Lecture 8

Bayesian Stats and Sampling

Today:

- Bayesian Stats recap
- replicative posterior predictives
- Normal-normal Model
- exponential model
- Inverse Transform Sampling
- Rejection Sampling

Last Time

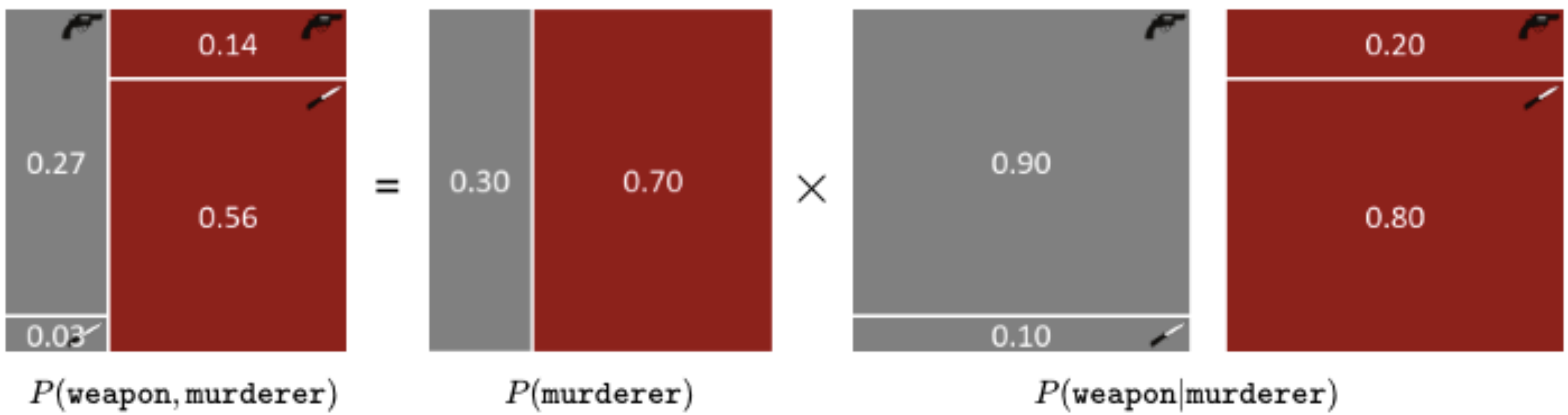
- Entropy
- Maximum Likelihood and Entropy
- Bayesian Stats

Bayesian Stats

- assume sample IS the data, no stochasticity
- parameters θ are stochastic random variables
- associate the parameter θ with a prior distribution $p(\theta)$
- The prior distribution generally represents our belief on the parameter values when we have not observed any data yet (to be qualified later)
- obtain posterior distributions
- predictive distribution from the posterior

Basic Idea

Get the joint Probability distribution



Now we condition on some random variables and learn the values of others.

Rules

1. $P(A, B) = P(A \mid B)P(B)$
2. $P(A) = \sum_B P(A, B) = \sum_B P(A \mid B)P(B)$

$P(A)$ is called the **marginal** distribution of A, obtained by summing or marginalizing over B.

Posterior

$$p(\theta|D = \{y\}) = \frac{p(D|\theta)\, p(\theta)}{p(D)}$$

Posterior: $p(\theta|D) \propto p(D|\theta)\, p(\theta)$

Evidence:

$$p(D = \{y\}) = \int d\theta\, p(\theta, D) = \int d\theta\, p(D|\theta) p(\theta).$$

Marginalization

Marginal posterior: $p(\theta_1|D) = \int d\theta_{-1} p(\theta|D)$.

Posterior Predictive:

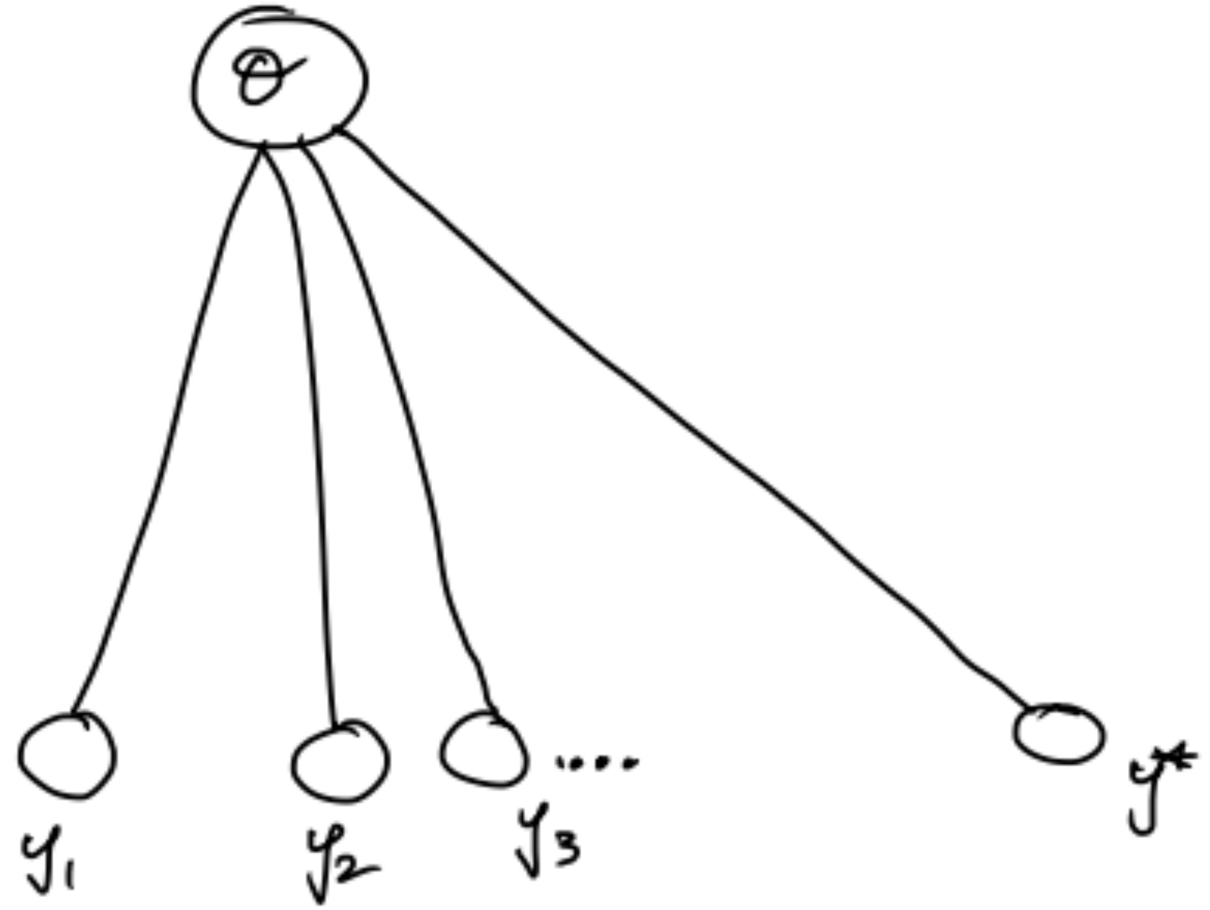
$$p(y^*|D = \{y\}) = \int d\theta p(y^*, \theta|\{y\}).$$

Basic Graph

$$p(\theta, y, y^*) = p(\theta)p(y|\theta)p(y^*|\theta)$$
$$= p(\theta|y)p(y)p(y^*|\theta)$$

$$p(y^*|y) = \int d\theta\, p(\theta, y^*|y)$$
$$= \int d\theta\, \frac{p(y^*, y, \theta)}{p(y)}$$

$$p(y^*|y) = \int d\theta\, p(\theta|y)p(y^*|\theta)$$



Predictives

The distribution of a future data point y^*:

Posterior predictive:

$$p(y^*|D = \{y\}) = \int d\theta p(y^*|\theta)p(\theta|\{y\}).$$

The distribution of a data point y from the prior:

Prior predictive:

$$p(y) = \int d\theta \, p(\theta, y) = \int d\theta \, p(y|\theta)p(\theta).$$

Globe Toss Model

- Seal tosses globe, p is true water fraction
- data `WLWWWLWLW`
- Modeled using the Binomial Distribution, which is the distribution of a set of Bernoulli random variables.

Griddy Posterior

```
prior_pdf = lambda p: 1
like_pdf = lambda p: binom.pmf(k=6, n=9, p=p)
post_pdf = lambda p: like_pdf(p)*prior_pdf(p)
p_grid = np.linspace(0., 1., 1000)
post_vals = post_pdf(p_grid)
post_vals_normed = post_vals/np.sum(post_vals)
grid_post_samples = np.random.choice(p_grid, size=10000, replace=True, p=post_vals_normed)
```

- create a grid, evaluate posterior on it
- discrete-normalize this posterior to get probabilities
- sample the grid according to these probabilities

Laplace Approximation for p^*

Unnormalized posterior:

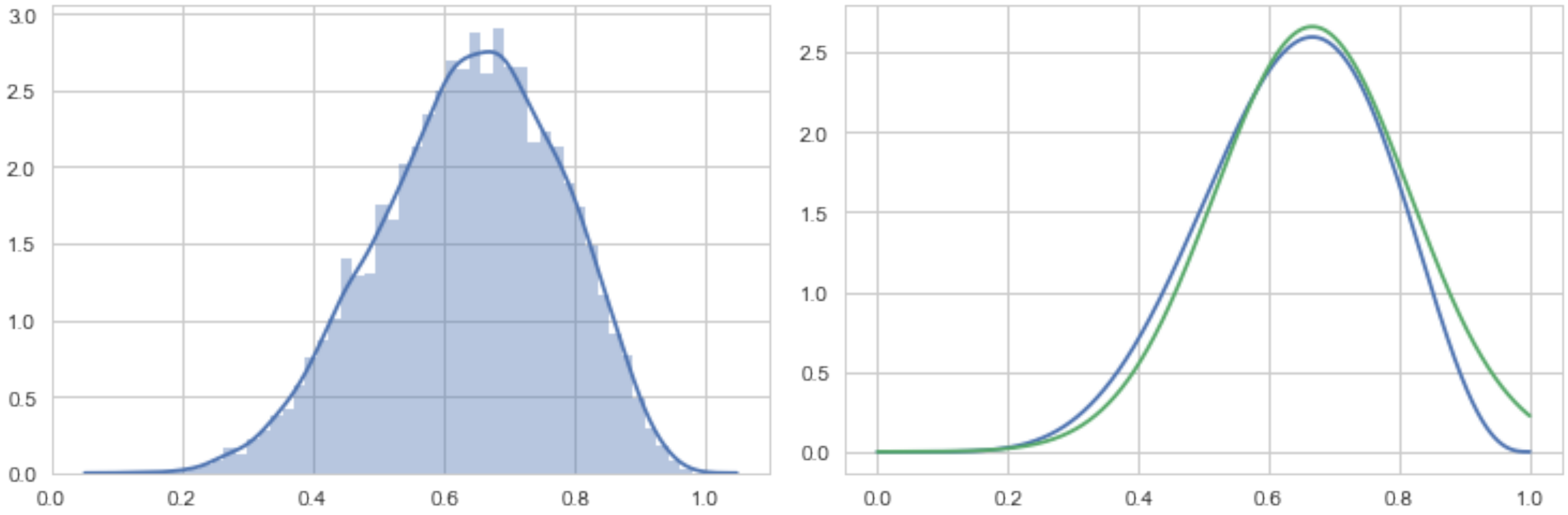
$$\log p^*(\theta|x) = \log p^*(\theta_{MAP}|x) + \frac{1}{2}(\theta - \theta_{MAP})^2 \left[\frac{d^2}{d\theta^2}\log p^*(\theta|x)\right]_{\theta=\theta_{MAP}} + \dots$$

Let $c = -[\frac{d^2}{d\theta^2}\log p^*(\theta|x)]_{\theta=\theta_{MAP}}$ then we get un-normalized Gaussian:

$$q^*(\theta) = p^*(\theta_{MAP})e^{-\frac{c}{2}(\theta-\theta_{MAP})^2},$$

whose normalization ($p^*(\theta_{MAP})\sqrt{\frac{2\pi}{c}}$) we then use to approximate the normalization of p^* .

Griddy and Laplace, together



Conjugate Prior

- A **conjugate prior** is one which, when multiplied with an appropriate likelihood, gives a posterior with the same functional form as the prior.
- Likelihoods in the exponential family have conjugate priors in the same family
- analytical tractability AND interpretability

- The Beta distribution is conjugate to the Binomial distribution

$$p(p|y) \propto p(y|p)P(p) = Binom(n, y, p) \times Beta(\alpha, \beta)$$

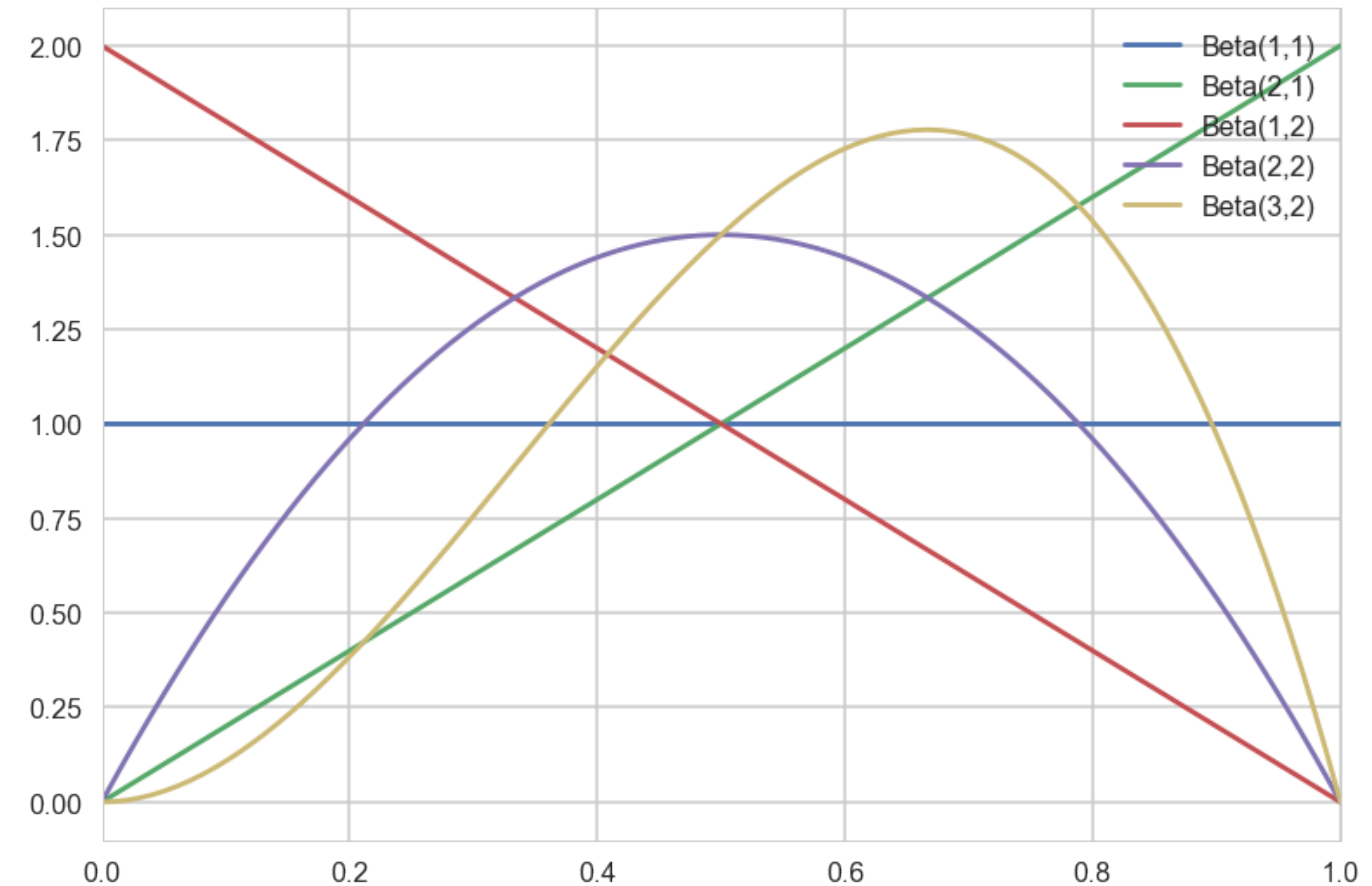
Because of the conjugacy, this turns out to be:

$$Beta(y + \alpha, n - y + \beta)$$

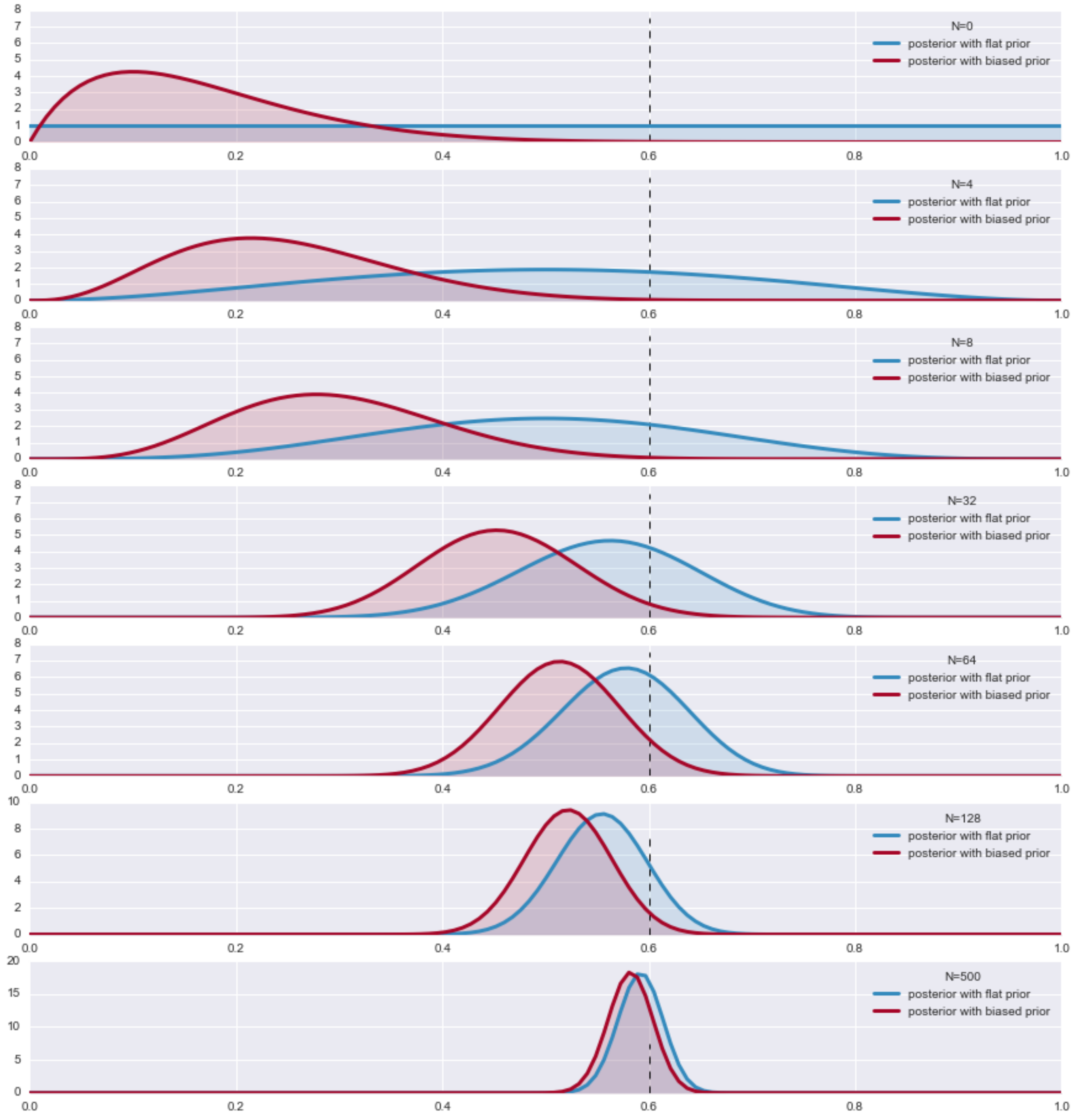
- a $Beta(1, 1)$ prior is equivalent to a uniform distribution.

Priors Regularize

- think of a prior as a regularizer.
- $Bata(1,1)$ is an **uninformative prior**. Here the prior adds one heads and one tails to the actual data, providing some "towards-center" regularization
- especially useful where in a few tosses you got all heads, clearly at odds with your beliefs.
- a $Beta(2,1)$ prior would bias you to more heads



Data overwhelms prior eventually



Bayesian Updating "on-line"

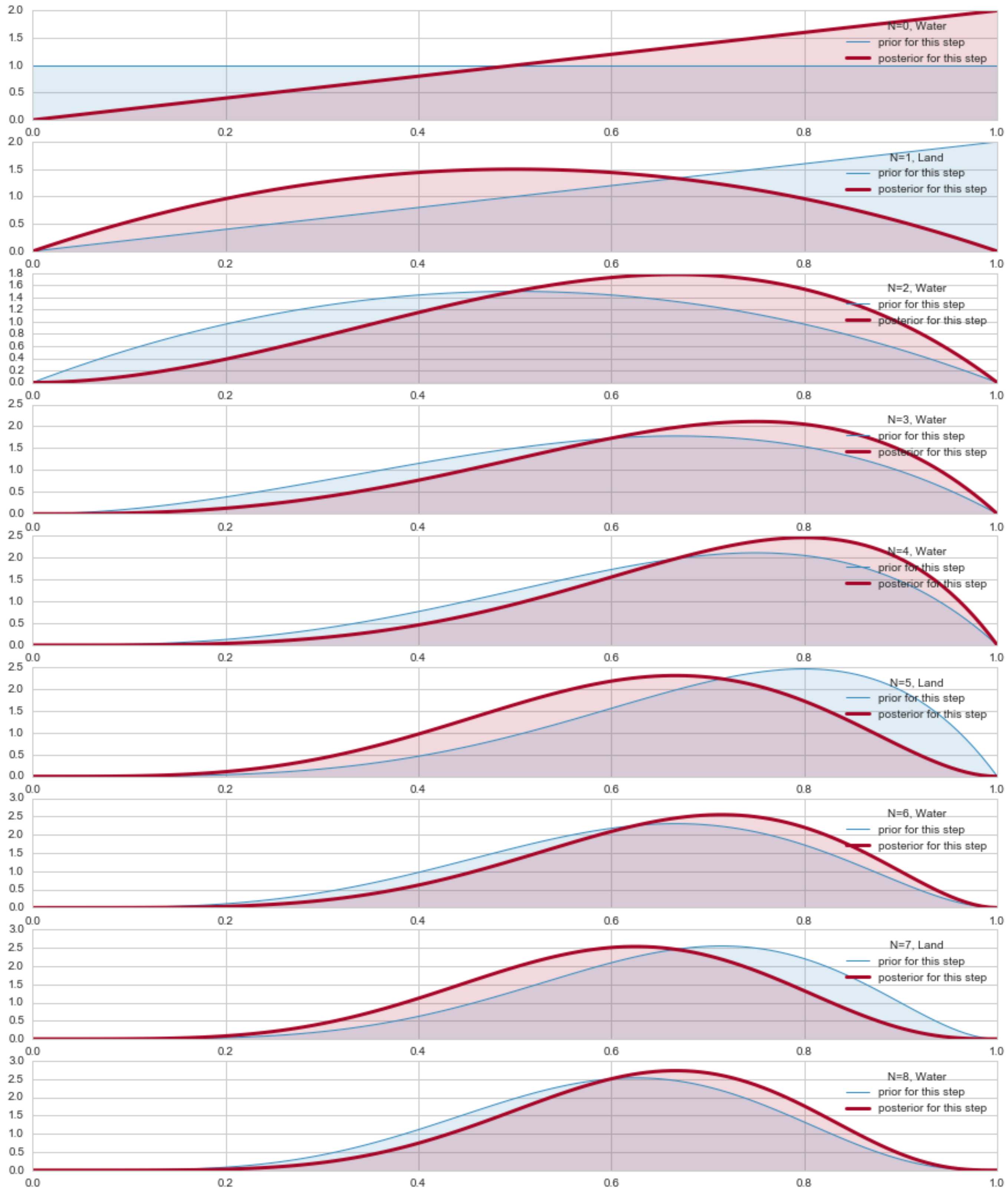
- can update prior to posterior all at once, or one by one
- as each piece of data comes in, you update the prior by multiplying by the one-point likelihood.
- the posterior you get becomes the prior for our next step

$$p(\theta \mid \{y_1, \ldots, y_{n+1}\}) \propto p(\{y_{n+1}\} \mid \theta) \times p(\theta \mid \{y_1, \ldots, y_n\})$$

- the posterior predictive is the distribution of the next data point!

$$p(y_{n+1} \vert \{y_1, \ldots y_n\}) = E_{p(\theta \vert \{y_1, \ldots y_n\})}[p(y_{n+1} \vert \theta)] = \int d\theta \, p(y_{n+1} \vert \theta) p(\theta \vert \{y_1, \ldots y_n\})$$

.



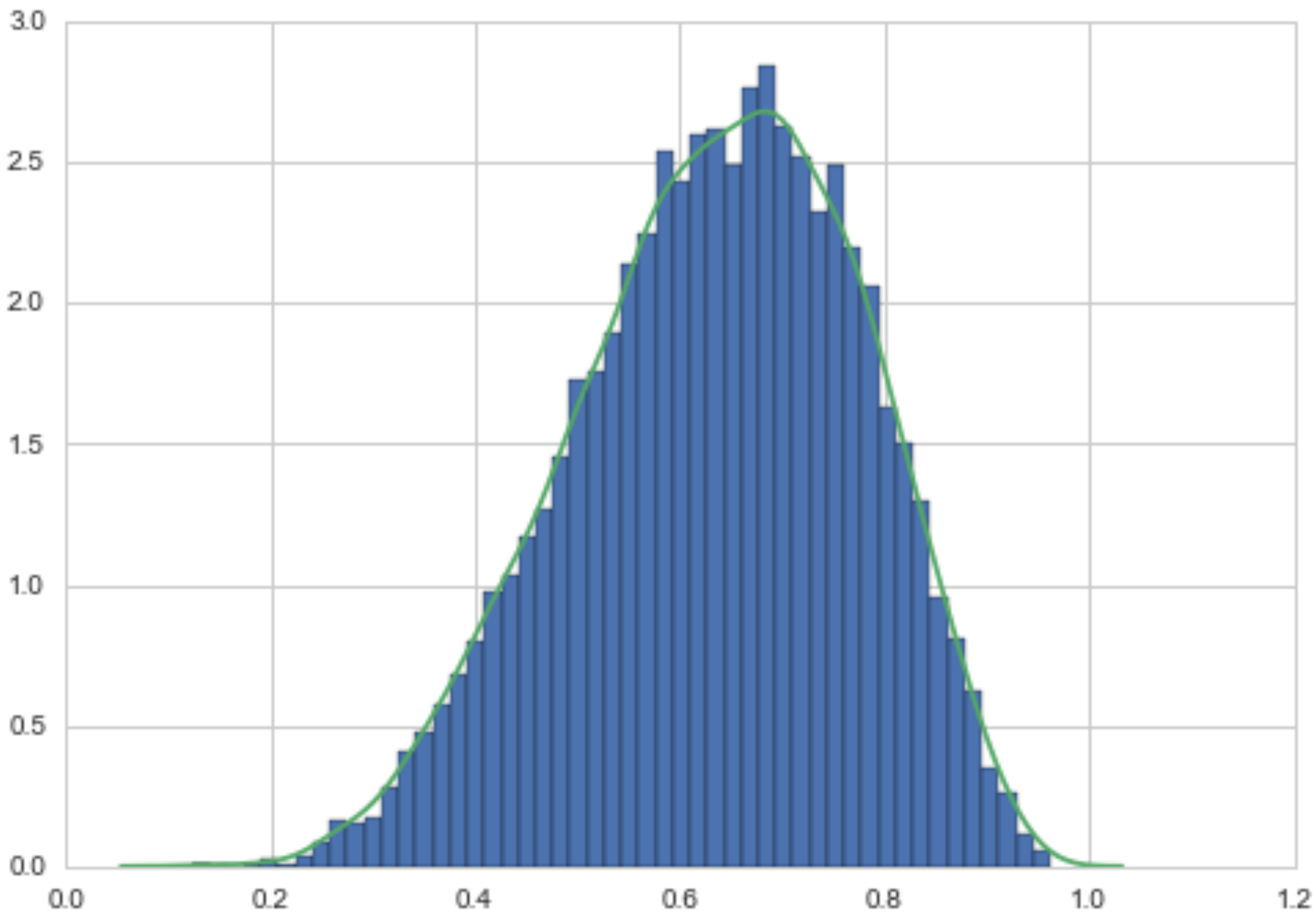
Bayesian Updating of globe

- notice how the posterior shifts left and right depending on new data

At each step:

$$Beta(y + \alpha, n - y + \beta)$$

Posterior properties



- The probability that the amount of water is less than 50%: `np.mean(samples < 0.5) = 0.173`
- **Credible Interval**: amount of probability mass. `np.percentile(samples, [10, 90]) = [ 0.44604094, 0.81516349]`
- `np.mean(samples), np.median(samples) = (0.6378734440335842, 0.6473143052303143)`

Point estimates: MAP

$$\begin{aligned}\theta_{\text{MAP}} &= \arg\max_{\theta} \, p(\theta|D) \\ &= \arg\max_{\theta} \frac{\mathcal{L}\, p(\theta)}{p(D)} \\ &= \arg\max_{\theta} \, \mathcal{L}\, p(\theta)\end{aligned}$$

```
sampleshisto = np.histogram(samples, bins=50)
maxcountindex = np.argmax(sampleshisto[0])
mapvalue = sampleshisto[1][maxcountindex]
print(maxcountindex, mapvalue)
```

```
31 0.662578641304
```

OR Optimize!

Point estimates: mean

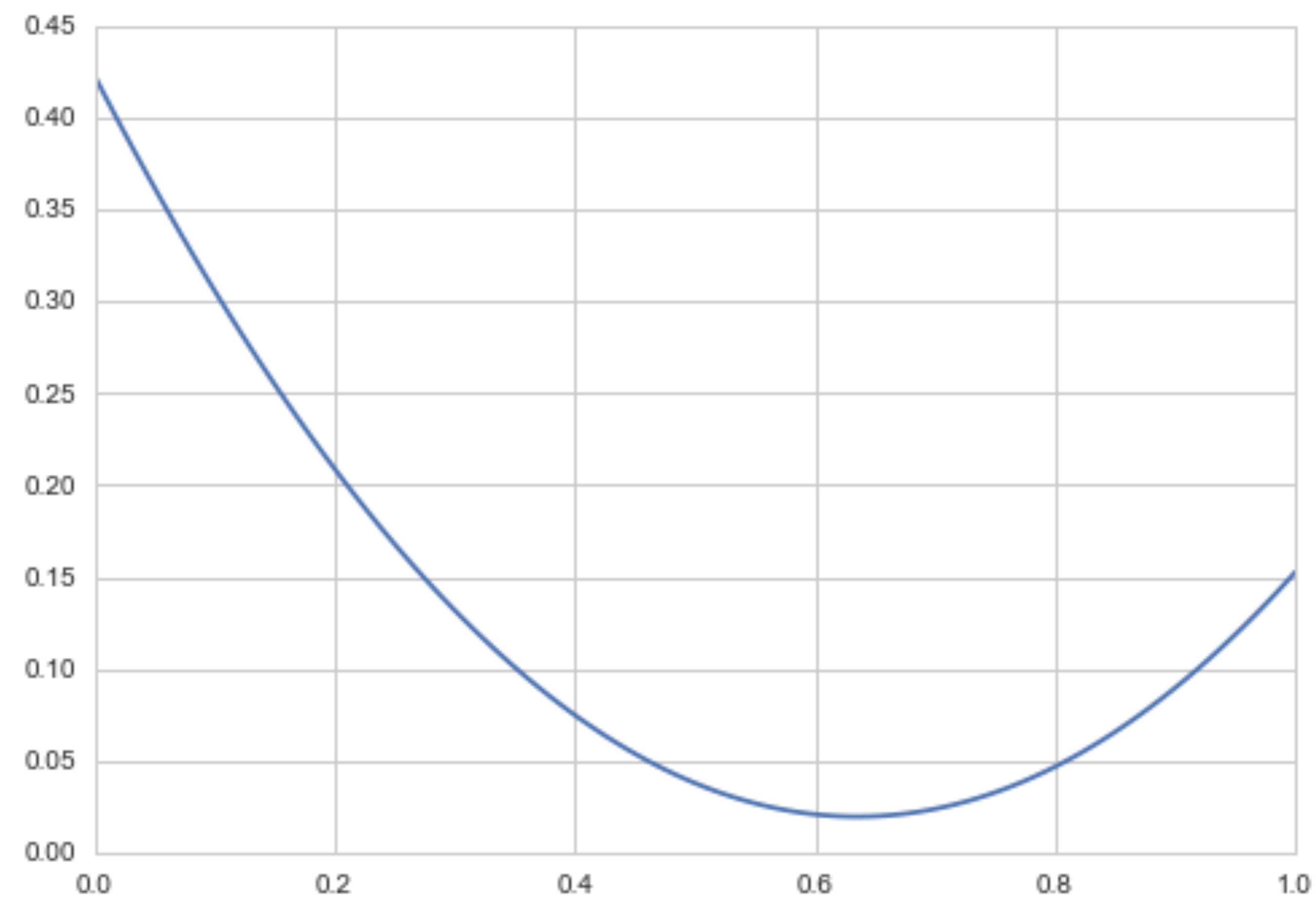
$$R(t) = E_{p(\theta|D)}[(\theta - t)^2] = \int d\theta (\theta - t)^2 p(\theta|D)$$

$$\frac{dR(t)}{dt} = 0 \implies t = \int d\theta \theta \, p(\theta|D)$$

```
mse = [np.mean((xi-samples)**2) for xi in x]
plt.plot(x, mse);
```

Mean is at 0.638.

This is **Decision Theory**.



Posterior predictive for globe tosses

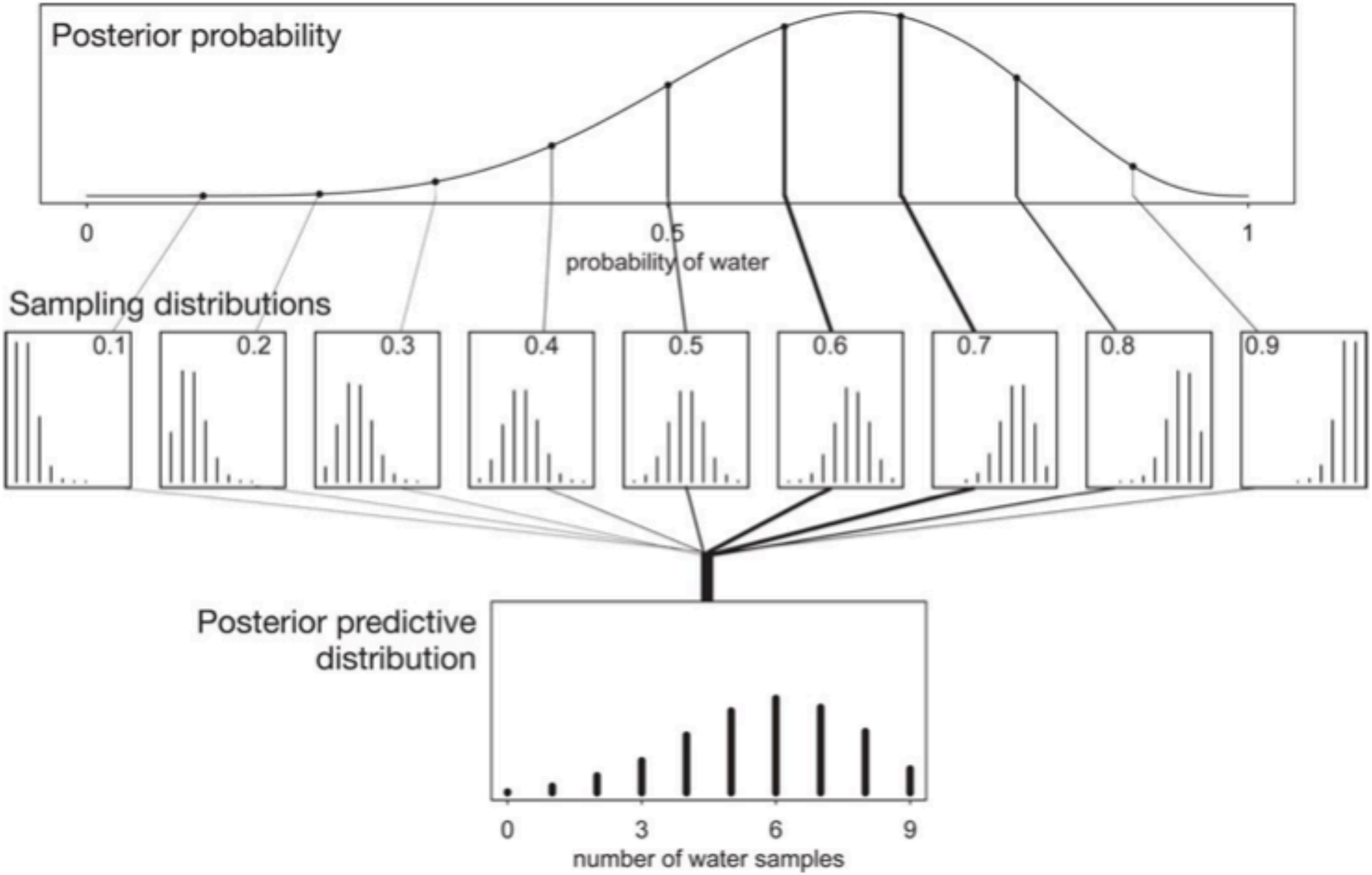
$$p(y^*|D) = \int d\theta p(y^*|\theta)p(\theta|D)$$

Its a Beta-Binomial distribution.

Can use $p(y^*|D) = p(y^*|\theta_{MAP})$ a sampling distribution.

Underestimates spread.

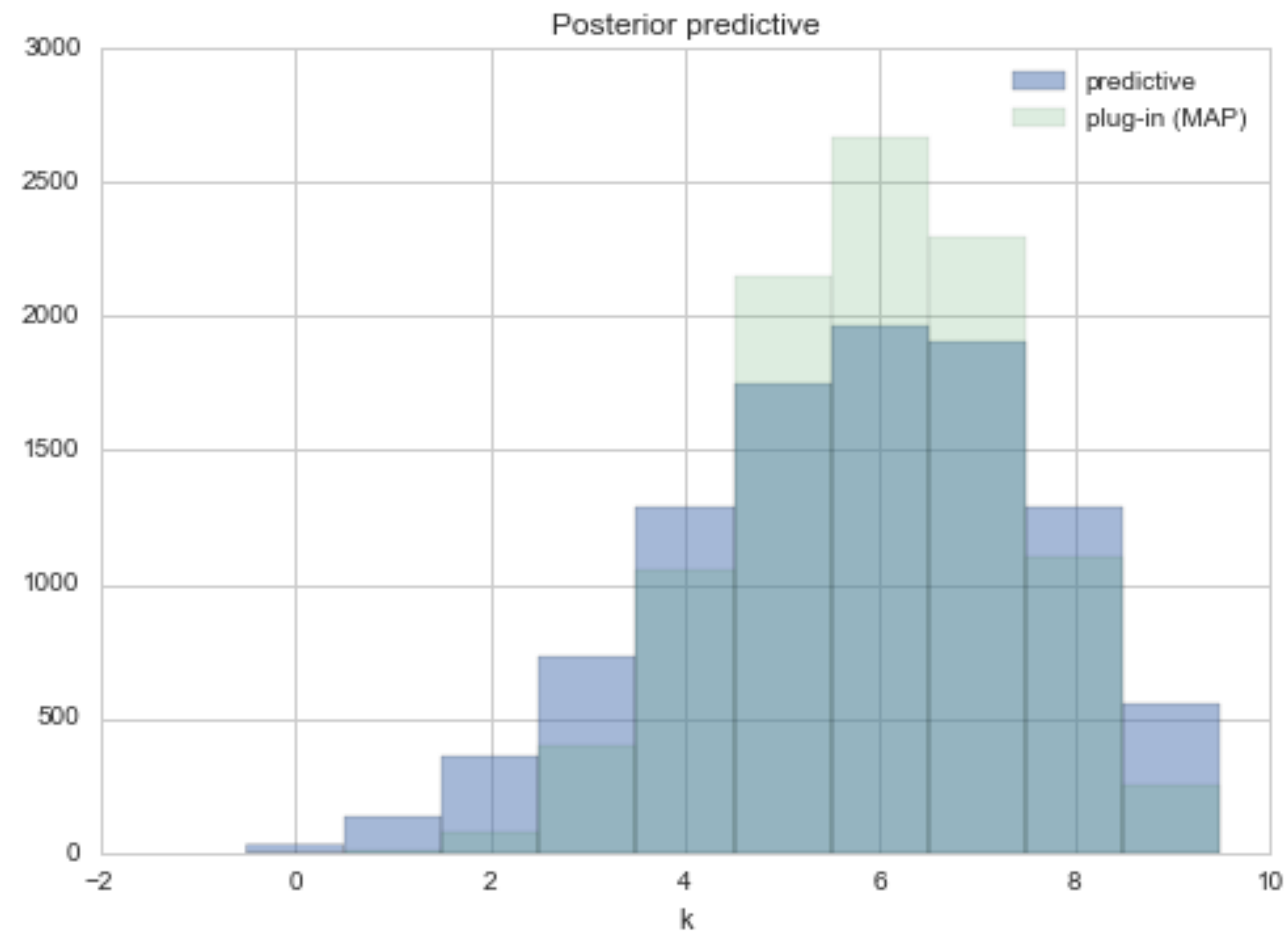
Sample instead.


Posterior probability
0
0.5
1
probability of water
Sampling distributions
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Posterior predictive distribution
0
3
6
9
number of water samples

Posterior predictive from sampling

- draw the thetas from posterior
- then draw y's from the sampling distribution
- and histogram it
- these are draws from joint y, θ

```
postpred = np.random.binomial(n,samples)
```



Replicative Posterior Predictive

$$p(\{y^*\}) = \int p(\{y^*\}|\theta)p(\theta|\mathcal{D})d\theta, \text{ observed data: } \mathcal{D} = \{y\}$$

Replicated Data: $\{y_r\}$: data seen tomorrow if experiment replicated with same model and value of θ producing todays data $\{y\}$.

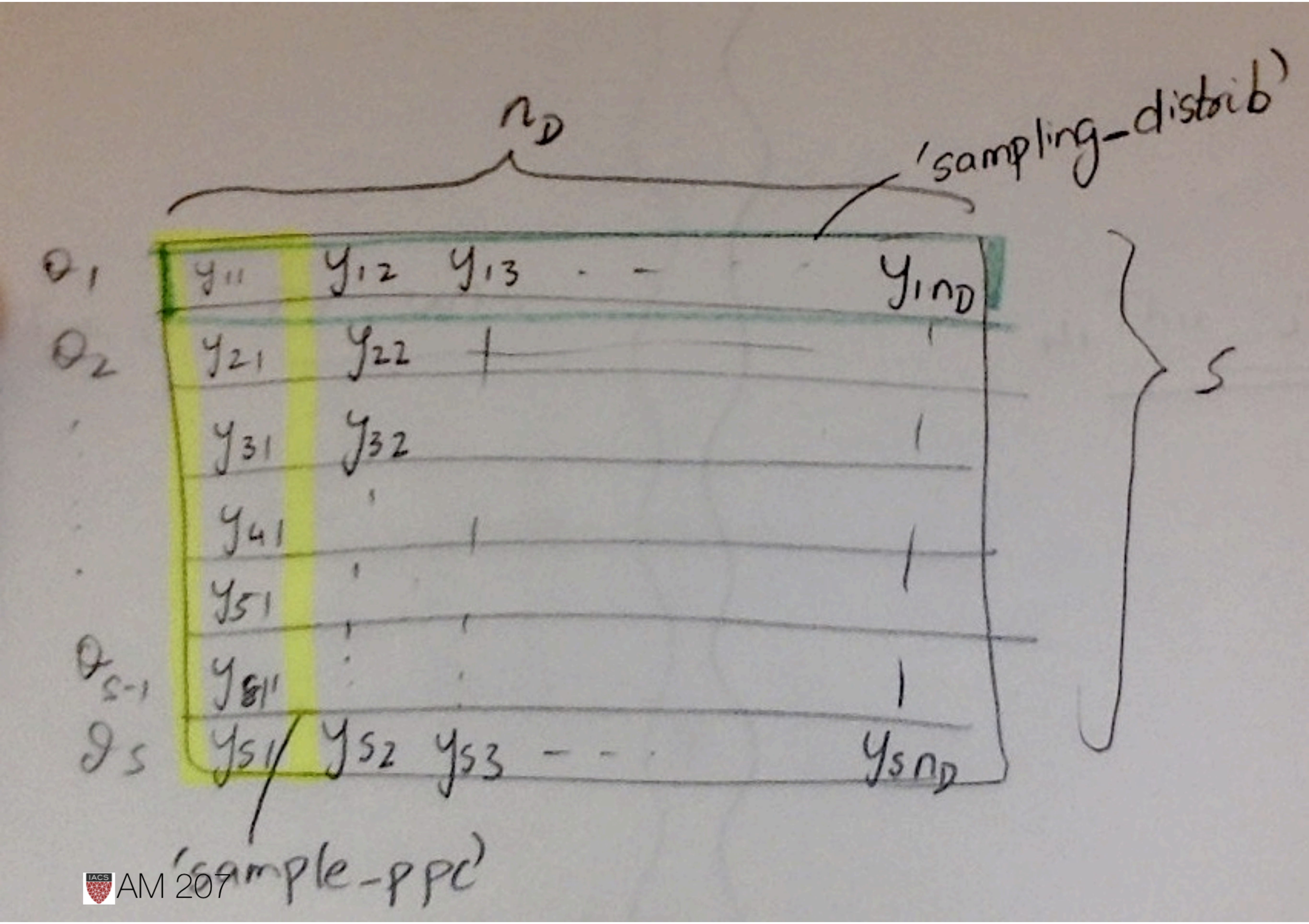
$\{y_r\}$ comes from posterior predictive. The idea is to make as many replications as the size of your dataset.

Another way to sample

```
ppc_rep=np.empty((dataset_size, num_samples))
for i in range(dataset_size):
    ppc_rep[i,:] = distrib.rvs(param=posterior_samples)
```

For each data point, sample using the likelihood(sampling distribution) from S samples of the posterior. Gives an S sized posterior predictive at each "data point".

You can then slice the other way to get a dataset sized posterior-predictive


n_D
'sampling-distrib'
θ_1 y_{11} y_{12} y_{13} y_{1n_D}
θ_2 y_{21} y_{22}
y_{31} y_{32}
y_{41}
y_{51}
θ_{S-1} y_{81}
θ_S y_{S1} y_{S2} y_{S3} y_{Sn_D}
S
'sample-ppc'

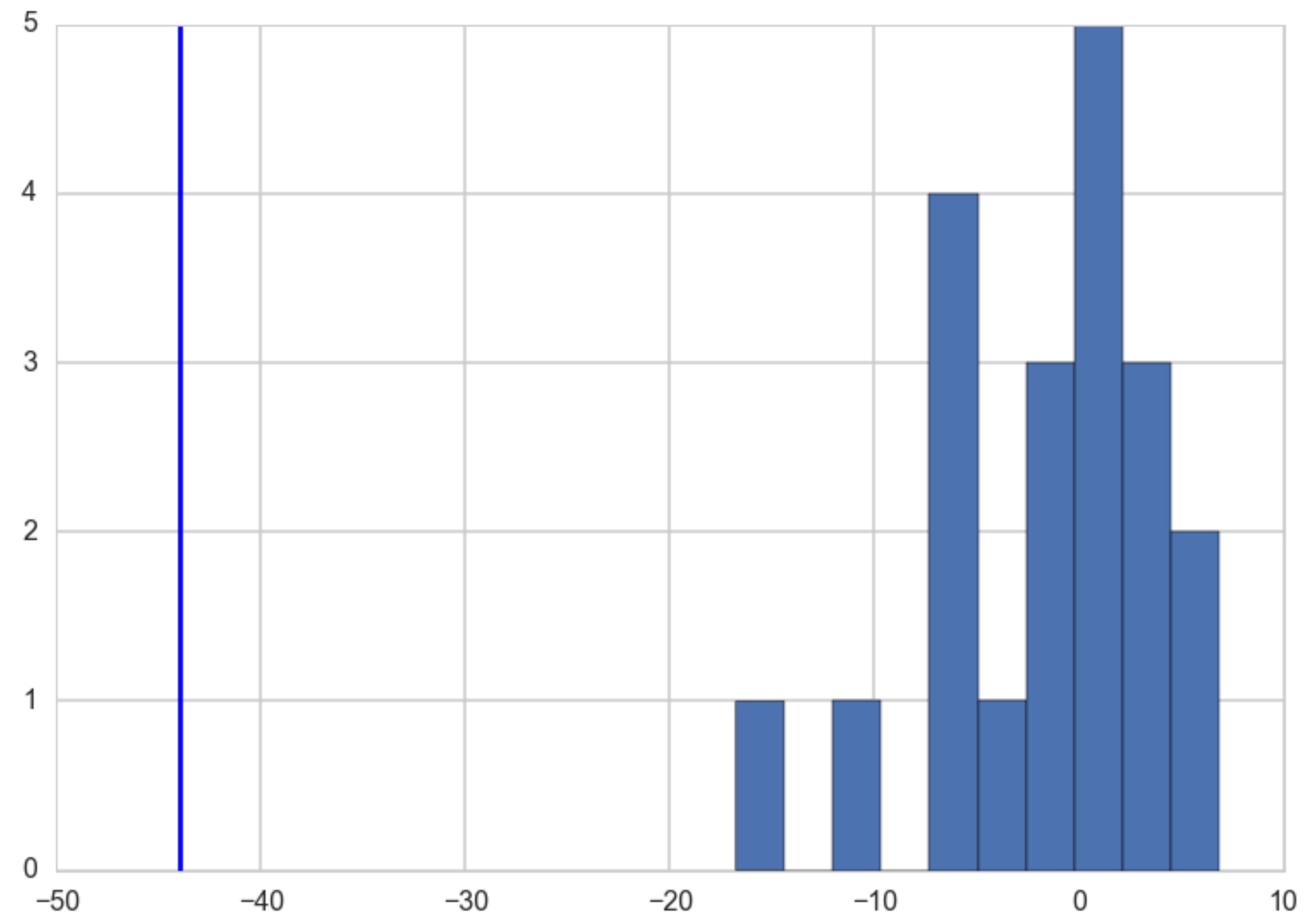
Departure from usual predictive sampling

Sample an entire $\{y_r\}$ at each θ from trace.

This allows to compute distributions from the posterior predictive replications for informal test statistics.

These processes are called **Posterior Predictive Checks**.

Replicative prior predictives are also useful for callibration.



Normal-Normal Model

$$p(\mu, \sigma^2) = p(\mu|\sigma^2)p(\sigma^2)$$

- **fixed σ prior**: $p(\sigma^2) = \delta(\sigma^2 - \sigma_0^2)$
- **non-fixed σ prior**: Choose a functional form that is mildly informative, e.g., normal, half cauchy, half normal. But NOT CONJUGATE. See Murphy
- **μ prior**: Mildly informative normal with prior mean and wide standard deviation

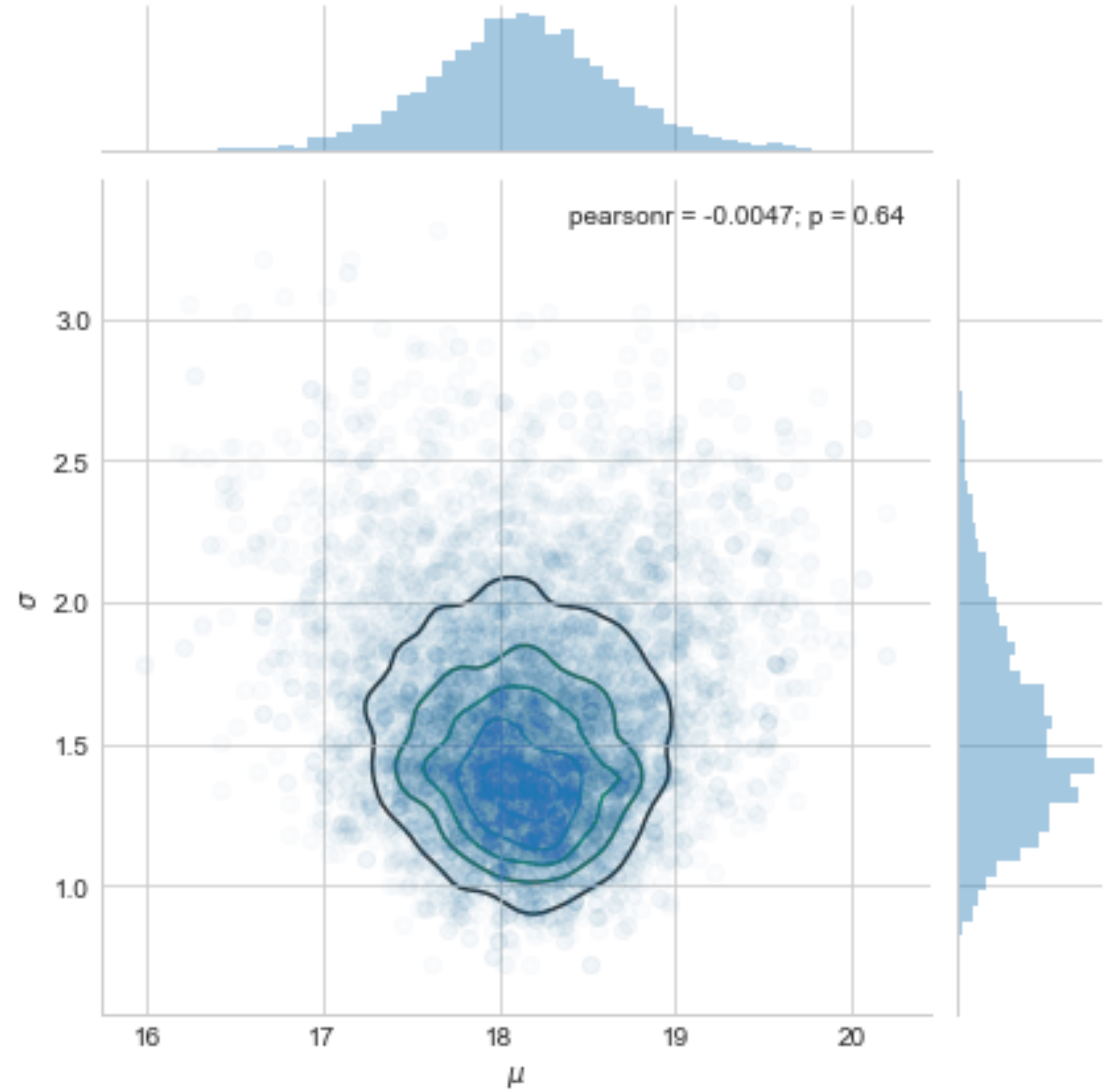
Marginalization

Marginal posterior:

$$p(\theta_1|D) = \int d\theta_{-1} p(\theta|D).$$

```
samps[20000::,:].shape #(10001, 2)
```

```
sns.jointplot(
    pd.Series(samps[20000::,0], name="$\mu$"),
    pd.Series(samps[20000::,1], name="$\sigma$"),
    alpha=0.02)
    .plot_joint(
        sns.kdeplot,
    zorder=0, n_levels=6, alpha=1)
```



Marginals are just 1D histograms

```
plt.hist(samps[20000::,0])
```

Normal-Normal Model

Posterior for a gaussian likelihood:

$$p(\mu, \sigma^2 | y_1, \ldots, y_n, \sigma^2) \propto \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2} \sum (y_i - \mu)^2} p(\mu, \sigma^2)$$

What is the posterior of μ assuming we know σ^2?

Prior for σ^2 is $p(\sigma^2) = \delta(\sigma^2 - \sigma_0^2)$

$$p(\mu|y_1,\ldots,y_n,\sigma^2=\sigma_0^2) \propto p(\mu|\sigma^2=\sigma_0^2)\, e^{-\frac{1}{2\sigma_0^2}\sum(y_i-\mu)^2}$$

The conjugate of the normal is the normal itself.

Say we have the prior

$$p(\mu|\sigma^2) = \exp\left\{-\frac{1}{2\tau^2}(\hat{\mu}-\mu)^2\right\}$$

posterior: $p(\mu|y_1,\ldots,y_n,\sigma^2) \propto \exp\left\{-\frac{a}{2}(\mu-b/a)^2\right\}$

Here

$$a = \frac{1}{\tau^2} + \frac{n}{\sigma_0^2}, \qquad b = \frac{\hat{\mu}}{\tau^2} + \frac{\sum y_i}{\sigma_0^2}$$

Define $\kappa = \sigma^2 / \tau^2$

$$\mu_p = \frac{b}{a} = \frac{\kappa}{\kappa + n}\hat{\mu} + \frac{n}{\kappa + n}\bar{y}$$

which is a weighted average of prior mean and sampling mean.

The variance is

$$\tau_p^2 = \frac{1}{1/\tau^2 + n/\sigma^2}$$

or better

$$\frac{1}{\tau_p^2} = \frac{1}{\tau^2} + \frac{n}{\sigma^2}.$$

as n increases, the data dominates the prior and the posterior mean approaches the data mean, with the posterior distribution narrowing...

The variance is

$$\tau_p^2 = \frac{1}{1/\tau^2 + n/\sigma^2}$$

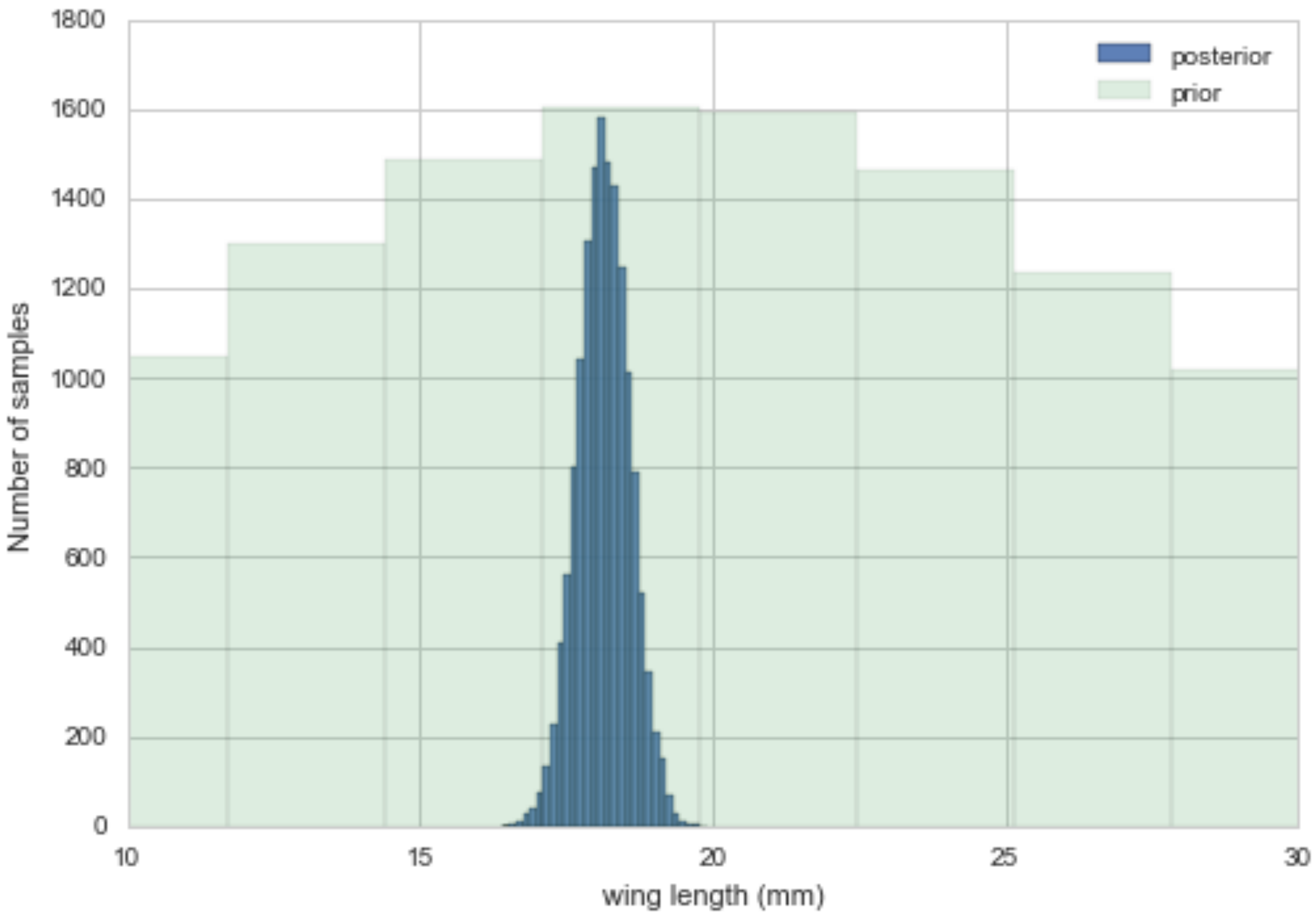
or better

$$\frac{1}{\tau_p^2} = \frac{1}{\tau^2} + \frac{n}{\sigma^2}.$$

as n increases, the data dominates the prior and the posterior mean approaches the data mean, with the posterior distribution narrowing...

Moth wing posterior

```
Y = [16.4, 17.0, 17.2, 17.4,
    18.2, 18.2, 18.2, 19.9, 20.8]
# data mean is 18.1
#Data Quantities
sig = np.std(Y)
# assume that is the value of KNOWN sigma
# (in the likelihood)
mu_data = np.mean(Y)
n = len(Y)
# Prior mean is 19.5
mu_prior = 19.5
# prior std
tau = 10
# plug in formulas
kappa = sig**2 / tau**2
sig_post =np.sqrt(1./( 1./tau**2 + n/sig**2));
# posterior mean
mu_post = kappa / (kappa + n) *mu_prior
    + n/(kappa+n)* mu_data
#samples
N = 15000
theta_prior = np.random.normal(loc=mu_prior,
    scale=tau, size=N);
theta_post = np.random.normal(loc=mu_post,
    scale=sig_post, size=N);
```



Sufficient Statistics and the exponential family

$$p(y_i|\theta) = f(y_i)g(\theta)e^{\phi(\theta)^T u(y_i)}.$$

Likelihood:

$$p(y|\theta) = \left(\prod_{i=1}^{n} f(y_i)\right) g(\theta)^n \ \exp\left(\phi(\theta)\sum_{i=1}^{n} u(y_i)\right)$$

$\sum_{i=1}^{n} u(y_i)$ is said to be a **sufficient statistic** for θ

Poisson Gamma Example

The data consists of 155 women who were 40 years old. We are interested in the birth rate of women with a college degree and women without. We are told that 111 women without college degrees have 217 children, while 44 women with college degrees have 66 children.

Let $Y_{1,1}, \ldots, Y_{n_1,1}$ children for the n_1 women without college degrees, and $Y_{1,2}, \ldots, Y_{n_2,2}$ for n_2 women with college degrees.

Exchangeability

Lets assume that the number of children of a women in any one of these classes can me modelled as coming from ONE birth rate.

The in-class likelihood for these women is invariant to a permutation of variables.

This is really a statement about what is IID and what is not.

It depends on how much knowledge you have...

Poisson likelihood

$$Y_{i,1} \sim Poisson(\theta_1), Y_{i,2} \sim Poisson(\theta_2)$$

$$p(Y_{1,1}, \ldots, Y_{n_1,1}|\theta_1) = \prod_{i=1}^{n_1} p(Y_{i,1}|\theta_1) = \prod_{i=1}^{n_1} \frac{1}{Y_{i,1}!} \theta_1^{Y_{i,1}} e^{-\theta_1}$$

$$= c(Y_{1,1}, \ldots, Y_{n_1,1}) \ (n_1\theta_1)^{\sum Y_{i,1}} e^{-n_1\theta_1} \sim Poisson(n_1\theta_1)$$

$$Y_{1,2}, \ldots, Y_{n_1,2}|\theta_2 \sim Poisson(n_2\theta_2)$$

Posterior

$$c_1(n_1, y_1, \ldots, y_{n_1})\ (n_1\theta_1)^{\sum Y_{i,1}} e^{-n_1\theta_1}\ p(\theta_1)$$
$$\times\ \ c_2(n_2, y_1, \ldots, y_{n_2})\ (n_2\theta_2)^{\sum Y_{i,2}} e^{-n_2\theta_2}\ p(\theta_2)$$

$\sum Y_i$, total number of children in each class of mom,

is **sufficient statistics**

Conjugate prior

Sampling distribution for θ: $p(Y_1, \ldots, y_n|\theta) \sim \theta^{\sum Y_i} e^{-n\theta}$

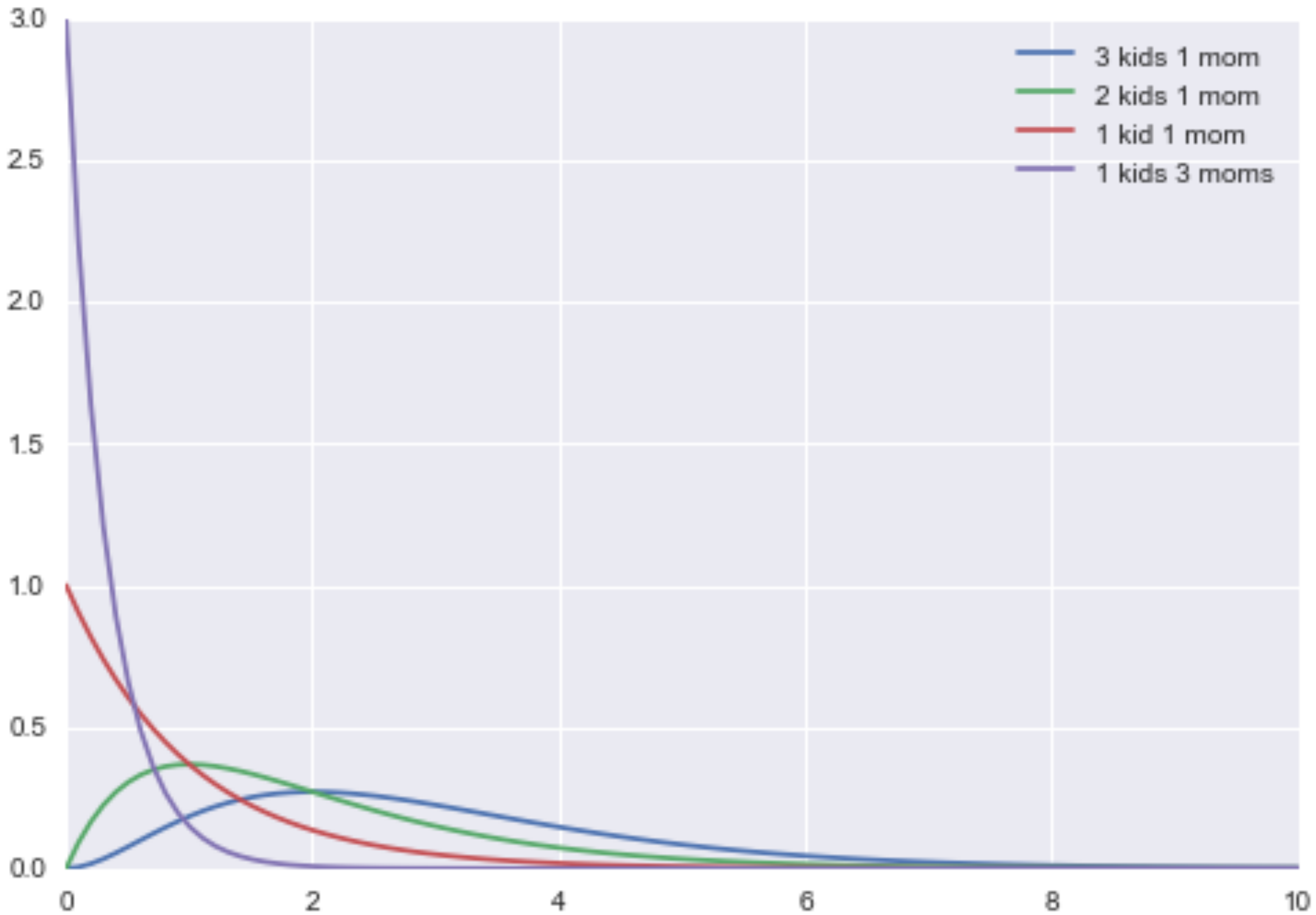
Form is of $Gamma$. In shape-rate parametrization (wikipedia)

$$p(\theta) = \mathrm{Gamma}(\theta, \mathrm{a}, \mathrm{b}) = \frac{\mathrm{b}^{\mathrm{a}}}{\Gamma(\mathrm{a})} \theta^{\mathrm{a}-1} \mathrm{e}^{-\mathrm{b}\theta}$$

Posterior:

$p(\theta|Y_1, \ldots, Y_n) \propto p(Y_1, \ldots, y_n|\theta)p(\theta) \sim \mathrm{Gamma}(\theta, \mathrm{a} + \sum \mathrm{Y_i}, \mathrm{b} + \mathrm{n})$

Priors and Posteriors



We choose 2,1 as our prior.

$$p(\theta_1 | n_1, \sum_i^{n_1} Y_{i,1}) \sim \text{Gamma}(\theta_1, 219, 112)$$

$$p(\theta_2 | n_2, \sum_i^{n_2} Y_{i,2}) \sim \text{Gamma}(\theta_2, 68, 45)$$

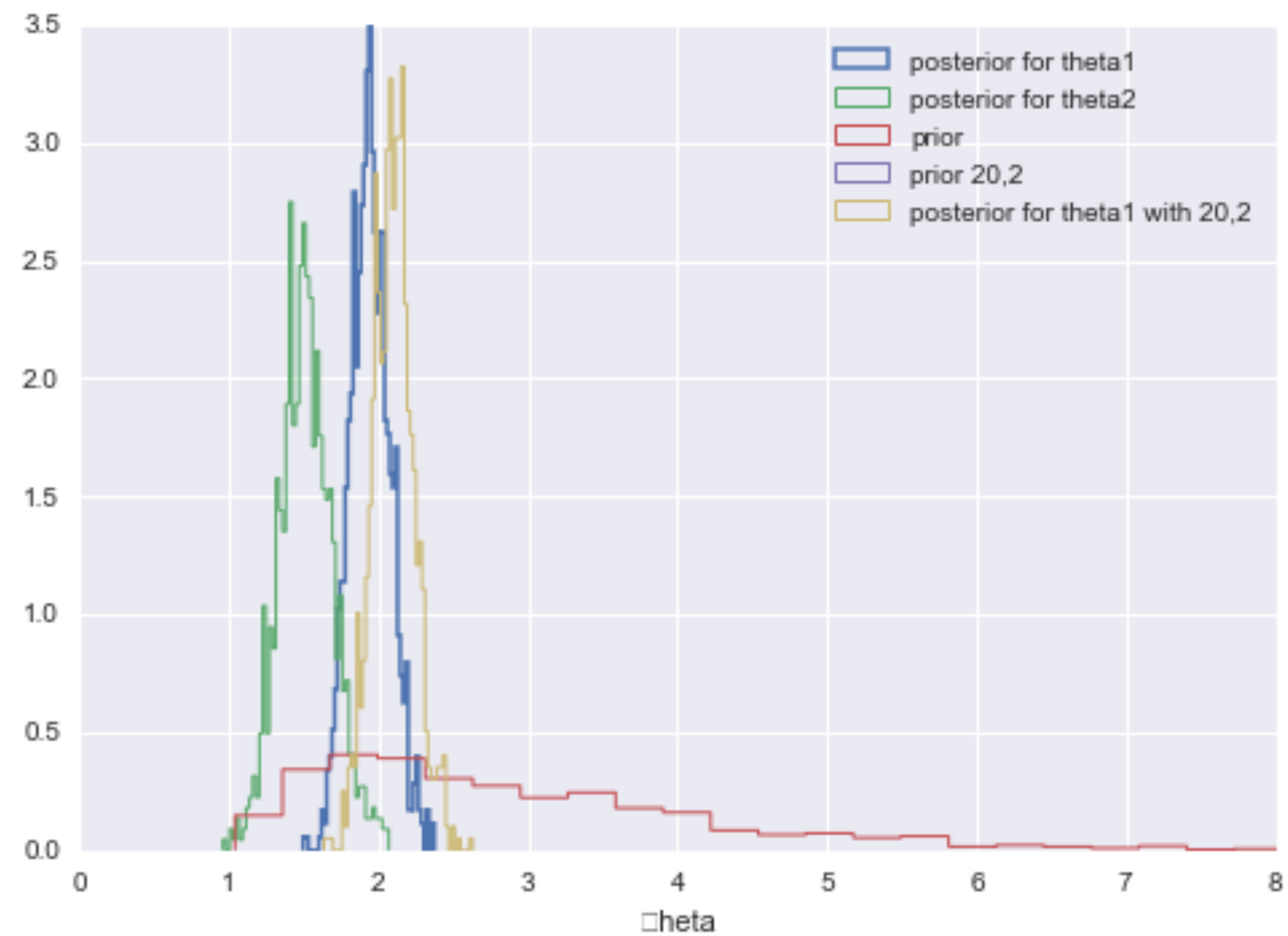
Prior mean, variance:
$E[\theta] = a/b, var[\theta] = a/b^2$.

Posteriors

$$E[\theta] = (a + \sum y_i)/(b + N)$$
$$var[\theta] = (a + \sum y_i)/(b + N)^2.$$

```
np.mean(theta1),
np.var(theta1) =
(1.9516881521791478,
0.018527204185785785)
```

```
np.mean(theta2),
np.var(theta2) =
(1.5037252100213609,
0.034220717257786061)
```



Posterior Predictives

$$p(y^*|D) = \int d\theta p(y^*|\theta)p(\theta|D)$$

Sampling makes it easy:

```
postpred1 = poisson.rvs(theta1)
postpred2 = poisson.rvs(theta2)
```

Negative Binomial:

$$E[y^*] = \frac{(a + \sum y_i)}{(b + N)}$$

$$var[y^*] = \frac{(a + \sum y_i)}{(b + N)^2}(N + b + 1).$$

But see width:

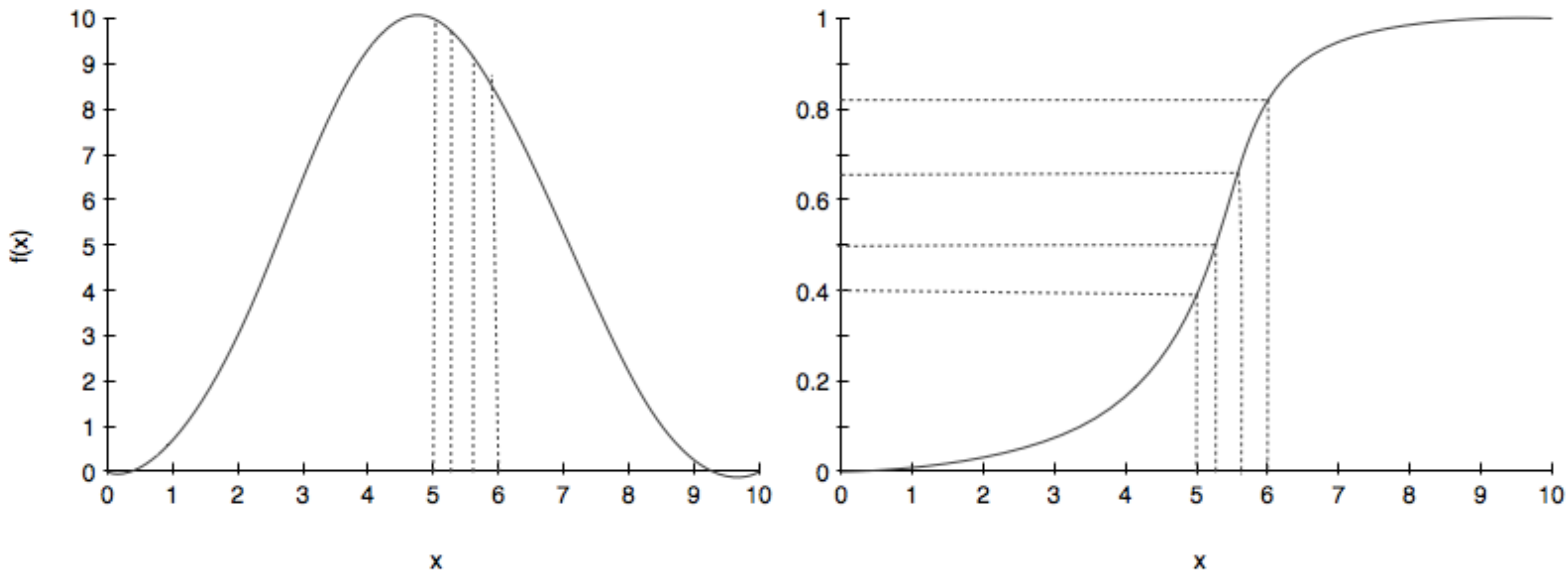
```
np.mean(postpred1),
np.var(postpred1)=(1.976,
1.8554239999999997)
```

Posterior predictive smears out posterior error with sampling distribution

Ok. We need Samples

- to compute expectations, integrals and do statistics, we need samples
- we start that journey today
- inverse transform
- rejection sampling
- importance sampling: a direct, low-variance way to do integrals and expectations

Inverse transform



algorithm

The CDF F must be invertible!

1. get a uniform sample u from $Unif(0, 1)$
2. solve for x yielding a new equation $x = F^{-1}(u)$ where F is the CDF of the distribution we desire.
3. repeat.

Why does it work?

$$F^{-1}(u) = \text{smallest x such that } F(x) >= u$$

What distribution does random variable $y = F^{-1}(u)$ follow?

The CDF of y is $p(y <= x)$. Since F is monotonic:

$$p(y <= x) = p(F(y) <= F(x)) = p(u <= F(x)) = F(x)$$

F is the CDF of y, thus f is the pdf.

Example: exponential

pdf: $f(x) = \frac{1}{\lambda} e^{-x/\lambda}$ for $x \geq 0$ and $f(x) = 0$ otherwise.

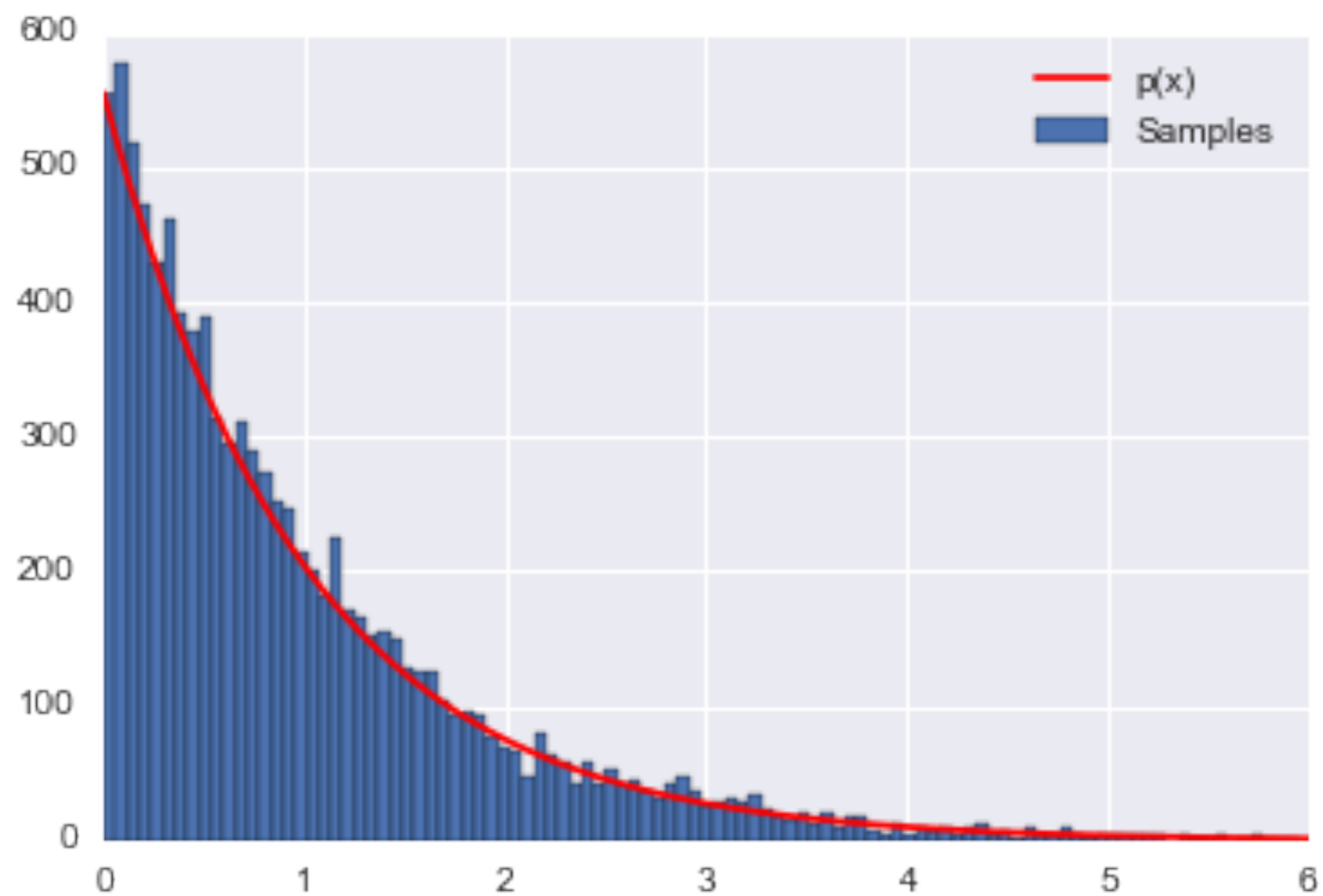
$$u = \int_0^x \frac{1}{\lambda} e^{-x'/\lambda} dx' = 1 - e^{-x/\lambda}$$

Solving for x

$$x = -\lambda \ln(1-u)$$

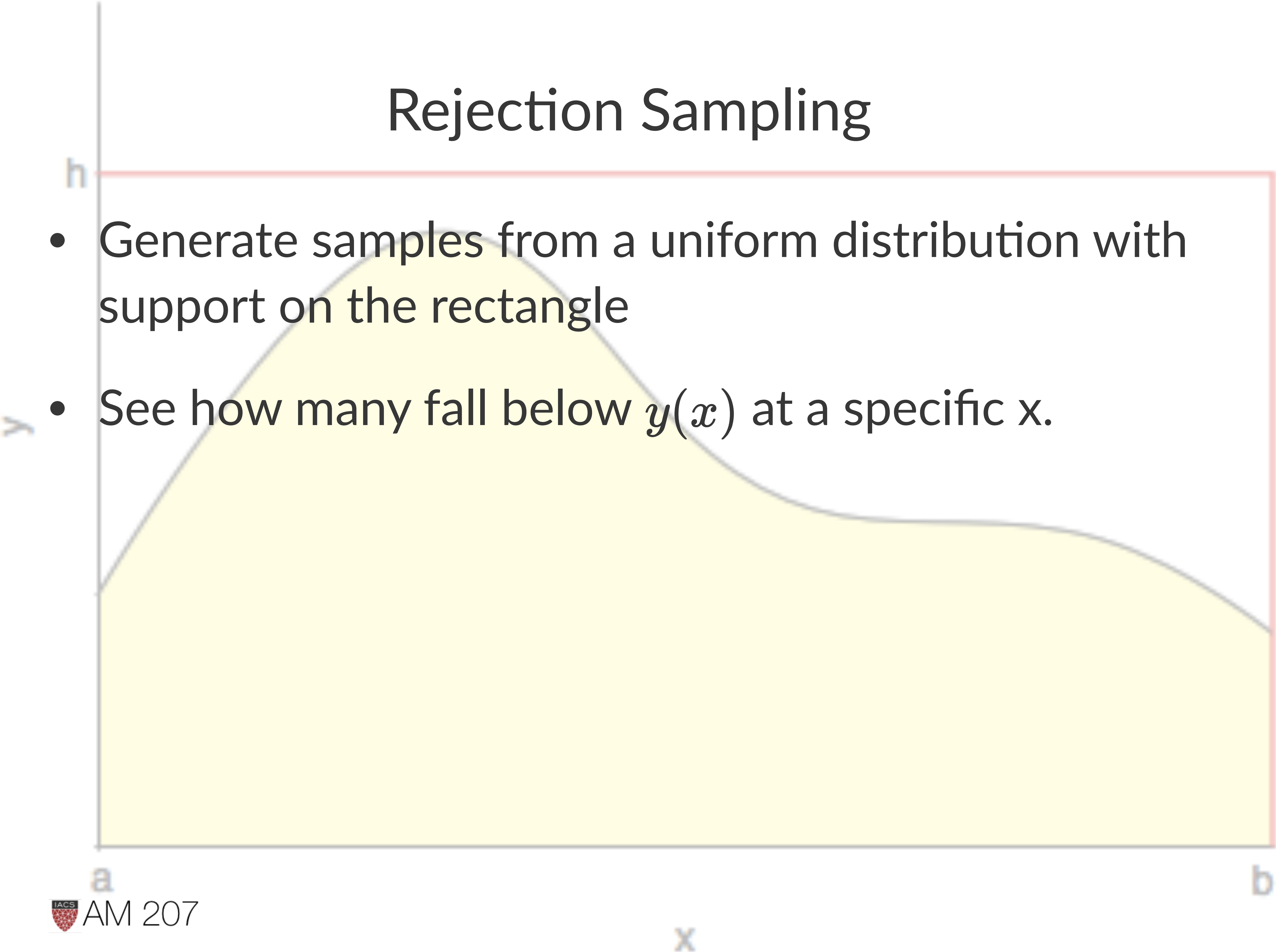
code

```
p = lambda x: np.exp(-x)
CDF = lambda x: 1-np.exp(-x)
invCDF = lambda r: -np.log(1-r) # invert the CDF
xmin = 0 # the lower limit of our domain
xmax = 6 # the upper limit of our domain
rmin = CDF(xmin)
rmax = CDF(xmax)
N = 10000
# generate uniform samples in our range then invert the CDF
# to get samples of our target distribution
R = np.random.uniform(rmin, rmax, N)
X = invCDF(R)
hinfo = np.histogram(X,100)
plt.hist(X,bins=100, label=u'Samples');
# plot our (normalized) function
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*p(xvals), 'r', label=u'p(x)')
plt.legend()
```



Rejection Sampling

- Generate samples from a uniform distribution with support on the rectangle
- See how many fall below $y(x)$ at a specific x.



Algorithm

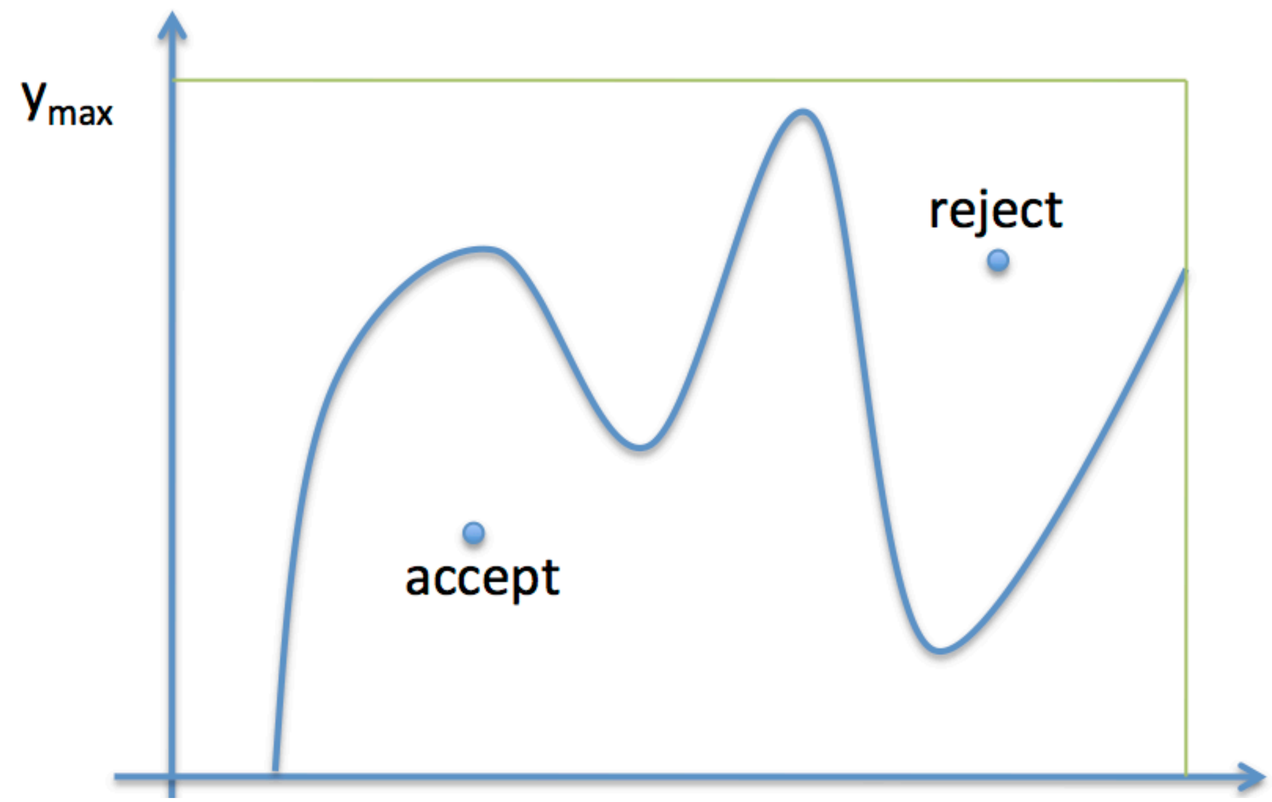
1. Draw x uniformly from $[x_{min}, x_{max}]$

2. Draw y uniformly from $[0, y_{max}]$

3. if $y < f(x)$, accept the sample

4. otherwise reject it

5. repeat



example

```python
P = lambda x: np.exp(-x)
xmin = 0 # the lower limit of our domain
xmax = 10 # the upper limit of our domain
ymax = 1
#you might have to do an optimization to find this.
N = 10000 # the total of samples we wish to generate
accepted = 0 # the number of accepted samples
samples = np.zeros(N)
count = 0 # the total count of proposals

while (accepted < N):
    # pick a uniform number on [xmin, xmax) (e.g. 0...10)
    x = np.random.uniform(xmin, xmax)
    # pick a uniform number on [0, ymax)
    y = np.random.uniform(0,ymax)
    # Do the accept/reject comparison
    if y < P(x):
        samples[accepted] = x
        accepted += 1

    count +=1

print("Count",count, "Accepted", accepted)
hinfo = np.histogram(samples,30)
plt.hist(samples,bins=30, label=u'Samples');
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*P(xvals), 'r', label=u'P(x)')
plt.legend()
```


```
Count 100294 Accepted 10000
```

problems

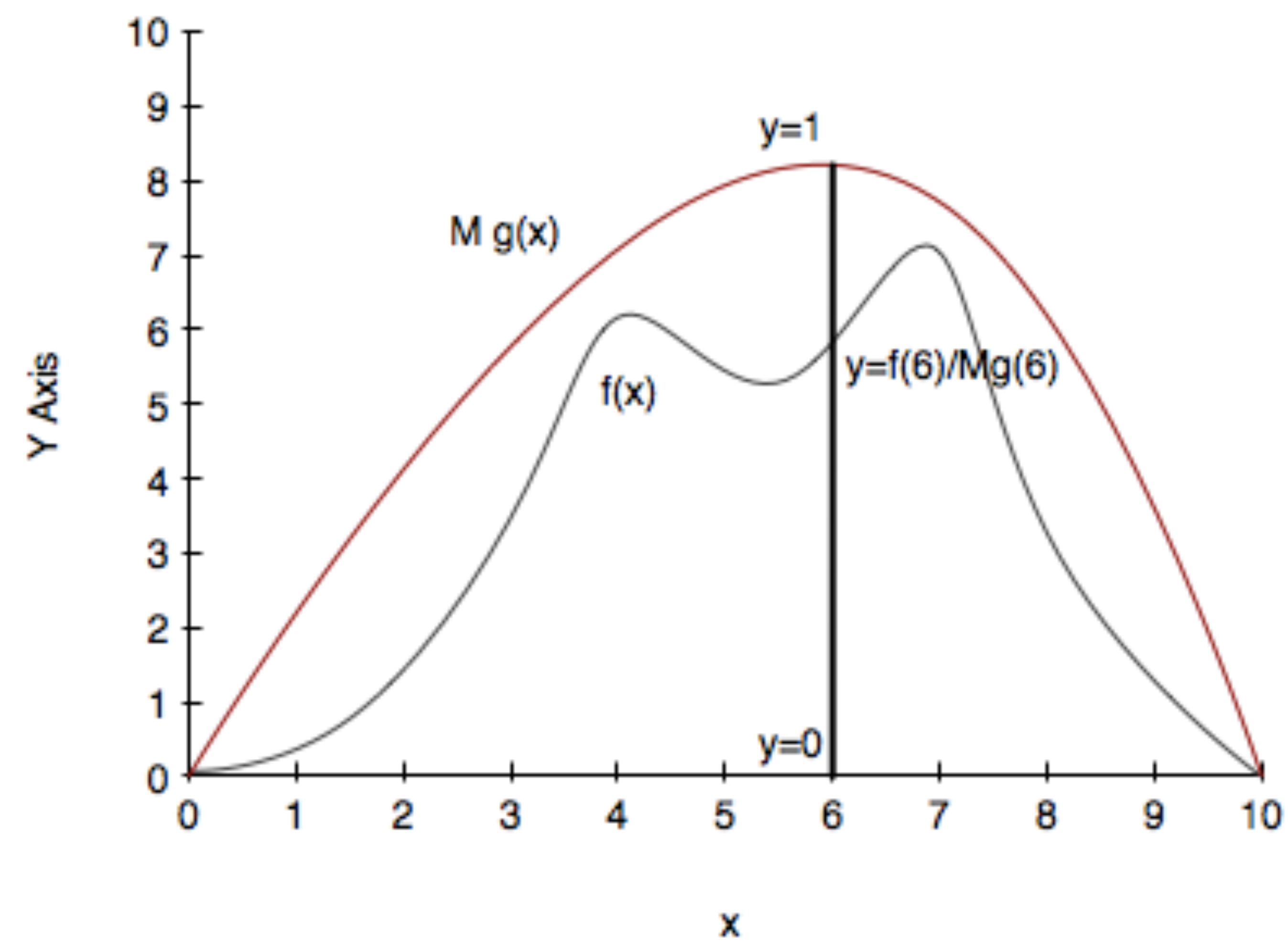
- determining the supremum may be costly
- the functional form may be complex for comparison
- even if you find a tight bound for the supremum, basic rejection sampling is very inefficient: **low acceptance probability**
- infinite support

Variance Reduction

Rejection on steroids

Introduce a **proposal density** $g(x)$.

- $g(x)$ is easy to sample from and (calculate the pdf)
- Some M exists so that $M\,g(x) > f(x)$ in your entire domain of interest
- ideally $g(x)$ will be somewhat close to f
- optimal value for M is the supremum over your domain



Algorithm

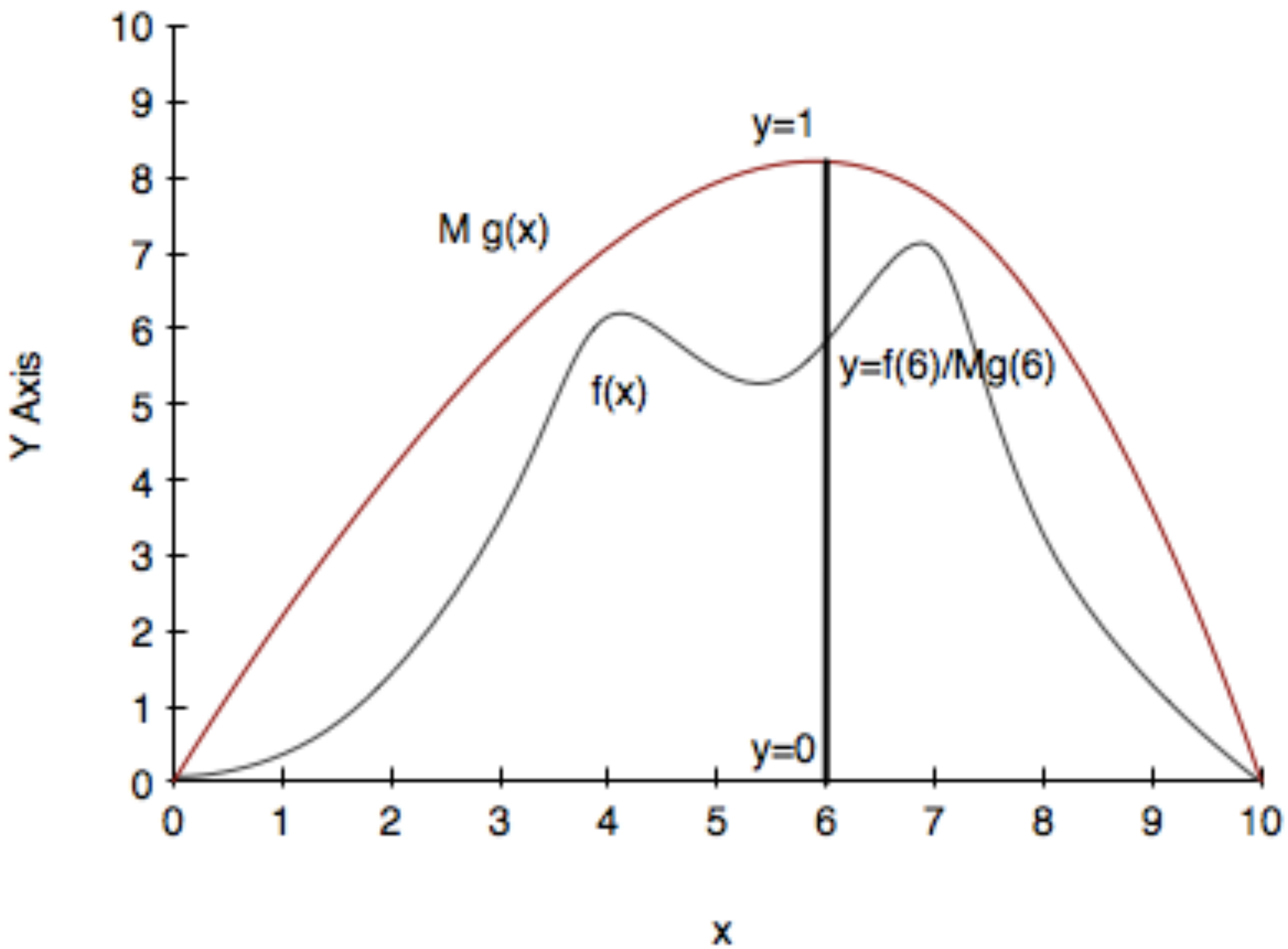
1. Draw x from your proposal distribution $g(x)$

2. Draw y uniformly from [0,1]

3. if $y < f(x)/M\,g(x)$, accept the sample

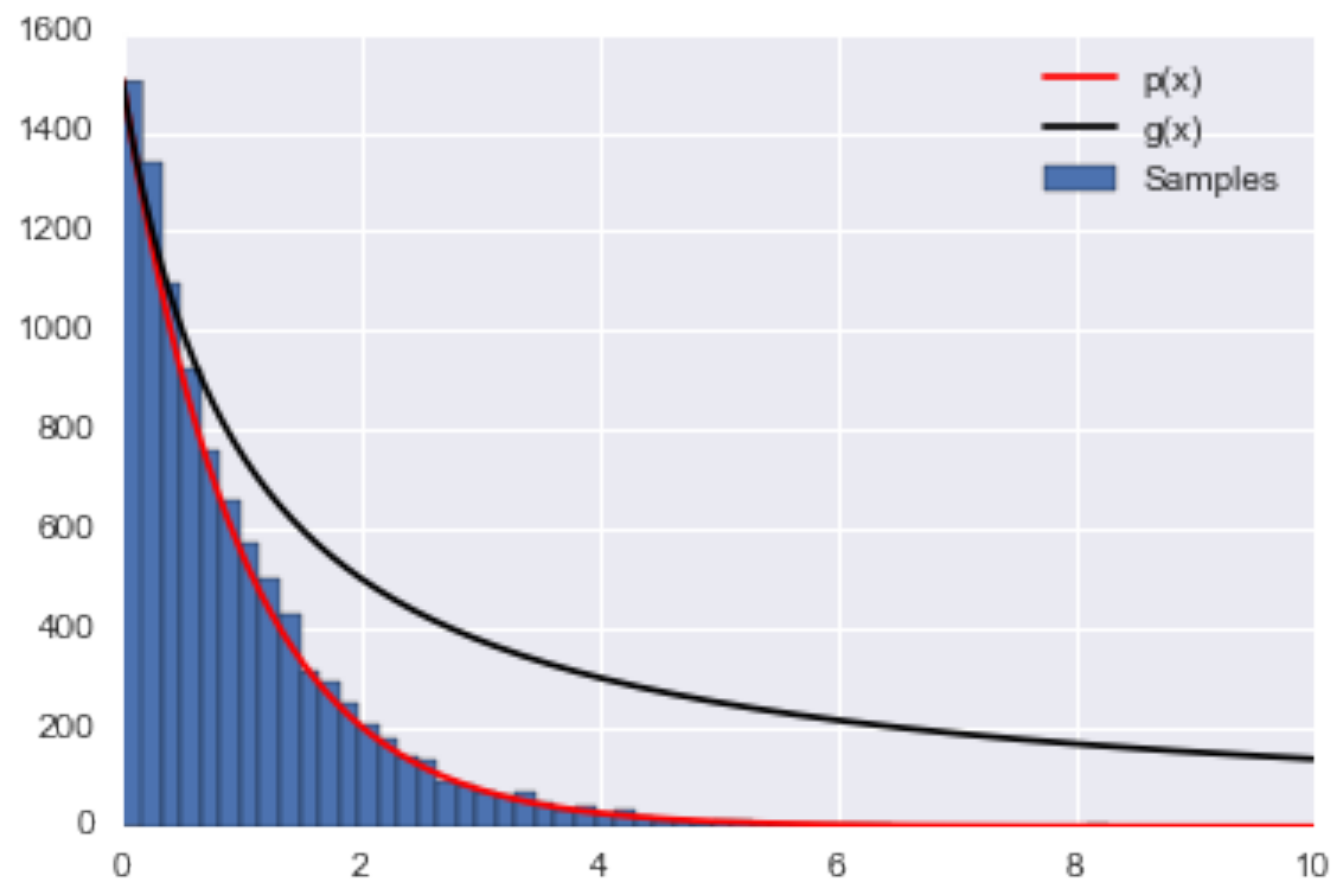
4. otherwise reject it

5. repeat



Example

```
p = lambda x: np.exp(-x)  # our distribution
g = lambda x: 1/(x+1)  # our proposal pdf (we're thus choosing M to be 1)
invCDFg = lambda x: np.log(x +1) # generates our proposal using inverse sampling
xmin = 0 # the lower limit of our domain
xmax = 10 # the upper limit of our domain
# range limits for inverse sampling
umin = invCDFg(xmin)
umax = invCDFg(xmax)
N = 10000 # the total of samples we wish to generate
accepted = 0 # the number of accepted samples
samples = np.zeros(N)
count = 0 # the total count of proposals

while (accepted < N):

    # Sample from g using inverse sampling
    u = np.random.uniform(umin, umax)
    xproposal = np.exp(u) - 1

    # pick a uniform number on [0, 1)
    y = np.random.uniform(0,1)

    # Do the accept/reject comparison
    if y < p(xproposal)/g(xproposal):
        samples[accepted] = xproposal
        accepted += 1

    count +=1

print("Count", count, "Accepted", accepted)
# get the histogram info
hinfo = np.histogram(samples,50)
plt.hist(samples,bins=50, label=u'Samples');
xvals=np.linspace(xmin, xmax, 1000)
plt.plot(xvals, hinfo[0][0]*p(xvals), 'r', label=u'p(x)')
plt.plot(xvals, hinfo[0][0]*g(xvals), 'k', label=u'g(x)')
plt.legend()
```




```
Count 23809 Accepted 10000
```

Importance sampling

The basic idea behind importance sampling is that we want to draw more samples where $h(x)$, a function whose integral or expectation we desire, is large. In the case we are doing an expectation, it would indeed be even better to draw more samples where $h(x)f(x)$ is large, where $f(x)$ is the pdf we are calculating the integral with respect to.

Unlike rejection sampling we use all samples!!

$$E_f[h] = \int_V f(x)h(x)dx.$$

Choosing a proposal distribution $g(x)$:

$$E_f[h] = \int h(x)g(x)\frac{f(x)}{g(x)}dV$$

$$E_f[h] = \lim_{N\to\infty}\frac{1}{N}\sum_{x_i\sim g(.)} h(x_i)\frac{f(x_i)}{g(x_i)}$$

If $w(x_i) = f(x_i)/g(x_i)$:

$$E_f[h] = \lim_{N\to\infty}\frac{1}{N}\sum_{x_i\sim g(.)} w(x_i)h(x_i)$$