Repository of the code and slides:

https://bit.ly/2JrHNLd

ALL THAT LIKELIHOOD

with PyMC3

Junpeng Lao

July 2018 @ Berlin

Powered by PyMC3

From the [PyMC3 documentation]:

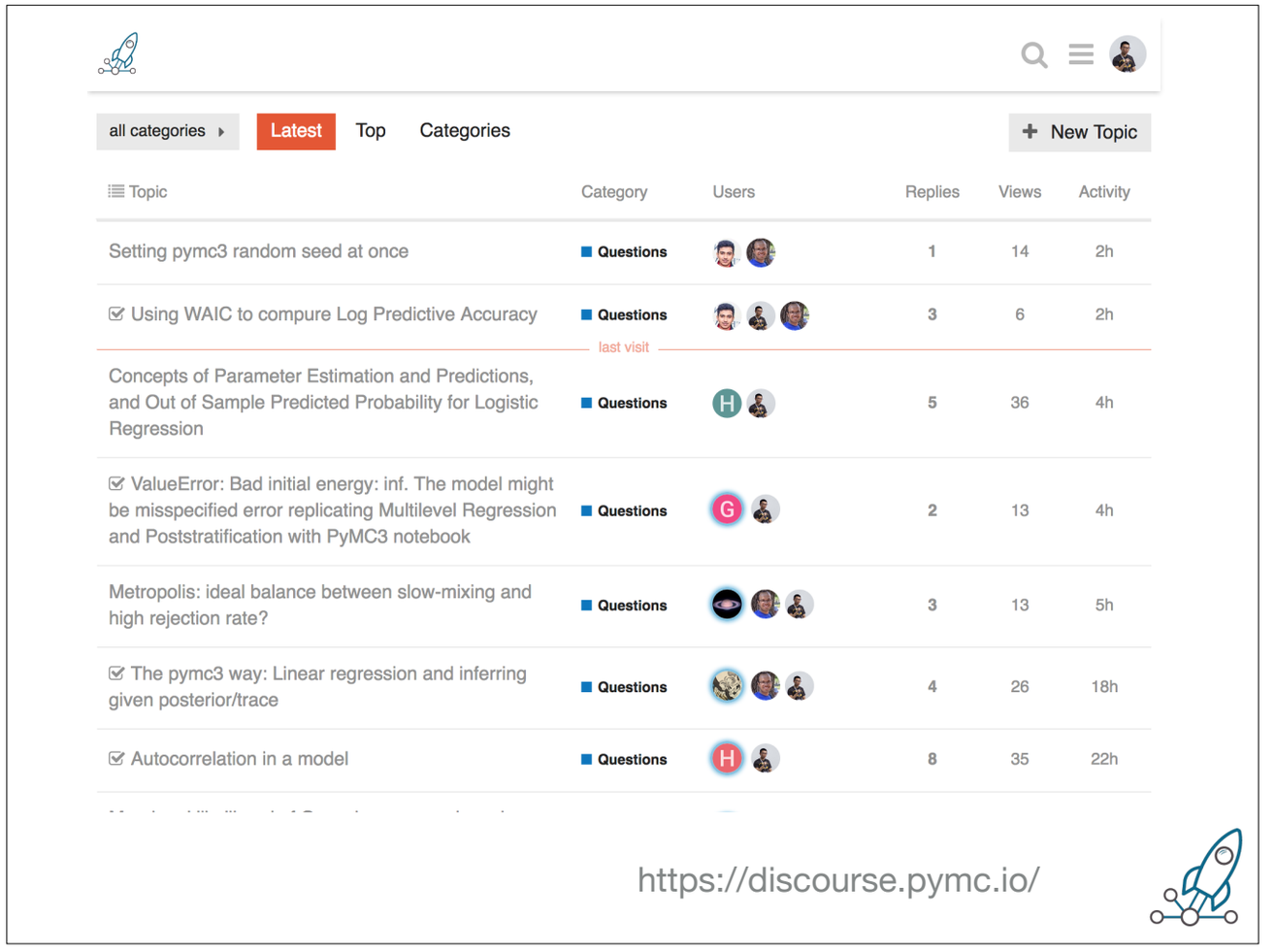
PyMC3 is a Python package for Bayesian statistical modeling and Probabilistic Machine Learning which focuses on **advanced Markov chain Monte Carlo** and variational fitting algorithms. Its flexibility and extensibility make it applicable to a large suite of problems.

Theano is a package that allows us to define functions involving array operations and linear algebra.

Why PyMC3?

Easy to interact with different components of your model.

- in Python
- Has the best inference (NUTS and VI)
- Active development
- Awesome community



I have been developing for pymc3 for the last year or so, currently I mainly modulate the community.

Some haunting questions:

- Why do we sample from the posterior? And how?
- Why does reparameterization “works”?

$x \sim \text{Normal}(\mu, \sigma)$ $\epsilon \sim \text{Normal}(0,1), x = \mu + \sigma\epsilon$

I am from a Psychology background, in my journey of learning Bayesian statistics and computation I had face many challenges:

It all can be better understand…

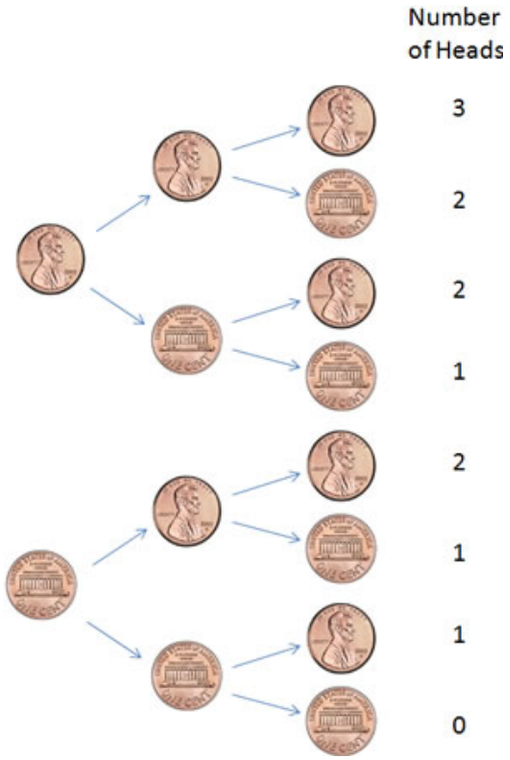
- Probability density/mass function and likelihood function
- Computing Likelihood in PyMC3

Repository of the code and slides:

https://bit.ly/2JrHNLd

Surprisingly, I have a much better perspective of the answers to these question after a deeper understanding of likelihood function.

Random Variable



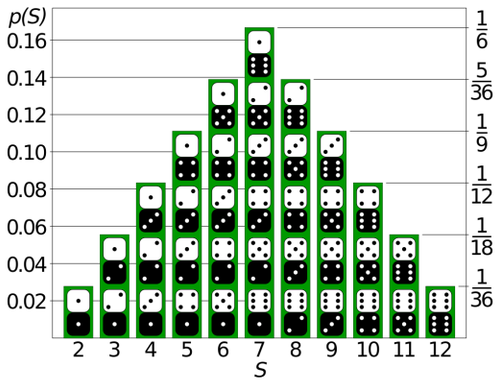
In our daily modelling activity, we are dealing with random variables.

A random variable is a numerical measure of the outcome of a probability experiment whose value is determined by chance.
More formally, it defined as a function that maps the outcomes of unpredictable processes to numerical quantities (labels), typically real numbers.

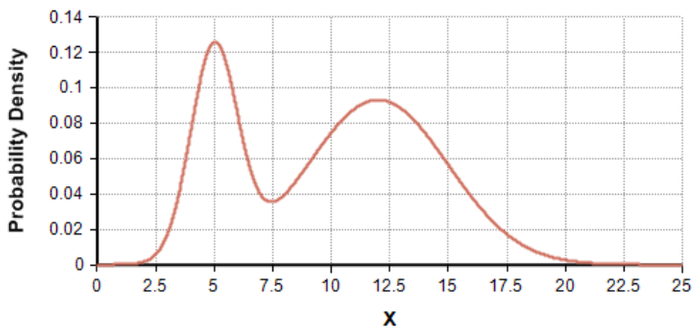
image source:
https://faculty.elgin.edu/dkernler/statistics/ch06/6-1.html

Probability distribution

Discrete case:



Continuous case



https://en.wikipedia.org/wiki/Probability_distribution

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
  - σ-algebra
- Measurable mapping
  - pushforward measure
- Integral
  - expectation

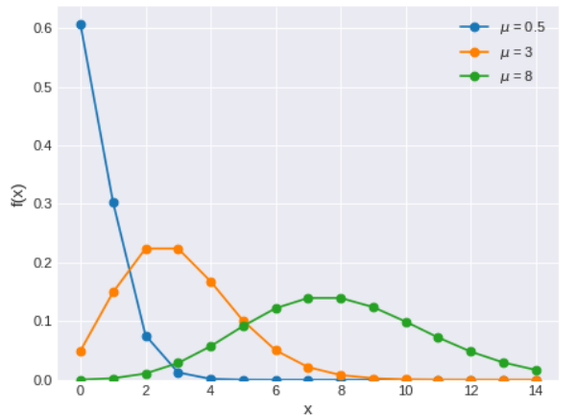
https://betanalpha.github.io/assets/case_studies/probability_theory.html

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

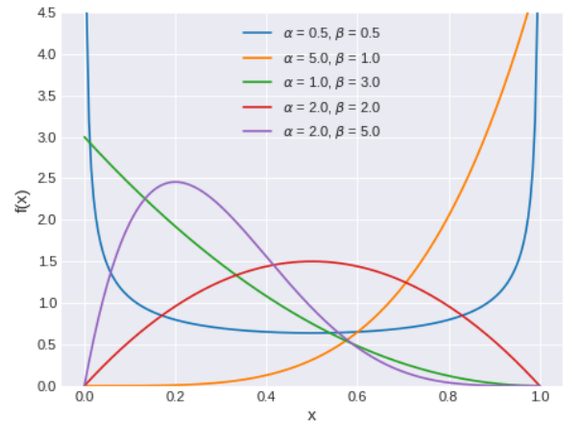
Function/mapping

Probability distribution

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



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



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 number of heads.

`Binomial(y|p,n)`

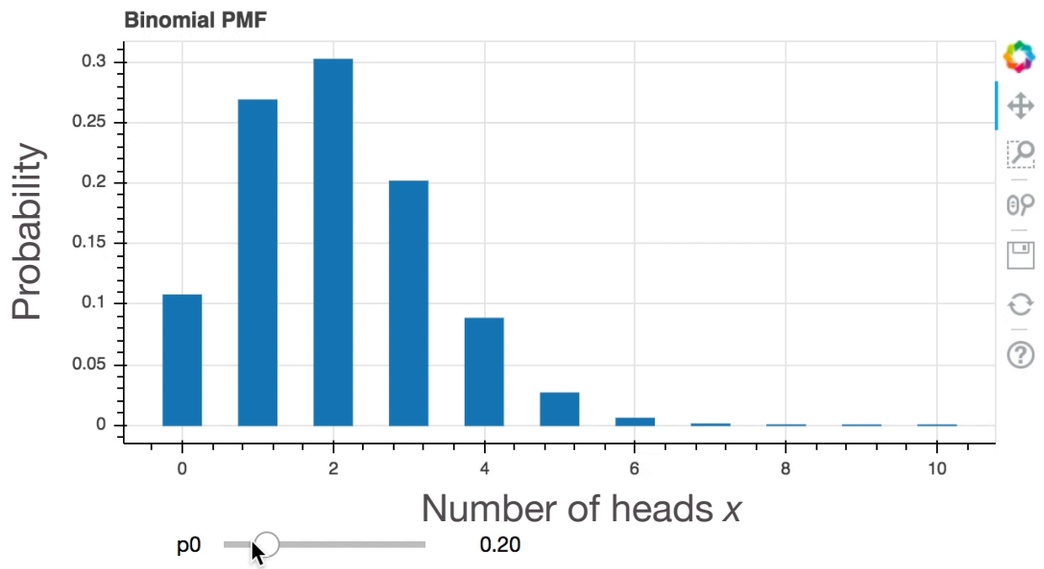
$$\pi(x \mid n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

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

Likelihood_visual_demo.ipynb

PMF and Likelihood

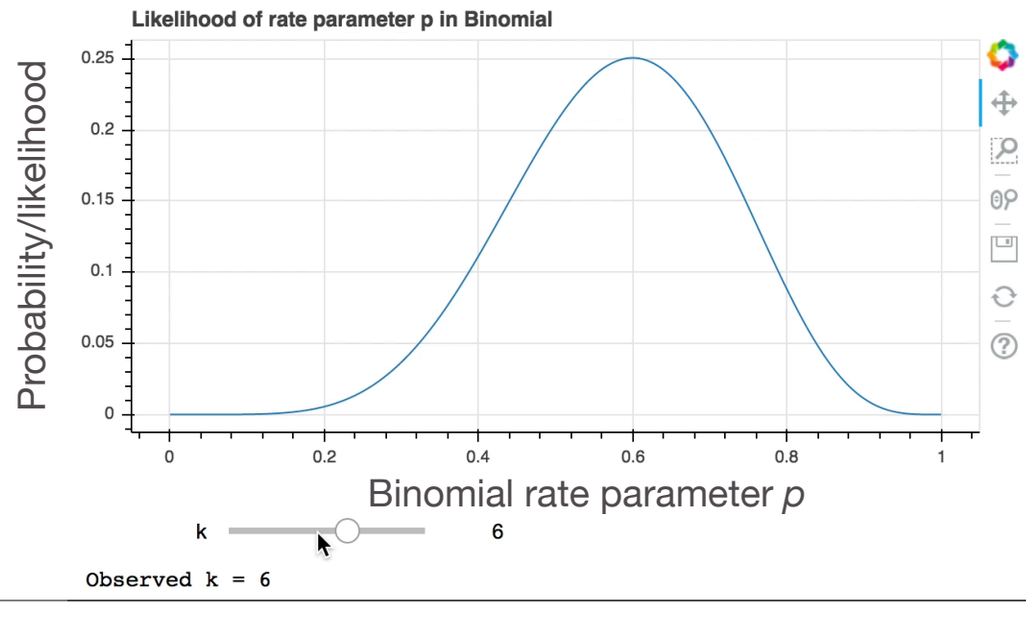
```
import scipy.stats as st
n0, p0 = 10, .2
x0 = np.arange(n0+1)
y0 = st.binom.pmf(x0, n0, p0)
```



Coin flip example: flip a coin 10 times, how many times it will come up head?
This is a typical Probability mass function

PMF and Likelihood

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

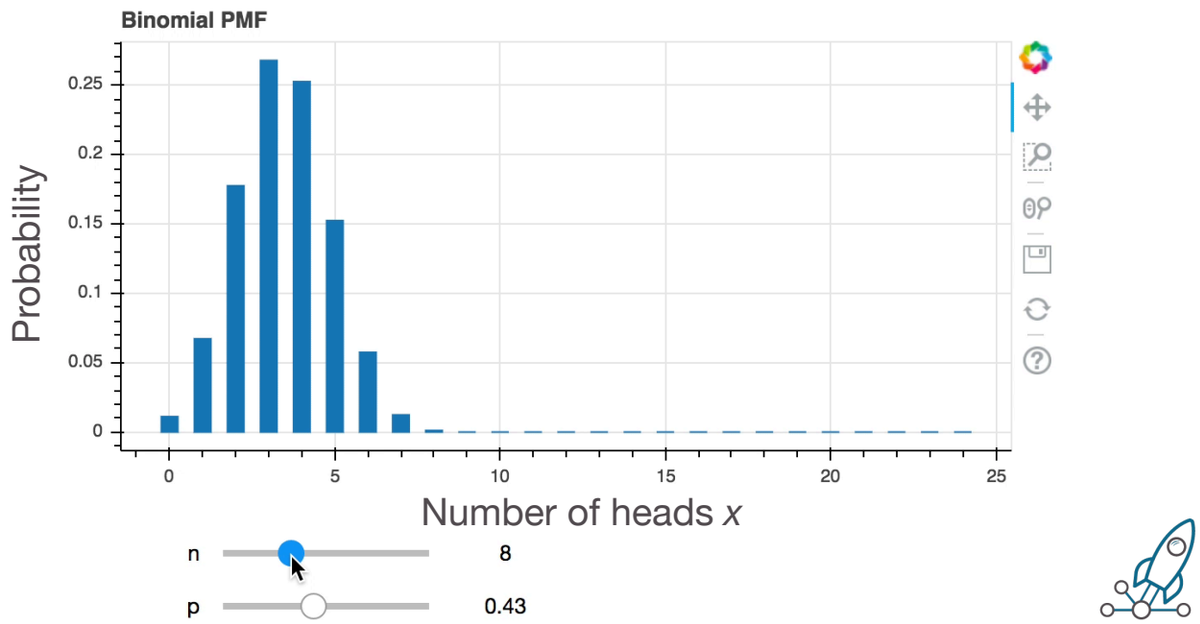



```
Observed k = 6
```

Now say we observed the number of head K and the total number of coin clip, with the binomial rate unknown. We got something like this, which is typically call a likelihood function.
The function is not a PDF - but does that really matters?
Observed that we are using the the same scipy function.
“likelihood function is the … function of the parameter equal to the density indexed by this parameter at the observed value.”

PMF and Likelihood

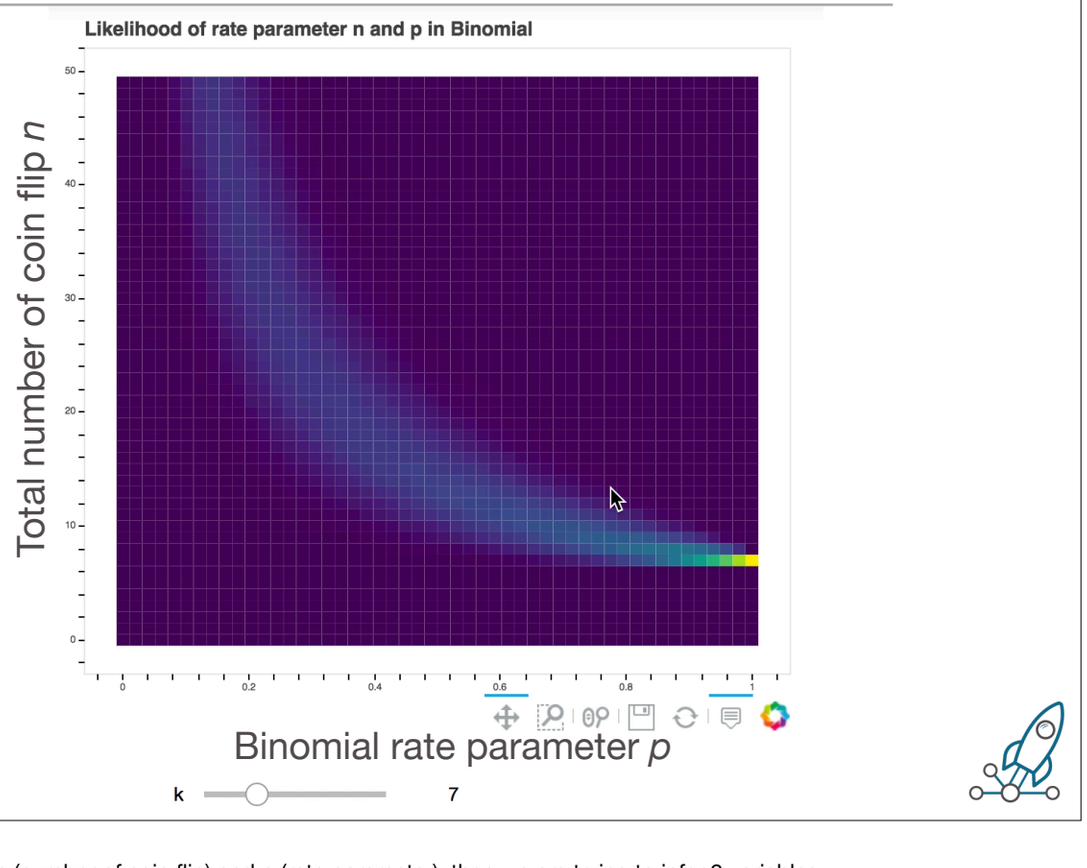
$$\pi(x \mid n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$



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 $\text{Binomial}(n, p)$ is a _family_ of distribution, once the n and p is fixed, it becomes a distribution of the outcome k.

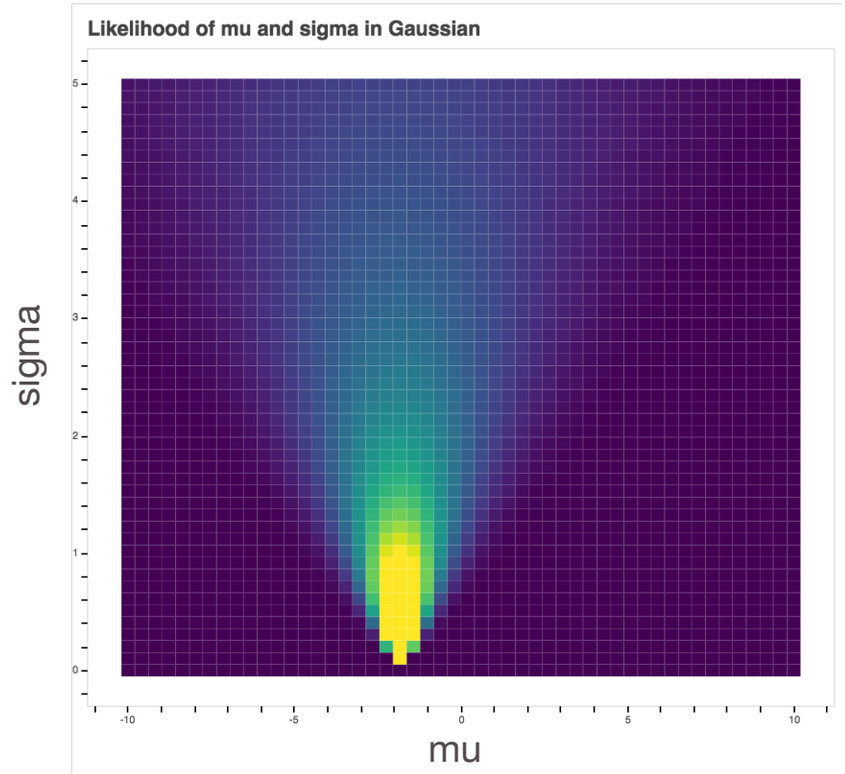
PMF and Likelihood



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.

Observe the points with 0 probability (impossible parameters), the banana shape of the space, the rather meaningless at local maximum.

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)`

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

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

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

From a programming or computation persecptive, 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. Conventionally, the observed number of heads from coin flip experiment is our Random Variable.

Combining probabilities

Summary of probabilities

Event	Probability
A	$P(A) \in [0,1]$
not A	$P(A^\complement) = 1 - P(A)$
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	$P(A \mid B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B\|A)P(A)}{P(B)}$

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

Sum rule and product rule

Sum rule:

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

Product rule:

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

https://en.wikipedia.org/wiki/Law_of_total_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
```

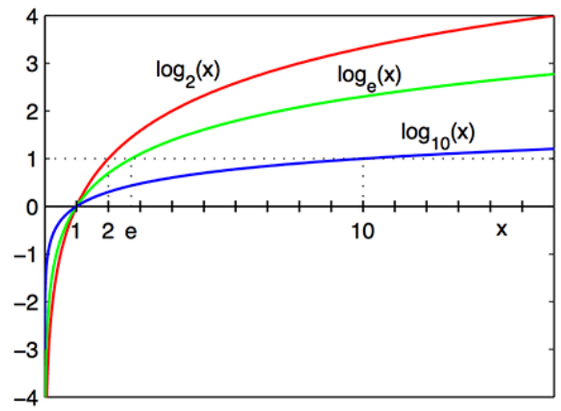
```
array([0.3     , 0.09    , 0.027   , 0.0081  , 0.00567 , 0.003969])
```

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

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

Now, imagine we have a new coin flip experiment which we flip the coin once and record the output. But this time we repeat this experiment 6 times. Say the binomial rate is .7, by the rule of combining different probability, the probability of observing exactly this sequence:

Log-Likelihood




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

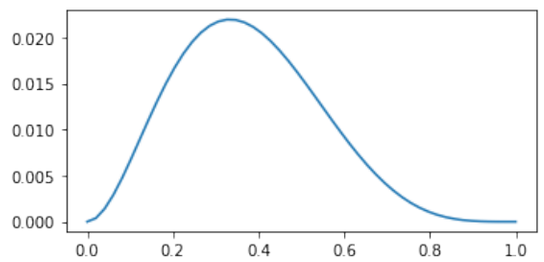
```
array([-1.2039728 , -2.40794561, -3.61191841, -4.81589122, -5.17256616,
       -5.52924111])
```

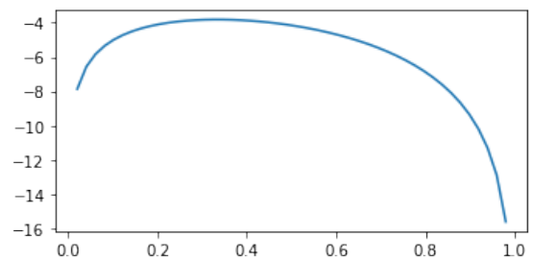
```
st.bernoulli.logpmf(y, p).cumsum()
```

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()
```

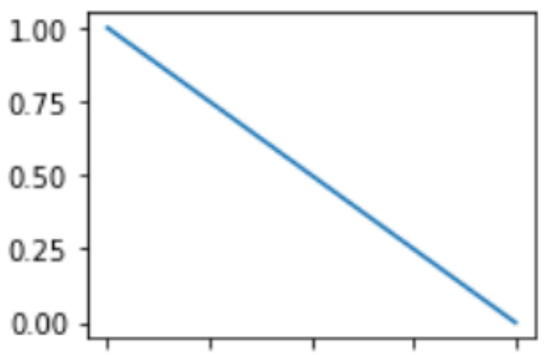




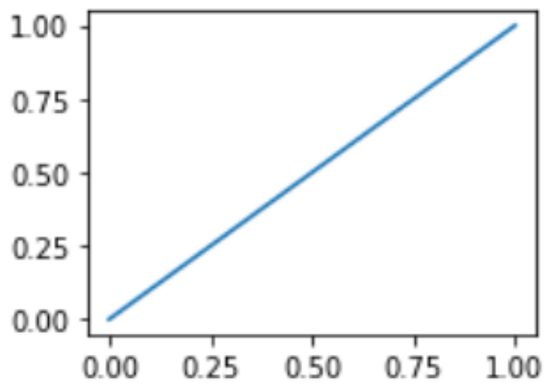
So what is the likelihood function for the parameter p?
In scipy we can evaluate at an array as input for the bernoulli PMF, we can have an expression to evaluate on (a grid) of binomial rate parameters (ie, the likelihood function for observing n repeat experiment).
But how do we get this function (with such shape)?

Likelihood of 1 coin flip

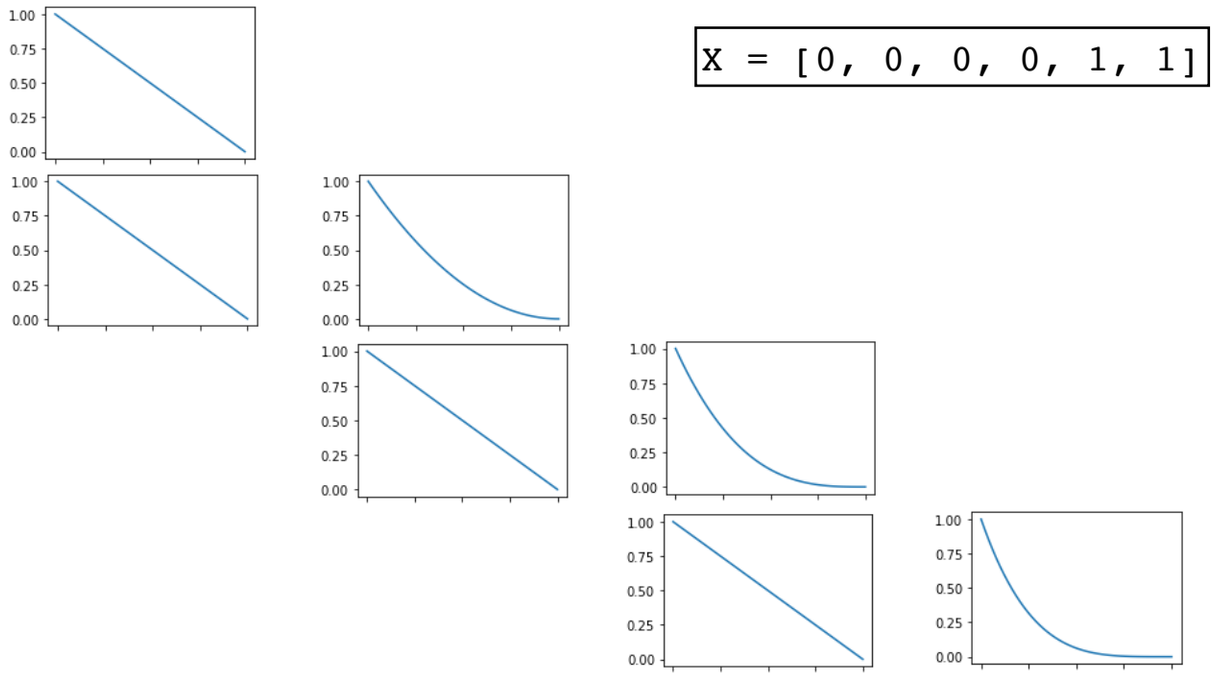
`X = [0]`



`X = [1]`



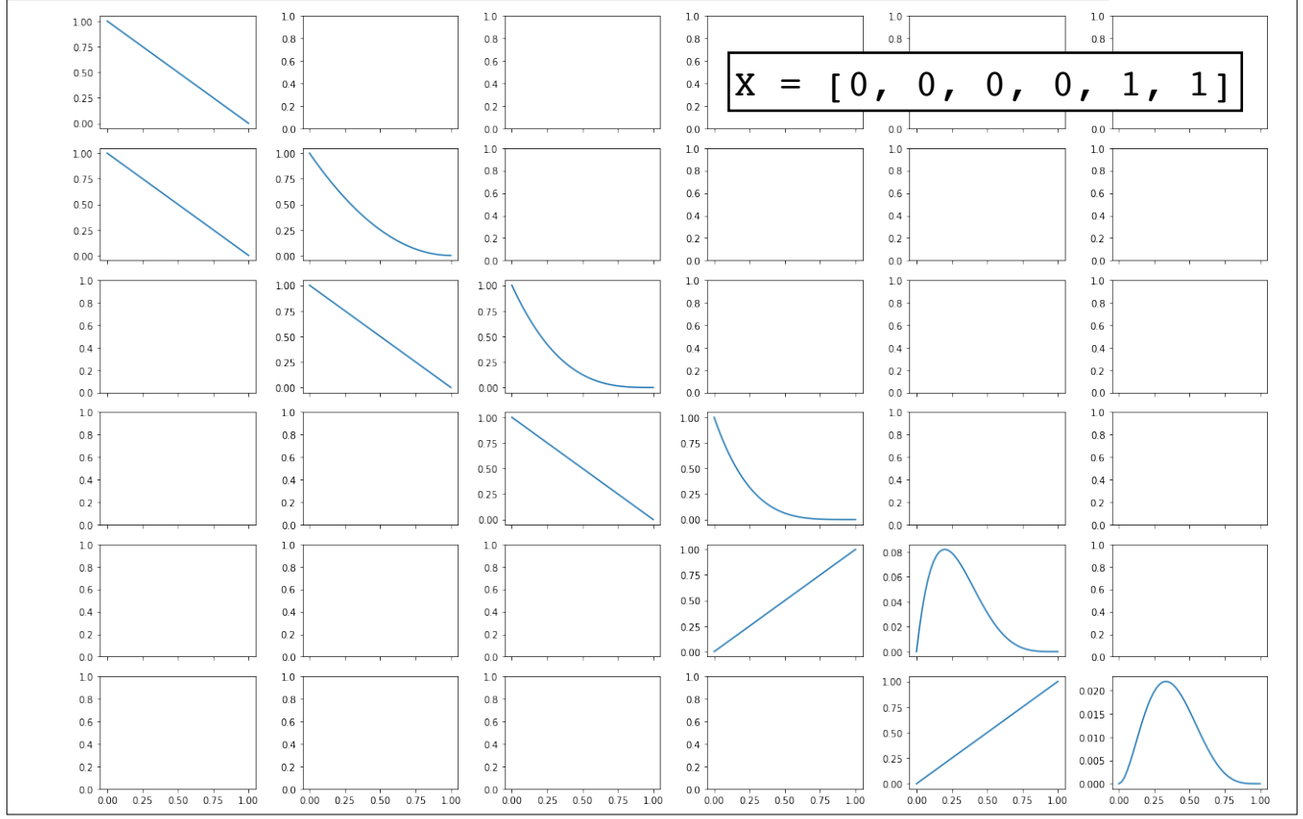
Update of Likelihood function




```
X = [0, 0, 0, 0, 1, 1]
```

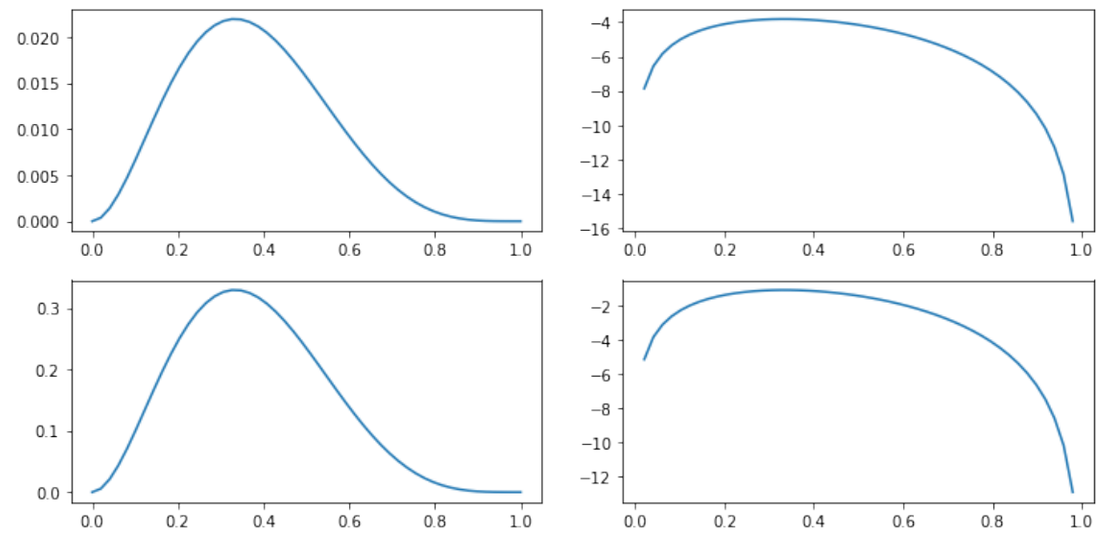
...

Update of Likelihood function



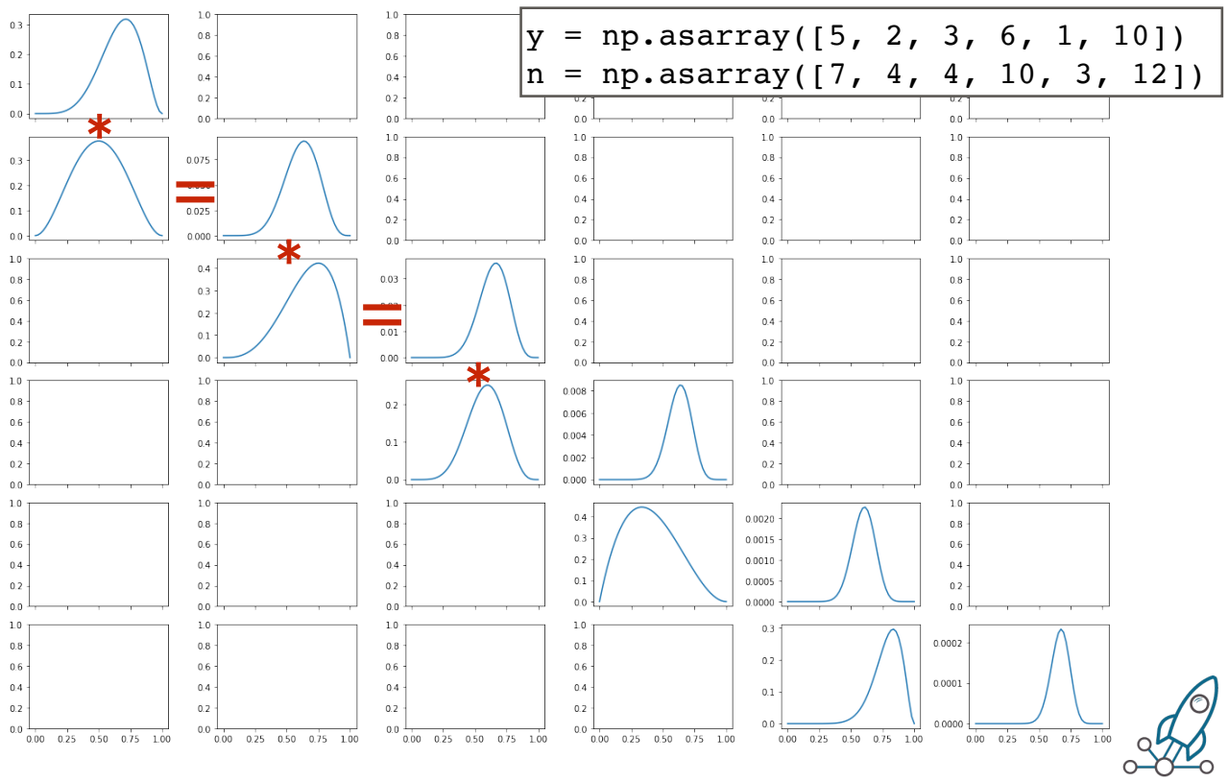
Likelihood function

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

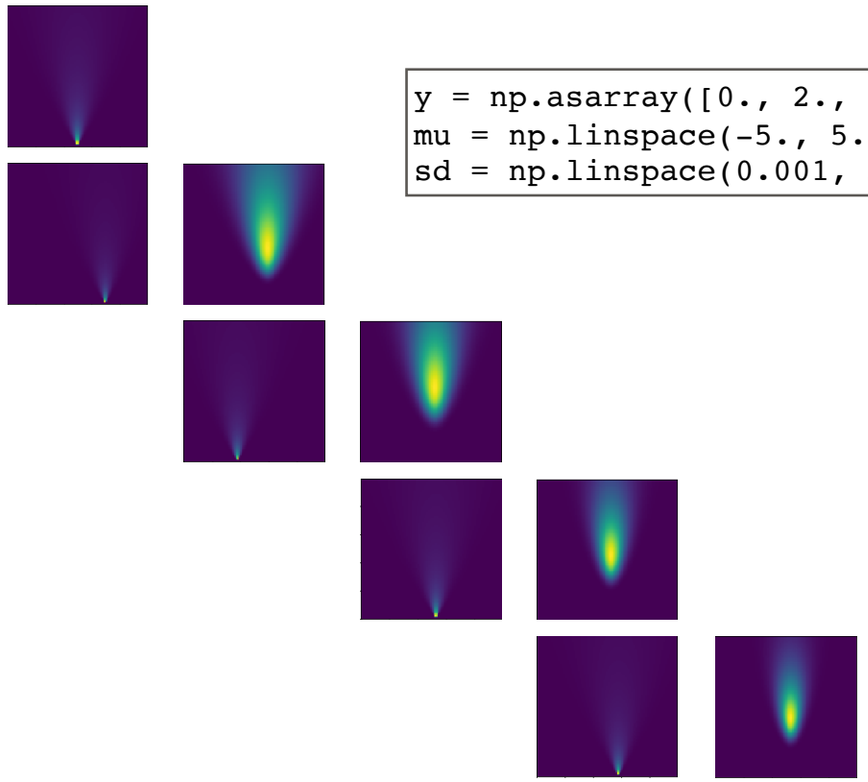


Update, or cumulating information

```
y = np.asarray([5, 2, 3, 6, 1, 10])
n = np.asarray([7, 4, 4, 10, 3, 12])
```



A Gaussian likelihood example

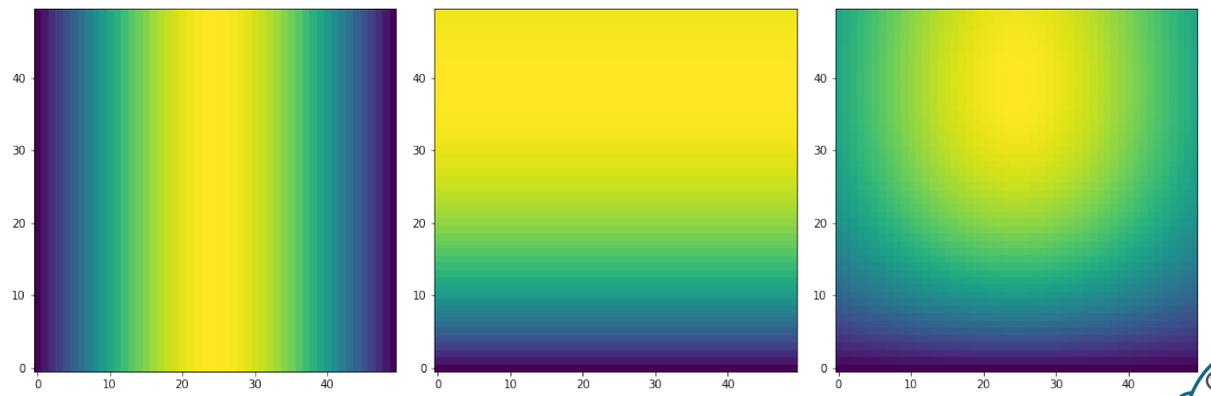


```
y = np.asarray([0., 2., -1.2, 0.3, .8])
mu = np.linspace(-5., 5., 100)
sd = np.linspace(0.001, 2.5, 100)
```

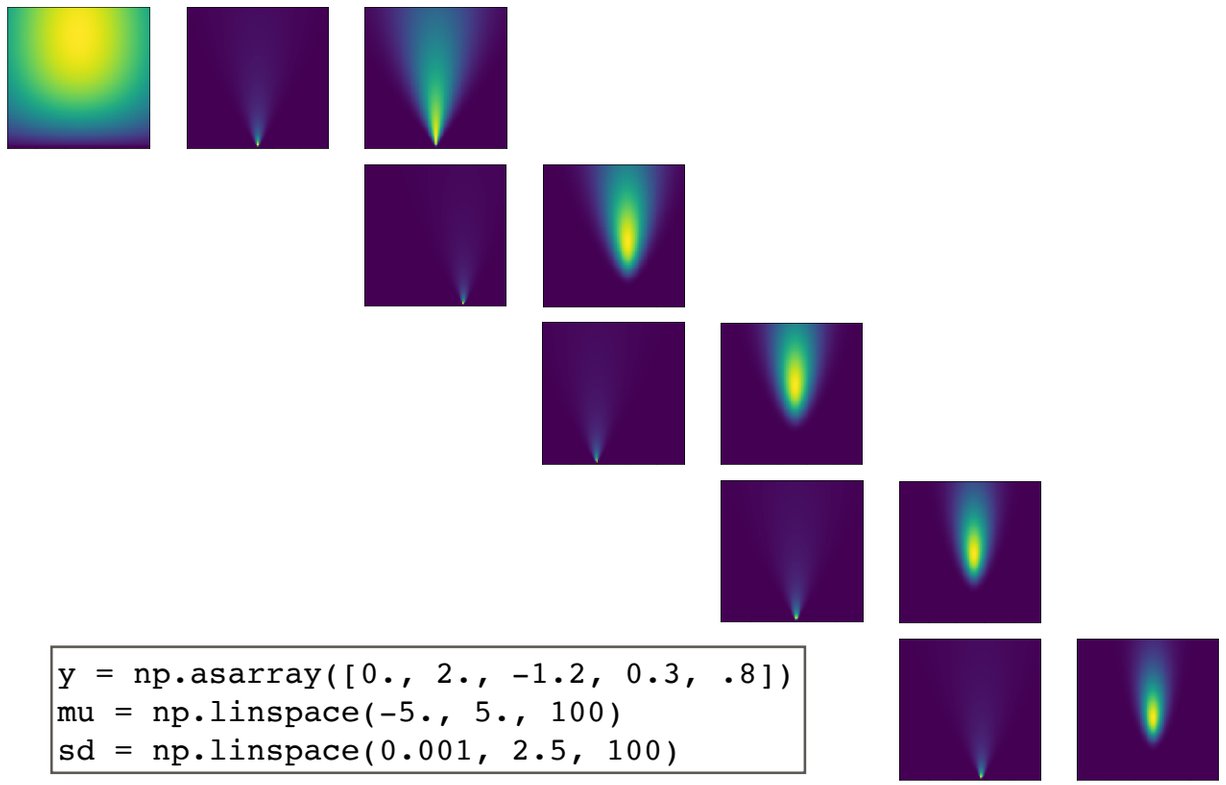
Priors

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)
```



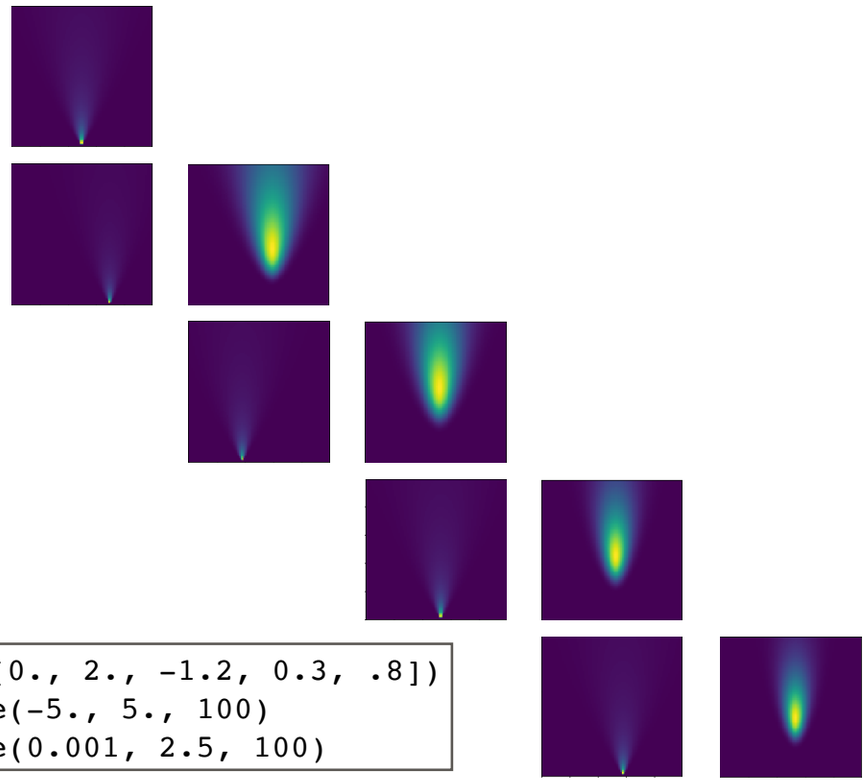
Update, or cumulating information



```
y = np.asarray([0., 2., -1.2, 0.3, .8])
mu = np.linspace(-5., 5., 100)
sd = np.linspace(0.001, 2.5, 100)
```

Still same sequence of observations.

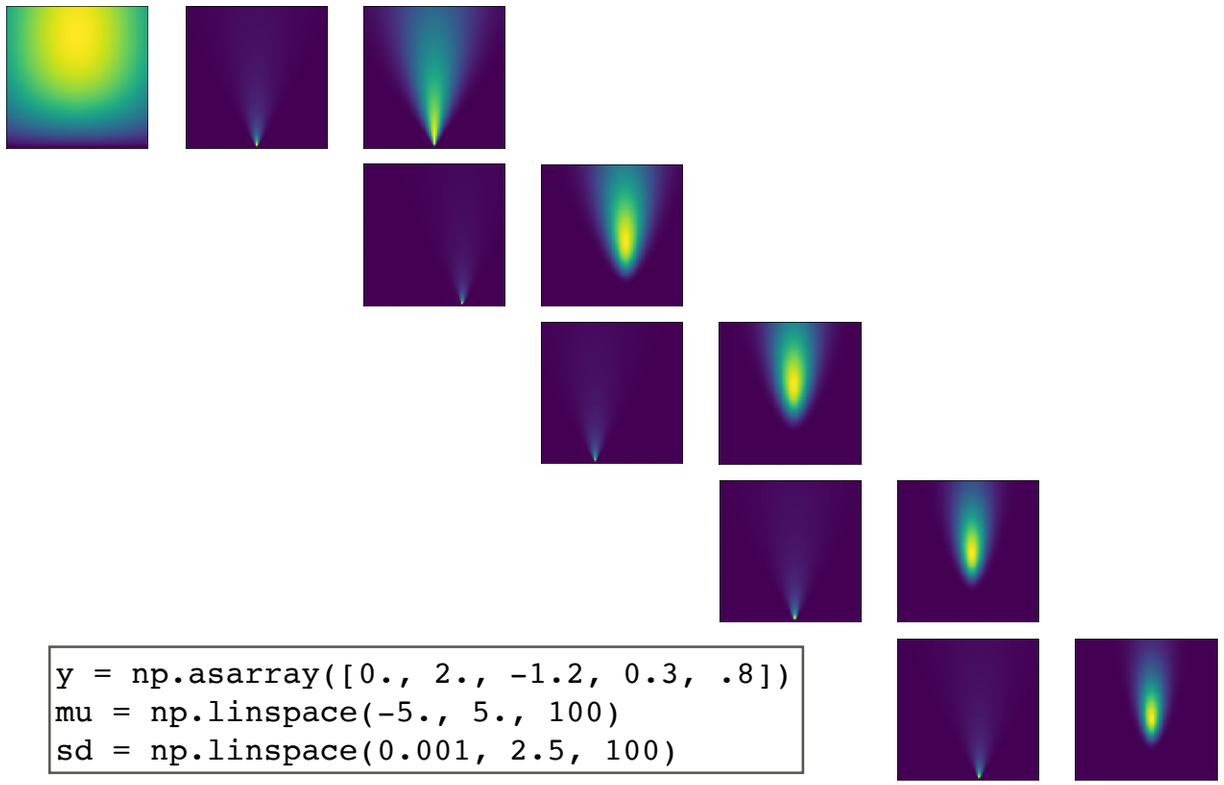
As comparison…



```
y = np.asarray([0., 2., -1.2, 0.3, .8])
mu = np.linspace(-5., 5., 100)
sd = np.linspace(0.001, 2.5, 100)
```

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

As comparison…



```
y = np.asarray([0., 2., -1.2, 0.3, .8])
mu = np.linspace(-5., 5., 100)
sd = np.linspace(0.001, 2.5, 100)
```

Still same sequence of observations.

Coin flip in PyMC3

$$
\begin{aligned}
k &= 7 \\
k &\sim \text{Binomial}(n, p) \\
n &\sim \text{DiscreteUniform}(0, 25) \\
p &\sim \text{Uniform}(0., 1.)
\end{aligned}
$$

```
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)}
```

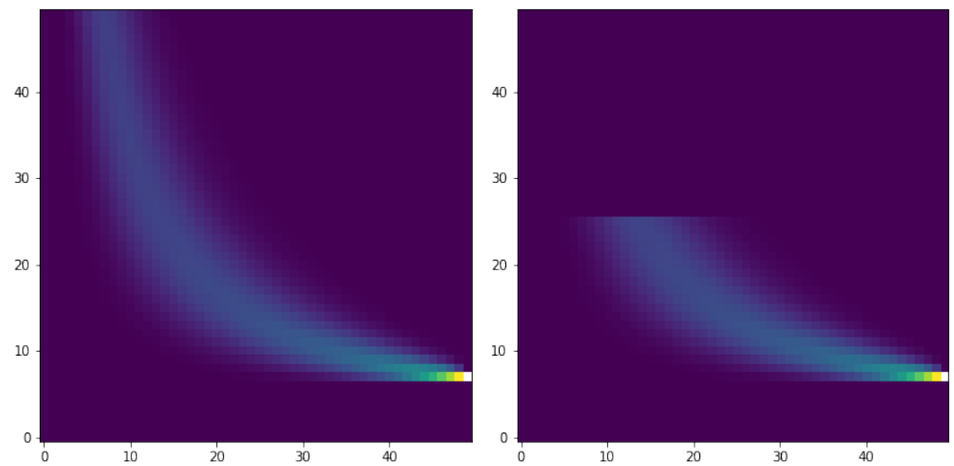
```
logp_m = m.logp
logp_y = y.logp
```

```
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.logp
```

$$f\colon \mathbb{R} \times \mathbb{N} \mapsto \mathbb{R}$$

```
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` $f\colon \mathbb{R} \times \mathbb{N} \mapsto \mathbb{R}$

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

```
n.distribution.logp
```

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

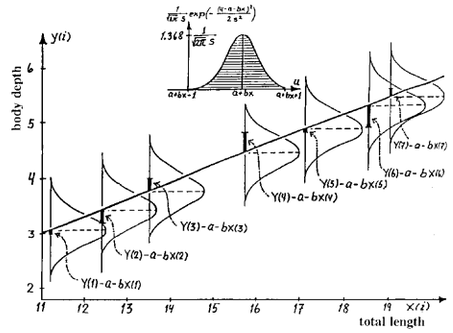
```
n.logpt
```

How PyMC3 use Theano

`m.logp` $f: \mathbb{R} \times \mathbb{N} \mapsto \mathbb{R}$

```
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



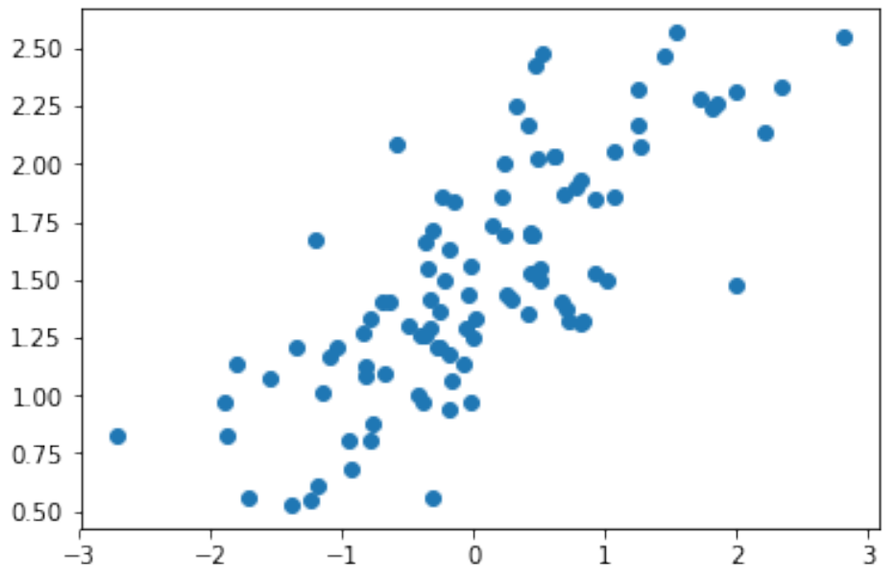
ANOVA

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

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1k} \\ 1 & x_{21} & \dots & x_{2k} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n1} & \dots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

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)

Reparameterization: regression

```
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)
```

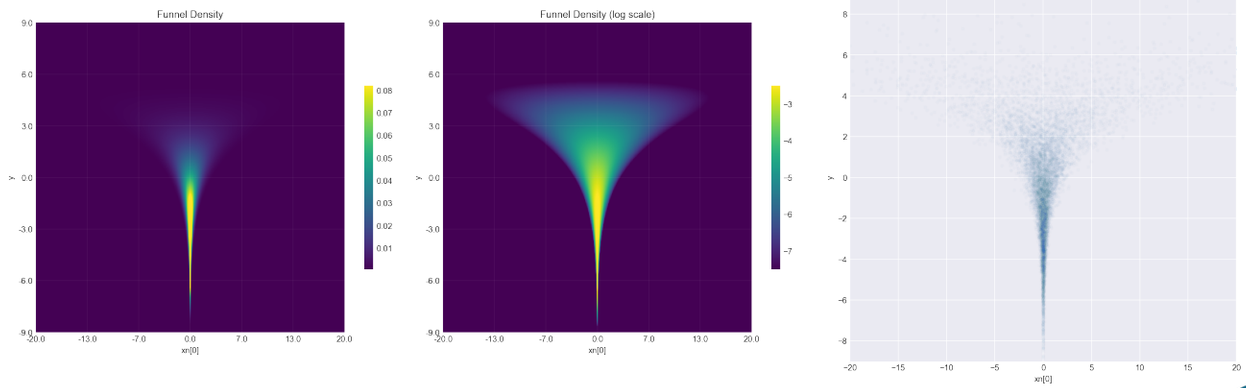
Reparameterization: regression

```
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)? You would find that the result is not the same compare to the other two, because this transformation actually change the volume.

Reparameterization: Neal's Funnel

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



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.

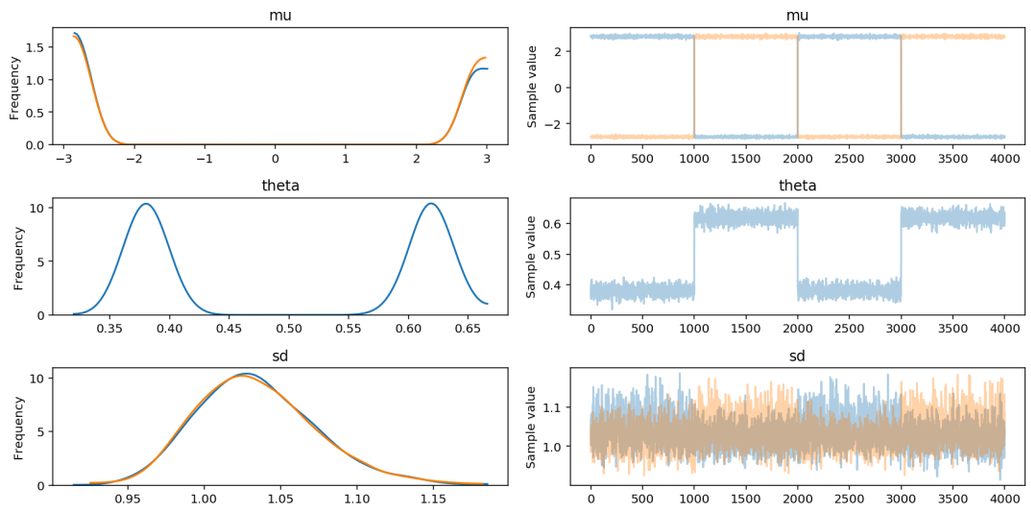
(Neal, 2003) defines a distribution that exemplifies the difficulties of sampling from some hierarchical models.

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

Mixture model likelihood

Mixture models are difficult to fit…

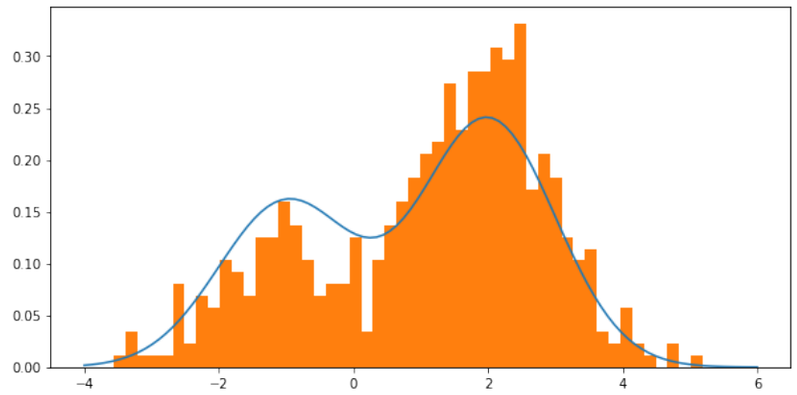
[Identifying Bayesian Mixture Models] by Michael Betancourt [PyMC3 port]



Mixture models are difficult to sample from - problem of label switching.

Mixture model reparameterized

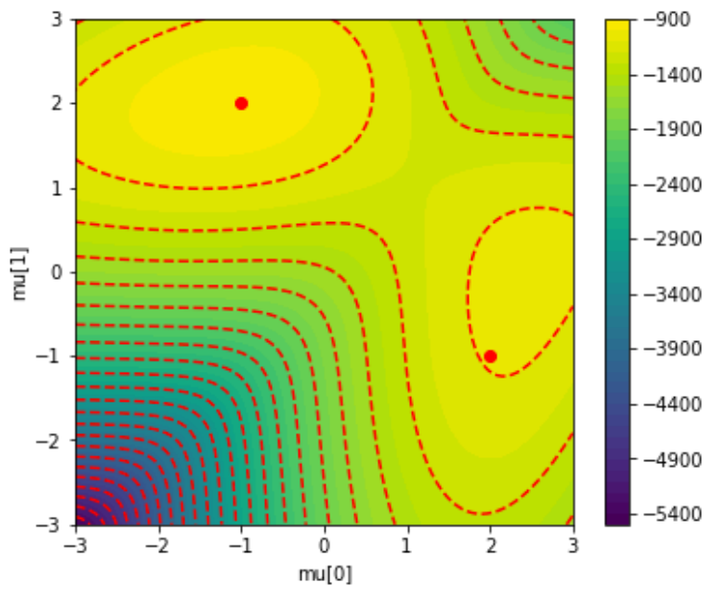
```
w0 = np.array([.3, .7])
mu0 = np.array([-1, 2])
sd0 = 1.
x = pm.NormalMixture.dist(w=w0, mu=mu0,
sd=sd0).random(size=500)
```



Normal_mixture_logp.ipynb

Mixture model reparameterized

```
with pm.Model() as m0:
    mu = pm.Normal('mu', 0., 5., shape=2)
    pm.NormalMixture('y',
                     w=w0,
                     mu=mu,
                     sd=theano.shared(1.),
                     observed=x)
```



Mixture model reparameterized

Non-exchangeable Priors

Enforcing an Ordering

```
import pymc3.distributions.transforms as tr
with pm.Model() as mixture_model:
    …
    mu = pm.Normal('mu', 0, 2, shape=2,
                   transform=tr.ordered,
                   testval=[0.1, 0.2])
    …
```

There are a couple of ways to avoid the problem of label switching: Non-exchangeable priors, enforcing an ordering, etc.

Mixture model reparameterized

Enforcing an Ordering

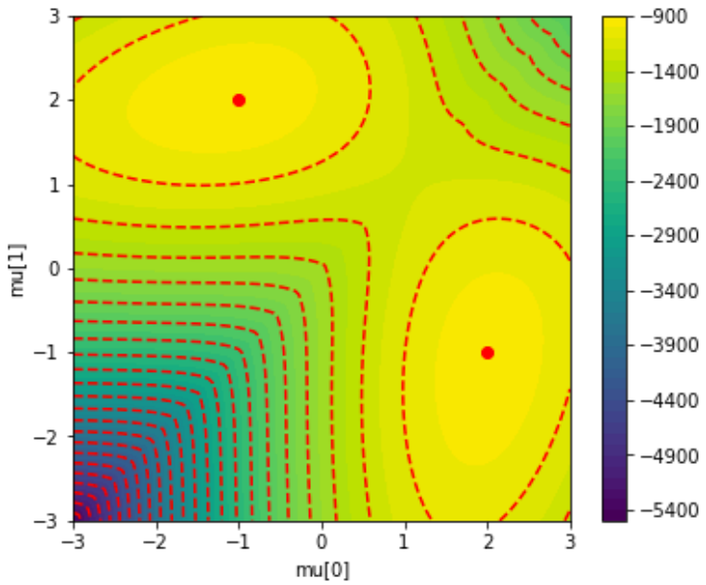
```
with pm.Model() as mixture_model:
    …
    mu = pm.Normal('mu', 0, 2, shape=2,
                   transform=tr.ordered,
                   testval=[0.1, 0.2])
    …
```

```
with pm.Model() as mixture_model_:
    …
    mu_ = pm.Normal('mu_', 0, 2, shape=2)
    mu = tt.sort(mu_)
    …
```

What is the differences between ordering and sorting the RV?

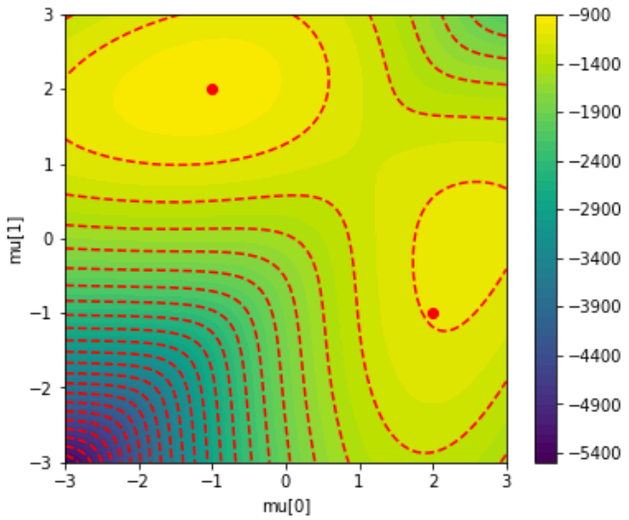
Mixture model reparameterized

```
with pm.Model() as m1:
    mu = pm.Normal('mu', 0., 5., shape=2)
    mu_ = tt.sort(mu)
    pm.NormalMixture('y',
                     w=w0,
                     mu=mu_,
                     sd=theano.shared(1.),
                     observed=x)
```

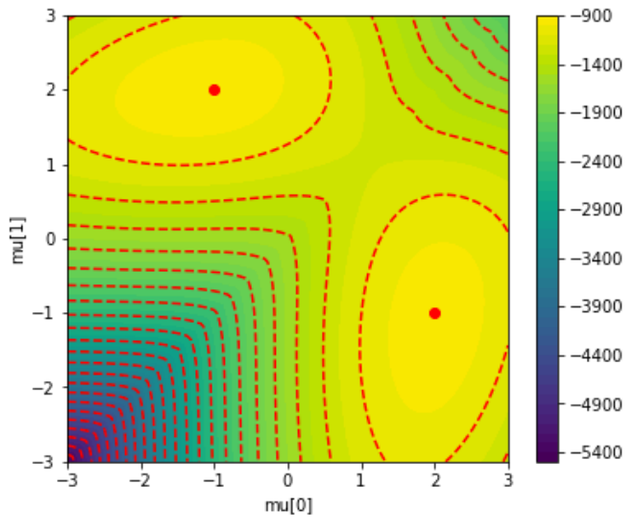


Mixture model reparameterized

Origin model



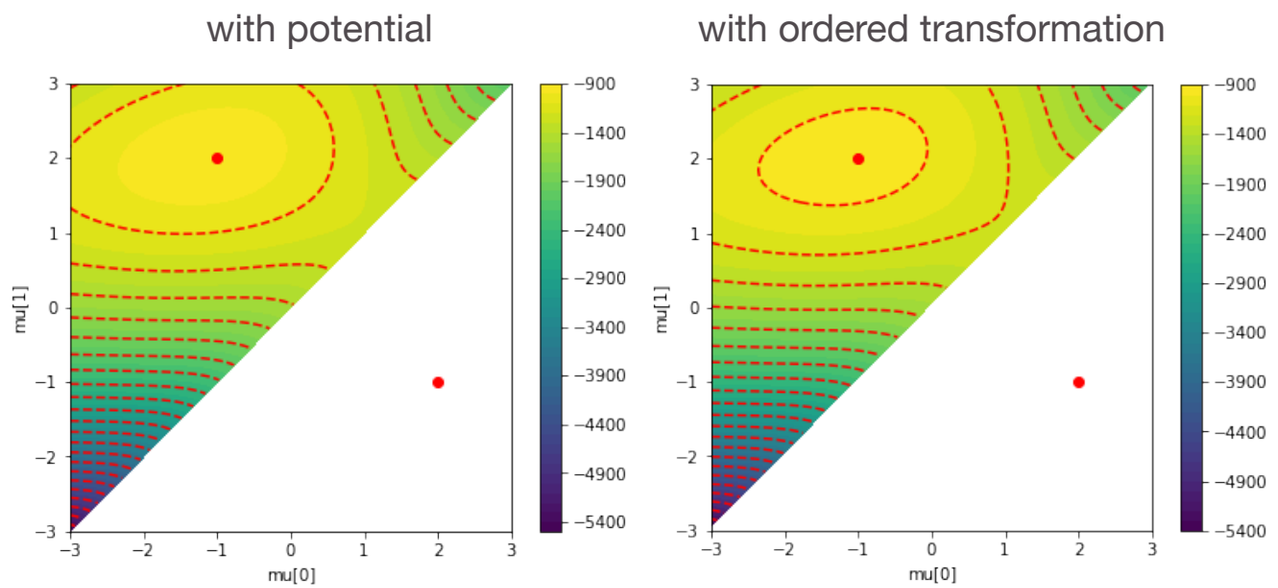
with tt.sort



Mixture model reparameterized

```
with pm.Model() as m2:
    mu = pm.Normal('mu', 0., 5., shape=2)
    pm.NormalMixture('y',
                     w=w0,
                     mu=mu,
                     sd=theano.shared(1.),
                     observed=x)
    pm.Potential('order', tt.switch(mu[1]<mu[0], -np.inf, 0))


with pm.Model() as m3:
    mu = pm.Normal('mu', 0., 5.,
                   shape=2,
                   transform=tr.ordered,
                   testval=np.array([0., 1.]))
    pm.NormalMixture('y',
                     w=w0,
                     mu=mu,
                     sd=theano.shared(1.),
                     observed=x)
```

Mixture model reparameterized



Parameterization

There are many ways to write down the same model, but reparameterization always change the model logp:

- 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/

Recap:

The distinction between probability and likelihood is probably unnecessary.

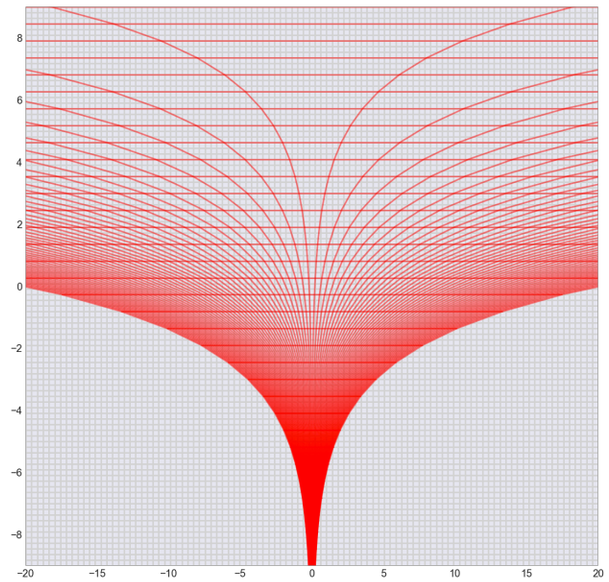
We “glue” random variables together in a conditional network to create a mapping that is the (log-) likelihood function.

- 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.

Different coordinate systems



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.

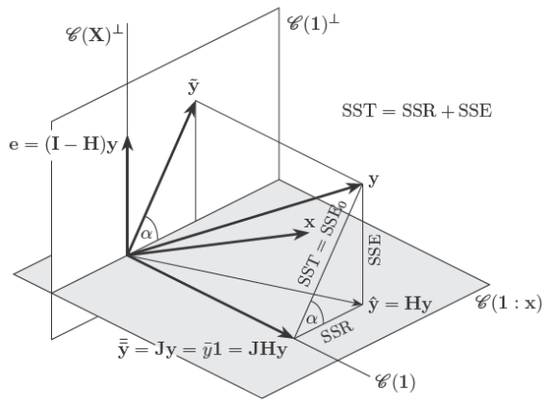
L1 and L2 regularization

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.

Recap:

The distinction between probability and likelihood is probably unnecessary.

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)

Plotting a slice of the likelihood “space” is very useful for debugging.

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.

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 $y(t) = Asin(\omega t) + Bcos(\omega t) + noise$ where A and B are drawn as Normal random variables.
2. Draw A and B as Normal random variables, and fit $y(t) = Csin(\omega t + \phi)$ 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:

Timeserie_model.ipynb

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)

Confusion about likelihood is unfortunate from a partial focus of this concept. (eg, see https://xianblog.wordpress.com/2018/06/07/are-there-a-frequentist-and-a-bayesian-likelihoods/)

Thanks!