

---

Repository of the code and slides:

<https://bit.ly/2JrHNLD>

# ALL THAT LIKELIHOOD

---

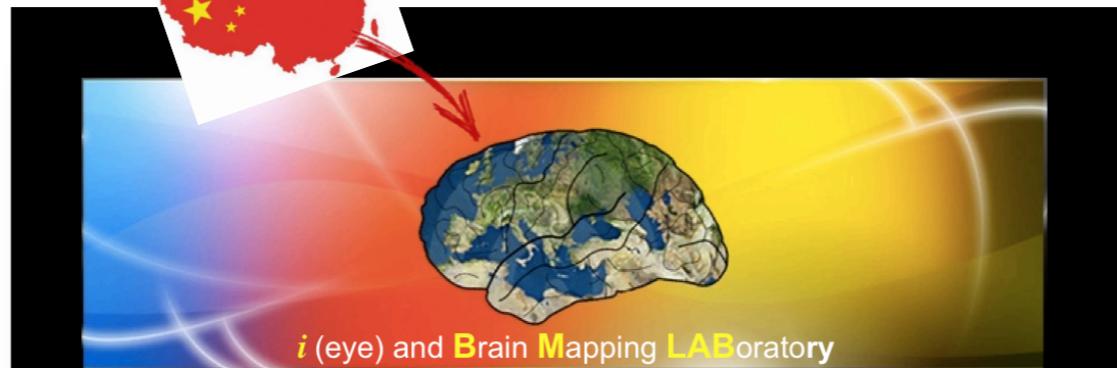
with PyMC3

Junpeng Lao

June 2018 @ Düsseldorf

Powered by  **PyMC3**

## About Me



University  
of Glasgow

**UNI**  
**FR**  
■

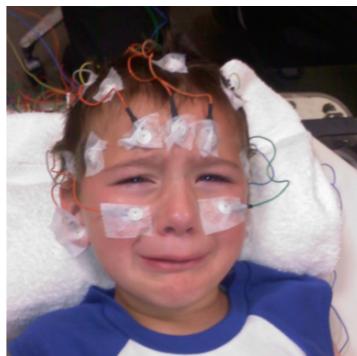
UNIVERSITÉ DE FRIBOURG  
UNIVERSITÄT FREIBURG

## About Me

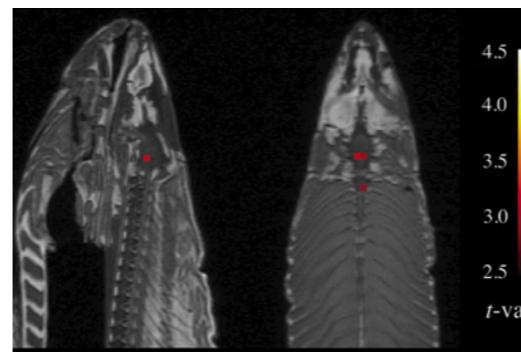
Eye Tracking



EEG



fMRI





all categories ▾

Latest

Top

Categories

+ New Topic

Topic

Category

Users

Replies

Views

Activity

Setting pymc3 random seed at once

Questions



1

14

2h

Using WAIC to compure Log Predictive Accuracy

Questions



3

6

2h

last visit

Concepts of Parameter Estimation and Predictions,  
and Out of Sample Predicted Probability for Logistic  
Regression

Questions



5

36

4h

ValueError: Bad initial energy: inf. The model might  
be misspecified error replicating Multilevel Regression  
and Poststratification with PyMC3 notebook

Questions



2

13

4h

Metropolis: ideal balance between slow-mixing and  
high rejection rate?

Questions



3

13

5h

The pymc3 way: Linear regression and inferring  
given posterior/trace

Questions



4

26

18h

Autocorrelation in a model

Questions



8

35

22h

<https://discourse.pymc.io/>

## Outline:

---

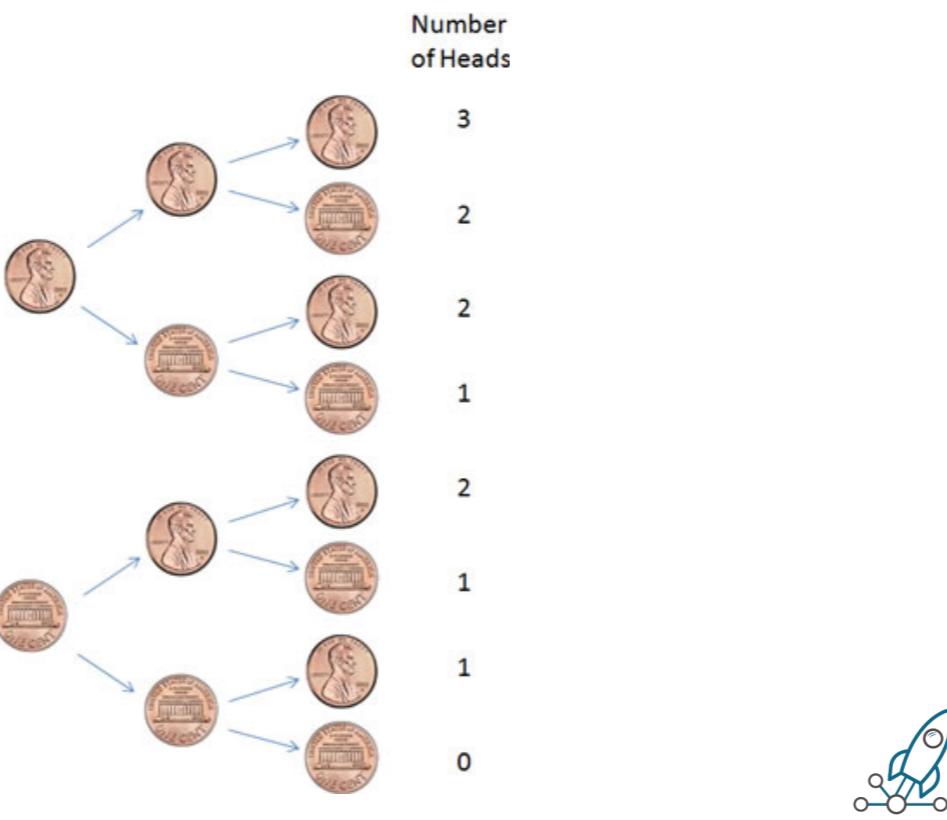
- Probability density/mass function and likelihood function
- Computing Likelihood in PyMC3

Repository of the code and slides:

<https://bit.ly/2JrHNLD>



# Random Variable



A random variable is a numerical measure of the outcome of a probability experiment whose value is determined by chance.

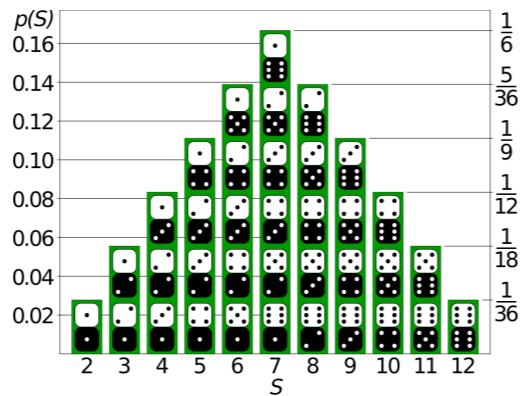
A random variable is defined as a function that maps the outcomes of unpredictable processes to numerical quantities (labels), typically real numbers.

image source:

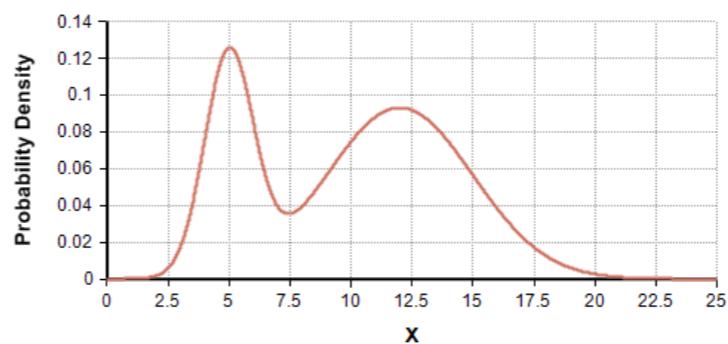
<https://faculty.elgin.edu/dkernler/statistics/ch06/6-1.html>

# Probability distribution

Discrete case:



Continuous case



[https://en.wikipedia.org/wiki/Probability\\_distribution](https://en.wikipedia.org/wiki/Probability_distribution)



A random variable has a probability distribution, which specifies the probability that its value falls in any given interval. It provides the probabilities of occurrence of different possible outcomes in an experiment.

In the discrete case, it is sufficient to specify a probability mass function assigning a probability to each possible outcome

The probability density function describes the infinitesimal probability of any given value, and the probability that the outcome lies in a given interval can be computed by integrating the probability density function over that interval.

# Probability theory

## Measure-Theoretical Foundations of Probability Theory

- Measure theory
  - $\sigma$ -algebra
- Measurable mapping
  - pushforward measure
- Integral
  - expectation



[https://betanalpha.github.io/assets/case\\_studies/probability\\_theory.html](https://betanalpha.github.io/assets/case_studies/probability_theory.html)

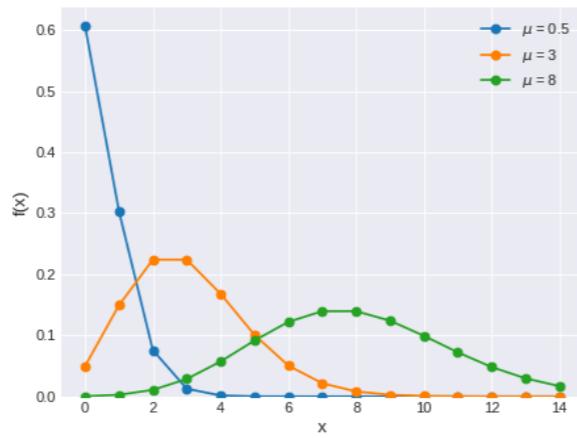


Probability theory is nothing but common sense reduced to calculation. -Pierre-Simon Laplace

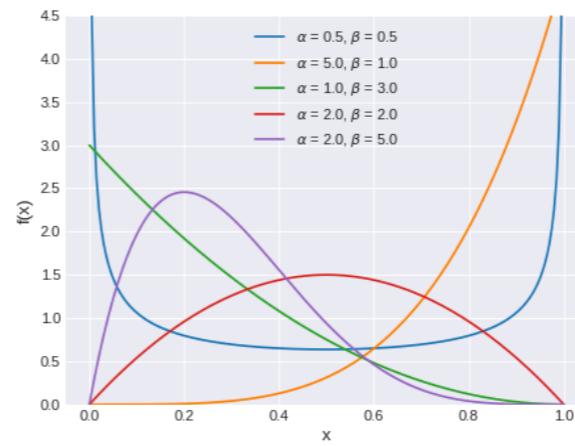
Function/mapping

# Probability distribution

$$f(x | \mu) = \frac{e^{-\mu} \mu^x}{x!}$$



$$f(x | \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}$$



More conventional distributions that maps parameter and observation to a R+ value

Family of distribution parameterised by different input, once the parameter is set (e.g., mu and sd for a Normal), it becomes one distribution.

# Random Variable

$$y \sim \pi(\theta)$$

“Alice flipped a coin 5 times and observed 3 heads”

- Setting 1: Flip total 5 times and records the output.  
**Binomial(y|p,n)**

- Setting 2: Flip until observed 3 heads.  
**Negative binomial(n|p,y)**



A random variable a function that maps the outcomes of unpredictable processes to numerical quantities (labels). You can think of it as a way to describe or summarise a series of random outcomes/events.  
Interesting consequence eg Likelihood Principle

# PMF and Likelihood

---

- Setting 1: Alice flip a coin 10 times and records the output.

$\text{Binomial}(y|p,n)$

$$\pi(x | n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$

$$f(x, n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$

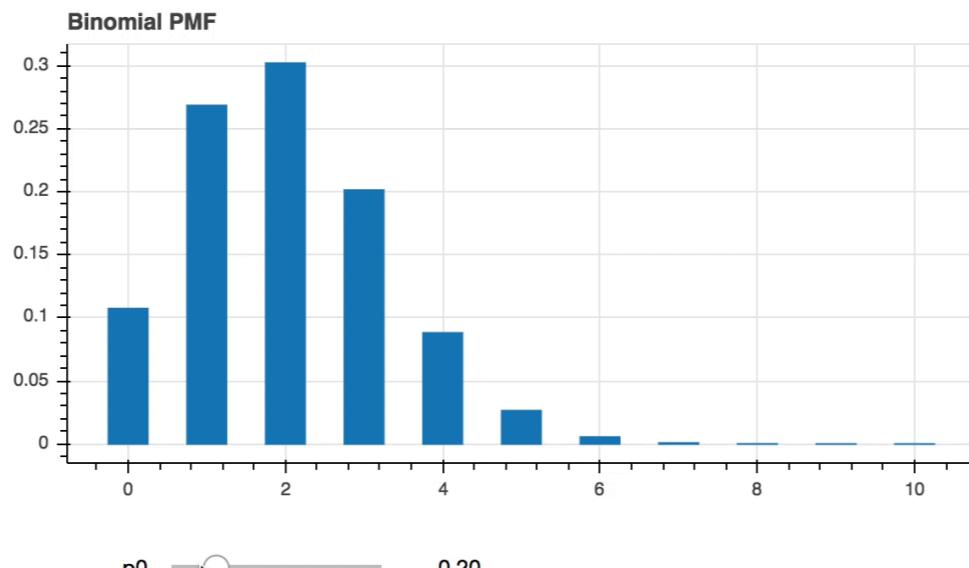


Likelihood\_visual\_demo.ipynb



# PMF and Likelihood

```
import scipy.stats as st
n0, p0 = 10, .2
x0 = np.arange(n0+1)
y0 = st.binom.pmf(x0, n0, p0)
```



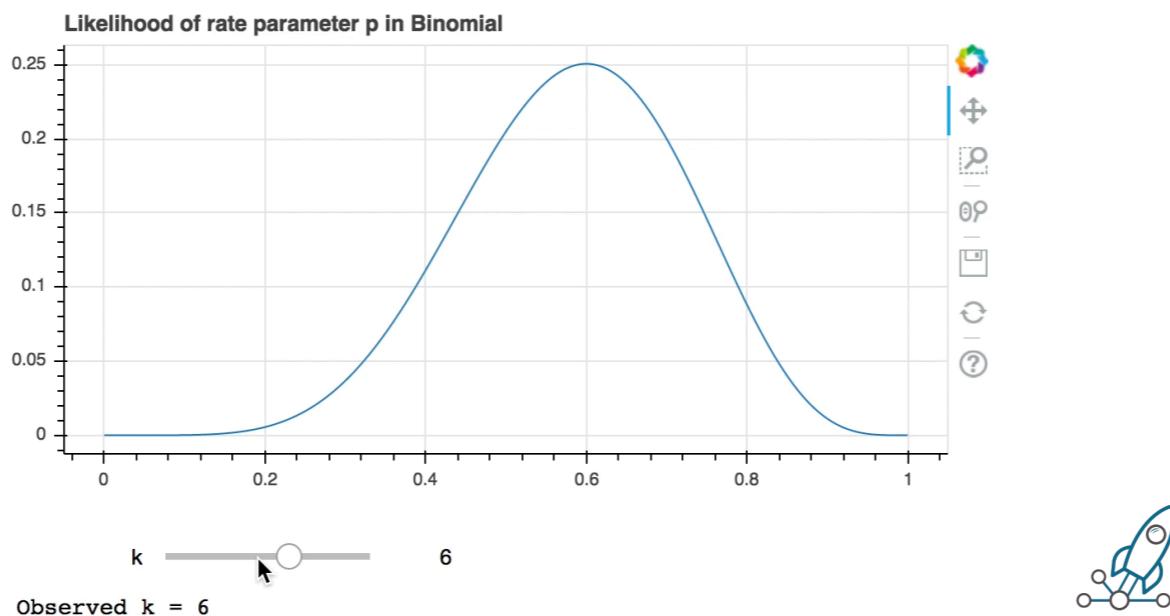
p0  0.20



Coin flip example: flip a coin 10 times, how many times it will come up head?

# PMF and Likelihood

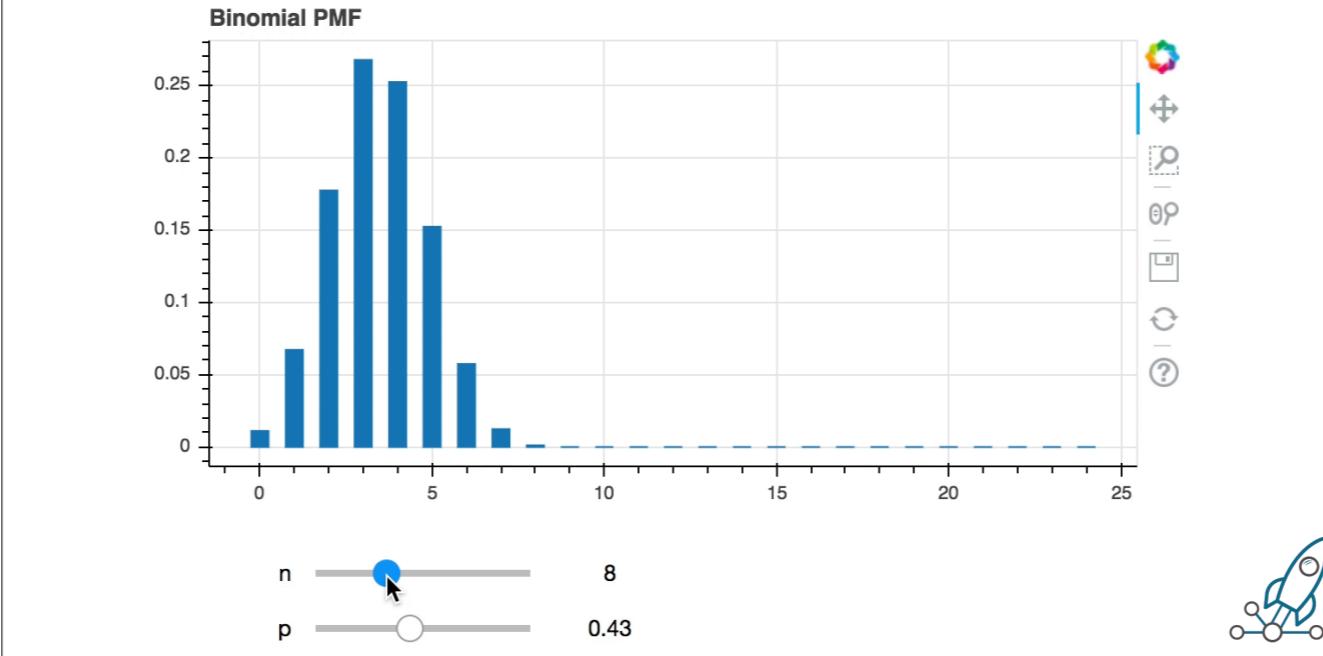
```
k = 6  
xp = np.linspace(0., 1., 200)  
ll = st.binom.pmf(k, n0, xp)
```



Now say we observed the number of head K

# PMF and Likelihood

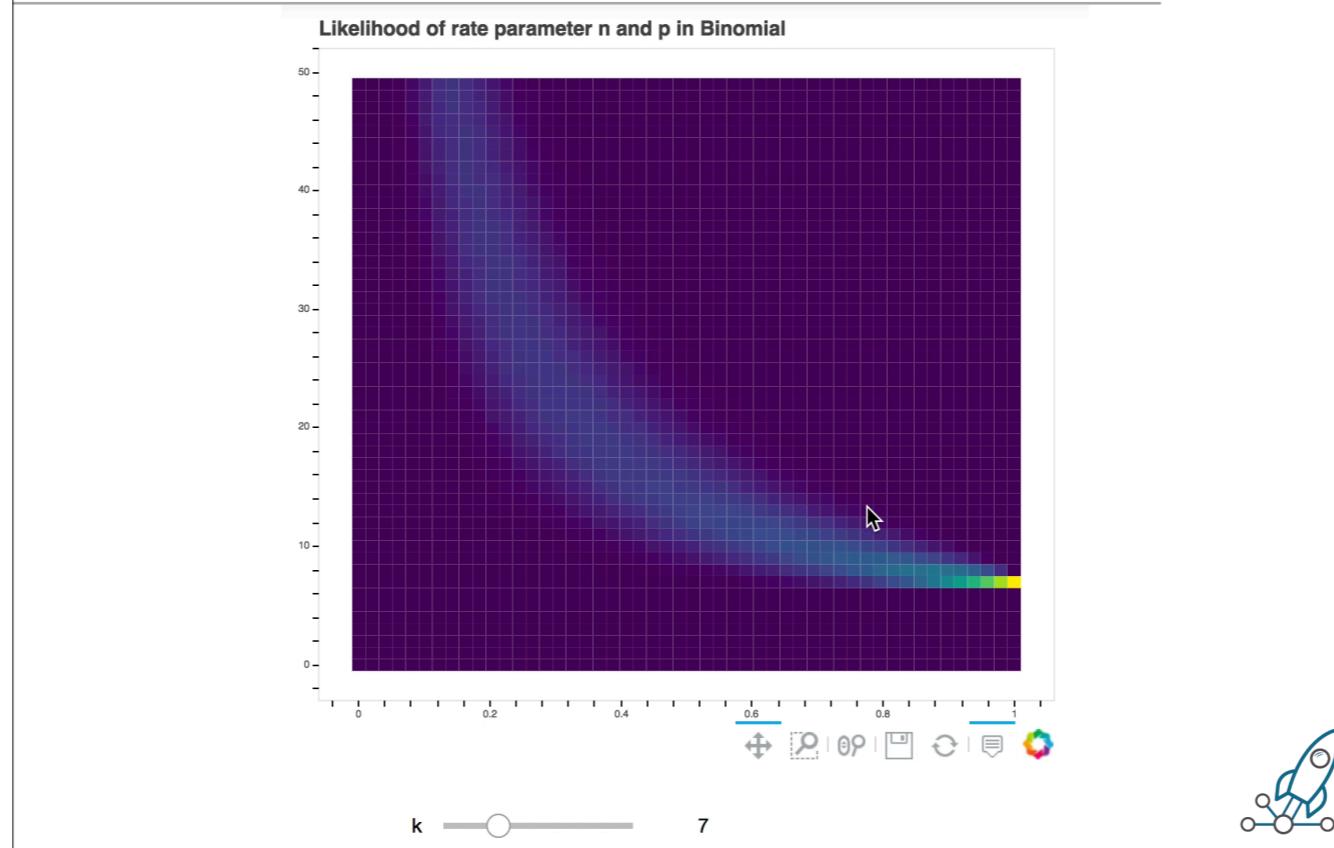
$$\pi(x | n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$



Binomial distribution has two parameters, n and p. Thus we can vary both and see what are the possible outputs.

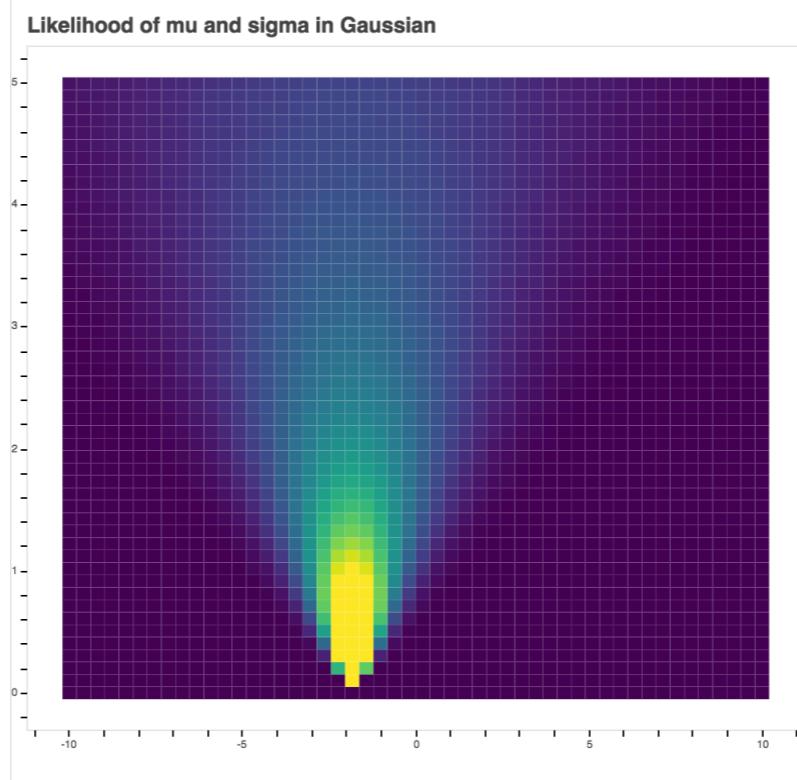
Note: the entirety of Binomial distribution with `Binomial(n, p)` is a family of distribution, once the n and p is fixed, it becomes a distribution of the outcome k.

# PMF and Likelihood



Thus, when you observed a certain output (say 7 heads), both no information of n (number of coin flip) and p (rate parameter), then we are trying to infer 2 variables:

# Gaussian Likelihood



Another example: Gaussian likelihood, which is more natural to think about 2 parameters (mu and sigma)

# PMF and Likelihood

- Setting 1: Alice flip a coin 10 times and records the output.

**Binomial( $y|p,n$ )**

$$\pi(x | n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$

$$f(x, n, p) = \binom{n}{x} p^x (1 - p)^{n-x}$$

`st.binom.pmf(k, n, p)`



From a programming or computation perspective, we are using the same function from scipy, just feeding different input. The differences of pmf and likelihood being from which coordinate/axis we are thinking about in terms of the input

# Combining probabilities

Summary of probabilities

Event	Probability
A	$P(A) \in [0, 1]$
not A	$P(A^c) = 1 - P(A)$
A or B	$P(A \cup B) = P(A) + P(B) - P(A \cap B)$ $P(A \cup B) = P(A) + P(B)$ if A and B are mutually exclusive
A and B	$P(A \cap B) = P(A B)P(B) = P(B A)P(A)$ $P(A \cap B) = P(A)P(B)$ if A and B are independent
A given B	$P(A   B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B A)P(A)}{P(B)}$

<https://en.wikipedia.org/wiki/Probability>



# Sum rule and product rule

---

Sum rule:

$$P(X) = \int_Y P(X, Y)$$

Product rule:

$$P(X, Y) = P(X | Y)P(Y)$$



[https://en.wikipedia.org/wiki/Law\\_of\\_total\\_probability](https://en.wikipedia.org/wiki/Law_of_total_probability)

[https://en.wikipedia.org/wiki/Chain\\_rule\\_\(probability\)](https://en.wikipedia.org/wiki/Chain_rule_(probability))

## Likelihood of observing:

---

$X = [0, 0, 0, 0, 1, 1]$  with  $p = .7$

```
p = .7
y = np.asarray([0, 0, 0, 0, 1, 1])

prob = [p if y_ == 1 else (1-p) for y_ in y]
prob = np.cumprod(prob)
prob

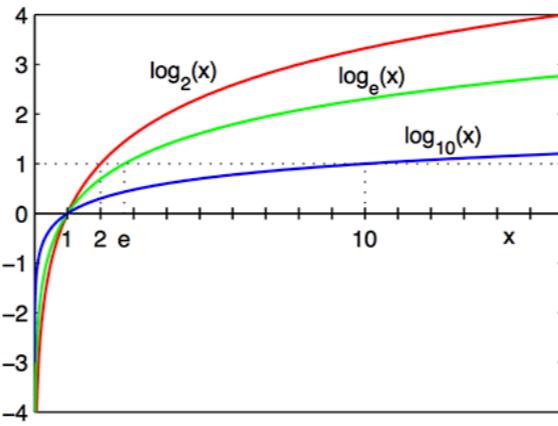
array([0.3      , 0.09      , 0.027     , 0.0081    , 0.00567   , 0.003969])
```

```
prob2 = np.cumprod(st.bernoulli.pmf(y, p))
prob2

array([0.3      , 0.09      , 0.027     , 0.0081    , 0.00567   , 0.003969])
```



# Log-Likelihood



```
np.cumsum(np.log(st.bernoulli.pmf(y, p)))
```

```
array([-1.2039728 , -2.40794561, -3.61191841, -4.81589122, -5.17256616,
       -5.52924111])
```

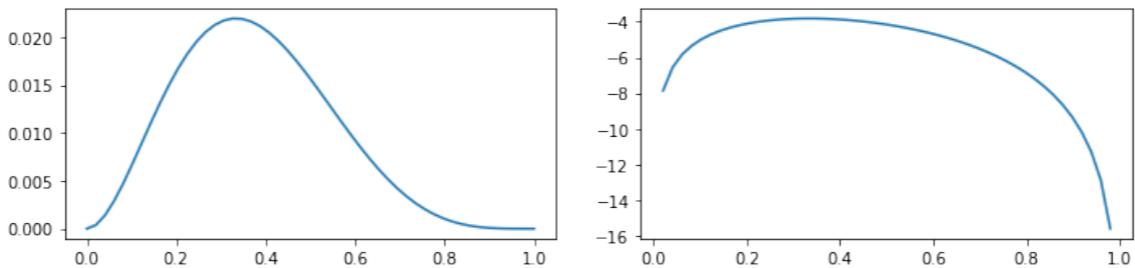
```
st.bernoulli.logpmf(y, p).cumsum()
```



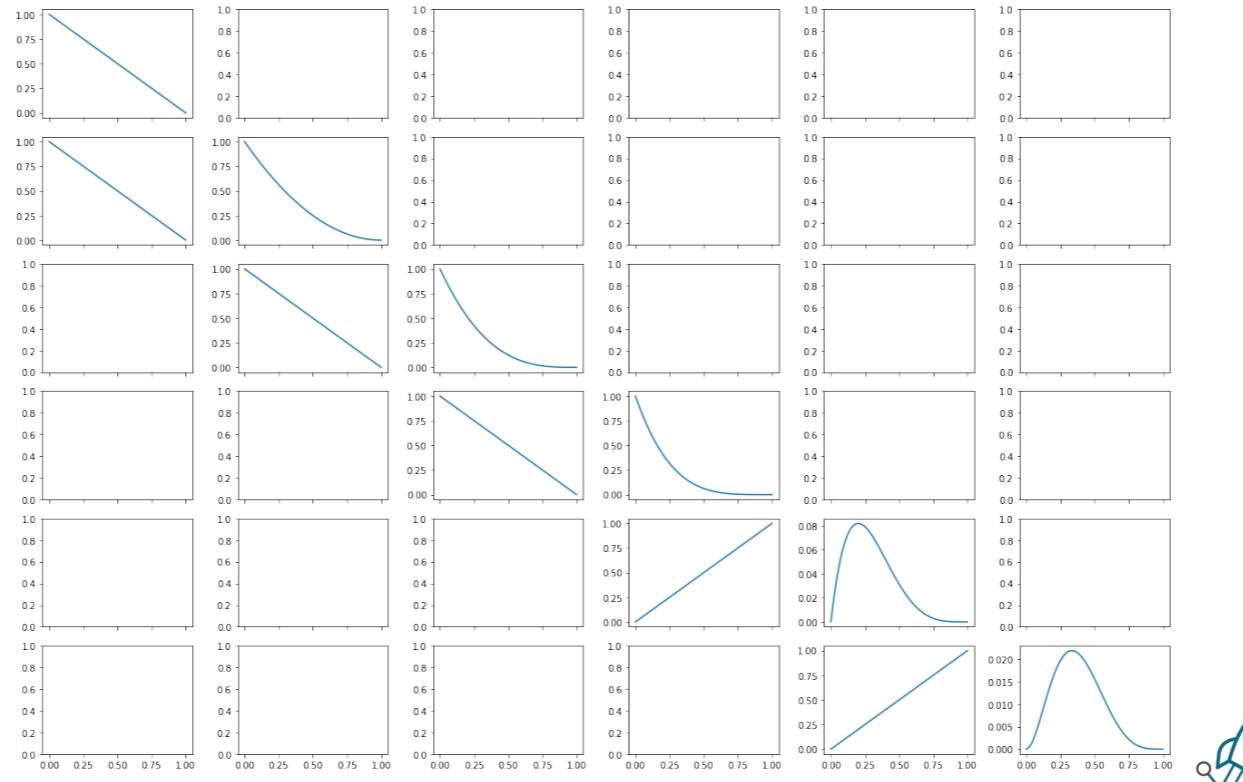
Logarithm function. It has many nice properties: monotonic, turn product into sum, nice for computer to work with.

# Combining Likelihood

```
X = [0, 0, 0, 0, 1, 1]  
lambda p: st.bernoulli.pmf(y, p).prod()
```



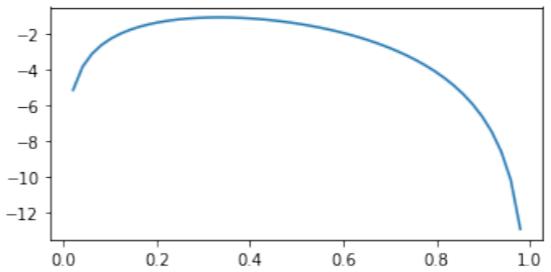
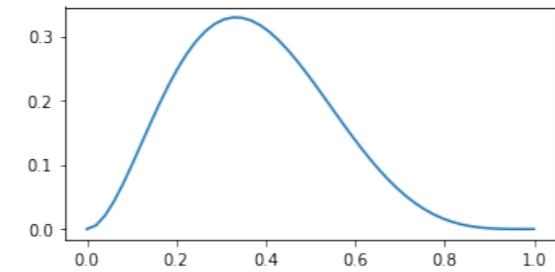
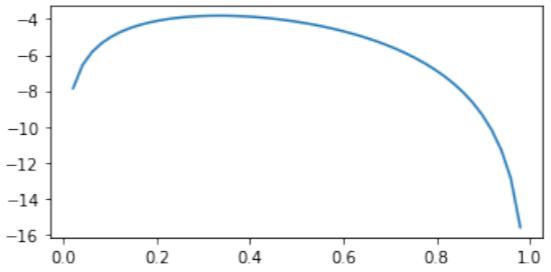
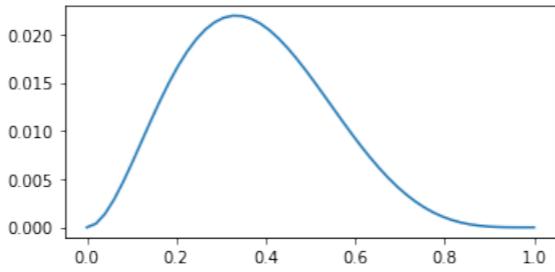
# Update of Likelihood function



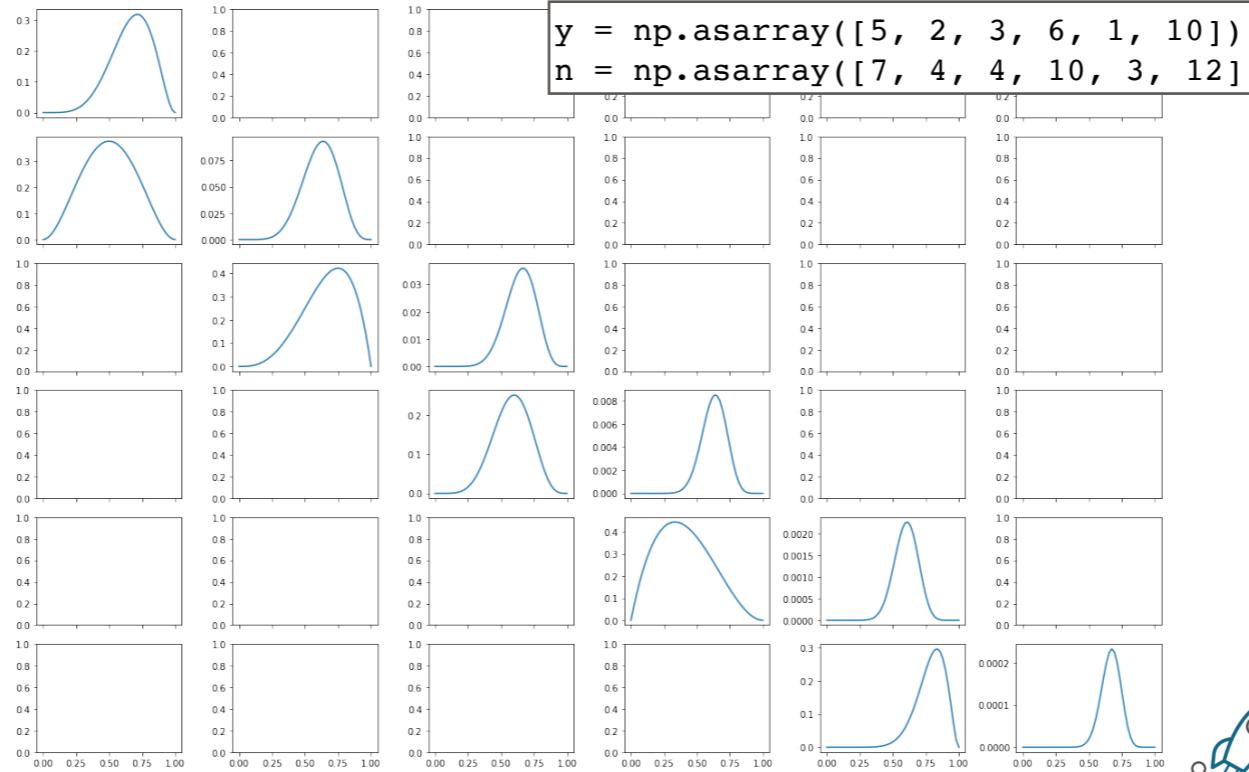
# Likelihood function

```
lambda p: st.bernoulli.pmf(y, p).prod()
```

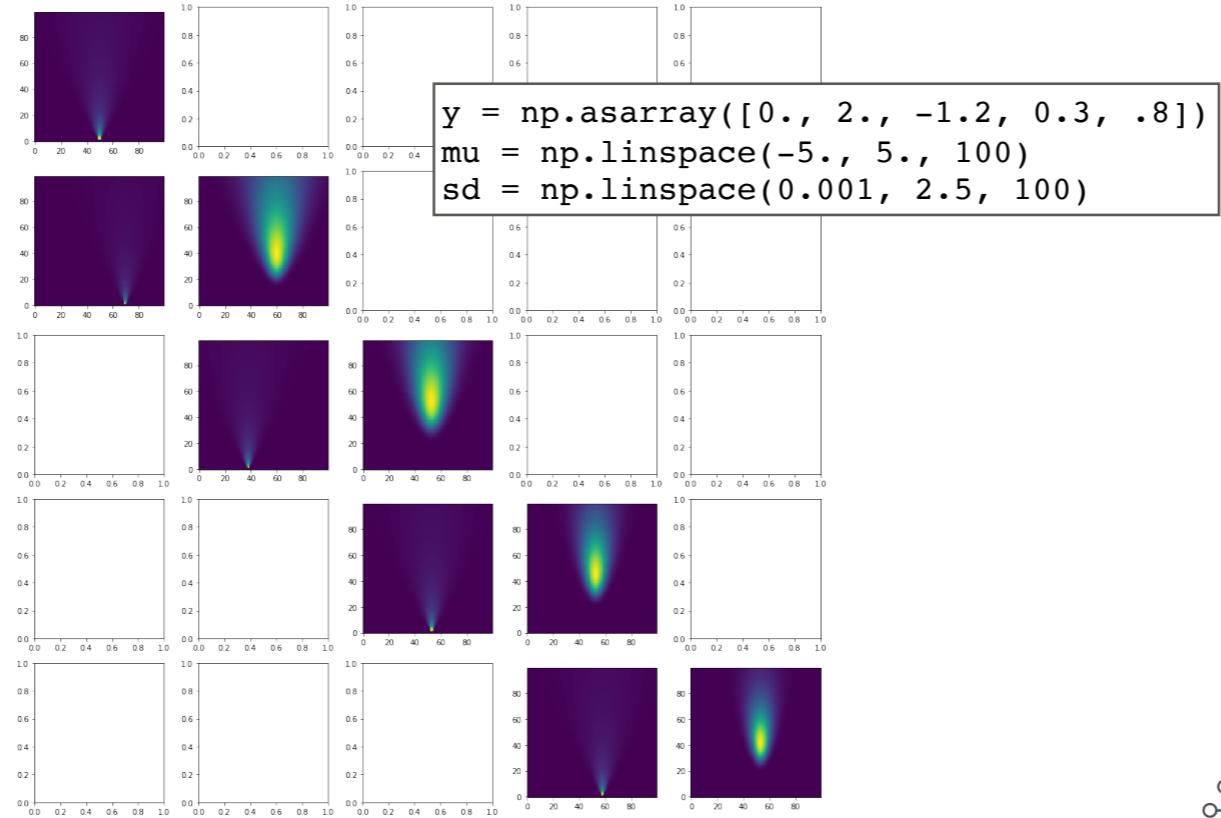
```
lambda p: st.binomial.pmf(y.sum(), len(y), p).prod()
```



# Update, or cumulating information



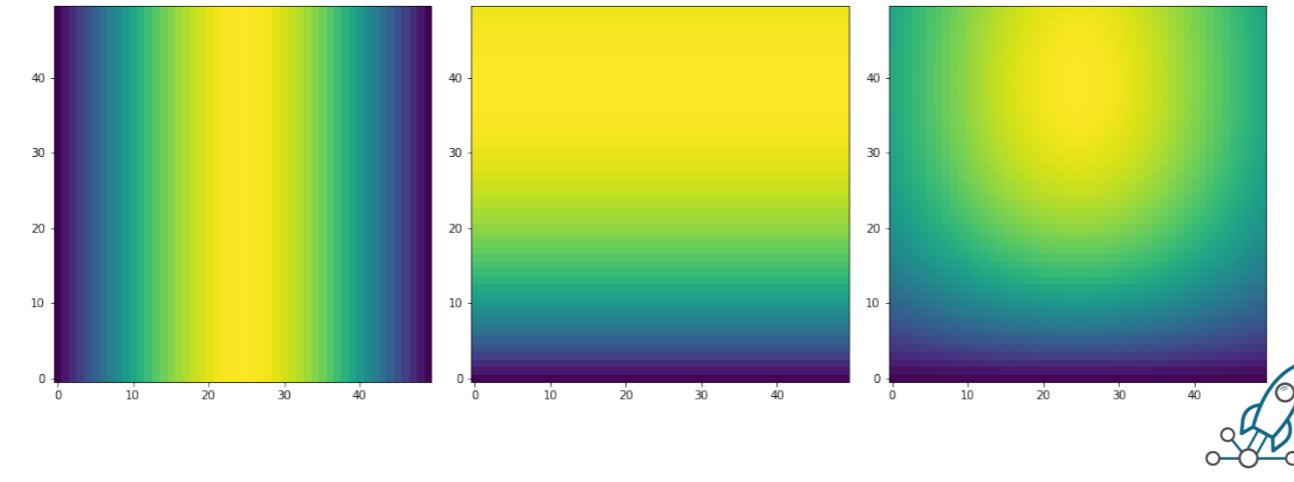
# Update, or cumulating information



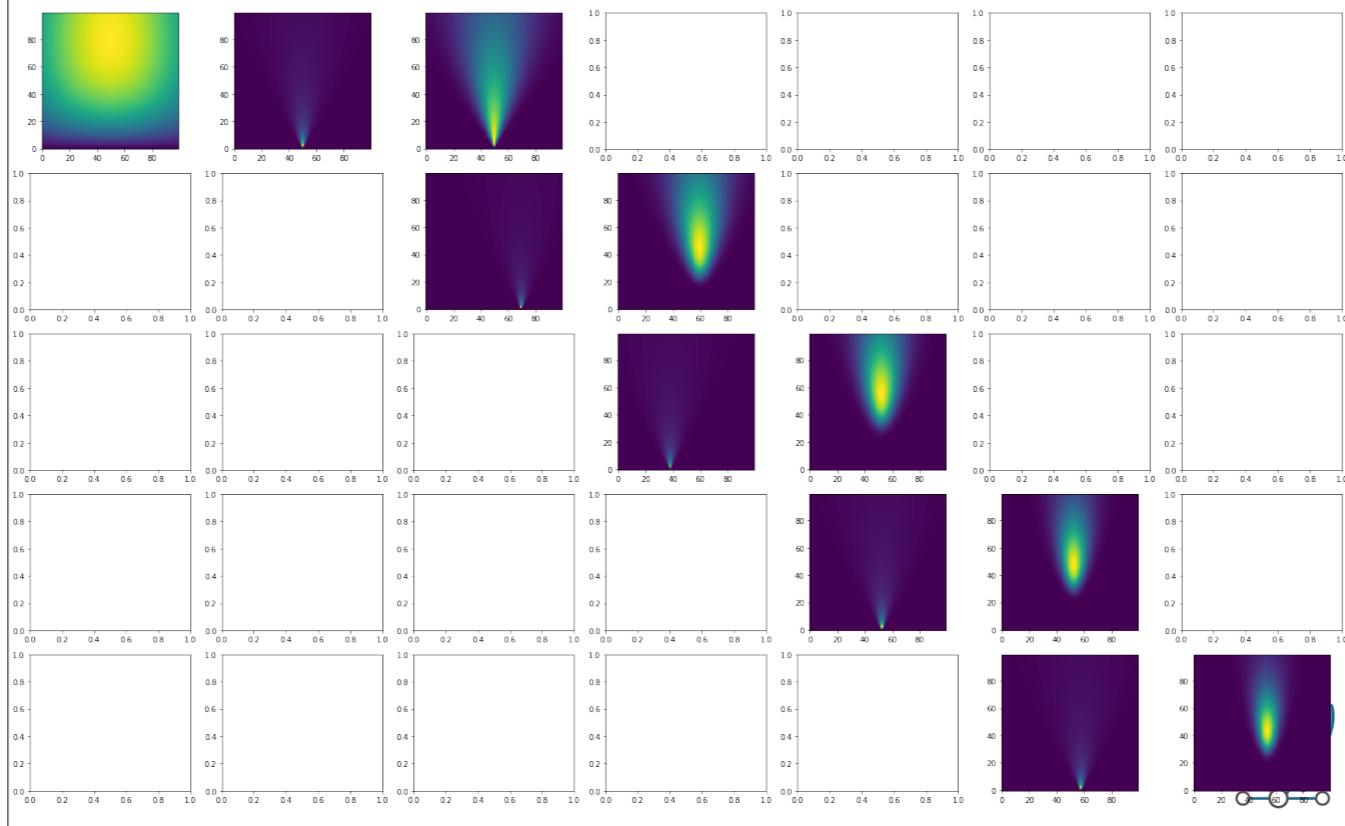
# Priors

A and B	$P(A \cap B) = P(A B)P(B) = P(B A)P(A)$ $P(A \cap B) = P(A)P(B)$ if A and B are independent
---------	--

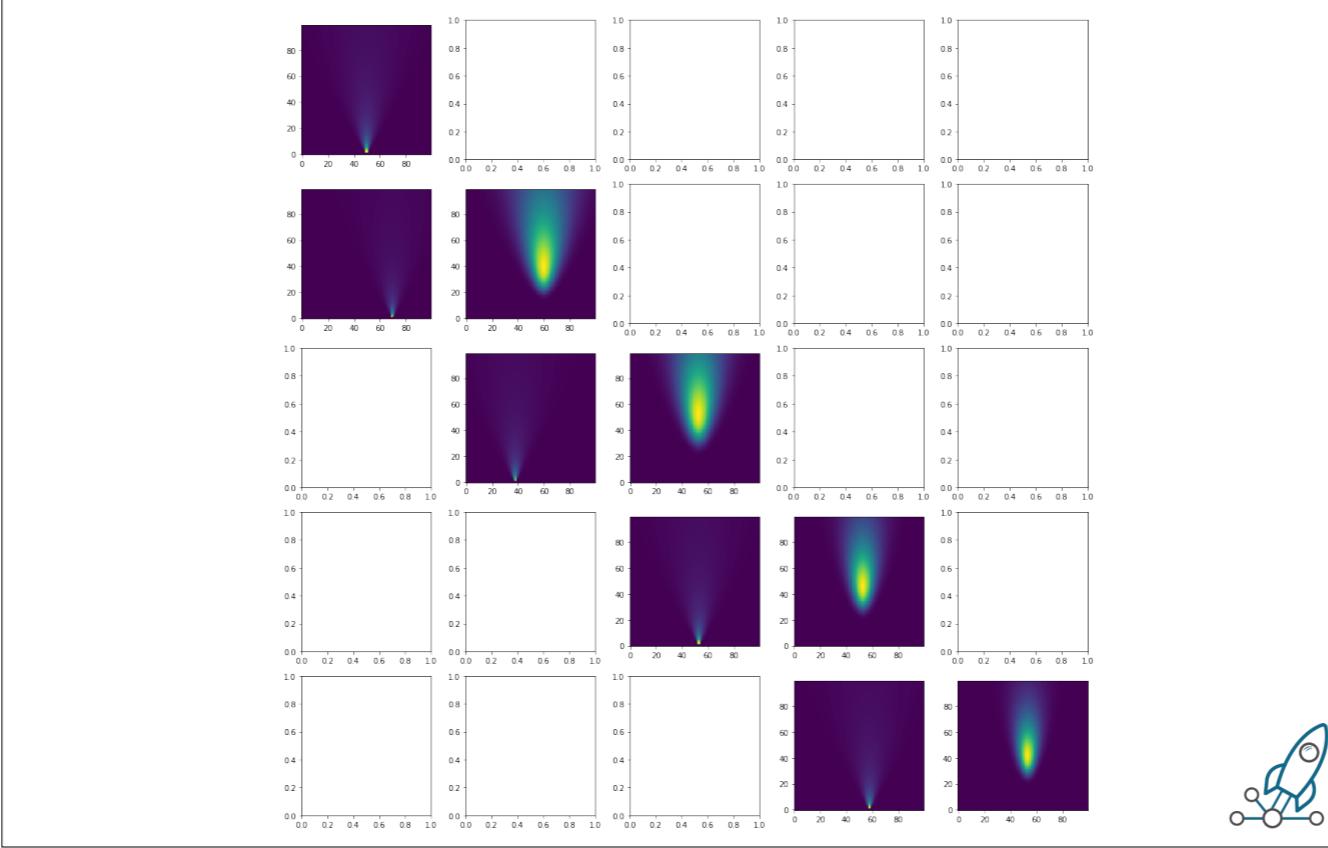
```
mu_prior = st.norm.pdf(mu, 0, 5)
sd_prior = st.gamma.pdf(sd, 2., scale=1.0/.5)
```



# Update, or cumulating information



# As comparison...



change the first data point, see how the joint likelihood after the first data point looks like

# Coin flip in PyMC3

```
k = 7  
k ~ Binomial(n=n, p=p)  
n ~ DiscreteUniform(0, 25)  
p ~ Uniform(0., 1.)
```

```
with pm.Model() as m:  
    n = pm.DiscreteUniform('n', 0, 25)  
    p = pm.Uniform('p', 0., 1., transform=None)  
    y = pm.Binomial('k', n, p, observed=7)
```



# Coin flip in PyMC3

```
with pm.Model() as m:  
    n = pm.DiscreteUniform('n', 0, 25)  
    p = pm.Uniform('p', 0., 1., transform=None)  
    y = pm.Binomial('k', n, p, observed=7)
```

```
point = m.test_point  
{'n': array(12), 'p': array(0.5)}
```

```
logp_m = m.logp  
logp_y = y.logp
```

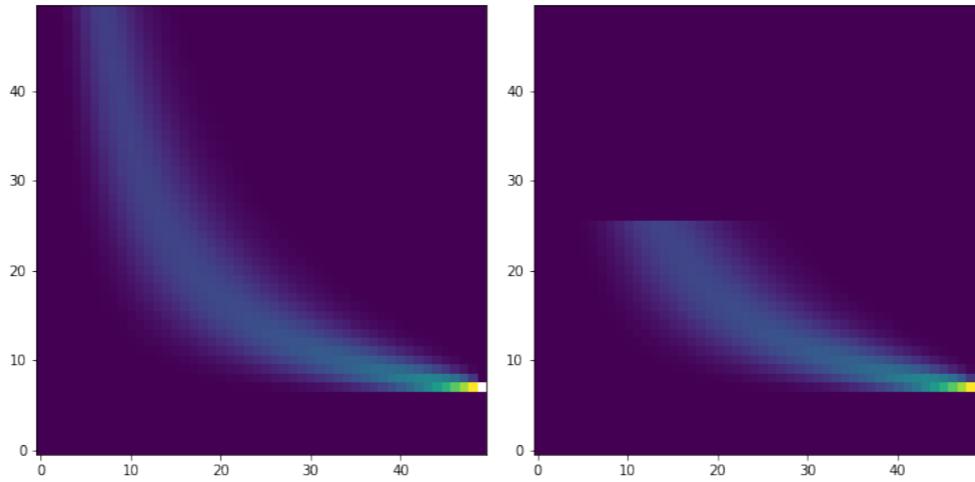
```
logp_m
```

```
<pymc3.model.LoosePointFunc at 0x125656048>
```



# Coin flip in PyMC3

```
for i in range(len(lly)):  
    point[ 'n' ] = nv_.flatten()[i]  
    point[ 'p' ] = pv_.flatten()[i]  
    lly[i] = np.exp(logp_y(point))  
    llm[i] = np.exp(logp_m(point))
```



next observed k = 5?

Implicit condition: n >= 7

What if we observed k = 10? should we update this implicit condition? or it doesn't matter?

k = 30? -> impossible model

Importance of thinking about the prior - otherwise waste of computation resource, worse model being modified silently.

# How PyMC3 use Theano

When we define a PyMC3 model, we implicitly build up a Theano function from the space of our parameters to their posterior probability density up to a constant factor.

`m.loge`

$$f: \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$$

`m.free_RVs`  
[n, p]

<http://docs.pymc.io/theano.html>



When we define a PyMC3 model, we implicitly build up a Theano function from the space of our parameters to their posterior probability density up to a constant factor. We then use symbolic manipulations of this function to also get access to its gradient.

# How PyMC3 use Theano

---

```
m.logp
```

$$f: \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$$

```
m.free_RVs  
[n, p]
```

```
n.distribution.logp
```

```
<bound method DiscreteUniform.logp of  
<pymc3.distributions.discrete.DiscreteUniform  
object at 0x125ac6358>>
```

```
n.logpt
```



# How PyMC3 use Theano

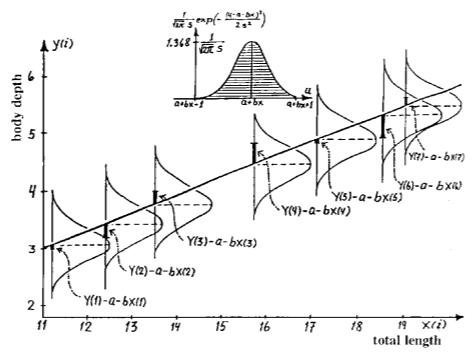
m.logp

$$f: \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$$

```
m.logpt??  
@property  
def logpt(self):  
    """Theano scalar of log-probability of the model"""  
    with self:  
        factors = [var.logpt for var in self.basic_RVs] +  
self.potentials  
        logp = tt.sum([tt.sum(factor) for factor in factors])  
        if self.name:  
            logp.name = '__logp_%s' % self.name  
        else:  
            logp.name = '__logp'  
    return logp
```



# General linear model



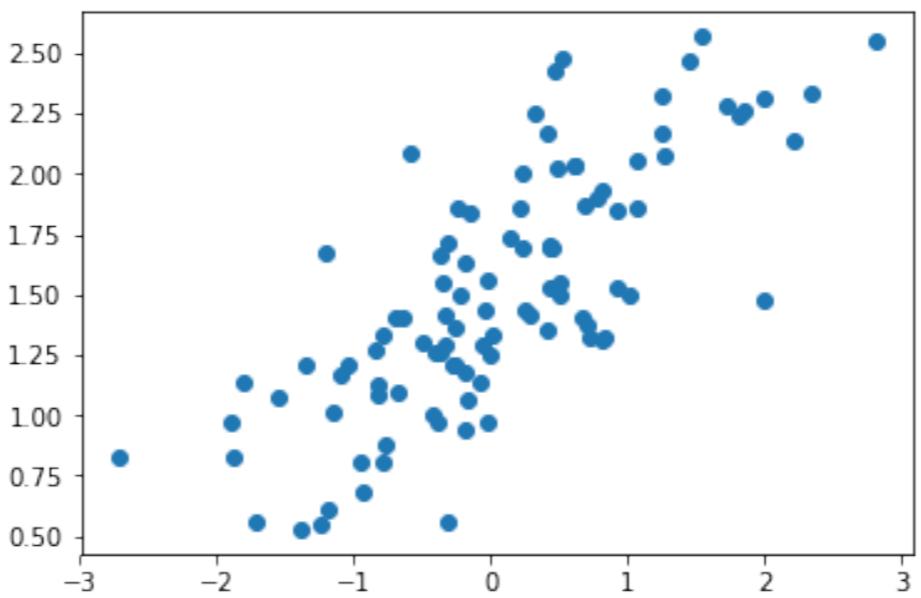
ANOVA

Source	d.f.	SS	MS	F
Treatment	$a - 1$	$SS_{\text{treat}}$	$\frac{SS_{\text{treat}}}{a-1}$	$\frac{MS_{\text{treat}}}{MS_{\text{error(a)}}}$
Error (a)	$N - a$	$SS_{\text{error(a)}}$	$\frac{SS_{\text{error(a)}}}{N-a}$	
Time	$t - 1$	$SS_{\text{time}}$	$\frac{SS_{\text{time}}}{t-1}$	$\frac{MS_{\text{time}}}{MS_{\text{error(b)}}}$
Treat x Time	$(a - 1)(t - 1)$	$SS_{\text{treat} \times \text{time}}$	$\frac{SS_{\text{treat} \times \text{time}}}{(a-1)(t-1)}$	$\frac{MS_{\text{treat} \times \text{time}}}{MS_{\text{error(b)}}}$
Error (b)	$(N - a)(t - 1)$	$SS_{\text{error(b)}}$	$\frac{SS_{\text{error(b)}}}{(N-a)(t-1)}$	
Total	$Nt - 1$	$SS_{\text{total}}$		

$$\mathbf{y} = \begin{pmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{31} \\ y_{32} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 2 \\ 1 & 2 \\ 1 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{12} \\ \varepsilon_{21} \\ \varepsilon_{22} \\ \varepsilon_{23} \\ \varepsilon_{31} \\ \varepsilon_{32} \end{pmatrix} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \quad \mathbf{y} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} + \boldsymbol{\varepsilon} = \mathbf{X}_* \boldsymbol{\beta}_* + \boldsymbol{\varepsilon},$$

let's look at another example

# Linear regression in PyMC3



Regression with a twist.ipynb



What are the free parameters (i.e., input to the logp function)

# Linear regression in PyMC3

```
with pm.Model() as m0:
    beta = pm.Normal('beta', 0., 10.)
    a = pm.Normal('a', 0., 10.)
    pm.Normal('y', X*beta+a, 1., observed=y)

with pm.Model() as m1:
    beta = pm.Flat('beta')
    a = pm.Flat('a')

    pm.Potential('logp_beta',
                 pm.Normal.dist(0., 10).logp(beta))
    pm.Potential('logp_a',
                 pm.Normal.dist(0., 10).logp(a))
    pm.Potential('logp_obs',
                 pm.Normal.dist(X*beta+a, 1.).logp(y))
```



What are the free parameters (i.e., input to the logp function)

# Linear regression in PyMC3

```
with pm.Model() as m0:  
    ...  
    pm.Normal('y', X*beta+a, sd, observed=y)  
  
with pm.Model() as m1:  
    ...  
    pm.Normal('eps', 0, sd, observed=y - X*beta - a)
```



# Linear regression in PyMC3

```
with pm.Model() as m0:  
    ...  
    pm.Normal('y', X*beta+a, sd, observed=y)  
  
with pm.Model() as m1:  
    ...  
    pm.Normal('eps', 0, sd, observed=y - X*beta - a)  
  
with pm.Model() as m2:  
    ...  
    pm.Normal('eps', 0, 1,  
              observed=(y - X*beta - a)/sd)
```



what about we divide the observed by sd and assumpt distributed from Normal(0, 1)?

## Recap:

---

We “glue” random variables together in a conditional network to create a mapping that is the (log-) likelihood function.

The dimension of the parameter space is the same as the number of unknowns.

When we are building model, we are setting up the space (concept of coordinate system)



It is fun to then think about what inference it is exactly: what is model fitting? why do we want to sample from the posterior?  
Also interesting to think about minibatch method in machine learning, gradient decent.

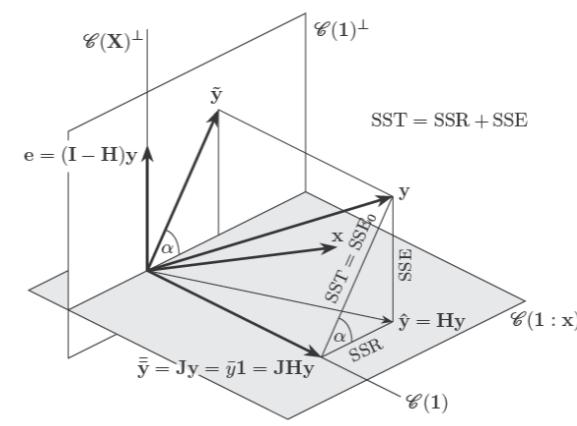
# L1 and L2 regularization

**Lasso Regression** (Least Absolute Shrinkage and Selection Operator)

$$\sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

**Ridge regression**

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$



In that regard, it is interesting to reconsider the usual regularisation we have in machine learning.

# Parameterization

---

There are many ways to write down the same model:

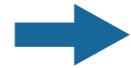
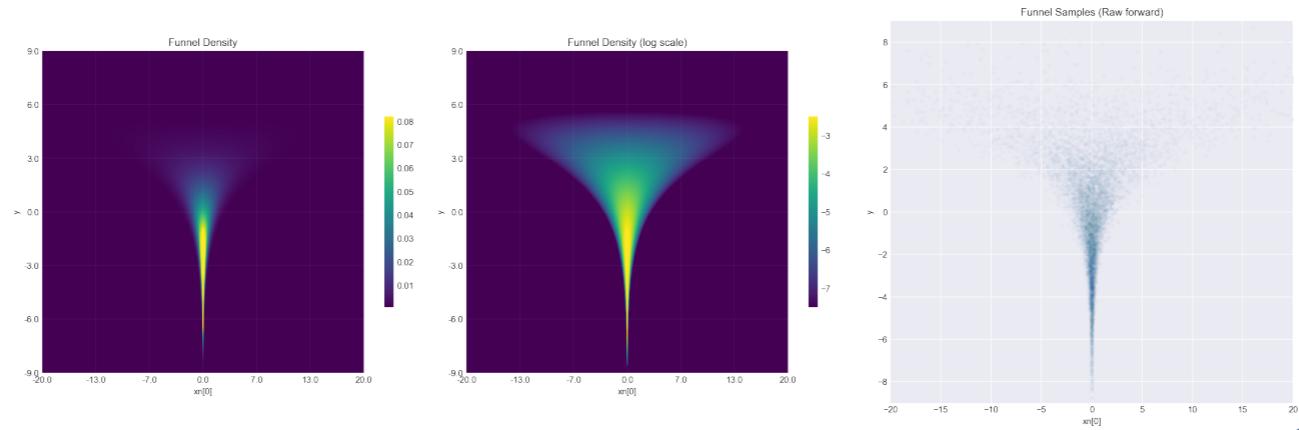
- Make sure it is correct
- Slow model usually indicates problem
  - *The folk theorem of statistical computing: When you have computational problems, often there's a problem with your model*
- A good parameterisation should be easy to understand



[http://andrewgelman.com/2008/05/13/the\\_folk\\_theore/](http://andrewgelman.com/2008/05/13/the_folk_theore/)

# Parameterization: Neal's Funnel

$$p(y, x_n) = \text{Normal}(y \mid 0, 3) \times \prod_{n=1}^9 \text{Normal}(x_n \mid 0, \exp(y/2))$$



Neals\_funnel.ipynb



Stan user manual 28.6

a general transform from a centered to a non-centered parameterization Papaspiliopoulos et al. (2007). This reparameterization is helpful when there is not much data, because it separates the hierarchical parameters and lower-level parameters in the prior.

(Neal, 2003) defines a distribution that exemplifies the difficulties of sampling from some hierarchical models.

The probability contours are shaped like ten-dimensional funnels. The funnel's neck is particularly sharp because of the exponential function applied to  $y$ .

# Use logp function to check model

## Inferencing trigonometric time series model

### Questions



narendramukherjee

2  6d

Following the conversation in [How to model sinusoids in white gaussian noise](#), I have been trying to fit a sinusoidal model as well. I have tried a lot of different parametrizations with NUTS, some of which are as follows:

1. Try to fit  $y(t) = A\sin(\omega t) + B\cos(\omega t) + \text{noise}$  where  $A$  and  $B$  are drawn as Normal random variables.
2. Draw  $A$  and  $B$  as Normal random variables, and fit  $y(t) = C\sin(\omega t + \phi)$  where  $C = \sqrt{A^2 + B^2}$  and  $\phi = \tan^{-1} \frac{B}{A}$ .

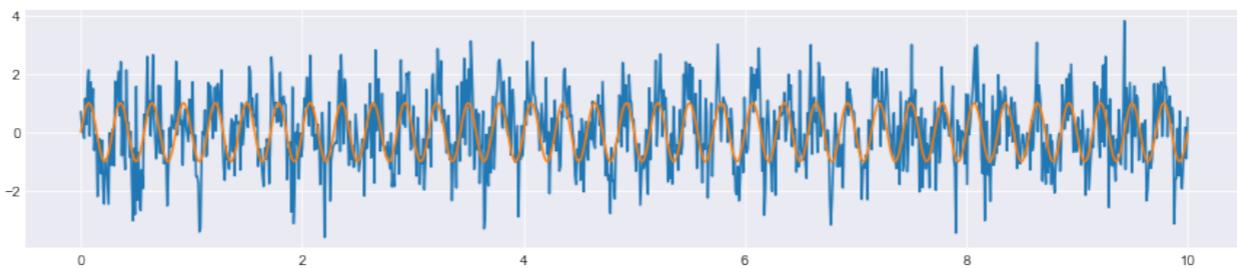
I tried some others which were worse than these two.

The conversation has revolved around the phase parameter mostly in this discussion, but I think the issue lies more with NUTS having problems in a nonlinear model. VERY suprisingly (to me at least!), Metropolis works fabulously in this model (I tried the parametrization number 1 above). There has been at least one previous report of NUTS being miserable in a nonlinear model when Metropolis worked well, and it seems that issue wasn't resolved back then:

<https://discourse.pymc.io/t/inferencing-trigonometric-time-series-model/1190>



# Trigonometric time series model



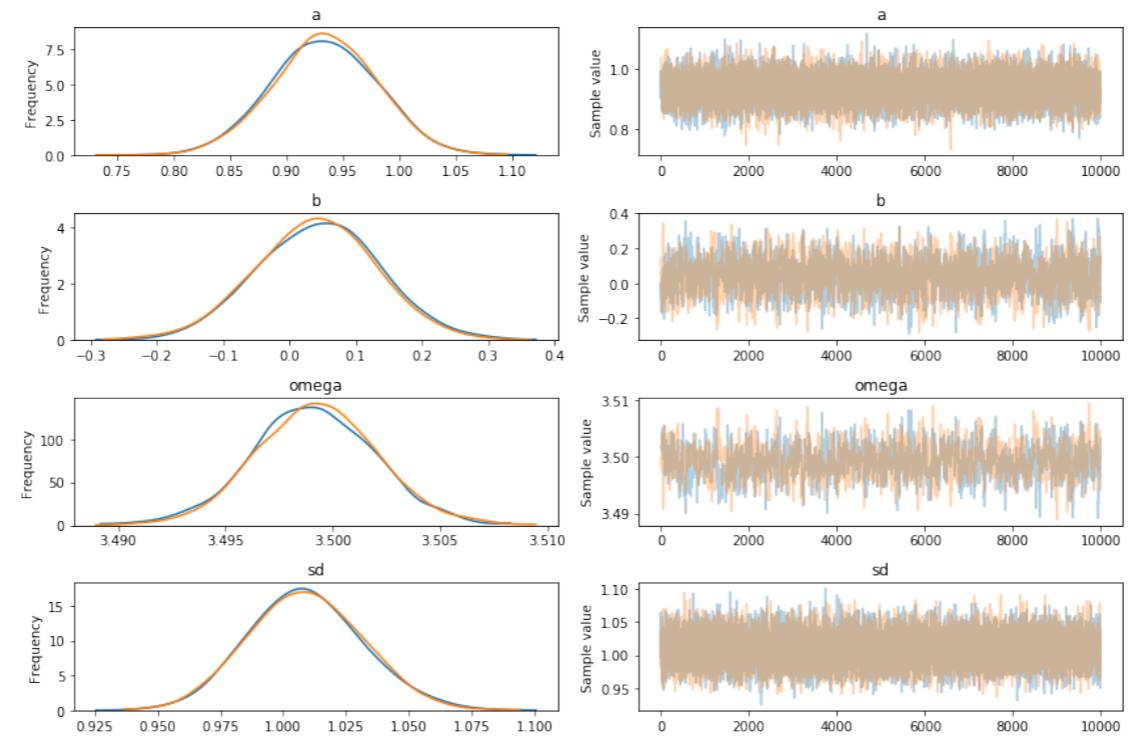
```
with pm.Model() as model:
    a = pm.Normal("a", mu=1, sd=2)
    b = pm.Normal("b", mu=1, sd=2)
    omega = pm.Gamma("omega", 1., 1.)
    regression = a * tt.sin(2 * np.pi * omega * t) +\
                 b * tt.cos(2 * np.pi * omega * t)
    sd = pm.HalfCauchy("sd", 0.5)
    observed = pm.Normal("observed",
                          mu=regression,
                          sd=sd,
                          observed=data)
```



Timeserie\_model.ipynb

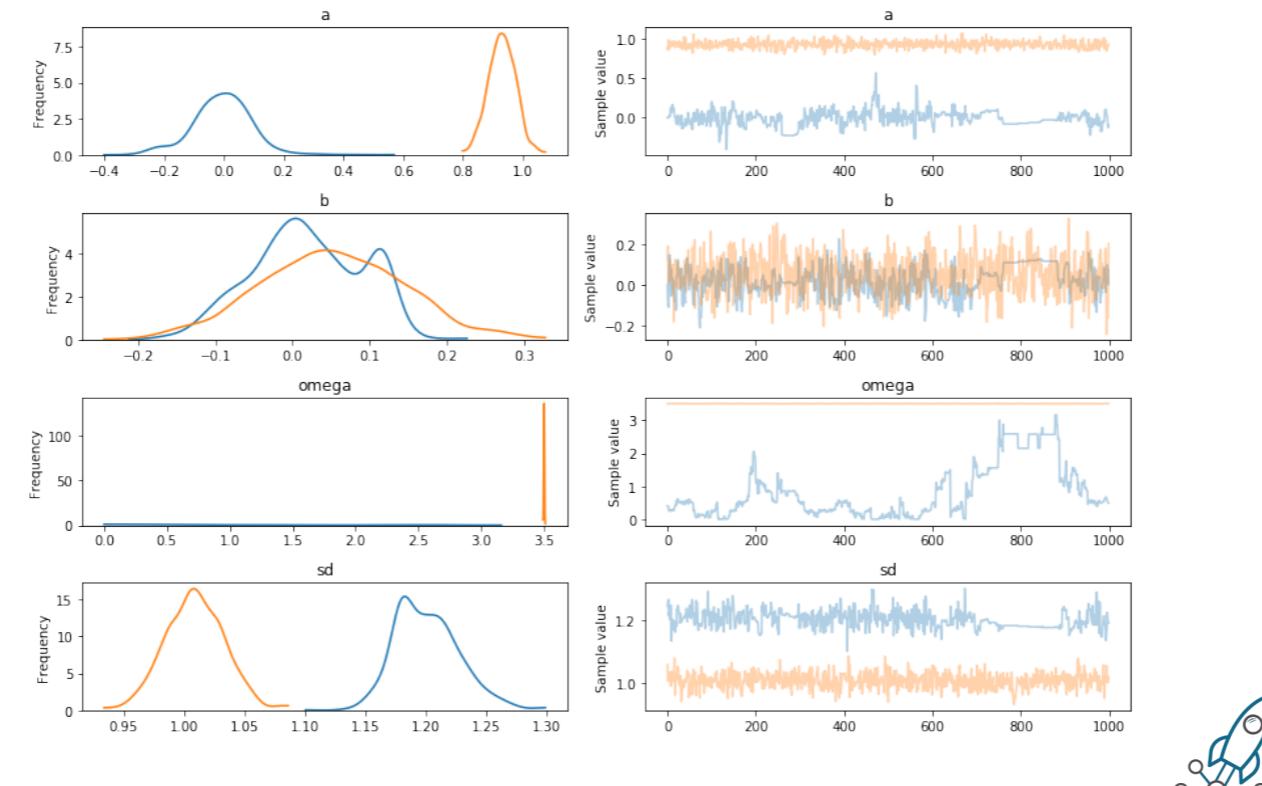


# Trigonometric time series model



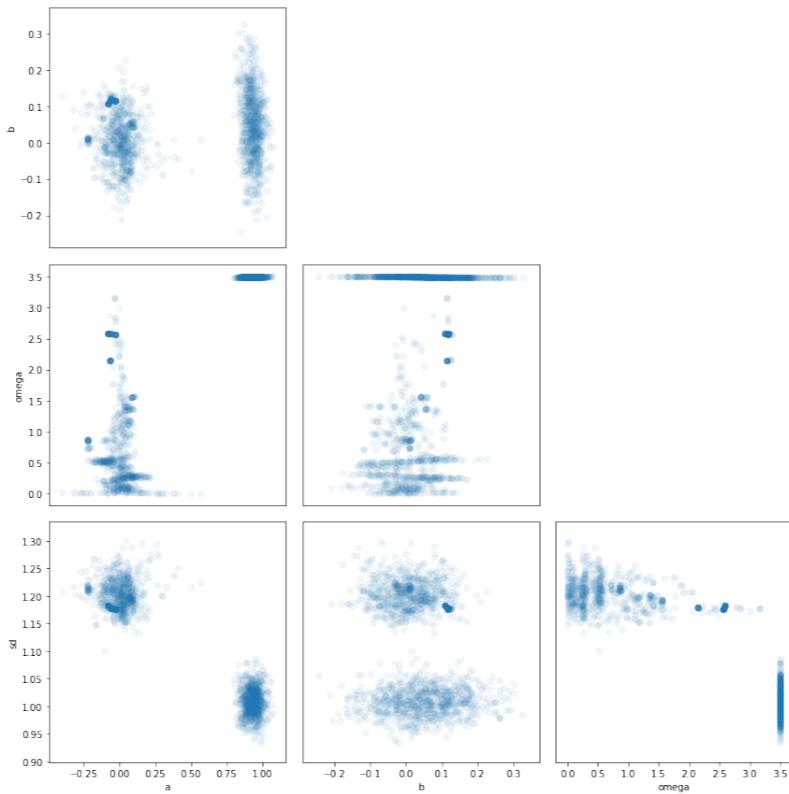
Sampled with metropolis

# Trigonometric time series model



sampled with NUTS

# Trigonometric time series model



# Trigonometric time series model

```
model.free_RVs  
[a, b, omega_log__, sd_log__]  
  
logp_dlogp_cond = model.logp_dlogp_function([model.free_RVs[0],  
                                              model.free_RVs[2]])
```



# Trigonometric time series model

```
pt = model.test_point  
pt  
  
{'a': array(1.),  
 'b': array(1.),  
 'omega_log_': array(0.),  
 'sd_log_': array(-0.69314718)}
```

```
pt['b'] = np.array(0.)  
pt['sd_log_'] = np.log(0.5)  
logp_dlogp_cond.set_extra_values(pt)
```

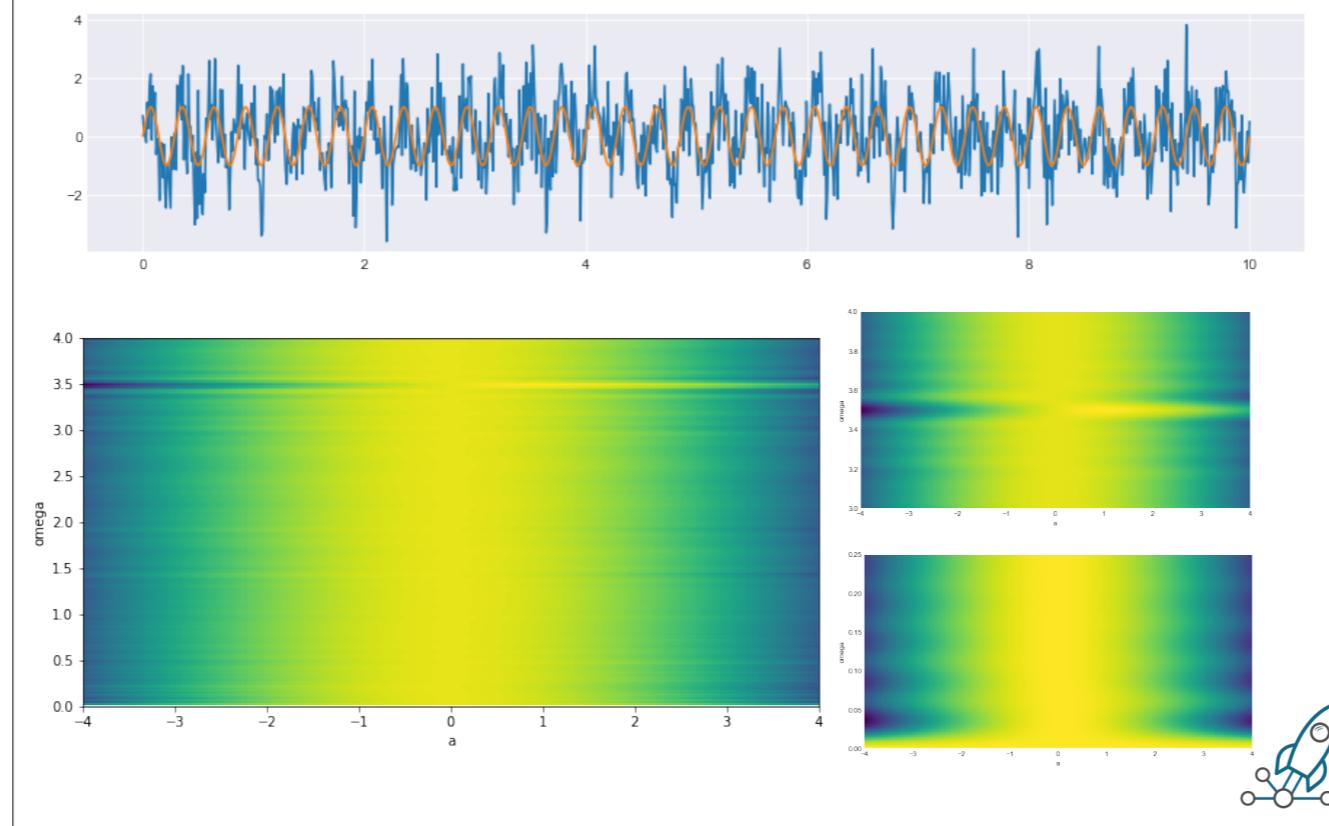
```
pt['a'] = np.array(2.)  
pt
```

```
{'a': array(2.),  
 'b': array(0.),  
 'omega_log_': array(0.),  
 'sd_log_': -0.6931471805599453}
```

```
logp_dlogp_cond.dict_to_array(pt)  
  
array([2., 0.])
```



# Trigonometric time series model



Quite problematically, Random walk Metropolis can fail silently.

Of course, it is not always easy to check the conditional logp this way, as some times the logp function is expensive to evaluate. Also, grid evaluation is slow!!!

## Some thoughts:

- Likelihood is an important concept in Bayesian Computation
  - Not always available or expensive to evaluate (ABC)
- Think in terms of the parameter space
  - Model fitting - what it is really?
- Designing model specific inference
  - Laplace approximation to take expectation of part of the variable
  - Expectation maximisation
- Connection to predictive distribution (elpd)



---

Thanks!