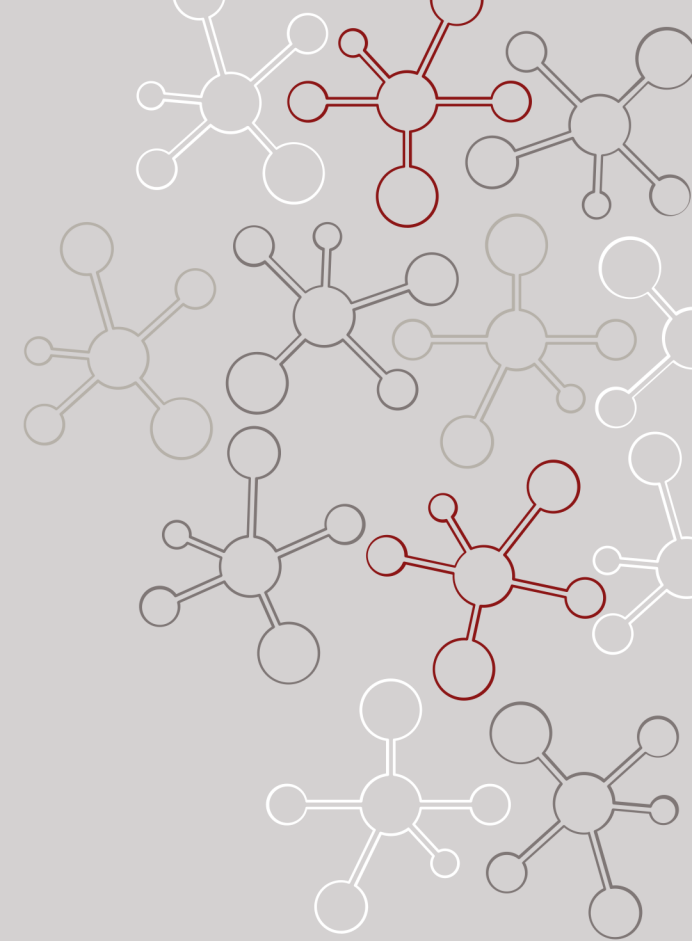


Simulation-Based Inference I

Michael Kagan
SLAC

July 19, 2023



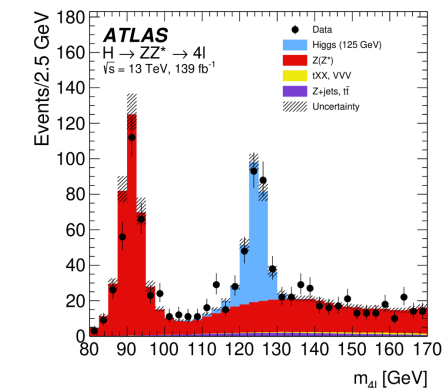
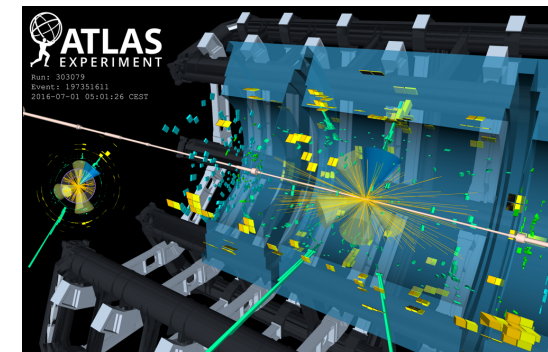
We have a scientific theory

From which we can make testable predictions

Then compare to experimental data
and draw inferences about the theory

$$\mathcal{L} = \mathcal{L}_{SM} + \mathcal{L}_{Higgs} + \mathcal{L}_{\text{new}}$$
$$\mathcal{L}_{SM} = \sum_f \bar{\psi}_f (i \not{D} - m_f) \psi_f + \sum_{W, Z} -\frac{1}{4} W_{\mu\nu}^2 - \frac{1}{4} Z_{\mu\nu}^2 - \frac{1}{2} (D_\mu H)^\dagger (D_\mu H) - \frac{1}{2} M_W^2 W_\mu^2 - \frac{1}{2} M_Z^2 Z_\mu^2 - \frac{1}{2} m_H^2 H^\dagger H - \frac{1}{4} \lambda (H^\dagger H)^2$$
$$\mathcal{L}_{Higgs} = -\frac{1}{2} (D_\mu H)^\dagger (D_\mu H) - \frac{1}{2} m_H^2 H^\dagger H - \frac{1}{4} \lambda (H^\dagger H)^2$$
$$\mathcal{L}_{\text{new}} = \sum_i \bar{\psi}_i (i \not{D} - m_i) \psi_i + \sum_{\text{new}} \bar{\psi}_{\text{new}} (i \not{D} - m_{\text{new}}) \psi_{\text{new}} + \dots$$

The Standard Model

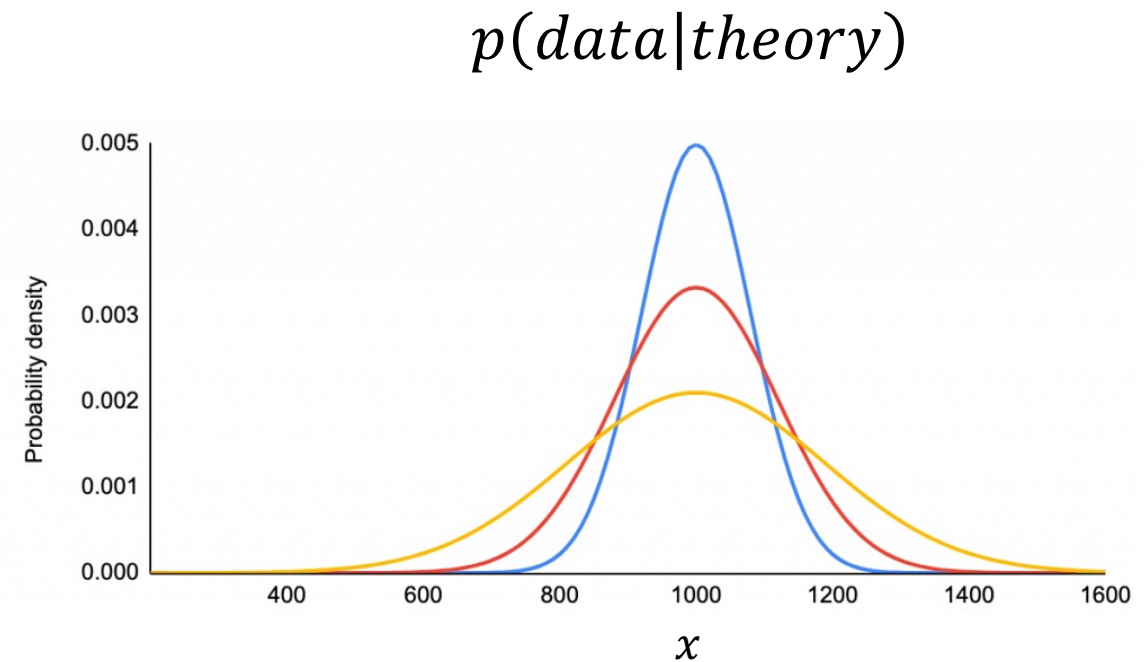


Our Prediction are Not Deterministic

Our predictions are almost never exact, e.g. due to

- Randomness of physical processes
- Randomness in measurement process
- Incomplete information
- Quantum Mechanics
- ...

Our predictions are often statistical



Likelihood Based Inference

Given data to analyze, x , that was generated through a random process,

We develop the *likelihood*, a model of the data generation process that describes the probability of the data given some parameters θ

$$p(x|\theta)$$

Likelihood Based Inference

Given data to analyze, x , that was generated through a random process,

We develop the *likelihood*, a model of the data generation process that describes the probability of the data given some parameters θ

$$p(x|\theta)$$

Example:

$$p(x|\theta) = N(\theta, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x-\theta)^2}$$

Likelihood Based Inference

Given data to analyze, x , that was generated through a random process,

We develop the *likelihood*, a model of the data generation process that describes the probability of the data given some parameters θ

$$p(\text{data} \mid \text{theory})$$

Parameters of the model can have meaning, so we want to know what the data tells us about the parameters

Think of params as describing our theory (e.g. parameters of Standard Model)

$$p(x|\theta)$$

Goal: perform inference on the parameters θ using the data x

$p(x|\theta)$

Frequentist Inference

$$r(x|\theta_1, \theta_0) = \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Hypothesis testing, confidence intervals...

Goal: perform inference on the parameters θ using the data x

$$p(x|\theta)$$

Frequentist Inference

$$r(x|\theta_1, \theta_0) = \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Hypothesis testing, confidence intervals...

Bayesian Inference

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

Posterior inference, credible intervals...

Goal: perform inference on the parameters θ using the data x

$$p(x|\theta)$$

In typical statistics, we write down an analytic form for the likelihood

Nice case: Can analytically derive posterior, confidence intervals, etc...

Not as nice: Can't analytically derive posterior, use sampling (e.g. MCMC), VI, ...

$$p(x|\theta)$$

In typical statistics, we write down an analytic form for the likelihood

Nice case: Can analytically derive posterior, confidence intervals, etc...

Not as nice: Can't analytically derive posterior, use sampling (e.g. MCMC), VI, ...

But what if we can't write down the likelihood? → **Simulation-Based Inference***

*sometimes also called Likelihood-Free Inference

Explicit Models

Can explicitly evaluate density

Often available analytically, e.g.

$$p(x|\lambda) = \text{Exp}(x) = \lambda e^{-\lambda x}$$

Can sample from the distribution

$$x \sim p(x|\lambda)$$

Implicit Models

Don't know density analytically

Can't evaluate density $p(x|\theta)$

But can sample from density

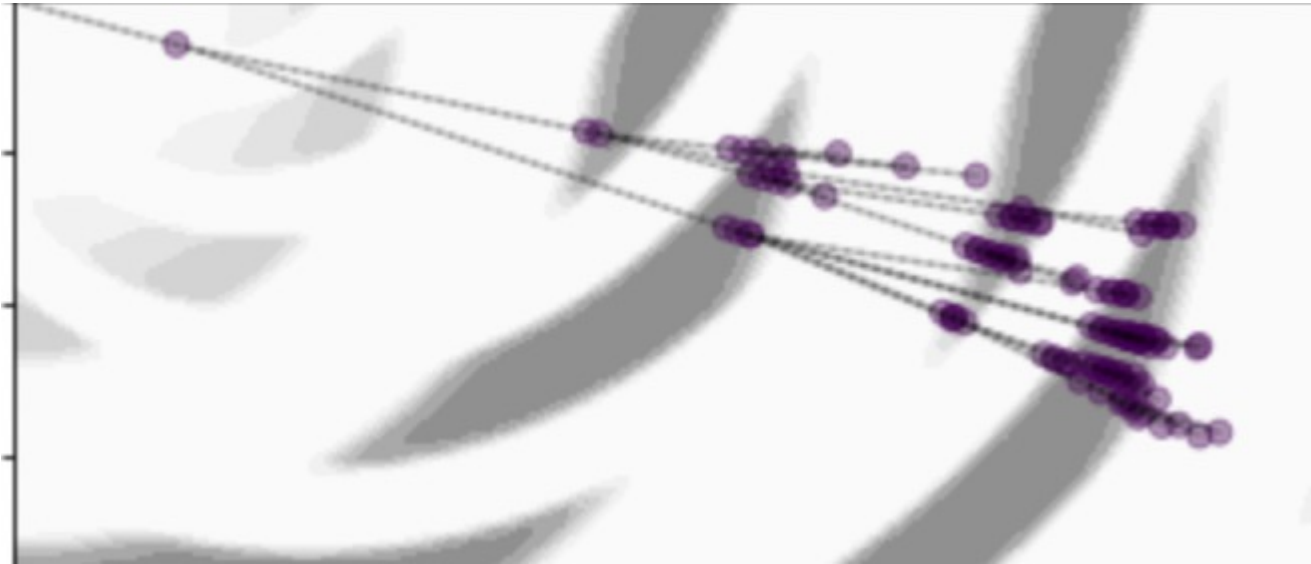
$$x \sim p(x|\theta)$$

Often describe density in code,
built up mechanistically as a set of
processes with randomness

What do we do without the Likelihood?

In science, often we know how to mechanistically describe how the data is generated, perhaps involving many random and unobserved processes'

i.e. we know how to simulate the process!



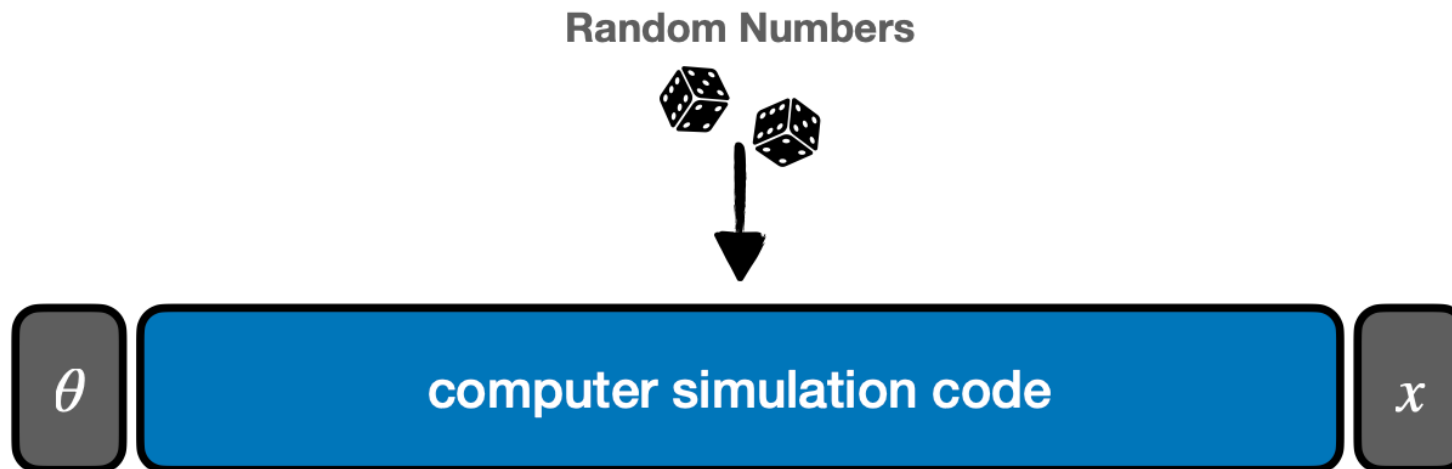
What do we do without the Likelihood?

Can implement the data generation process in a *stochastic simulator*

- Simulator depends on random process (often internally) to generate an output

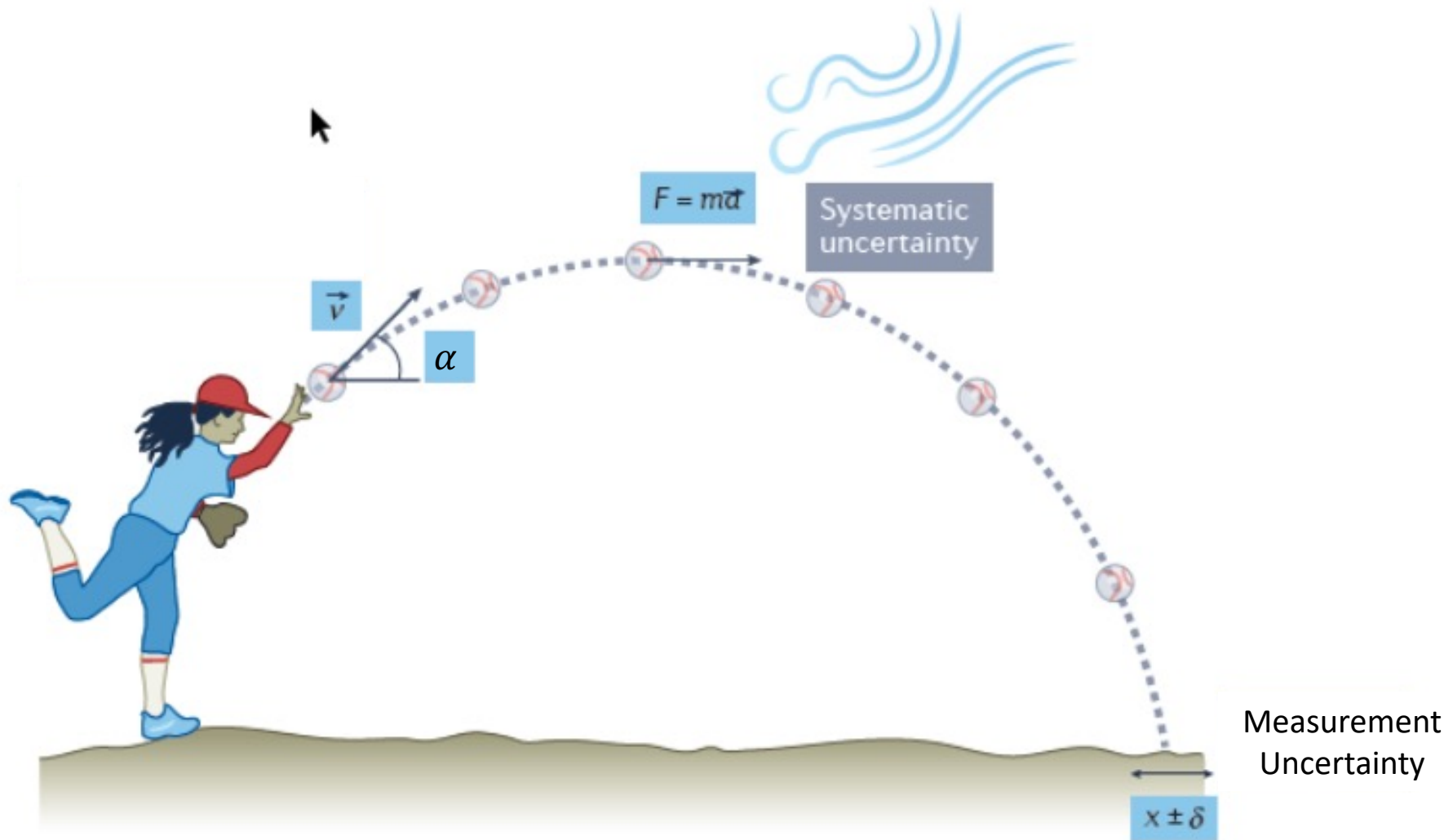
Allows us to sample observations given a set of parameters

In this way, we can implicitly define $p(x|\theta)$



$$x = SIM(\text{die}, \theta) \sim p(x|\theta)$$

Example



$$v_x = v \cos(\alpha), \quad v_y = v \sin(\alpha),$$

$$\frac{dx}{dt} = v_x, \quad \frac{dy}{dt} = v_y, \quad \frac{dv_y}{dt} = -G.$$

Example

Simulator with unobserved *latent random variables*

```
def simulate(v, alpha, dt=0.001):  
    v_x = v * np.cos(alpha) # x velocity m/s  
    v_y = v * np.sin(alpha) # y velocity m/s  
    y = 1.1 + 0.3 * random.normal() # Randomness in initial height  
    x = 0.0  
  
    while y > 0: # simulate until ball hits floor  
        v_y += dt * -G # acceleration due to gravity  
        x += dt * v_x  
        y += dt * v_y  
  
    return x + 0.25 * random.normal() # Randomness in measurement
```

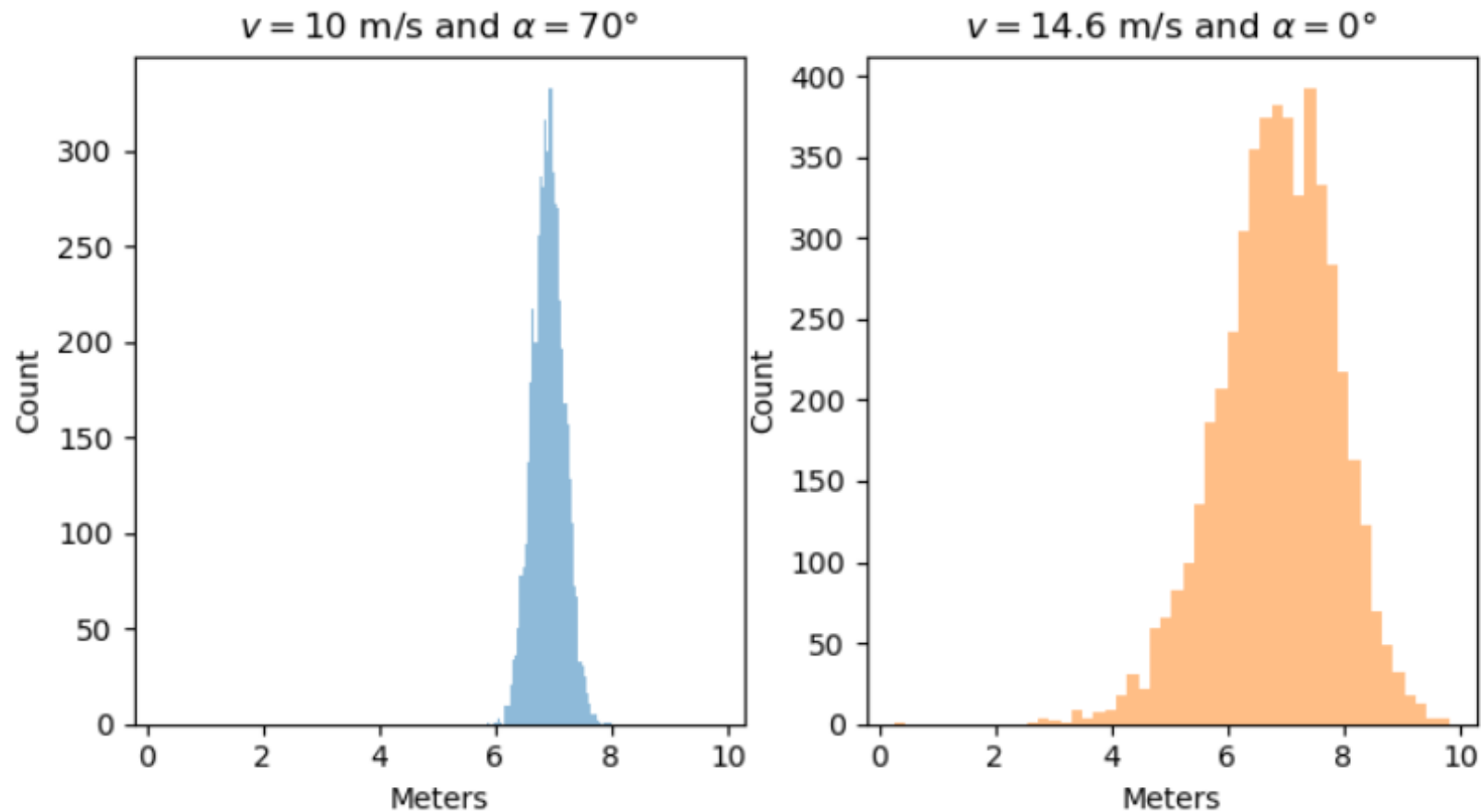
Randomness in initial height

Randomness in measurement

Example

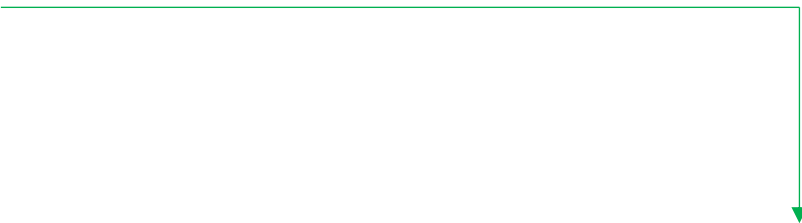
The simulator defines the likelihood $p(x|\theta)$ implicitly

$x = \text{data}$
 $\theta = \text{parameters}$



Summarizing the Components

Parameters of the model = θ



```
def simulate(v, alpha, dt=0.001):
    v_x = v * np.cos(alpha) # x velocity m/s
    v_y = v * np.sin(alpha) # y velocity m/s
    y = 1.1 + 0.3 * random.normal()
    x = 0.0

    while y > 0: # simulate until ball hits floor
        v_y += dt * -G # acceleration due to gravity
        x += dt * v_x
        y += dt * v_y

    return x + 0.25 * random.normal()
```

Summarizing the Components

Parameters of the model = θ

Unobserved latent random variable = z

```
def simulate(v, alpha, dt=0.001):  
    v_x = v * np.cos(alpha) # x velocity m/s  
    v_y = v * np.sin(alpha) # y velocity m/s  
    y = 1.1 + 0.3 * random.normal()  
    x = 0.0  
  
    while y > 0: # simulate until ball hits floor  
        v_y += dt * -G # acceleration due to gravity  
        x += dt * v_x  
        y += dt * v_y  
  
    return x + 0.25 * random.normal()
```

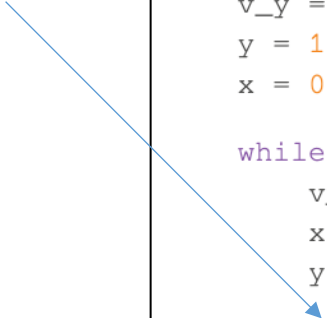

Summarizing the Components

Parameters of the model = θ

Unobserved latent random variable = z

Use simulator to generate observations = x

```
def simulate(v, alpha, dt=0.001):  
    v_x = v * np.cos(alpha) # x velocity m/s  
    v_y = v * np.sin(alpha) # y velocity m/s  
    y = 1.1 + 0.3 * random.normal()  
    x = 0.0  
  
    while y > 0: # simulate until ball hits floor  
        v_y += dt * -G # acceleration due to gravity  
        x += dt * v_x  
        y += dt * v_y  
  
    return x + 0.25 * random.normal()
```



Summarizing the Components

Parameters of the model = θ

Unobserved latent random variable = z

Use simulator to generate observations = x

Probability of a simulation run depends jointly on observation and latent variables

$$p(x, z | \theta)$$

```
def simulate(v, alpha, dt=0.001):  
    v_x = v * np.cos(alpha) # x velocity m/s  
    v_y = v * np.sin(alpha) # y velocity m/s  
    y = 1.1 + 0.3 * random.normal()  
    x = 0.0  
  
    while y > 0: # simulate until ball hits floor  
        v_y += dt * -G # acceleration due to gravity  
        x += dt * v_x  
        y += dt * v_y  
  
    return x + 0.25 * random.normal()
```

Summarizing the Components

Parameters of the model = θ

Unobserved latent random variable = z

Use simulator to generate observations = x

Probability of a simulation run depends jointly on observation and latent variables

$$p(x, z|\theta)$$

But we only observe x , so likelihood we need is marginalized over latent variables

$$p(x|\theta) = \int p(x, z|\theta) dz$$

```
def simulate(v, alpha, dt=0.001):
    v_x = v * np.cos(alpha) # x velocity m/s
    v_y = v * np.sin(alpha) # y velocity m/s
    y = 1.1 + 0.3 * random.normal()
    x = 0.0

    while y > 0: # simulate until ball hits floor
        v_y += dt * -G # acceleration due to gravity
        x += dt * v_x
        y += dt * v_y

    return x + 0.25 * random.normal()
```

Summarizing the Components

Parameters of the model = θ

Unobserved latent random variable = z

Use simulator to generate observations = x

Probability of a simulation run depends jointly on observation and latent variables

$$p(x, z|\theta)$$

But we only observe x , so likelihood we need is marginalized over latent variables

$$p(x|\theta) = \int p(x, z|\theta) dz$$

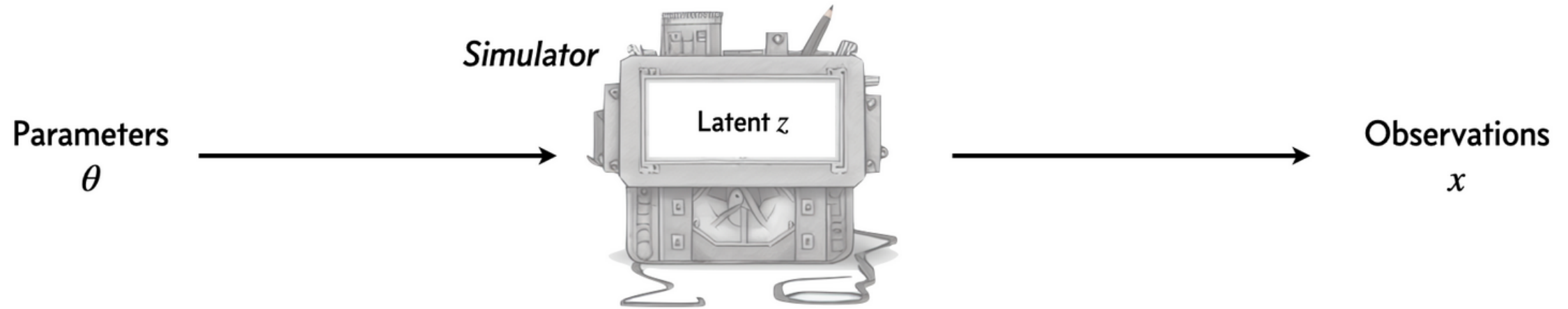
```
def simulate(v, alpha, dt=0.001):
    v_x = v * np.cos(alpha) # x velocity m/s
    v_y = v * np.sin(alpha) # y velocity m/s
    y = 1.1 + 0.3 * random.normal()
    x = 0.0

    while y > 0: # simulate until ball hits floor
        v_y += dt * -G # acceleration due to gravity
        x += dt * v_x
        y += dt * v_y

    return x + 0.25 * random.normal()
```

Simulator defines and samples this likelihood implicitly: $x_i = SIM(\theta) \sim p(x|\theta)$

Simulators, Prediction, and Inference



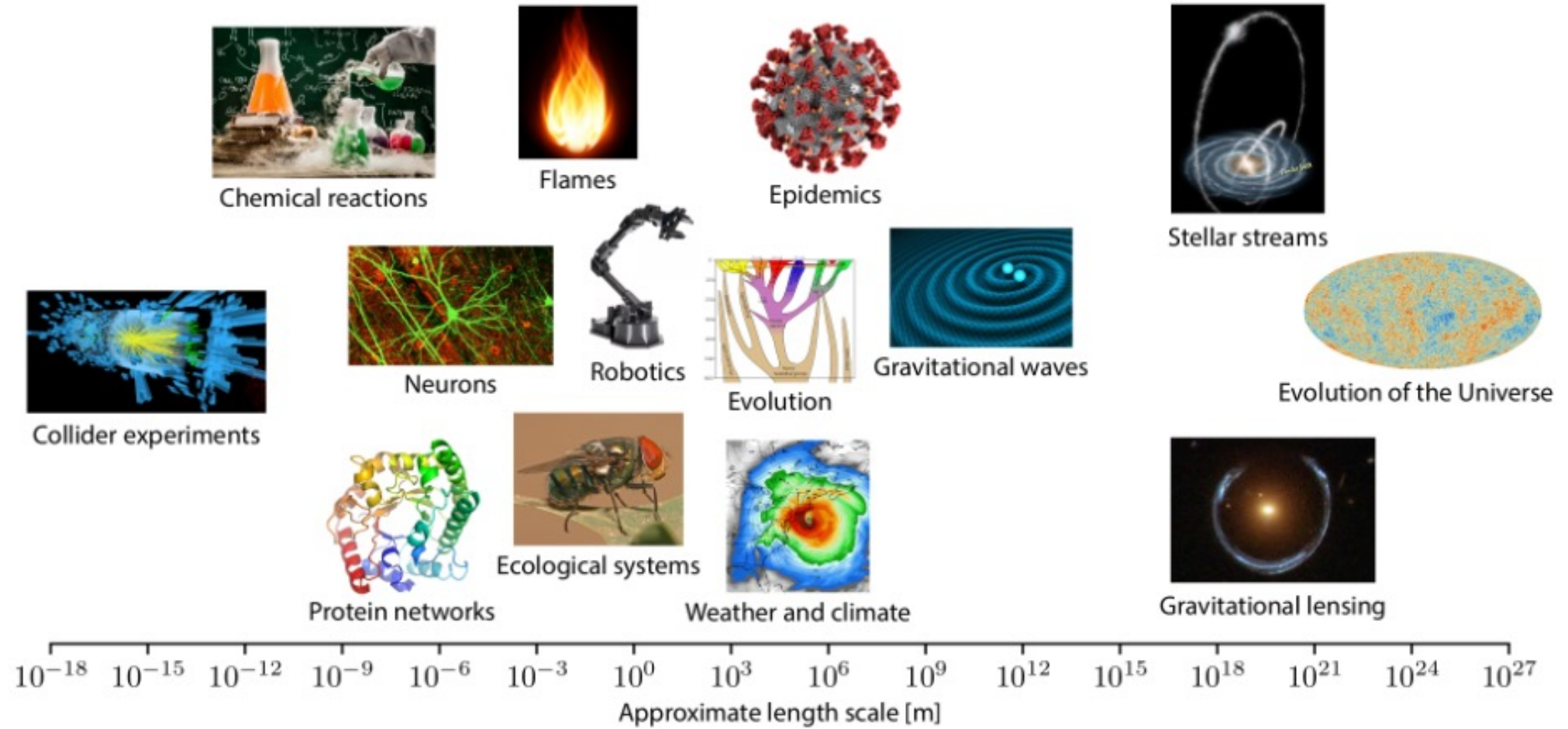
Prediction:

- Mechanistic forward model
- We can generate samples from a simulator $x \sim p(x | \theta)$

Inference:

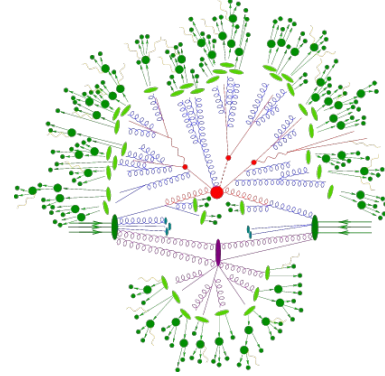
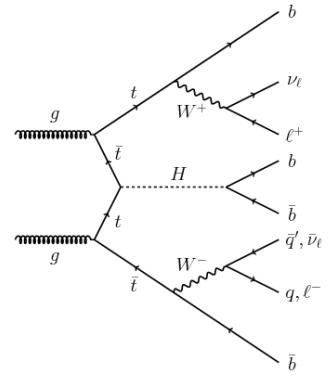
- Likelihood $p(x | \theta) = \int dz p(x, z | \theta)$ is intractable
- *Inference is challenging*

Simulators Across Science



Data Generation Process

$$\begin{aligned}
 & -\frac{1}{2}g_s^2\theta_{12}^2 - g_s^2\theta_{13}^2\theta_{23}^2 - \frac{1}{2}g_s^2\theta_{12}^2\theta_{23}^2 + \\
 & \frac{1}{2}g_s^2\theta_{12}^2\theta_{23}^2 + \theta_{12}^2\theta_{23}^2 + g_s^2\theta_{12}^2\theta_{23}^2 - \theta_{12}^2\theta_{23}^2 - \\
 & M^2W_1^2 - \frac{1}{2}M^2W_2^2 - \frac{1}{2}M^2W_3^2 - \frac{1}{2}M^2W_4^2 - \frac{1}{2}M^2W_5^2 - \\
 & \frac{1}{2}M^2W_6^2 - \frac{1}{2}M^2W_7^2 - \frac{1}{2}M^2W_8^2 - \frac{1}{2}M^2W_9^2 - \frac{1}{2}M^2W_{10}^2 - \\
 & \frac{1}{2}M^2W_{11}^2 - \frac{1}{2}M^2W_{12}^2 - \frac{1}{2}M^2W_{13}^2 - \frac{1}{2}M^2W_{14}^2 - \frac{1}{2}M^2W_{15}^2 - \\
 & \frac{1}{2}M^2W_{16}^2 - \frac{1}{2}M^2W_{17}^2 - \frac{1}{2}M^2W_{18}^2 - \frac{1}{2}M^2W_{19}^2 - \frac{1}{2}M^2W_{20}^2 - \\
 & \frac{1}{2}M^2W_{21}^2 - \frac{1}{2}M^2W_{22}^2 - \frac{1}{2}M^2W_{23}^2 - \frac{1}{2}M^2W_{24}^2 - \frac{1}{2}M^2W_{25}^2 - \\
 & \frac{1}{2}M^2W_{26}^2 - \frac{1}{2}M^2W_{27}^2 - \frac{1}{2}M^2W_{28}^2 - \frac{1}{2}M^2W_{29}^2 - \frac{1}{2}M^2W_{30}^2 - \\
 & \frac{1}{2}M^2W_{31}^2 - \frac{1}{2}M^2W_{32}^2 - \frac{1}{2}M^2W_{33}^2 - \frac{1}{2}M^2W_{34}^2 - \frac{1}{2}M^2W_{35}^2 - \\
 & \frac{1}{2}M^2W_{36}^2 - \frac{1}{2}M^2W_{37}^2 - \frac{1}{2}M^2W_{38}^2 - \frac{1}{2}M^2W_{39}^2 - \frac{1}{2}M^2W_{40}^2 - \\
 & \frac{1}{2}M^2W_{41}^2 - \frac{1}{2}M^2W_{42}^2 - \frac{1}{2}M^2W_{43}^2 - \frac{1}{2}M^2W_{44}^2 - \frac{1}{2}M^2W_{45}^2 - \\
 & \frac{1}{2}M^2W_{46}^2 - \frac{1}{2}M^2W_{47}^2 - \frac{1}{2}M^2W_{48}^2 - \frac{1}{2}M^2W_{49}^2 - \frac{1}{2}M^2W_{50}^2 - \\
 & \frac{1}{2}M^2W_{51}^2 - \frac{1}{2}M^2W_{52}^2 - \frac{1}{2}M^2W_{53}^2 - \frac{1}{2}M^2W_{54}^2 - \frac{1}{2}M^2W_{55}^2 - \\
 & \frac{1}{2}M^2W_{56}^2 - \frac{1}{2}M^2W_{57}^2 - \frac{1}{2}M^2W_{58}^2 - \frac{1}{2}M^2W_{59}^2 - \frac{1}{2}M^2W_{60}^2 - \\
 & \frac{1}{2}M^2W_{61}^2 - \frac{1}{2}M^2W_{62}^2 - \frac{1}{2}M^2W_{63}^2 - \frac{1}{2}M^2W_{64}^2 - \frac{1}{2}M^2W_{65}^2 - \\
 & \frac{1}{2}M^2W_{66}^2 - \frac{1}{2}M^2W_{67}^2 - \frac{1}{2}M^2W_{68}^2 - \frac{1}{2}M^2W_{69}^2 - \frac{1}{2}M^2W_{70}^2 - \\
 & \frac{1}{2}M^2W_{71}^2 - \frac{1}{2}M^2W_{72}^2 - \frac{1}{2}M^2W_{73}^2 - \frac{1}{2}M^2W_{74}^2 - \frac{1}{2}M^2W_{75}^2 - \\
 & \frac{1}{2}M^2W_{76}^2 - \frac{1}{2}M^2W_{77}^2 - \frac{1}{2}M^2W_{78}^2 - \frac{1}{2}M^2W_{79}^2 - \frac{1}{2}M^2W_{80}^2 - \\
 & \frac{1}{2}M^2W_{81}^2 - \frac{1}{2}M^2W_{82}^2 - \frac{1}{2}M^2W_{83}^2 - \frac{1}{2}M^2W_{84}^2 - \frac{1}{2}M^2W_{85}^2 - \\
 & \frac{1}{2}M^2W_{86}^2 - \frac{1}{2}M^2W_{87}^2 - \frac{1}{2}M^2W_{88}^2 - \frac{1}{2}M^2W_{89}^2 - \frac{1}{2}M^2W_{90}^2 - \\
 & \frac{1}{2}M^2W_{91}^2 - \frac{1}{2}M^2W_{92}^2 - \frac{1}{2}M^2W_{93}^2 - \frac{1}{2}M^2W_{94}^2 - \frac{1}{2}M^2W_{95}^2 - \\
 & \frac{1}{2}M^2W_{96}^2 - \frac{1}{2}M^2W_{97}^2 - \frac{1}{2}M^2W_{98}^2 - \frac{1}{2}M^2W_{99}^2 - \frac{1}{2}M^2W_{100}^2
 \end{aligned}$$



O(20) Fundamental physics parameters θ

O(10) particles

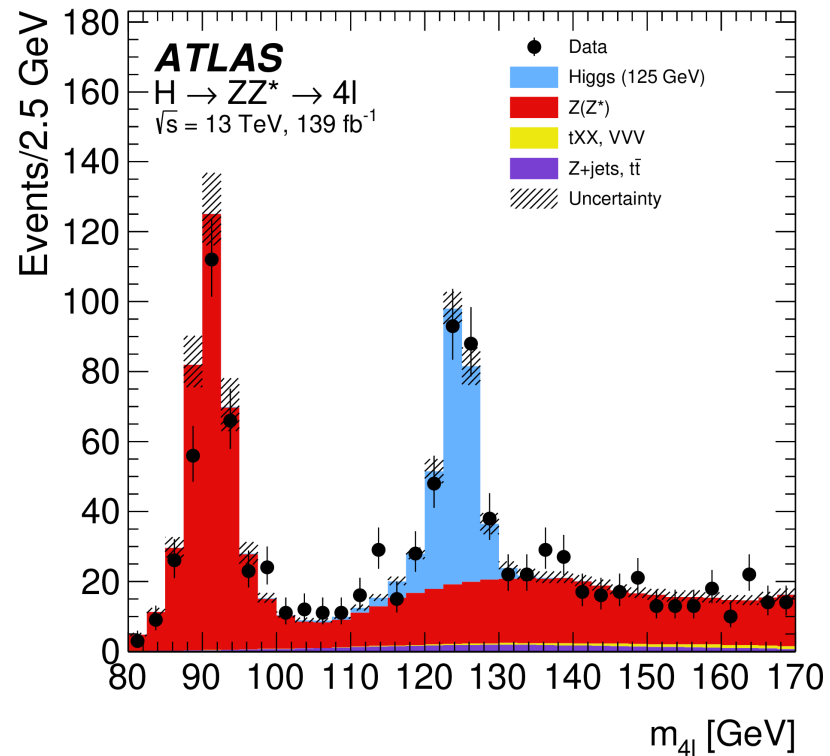
O(100) particles

Evolution (parton shower, hadronization)

$$p(z_h | z_p) p(z_p | \theta)$$

What do we do with samples of $x \sim p(x|\theta)$?

Generate lots of samples \rightarrow Can use samples to approximate things!



Calculate useful statistics

- Mean, variance, ...

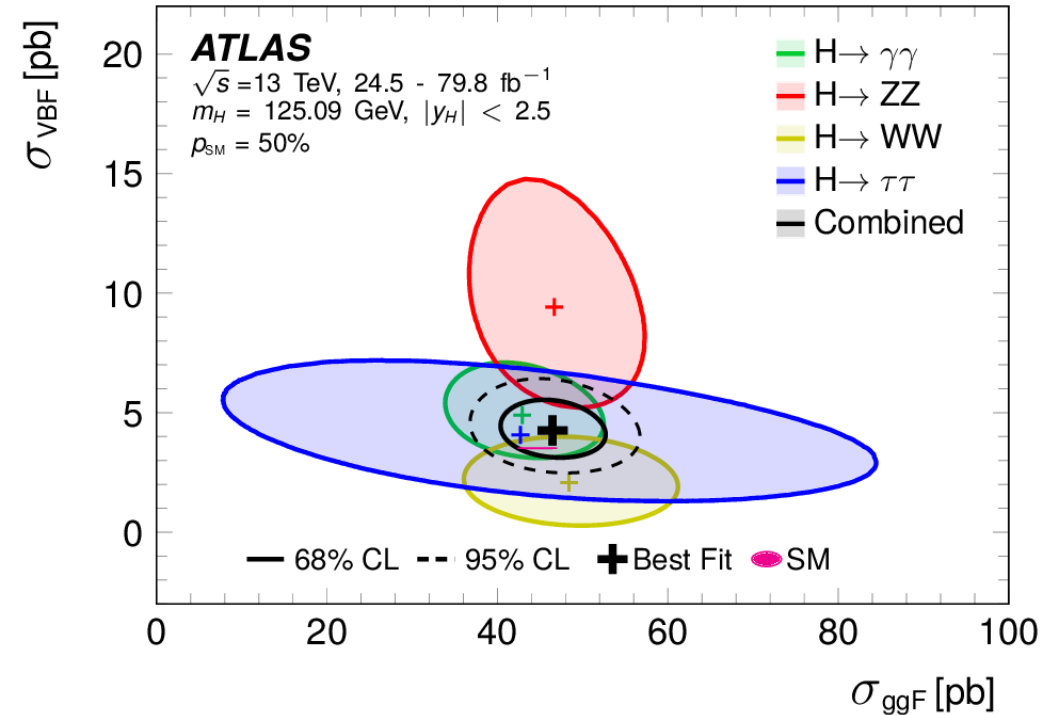
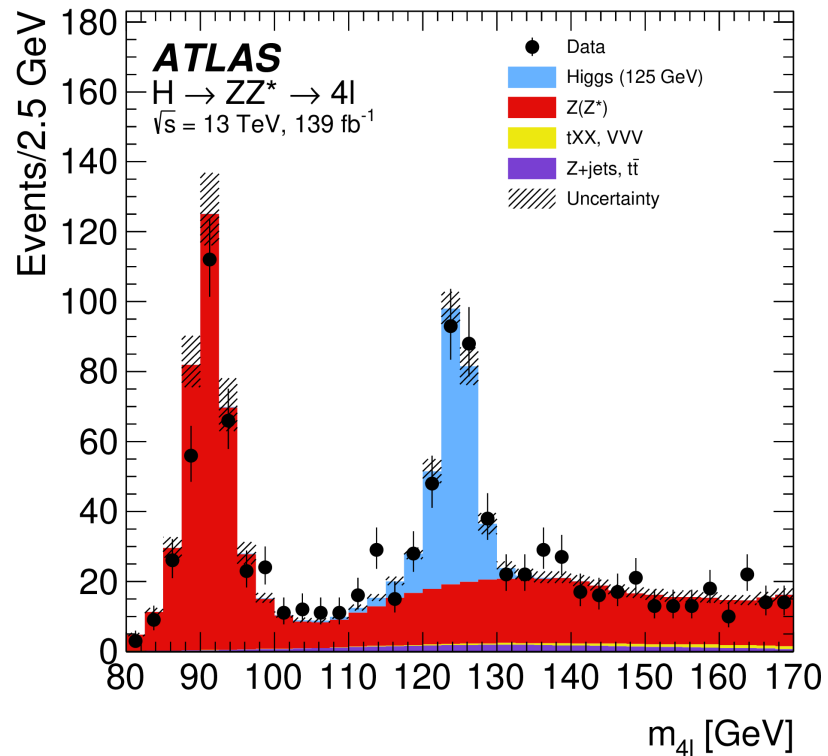
Density estimation

Can compare with observed data and use for inference

We do this all the time at the LHC!

Choose a good 1D summary statistic that is sensitive to the parameter

Make Histogram \rightarrow Density estimation \Rightarrow Estimate likelihood for inference



This is an example of Simulation Based Inference!

Simulation-based Inference

Simulation-Based Inference

When we don't have a tractable likelihood, but have a "good" simulator

Given:

- Simulator that can generate samples

$$x_i \sim p(x_i | \theta_i)$$

- Observed data

$$x_{obs} \sim p(x_{obs} | \theta_{true})$$

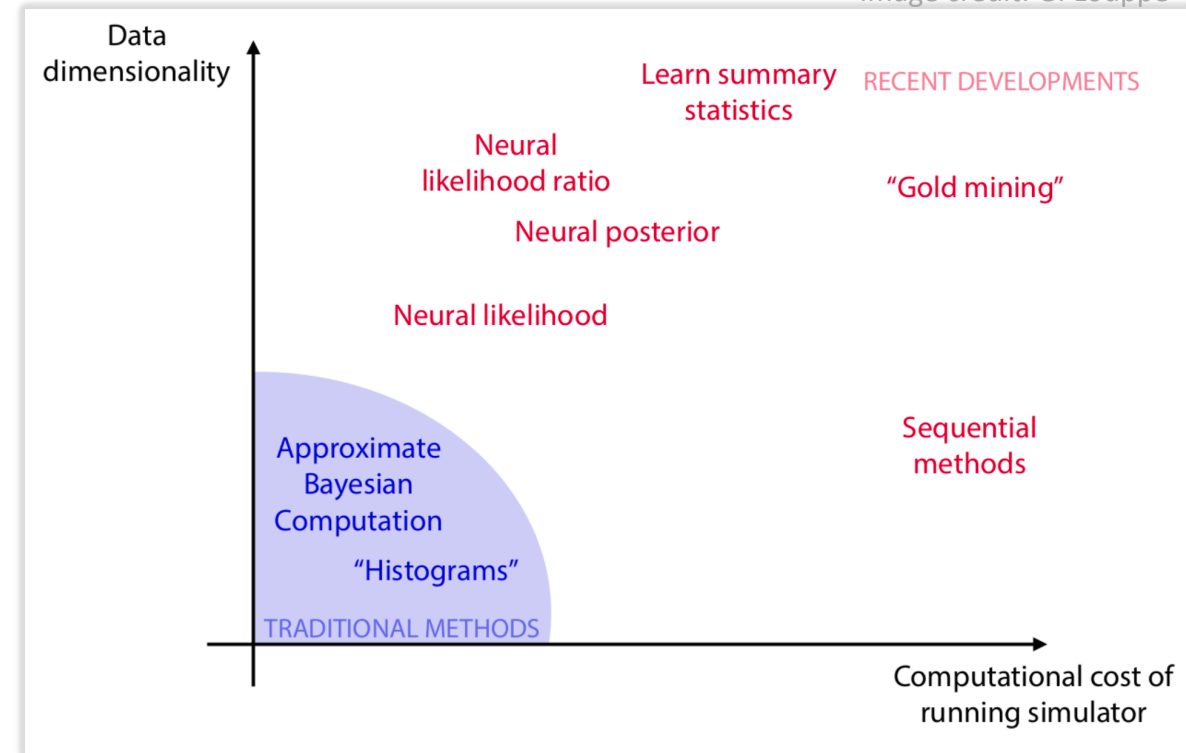
Estimate Likelihood ratio

$$r(x | \theta_1, \theta_2) = \frac{p(x_{obs} | \theta_0)}{p(x_{obs} | \theta_1)}$$

Or Posterior (also given prior $p(\theta)$)

$$p(\theta | x_{obs}) = \frac{p(x_{obs} | \theta) p(\theta)}{p(x_{obs})}$$

Image credit: G. Louppe



Simplest Method: Density Estimation in Low Dimensions

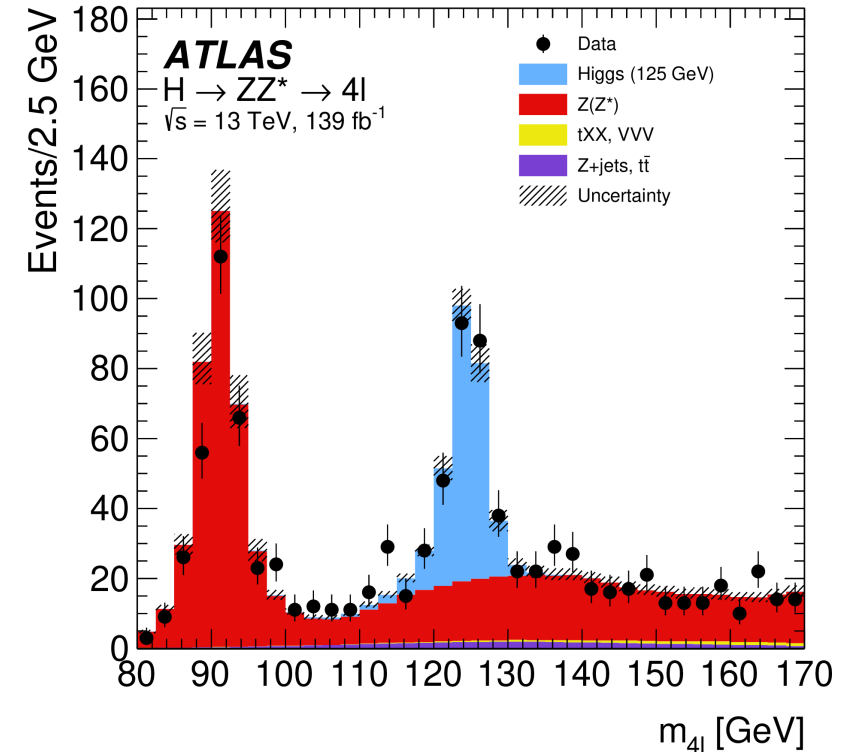
Estimate likelihoods using histograms or kernel density estimator (KDE)

Choose feature that discriminate hypotheses

Challenge

- Histograms / KDE don't scale to high-dims
- Typically use 1D density
- Can be big (lossy) compression of information

At LHC, compress $\mathbb{R}^{10^8} \rightarrow \mathbb{R}^1$



Simplest Method: Density Estimation in Low Dimensions

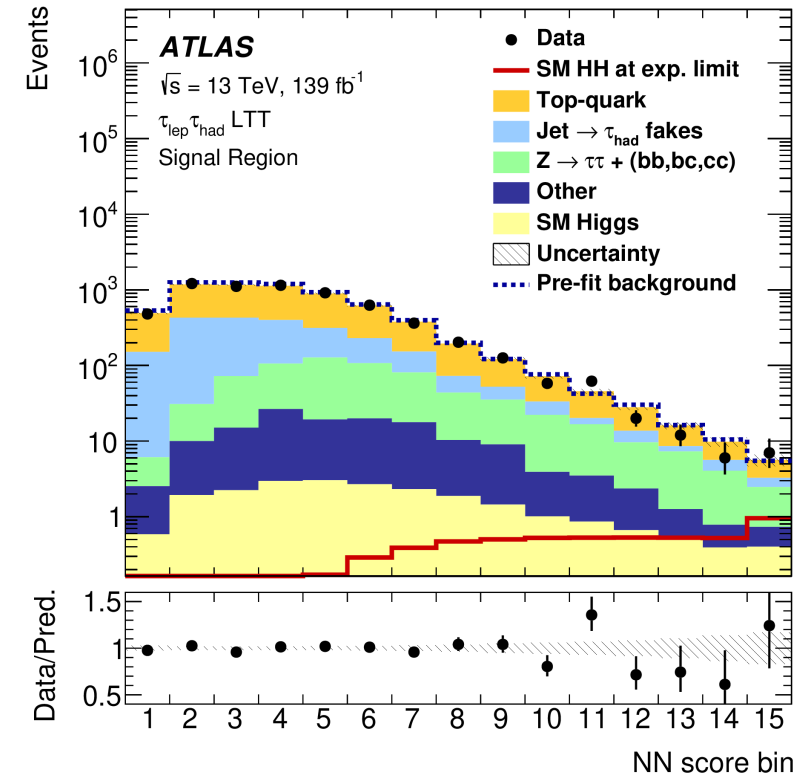
Estimate likelihoods using histograms or kernel density estimator (KDE)

Choose feature that discriminate hypotheses

Challenge

- Histograms / KDE don't scale to high-dims
- Typically use 1D density
- Can be big (lossy) compression of information

At LHC, compress $\mathbb{R}^{10^8} \rightarrow \mathbb{R}^1$



Sometimes can be smart about choice of feature...

e.g. use NN output score

Approximate Bayesian Computation (ABC)

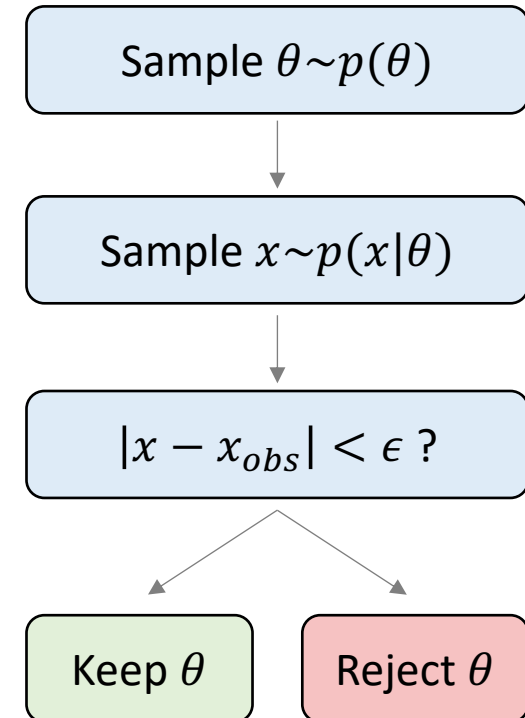
A method to sample approximate posterior

Idea:

Keep θ 's that produce samples x close to the observed x_{obs}

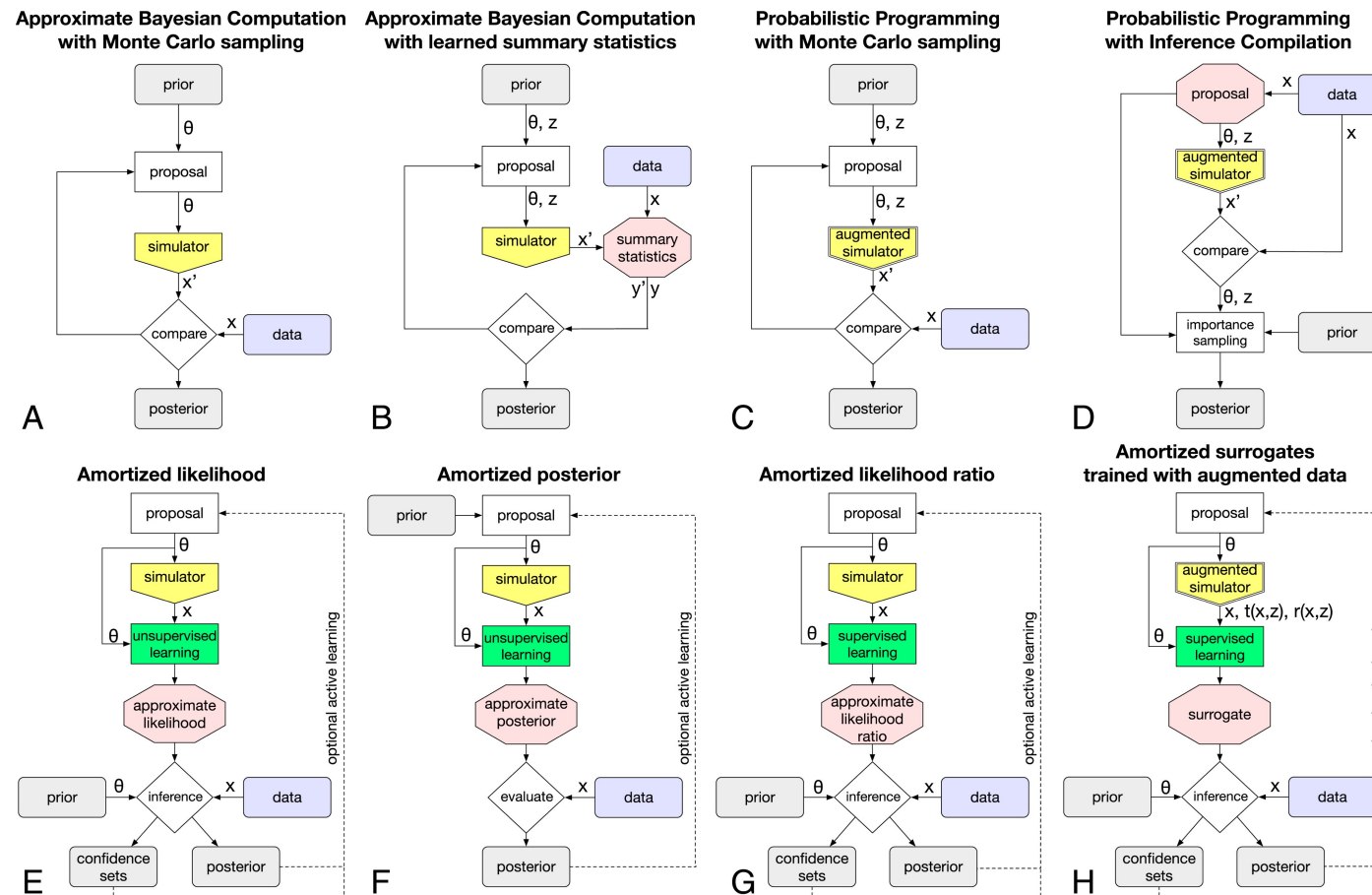
Challenge:

- Often very inefficient and approximate:
In high-dims, unlikely to simulate $x \approx x_{obs}$
- Often use summary statistic $s(x)$:
Which summary to use?

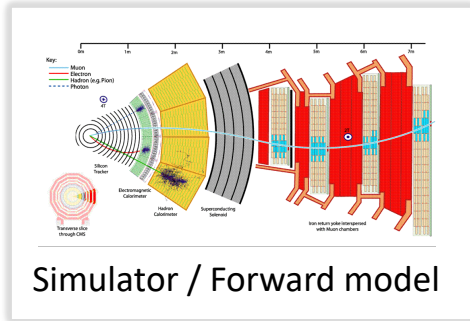


Simulation-Based Inference with ML

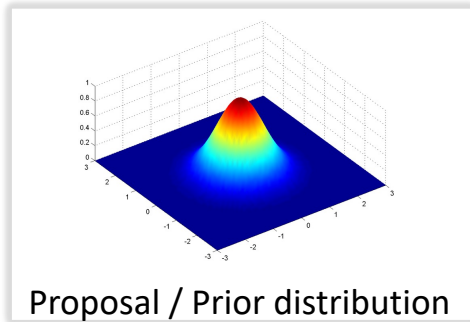
Use simulator to train neural network to approximate likelihood, posterior, or likelihood ratio. Then use neural net for inference on observed data



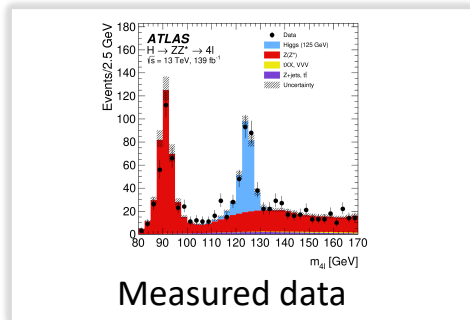
$$x \sim p(x|\theta)$$



$$\pi(\theta)$$



$$y$$



Likelihood ratio estimation

$$r_\phi(x|\theta, \theta') \approx \frac{p(x|\theta)}{p(x|\theta')}$$

Likelihood estimation

$$q_\phi(x|\theta) \approx p(x|\theta)$$

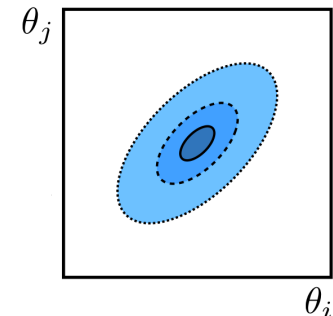
Posterior estimation

$$q_\phi(\theta|x) \approx p(\theta|x) \propto p(x|\theta)\pi(\theta)$$

Neural Net

Evaluate model on measured data

Posterior / Confidence intervals



Neural Ratio Estimation (NRE):

- Estimate the density ratio $\frac{p(x|\theta)}{p(x)}$ or $\frac{p(x|\theta_0)}{p(x|\theta_1)}$ using neural network classifier
- Can be used for Frequentist inference
- Can be used to draw samples from posterior with sampling algorithm

Neural Ratio Estimation (NRE):

- Estimate the density ratio $\frac{p(x|\theta)}{p(x)}$ or $\frac{p(x|\theta_0)}{p(x|\theta_1)}$ using neural network classifier
- Can be used for Frequentist inference
- Can be used to draw samples from posterior with sampling algorithm

Neural Likelihood Estimation (NLE):

- Estimate the likelihood $p(x|\theta)$ using a normalizing flow
- Can be used to draw samples from posterior with sampling algorithm

Neural Ratio Estimation (NRE):

- Estimate the density ratio $\frac{p(x|\theta)}{p(x)}$ or $\frac{p(x|\theta_0)}{p(x|\theta_1)}$ using neural network classifier
- Can be used for Frequentist inference
- Can be used to draw samples from posterior with sampling algorithm

Neural Likelihood Estimation (NLE):

- Estimate the likelihood $p(x|\theta)$ using a normalizing flow
- Can be used to draw samples from posterior with sampling algorithm

Neural Posterior Estimation (NPE):

- Estimate the posterior $p(\theta|x)$ using a normalizing flow
- Can draw samples posterior samples directly

Neural Posterior Estimation (NPE) Neural Likelihood Estimation (NLE)

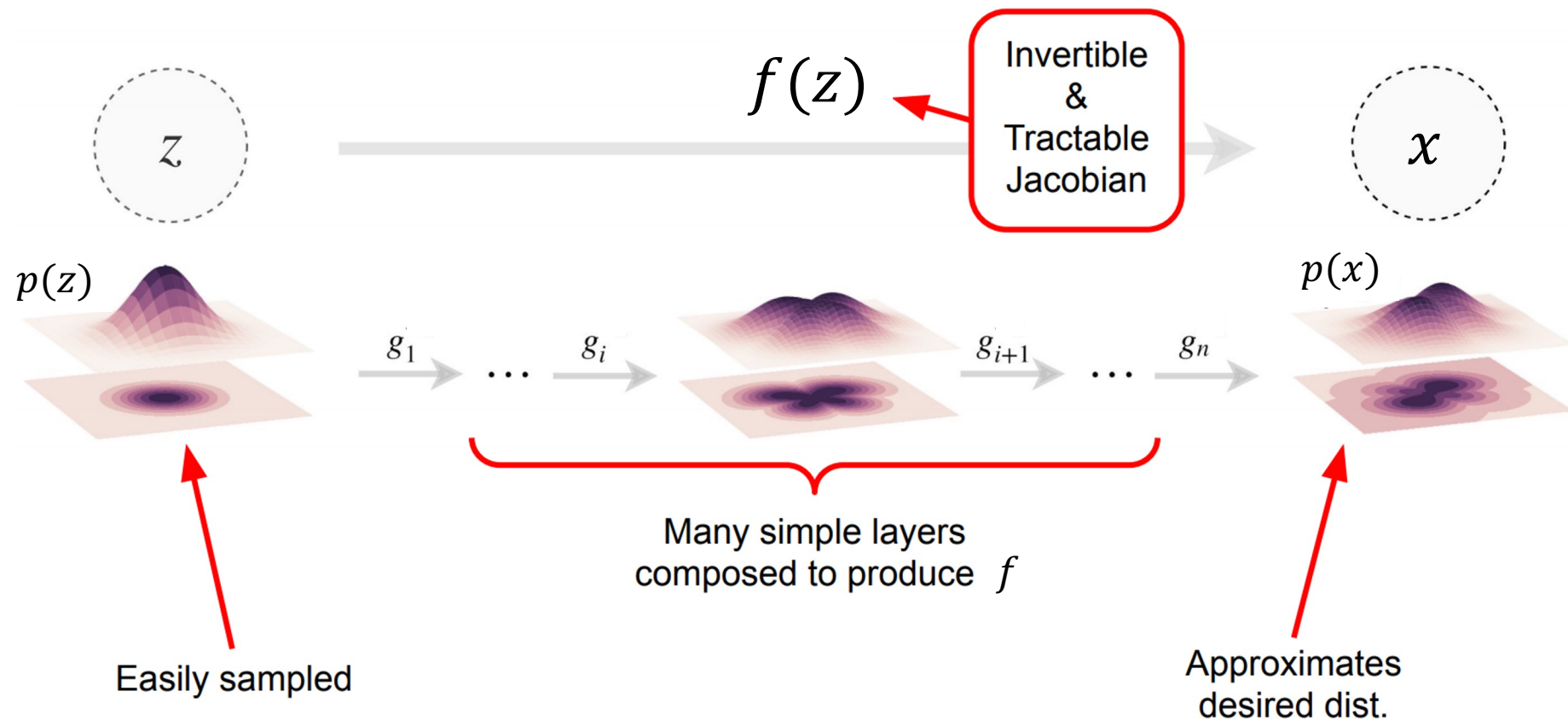
$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$$

We need to estimate densities for NPE and NLE

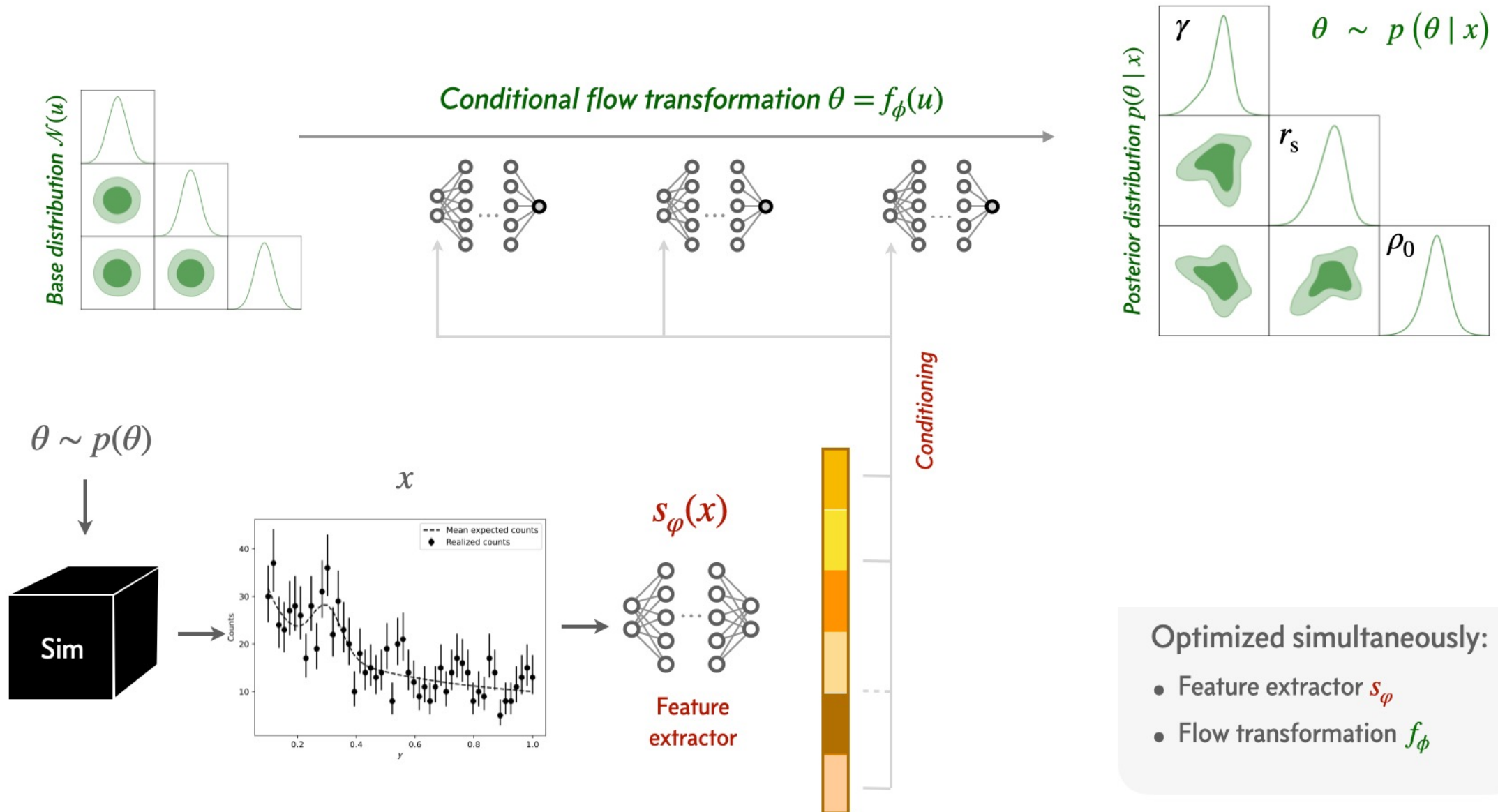
Choose a flexible density estimator: $q_\phi(\cdot)$ \rightarrow Normalizing Flow

Normalizing Flows

Series of *easily invertible bijections*: $x = f(z)$ $p_x(x) = p_z(z) \left| \det \left(\frac{\partial f(z)}{\partial z} \right)^{-1} \right|$
Can both generate events and evaluate density

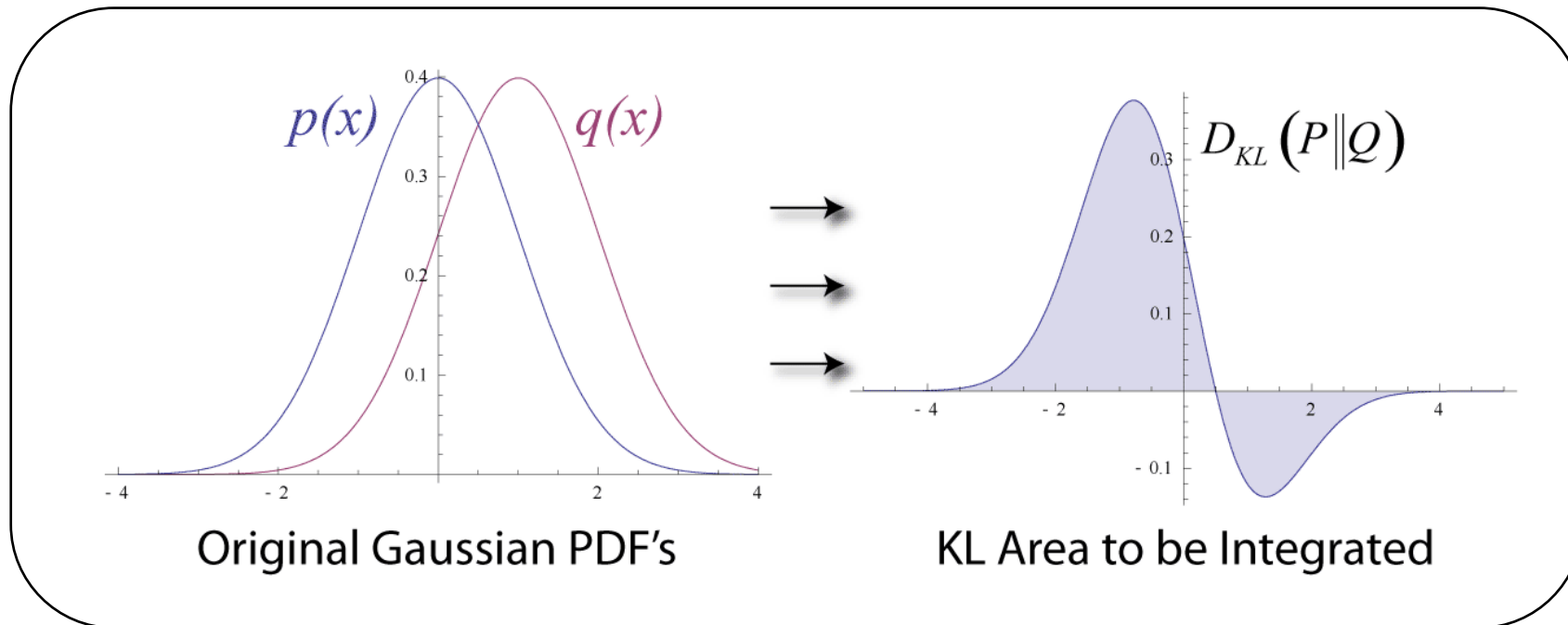


Conditional Neural Posterior Density Estimation



$$KL(p(\theta|x) || q_\phi(\theta|x))$$

$$= \mathbb{E}_{p(\theta|x)} \left[\log \left(\frac{p(\theta|x)}{q_\phi(\theta|x)} \right) \right]$$



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

Amortize:

Want model to work for different possible x

→ Take expected value over x

$$\begin{aligned} & \mathbb{E}_{p(x)} \left[KL \left(p(\theta|x) \parallel q_\phi(\theta|x) \right) \right] \\ &= \mathbb{E}_{p(x)} \mathbb{E}_{p(\theta|x)} \left[\log \left(\frac{p(\theta|x)}{q_\phi(\theta|x)} \right) \right] \\ &= \mathbb{E}_{p(x,\theta)} \left[-\log q_\phi(\theta|x) \right] + \text{const}_{w.r.t \phi} \end{aligned}$$

$$\begin{aligned} \min_{\phi} \mathbb{E}_{p(x)} \left[KL \left(p(\theta|x) \parallel q_{\phi}(\theta|x) \right) \right] \\ &= \min_{\phi} \mathbb{E}_{p(x)} \mathbb{E}_{p(\theta|x)} \left[\log \left(\frac{p(\theta|x)}{q_{\phi}(\theta|x)} \right) \right] \\ &= \min_{\phi} \mathbb{E}_{p(x,\theta)} \left[-\log q_{\phi}(\theta|x) \right] + \text{const}_{w.r.t \phi} \end{aligned}$$

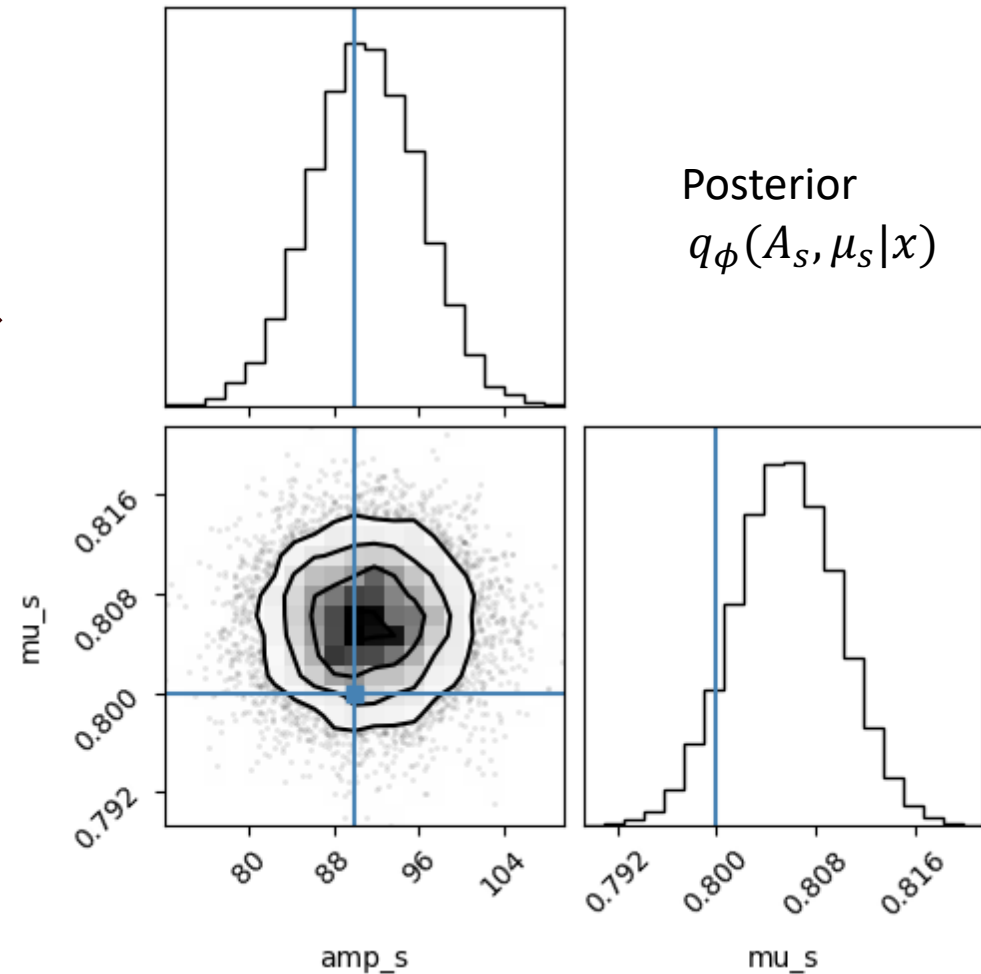
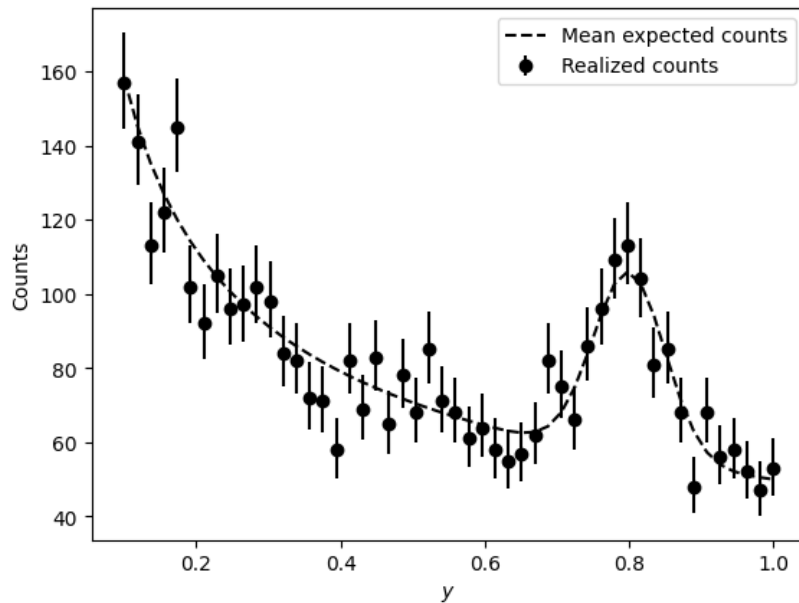
Using Neural Posterior Estimation (NPE)

Once trained, we can sample the model $q_\phi(\theta|x_{obs})$ directly

$$x_b = 50y^{-0.5}$$

$$x_s = A_s e^{-\frac{(y-\mu_s)^2}{2(0.05)^2}}$$

$$x \sim p(x|A_s, \mu_s) = \text{Pois}(x_b + x_s)$$

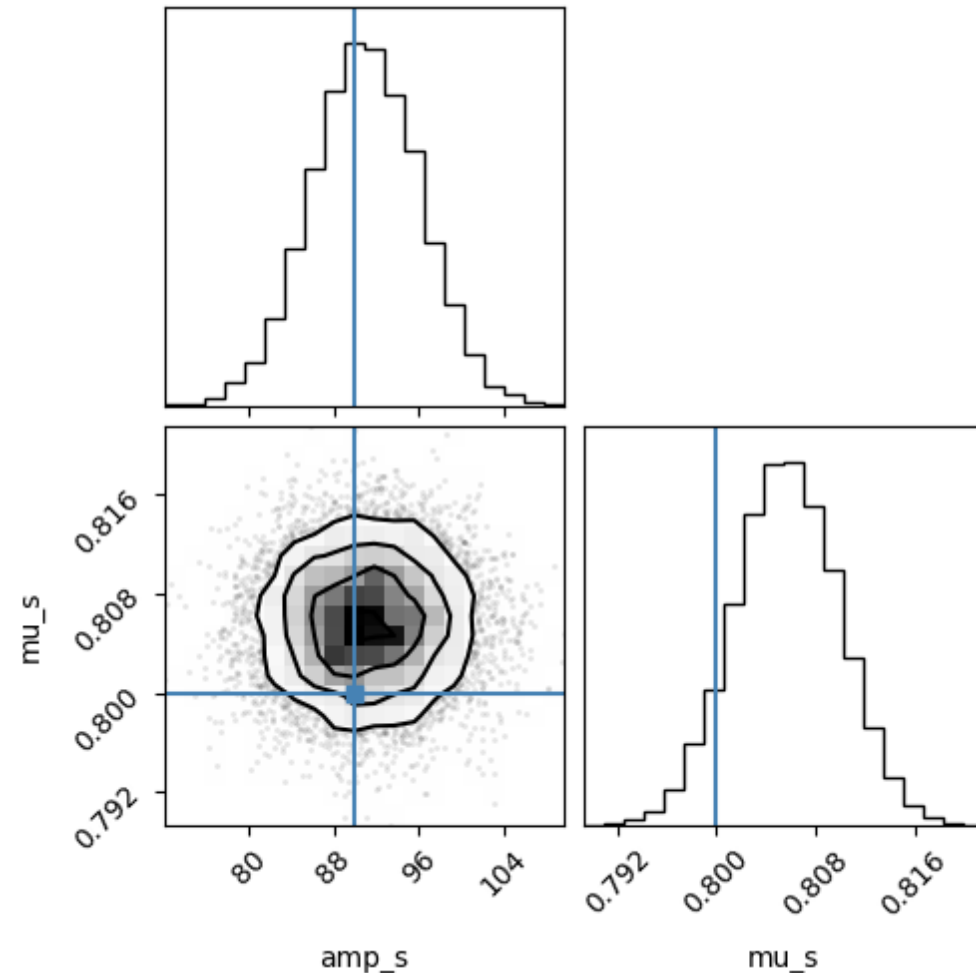


Using Neural Posterior Estimation (NPE)

Once trained, we can sample the model $q_{\phi}(\theta|x_{obs})$ directly

Challenges:

- Must retrain if we change prior
- If prior and posterior are very different, may not learn density well because most simulated samples carry little information about posterior
- In high dimensions, can be hard to learn density & require many samples



$$\begin{aligned} \min_{\phi} \mathbb{E}_{p(\theta)} \left[KL \left(p(x|\theta) \parallel q_{\phi}(x|\theta) \right) \right] \\ &= \min_{\phi} \mathbb{E}_{p(\theta)} \mathbb{E}_{p(x|\theta)} \left[\log \left(\frac{p(x|\theta)}{q_{\phi}(x|\theta)} \right) \right] \\ &= \min_{\phi} \mathbb{E}_{p(x,\theta)} \left[-\log q_{\phi}(x|\theta) \right] + \text{const}_{w.r.t \phi} \end{aligned}$$

Using NLE is more complicated...

For Frequentist inference, we could evaluate: $r(x) = \frac{q(x|\theta_0)}{q(x|\theta_1)}$

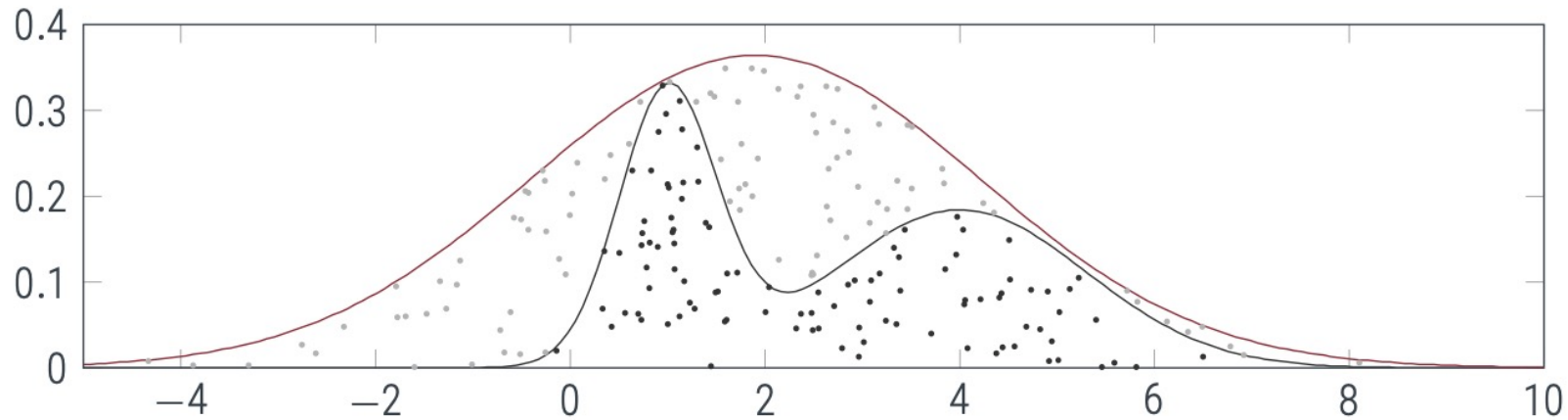
But as we will see soon... there is another (easier) way to estimate the ratio

For Bayesian inference, we want: $q(\theta|x) \propto q_\phi(x|\theta)p(\theta)$

To estimate this posterior, need to sample this (unnormalized) distribution

Interlude: (Brief) Introduction to Sampling Methods

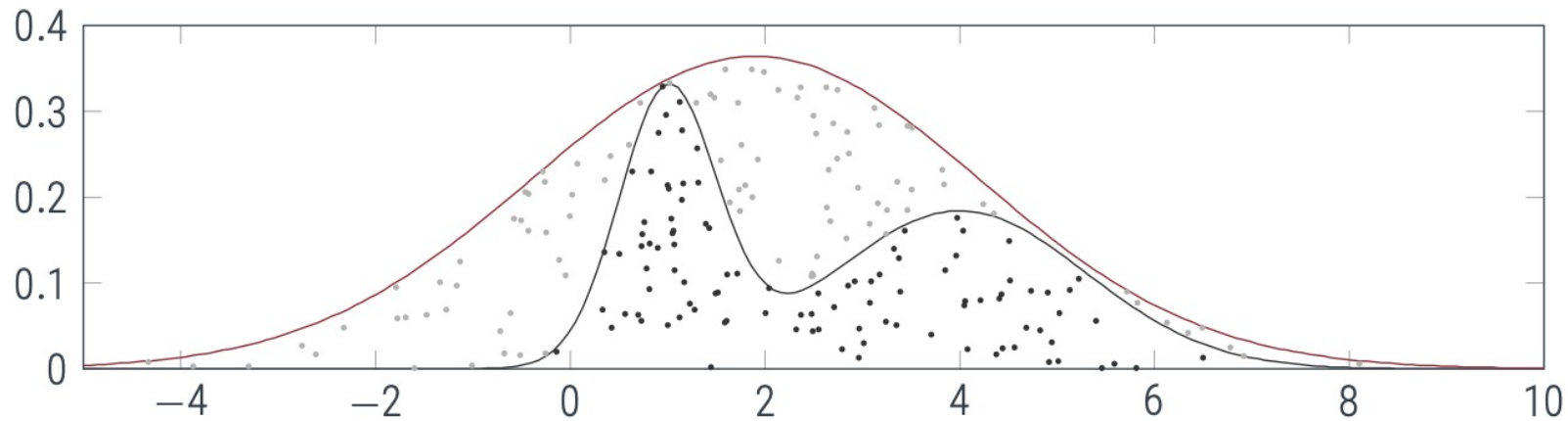
Rejection Sampling



Want to sample $p(x) = \tilde{p}(x)/Z$ (normalization Z not required)

Algorithm:

- Choose $q(x)$ such that $cq(x) \geq \tilde{p}(x)$
- Sample $x_s \sim q(x_s)$
- Sample $u \sim \text{Uniform}[0, cq(x_s)]$
- Reject if $u > \tilde{p}(x)$



Want to sample $p(x) = \tilde{p}(x)/Z$ (normalization Z not required)

Algorithm:

- Choose $q(x)$ such that $cq(x) \geq \tilde{p}(x)$
- Sample $x_s \sim q(x_s)$
- Sample $u \sim \text{Uniform}[0, cq(x_s)]$
- Reject if $u > \tilde{p}(x)$

Challenges:

- If $q(x)$ very different from $\tilde{p}(x)$, rejection rate is high
- Rejection rate can rise exponentially with dimension of x

$$\int f(x)p(x)dx = \int f(x) \frac{p(x)}{q(x)} q(x)dx$$

$$\approx \frac{1}{N} \sum_N f(x_s) \frac{p(x_s)}{q(x_s)} \equiv \frac{1}{N} \sum_N f(x_s) w_s$$

Where $x_s \sim q(x)$

Algorithm:

- Choose proposal $q(x)$
- Sample $x_s \sim q(x)$
- Compute importance weight $w_s = \frac{p(x_s)}{q(x_s)}$
- Weight event by w_s in computations

$$\int f(x)p(x)dx = \int f(x) \frac{p(x)}{q(x)} q(x)dx$$

$$\approx \frac{1}{N} \sum_N f(x_s) \frac{p(x_s)}{q(x_s)} \equiv \frac{1}{N} \sum_N f(x_s) w_s \quad \text{Where } x_s \sim q(x)$$

Algorithm:

- Choose proposal $q(x)$
- Sample $x_s \sim q(x)$
- Compute importance weight $w_s = \frac{p(x_s)}{q(x_s)}$
- Weight event by w_s in computations

Challenges:

- If $q(x) \ll p(x)$ then $w_s \rightarrow \infty$
- If $q(x)$ very different from $p(x)$, large variance: $\text{var}_q\left(f \frac{p}{q}\right)$

Happens frequently in high dims

Joint distribution of sequence $[x_1, \dots, x_n]$ has the **Markov Property** if

$$p(x_{i+1} | x_i, x_{i-1}, \dots, x_1) = p(x_{i+1} | x_i)$$

The sequence is called a **Markov Chain**



Metropolis-Hastings (MH) Markov Chain Monte Carlo (MCMC)

Want to sample a (possibly un-normalized) distribution $p(x)$

We'll build a Markov Chain of samples $\{x_t\}$

Transition to a new sample with a transition probability $q(x_{t+1}|x_t)$

Samples will no longer be independent

The distribution of sample x_t will approach $p(x)$ as $t \rightarrow \infty$

- i.e. the *equilibrium* or *stationary distribution* of the chain will be $p(x)$

Metropolis-Hastings

Want to sample from $p(x)$ (possibly un-normalized)

Choose proposal distribution $q(x'|x_t)$

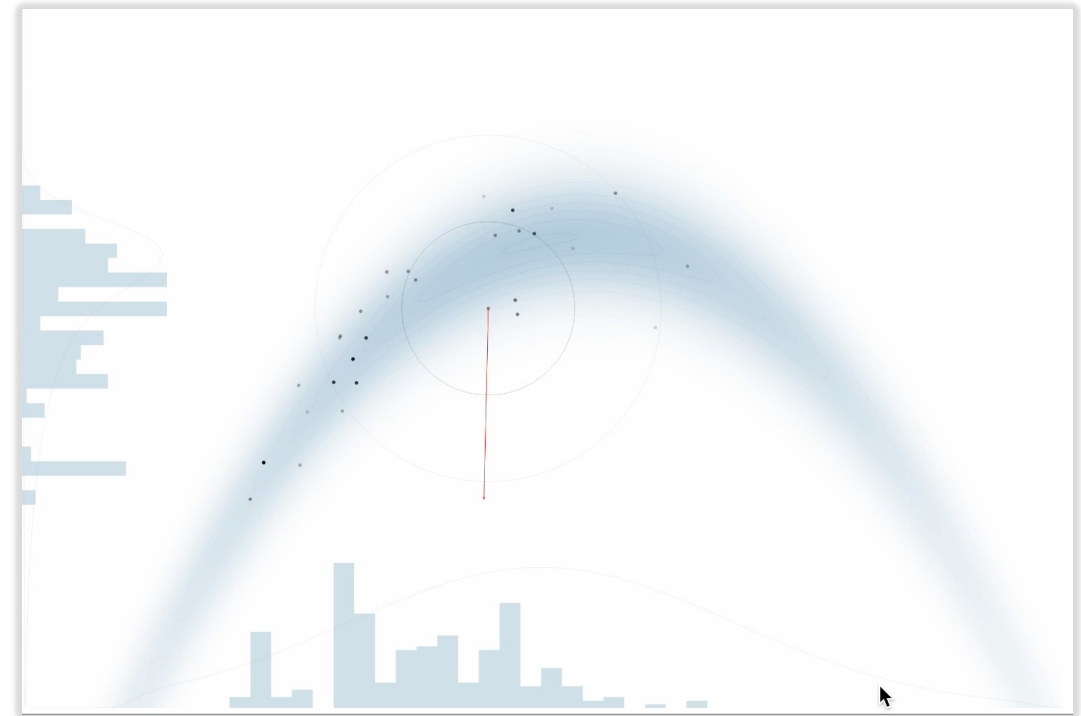
- For example, $q(x'|x_t) = \mathcal{N}(x'|x_t, \sigma_t)$

Algorithm:

- Given current sample x_t
- Draw proposal $x' \sim q(x'|x_t)$
- Evaluate

$$a = \min \left[1, \frac{p(x') q(x_t|x')}{p(x_t) q(x'|x_t)} \right]$$

- Accept with probability a : $x_{t+1} \leftarrow x'$
- Stay with probability $1 - a$: $x_{t+1} \leftarrow x_t$



MH is performing a (biased) random walk

Want to sample from $p(x)$ (possibly un-normalized)

Choose proposal distribution $q(x'|x_t)$

- For example, $q(x'|x_t) = \mathcal{N}(x'|x_t, \sigma_t)$

Algorithm:

- Given current sample x_t
- Draw proposal $x' \sim q(x'|x_t)$
- Evaluate

$$a = \min \left[1, \frac{p(x') q(x_t|x')}{p(x_t) q(x'|x_t)} \right]$$

- Accept with probability a : $x_{t+1} \leftarrow x'$
- Stay with probability $1 - a$: $x_{t+1} \leftarrow x_t$

Challenges:

- Requires a warm-up period before chain sufficiently converges to sampling from $p(x)$
- Depending on structure of $p(x)$, can be difficult for proposal to find a new sample with high a
- If $p(x)$ has “islands”, hard to traverse

Only Touched the Surface

Many Other Sampling Algorithms

- Hamiltonian Monte Carlo
- No-U-Turn Sampler
- Nested Sampling
- Sequential Monte Carlo
-

Interesting area of work, encourage you to check it out!

End Interlude

A note on Sequential NPE / NLE

If the prior distribution is very different from the posterior

- Many of the simulations will be in low density regions of the posterior
- Both NPE and NLE will use a lot of capacity of the neural network to learn about low density regions of the posterior / likelihood
- Ultimately this can lead to bad posterior approximations

To tackle this problem, *Sequential* versions of NPE / NLE has been proposed, to train the model in stages, where later stages use current posterior as a proposal for simulation

- Don't have time to cover here, but worth having a read!

Density Estimation is Hard in High Dimensions!

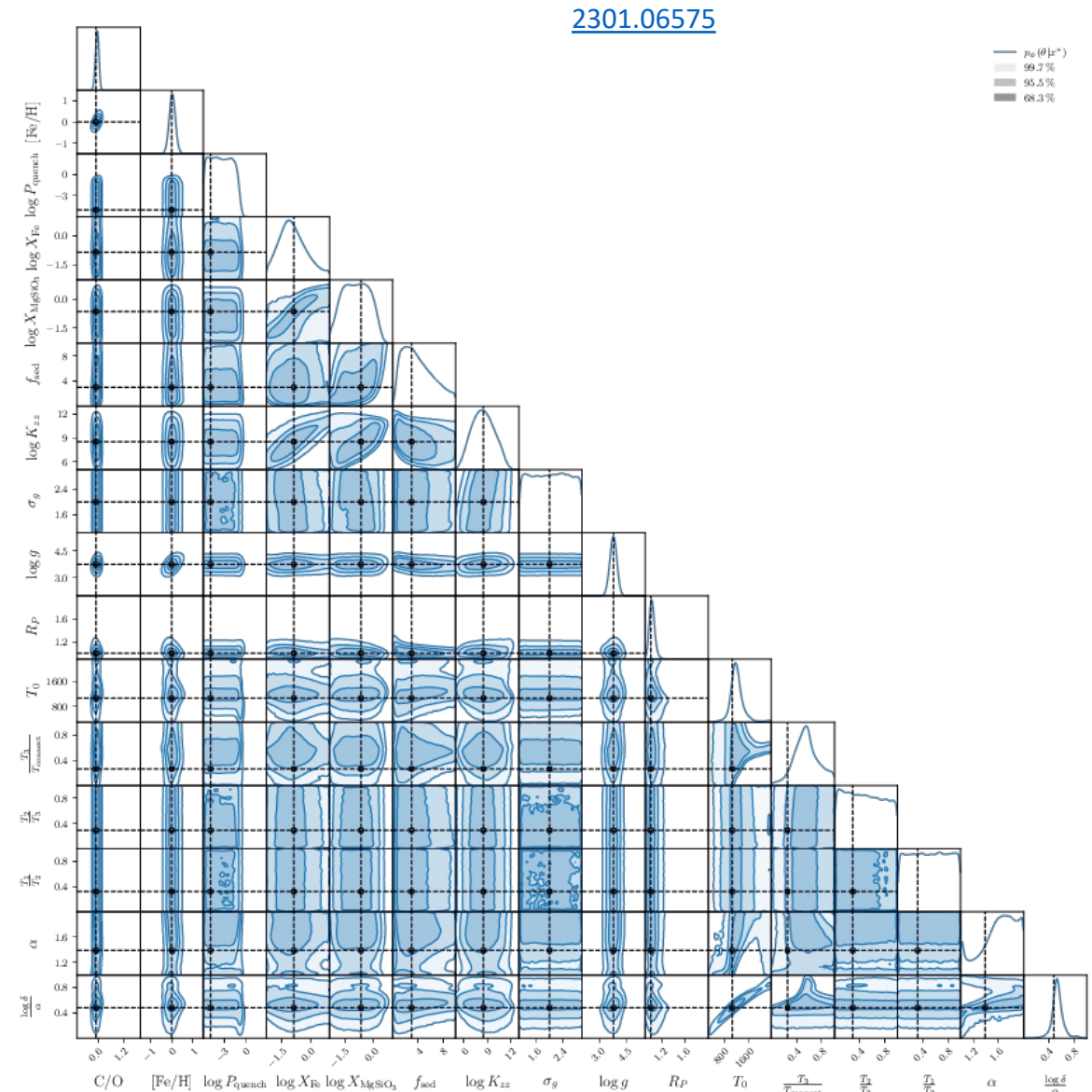
Estimating likelihood or posterior in high dimensions is hard!

Solution 1:

- Learn summaries $s(x)$ instead of x directly

Solution 2:

- Don't learn densities



Simulation-based inference is a set of methods for doing parameter inference when the likelihood is intractable, but we can simulate the process to generate simulated data

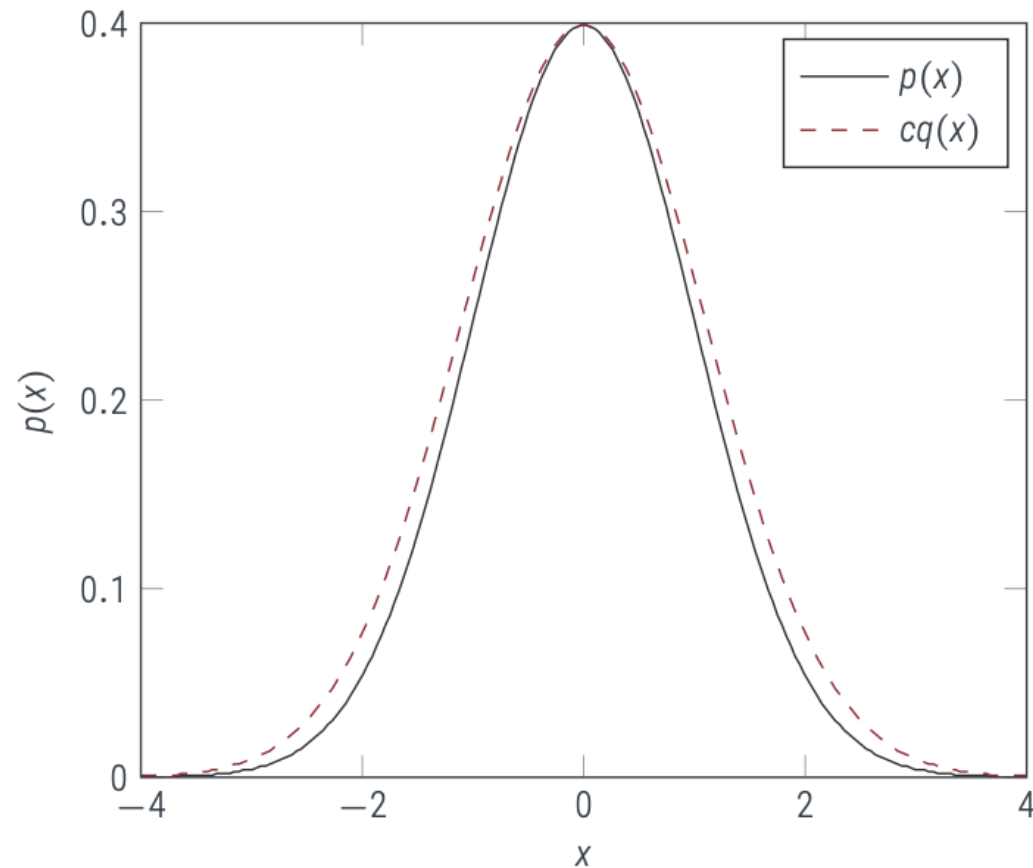
Situation arises frequently in science, where we have high fidelity simulators

So far we looked at Neural Likelihood and Posterior estimation, which required using density estimation techniques, e.g. Normalizing Flows

Next time, we will look at Neural Ratio Estimation

Backup

The Problem with Rejection Sampling



Example:

- ▶ $p(x) = \mathcal{N}(x; 0, \sigma_p^2)$
- ▶ $q(x) = \mathcal{N}(x; 0, \sigma_q^2)$
- ▶ $\sigma_q > \sigma_p$
- ▶ optimal c is given by

$$c = \frac{(2\pi\sigma_q^2)^{D/2}}{(2\pi\sigma_p^2)^{D/2}} = \left(\frac{\sigma_q}{\sigma_p}\right)^D = \exp\left(D \ln \frac{\sigma_q}{\sigma_p}\right)$$

- ▶ acceptance rate is ratio of volumes: $1/c$
- ▶ rejection rate rises **exponentially** in D
- ▶ for $\sigma_q/\sigma_p = 1.1$, $D = 100$, $1/c < 10^{-4}$