x), cov_func(x))
y = Y.predict(x2)
```

Conditioning:

```
p ~ Beta(1, 1)
z ~ Bernoulli(p) # z|p
```

Bayesian Inference



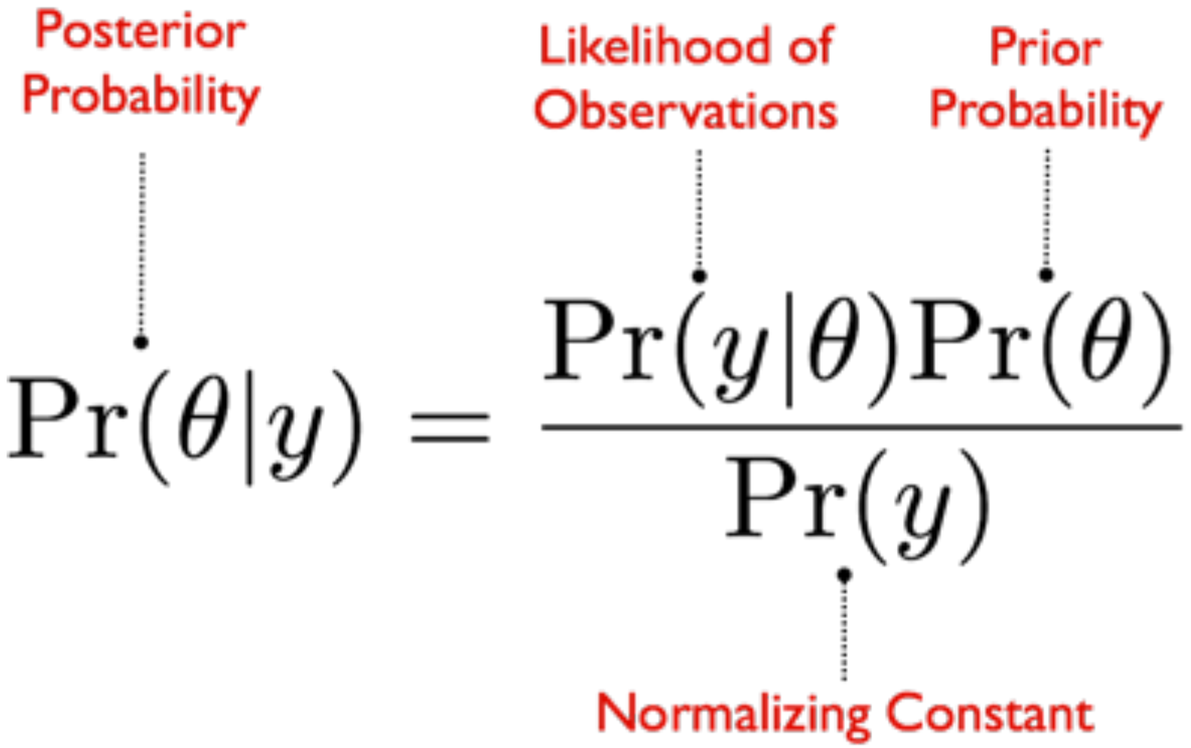
Inverse Probability

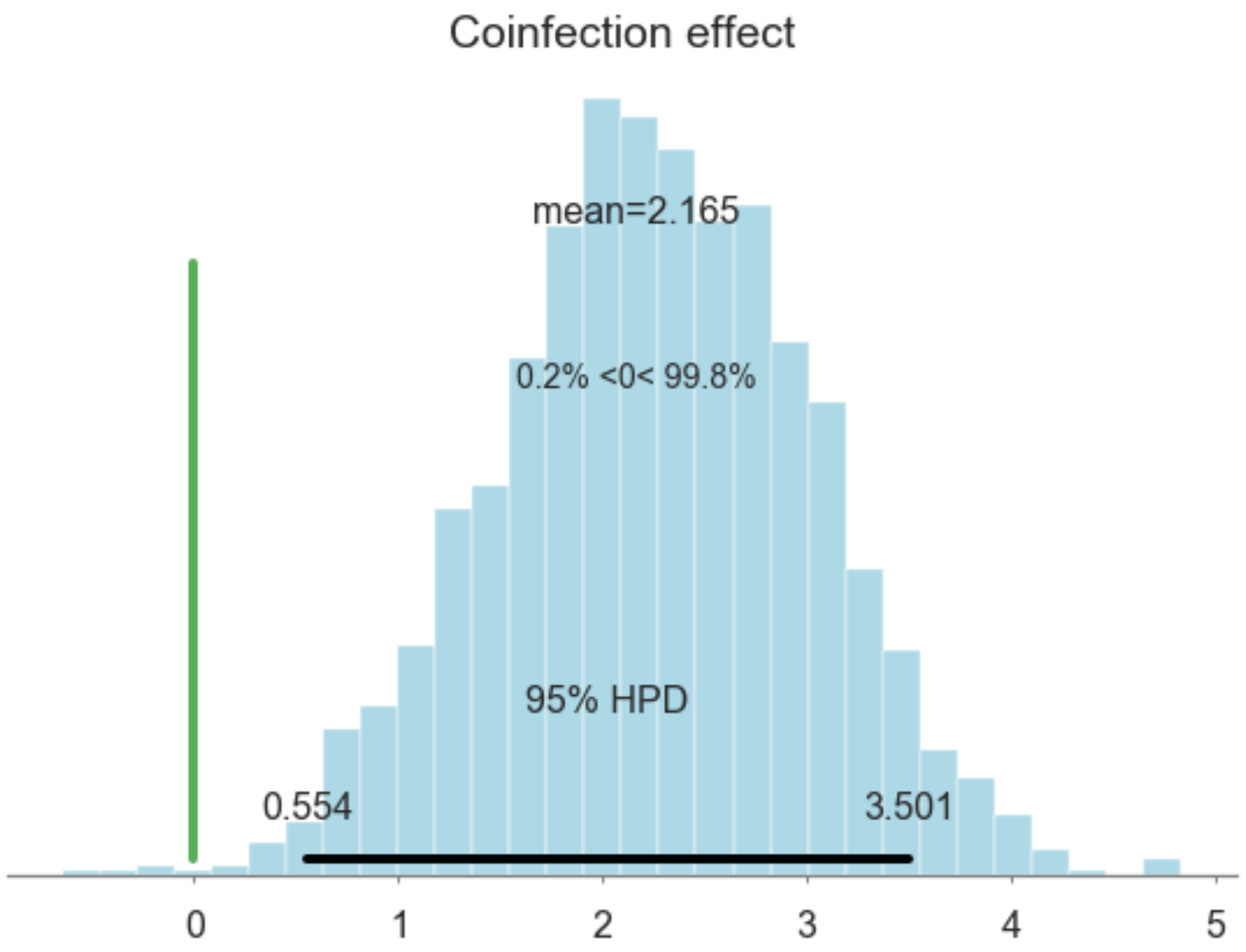
$$Pr(\theta|y)$$

Why Bayes?

“The Bayesian approach is attractive because it is **useful**. Its usefulness derives in large measure from its simplicity. Its simplicity allows the investigation of **far more complex models** than can be handled by the tools in the classical toolbox.”

—Link and Barker 2010


Posterior Probability
Likelihood of Observations
Prior Probability
$\Pr(\theta|y) = \frac{\Pr(y|\theta)\Pr(\theta)}{\Pr(y)}$
Normalizing Constant


Coinfection effect
mean=2.165
0.2% <0< 99.8%
95% HPD
0.554
3.501
0
1
2
3
4
5

Stochastic program

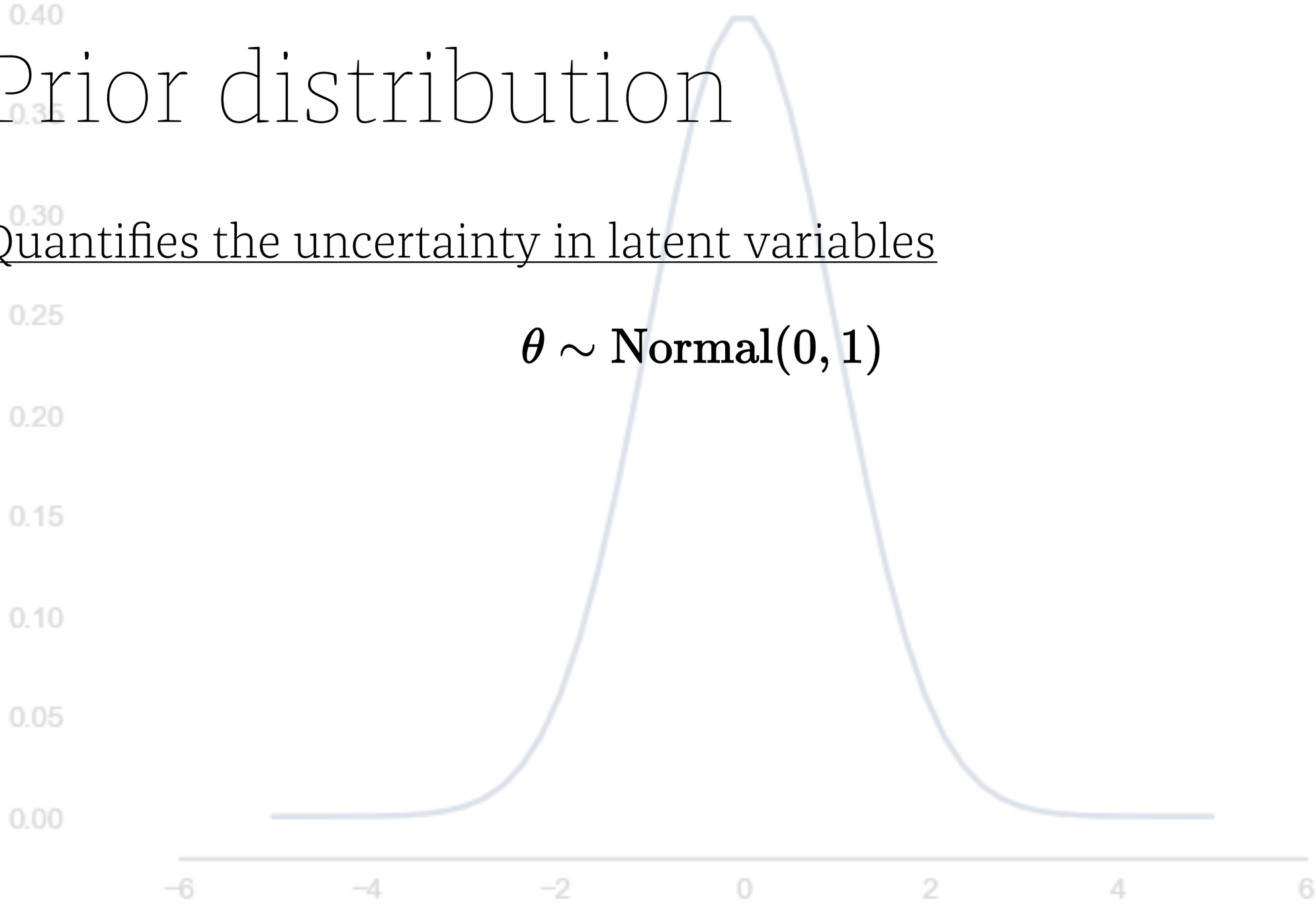
Joint distribution of latent variables and data

$$Pr(\theta, y) = Pr(y|\theta)Pr(\theta)$$

Prior distribution

Quantifies the uncertainty in latent variables

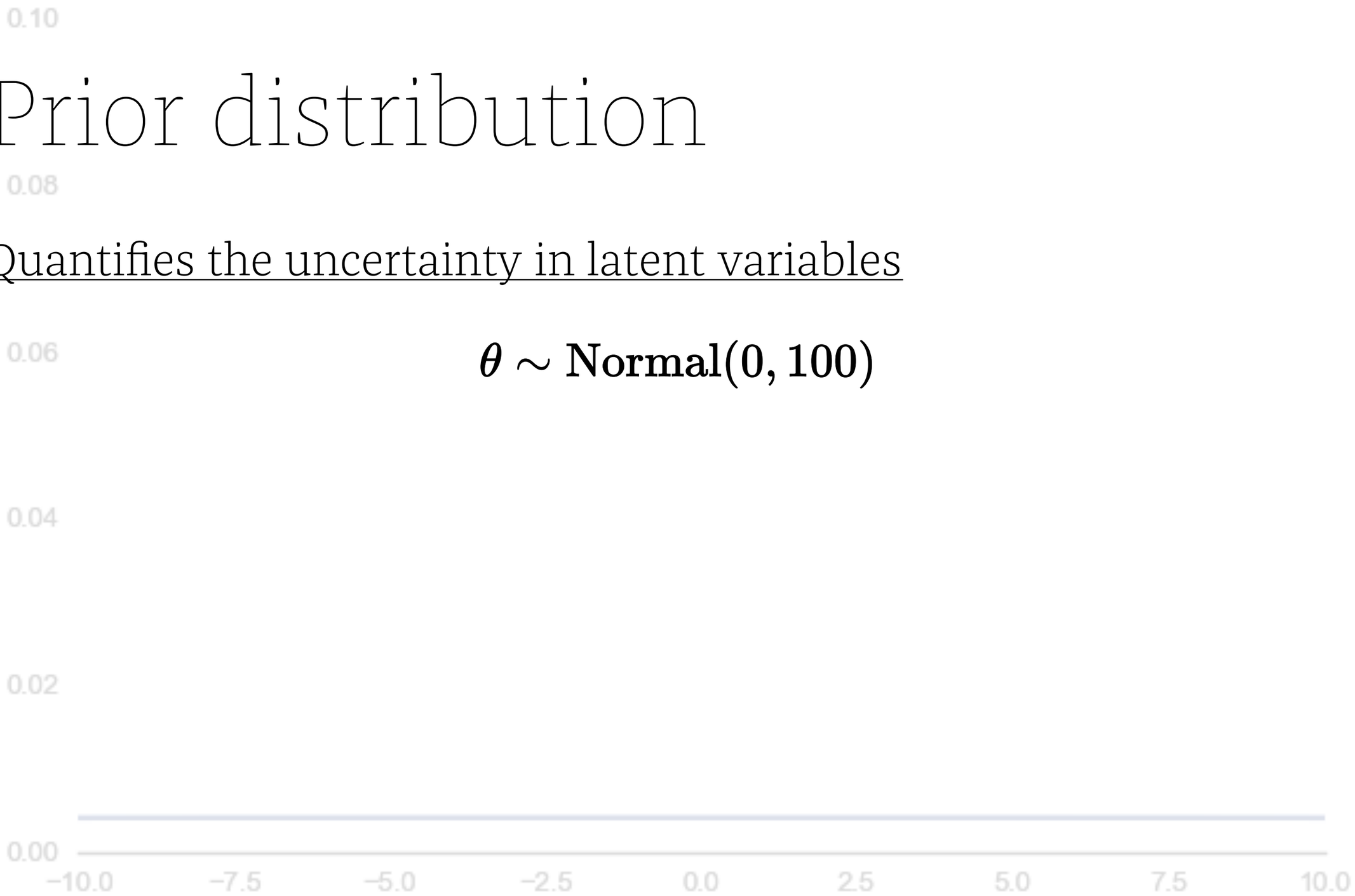
$$\theta \sim \mathbf{Normal}(0, 1)$$



Prior distribution

Quantifies the uncertainty in latent variables

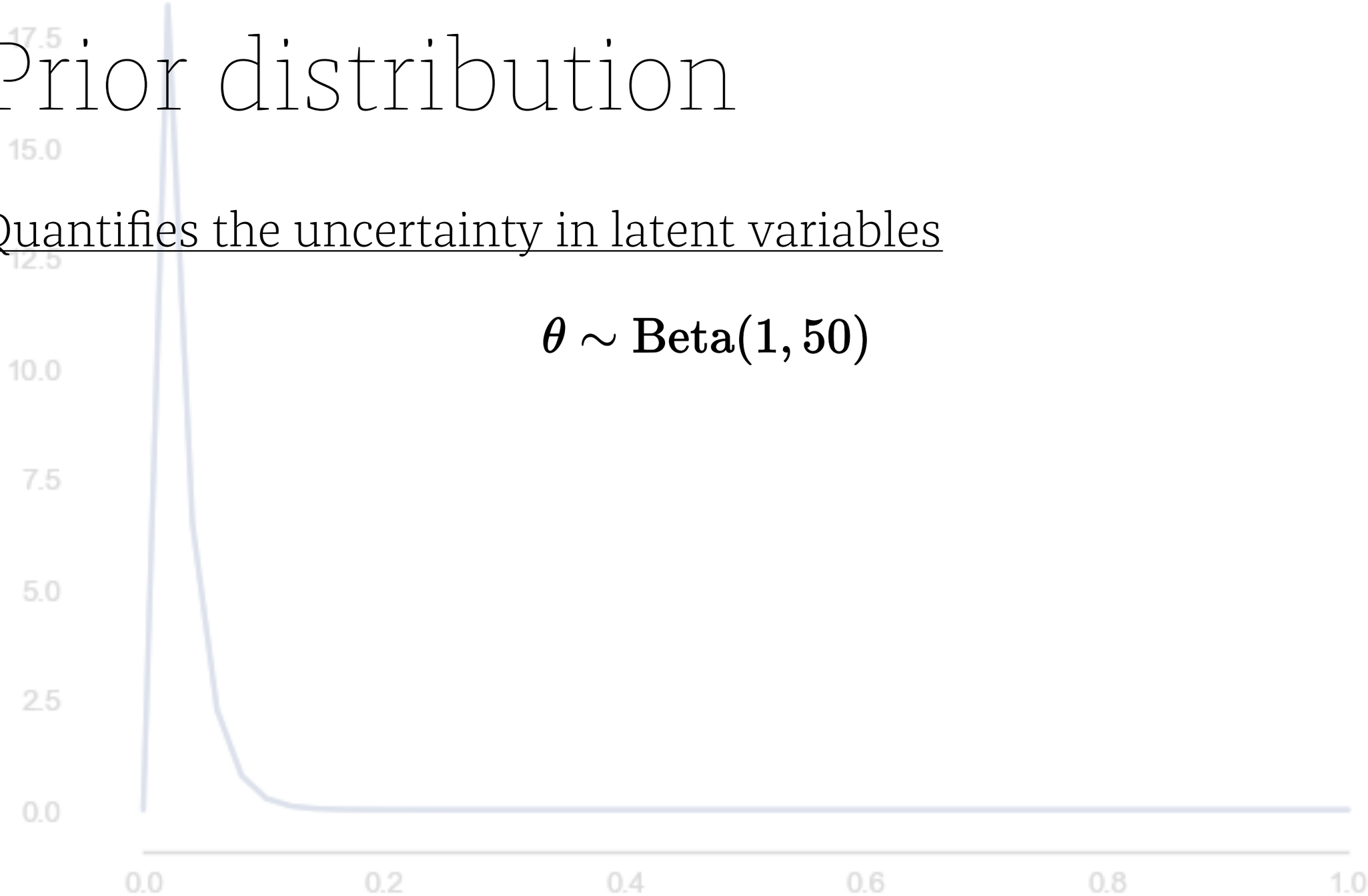
$$\theta \sim \mathbf{Normal}(0, 100)$$



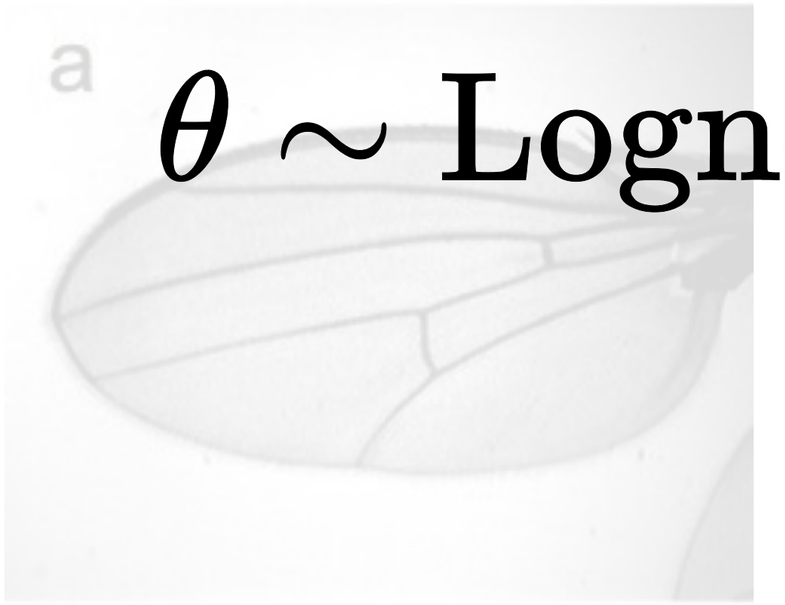
Prior distribution

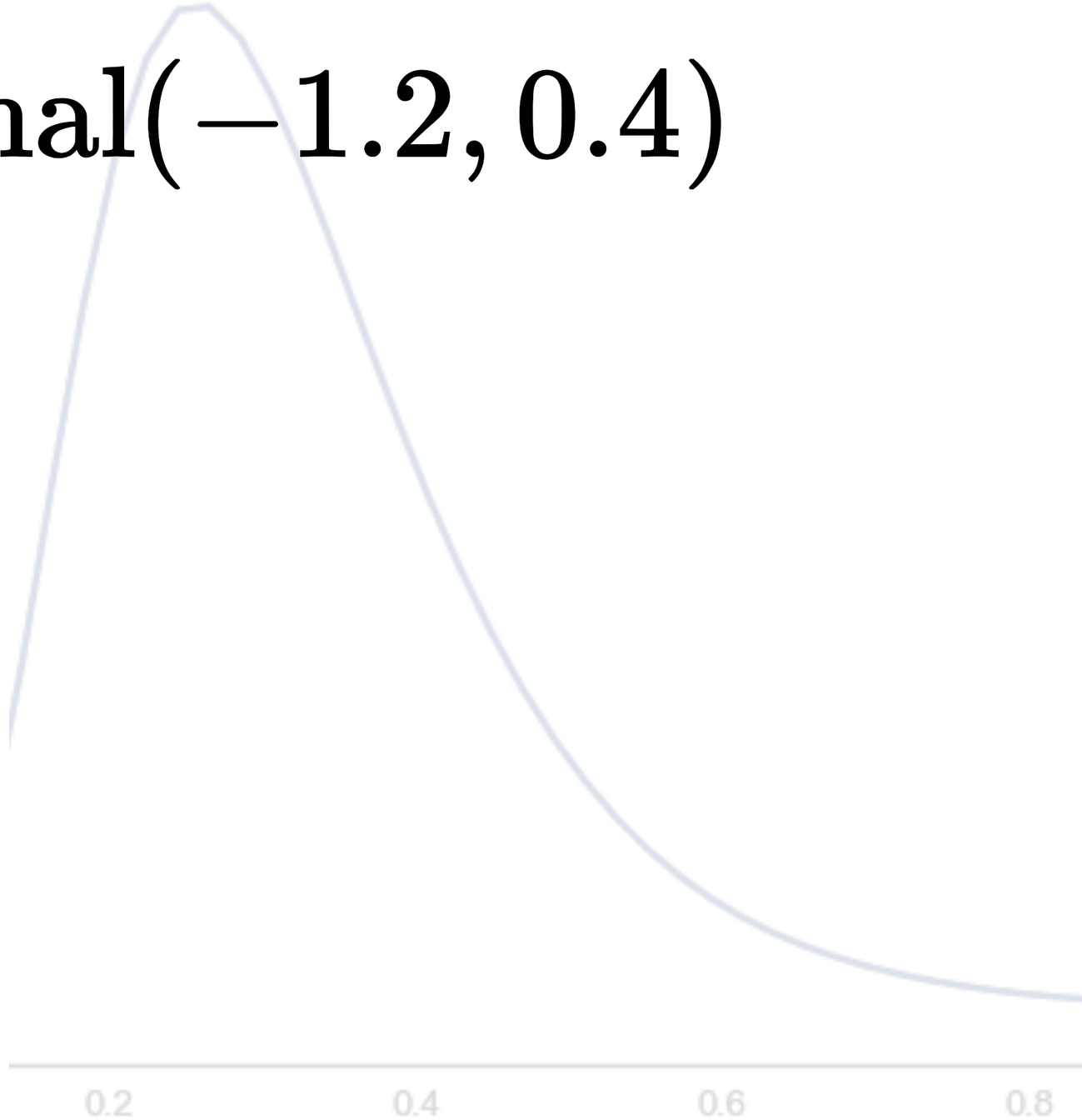
Quantifies the uncertainty in latent variables

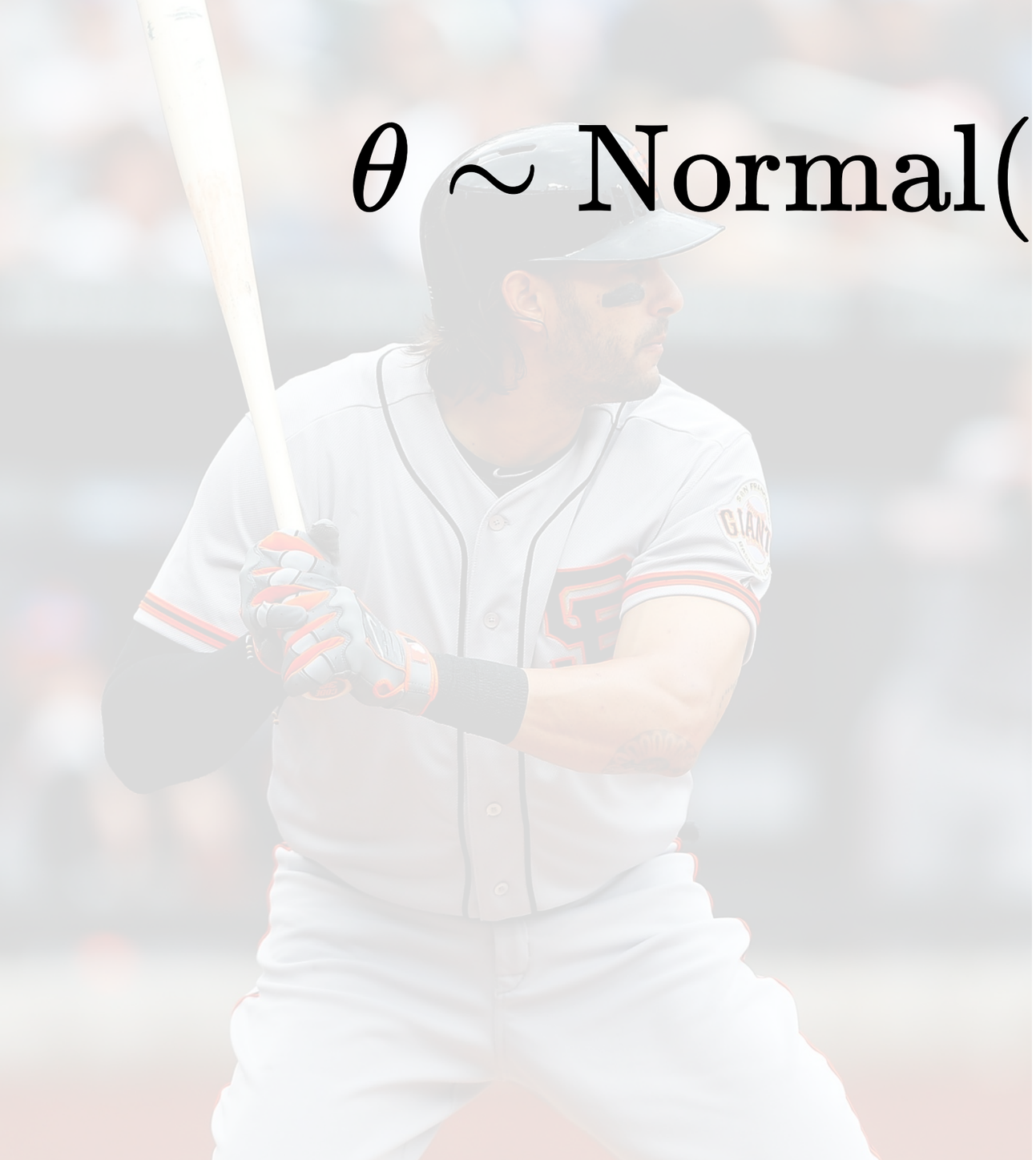
$$\theta \sim \mathbf{Beta}(1, 50)$$



$$\theta \sim \text{Lognormal}(-1.2, 0.4)$$







$$\theta \sim \text{Normal}(0.261, 0.034)$$

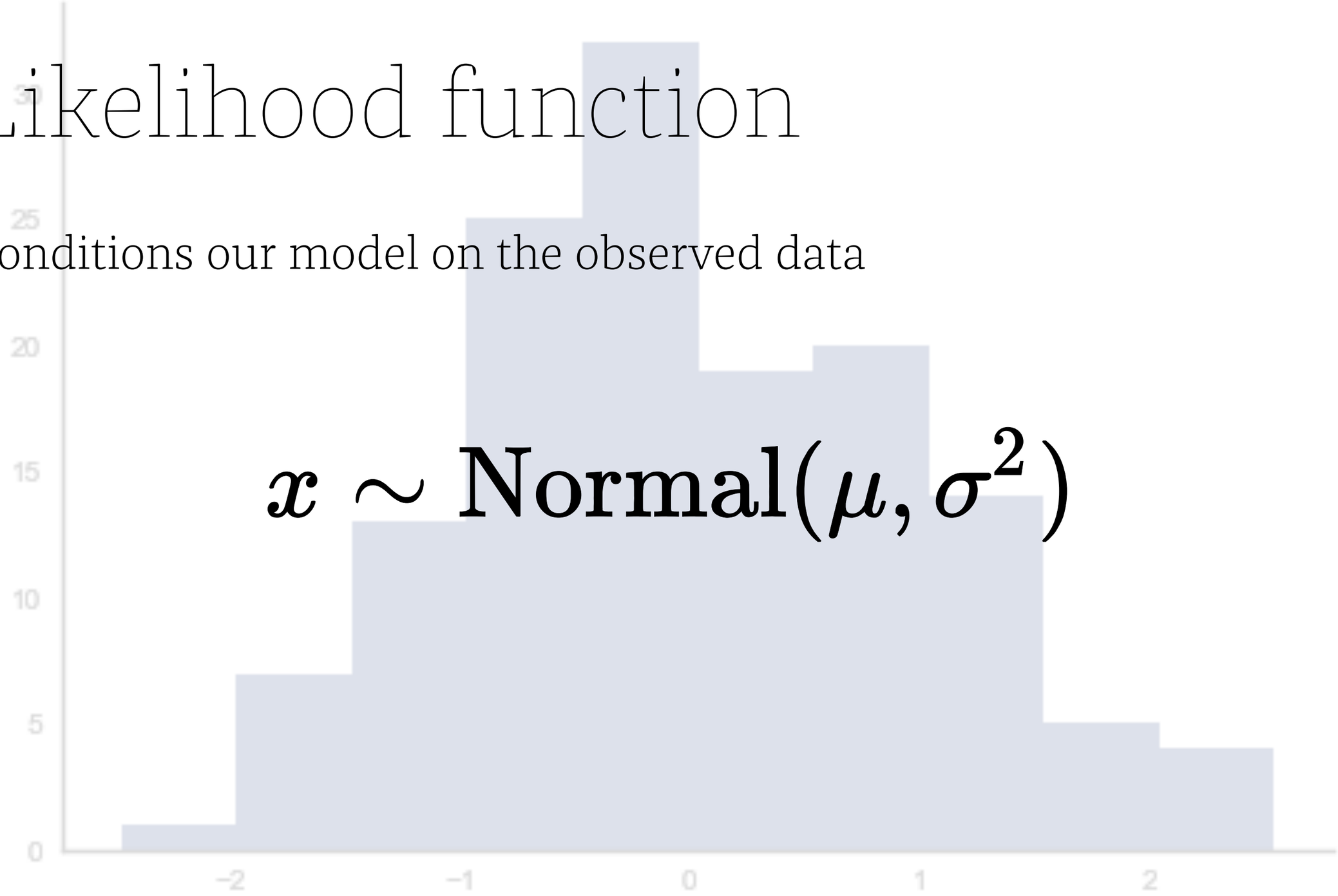
Likelihood function

Conditions our model on the observed data

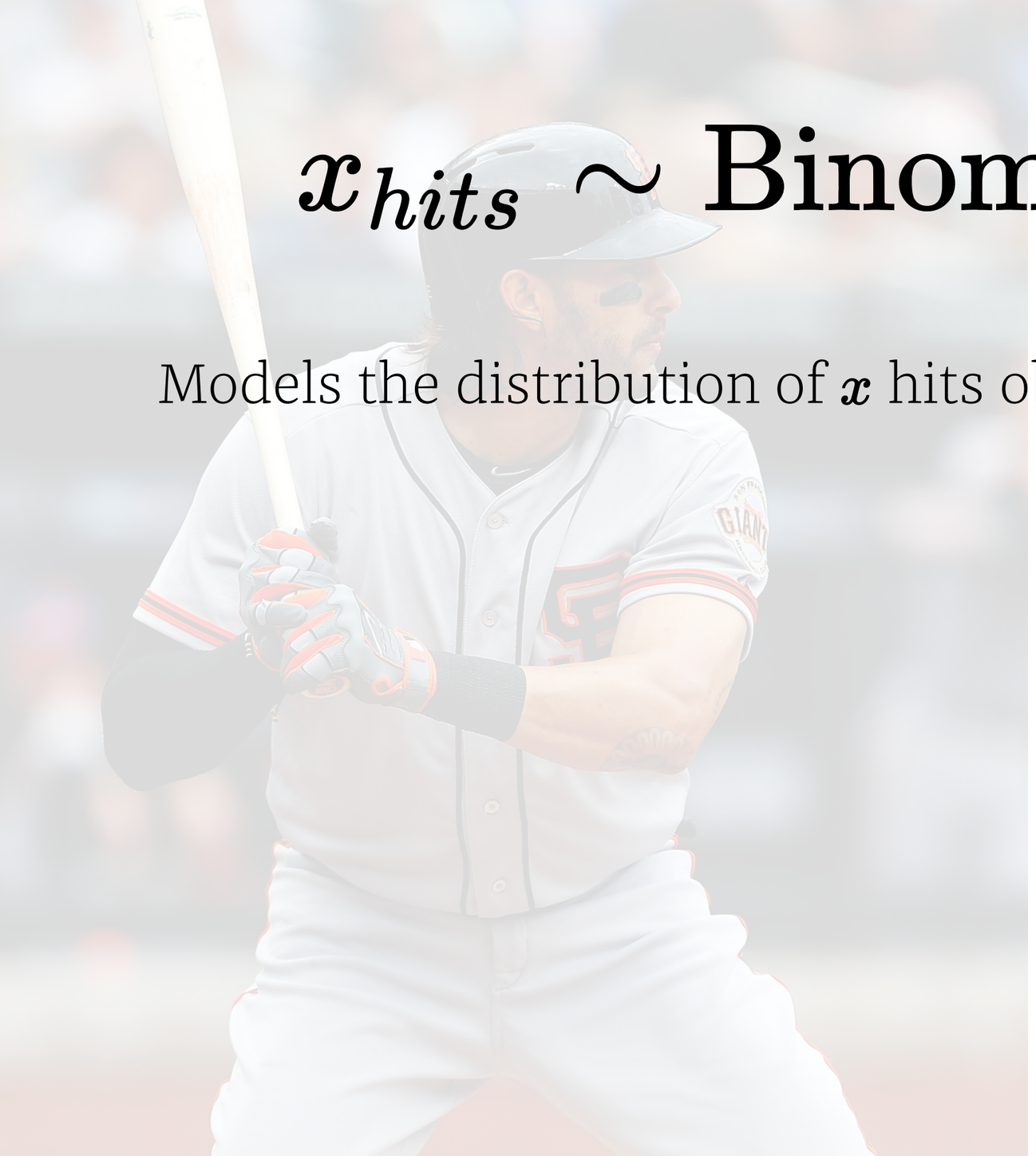
$$Pr(y|\theta)$$

Likelihood function

Conditions our model on the observed data

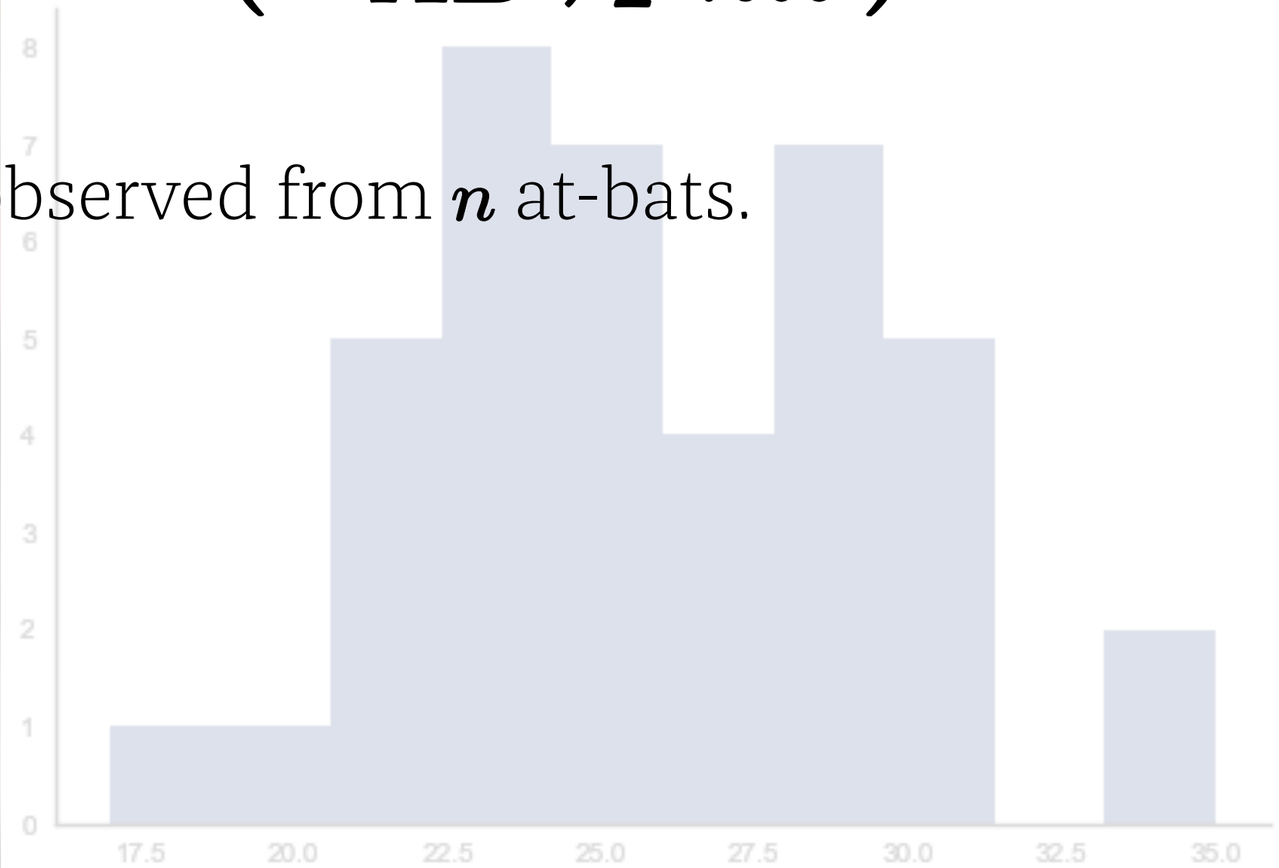


$$x \sim \mathbf{Normal}(\mu, \sigma^2)$$



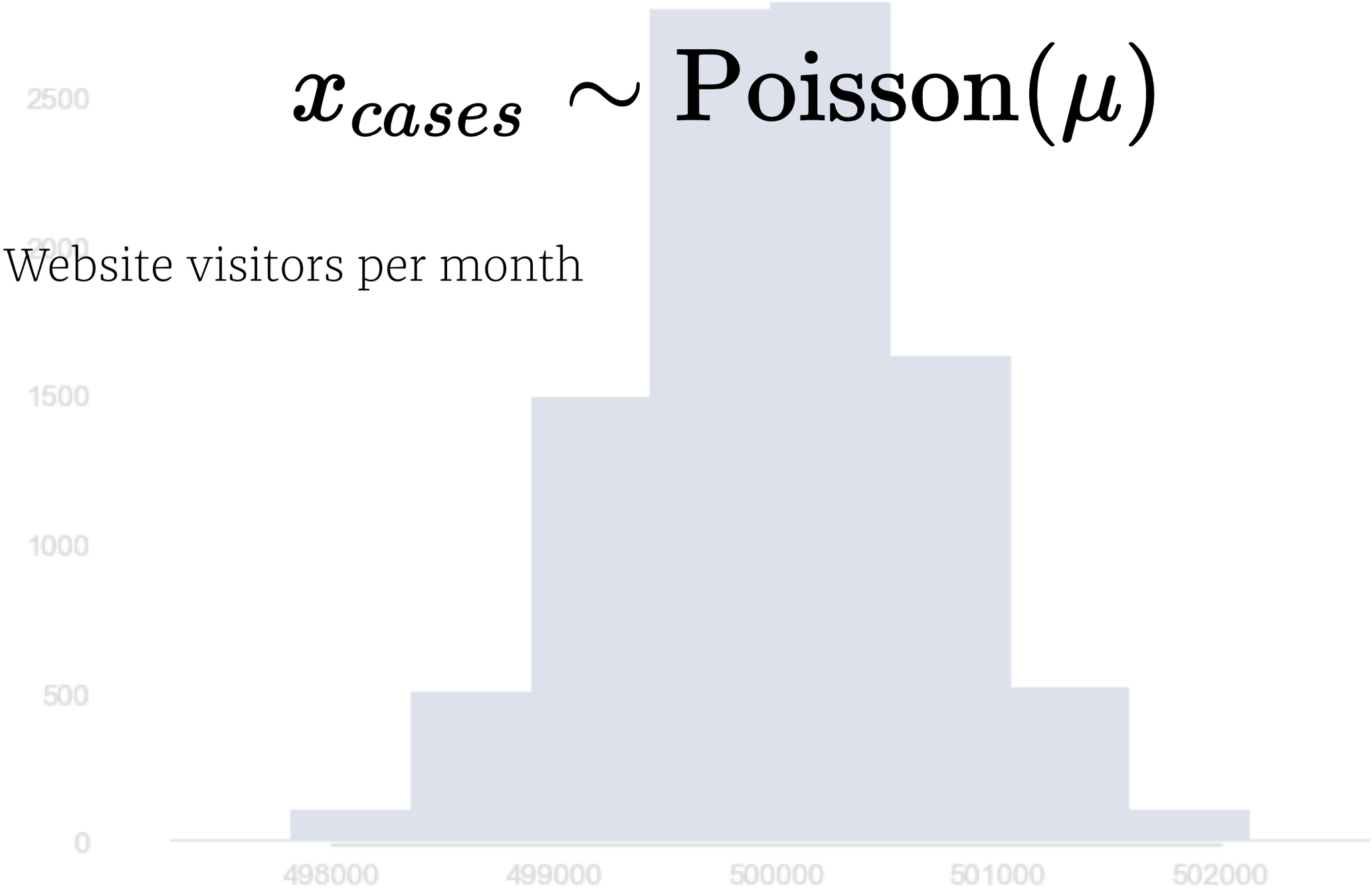
$$x_{hits} \sim \text{Binomial}(n_{AB}, p_{hit})$$

Models the distribution of x hits observed from n at-bats.



$$x_{cases} \sim \text{Poisson}(\mu)$$

Website visitors per month



Infer Values for latent variables

Calculate the posterior distribution

$$Pr(\theta|y) \propto Pr(y|\theta)Pr(\theta)$$

Posterior distribution

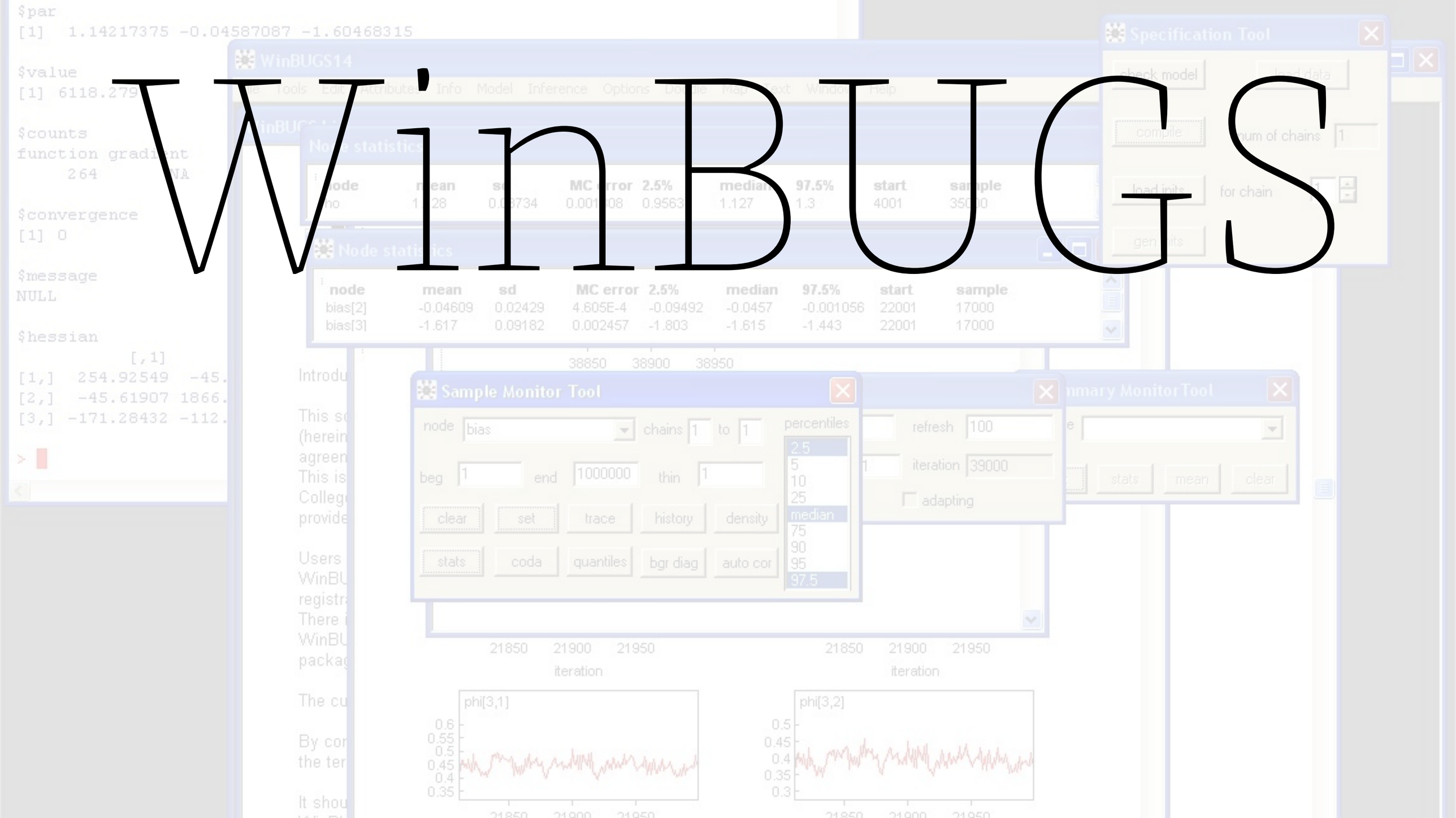
$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{Pr(y)}$$

Posterior distribution

$$Pr(\theta|y) = \frac{Pr(y|\theta)Pr(\theta)}{\int_\theta Pr(y|\theta)Pr(\theta)d\theta}$$

Probabilistic programming **abstracts** the inference procedure

WinBUGS


$par
[1] 1.14217375 -0.04587087 -1.60468315
$value
[1] 6118.279
$counts
function gradient
264 NA
$convergence
[1] 0
$message
NULL
$hessian
[,1]
[1,] 254.92549 -45.
[2,] -45.61907 1866.
[3,] -171.28432 -112.
WinBUGS14
Node statistics
node mean sd MC error 2.5% median 97.5% start sample
bias[2] -0.04609 0.02429 4.605E-4 -0.09492 -0.0457 -0.001056 22001 17000
bias[3] -1.617 0.09182 0.002457 -1.803 -1.615 -1.443 22001 17000
Sample Monitor Tool
node bias chains 1 to 1
beg 1 end 1000000 thin 1
clear set trace history density
stats coda quantiles bgr diag auto cor
percentiles
2.5
5
10
25
median
75
90
95
97.5
Specification Tool
check model
compile
load inits
gen inits
num of chains 1
for chain 1
refresh 100
iteration 39000
adapting
stats
mean
clear
iteration
phi[3,1]
phi[3,2]

```
$par
[1]  1.14217375 -0.04587087 -1.60468315

$value
[1] 6118.279

$counts
function gradient
     264       NA

$convergence
[1] 0

$message
NULL

$hessian
           [,1]
[1,]  254.92549  -45.
[2,]  -45.61907 1866.
[3,] -171.28432 -112.

>
```

WinBUGS14

File Tools Edit Attributes Info Model Inference Options Doodle Map Text Window Help

Node statistics

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
rho	1.128	0.08734	0.001308	0.9563	1.127	1.3	4001	35000

Node statistics

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
bias[2]	-0.04609	0.02429	4.605E-4	-0.09492	-0.0457	-0.001056	22001	17000
bias[3]	-1.617	0.09182	0.002457	-1.803	-1.615	-1.443	22001	17000

Specification Tool

check model | load data | compile | num of chains 1 | load inits | for chain 1 | gen inits

Sample Monitor Tool

node bias | chains 1 to 1 | beg 1 | end 1000000 | thin 1

clear | set | trace | history | density | stats | coda | quantiles | bgr diag | auto cor

percentiles: 2.5, 5, 10, 25, median, 75, 90, 95, 97.5

refresh 100 | iteration 39000 | adapting

stats | mean | clear

38850 38900 38950

21850 21900 21950 iteration

21850 21900 21950 iteration

phi[3,1]

0.6 0.55 0.5 0.45 0.4 0.35

phi[3,2]

0.5 0.45 0.4 0.35 0.3

Introdu

This s
(herein
agreen
This is
Colleg
provide

Users
WinBU
registr
There
WinBU
packag

The cu

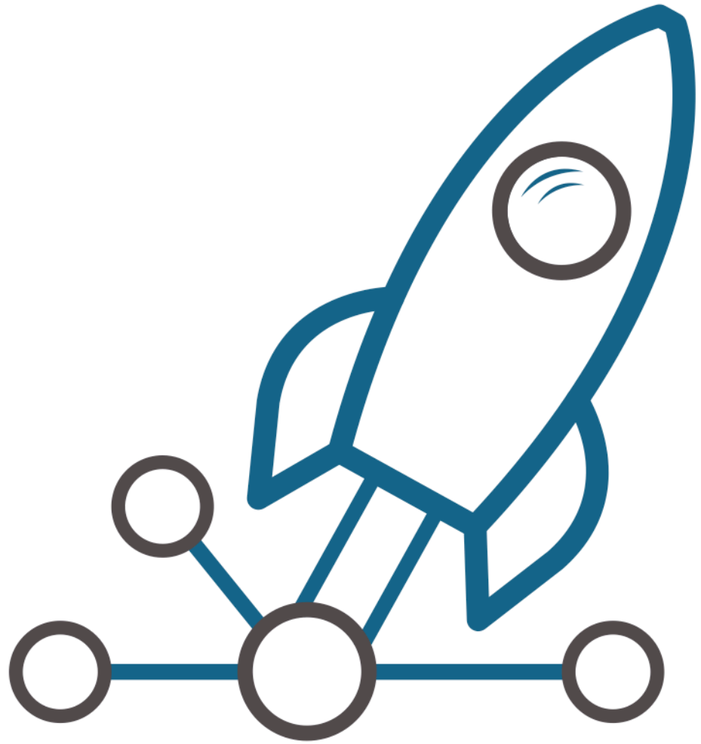
By cor
the ter

It shou

```
model {
      for (j in 1:J){
        y[j] ~ dnorm (theta[j], tau.y[j])
        theta[j] ~ dnorm (mu.theta, tau.theta)
        tau.y[j] <- pow(sigma.y[j], -2)
      }
      mu.theta ~ dnorm (0.0, 1.0E-6)
      tau.theta <- pow(sigma.theta, -2)
      sigma.theta ~ dunif (0, 1000)
    }
```

PyMC3

- ☞ started in 2003
- ☞ PP framework for fitting arbitrary probability models
- ☞ based on Theano
- ☞ implements "next generation" Bayesian inference methods
- ☞ 100 contributors
- ☞ used throughout academia and industry: Quantopian, Monetate, Allianz, GrubHub, Channel 4, VoiceBox, etc.



`github.com/pymc-devs/pymc3`

Salvatier, Wiecki and Fonnesbeck (2016)

NUMFOCUS

OPEN CODE = BETTER SCIENCE

SPONSORED PROJECTS

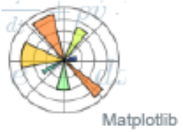

NumPy


jupyter


Open Journals


SymPy


julia


Matplotlib


IPython


ROpenSci

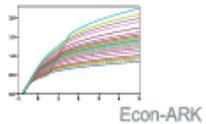

FENICS PROJECT


PyTables


pandas


nteract


QuantEcon


Econ-ARK


AstroPy


Stan


将軍


software carpentry


yt


sunpy


PyMC3


DATA CARPENTRY

Calculating Gradients in Theano

```
>>> from theano import function, tensor as tt
```

Calculating Gradients in Theano

```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
```

Calculating Gradients in Theano

```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
```

Calculating Gradients in Theano

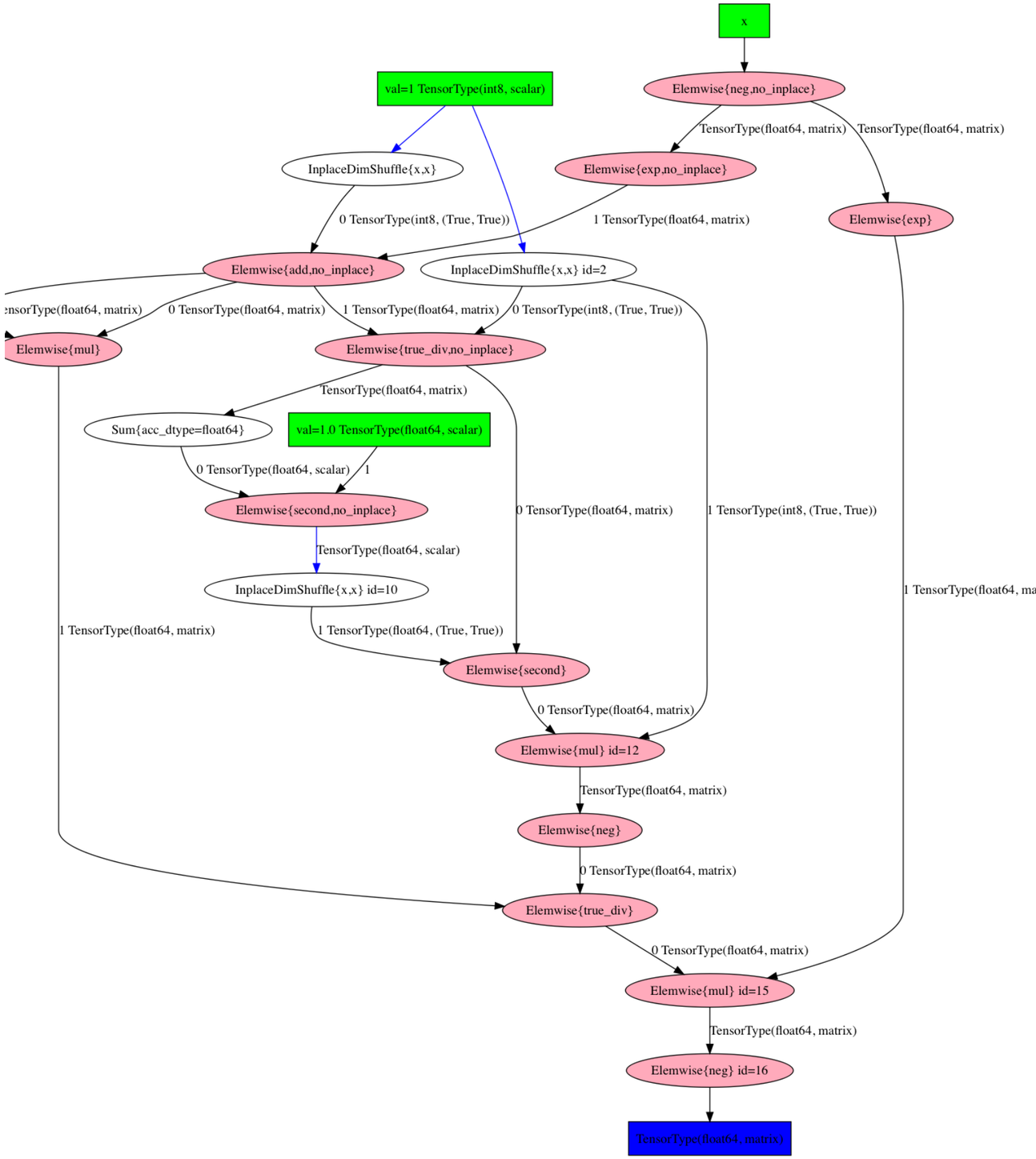
```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
>>> gs = tt.grad(s, x)
```

Calculating Gradients in Theano

```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
>>> gs = tt.grad(s, x)
>>> dlogistic = function([x], gs)
```

Theano graph

```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
>>> gs = tt.grad(s, x)
>>> dlogistic = function([x], gs)
```

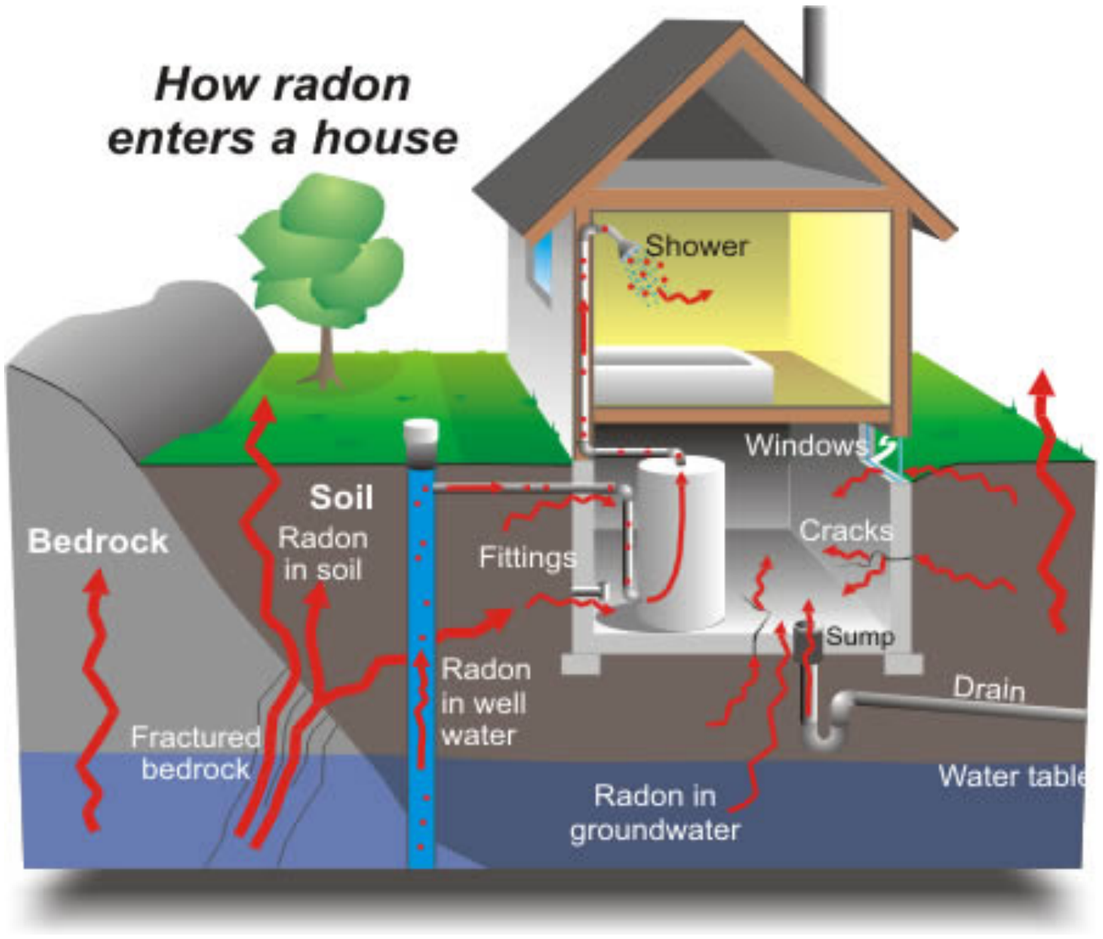


Calculating Gradients in Theano

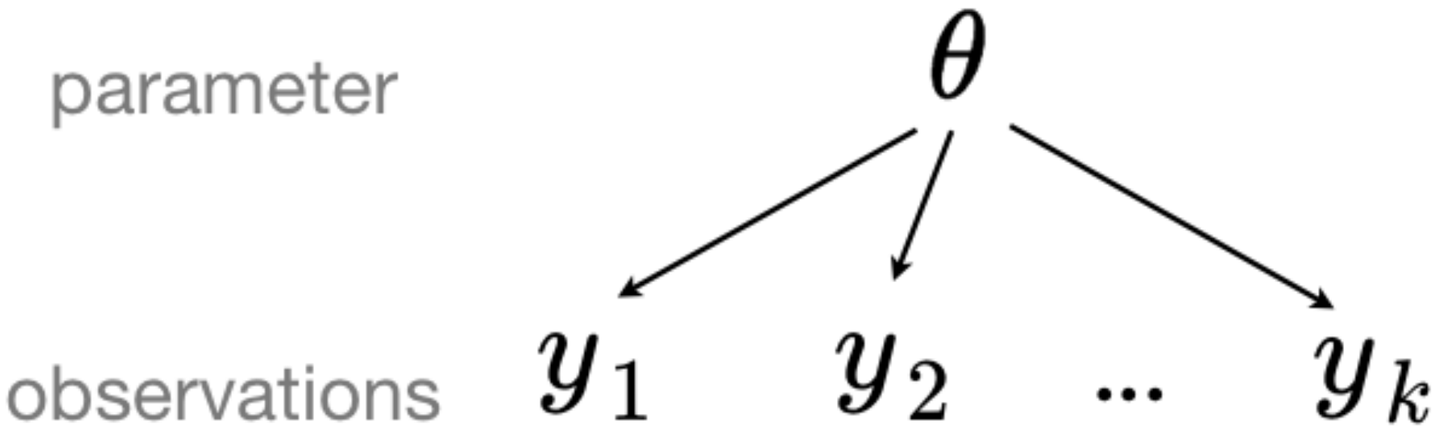
```
>>> from theano import function, tensor as tt
>>> x = tt.dmatrix('x')
>>> s = tt.sum(1 / (1 + tt.exp(-x)))
>>> gs = tt.grad(s, x)
>>> dlogistic = function([x], gs)
>>> dlogistic([[3, -1],[0, 2]])
array([[ 0.04517666,  0.19661193],
       [ 0.25      ,  0.10499359]])
```

Example: Radon exposure*

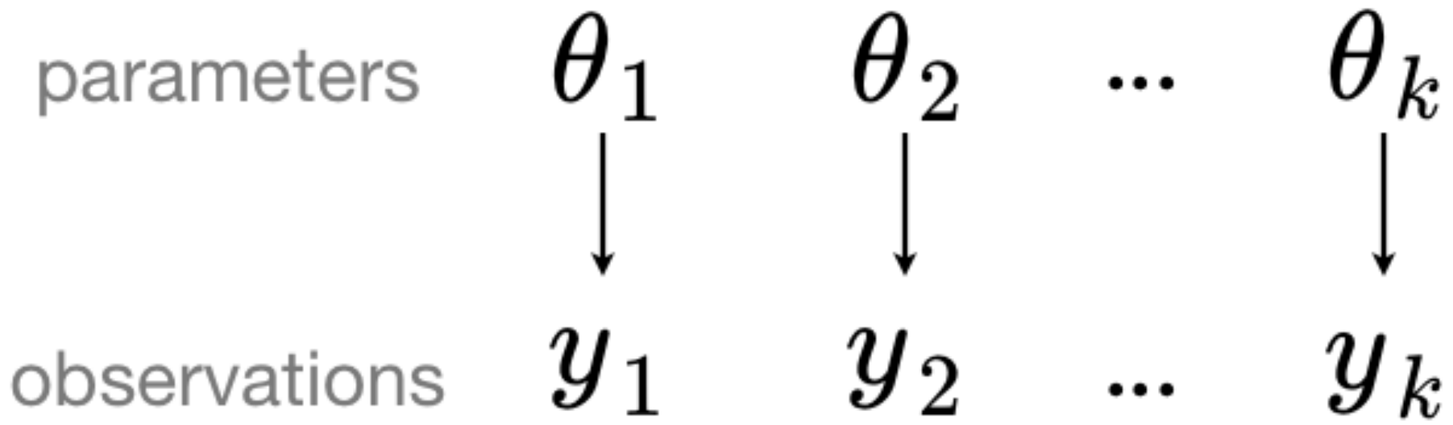
* Gelman et al. (2013) Bayesian Data Analysis

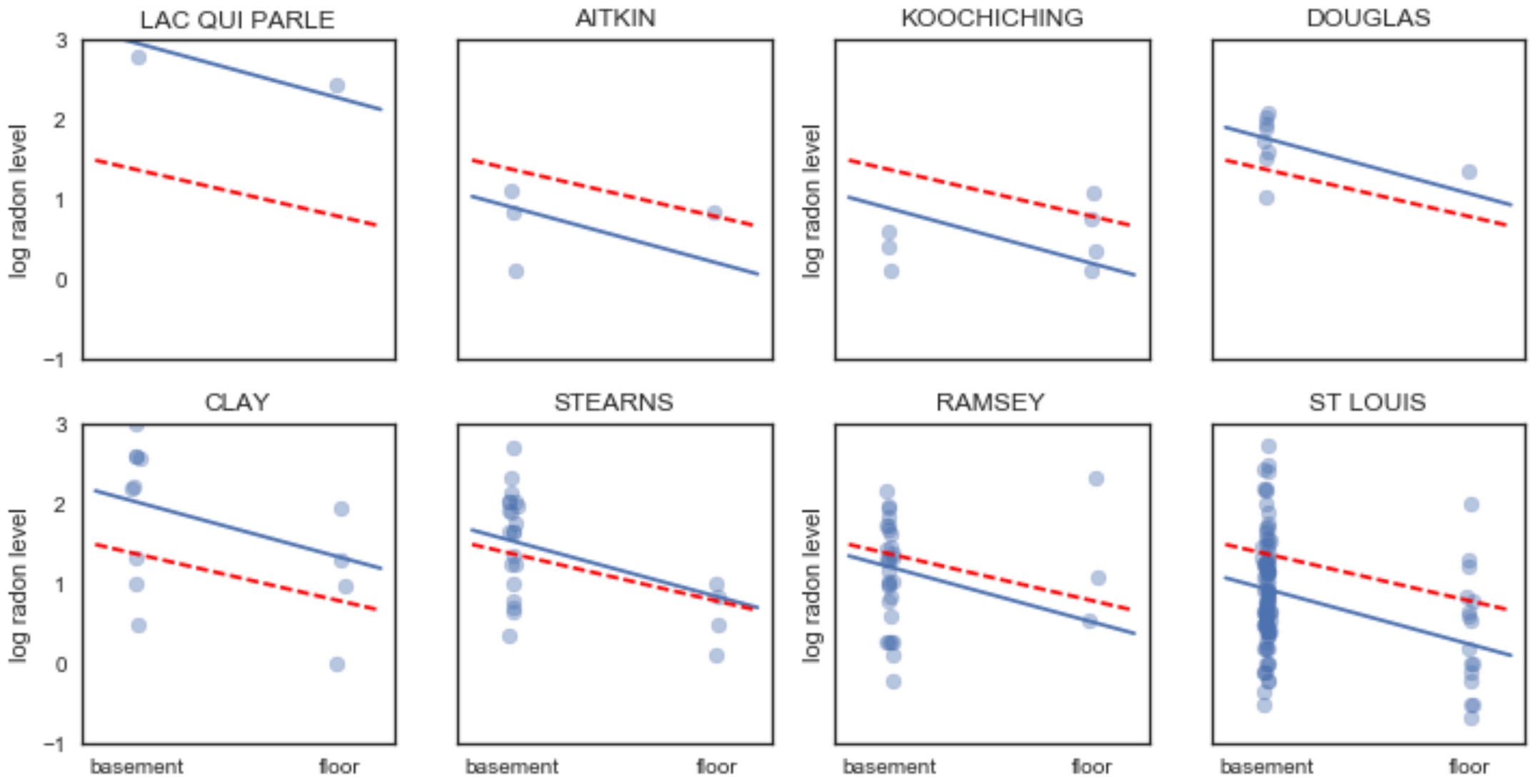

How radon enters a house
Shower
Windows
Soil
Radon in soil
Bedrock
Fittings
Cracks
Sump
Radon in well water
Drain
Fractured bedrock
Water table
Radon in groundwater

Pooled model

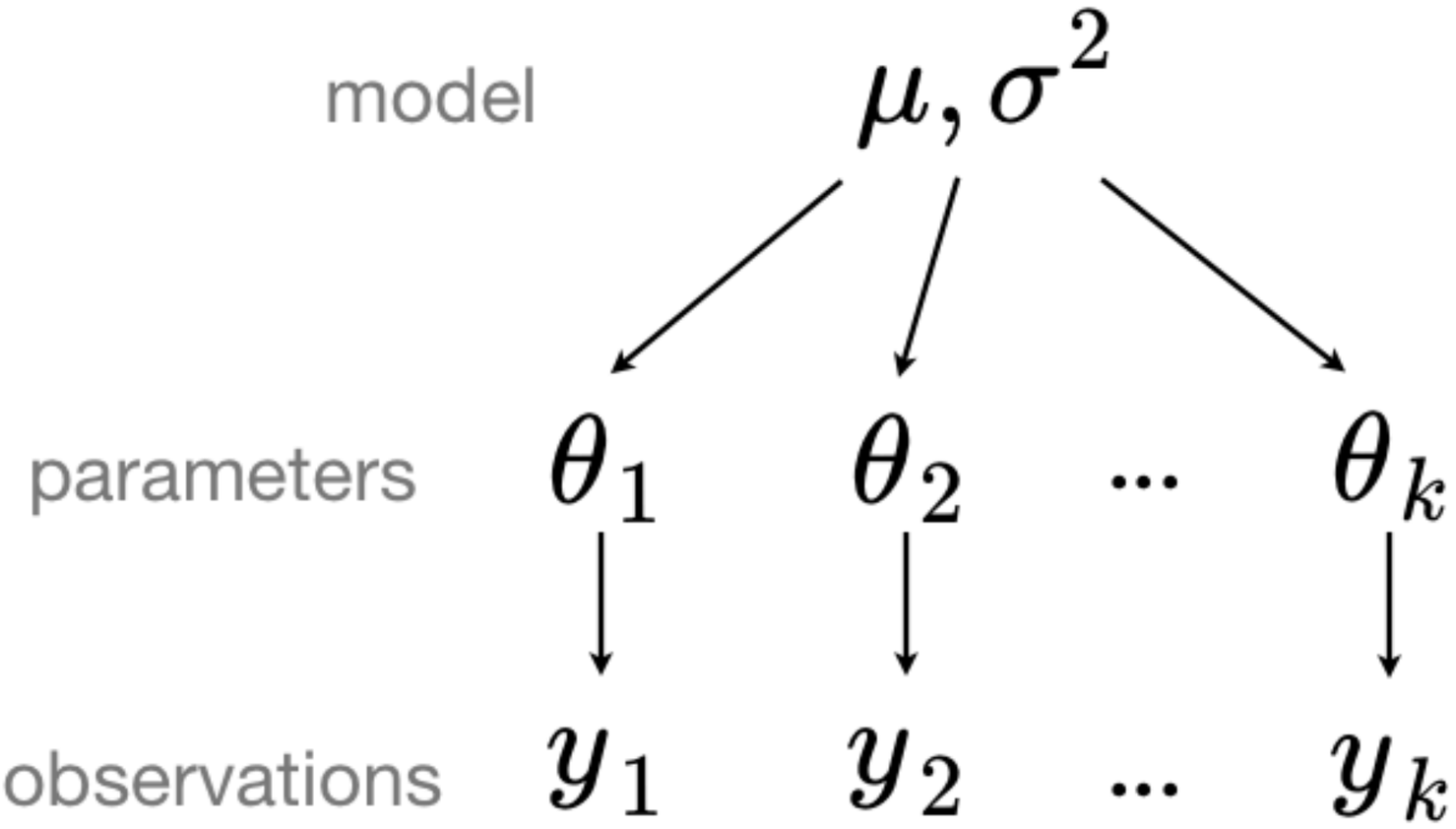


Unpooled model




LAC QUI PARLE
AITKIN
KOOCHICHING
DOUGLAS
CLAY
STEARNS
RAMSEY
ST LOUIS
log radon level
3
2
1
0
−1
basement
floor

Hierarchical model



Hierarchical model

Partially pool radon values among counties.

$$y_i = \alpha_{j[i]} + \beta x_i + \epsilon_i$$

$$\alpha_j \sim N(\mu, \tau)$$

$$\beta \sim N(0, 1000)$$

$$\epsilon_i \sim N(0, \sigma)$$

where $j = 1, \ldots, 85$ (counties)

Hierarchical random effect

```
with Model() as hierarchical_model:

    # Hierarchical priors
    μ = Normal('μ', mu=0., sd=1e5)
    τ = HalfCauchy('τ', 5)

    # Random effect for intercepts
    α = Normal('α', mu=μ, sd=τ, shape=counties)

    # Floor effect
    β = Normal('β', 0, sd=1e5)
```

```
>>> type(β)
pymc3.model.FreeRV
```

```
>>> type(β)
pymc3.model.FreeRV
>>> β.distribution.logp(-2.1).eval()
array(-12.431863999839544)
```

```
>>> type(β)
pymc3.model.FreeRV
>>> β.distribution.logp(-2.1).eval()
array(-12.431863998395 4)
>>> β.random(size=4)
array([ -10292.91760326,   22368.53416626,
         124851.2516102,   44143.62513182]])
```

Transformed variables

```
with hierarchical_model:

    θ = α[county] + β*floor
```

Likelihood

```
with hierarchical_model:

    # Prior on observation error
    σ = HalfCauchy('σ', 5)

    # Sampling distribution of data
    y = Normal('y', θ, sd=σ, observed=log_radon)
```

Calculating Posteriors

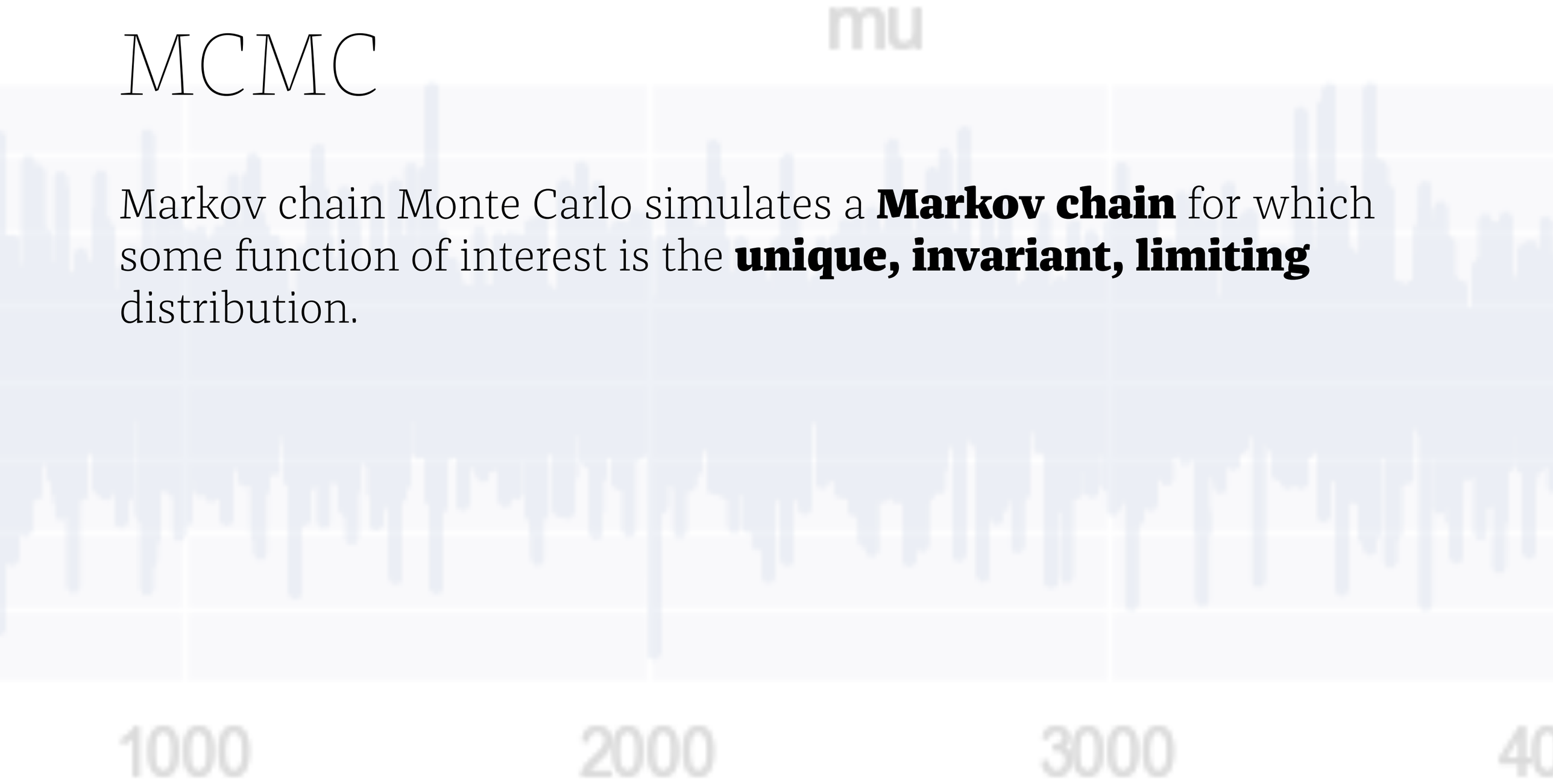
$$Pr(\theta|y) \propto Pr(y|\theta)Pr(\theta)$$

Bayesian approximation

- ☞ Maximum a posteriori (MAP) estimate
- ☞ Laplace (normal) approximation
- ☞ Rejection sampling
- ☞ Importance sampling
- ☞ Sampling importance resampling (SIR)
- ☞ Approximate Bayesian Computing (ABC)

MCMC

Markov chain Monte Carlo simulates a **Markov chain** for which some function of interest is the **unique, invariant, limiting** distribution.



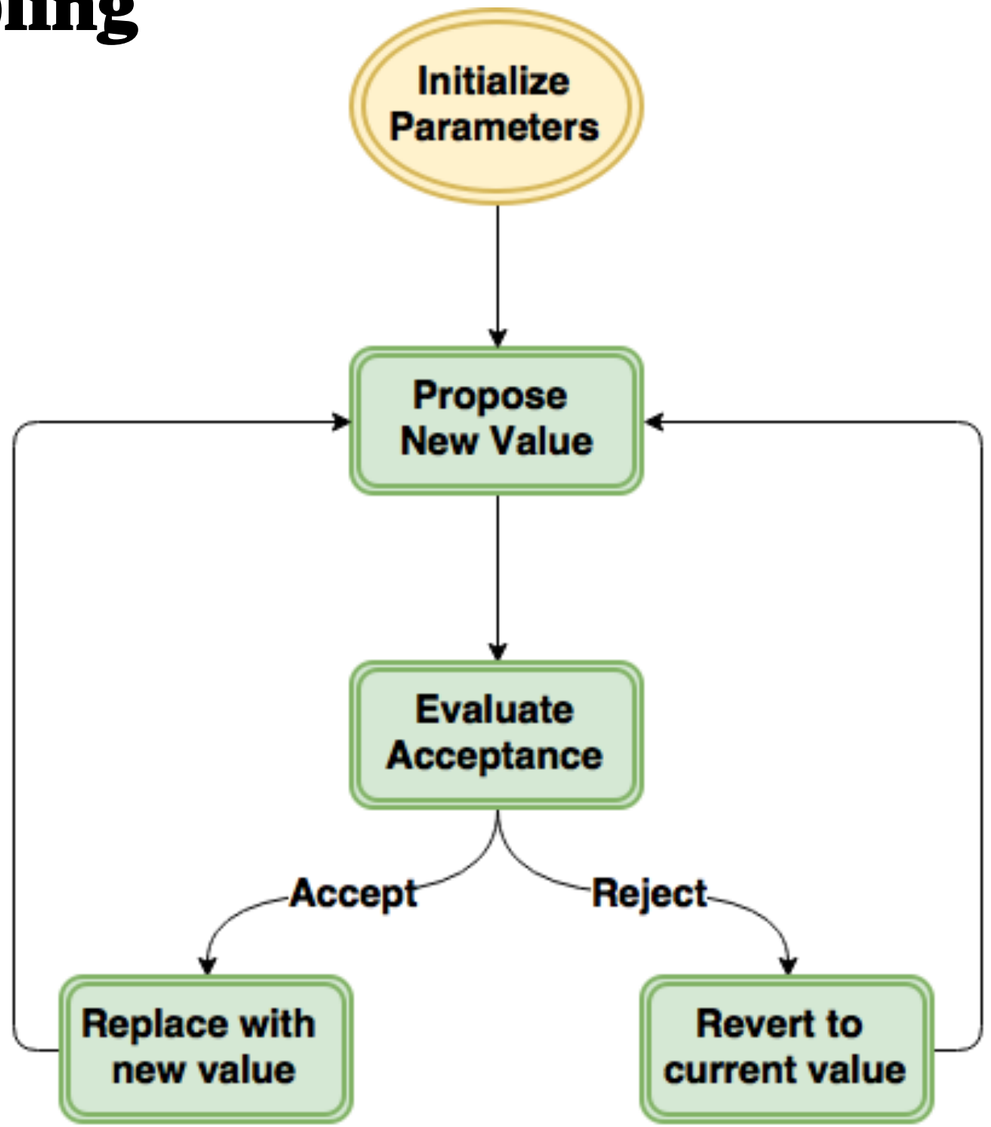
MCMC

Markov chain Monte Carlo simulates a **Markov chain** for which some function of interest is the **unique, invariant, limiting** distribution.

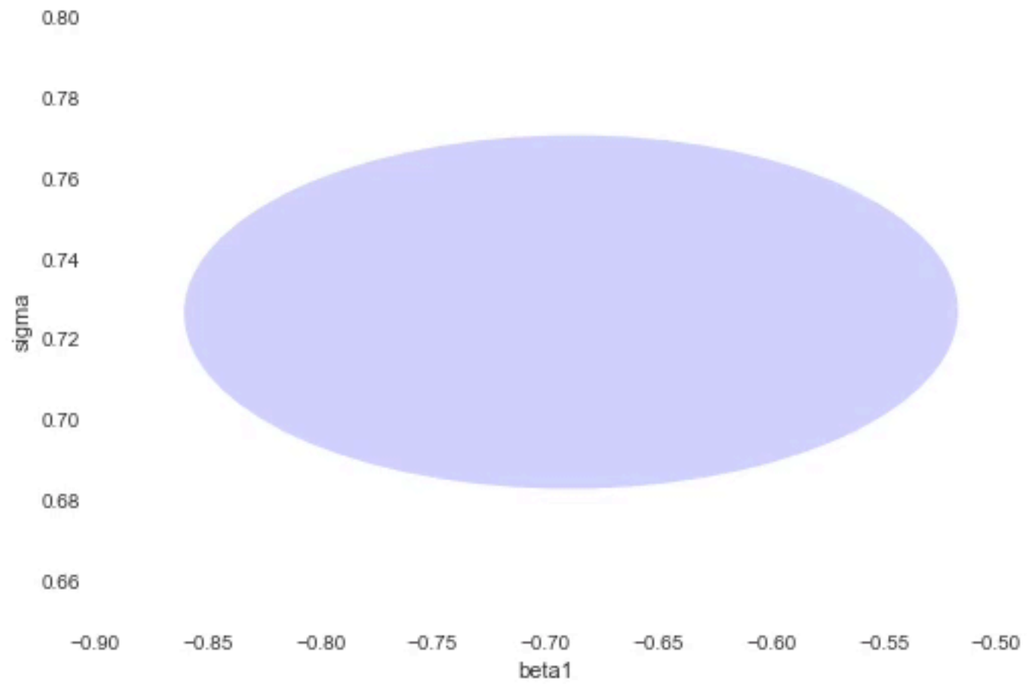
This is guaranteed when the Markov chain is constructed that satisfies the **detailed balance equation**:

$$\pi(x)Pr(y|x) = \pi(y)Pr(x|y)$$

Metropolis sampling



Metropolis sampling**



** 2000 iterations, 1000 tuning

Hamiltonian Monte Carlo

Uses a physical analogy of a frictionless particle moving on a hyper-surface

Requires an auxiliary variable to be specified

☞ position (unknown variable value)

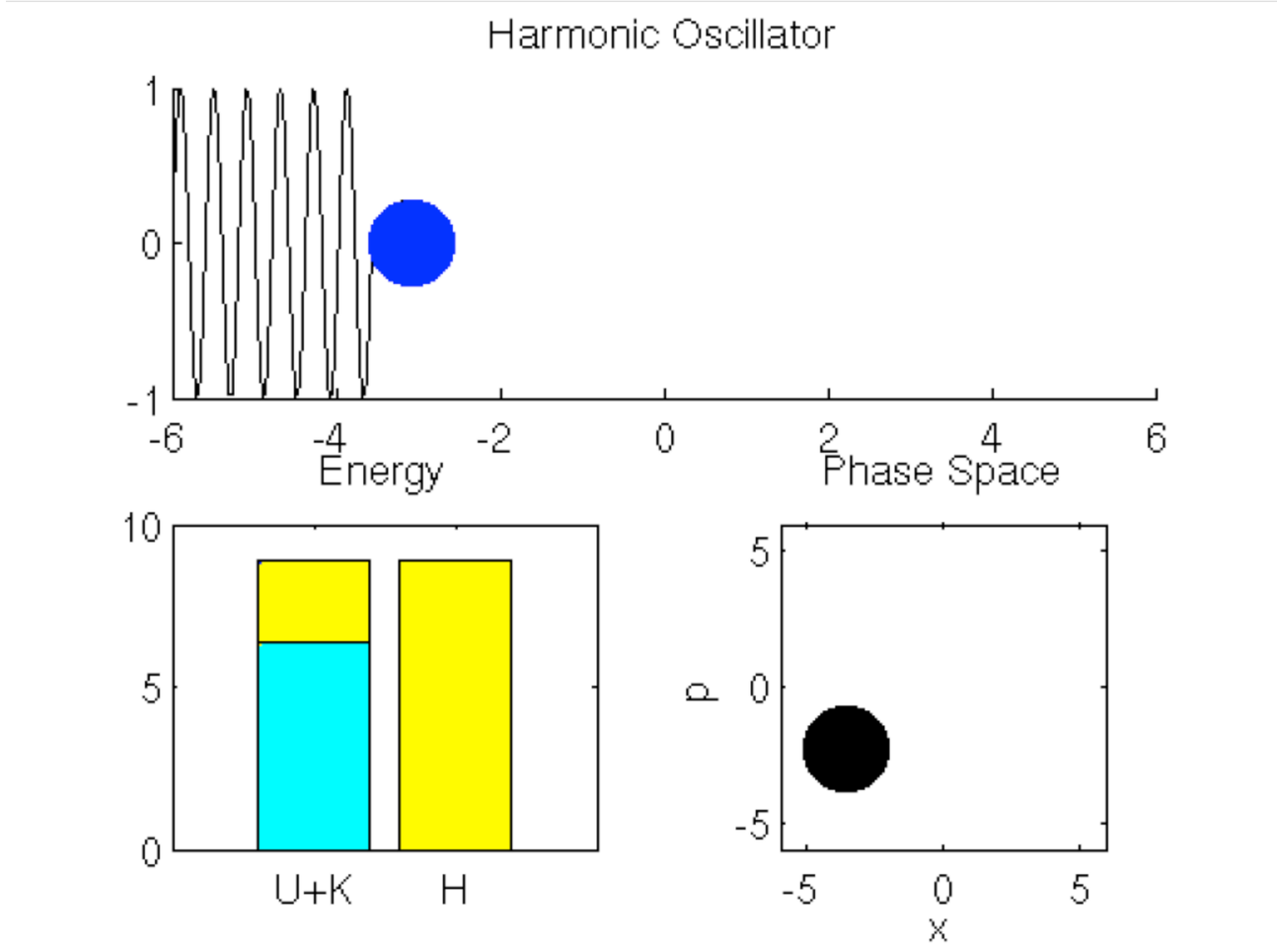
☞ momentum (auxiliary)

$$\mathcal{H}(\boldsymbol{s}, \boldsymbol{\phi}) = \boldsymbol{E}(\boldsymbol{s}) + \boldsymbol{K}(\boldsymbol{\phi}) = \boldsymbol{E}(\boldsymbol{s}) + \frac{\mathbf{1}}{\mathbf{2}}(\sum_{\boldsymbol{i}})\boldsymbol{\phi}_{\boldsymbol{i}}^{\mathbf{2}}$$

Hamiltonian Dynamics

$$\frac{ds_i}{dt} = \frac{\partial \mathcal{H}}{\partial \phi_i} = \phi_i$$

$$\frac{d\phi_i}{dt} = -\frac{\partial \mathcal{H}}{\partial s_i} = -\frac{\partial E}{\partial s_i}$$

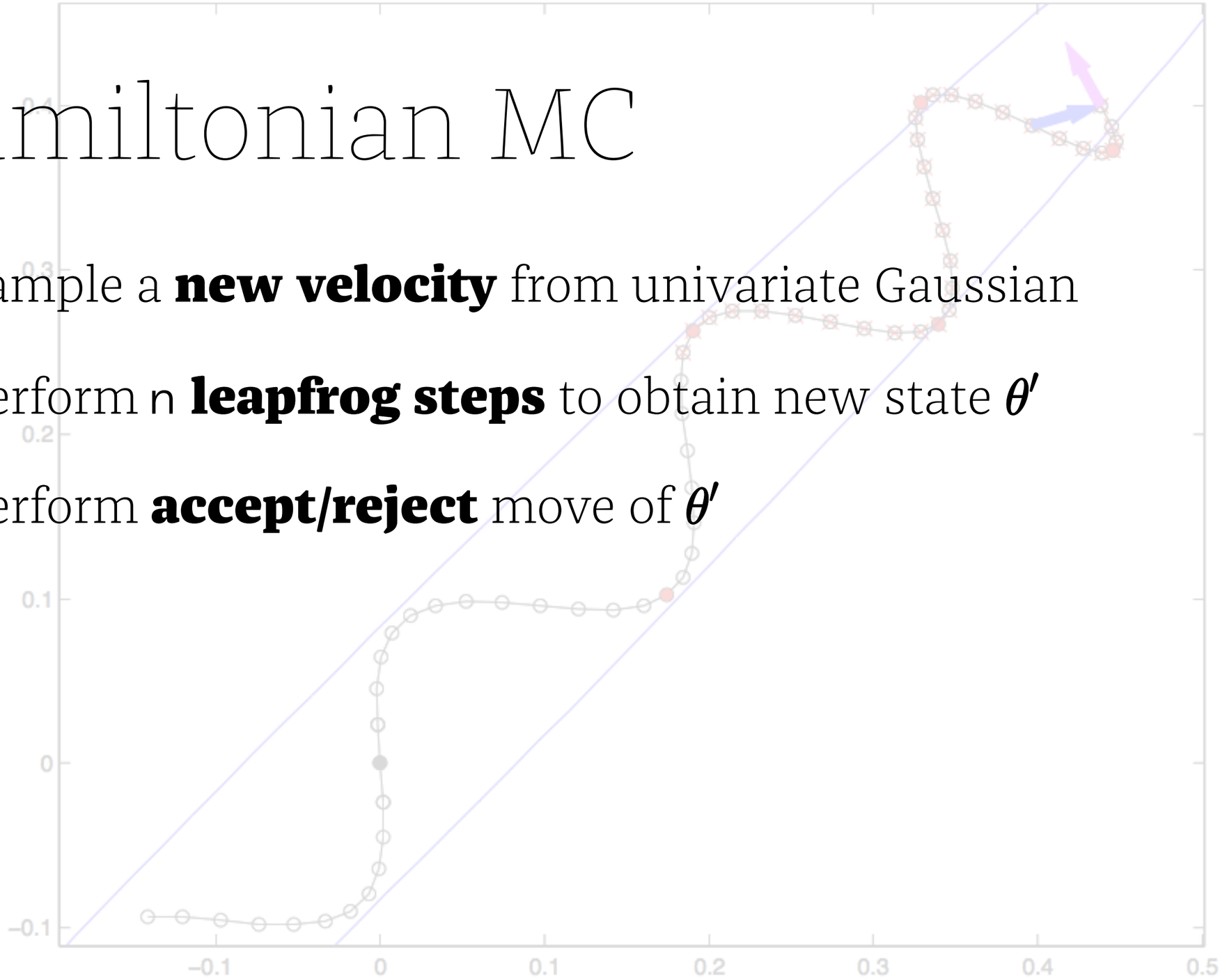


Hamiltonian MC

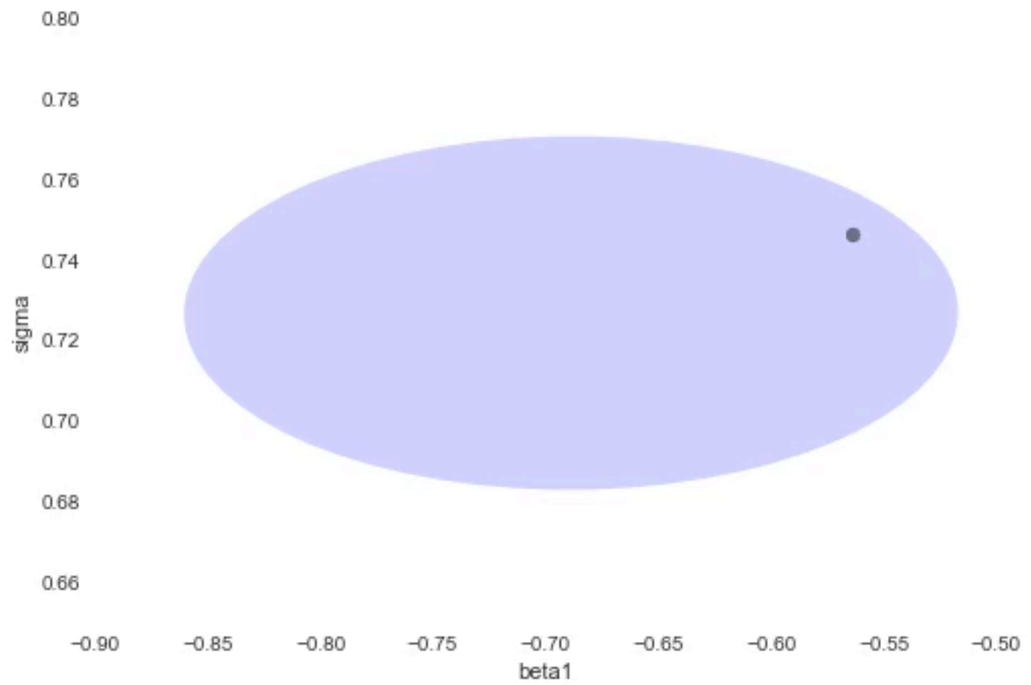
① Sample a **new velocity** from univariate Gaussian

② Perform n **leapfrog steps** to obtain new state θ'

③ Perform **accept/reject** move of θ'



Hamiltonian MC**



** 2000 iterations, 1000 tuning

No U-Turn Sampler (NUTS)

Hoffmann and Gelman (2014)

```
    sigma_y = HalfCauchy('sigma_y',5)

    # Expected value
    y_hat = a[county]

    # Data likelihood
    y_like = Normal('y_like', mu=y_hat, sd=sigma_y, observed=log_radon)
```

In [ ]:
```
with model:
    samples = sample(draws=1000, n_init=20000)
```

Start Recording

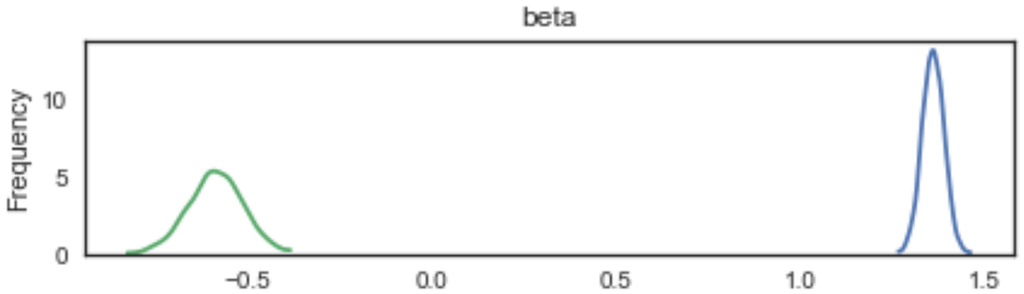
Varying intercept model

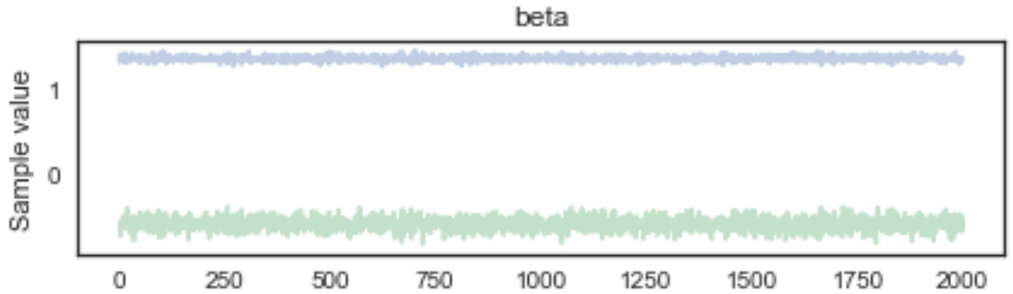
This model allows intercepts to vary across county, according to a random effect.

$$y_i = \alpha_{j[i]} + \beta x_i + \epsilon_i$$

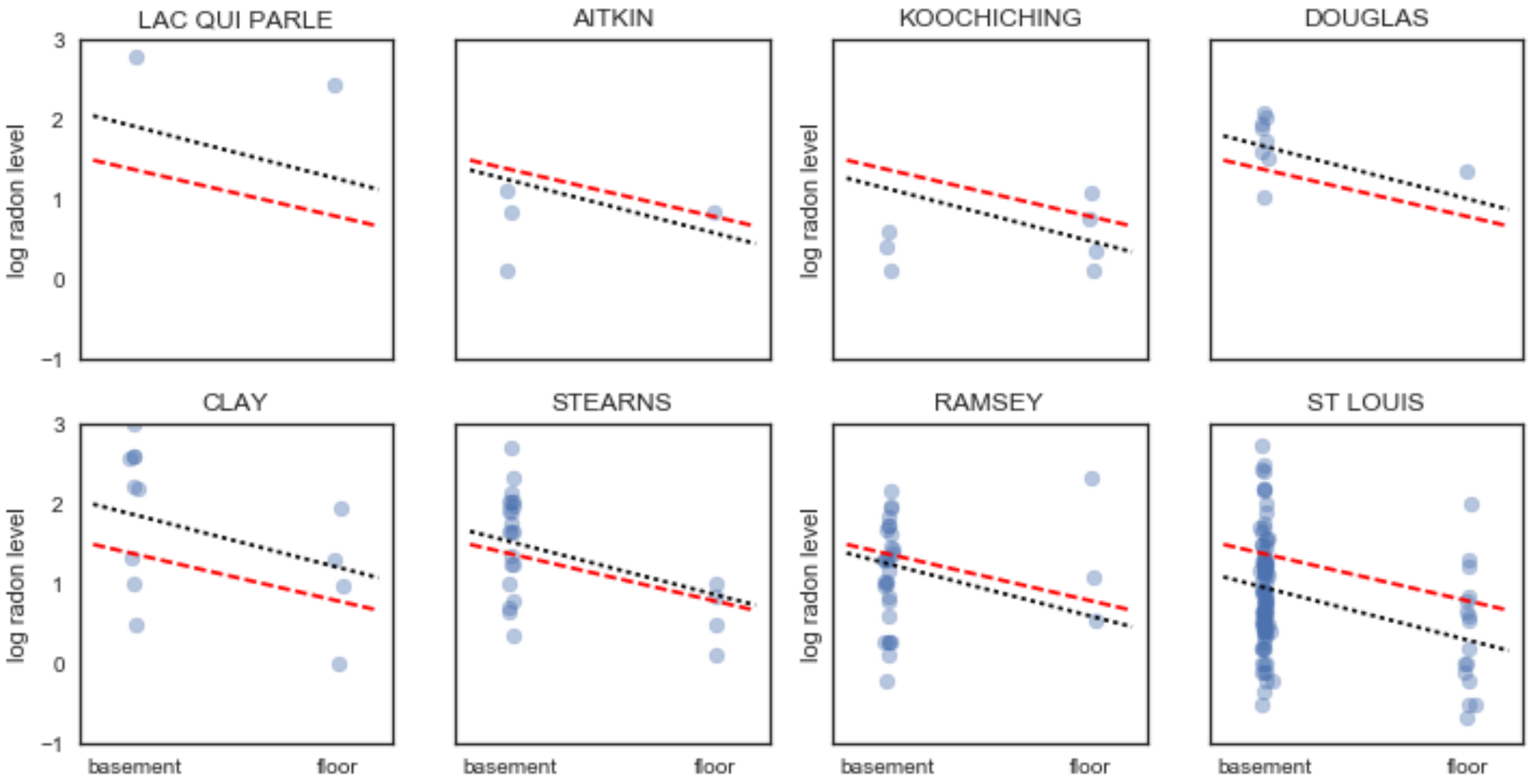
where

$$\epsilon_i \sim N(0, \sigma_y^2)$$

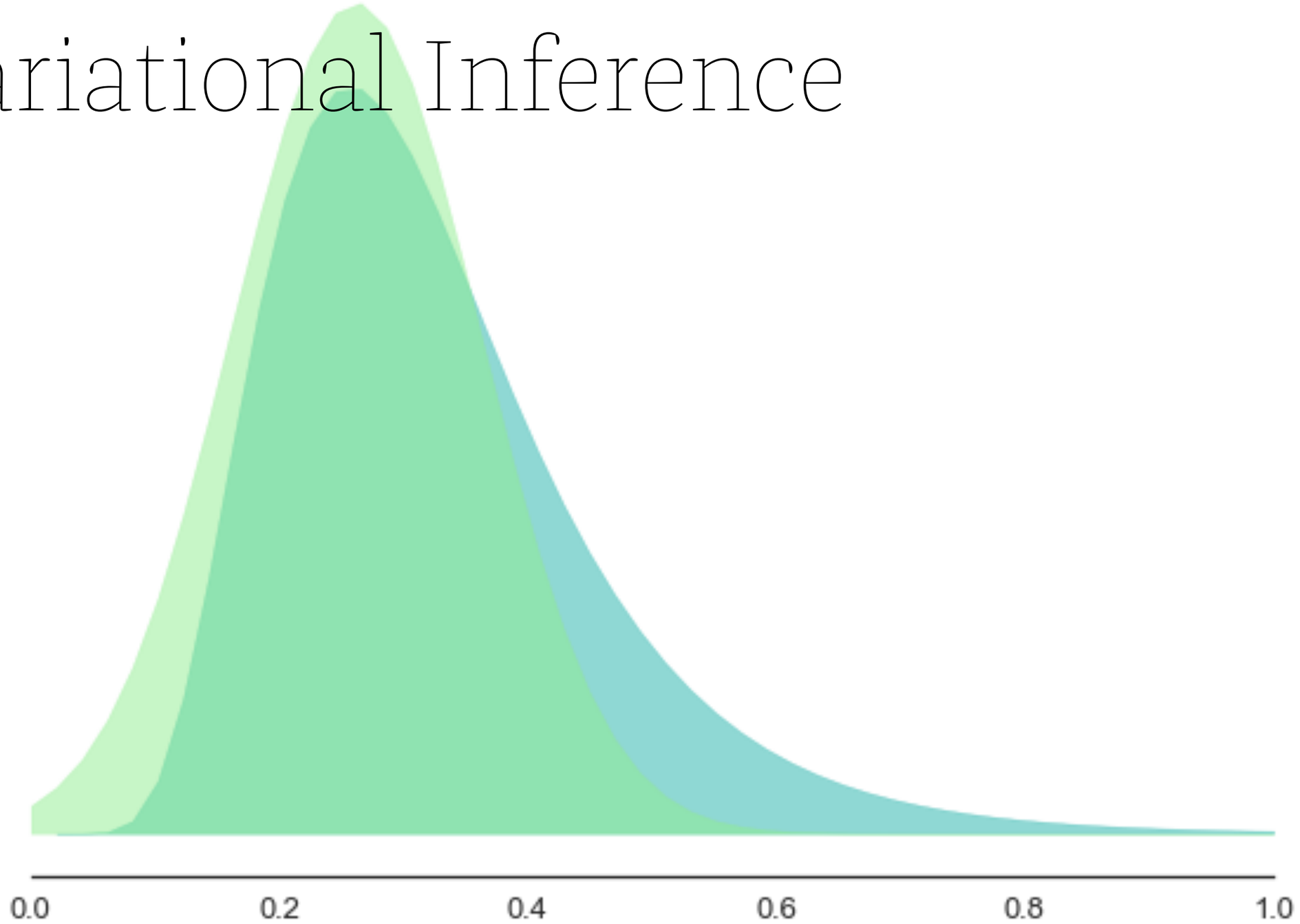




	mean	sd	mc_error	hpd_2.5	hpd_97.5
beta__0	1.362390	0.029348	0.000673	1.306512	1.420500
beta__1	-0.587654	0.073564	0.001590	-0.733906	-0.442381


LAC QUI PARLE
AITKIN
KOOCHICHING
DOUGLAS
CLAY
STEARNS
RAMSEY
ST LOUIS
log radon level
3
2
1
0
−1
basement
floor

Variational Inference



Variational Inference

Variational inference minimizes the **Kullback-Leibler divergence**

$$\mathbb{KL}(q(\theta) \parallel p(\theta \mid y)) = \int q(\theta, \phi) \frac{q(\theta, \phi)}{p(\theta|y)} d\theta$$

$$\Rightarrow \mathbb{E}_q \left(\log \left(\frac{q(\theta)}{p(\theta \mid y)} \right) \right)$$

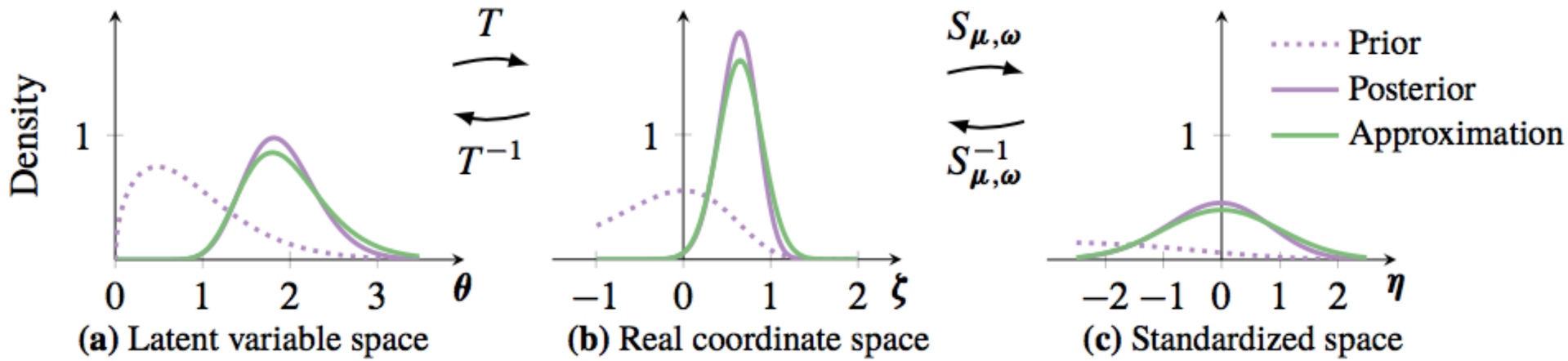
from approximate distributions, but we can't calculate the true posterior distribution.

Evidence Lower Bound

(ELBO)

$$\mathbb{KL}(q(\theta) \parallel p(\theta \mid \mathcal{D})) = -(\underbrace{\mathbb{E}_q(\log p(\mathcal{D}, \theta)) - \mathbb{E}_q(\log q(\theta))}_{\textbf{ELBO}}) + \log p(\mathcal{D})$$

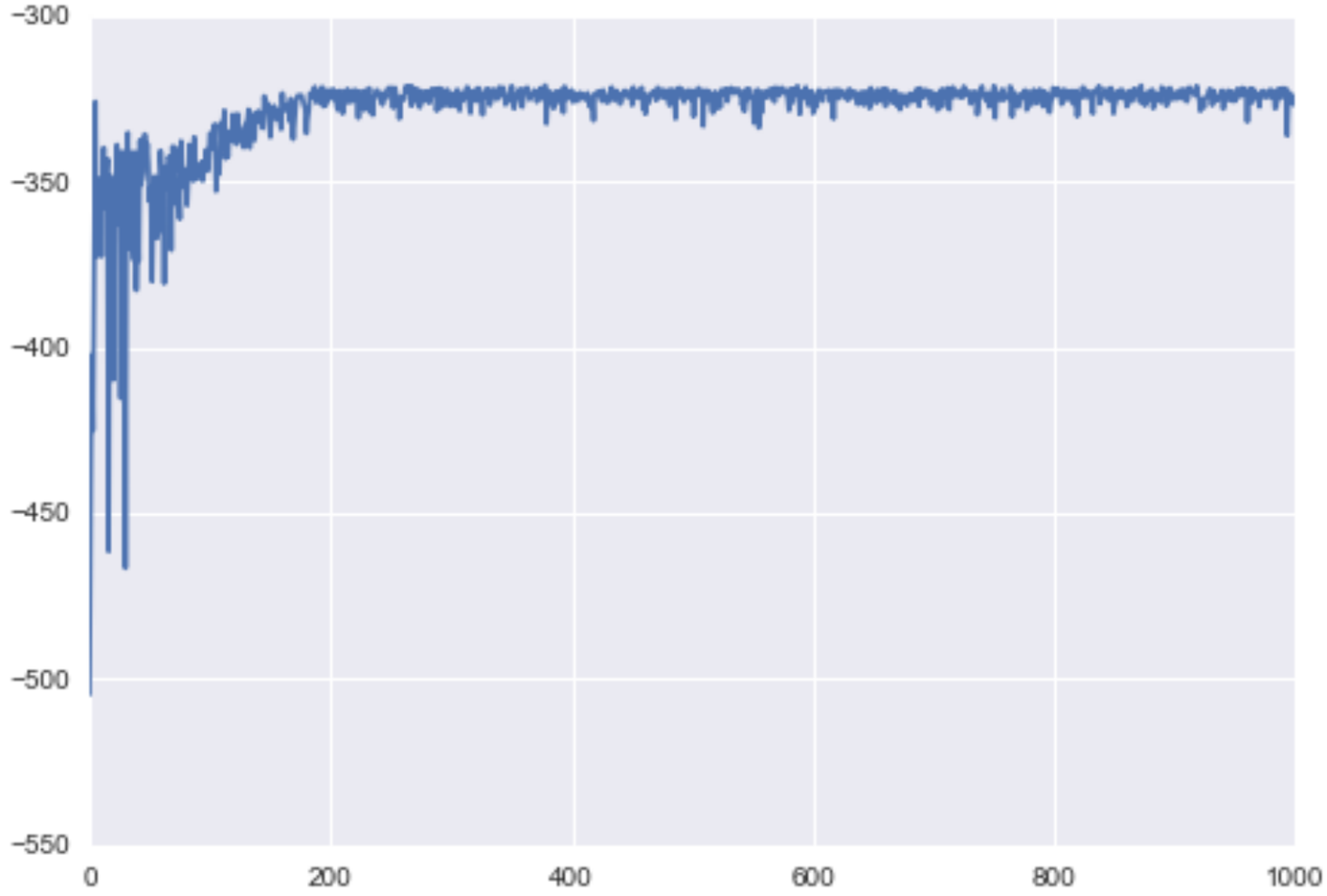
ADVI*



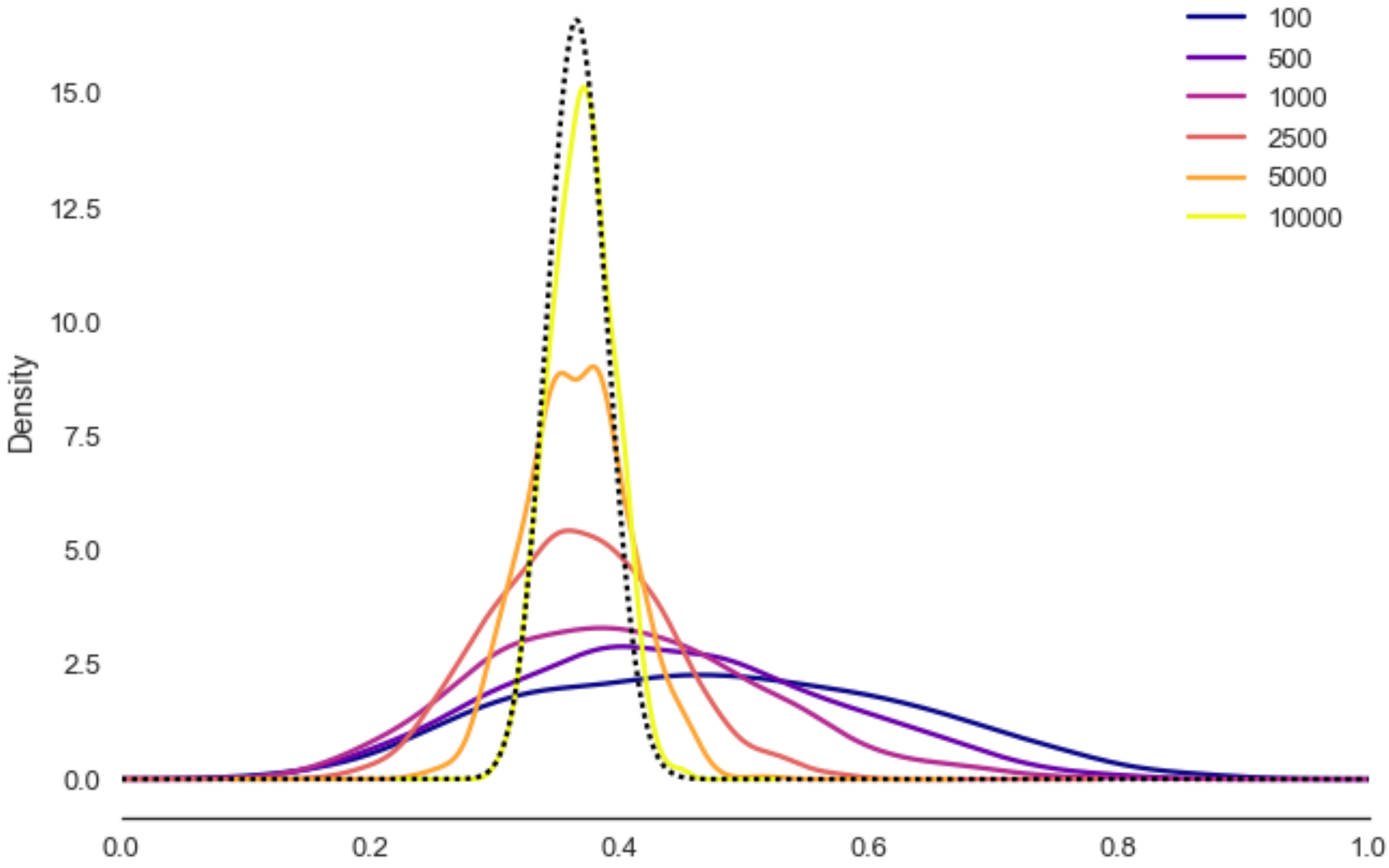
(a) Latent variable space (b) Real coordinate space (c) Standardized space

* Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2016, March 2). Automatic Differentiation Variational Inference. arXiv.org.

Maximizing the ELBO



Estimating `Beta(147, 255)` **posterior**



```
with hierarchical_model:

    approx = fit(n=100000)
```

```
Average Loss = 1,115.5: 100%|██████████| 100000/100000 [00:13<00:00, 7690.51it/s]
Finished [100%]: Average Loss = 1,115.5
```

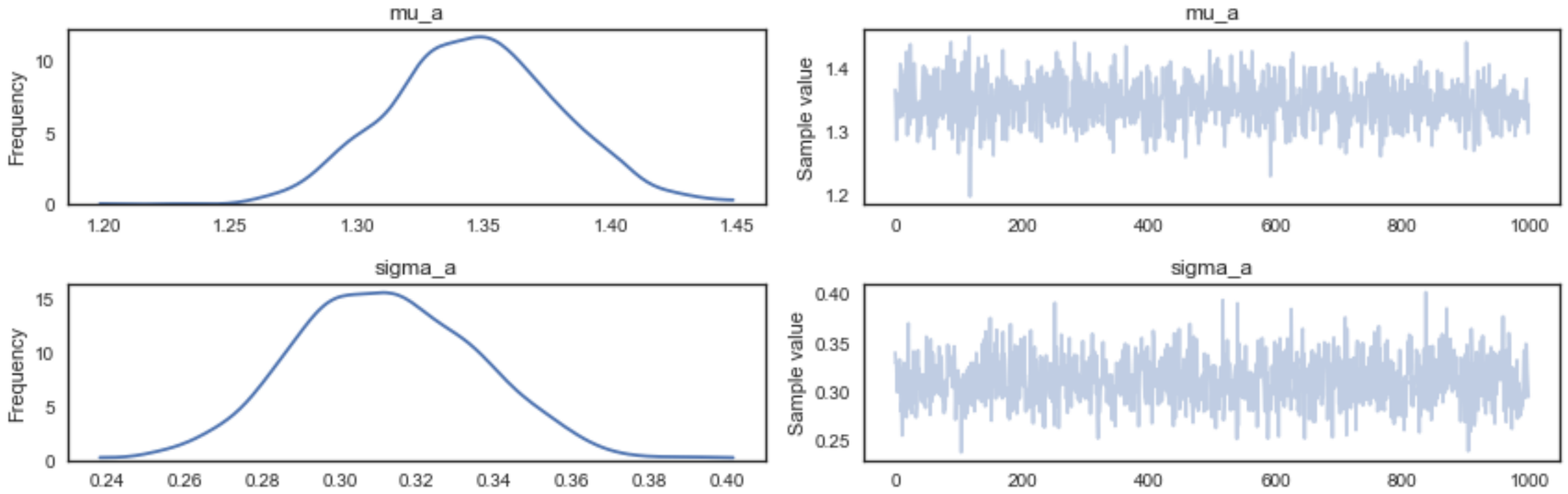
```
with hierarchical_model:

    approx = fit(n=100000)
```

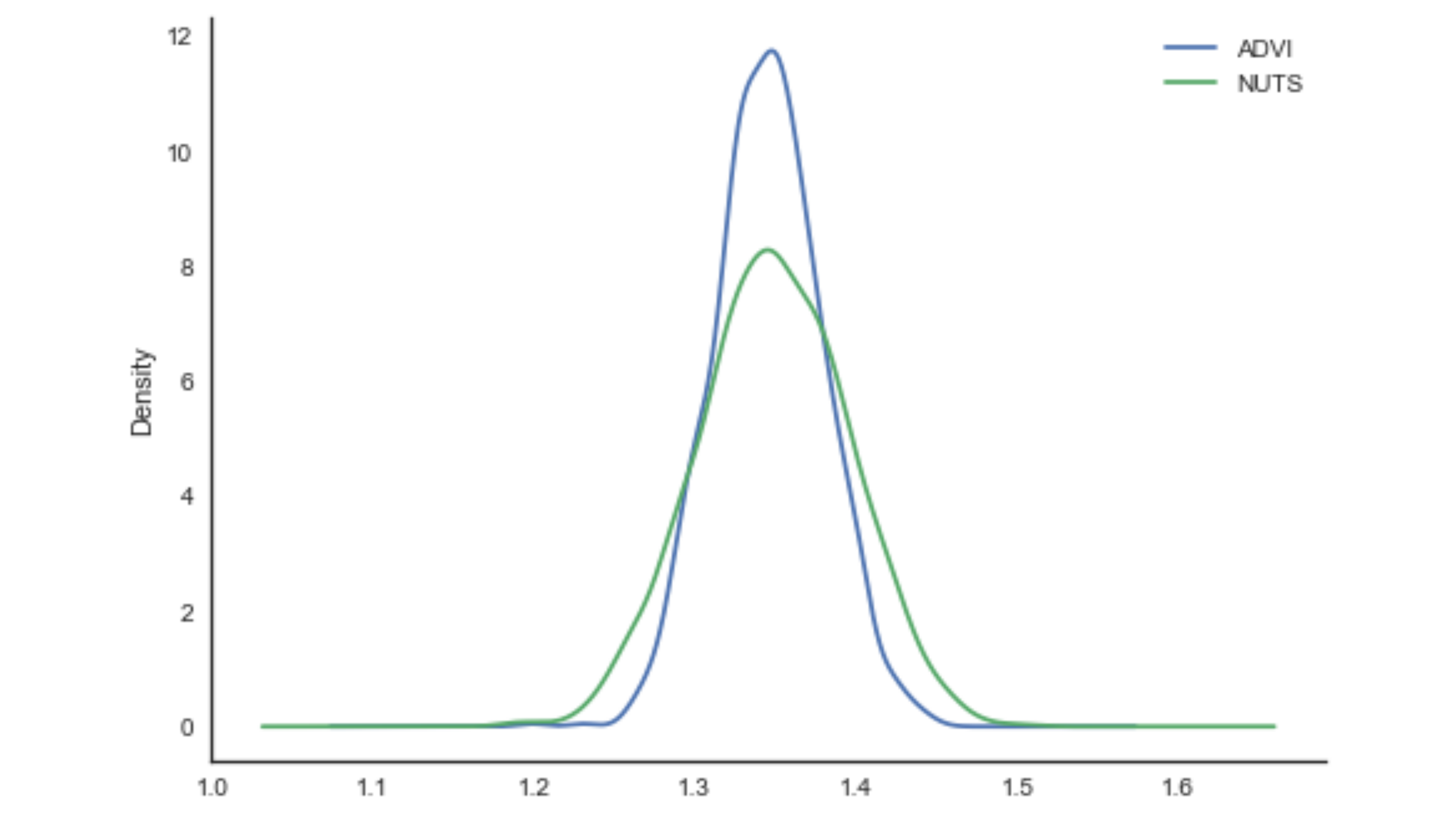
```
Average Loss = 1,115.5: 100%|██████████| 100000/100000 [00:13<00:00, 7690.51it/s]
Finished [100%]: Average Loss = 1,115.5
```

```
>>> approx
```

```
<pymc3.variational.approximations.MeanField at 0x119aa7c18>
```

```
with hierarchical_model:

    approx_sample = approx.sample(1000)

traceplot(approx_sample, varnames=['mu_a', 'σ_a'])
```




ADVI
NUTS
Density
12
10
8
6
4
2
0
1.0
1.1
1.2
1.3
1.4
1.5
1.6

Minibatch ADVI

```
minibatch_x = pm.Minibatch(X_train, batch_size=50)
minibatch_y = pm.Minibatch(Y_train, batch_size=50)
model = construct_model(minibatch_x, minibatch_y)

with model:
    approx = pm.fit(40000)
```

Gaussian processes

A latent, non-linear function $f(x)$ is modeled as being multivariate normally distributed (a Gaussian Process):

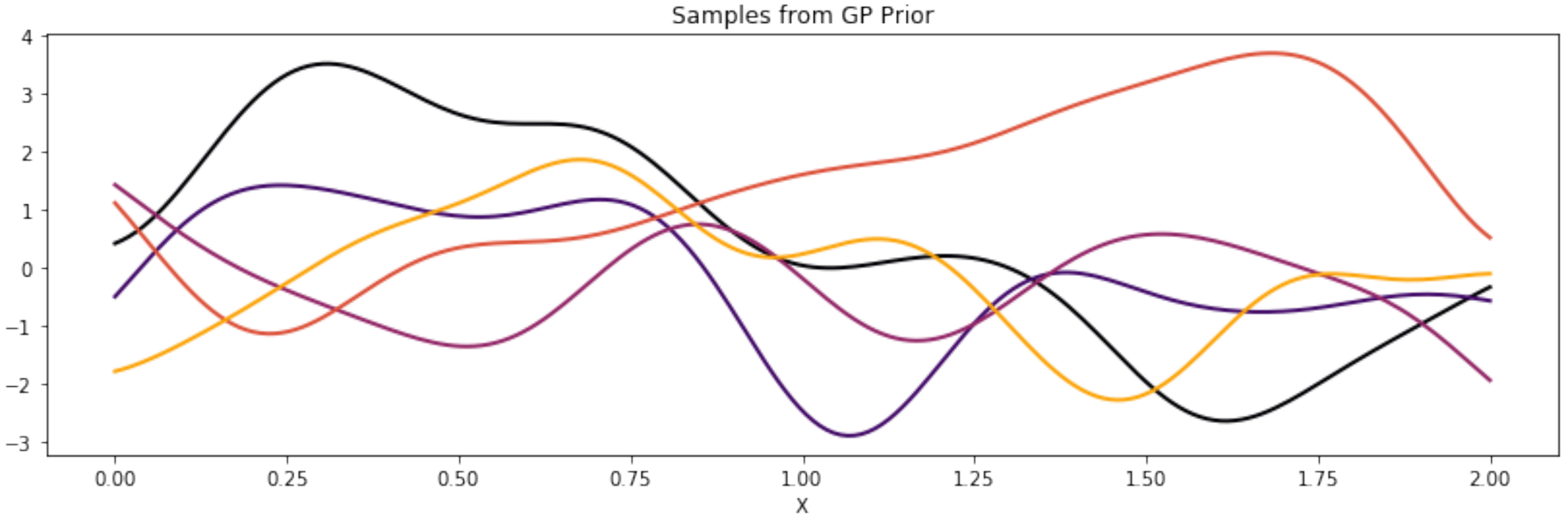
$$f(x) \sim \mathcal{GP}(m(x),\, k(x, x'))$$

☞ mean function, $m(x)$

☞ covariance function, $k(x, x')$

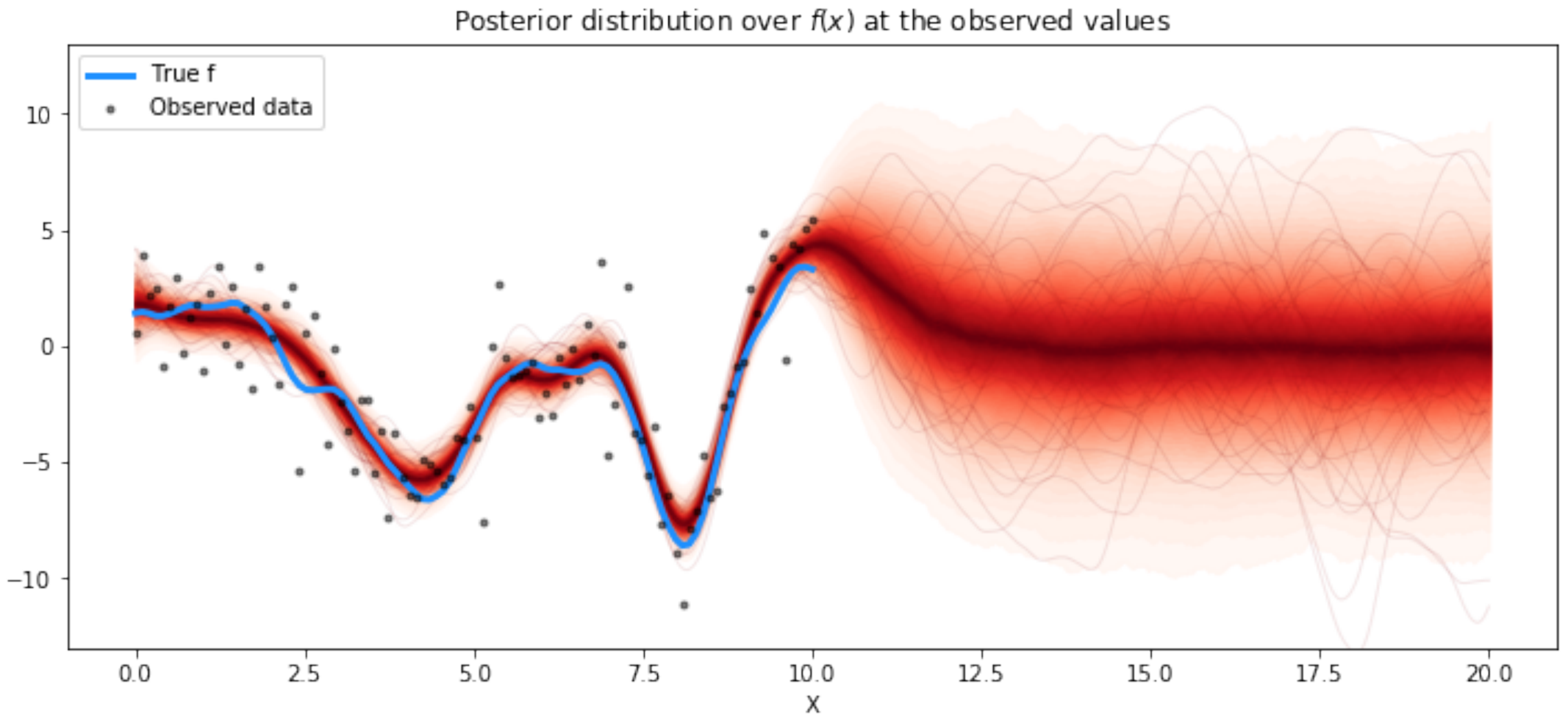
Quadratic Covariance Function

$$k(x, x') = \exp\left[-\frac{(x - x')^2}{2\ell^2}\right]$$



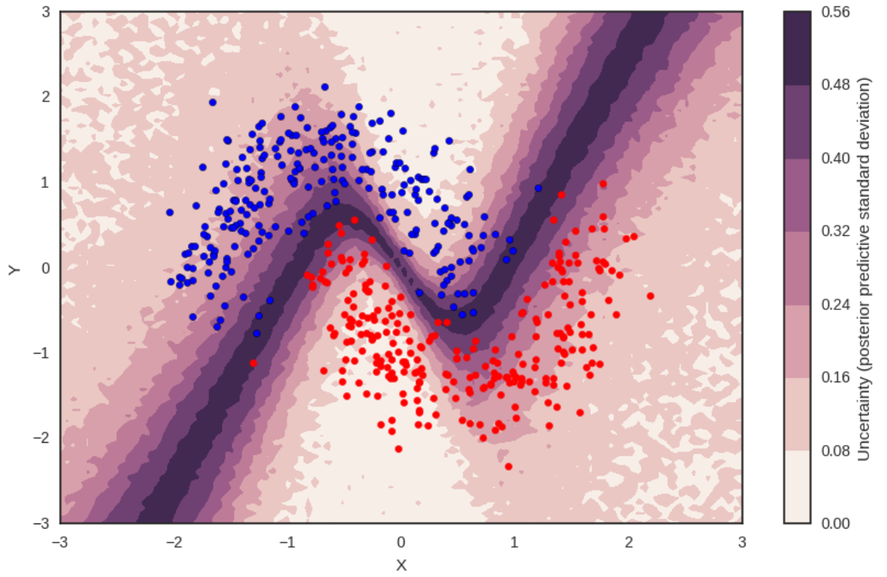
Marginal GP

```
with pm.Model() as model:
    ℓ = pm.Gamma("ℓ", alpha=2, beta=1)
    η = pm.HalfCauchy("η", beta=5)

    cov = η**2 * pm.gp.cov.Matern52(1, ℓ)
    gp = pm.gp.Marginal(cov_func=cov)

    σ = pm.HalfCauchy("σ", beta=5)
    y_ = gp.marginal_likelihood("y", n_points=n, X=X, y=y, noise=σ)
```


Posterior distribution over $f(x)$ at the observed values
True f
Observed data
10
5
0
−5
−10
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
X

Bayesian Machine Learning

Bayesian Deep Learning in PyMC3✧



✧ Thomas Wiecki 2016

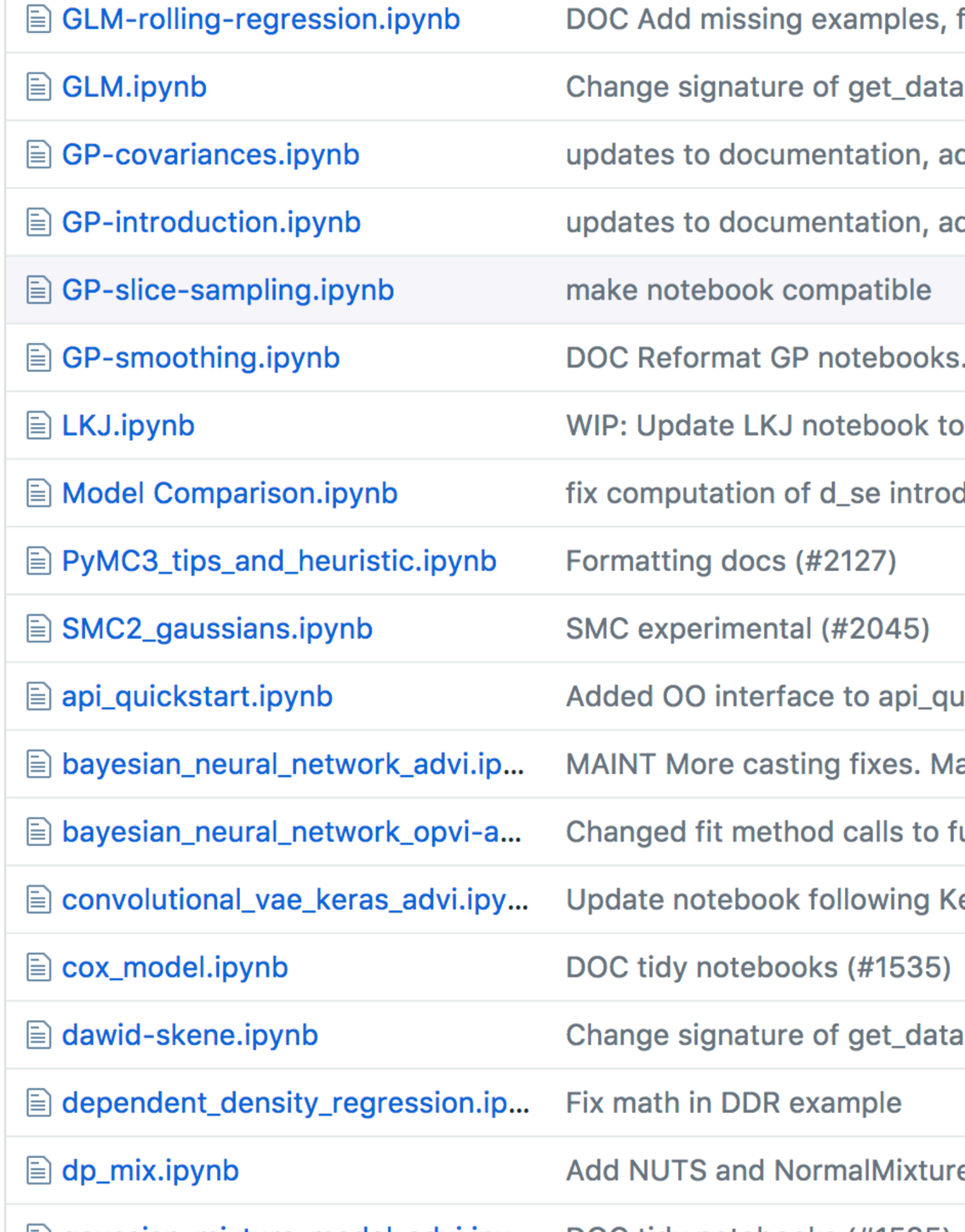
```
with pm.Model() as neural_network:
    # Weights from input to hidden layer
    weights_in_1 = pm.Normal('w_in_1', 0, sd=1,
                             shape=(X.shape[1], n_hidden),
                             testval=init_1)

    # Weights from 1st to 2nd layer
    weights_1_2 = pm.Normal('w_1_2', 0, sd=1,
                            shape=(n_hidden, n_hidden),
                            testval=init_2)

    # Weights from hidden layer to output
    weights_2_out = pm.Normal('w_2_out', 0, sd=1,
                              shape=(n_hidden,),
                              testval=init_out)

    # Build neural-network using tanh activation function
    act_1 = pm.math.tanh(pm.math.dot(ann_input,
                                     weights_in_1))
    act_2 = pm.math.tanh(pm.math.dot(act_1,
                                     weights_1_2))
    act_out = pm.math.sigmoid(pm.math.dot(act_2,
                                          weights_2_out))

    # Binary classification -> Bernoulli likelihood
    out = pm.Bernoulli('out',
                       act_out,
                       observed=ann_output)
```

The Future

- ☞ Discontinuous HMC
- ☞ Riemannian Manifold HMC
- ☞ Stochastic Gradient Fisher Scoring
- ☞ ODE solvers

Jupyter Notebook Gallery

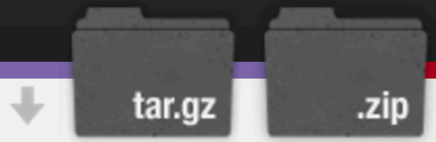
bit.ly/pymc3nb



Other Probabilistic Programming Tools[^🏳]

- ☞ Edward
- ☞ GPy/GPFlow
- ☞ PyStan
- ☞ emcee
- ☞ BayesPy

Probabilistic Programming & Bayesian Methods for Hackers



An intro to Bayesian methods and probabilistic programming from a computation/understanding-first, mathematics-second point of view.

Prologue

The PyMC3 Team

- ☞ Colin Carroll
- ☞ Peadar Coyle
- ☞ Bill Engels
- ☞ Maxim Kochurov
- ☞ Junpeng Lao
- ☞ Osvaldo Martin
- ☞ Kyle Meyer
- ☞ Austin Rochford
- ☞ John Salvatier
- ☞ Adrian Seyboldt
- ☞ Hannes Vasyura-Bathke
- ☞ Thomas Wiecki
- ☞ Taku Yoshioka