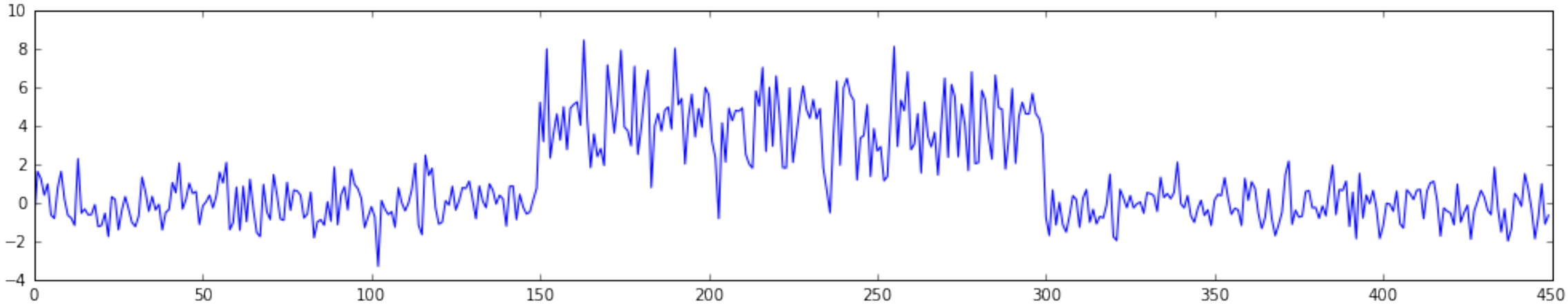
Switching to Sampling..

...in order to Switch



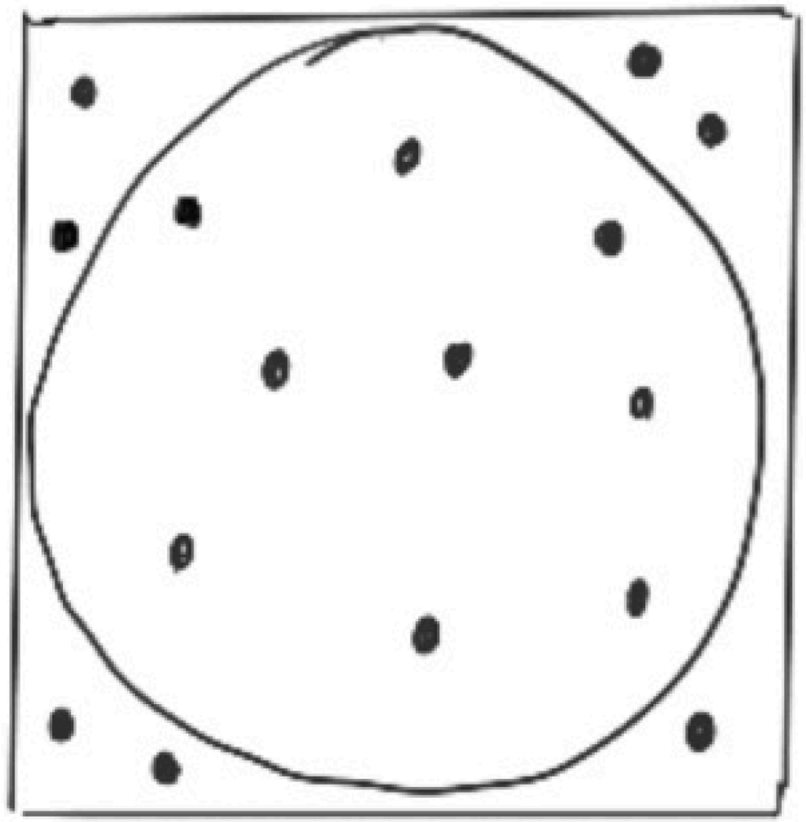
Vincent D. Warmerdam - GoDataDriven

koaning.io - fishnets88

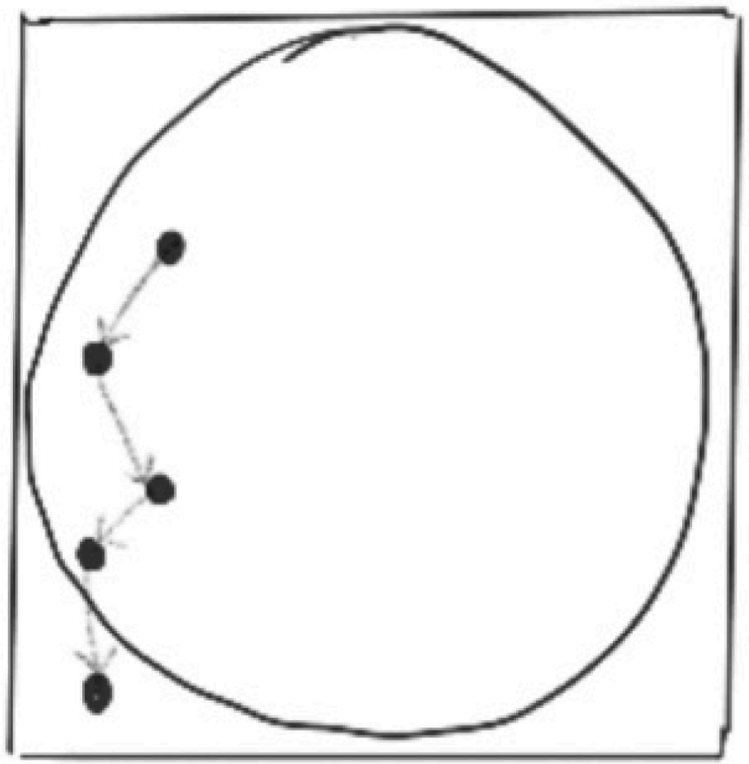
What I'll talk about

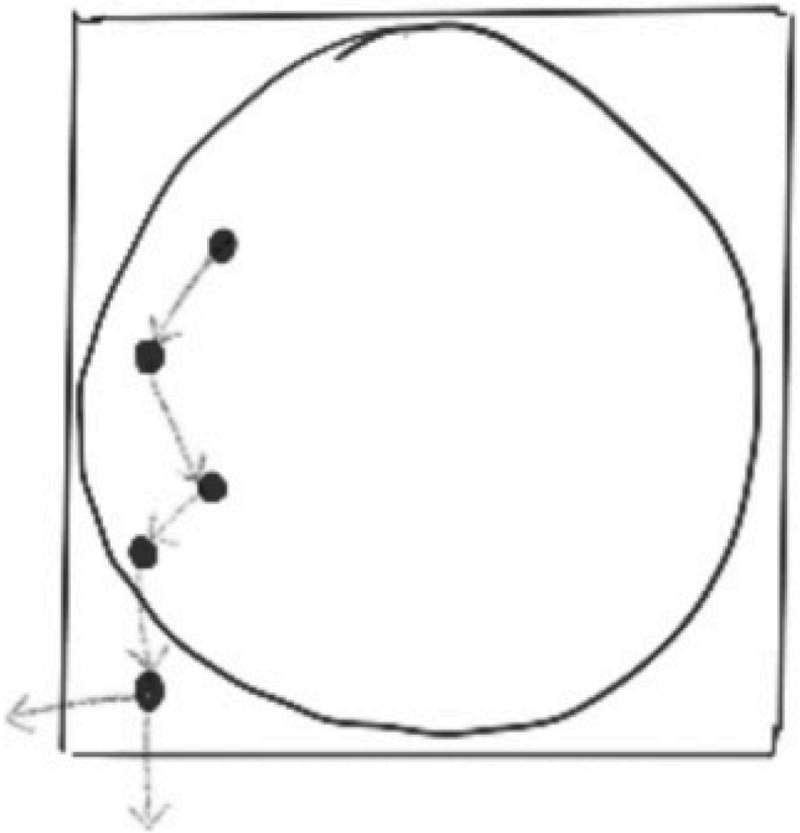
1. Briefly compare stateless sampling vs. statefull sampling
2. Briefly discuss the general algorithm of mcmc
3. Briefly discuss the idea of sampling for inference
4. Discuss the switchpoint problem
5. Demo how to solve this via Pymc3
6. Mention some things to look out for in the future

Idea of Sampling: π

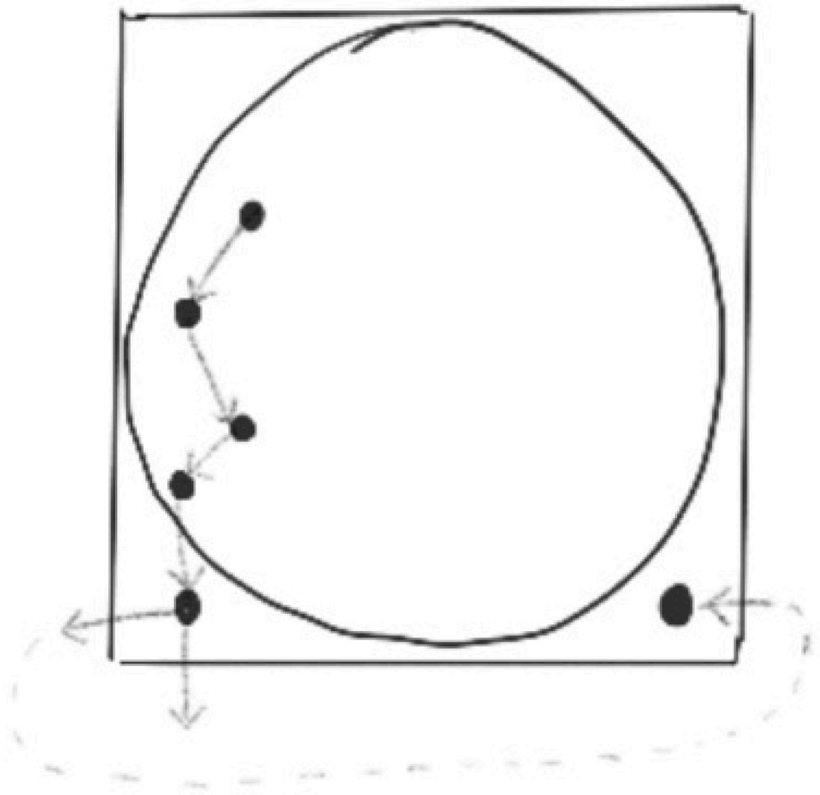


Idea of Sampling: π

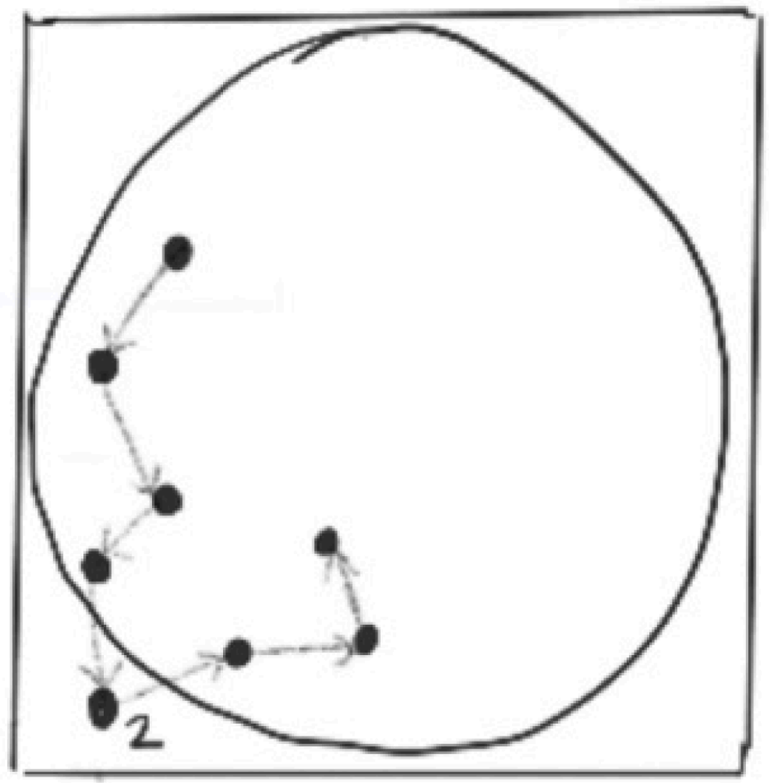




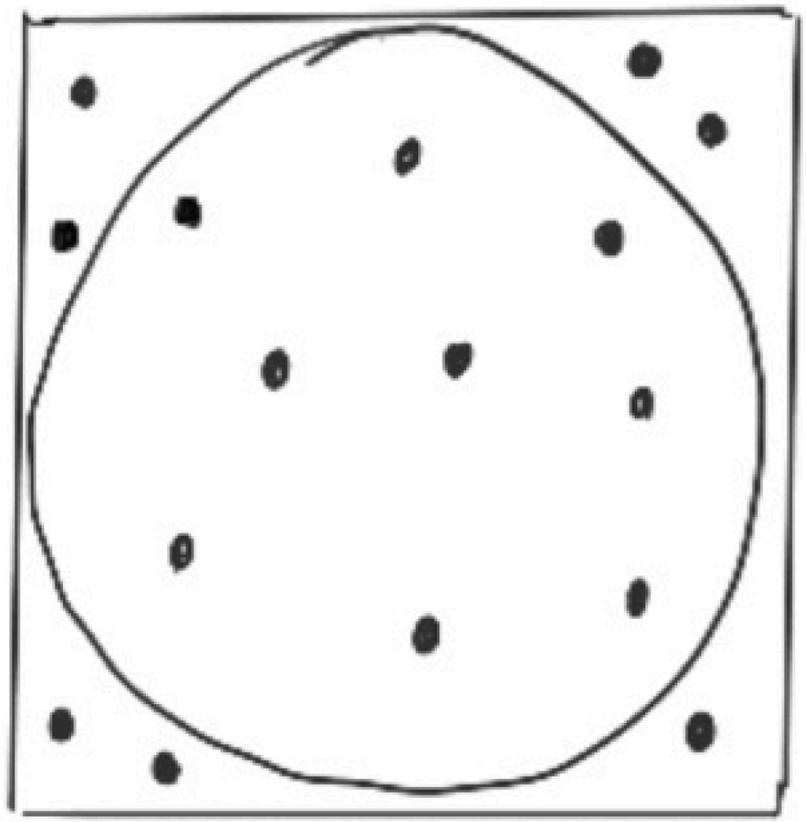
Idea of Sampling: π



Idea of Sampling: π



Idea of Sampling: π



Idea of Sampling: π

This approach doesn't just work for π, it actually works for a lot of general cases too. Sampling might be a very useful to discover stochastic patterns in a system.

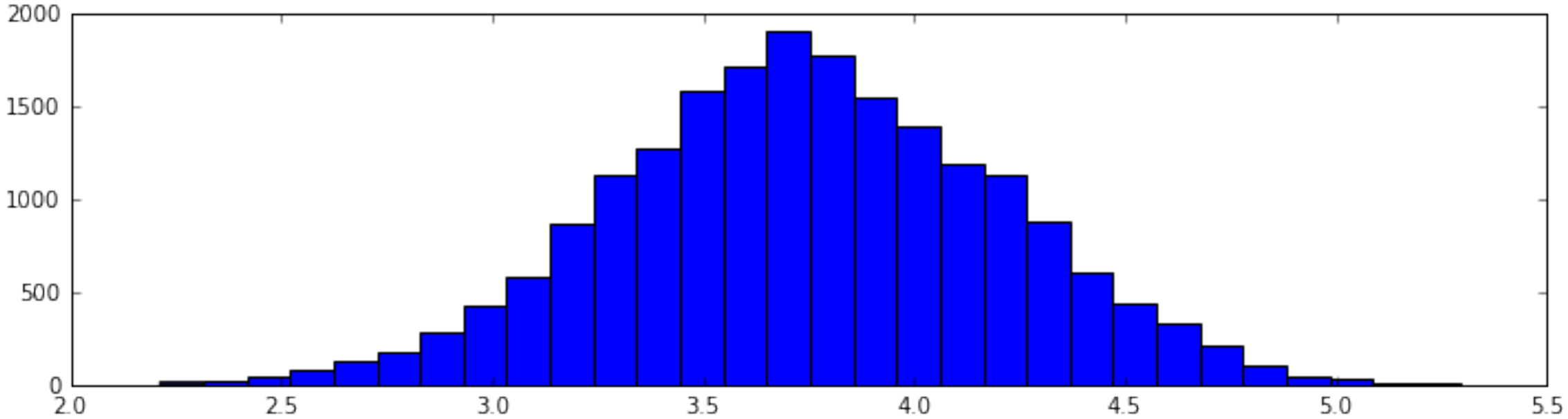
I'll demonstrate a simple numeric application.

MCMC 101 Python

```
nums = np.random.normal(3, 1, 10)
start_mu = 3
stepsize = 0.1
samples = []
for i in range(5000):
    new_mu = start_mu + np.random.normal(0, stepsize, 1)[0]
    old_loglik = np.prod([stats.norm.pdf(_, loc = start_mu) for _ in nums])
    new_loglik = np.prod([stats.norm.pdf(_, loc = new_mu) for _ in nums])
    if new_loglik > old_loglik:
        start_mu = new_mu
    else:
        if np.random.random() < new_loglik/old_loglik:
            start_mu = new_mu
    samples.append(start_mu)
```

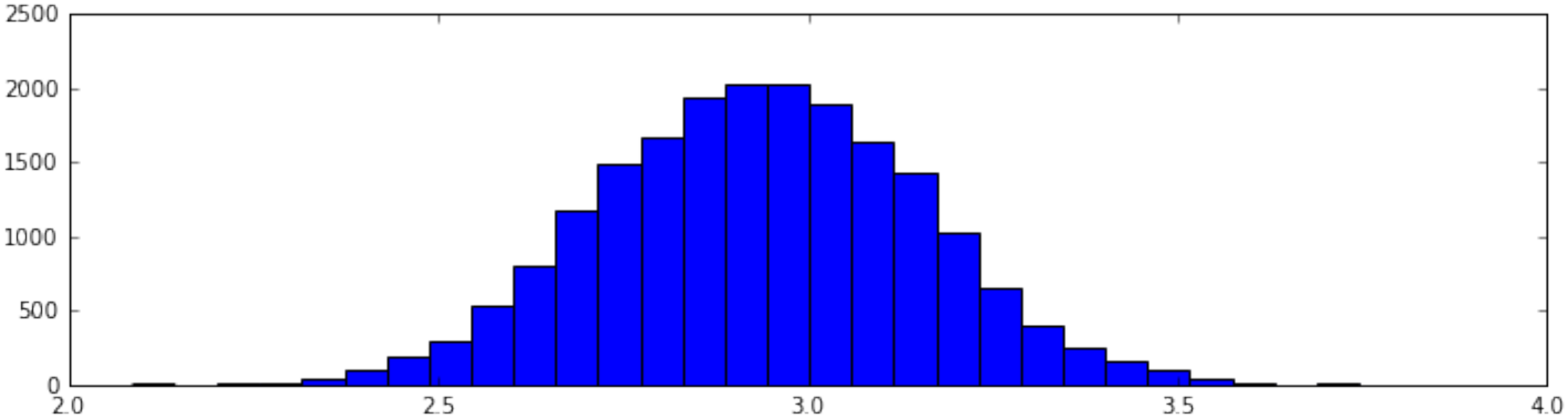
MCMC 101 Python

```
nums = np.random.normal(3, 1, 5)
```



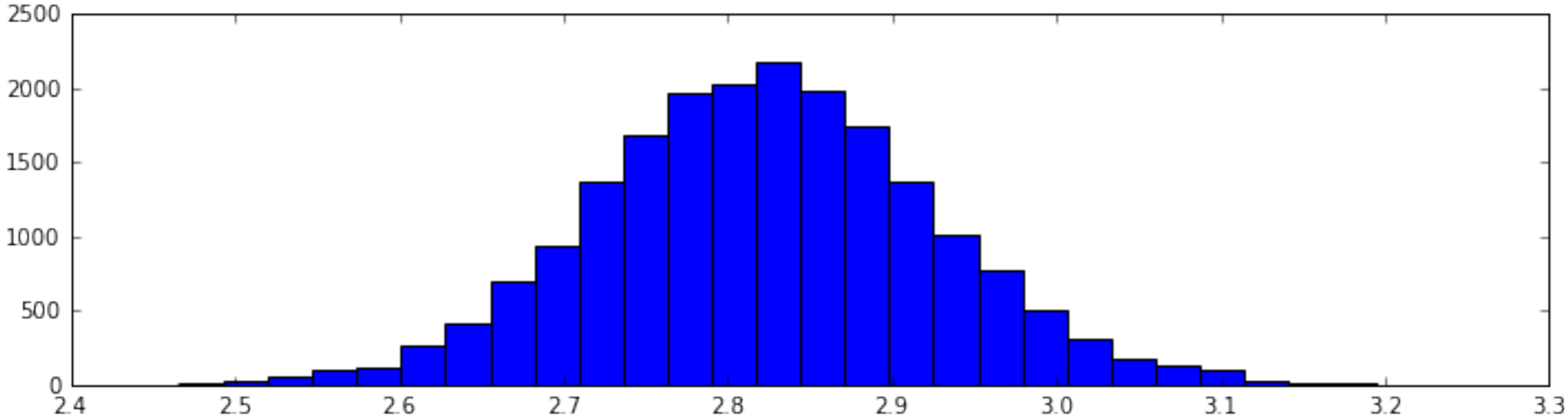
MCMC 101 Python

```
nums = np.random.normal(3, 1, 10)
```



MCMC 101 Python

```
nums = np.random.normal(3, 1, 100)
```

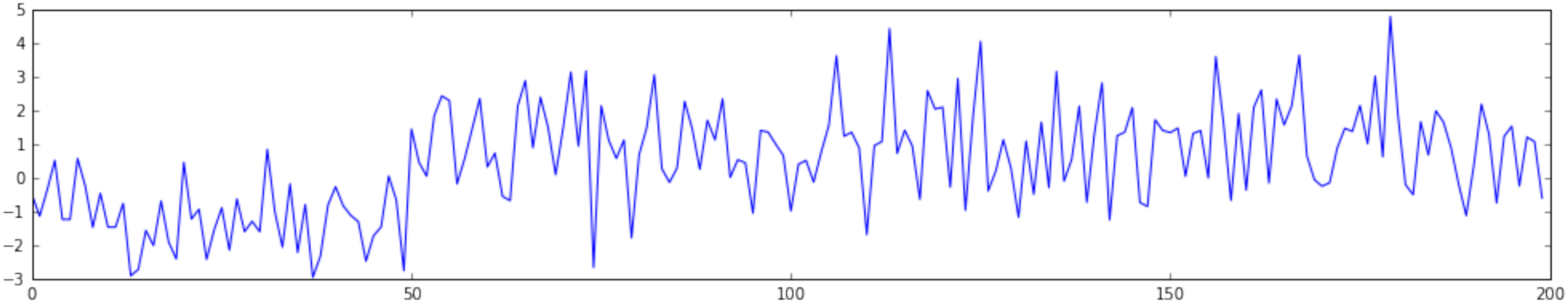


"Thats great Vincent, but what about applications?"

The Switchpoint Problem

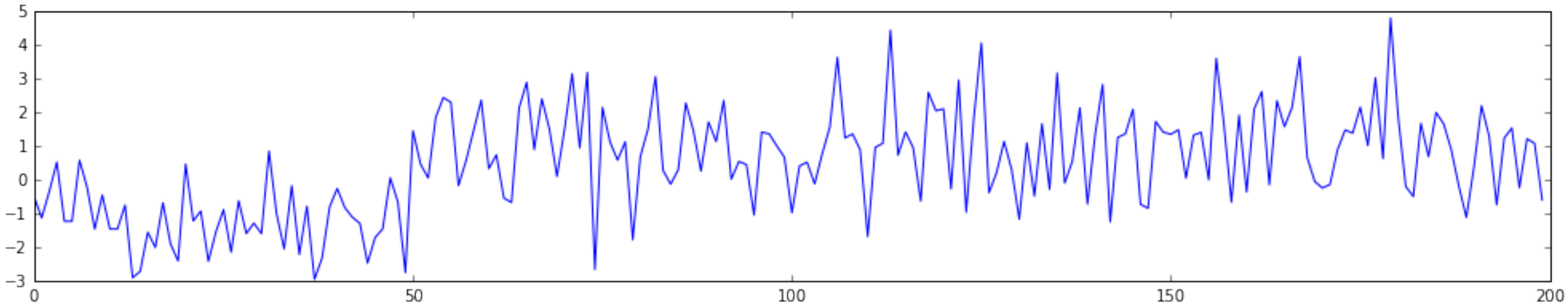
The Switchpoint Problem

There is a point in time when something change a system. We would like to find some algorithmic way of finding out when it happened and what the change was to the system.



The Switchpoint Problem

I'll first explain a simple example where the system has 5 parameters; τ, μ_1, σ_1, μ_2, σ_2.



The Switchpoint Problem

Given a dataset we're interested in finding;

$$\mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2 | D)$$

The Switchpoint Problem

Let's apply bayes rule.

$$\begin{aligned}\mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2 | D) &\propto \mathbb{P}(D | s, \mu_1, \mu_2, \sigma_1, \sigma_2) \mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2) \\ &= \mathbf{\Pi}_i \mathbb{P}(y_i | s, \mu_1, \mu_2, \sigma_1, \sigma_2) \mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2)\end{aligned}$$

The Switchpoint Problem

Let's apply bayes rule.

$$\mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2 | D) \propto \mathbb{P}(D | s, \mu_1, \mu_2, \sigma_1, \sigma_2) \mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2)$$
$$= \mathbf{\Pi}_i \mathbb{P}(y_i | s, \mu_1, \mu_2, \sigma_1, \sigma_2) \mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2)$$

I know some information about $\mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2)$.

For example; $\sigma_i > 0$. Priors are useful!

The Switchpoint Problem

The other part is the part that we model.

$$\mathbb{P}(y_i|s,\mu_1,\mu_2,\sigma_1,\sigma_2) = \begin{cases} y_i \sim N(\mu_1,\sigma_1) & \text{if } i \leq s \\ y_i \sim N(\mu_2,\sigma_2) & \text{if } i > s \end{cases}$$

The Switchpoint Problem

So that means we have the building blocks for;

$$\mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2 | D) \propto \Pi_i \mathbb{P}(y_i | s, \mu_1, \mu_2, \sigma_1, \sigma_2) \mathbb{P}(s, \mu_1, \mu_2, \sigma_1, \sigma_2)$$

This begs the question, why would we prefer to use MCMC here?

The Switchpoint Problem

I can come up with three reasons;

- Sampling is a very general framework. Can be parallized and take any shape.
- The search space in general isn't convex, so gradient methods lose their guarantee.
- By sampling, we get an impression of the distribution over all parameters; not just a mere MLE.

The Switchpoint Problem

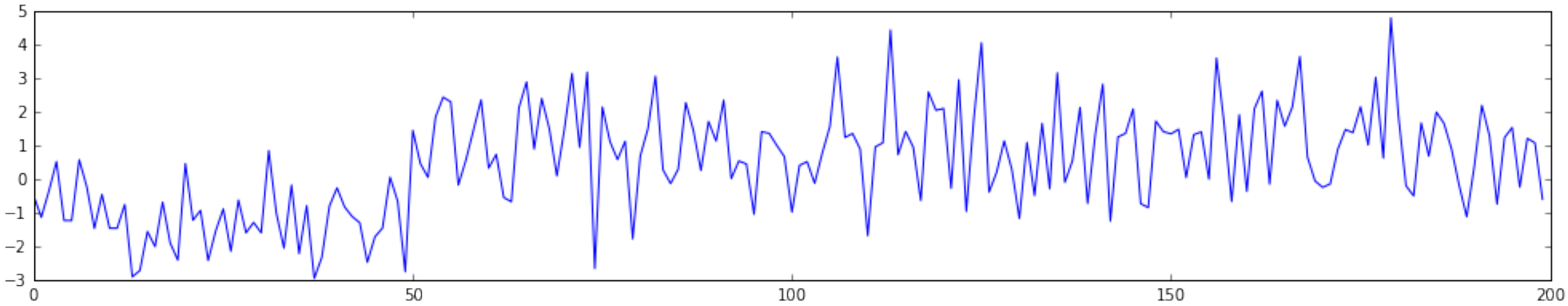
```
import pymc3 as pm
basic_model = pm.Model()
with basic_model:
    mu1 = pm.Normal('mu1', mu=0, sd=2)
    mu2 = pm.Normal('mu2', mu=0, sd=2)
    sigma1 = pm.HalfNormal('sigma1', sd=2)
    sigma2 = pm.HalfNormal('sigma2', sd=2)
    switchpoint = pm.DiscreteUniform('switchpoint', 0, time.max())

    tau_mu = pm.switch(time >= switchpoint, mu2, mu1)
    tau_sigma = pm.switch(time >= switchpoint, sigma2, sigma1)

    y = pm.Normal('y1', mu=tau_mu, sd=tau_sigma, observed=x)
    trace = pm.sample(10000)
```

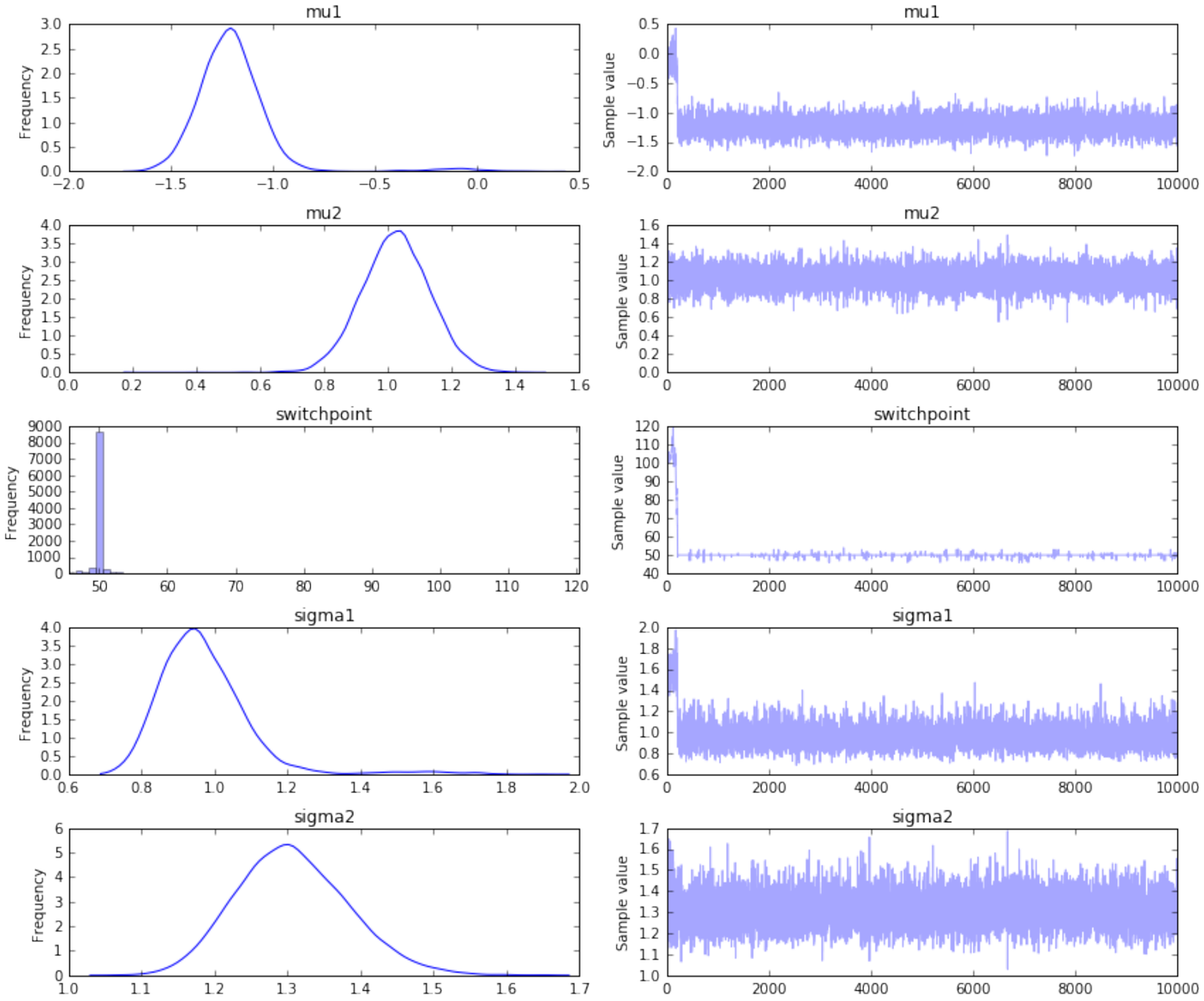
The Switchpoint Problem

This last mentioned `trace` variable can be used for plotting.

```
_ = pm.traceplot(trace)
```

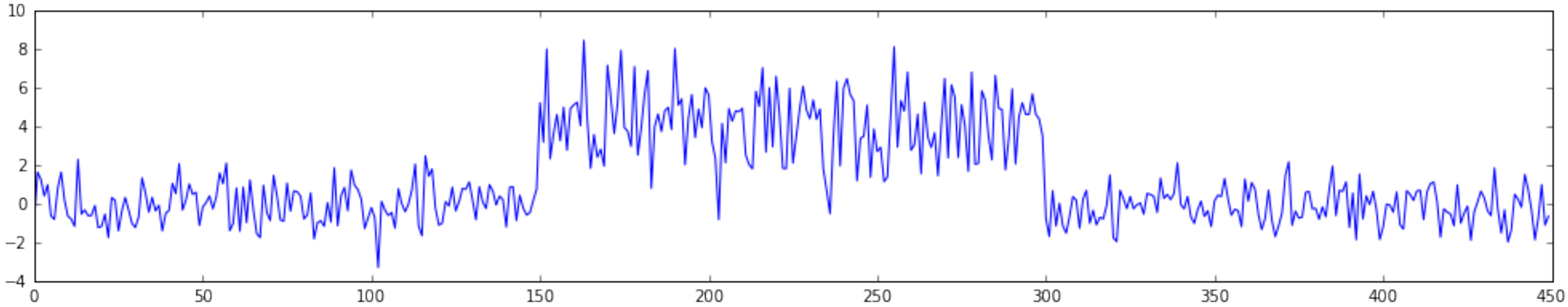
Let's look at the results for this dataset.




mu1
Frequency
mu1
Sample value
mu2
Frequency
mu2
Sample value
switchpoint
Frequency
switchpoint
Sample value
sigma1
Frequency
sigma1
Sample value
sigma2
Frequency
sigma2
Sample value

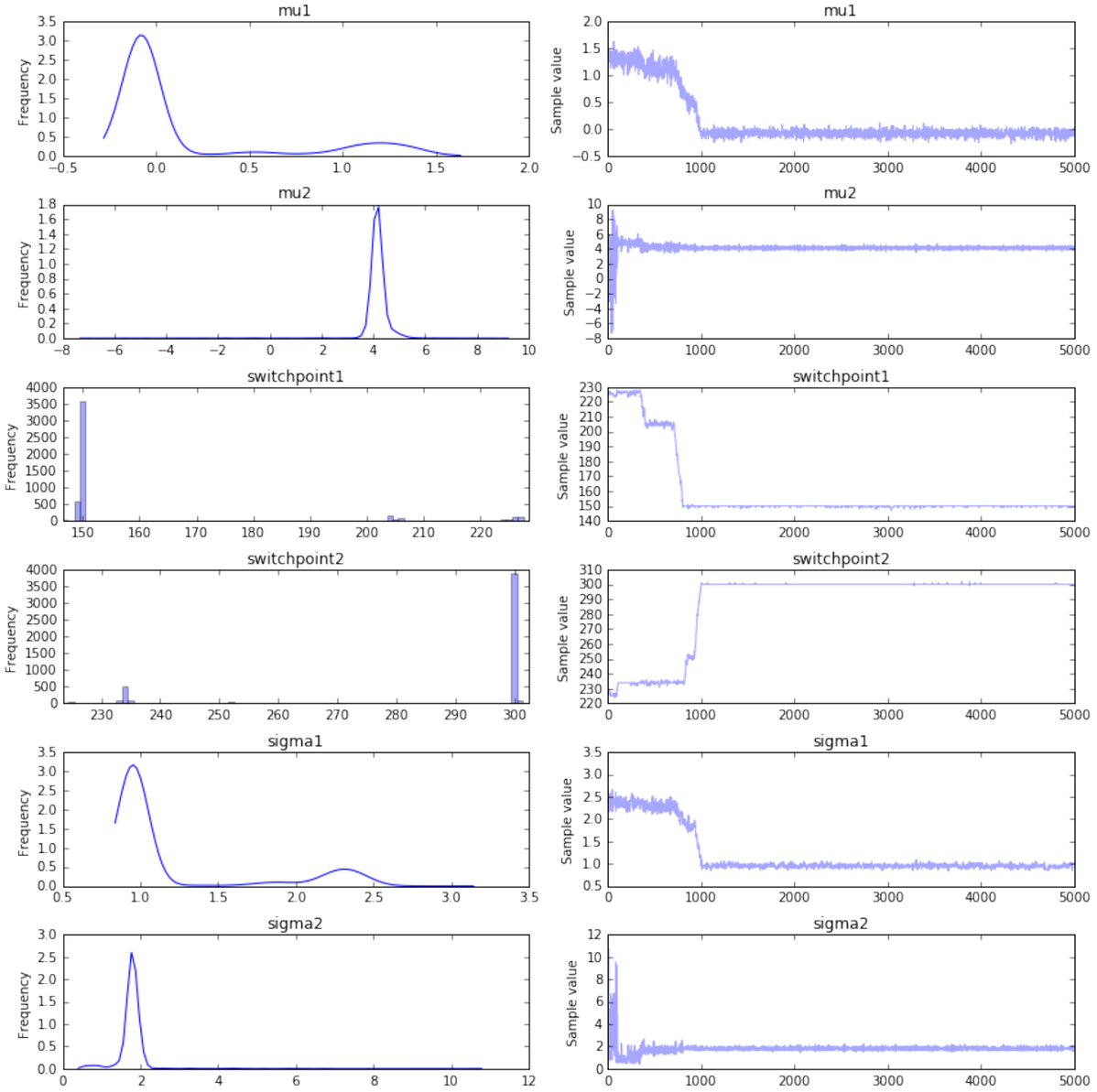
The Switchpoint Problem

Let's now change the problem.



The Switchpoint Problem

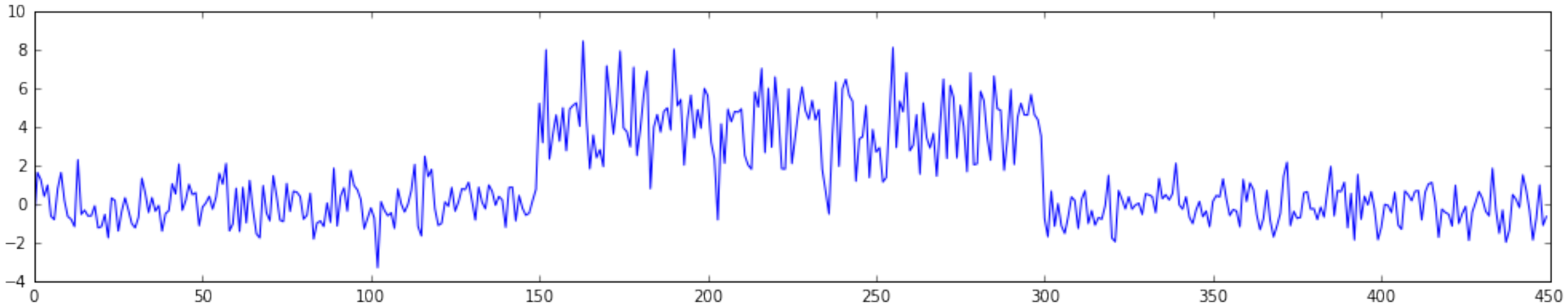
```
adv_model = pm.Model()
with adv_model:
    mu1 = pm.Normal('mu1', mu=0, sd=4)
    mu2 = pm.Normal('mu2', mu=0, sd=4)
    sigma1 = pm.HalfNormal('sigma1', sd=4)
    sigma2 = pm.HalfNormal('sigma2', sd=4)
    switchpoint1 = pm.DiscreteUniform('switchpoint1', 0, time.max() - 1)
    switchpoint2 = pm.DiscreteUniform('switchpoint2', switchpoint1, time.max())

    tau_mu1 = pm.switch(time >= switchpoint1, mu2, mu1)
    tau_mu2 = pm.switch(time >= switchpoint2, mu1, tau_mu1)
    tau_sigma1 = pm.switch(time >= switchpoint1, sigma2, sigma1)
    tau_sigma2 = pm.switch(time >= switchpoint2, sigma1, tau_sigma1)

    y = pm.Normal('y1', mu=tau_mu2, sd=tau_sigma2, observed=x)
    adv_trace = pm.sample(5000)
```

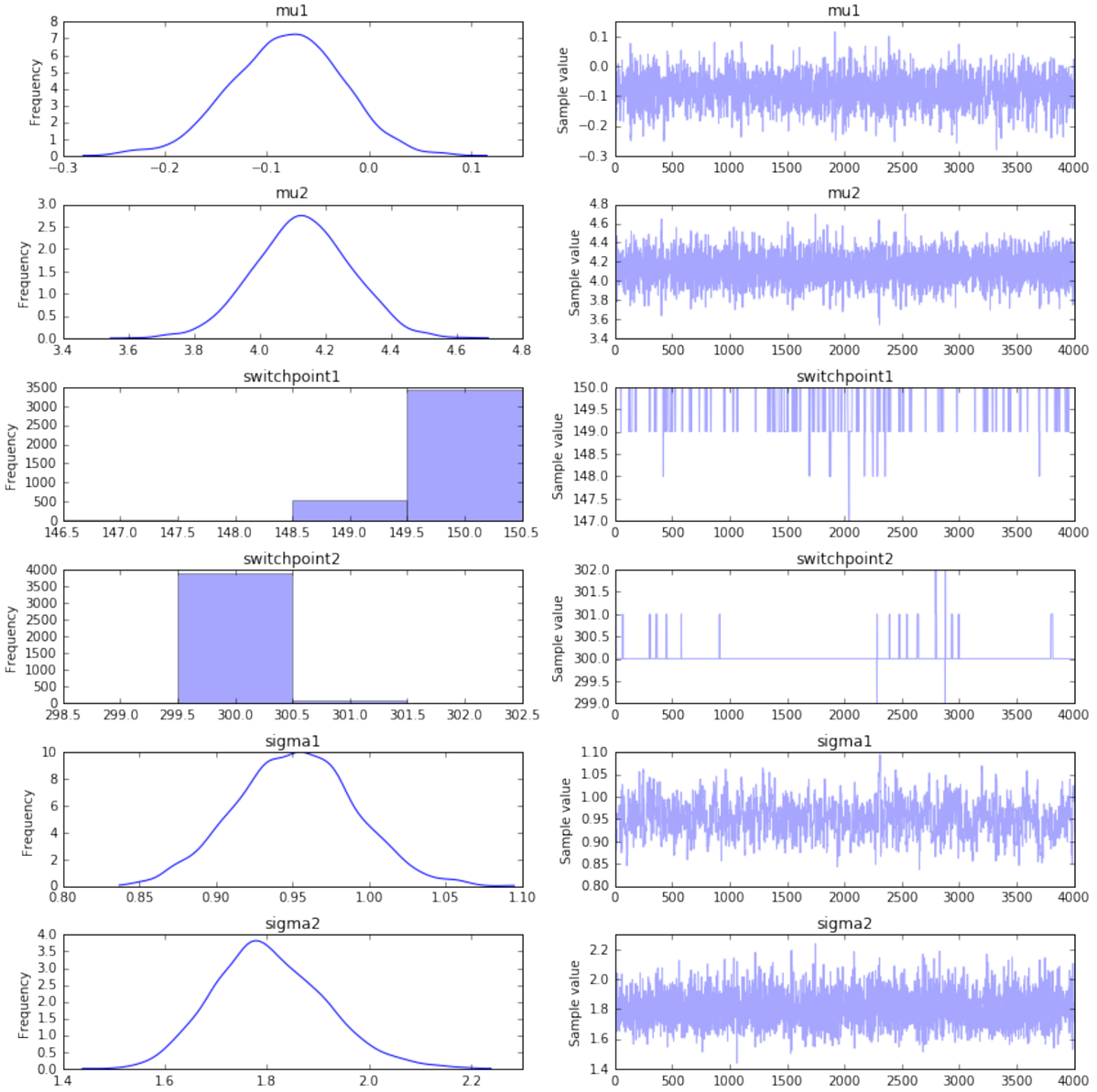

mu1
Frequency
mu1
Sample value
mu2
Frequency
mu2
Sample value
switchpoint1
Frequency
switchpoint1
Sample value
switchpoint2
Frequency
switchpoint2
Sample value
sigma1
Frequency
sigma1
Sample value
sigma2
Frequency
sigma2
Sample value

The Switchpoint Problem

Combat the start of the trace by ignoring it.

```
_ = pm.traceplot(trace[1000:])
```




mu1
Frequency
mu1
Sample value
mu2
Frequency
mu2
Sample value
switchpoint1
Frequency
switchpoint1
Sample value
switchpoint2
Frequency
switchpoint2
Sample value
sigma1
Frequency
sigma1
Sample value
sigma2
Frequency
sigma2
Sample value

PyMC3

You can also retreive statistics.

$$\arg\max_{\lambda} \mathbb{P}(D|\lambda)\mathbb{P}(\lambda)$$

```
> pm.find_MAP(model=adv_model)
{'mu1': array(1.3253008730440974),
 'mu2': array(0.0),
 'sigma1_log_': array(0.863997531498574 1),
 'sigma2_log_': array(1.3862943591402144),
 'switchpoint1': array(224),
 'switchpoint2': array(224)}
```

The future

There's people doing this type of thing for neural networks.

$$\mathbb{P}(w|D) \propto \mathbb{P}(D|w)\mathbb{P}(w)$$

Sampling this is crazy, so people sometimes resort to variational inference. It's an interesting thought and you can already start playing with it. Here is a blogpost from pymc3 contributor explaining how to use PyMC3 to train a neural network for the mnist dataset via lasagna.

The future

Outside of Pymc3 it seems like edward is another contender for variational inference and probibalistic modelling.

If you're interested in just maximum likelihood; maybe check out `pomegrenate`. If allows you to make hidden markov models as well as general PGMs (though there currently is no support for continous variables for general PGM).

If you're interested in other ways to do sampling; maybe check out `STAN` or `emcee`. Maybe consider GP's!