PRECISION WORKSHOP

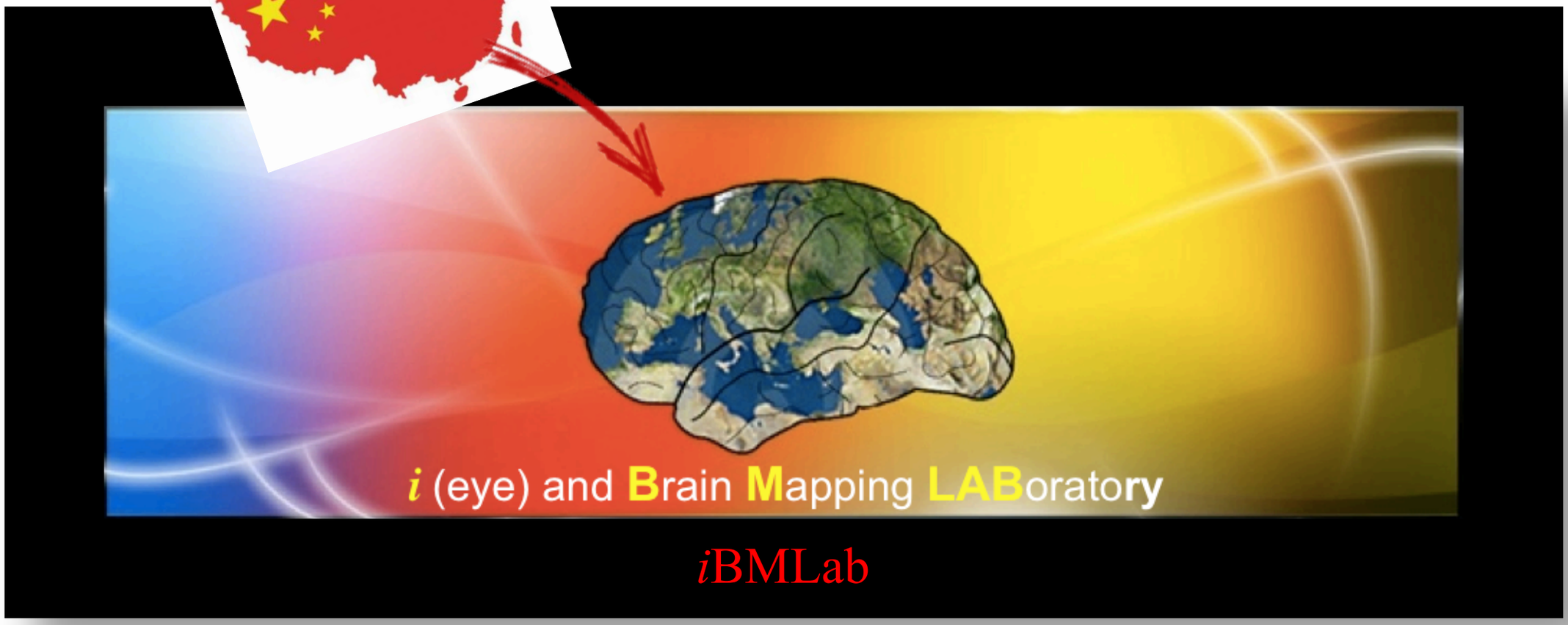
Practical Bayesian Computation with PyMC3

Junpeng Lao

May 2018 @ CEAi

Powered by 

About Me



University of Glasgow

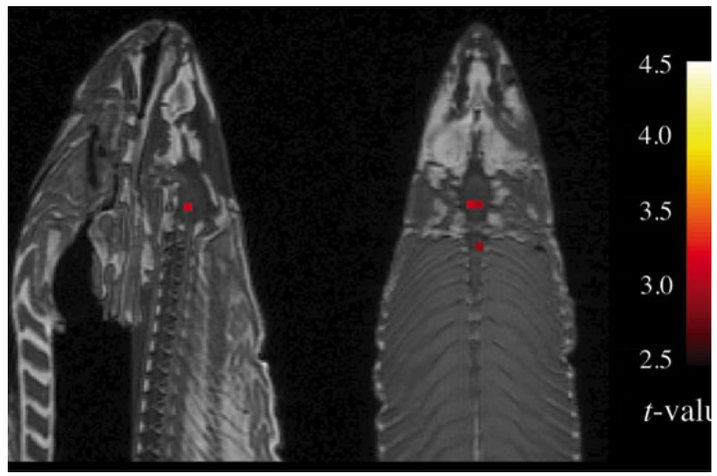
UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

About Me

Eye-tracking

EEG

fMRI



PyMC3

https://github.com/junpenglao/
http://junpenglao.xyz

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	H	5	36	4h
☑ ValueError: Bad initial energy: inf. The model might be misspecified error replicating Multilevel Regression and Poststratification with PyMC3 notebook	Questions	G	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	H	8	35	22h

https://discourse.pymc.io/

Prerequisites:

Branch: **master** ▾ **PrecisionWorkshop1_Prep** / **notebooks** /

Create new file | **Upload files** | **Find file** | **History**

This branch is 131 commits ahead of fonnesbeck:master. Pull request Compare

jurah Merge remote-tracking branch 'origin/master' into typos-201804 Latest commit `300113e` 12 hours ago

..		
images	commited face model diagrams for notebook 2a	2 months ago
0. Introduction to the material.ipynb	minor reformulation changes to introductory notebooks	a month ago
1a. Introduction to PyMC3.ipynb	fix some typos in text of 1a	22 days ago
1b. Case study in PyMC3 - Coal m...	overhauled initial notebooks 0 and 1	2 months ago
1c. Analytical solution: Beta Binom...	overhauled initial notebooks 0 and 1	2 months ago
1d. Analytical solution: Normal-No...	overhauled initial notebooks 0 and 1	2 months ago
2a. Model Building with PyMC3.ip...	added joint plot of eye positions to make it clear how degenerate sol...	a month ago
2b. Tagging name tokens.ipynb	removed trueskill notebooks and split face/tagging case study	2 months ago
3a. Sampling algorithms for Bayes...	Add reference to 3a notebook to add more info about law of large numbers	a month ago
3b. Importance sampling.ipynb	added Cauchy proposal distribution to 3b	a month ago
3c. Rejection sampling.ipynb	incorporated feedback into notebooks 3b-3c	a month ago
3d. MC in more than one dimensio...	fix some typos	a month ago
3e. Introduction to MCMC.ipynb	typos in text of 3e	22 days ago
3f. The Metropolis sampler.ipynb	typo in 3f	21 days ago
3g. Case study: reaction times of ...	typos in 3g	22 days ago
3g.1 Case study: reaction times of...	added solution to notebook 3g as 3g.1, added more text to notebook 3i	22 days ago

Objective of the workshop:

Understand what is being computed Bayesian modelling

Learn about ways to validate your model in PyMC3

Learn about different ways to perform proper inference

Evaluation of model fitting and model comparisons

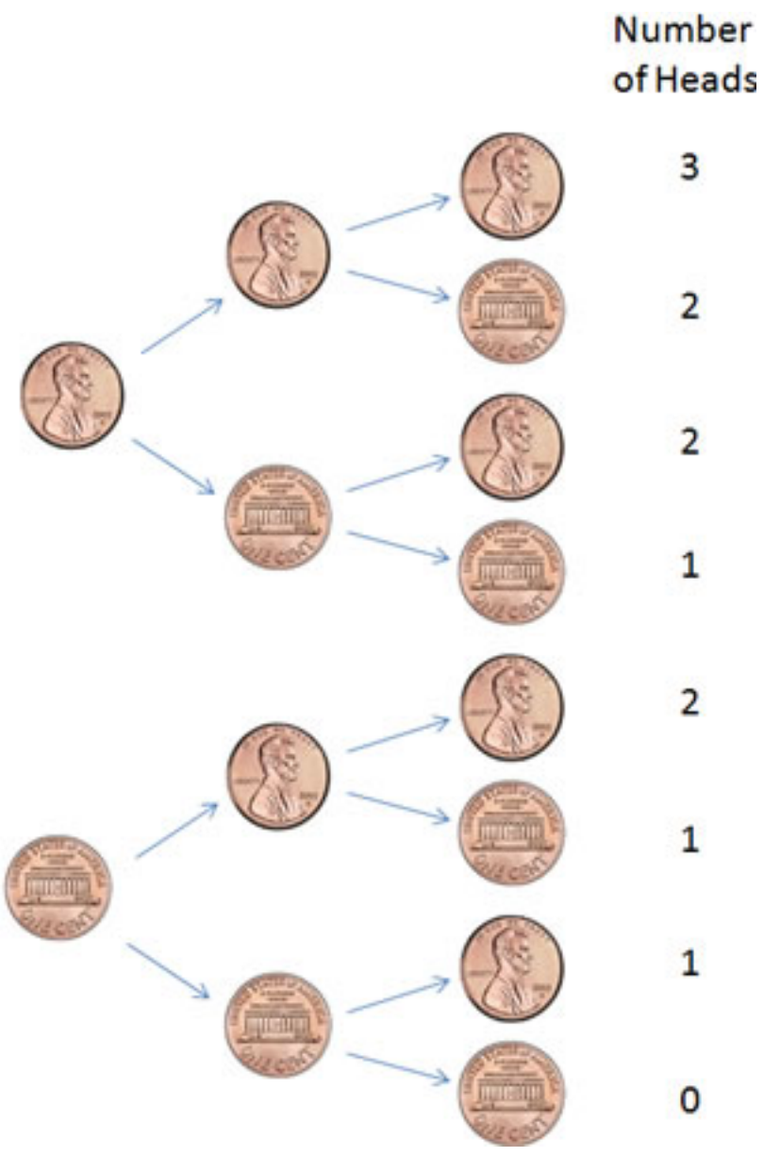
Probability theory

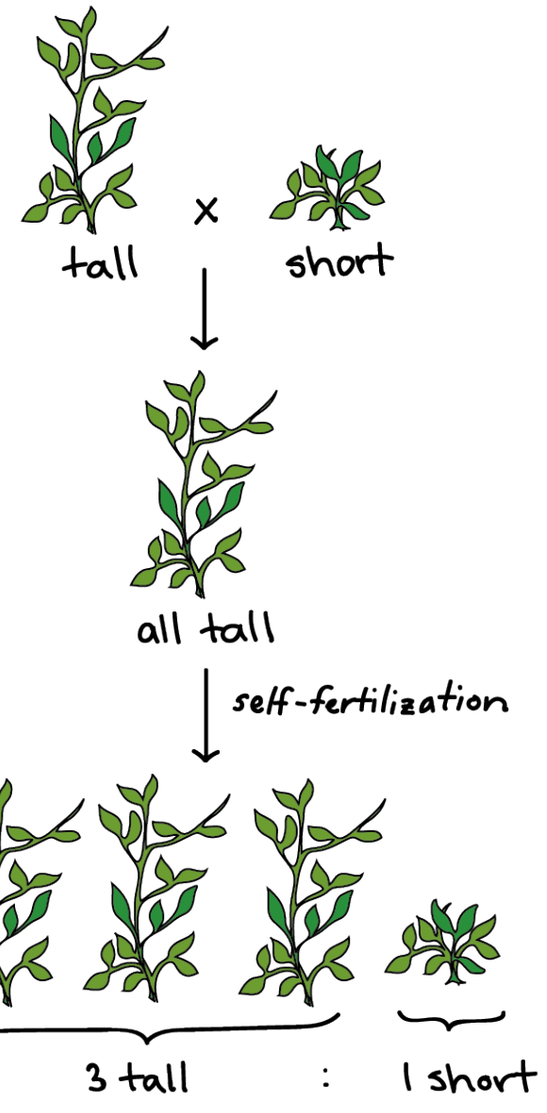
Measure-Theoretical Foundations of Probability Theory

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

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

Random Variable

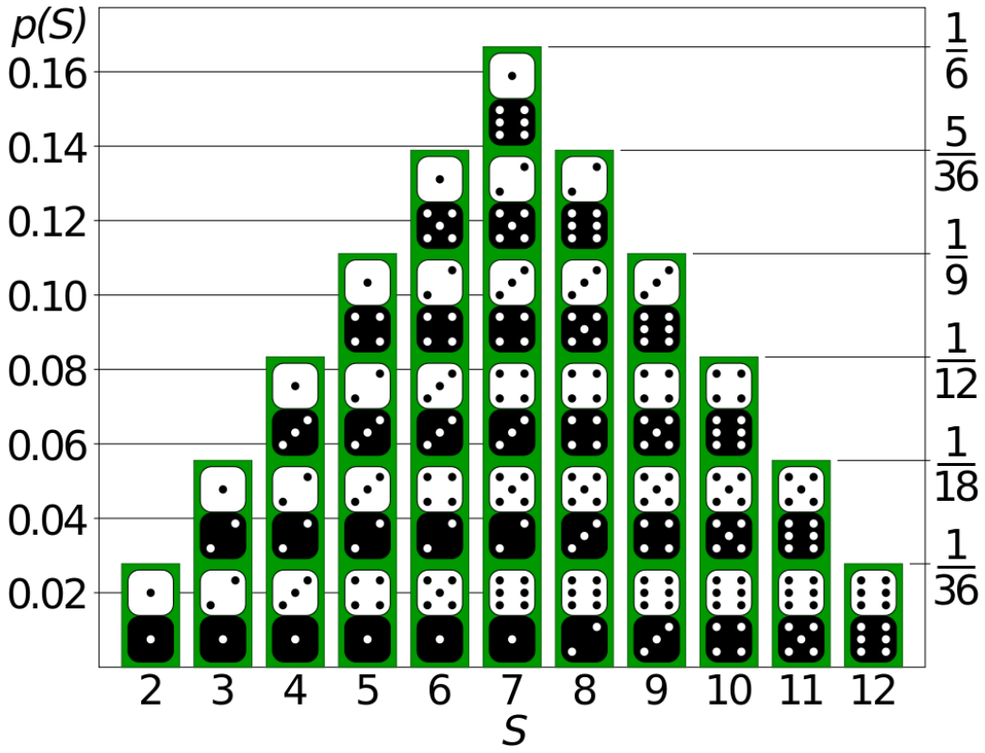




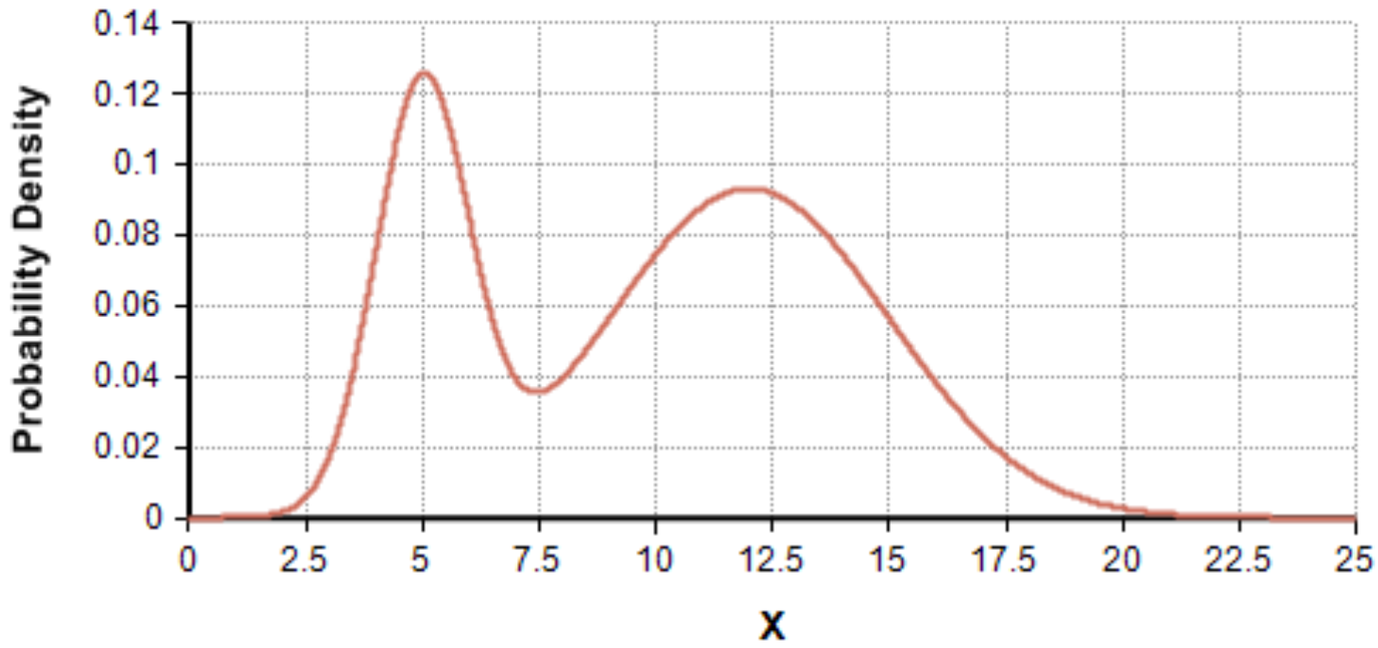
Mendel's actual numbers:
787 tall : 277 short (2.84:1).

Probability distribution

Discrete case:



Continuous case

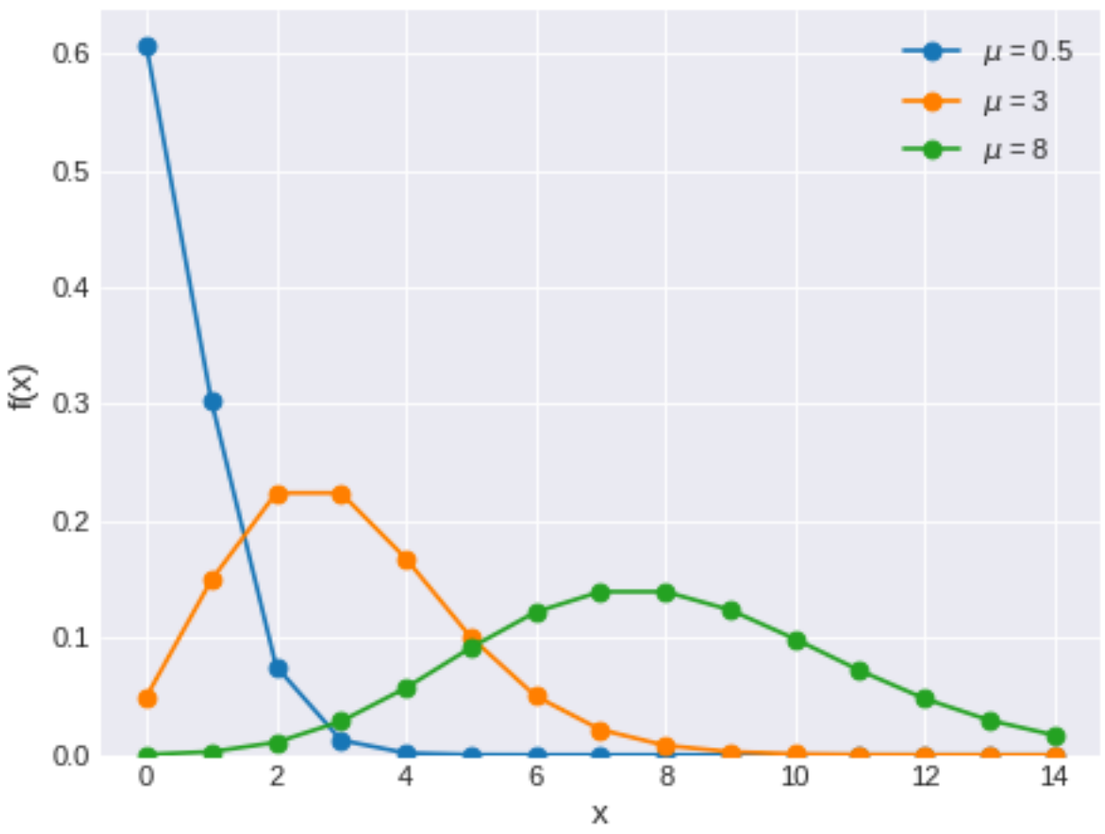


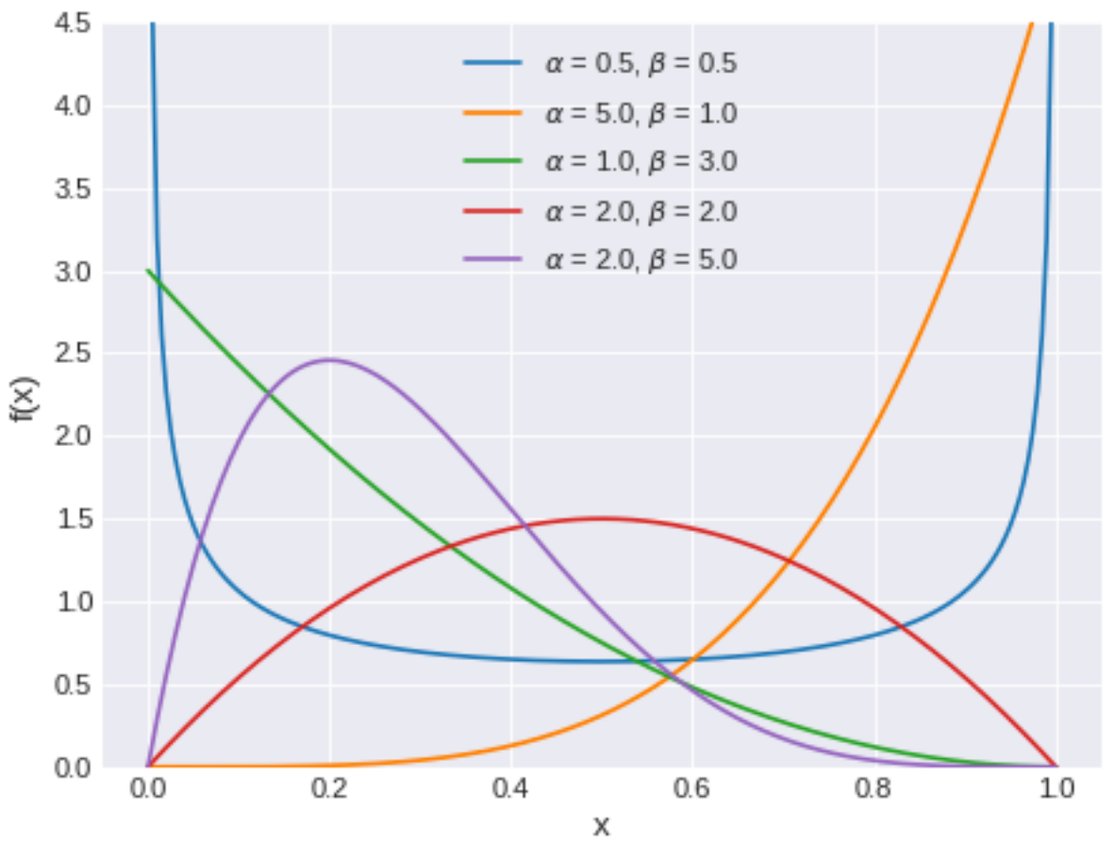
https://en.wikipedia.org/wiki/Probability_distribution

Probability distribution

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

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





Random Variable

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

“Bob is flipping a fair coin and he stop when he observed 5 heads”

X = the number of total coin flips

```
p = .5
flip_sq = st.bernoulli.rvs(p, size=5)  # first 5 flips
while flip_sq.sum() != 5:
    flip_sq = np.append(flip_sq, st.bernoulli.rvs(p))
print(flip_sq)
```

Code1 - Forward_Random.ipynb

Random Variable

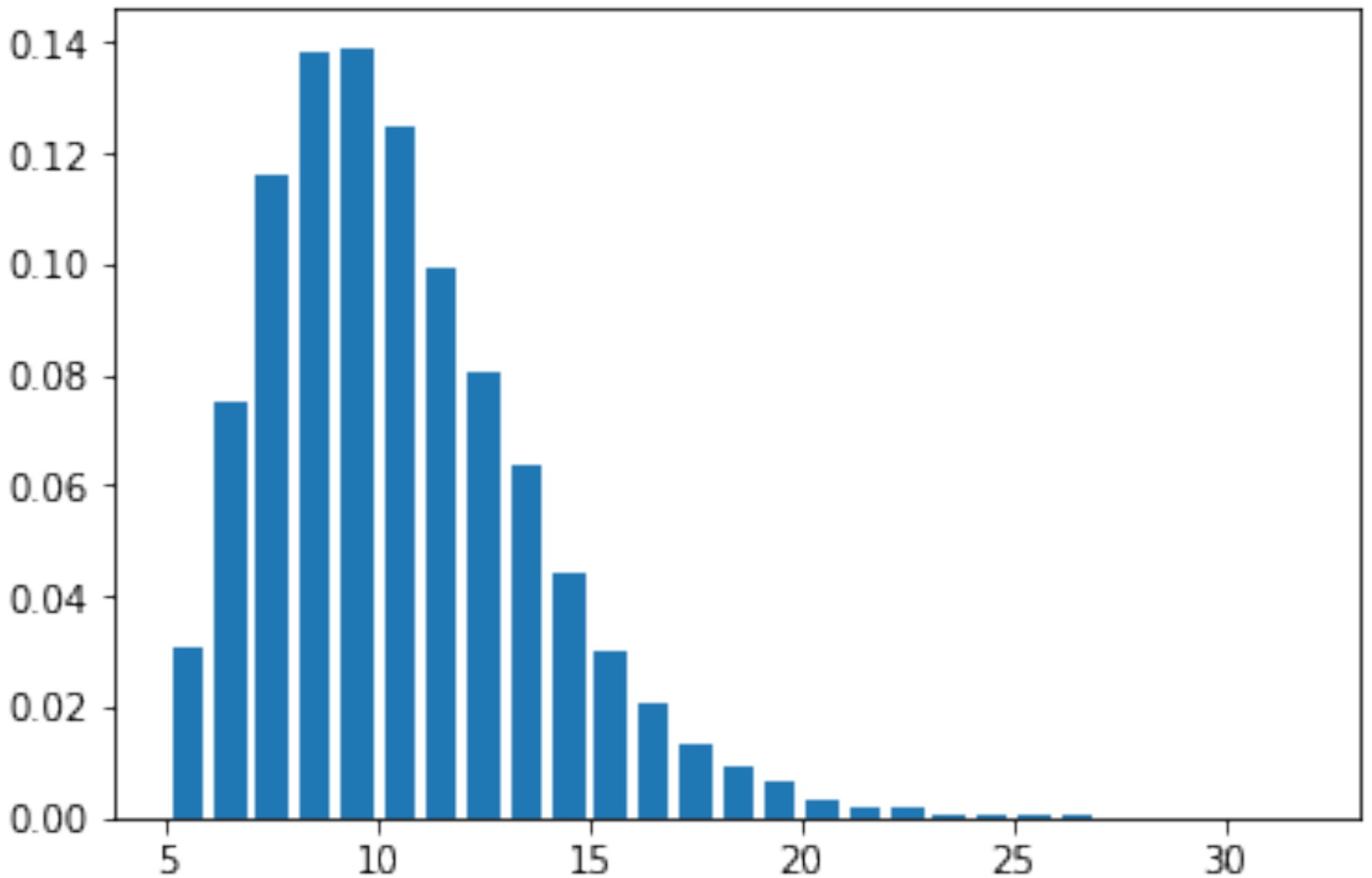
"Bob is flipping a fair coin and he stop when he observed 5 heads"

```
p = .5
flip_sq = st.bernoulli.rvs(p, size=5)  # first 5 flips
while flip_sq.sum() != 5:
    flip_sq = np.append(flip_sq, st.bernoulli.rvs(p))
print(flip_sq)
```

X = the number of total coin flips

```
 7 <-- [0 1 1 1 0 1 1]
 9 <-- [1 0 1 1 0 0 0 1 1]
10 <-- [0 0 1 0 1 0 0 1 1 1]
 9 <-- [1 0 0 0 0 1 1 1 1]
11 <-- [0 1 1 1 0 0 1 0 0 0 1]
10 <-- [1 0 0 1 0 0 1 1 0 1]
 6 <-- [0 1 1 1 1 1]
15 <-- [0 0 0 1 0 1 0 1 0 0 0 0 1 0 1]
11 <-- [0 0 0 1 0 1 1 1 0 0 1]
11 <-- [1 0 0 0 0 1 1 1 0 0 1]
```

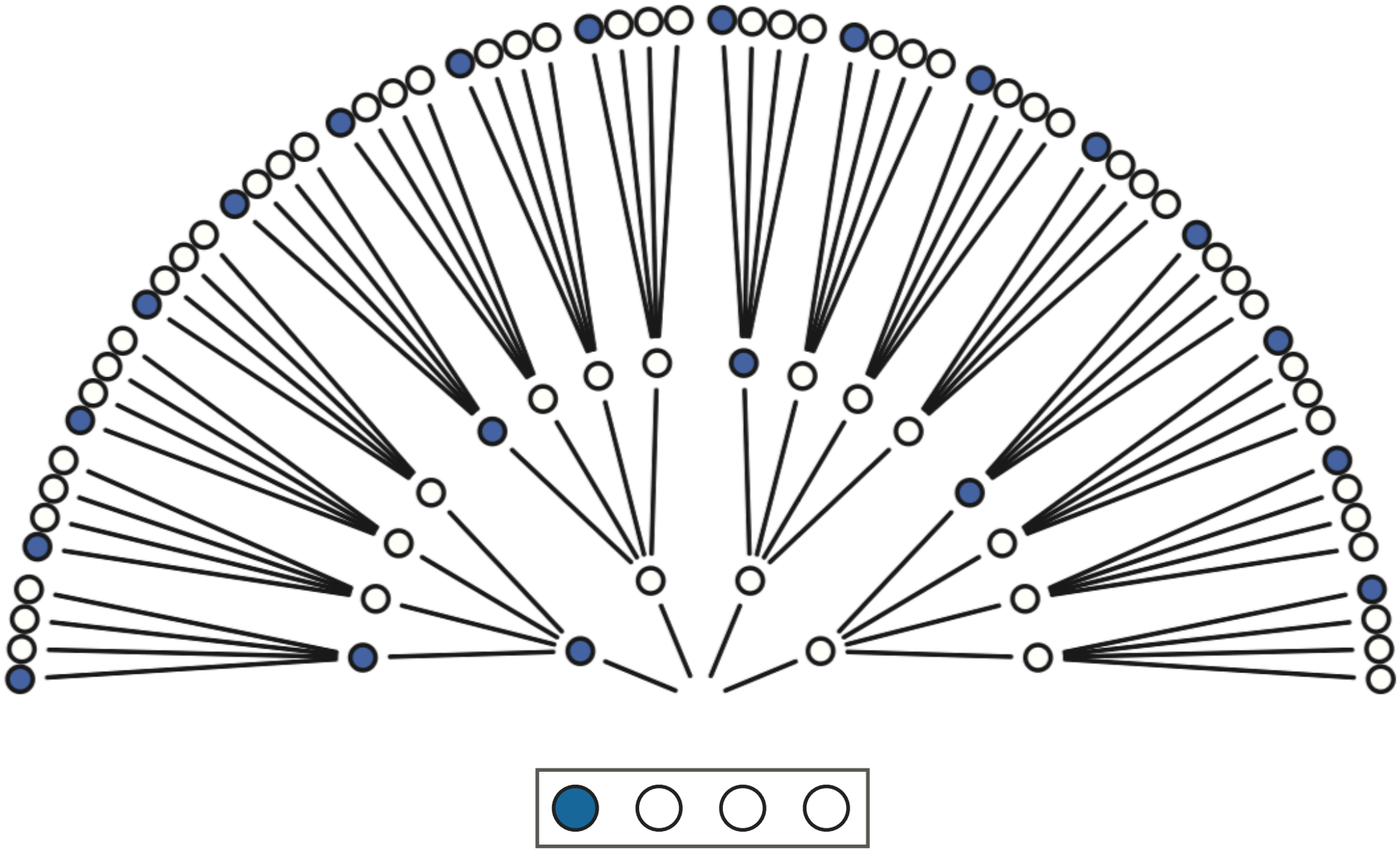
Underlying distribution



Negative binomial(nlp,y)

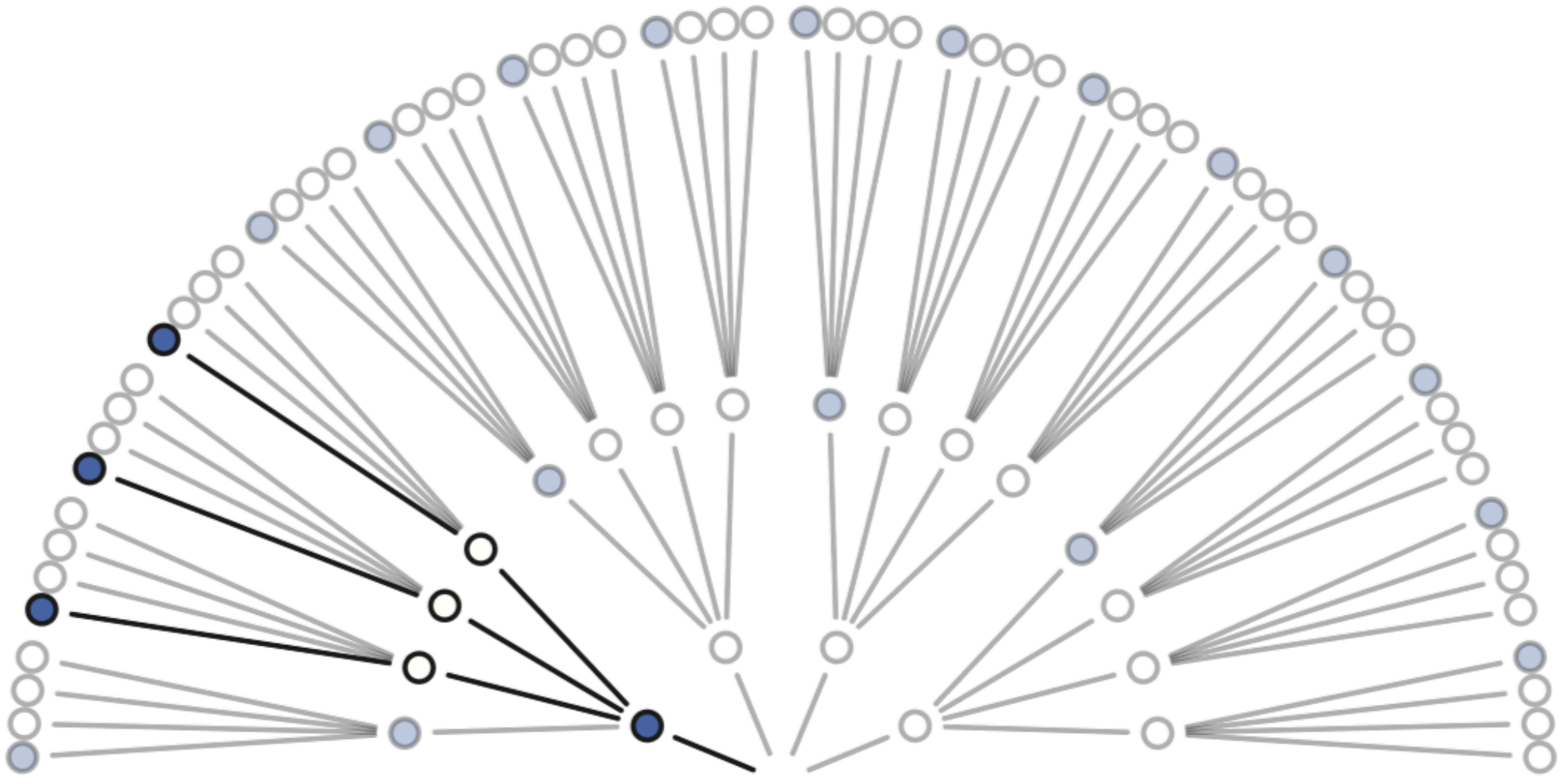
The garden of forking data

From the 2nd chapter of Richard McElreath's **Statistical Rethinking**

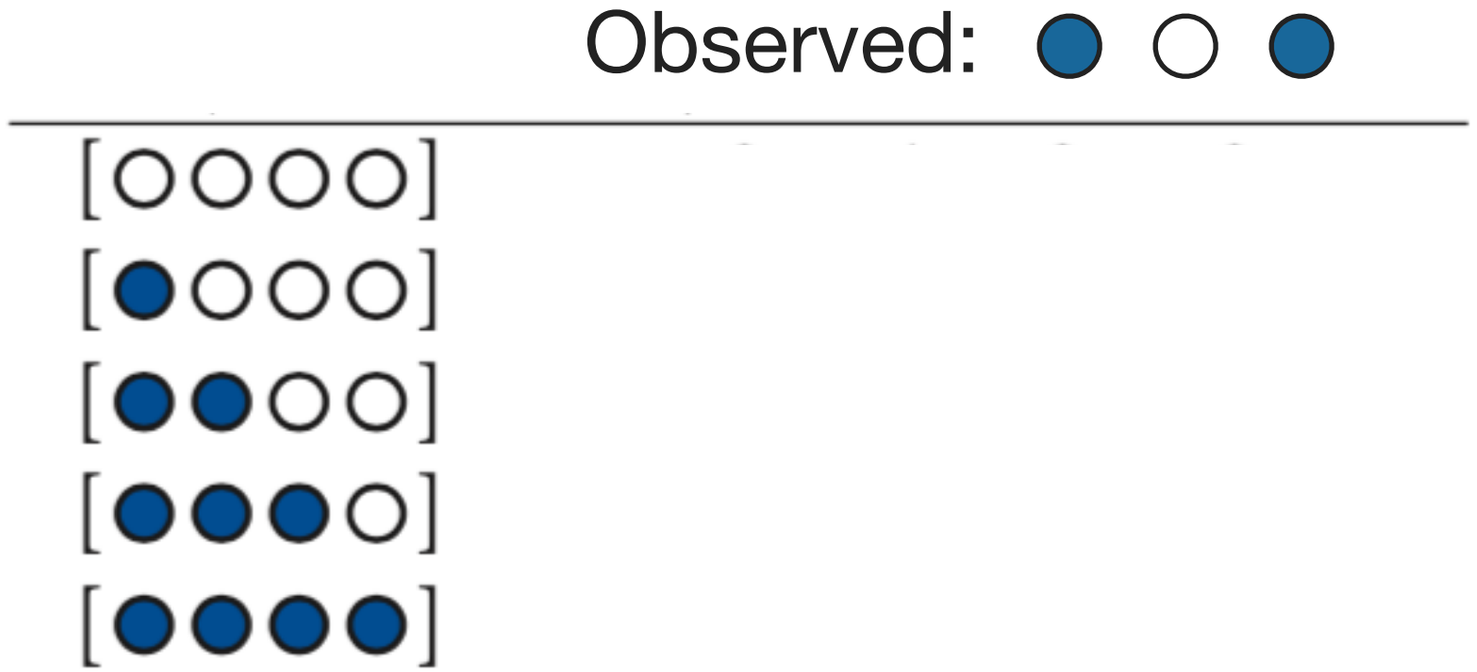


The garden of forking data

Observed: ● ○ ●

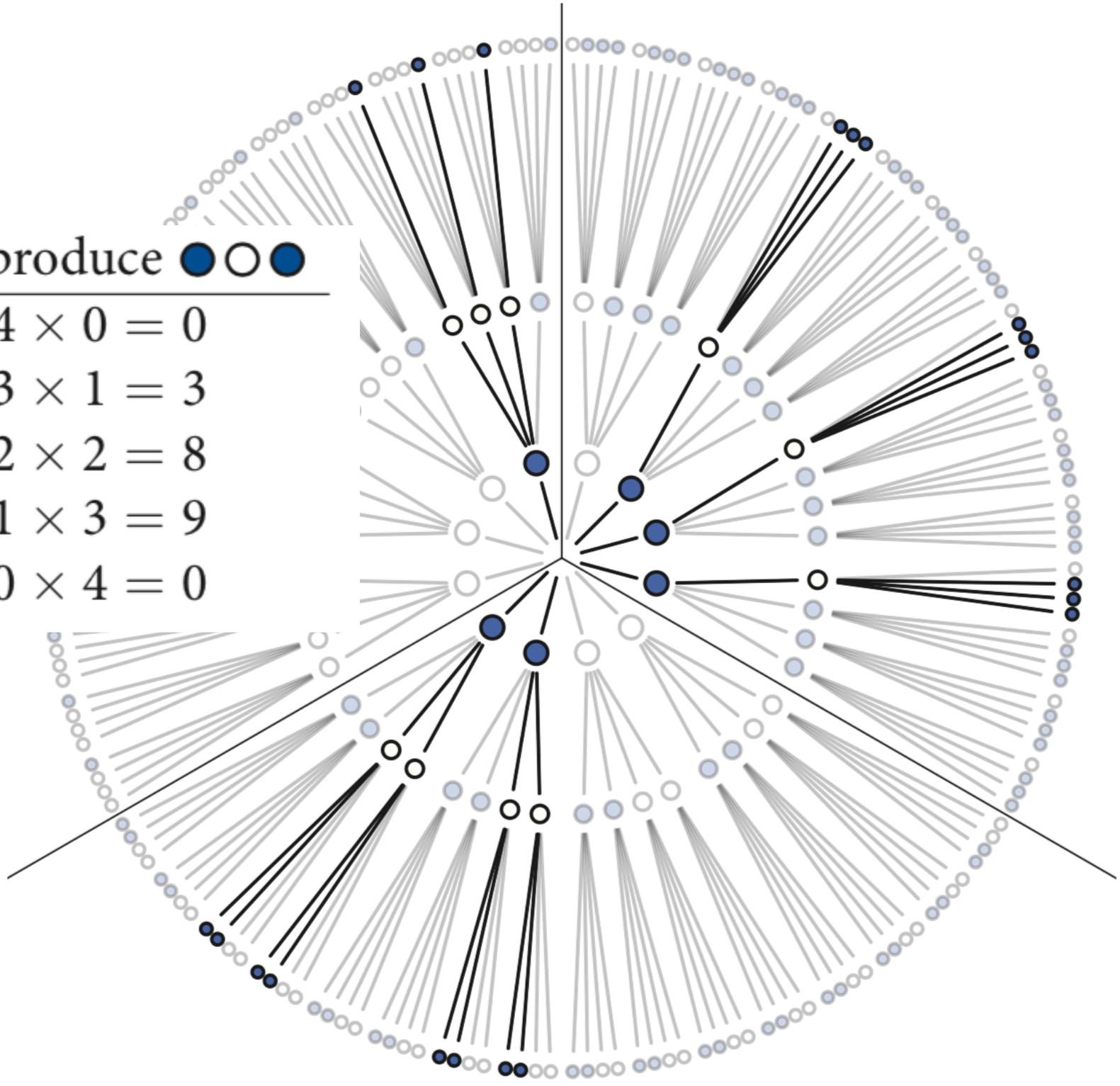


The garden of forking data



The garden of forking data

Conjecture	Ways to produce ●○●
[○○○○]	$0 \times 4 \times 0 = 0$
[●○○○]	$1 \times 3 \times 1 = 3$
[●●○○]	$2 \times 2 \times 2 = 8$
[●●●○]	$3 \times 1 \times 3 = 9$
[●●●●]	$4 \times 0 \times 4 = 0$



Counting the paths within a small world

“Bob is flipping a ~~fair~~ coin and he stop when he observed 5 heads”

[1 1 1 1 1]

[0 1 1 1 1 1] * 5

…

“He ends up flipping the coin 9 times”

“The bias of the coin is either 0.5 or 0.75”

Likelihood function

For example, binomial PMF

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

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

Code2 - likelihood.ipynb

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:

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

Likelihood of observing:

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

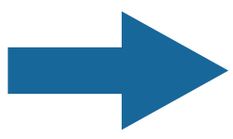
```
p = .7
y = np.asarray([0, 0, 0, 0, 1, 1])

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])
```

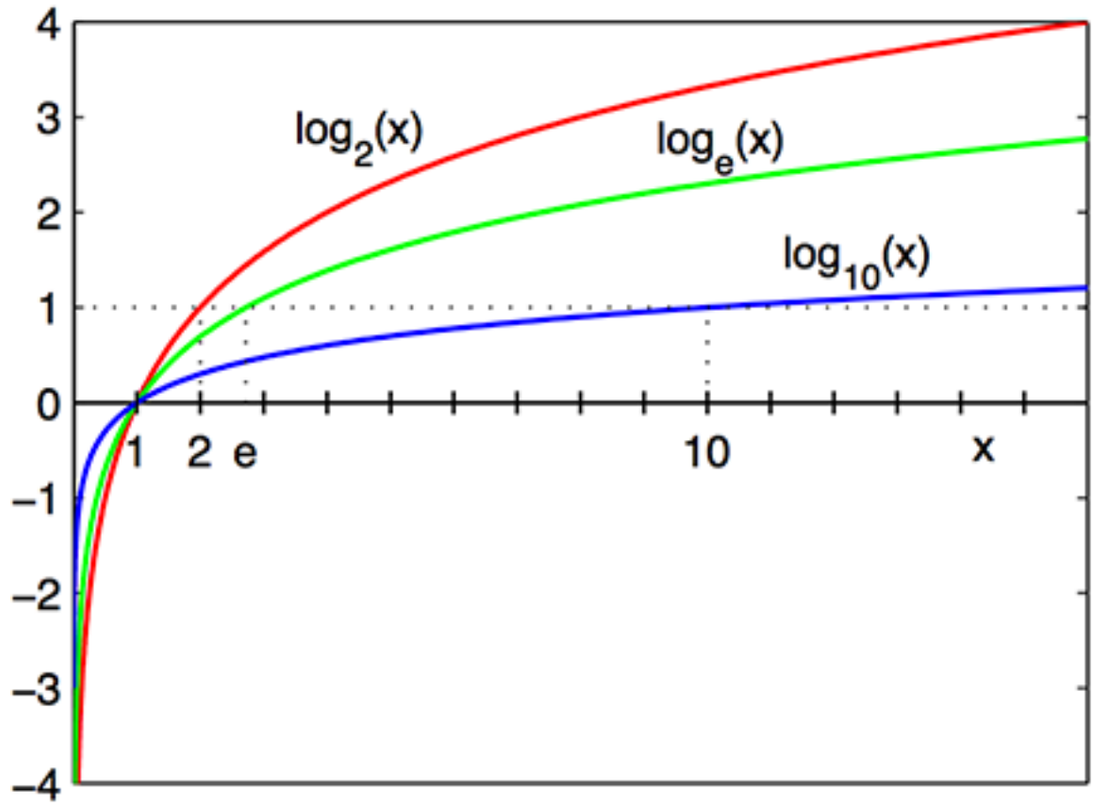
```
prob2 = np.cumprod(st.bernoulli.pmf(y, p))
prob2
```

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



Code3 - Combining_Likelihood.ipynb

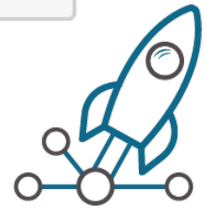
Log-Likelihood




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

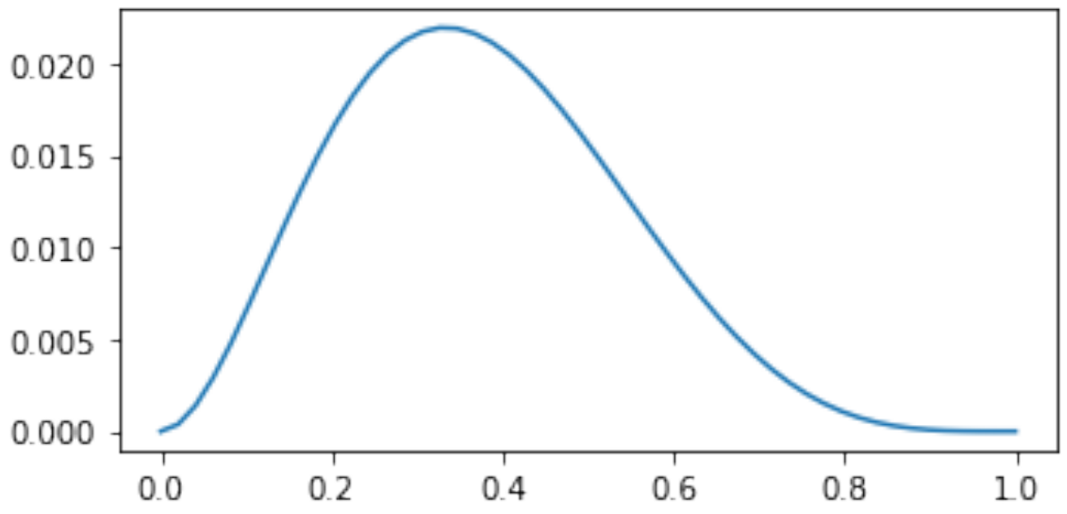
```
array([-1.2039728 , -2.40794561, -3.61191841, -4.81589122, -5.17256616,
       -5.52924111])
```

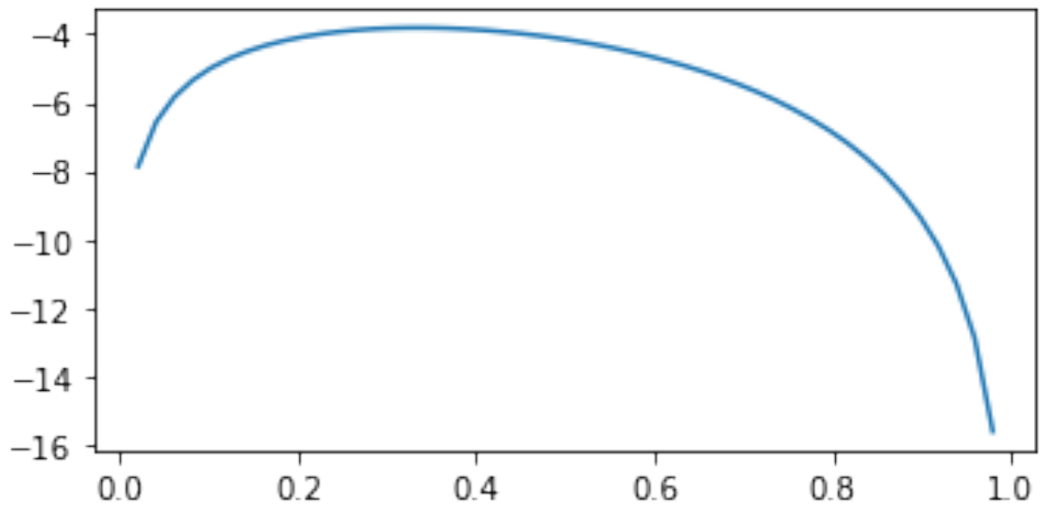
```
st.bernoulli.logpmf(y, p).cumsum()
```



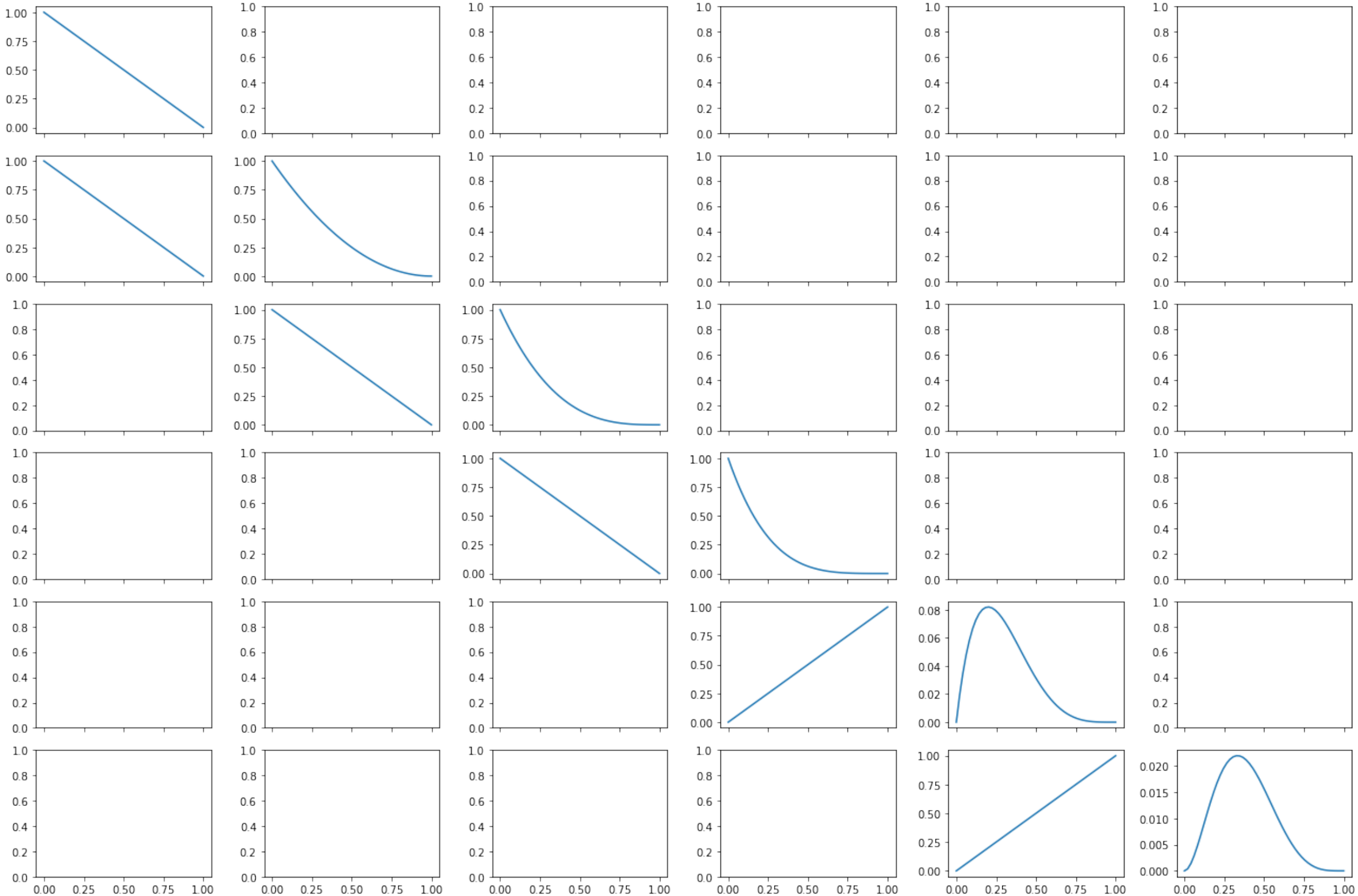
Combining Likelihood

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





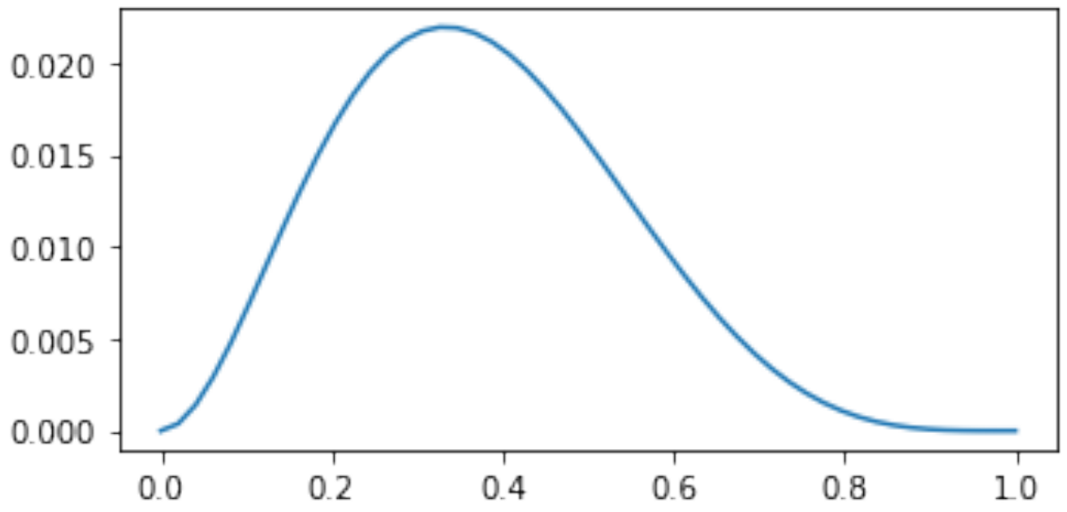
Update of Likelihood function

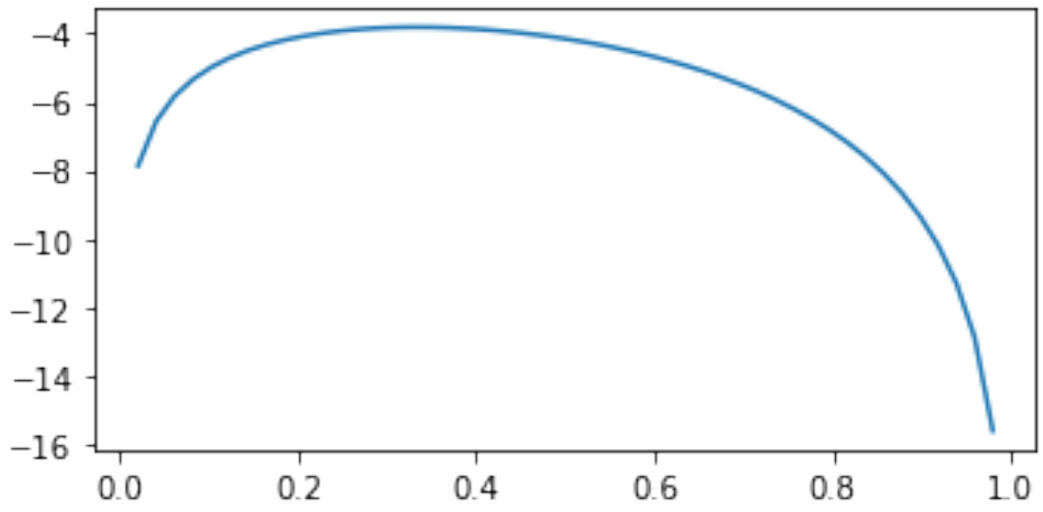


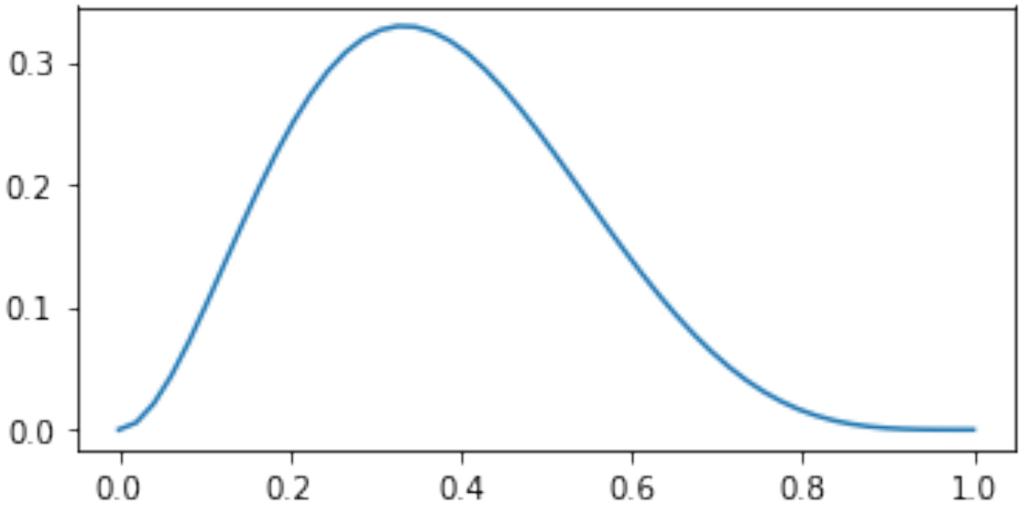
Likelihood function

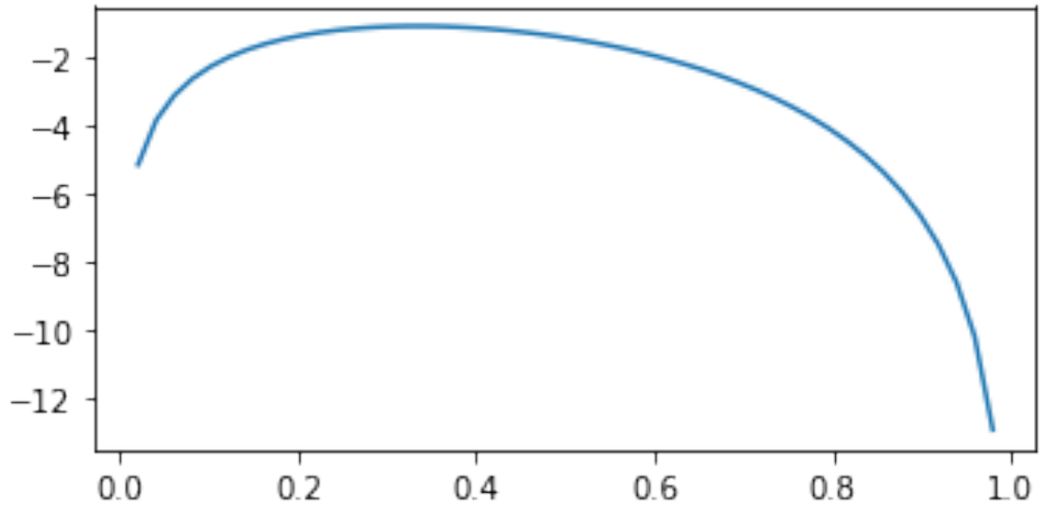
```
lambda p: st.bernoulli.pmf(y, p).prod()
```

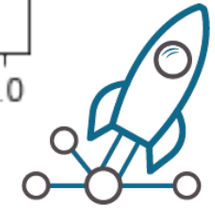
```
lambda p: st.binomial.pmf(y.sum(), len(y), p).prod()
```





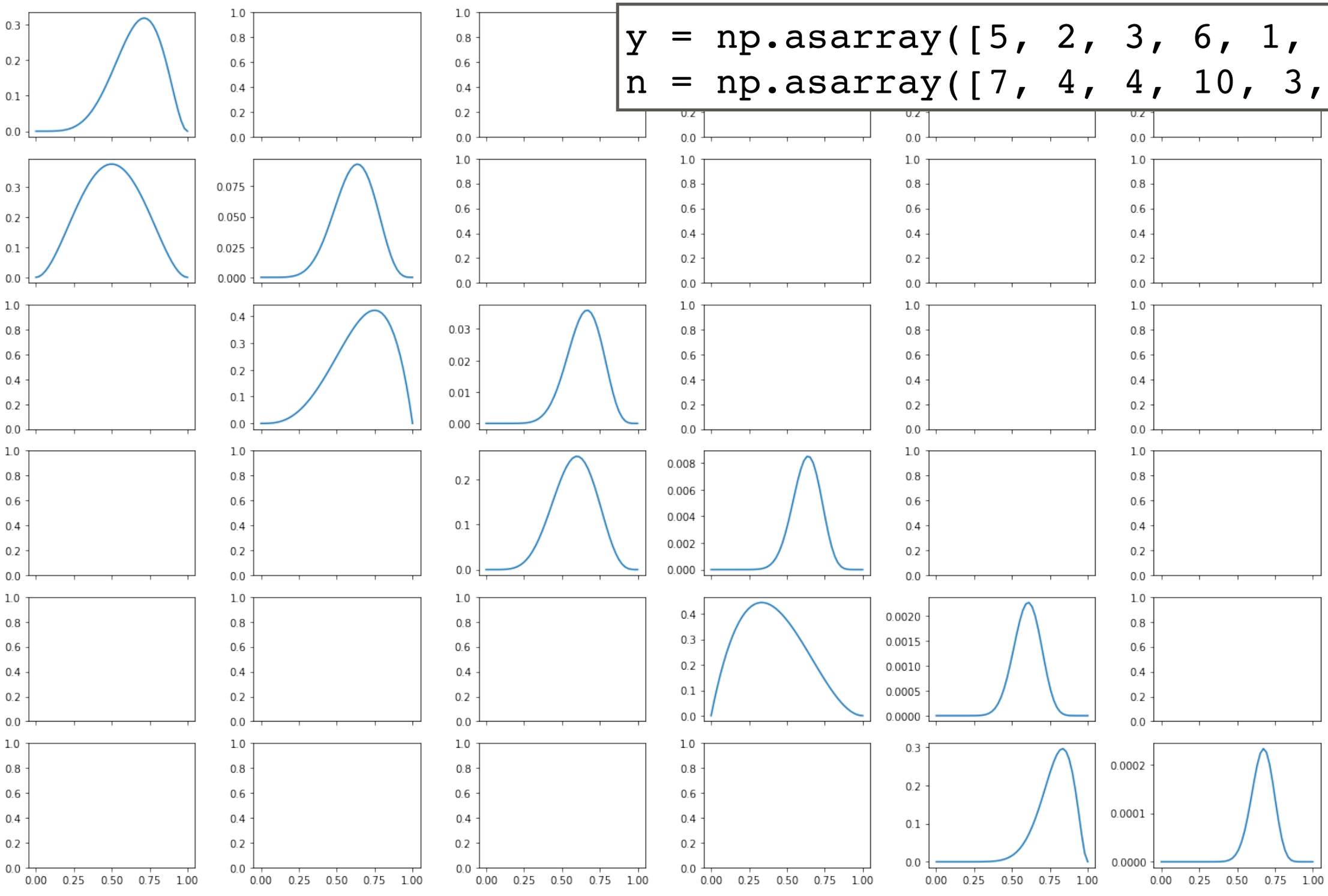




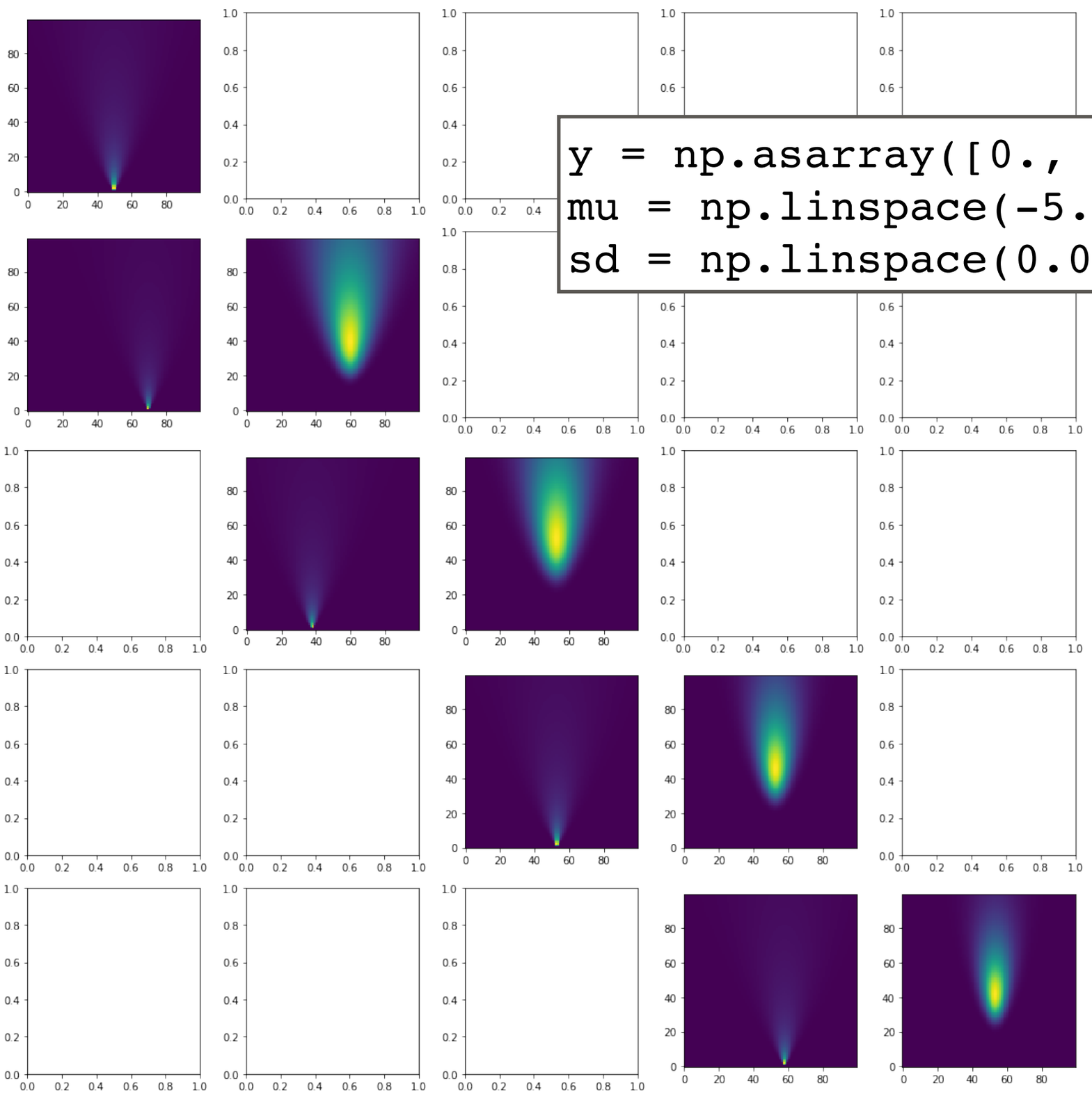


Update, or cumulating information

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



Update, or cumulating information

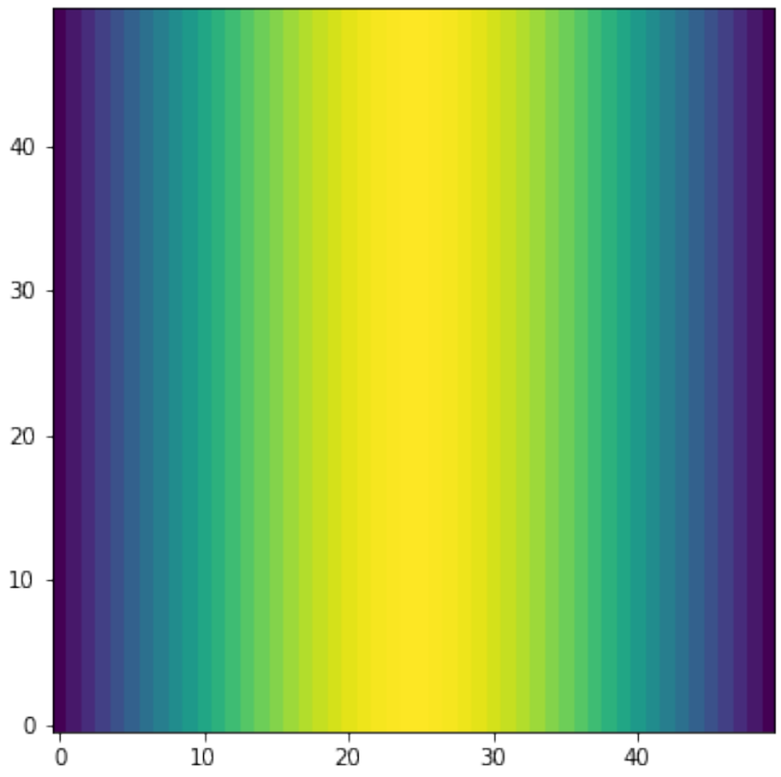


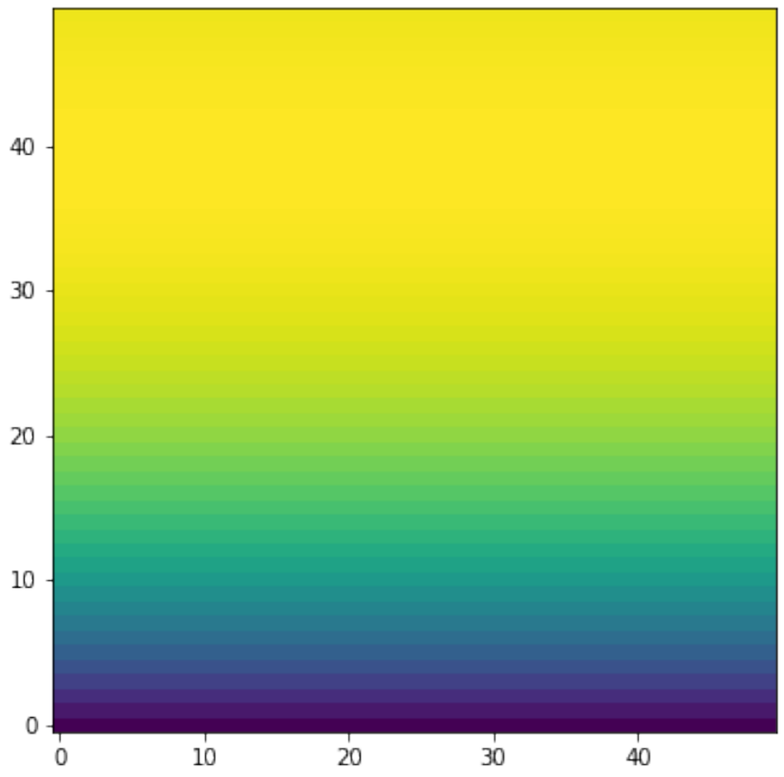
```
y = np.asarray([0., 2., -1.2, 0.3, .8])
mu = np.linspace(-5., 5., 100)
sd = np.linspace(0.001, 2.5, 100)
```

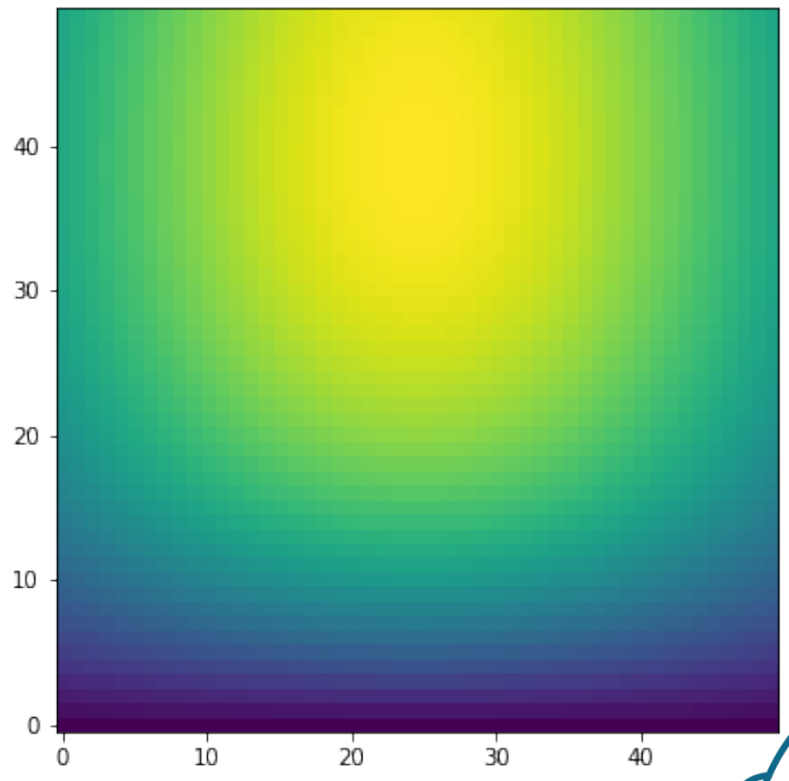
Priors

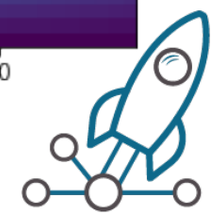
A and B	$P(A \cap B) = P(A\|B)P(B) = P(B\|A)P(A)$ $P(A \cap B) = P(A)P(B)$ if A and B are independent

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

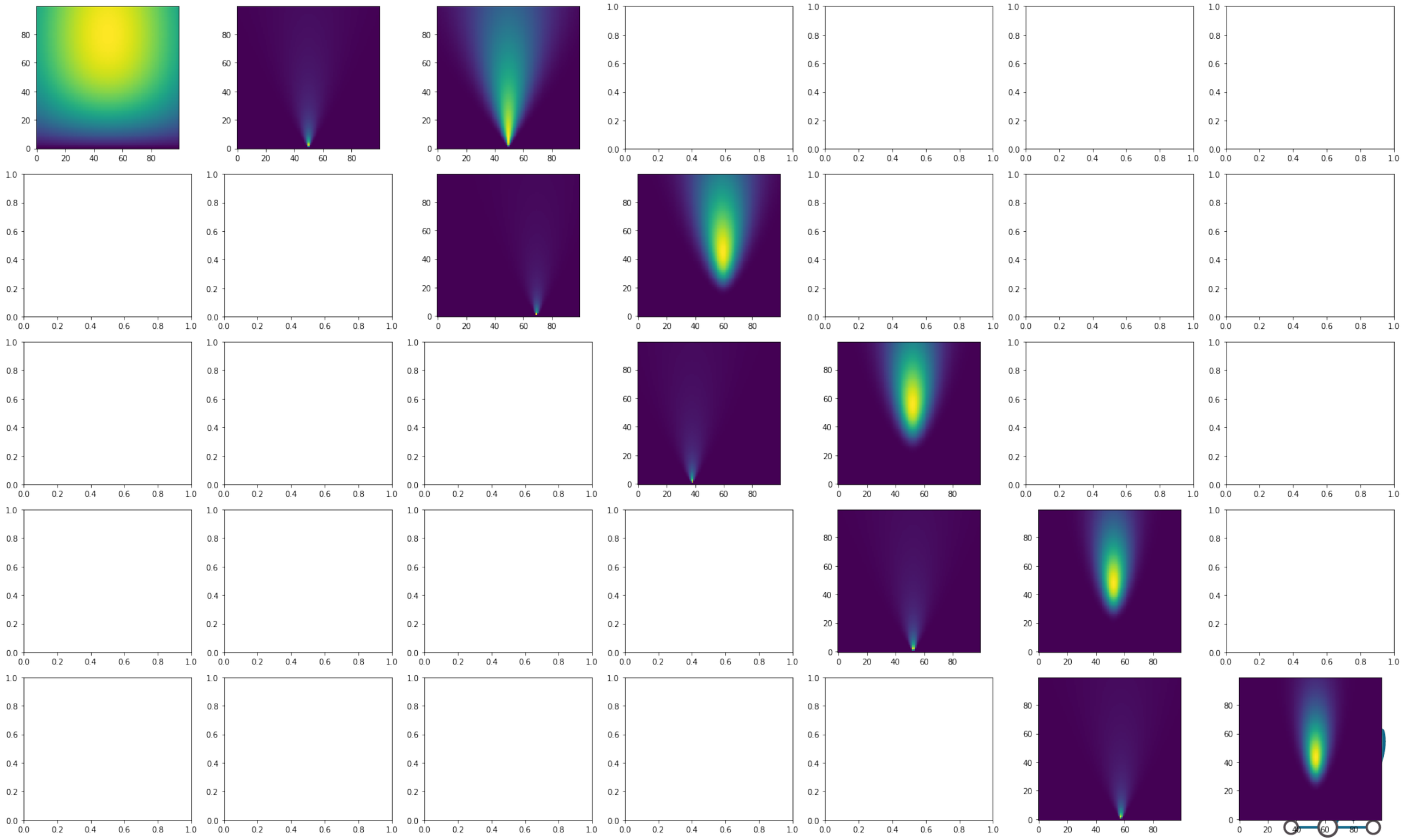




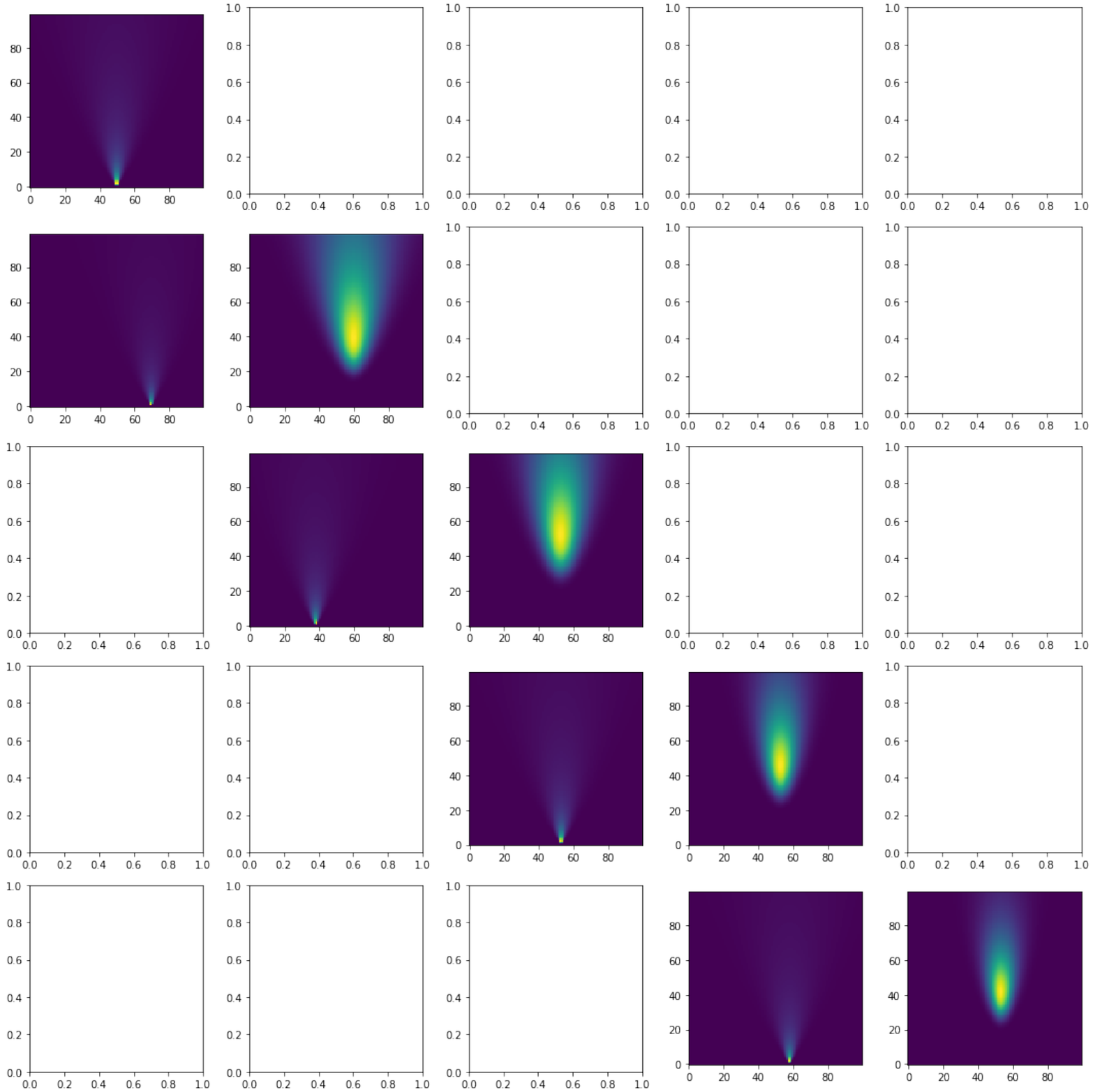




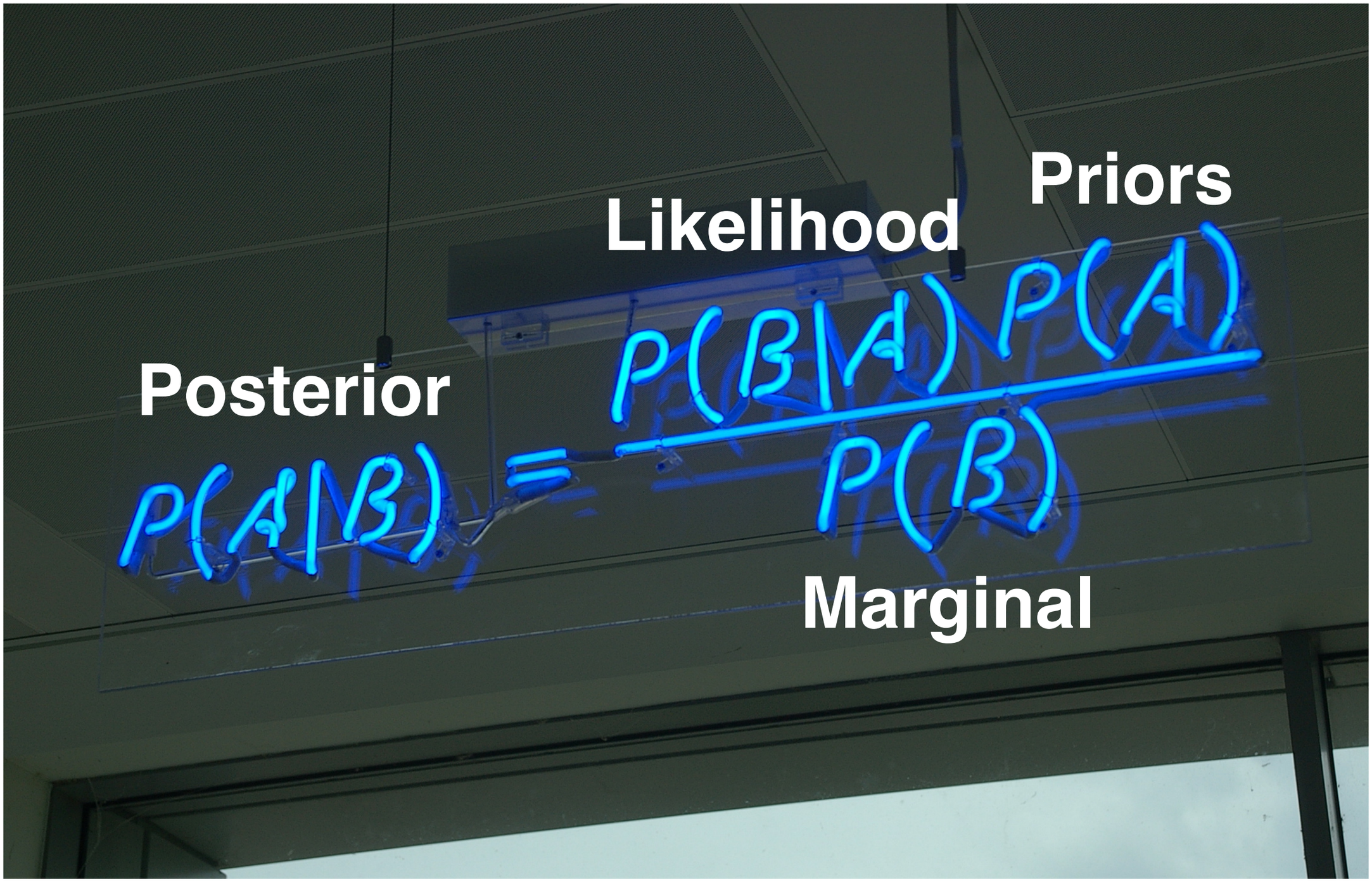
Update, or cumulating information



As comparison…



Bayes' theorem



Bayes's Rule

$$
\begin{aligned}
p(\theta|y) &= \frac{p(\theta, y)}{p(y)} && \text{[definition of conditional probability]} \\
&= \frac{p(y|\theta)\, p(\theta)}{p(y)} && \text{[chain rule]} \\
&= \frac{p(y|\theta)\, p(\theta)}{\int_\Theta p(y, \theta)\, d\theta} && \text{[law of total probability]} \\
&= \frac{p(y|\theta)\, p(\theta)}{\int_\Theta p(y|\theta)\, p(\theta)\, d\theta} && \text{[chain rule]} \\
&\propto p(y|\theta)\, p(\theta) && [y \text{ is fixed}]
\end{aligned}
$$

Bayesian models

$$p(\text{Parameters} \mid \text{Data}) \propto p(\text{Data} \mid \text{Parameters}) * p(\text{Parameters})$$

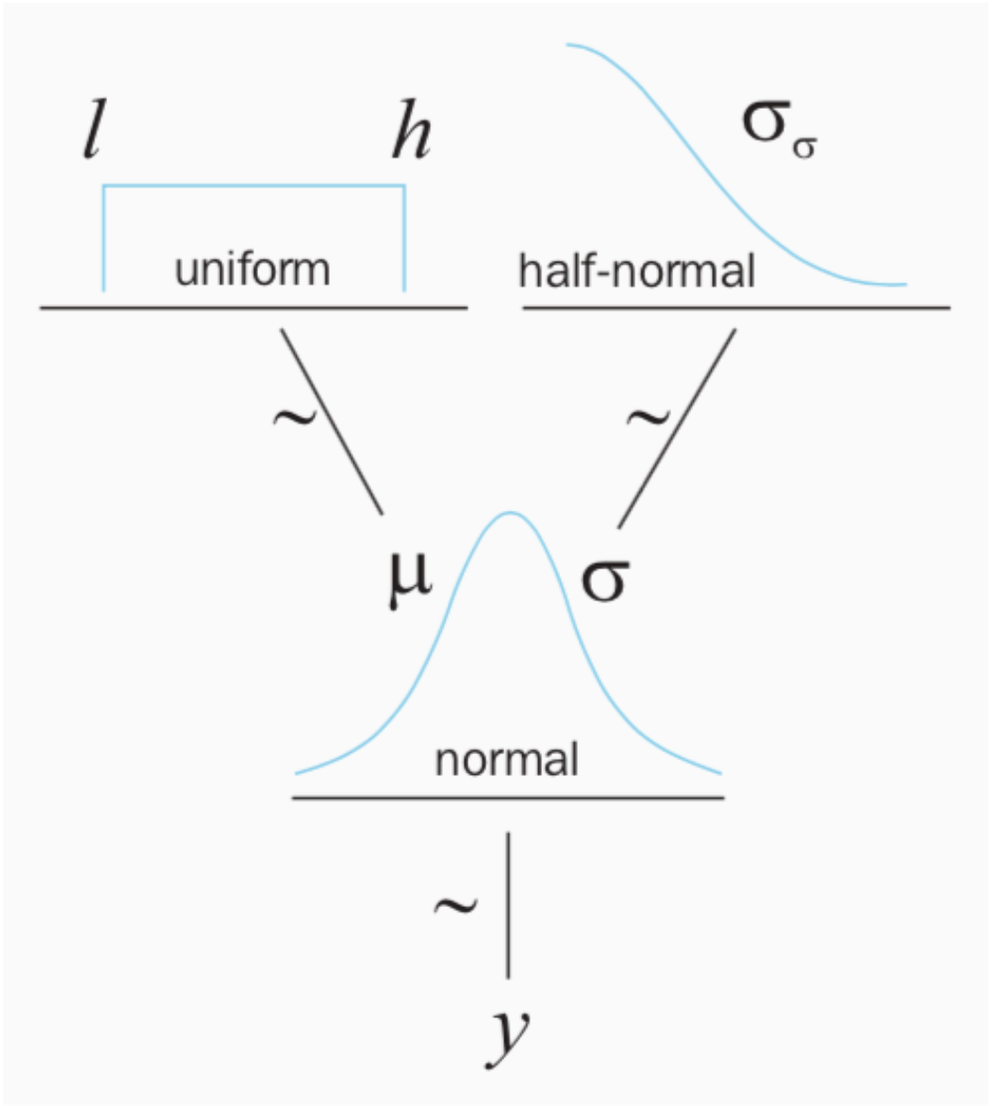
$$\mu \sim \text{Normal}(0, 5)$$
$$\sigma \sim \text{Gamma}(2, 5)$$
$$y \sim \text{Normal}(\mu, \sigma)$$

$$\alpha \sim \text{Gamma}(1, 1)$$
$$\beta_1, \ldots, \beta_K \sim \text{Beta}(1, \alpha)$$
$$w_i = \beta_i \prod_{j=i-1}^{i} (1 - \beta_j)$$

$$\lambda_1, \ldots, \lambda_K \sim U(0, 5)$$
$$\tau_1, \ldots, \tau_K \sim \text{Gamma}(1, 1)$$
$$\mu_i \mid \lambda_i, \tau_i \sim N\left(0, (\lambda_i \tau_i)^{-1}\right)$$

$$x \mid w_i, \lambda_i, \tau_i, \mu_i \sim \sum_{i=1}^{K} w_i \, N(\mu_i, (\lambda_i \tau_i)^{-1})$$

Generative model

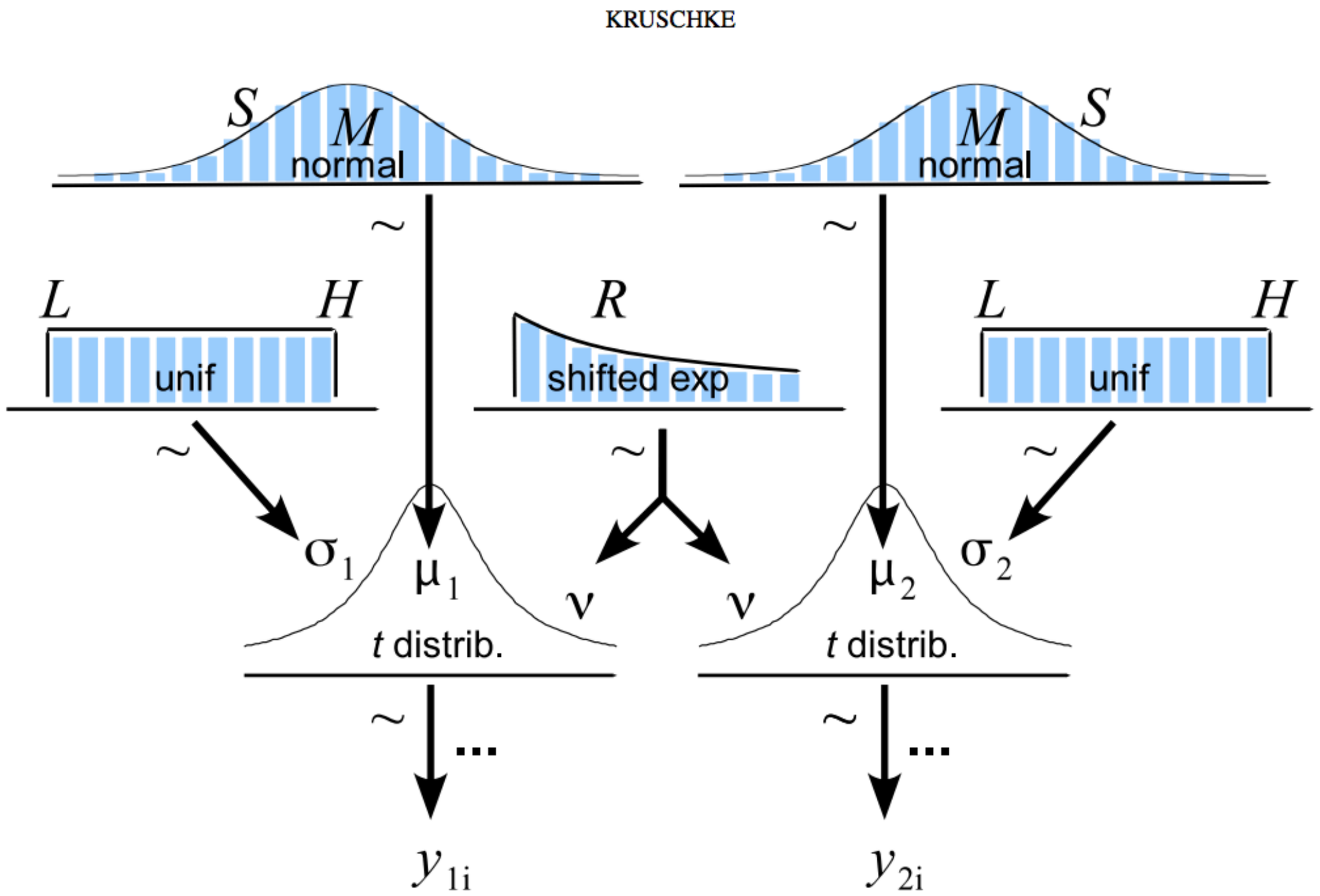


mu ~ Uniform(-5, 5)

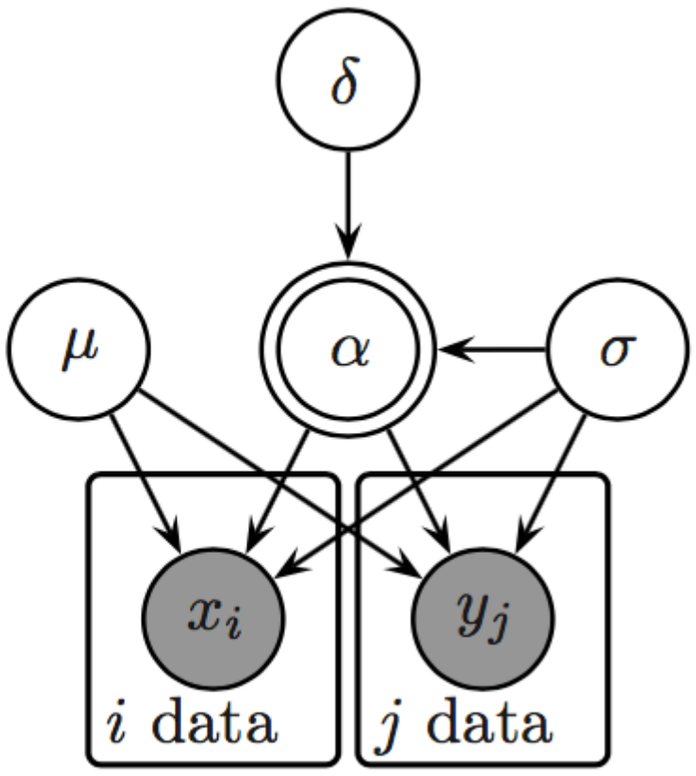
sd ~ HalfNormal(2.5)

y ~ Normal(mu, sd)

BEST


KRUSCHKE
S
M
normal
S
M
normal
~
~
L
H
unif
R
shifted exp
L
H
unif
~
~
~
σ1
μ1
ν
ν
μ2
σ2
t distrib.
t distrib.
~
~
~
...
...
y1i
y2i

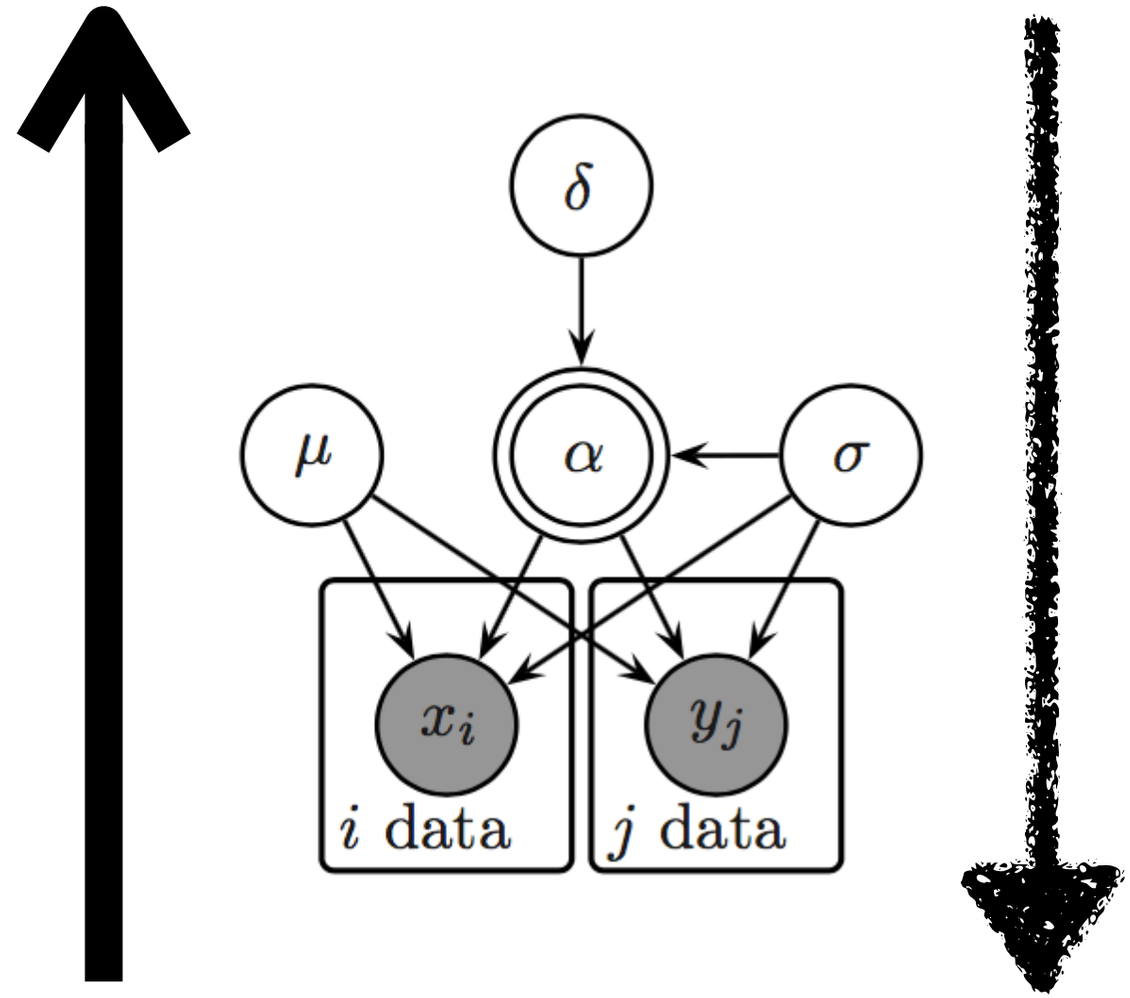
Bayesian Graphical Models



$$
\begin{aligned}
\delta &\sim \text{Cauchy}(0, 1) \\
\mu &\sim \text{Cauchy}(0, 1) \\
\sigma &\sim \text{Cauchy}(0, 1)_{\mathcal{I}(0,\infty)} \\
\alpha &\leftarrow \delta\sigma \\
x_i &\sim \text{Gaussian}(\mu + \tfrac{\alpha}{2}, 1/\sigma^2) \\
y_j &\sim \text{Gaussian}(\mu - \tfrac{\alpha}{2}, 1/\sigma^2)
\end{aligned}
$$

Bayesian Graphical Models

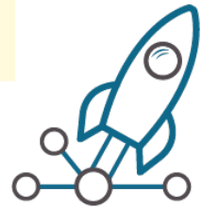
Likelihood/
(loss function)



Generative

Components in PPL

```
739     def __init__(self, lam, *args, **kwargs):
740         super(Exponential, self).__init__(*args, **kwargs)
741         self.lam = lam = tt.as_tensor_variable(lam)
742         self.mean = 1. / self.lam
743         self.median = self.mean * tt.log(2)
744         self.mode = tt.zeros_like(self.lam)
745 
746         self.variance = self.lam**-2
747 
748         assert_negative_support(lam, 'lam', 'Exponential')
749 
750     def random(self, point=None, size=None):
751         lam = draw_values([self.lam], point=point)[0]
752         return generate_samples(np.random.exponential, scale=1. / lam,
753                                 dist_shape=self.shape,
754                                 size=size)
755 
756     def logp(self, value):
757         lam = self.lam
758         return bound(tt.log(lam) - lam * value, value >= 0, lam > 0)
759 
```



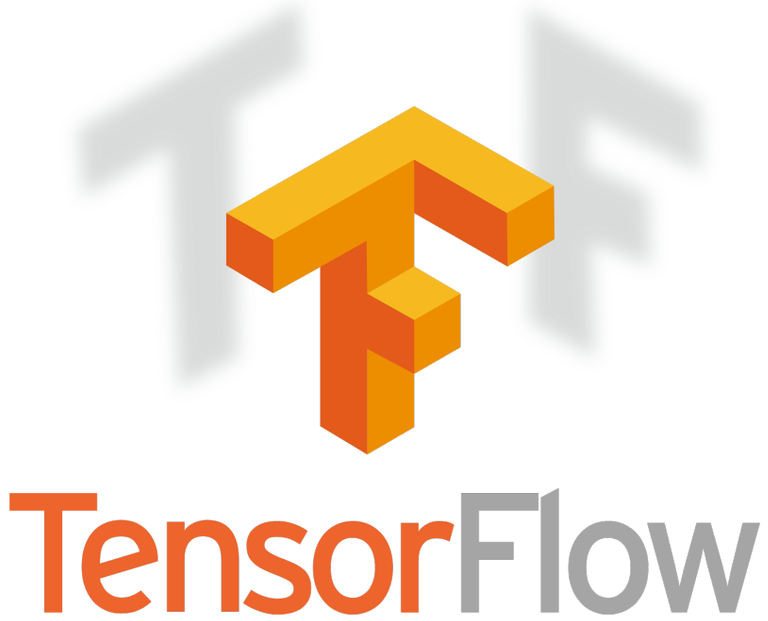
Components in PPL

```
47  _DISTRIBUTION_PUBLIC_METHOD_WRAPPERS = [
48      "batch_shape",
49      "batch_shape_tensor",
50      "cdf",
51      "covariance",
52      "cross_entropy",
53      "entropy",
54      "event_shape",
55      "event_shape_tensor",
56      "kl_divergence",
57      "log_cdf",
58      "log_prob",
59      "log_survival_function",
60      "mean",
61      "mode",
62      "prob",
63      "sample",
64      "stddev",
65      "survival_function",
66      "variance",
67  ]
68
```

TensorFlow distributions

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.



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.

Why PyMC3?

Easy to interact with different components of your model.

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)
```

Code3 - Combining_Likelihood.ipynb

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)}
```

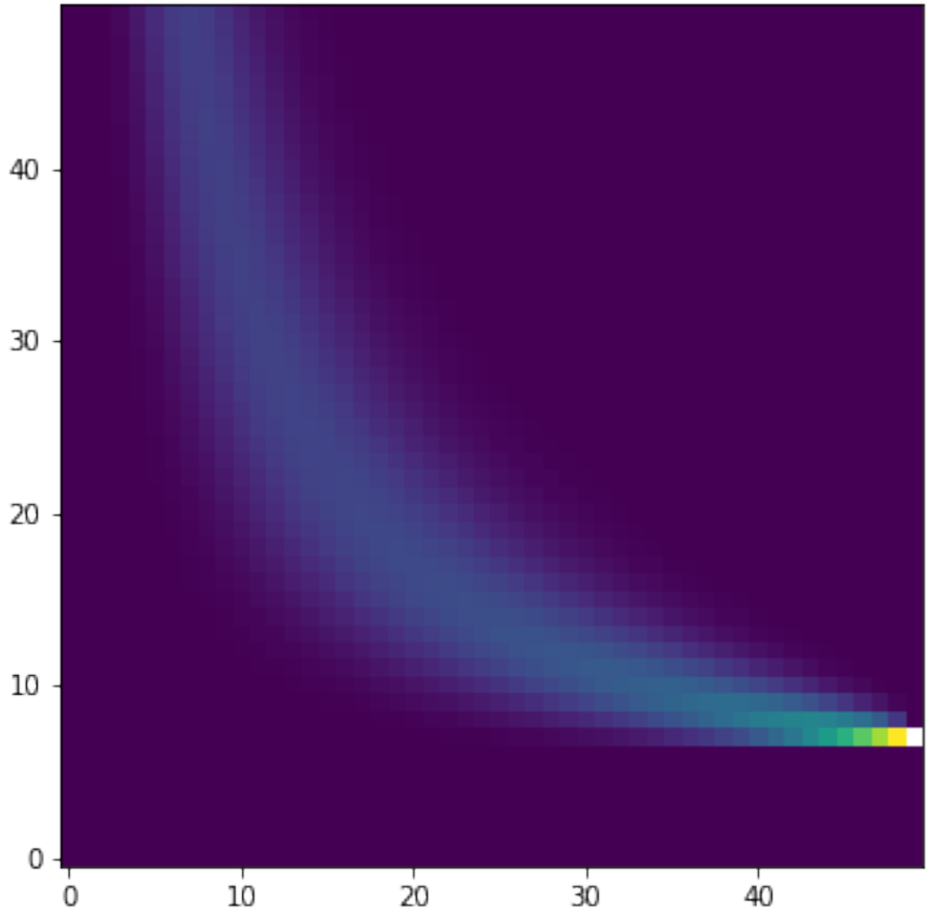
```
logp_m = m.logp
logp_y = y.logp
```

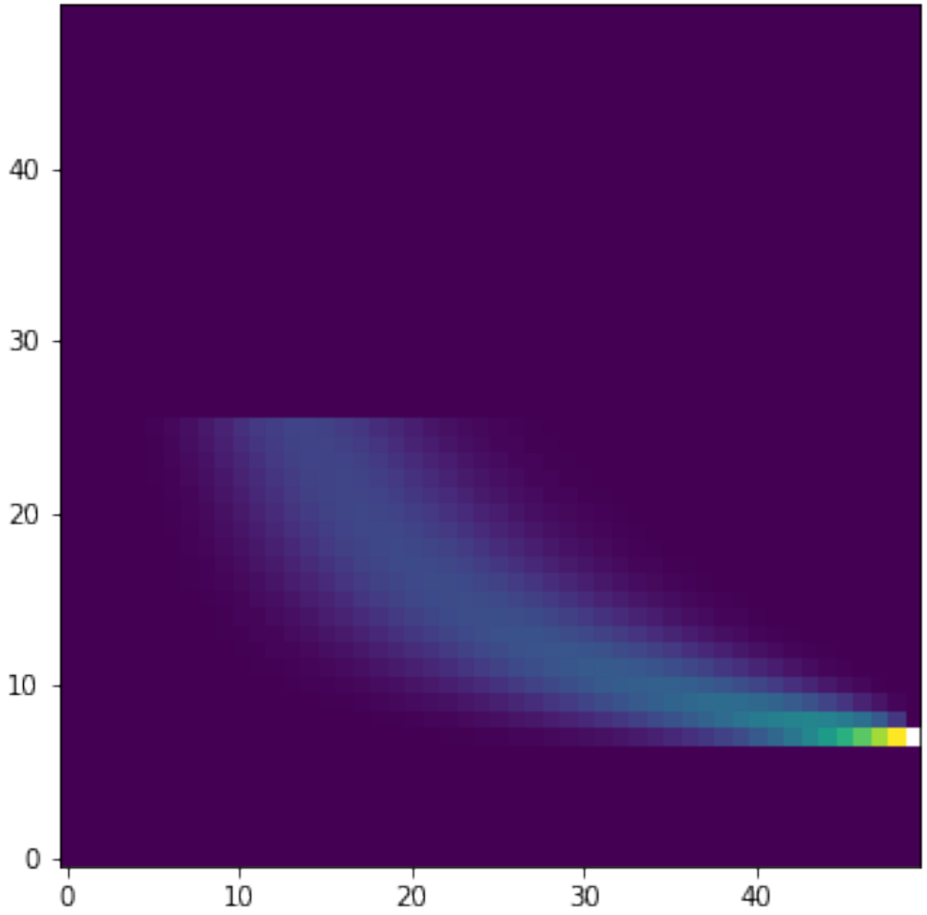
```
logp_m
```

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

Coin flip in PyMC3

```
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)
```





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
```

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

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

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

How PyMC3 use Theano

```
m.logp
```

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

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

```
n.distribution.logp
```

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

```
n.logpt
```

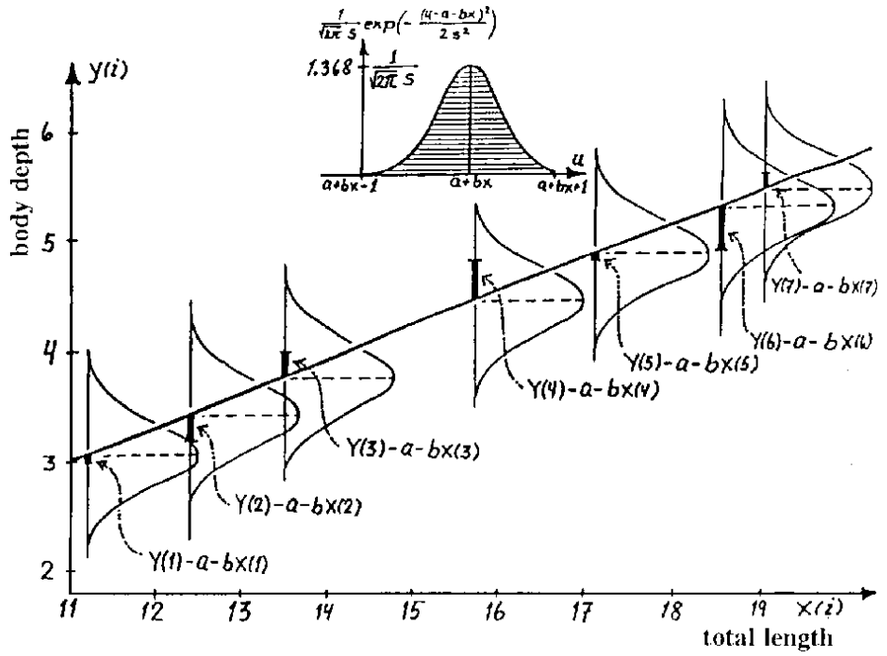
How PyMC3 use Theano

`m.logp`

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

```
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

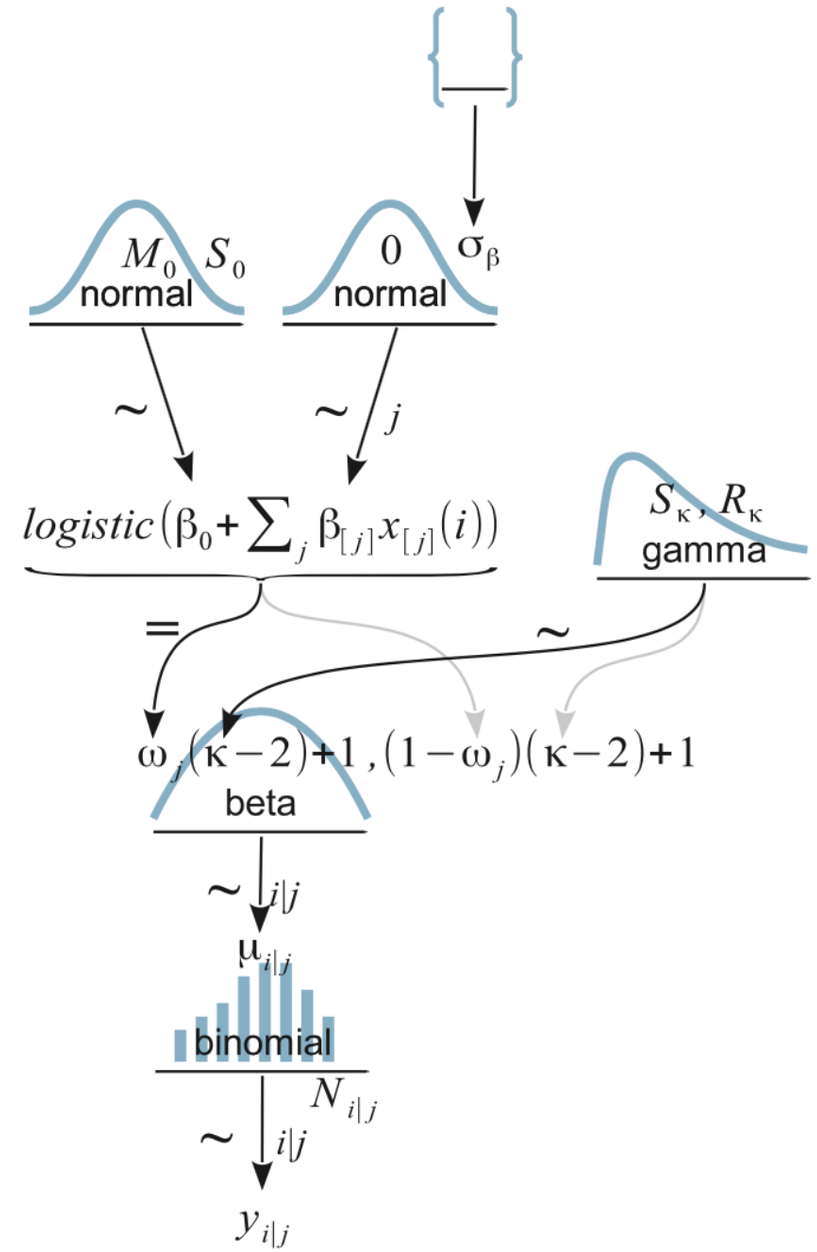
Source	d.f.	SS	MS	F
Treatment	$a-1$	SS_{treat}	$\frac{SS_{treat}}{a-1}$	$\frac{MS_{treat}}{MS_{error(a)}}$
Error (a)	$N-a$	$SS_{error(a)}$	$\frac{SS_{error(a)}}{N-a}$	
Time	$t-1$	SS_{time}	$\frac{SS_{time}}{t-1}$	$\frac{MS_{time}}{MS_{error(b)}}$
Treat x Time	$(a-1)(t-1)$	$SS_{treat\ x\ time}$	$\frac{SS_{treat\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

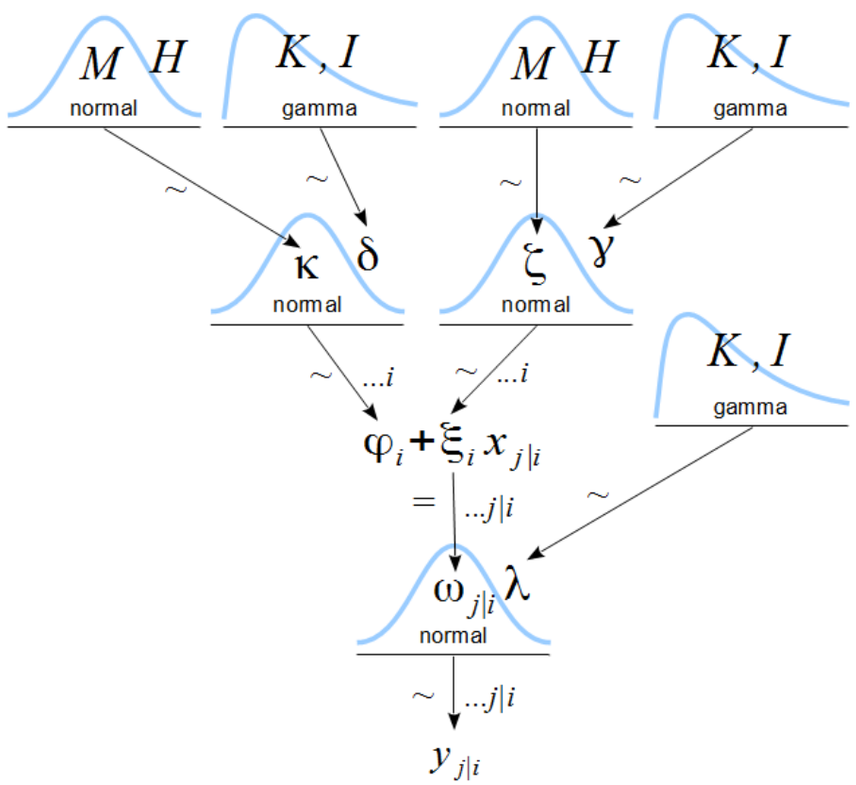
$$\mathbf{y} = \begin{pmatrix} y_{11} \\ y_{12} \\ y_{21} \\ y_{22} \\ y_{23} \\ y_{31} \\ y_{32} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 2 \\ 1 & 2 \\ 1 & 3 \\ 1 & 3 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{12}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

Generalised linear model

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon, \qquad \mathrm{E}(\mathbf{y}) = \mathbf{X}\beta = \boldsymbol{\mu}\,, \quad \mathrm{E}(\boldsymbol{\varepsilon}) = \mathbf{0}\,, \quad \mathrm{cov}(\mathbf{y}) = \mathrm{cov}(\boldsymbol{\varepsilon}) = \sigma^2\mathbf{V}\,.$$

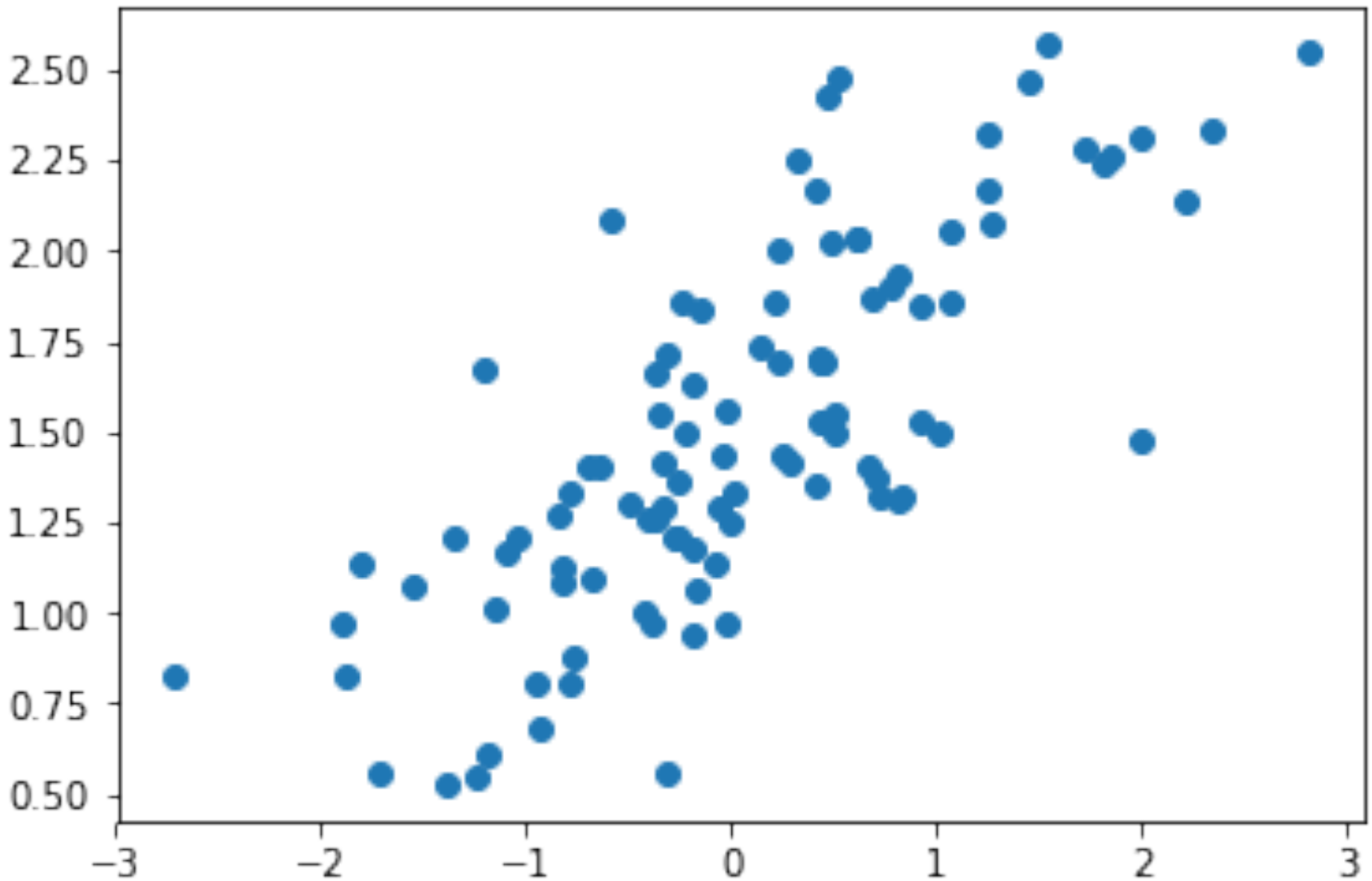
$$\begin{aligned}
\text{outcome}_i &\sim \text{Normal}(\mu_i, \sigma) \\
\mu_i &= \beta \times \text{predictor}_i \\
\beta &\sim \text{Normal}(0, 10) \\
\sigma &\sim \text{HalfCauchy}(0, 1)
\end{aligned}$$





Linear regression in PyMC3

Code4 - Linear_Regression.ipynb



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))
```

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 m1:
    …
    pm.Normal('eps', 0, 1,
              observed=(y - X*beta - a)/sd)
```

Recap:

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

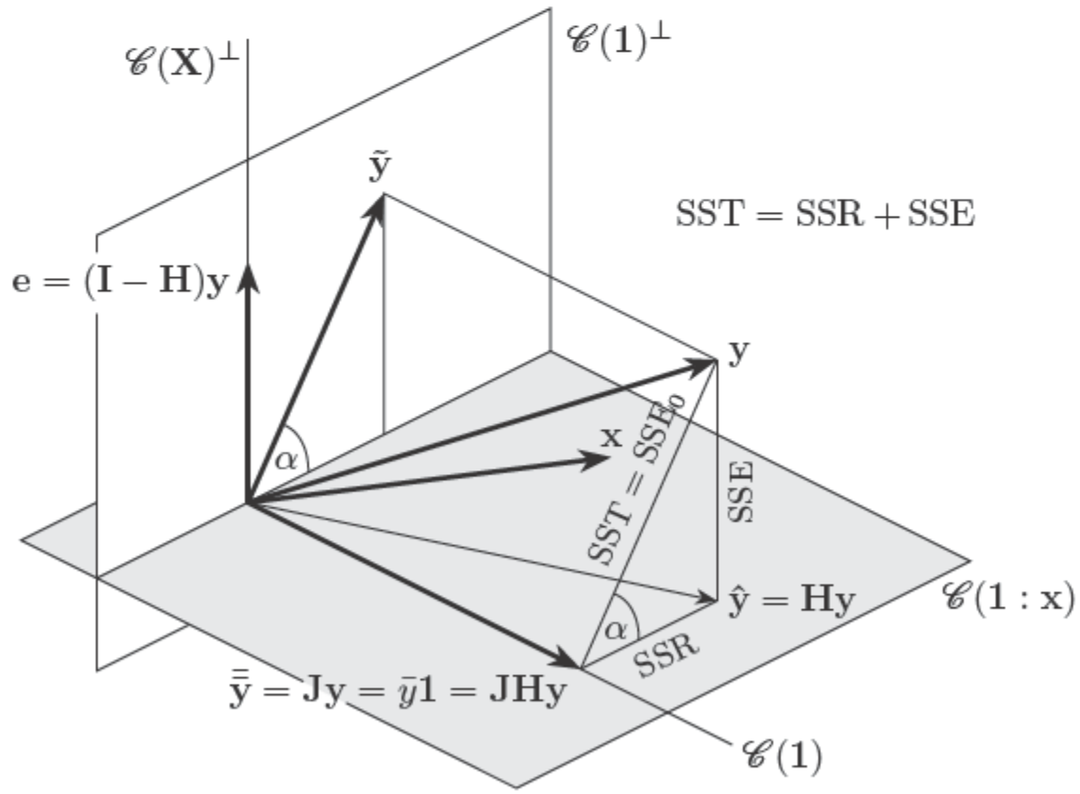
The dimension of the parameter space is the same as the number of unknowns.

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

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

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

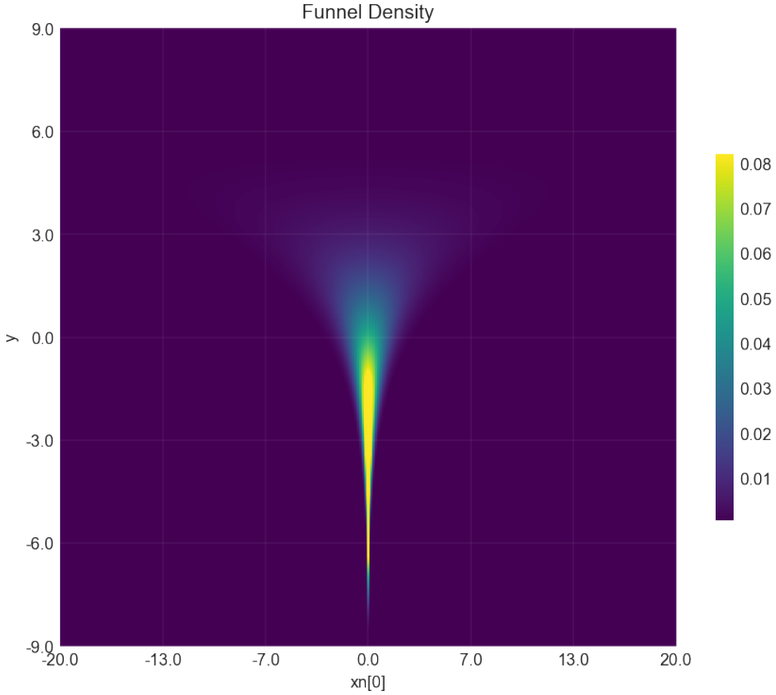
Parameterization

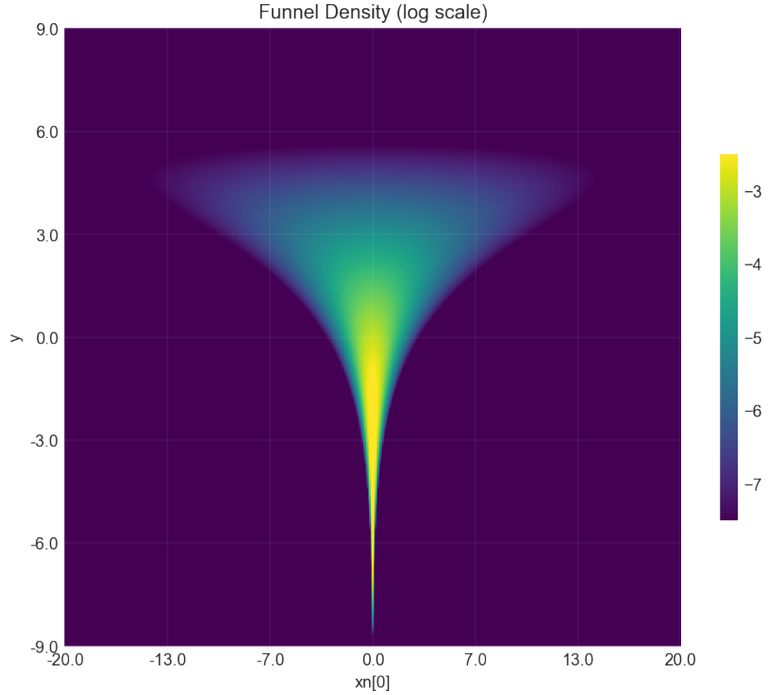
There are many ways to write down the same model:

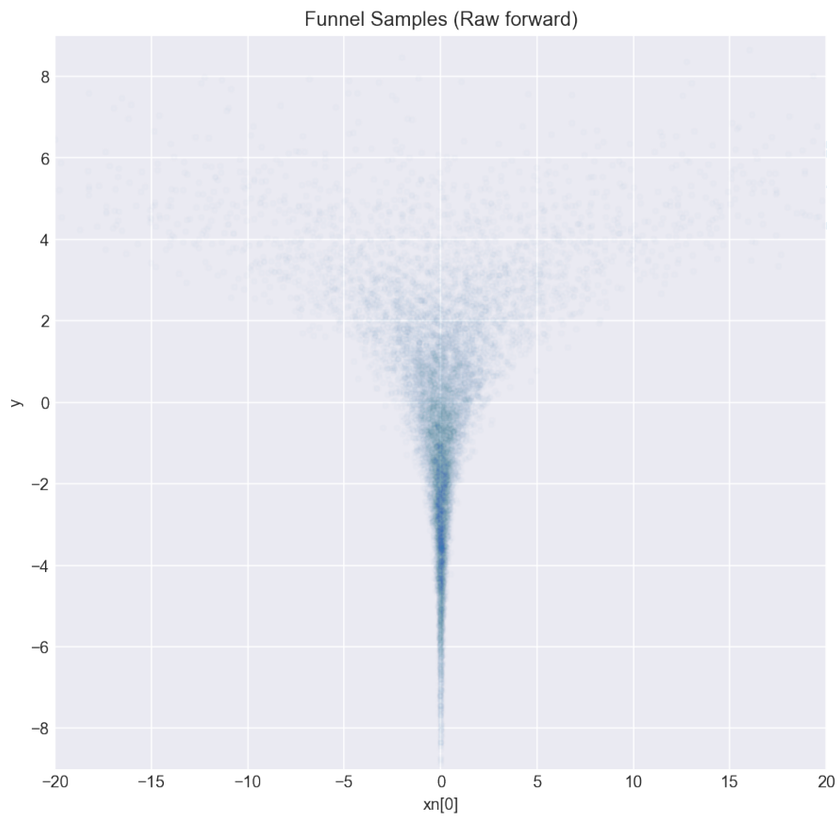
- Make sure it is correct
- Slow model usually indicates problem
  - *The folk theorem of statistical computing: When you have computational problems, often there's a problem with your model*
- A good parameterisation should be easy to understand

Example: Neal's Funnel

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







Code5 - Neals_funnel.ipynb

SESSION 1:
IN ALL LIKELIHOOD