Numerical Computing with Python: Markov Chain Monte Carlo

Ramses van Zon

SciNet HPC Consortium

December 4, 2018

Today's leture

We saw Monte Carlo methods already as a means to compute high-dimensional integrals.

Today we will look at more general Monte Carlo methods to do parameter estimation.

Thanks to Mike Nolta for the orginal slides.

Overview

Markov chain Monte Carlo (MCMC) is a method to generate samples from general probability distributions. It's considered one of the most important algorithms of the 20th century.

- Why is this important? (aka Bayesian inference)
- Monte-Carlo methods
- Markov chains
- MCMC sampling
- PyMC3

Uncertainty

Consider Newton's constant, G. The current value is

$$G = 6.67408 \pm 0.00031 \times 10^{-11} \mathrm{m}^3/\mathrm{kg}/\mathrm{s}^2$$

The uncertainty in G isn't a statement about the physical world.

It's a statement about human ignorance.

Bayesian Inference

Bayesian inference is a process for updating our beliefs when we acquire new information, using Bayes theorem:

$$P(X|d) = P(d|X)P(X)/P(d)$$

Terminology:

- d is the *data*, X are the *model parameters*
- $P(X|d)$ is the *posterior*, our beliefs after d
- $P(X)$ is the *prior*, our beliefs prior to d
- $P(d|X)$ is the *probability* of d given X, called the *likelihood*.
- $P(d)$ is the *model evidence* or *marginal likelihood*: $P(d) = \int P(d|X)P(X)dX$

Bayesian Inference Example

Let's say our data is some linear function of measured inputs plus noise:

$$y = ax + b + \epsilon$$

The noise is Gaussian with variance σ^2, so the likelihood function is:

$$Normal(ax + b, \sigma)$$

Our priors our $P(a)$, $P(b)$, and $P(\sigma)$. If any of these are known, then the prior is just a Dirac delta function.

Thus

$$P(a, b, \sigma|y) \propto Normal(ax + b, \sigma)P(a)P(b)P(\sigma)$$

Monte Carlo methods

Given a posterior $\pi(X)$, we typically want to compute expectation values:

$$E[f(X)] = \int f(X)\pi(X)dX$$

For example, the mean $E[X]$ or variance $E[(X-\mu)^2]$.

Monte Carlo methods

Given a posterior $\pi(X)$, we typically want to compute expectation values:

$$E[f(X)] = \int f(X)\pi(X)dX$$

For example, the mean $E[X]$ or variance $E[(X-\mu)^2]$.

However, $\pi(X)$ is usually complex and high-dimensional, so computing the integral is difficult.

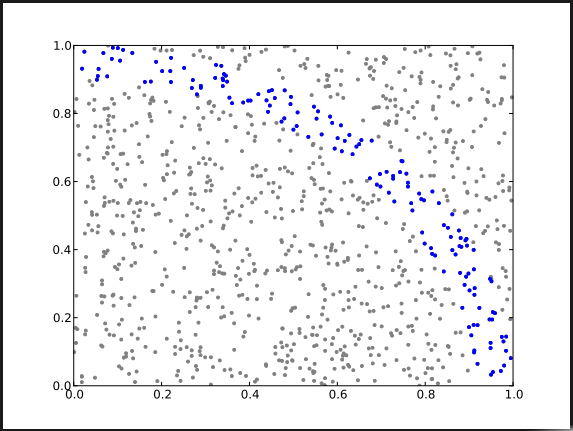
We can approximate the integral using the "law of large numbers":

$$E[f(X)] \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i), \qquad x_i \sim \pi(X)$$

The notation "$a \sim B$" means a is a random sample from probability distribution B.

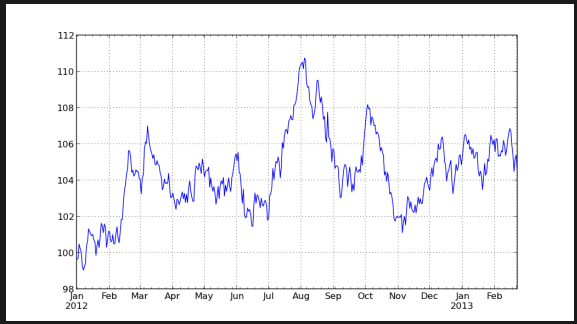
Random Sampling

- So how do we draw samples from an arbitrary probability distribution?
- One option is rejection sampling: draw samples from a known, larger distribution, and then reject samples based on the desired distribution.
- However, it's slow, particularly in high dimensions.
- Can we use the fact that high-probability regions are typically concentrated; that is, likely points are near other likely points?



Markov Chain

- Random samples are typically IID: independent and identically distributed. New samples don't depend on previous samples.
- A *Markov chain* is a sequence of random numbers $X_0, X_1, ..., X_n$ where the probability of X_{i+1} can depend on X_i.
- Distribution is $P(X_{i+1}|X_i)$ instead of $P(X_i)$.
- A classic example of a Markov chain is a random walk: $X_{i+1} = X_i + \epsilon$



Markov Chain Monte Carlo (MCMC)

MCMC is the construction of a Markov chain such that the density of points converges to a desired distribution.

There are many kinds of MCMC sampling — this is the original and simplest (Metropolis-Hastings):

To draw samples from a distribution $\pi(\boldsymbol{X})$, first choose a starting position $\boldsymbol{X_0}$. Then:

- Choose a new point $\boldsymbol{X_{i+1}}$ from the symmetric proposal distribution $\boldsymbol{q}(\boldsymbol{X_{i+1}}|\boldsymbol{X_i})$ (typically Gaussian).
- Accept the new point with probability

$$\min\left(1, \frac{\pi(\boldsymbol{X_{i+1}})}{\pi(\boldsymbol{X_i})}\right)$$

- Repeat.

PyMC3

PyMC3

- PyMC3 is a python package for Bayesian modeling via MCMC, also called probabilistic programming.
- You construct likelihood + priors by assembling known distributions.
- Implements a variety of sampling techniques beyond Metropolis (e.g., NUTS, Slice, Hamiltonian).
- http://docs.pymc.io/

PyMC3 on Niagara

To use on Niagara, create a virtual environment with pymc3:

```
$ ml anaconda3/5.2.0
$ conda create -n pymc3env
$ source activate pymc3env
$ conda install pymc3
```

This full of commands is only needed once. Next time you log in (or run a job), you can say

```
$ ml anaconda3 && source activate pymc3env
```

Put the following in `~/.theanorc` to speed up compilation by avoiding the filesystem:

```
[global]
base_compiledir=/dev/shm/<USER>/theano
```

Example 1: fitting a simple model

Let's generate some random gaussian samples, and then estimate the mean & std deviation.

```
>>> import numpy as np
>>> np.random.seed(12345) # for reproducibility
>>> n = 500
>>> data = 0.55 + 2.3*np.random.randn(n) # dataset
```

Example 1: fitting a simple model

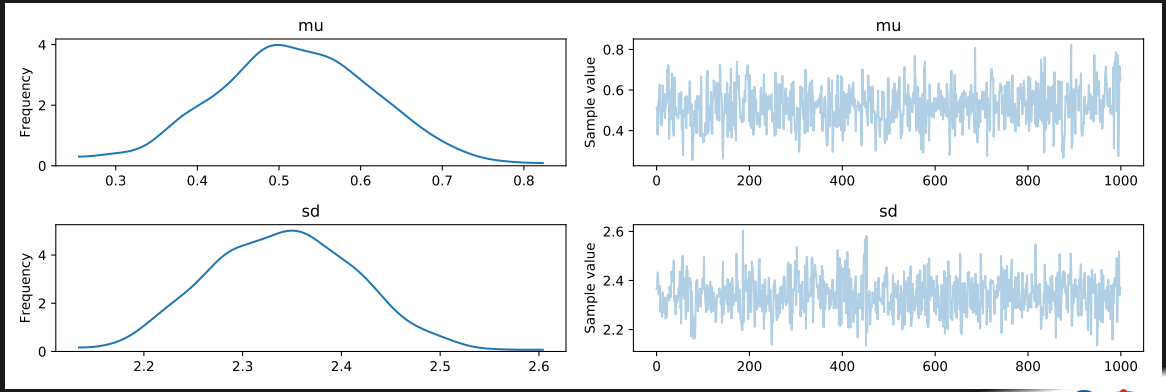
Let's generate some random gaussian samples, and then estimate the mean & std deviation.

```
>>> import numpy as np
>>> np.random.seed(12345) # for reproducibility
>>> n = 500
>>> data = 0.55 + 2.3*np.random.randn(n) # dataset
```

```
>>> import pymc3 as pm
>>> model = pm.Model()
>>> with model:
...     mu = pm.Uniform('mu', lower=-3, upper=3) # prior
...     sd = pm.Uniform('sd', lower=1, upper=4)  # prior
...     obs = pm.Normal('obs', mu=mu, sd=sd, observed=data) # likelihood
...     trace = pm.sample(1000, chains=1)
...
>>>
```

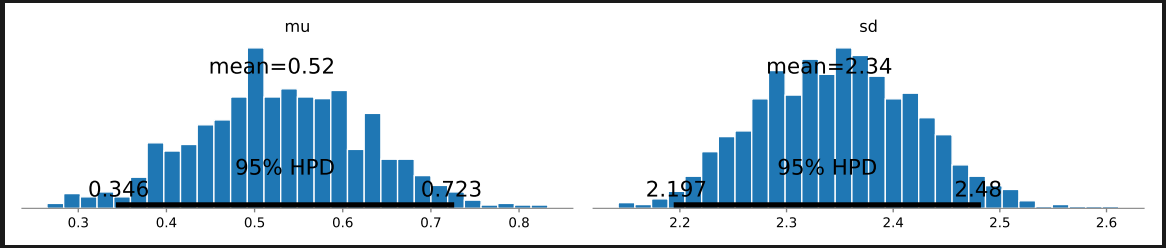
Example 1: traceplot

```
>>>> pm.traceplot(trace)
```



Example 1: plot_posterior

```
>>> pm.plot_posterior(trace)
```



Example 1: summary

```
>>> pm.summary(trace)
```

```
mu:
  Mean             SD               MC Error         95% HPD interval
  ------------------------------------------------------------------
  0.520            0.097            0.004            [0.346, 0.723]
  Posterior quantiles:
  2.5              25               50               75               97.5
  |--------------|================|================|--------------|
  0.326            0.455            0.519            0.585            0.705
sd:
  Mean             SD               MC Error         95% HPD interval
  ------------------------------------------------------------------
  2.340            0.075            0.003            [2.197, 2.480]
  Posterior quantiles:
  2.5              25               50               75               97.5
  |--------------|================|================|--------------|
  2.202            2.287            2.342            2.392            2.490
```

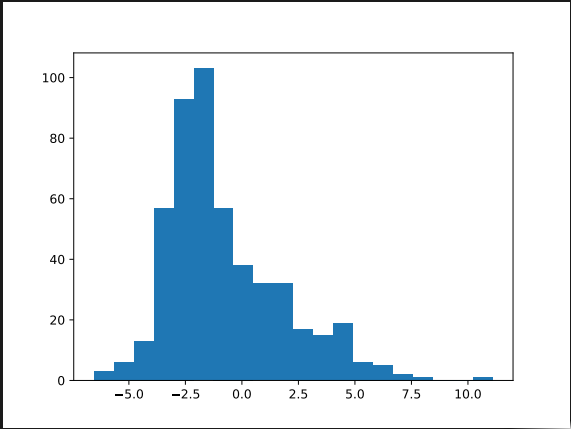
Example 1: sanity check

parameter	expected	expected	got
mu	$\mu \pm \sigma/\sqrt{n}$	0.550 ± 0.103	0.520 ± 0.097
sd	$\sigma \pm \sigma/\sqrt{2n-2}$	2.300 ± 0.073	2.340 ± 0.075

Example 2: mixture model

Samples are drawn randomly from one of two different gaussian distributions.

```
>>> import numpy as np
>>> np.random.seed(12345) # for reproducibility
>>> n = 500
>>> # simulate data from mixture distribution
>>> v = np.random.randint(0, 2, n)
>>> means = np.array([-2.0, 1.0])
>>> stddevs = np.array([1.0, 3.0])
>>> data=means[v]+stddevs[v]*np.random.randn(n)
```

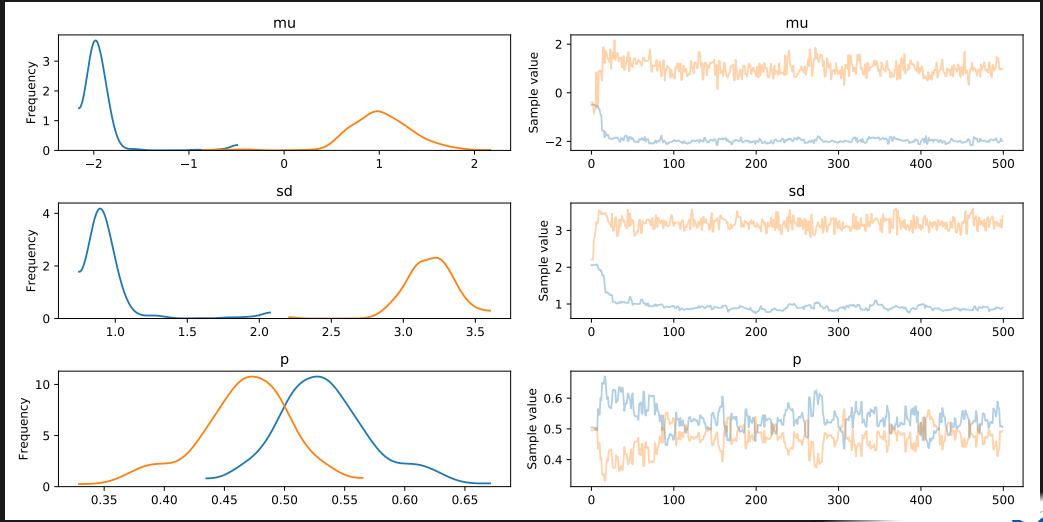


Example 2: pymc3 model

```
with pm.Model() as model:

 p = pm.Dirichlet('p', np.ones(2), shape=2)
 id = pm.Categorical('id', p, shape=n)

 mu = pm.Uniform('mu',lower=-5,upper=4,shape=2)
 sd = pm.Uniform('sd',lower=.1,upper=4,shape=2)

 obs = pm.Normal('obs',
                 mu=mu[id],
                 sd=sd[id],
                 observed=data)

 trace = pm.sample(500, chains=1)
```

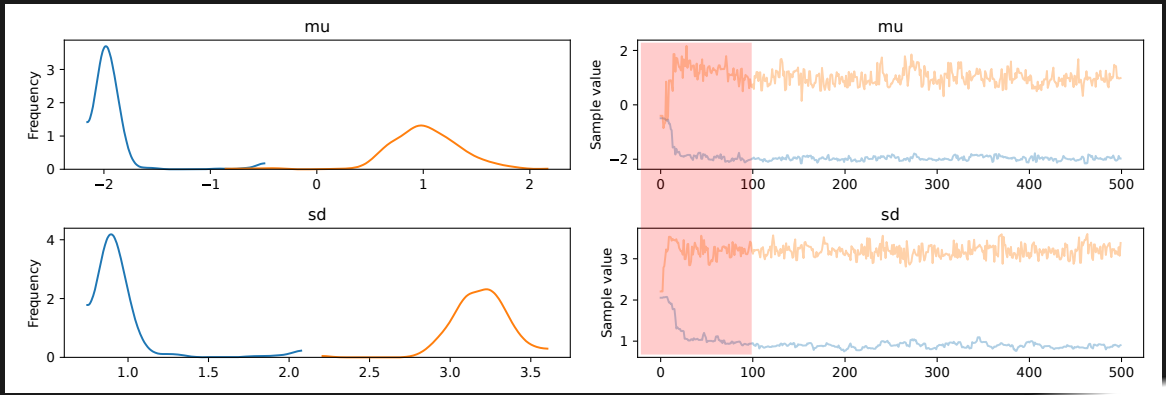
- The `Categorical` distribution chooses $id \in [0, 1]$ with probabilities $p = [p_0, p_1]$ for all n data points.
- Since we don't know what $p_{0,1}$ are, we draw them from a `Dirichlet` distribution.
- Then we use `id` to index $\mu = [\mu_0, \mu_1]$ and $\sigma = [\sigma_0, \sigma_1]$

Example 2: traces



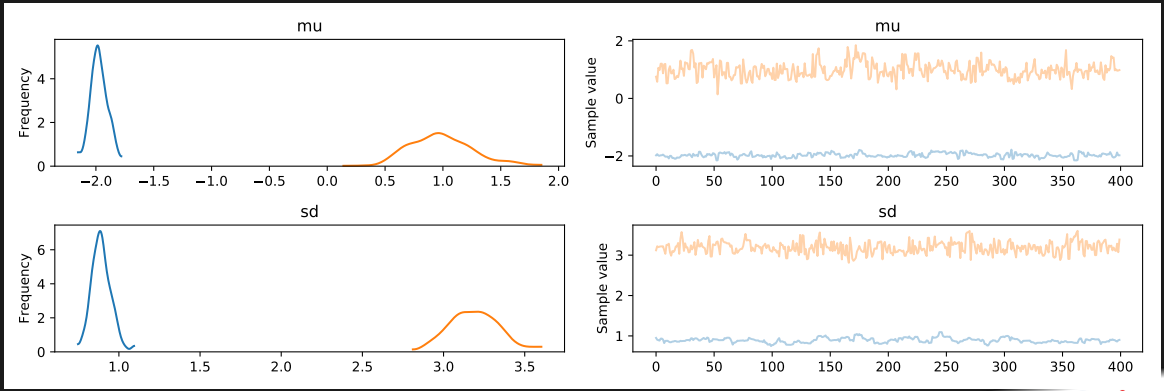
Example 2: “burn in”

Note that it takes a while for the chains to “settle down”, because they haven’t found the high-probability regions yet. This is called *“burn in”*.



Example 2: “burn in” removed

```
>>> pm.traceplot(trace[100:], ['mu','sd'])
```

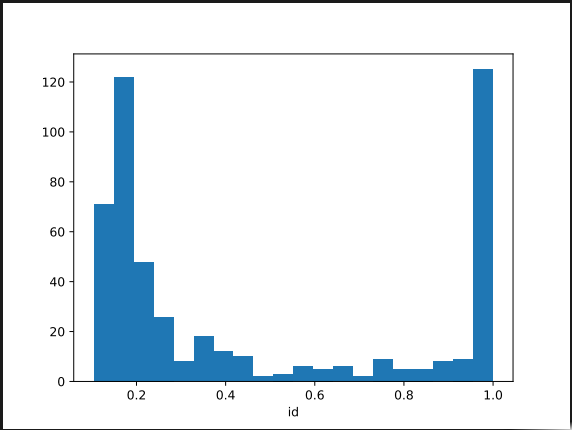


Example 2: average ID

- At right is the histogram of average cluster IDs:

```
>>> plt.hist( trace.id.mean(axis=0), bins=20)
```

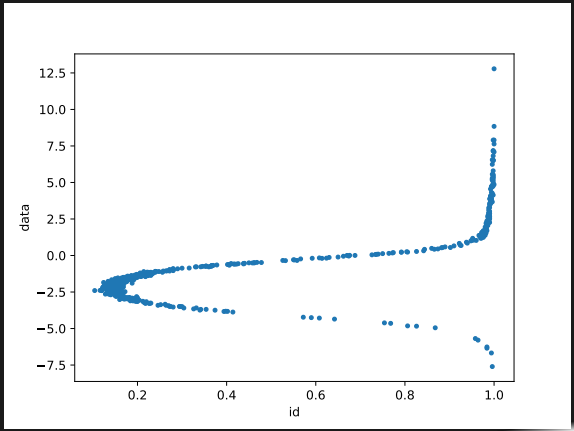
- As expected, a bimodal distribution.
- But since the two components overlap significantly, a lot of points don't have precise IDs.



Example 2: id vs data

```
>>> plt.plot( trace.id.mean(axis=0), data, '.')
```

- At the extremes it's clear which cluster the points belong to.



Example 2: caveats

- The model is potentially unstable: nothing to prevent cluster ID's (0 & 1) from being swapped.
  - One option would be to add an additional constraint to break symmetry, e.g., require mean(0) $<$ mean(1)
- Better to use PyMC3's built-in `NormalMixture` distribution.

Example 3: linear regression

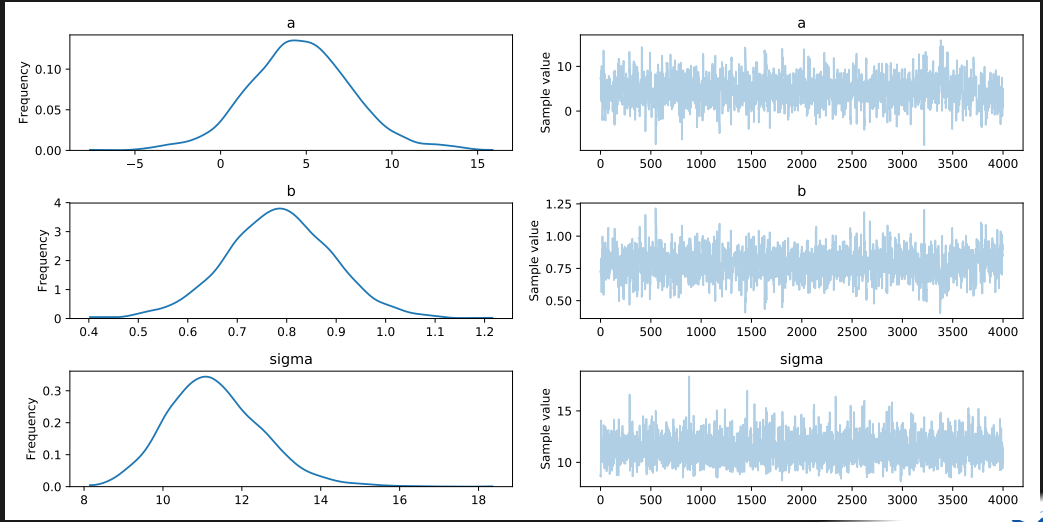
```
>>> n = 50
>>> x = np.arange(n)
>>> y = x + 10*np.random.randn(n)
```

Example 3: linear regression

```
>>> n = 50
>>> x = np.arange(n)
>>> y = x + 10*np.random.randn(n)
```

```
>>> model = pm.Model()
>>> with model:
...     a = pm.Normal('a', mu=0, sd=10)
...     b = pm.Normal('b', mu=0, sd=10)
...     sigma = pm.Uniform('sigma', lower=1, upper=100)
>>> pm.Normal('y',
...           mu=a + b*x,
...           sd=sigma,
...           observed=y)
>>> trace = pm.sample(500, chains=1)
```

Example 3: traces

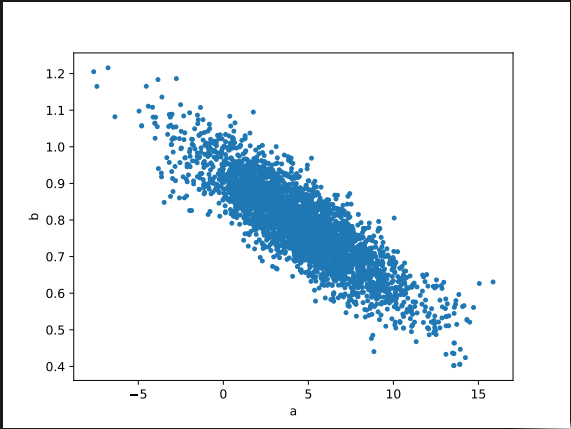


Example 3: correlation

- The correct answer $(a = 0, b = 1)$ doesn't seem particularly favored.

Example 3: correlation

- The correct answer $(a = 0, b = 1)$ doesn't seem particularly favored.
- One issue is that a and b are highly correlated. At right is plot of a vs b for all the samples in the chain.
- This illustrates one of the advantages of MCMC: it lets you explore the full correlation structure of your parameters.



Conclusion

- MCMC is a powerful technique, but it's not foolproof.
- How do i know if my chain has adequately sampled the distribution (aka converged)?
  - Run multiple chains with different starting points, and compare the inter-chain and intra-chain variances (Gelman-Rubin test).