Lecture 26

RECAP

BASIC STATS AND SAMPLING

Law of Large numbers, LOTUS, MC

Let $x_1, x_2, \ldots, x_n$ be a sequence of IID values from random variable X, which has finite mean μ. Let:

$$S_n = \frac{1}{n}\sum_{i=1}^{n} x_i \text{, then } S_n \to \mu \, as \, n \to \infty.$$

- Expectations become sample averages. Convergence for large N.

$$\mathbb{E}[g(X)] = \int g(x) f(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} g(x_i)$$

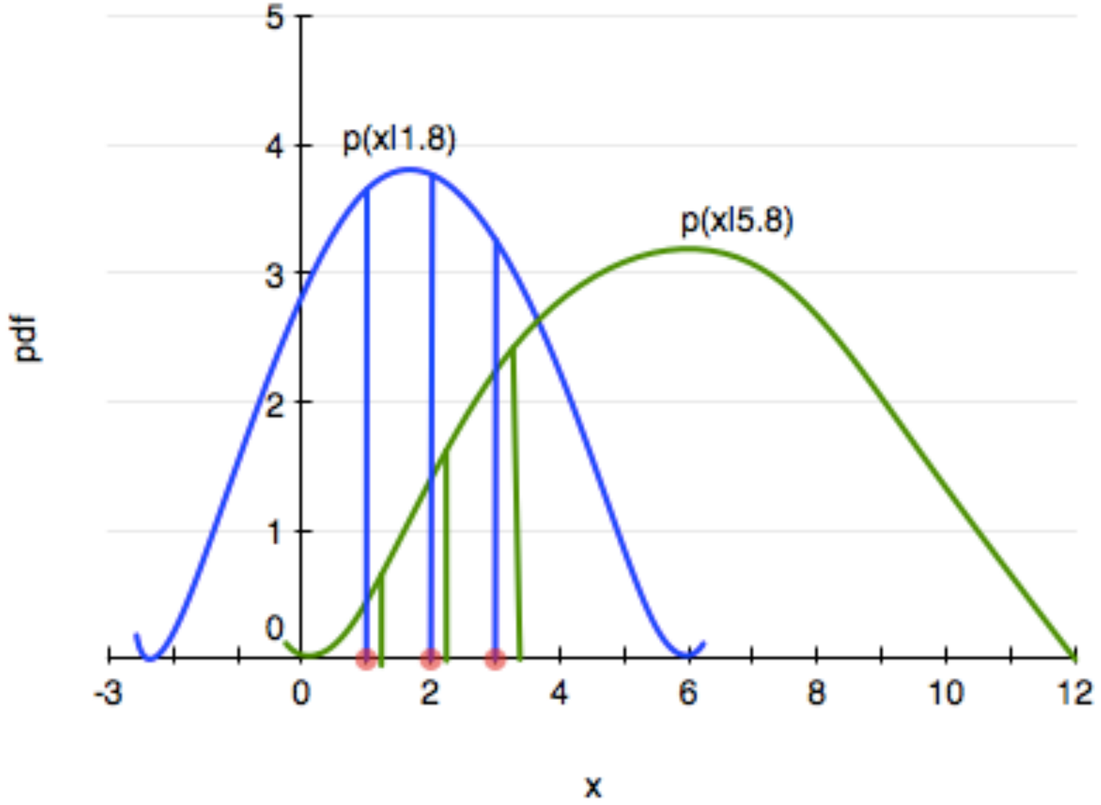
- allows for monte-carlo

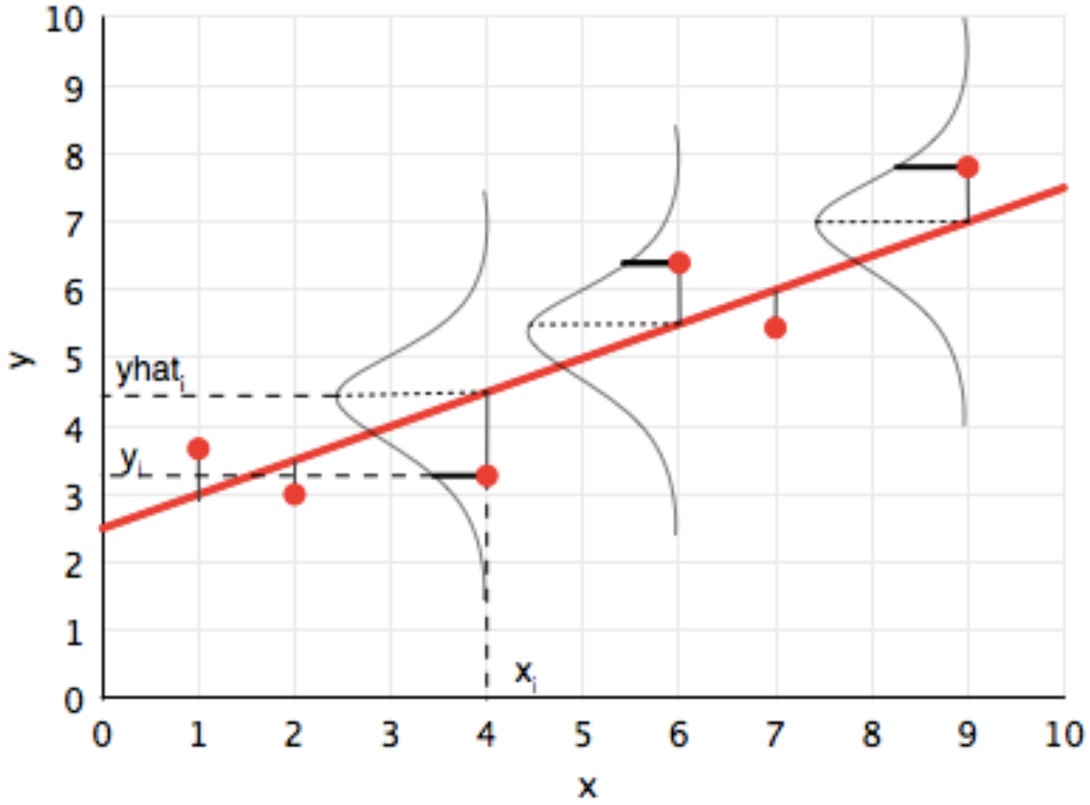
Frequentist Statistics

"data is a **sample** from an existing **population**"

- data is stochastic, variable; parameters fixed
- apply an estimator F to the sample data D, so $\hat{\mu} = F(D)$.
- If your model describes the true generating process for the data (not mis-specified), then there is some true μ^*.
- The best we can do is to estimate $\hat{\mu}$.

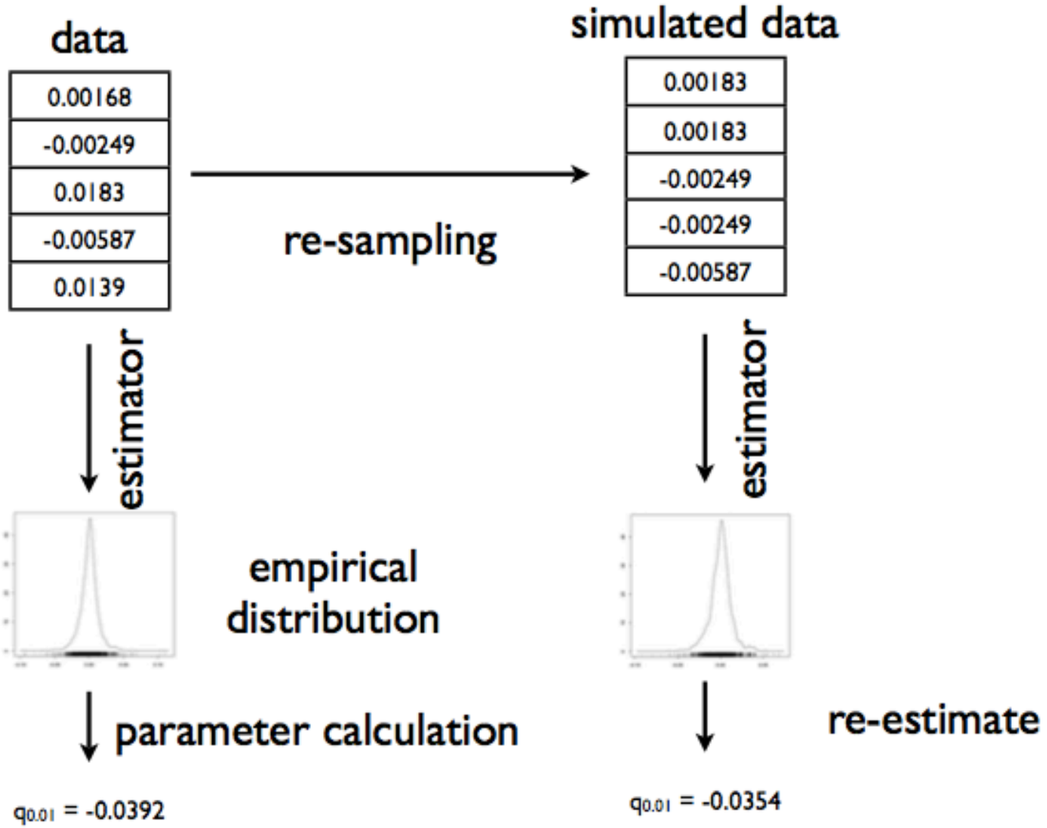
MLE


pdf
p(xl1.8)
p(xl5.8)
x
5
4
3
2
1
0
-3
0
2
4
6
8
10
12


y
x
yhat$_i$
y$_i$
x$_i$
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10

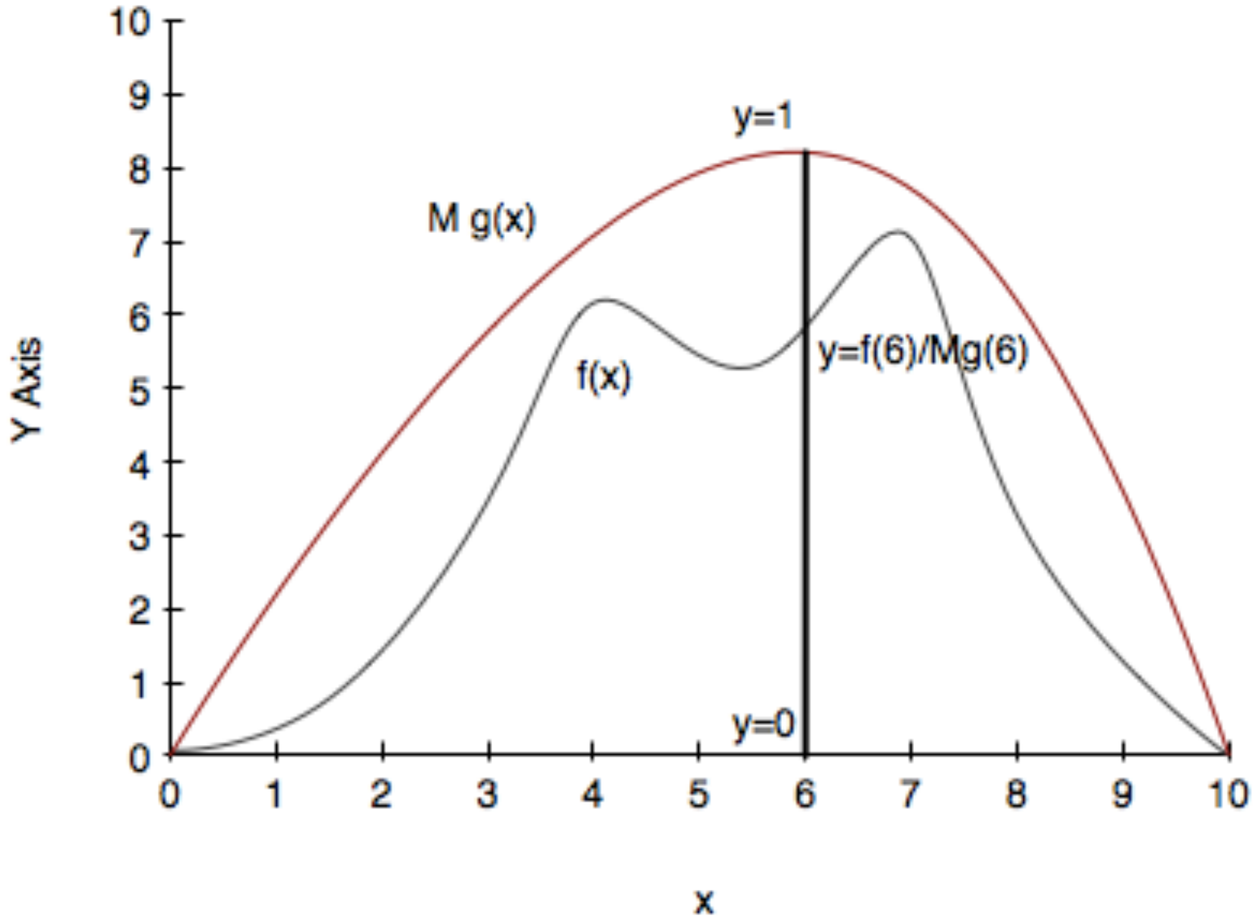
Sampling Distribution, bootstrap

- M data sets **drawn** from the population, thus M estimates
- As we let $M \to \infty$, the distribution induced on $\hat{\mu}$ is the empirical **sampling distribution of the estimator**.
- create data sets by BOOTSTRAP
- but we need samples



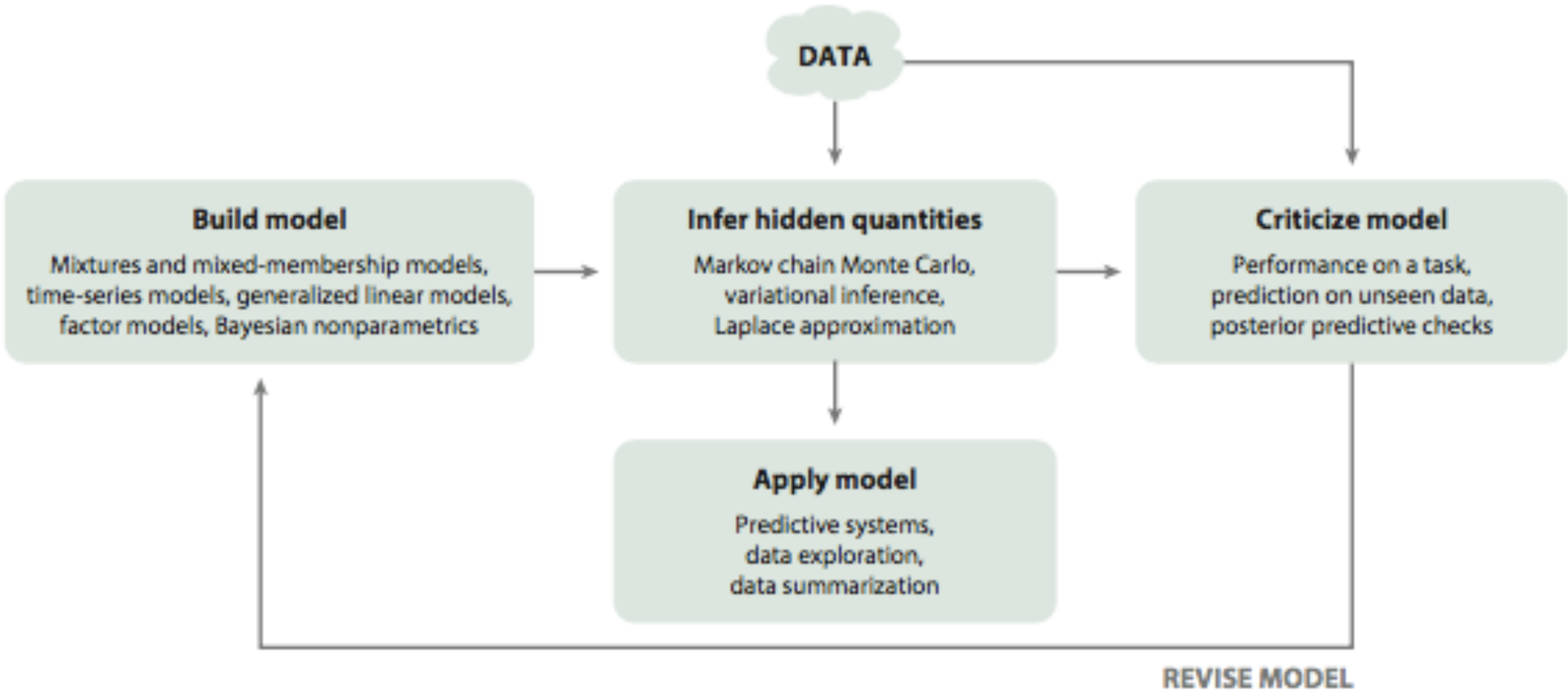
GENERATE THEM!

- Inverse method, Rejection (on steroids)
- Stratification to reduce variance
- Importance (for expectations)
- MCMC, MH, HMC, Slice, ADVI, etc
- integrals (marginalize) by ignoring dimensions in histogram

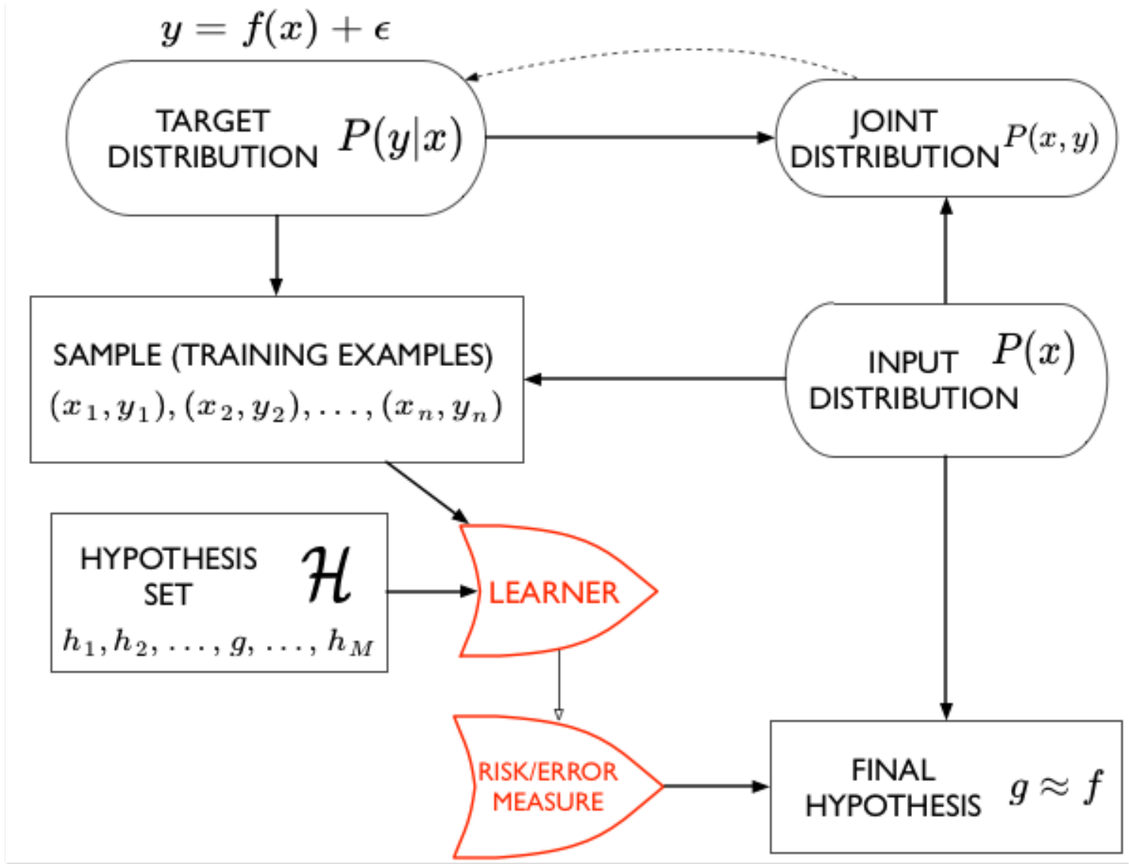


THE NATURE OF LEARNING

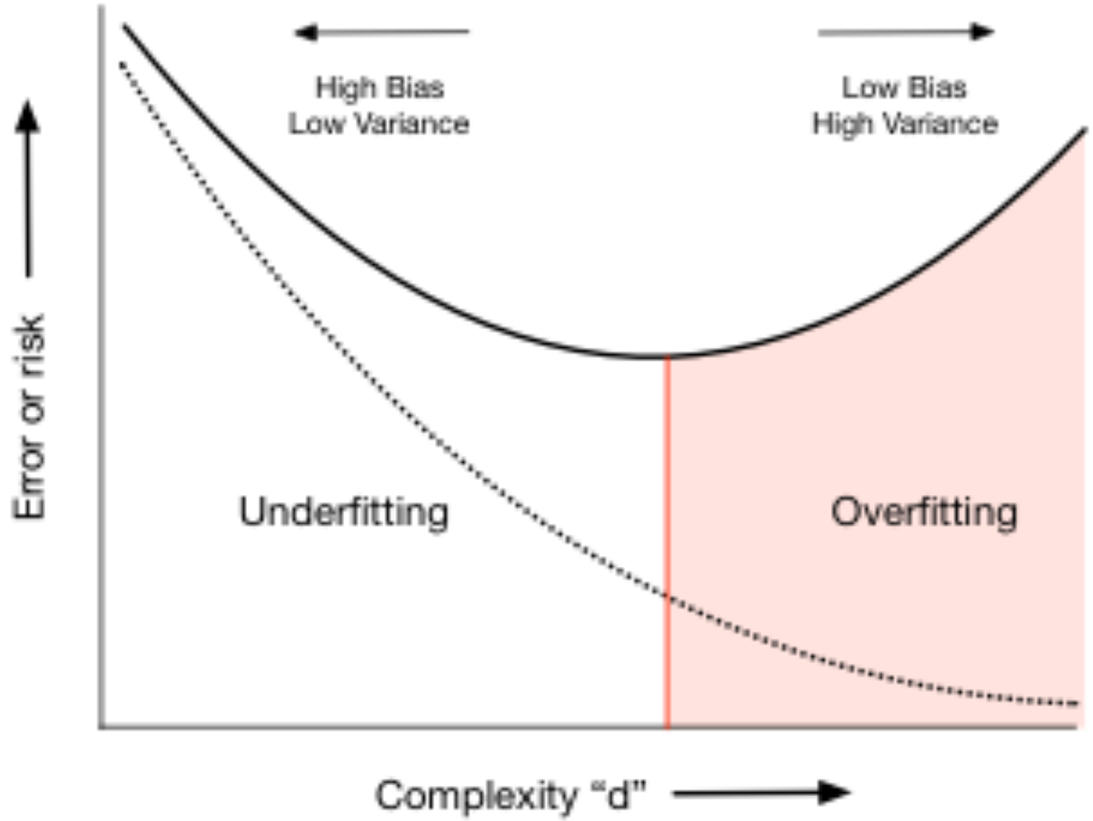
Box's loop


DATA
Build model
Mixtures and mixed-membership models, time-series models, generalized linear models, factor models, Bayesian nonparametrics
Infer hidden quantities
Markov chain Monte Carlo, variational inference, Laplace approximation
Criticize model
Performance on a task, prediction on unseen data, posterior predictive checks
Apply model
Predictive systems, data exploration, data summarization
REVISE MODEL

The nature of learning



Dont Overfit



KL-Divergence

$$D_{KL}(p,q) = E_p[log(p) - log(q)] = E_p[log(p/q)]$$
$$= \sum_i p_i log(\frac{p_i}{q_i}) \ or \ \int dP log(\frac{p}{q})$$

$D_{KL}(p,p) = 0$

KL divergence measures distance/dissimilarity of the two distributions p(x) and q(x).

- used for VI, EM, a probabilistic loss function

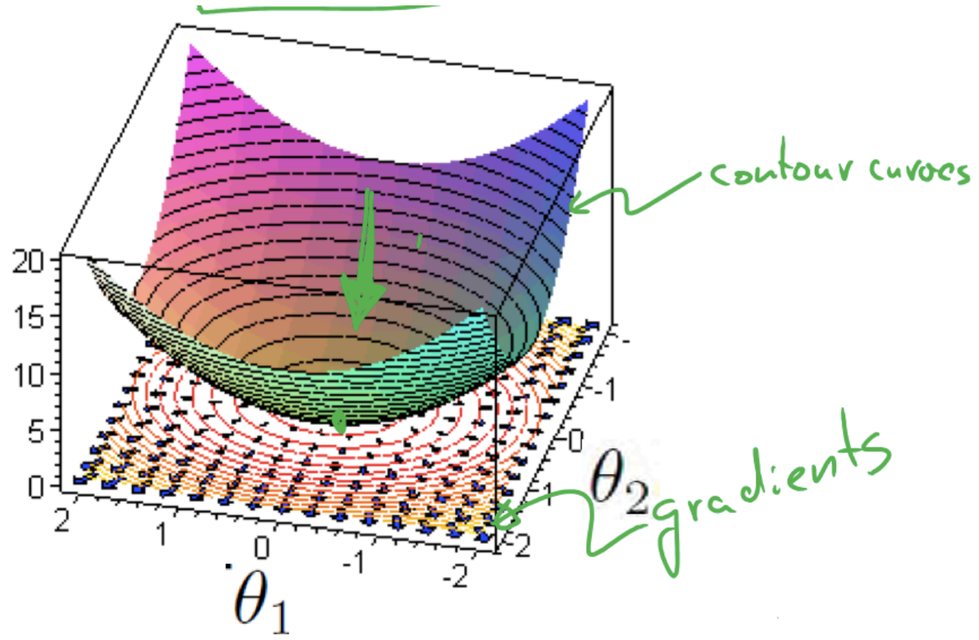
Information Entropy and MAXENT

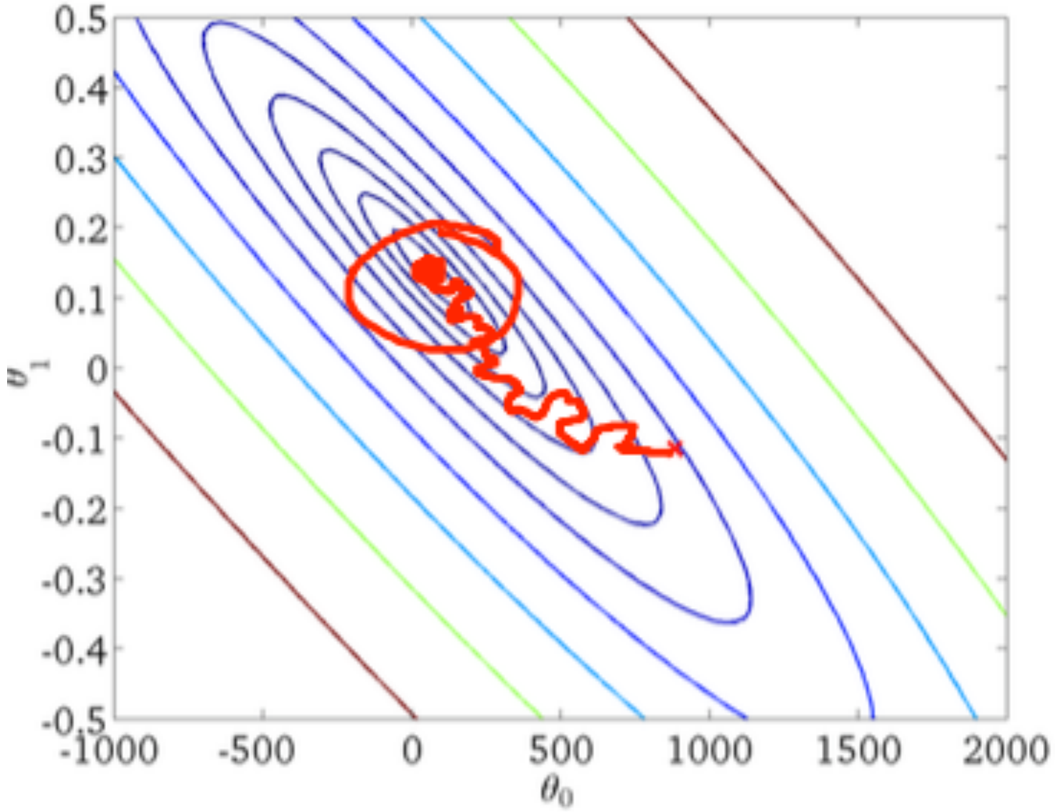
$$H(p) = -E_p[log(p)] = -\int p(x)log(p(x))dx \;\; OR \; -\sum_i p_i log(p_i)$$

- what would be the least surprising distribution, the one with the least additional assumptions (most conservative), the one that can happen in the most ways consistent with constraints
- most common distributions used as likelihoods (and priors) are in the exponential family, MAXENT subject to different constraints.

STOCHASTIC OPTIMIZATION

Gradient Descent and SGD





Simulated Annealing

Minimize f by identifying with the energy of an imaginary physical system undergoing an annealing process.

Move from x_i to x_j via a **proposal**.

If the new state has lower energy, accept x_j.

If the new state has higher energy, accept with $A = \exp\left(-\Delta f / kT\right)$

Lowering temperature slowly, the system reaches "thermal equilibrium" at each temperature. Boltzmann's distribution:

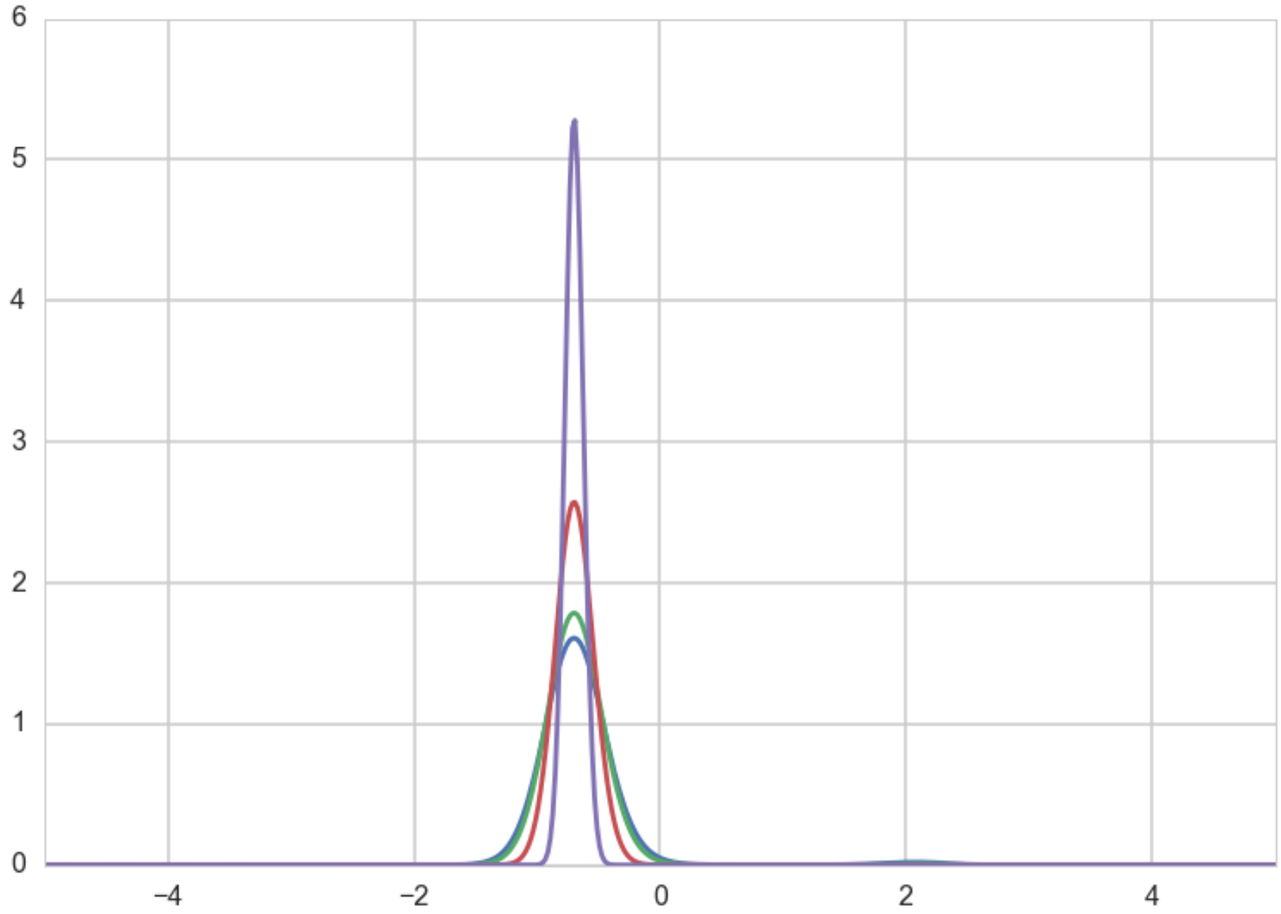
$$p(X = i) = \frac{1}{Z(T)} \exp\left(\frac{-E_i}{kT}\right)$$

If you identify

$$p_T(x) = e^{-f(x)/T} \text{ and } p(x) = e^{-f(x)}$$

Then:

$$P_T(x) = P(x)^{1/T}$$

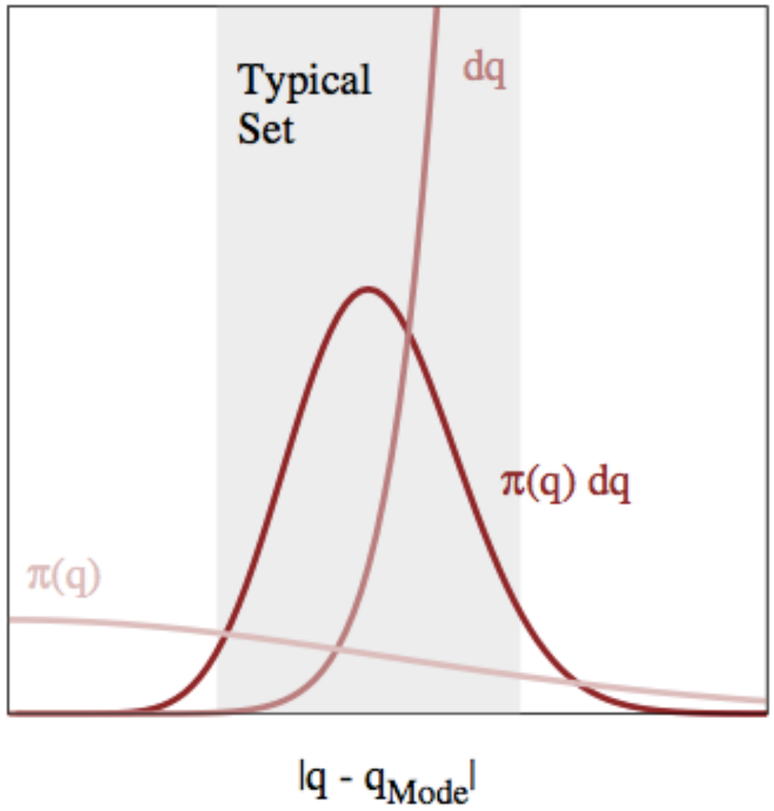


Sampling a Distribution

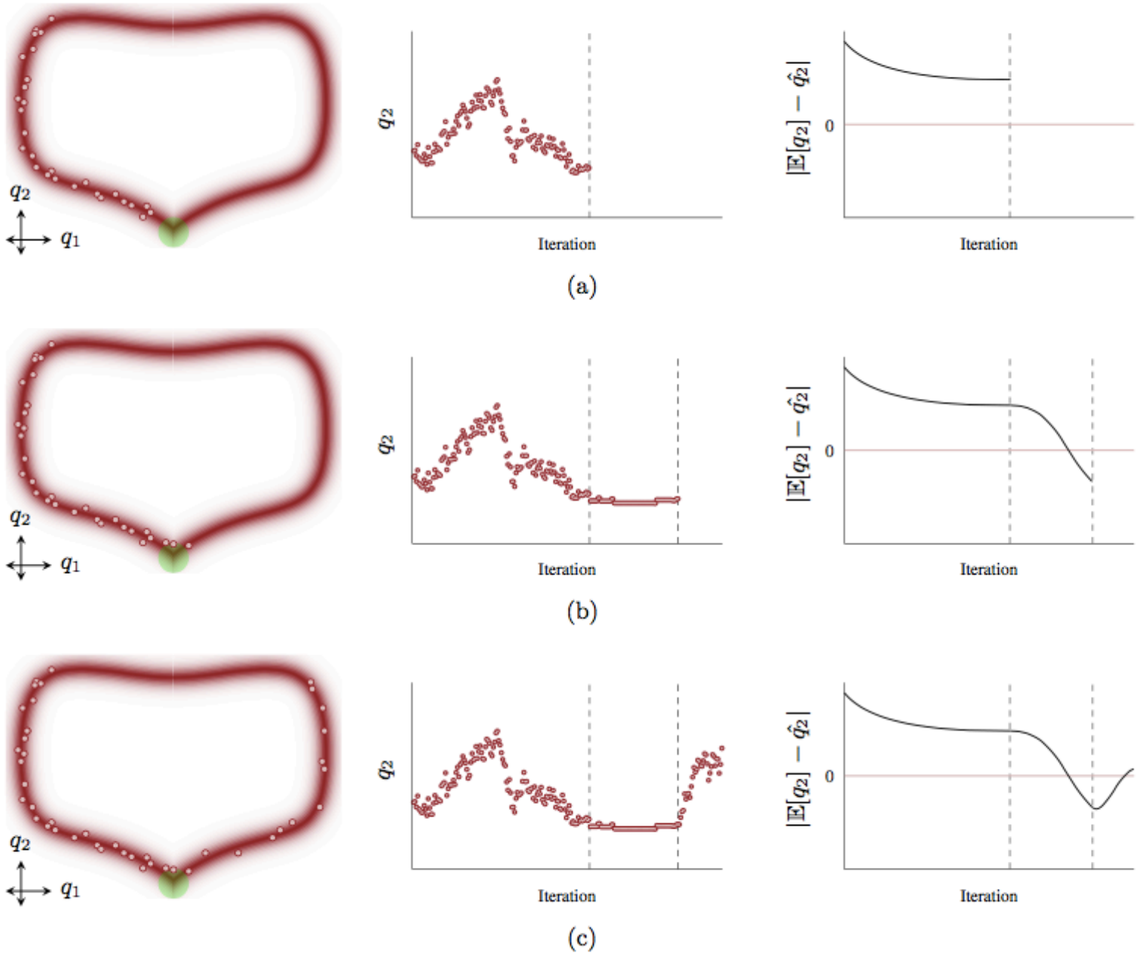
- Turn the question on its head.
- Suppose we wanted to sample from a distribution $p(x)$ (corresponding to a minimization of energy $-log(p(x))$).
- keep our symmetric proposal (reversibility!). Need irreducibility to sample from full distribution
- set T=1, and use our simulated annealing method: **Metropolis**

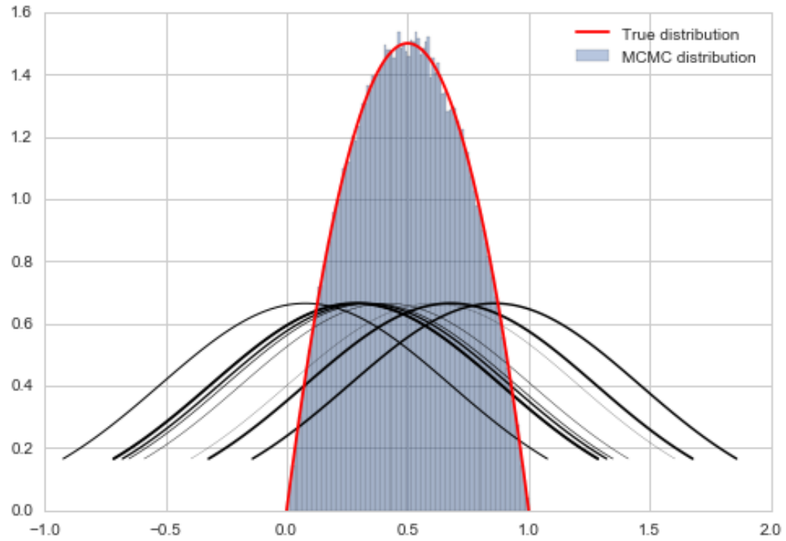
MCMC

Critical: explore the typical set

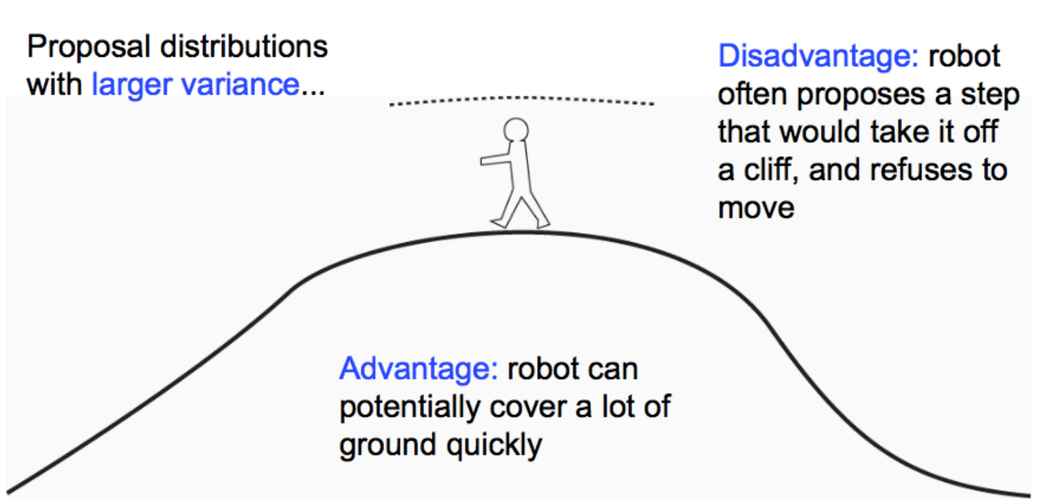


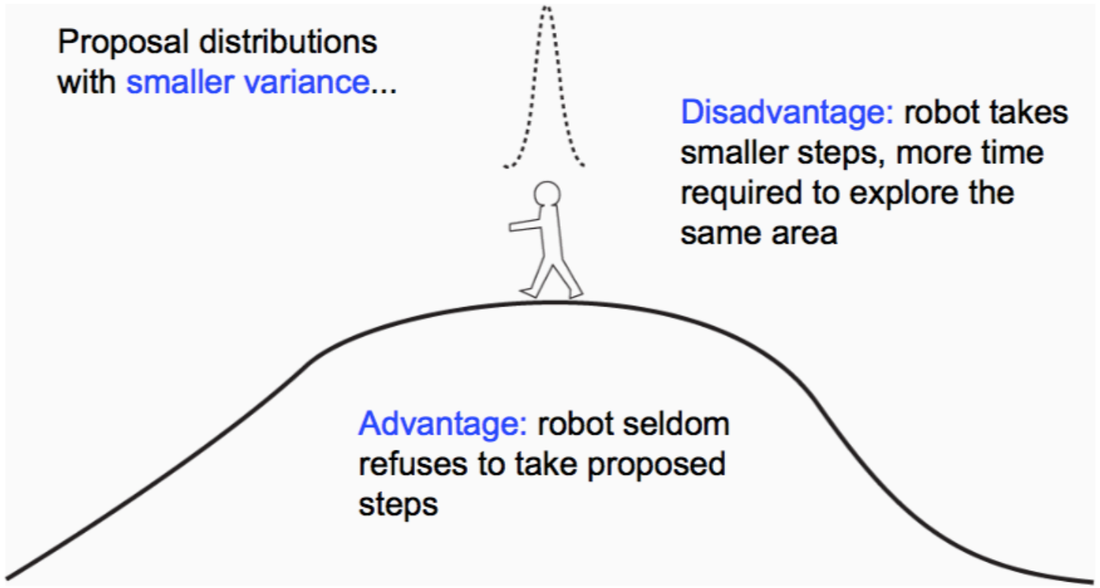
Metropolis and MH





- overshoot and oscillate at pinches
- need to specify step step sizes
- large steps can go outside typical set and are not accepted
- but can cover ground quickly
- small steps accepted but go nowhere
- large correlations


Proposal distributions with larger variance...
Disadvantage: robot often proposes a step that would take it off a cliff, and refuses to move
Advantage: robot can potentially cover a lot of ground quickly

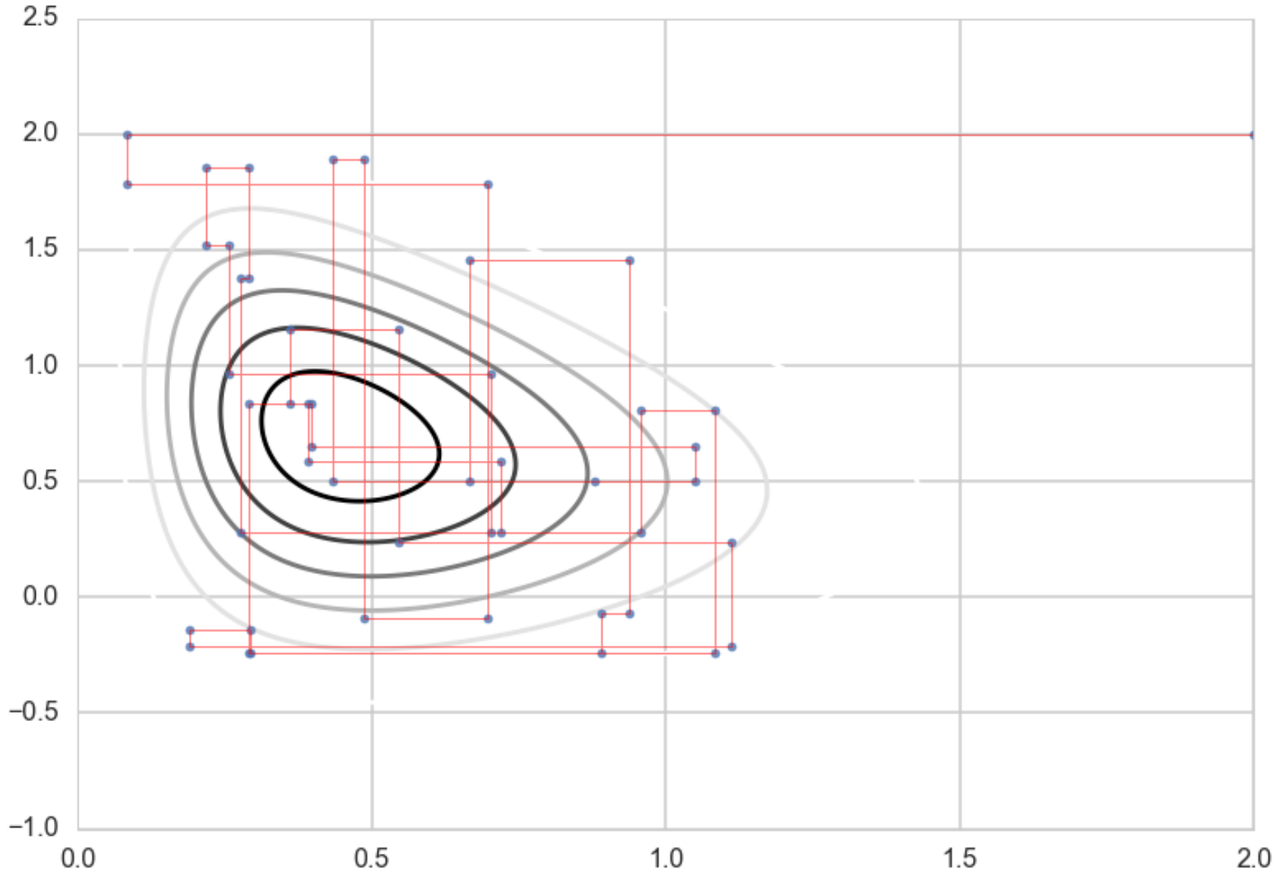

Proposal distributions with smaller variance...
Disadvantage: robot takes smaller steps, more time required to explore the same area
Advantage: robot seldom refuses to take proposed steps

The idea of Gibbs

$f(x_t) = \int h(x_t, x_{t-1}) f(x_{t-1}) dx_{t-1}$, a Stationary distribution.

$h(x, x') = \int dy f(x|y) f(y|x')$.: Sample alternately to get transitions.

Can sample x marginal and $x|y$ so can sample the joint x, y.



Data Augmentation

The difference from Gibbs Sampling: the other variable, say y, is to be treated as latent.

The game is to construct a joint $p(x, y)$ such that we can sample from $p(x|y)$ and $p(y|x)$, and then find the marginal

$$p(x) = \int dy\, p(x, y).$$

LATENT VARIABLES

and

BAYESIAN STATS

Latent Variables

- dont think of bayes/frequentist, think of observed x /Latent z

Any bayesian parameters can be considered as $\mathbf{z}$.

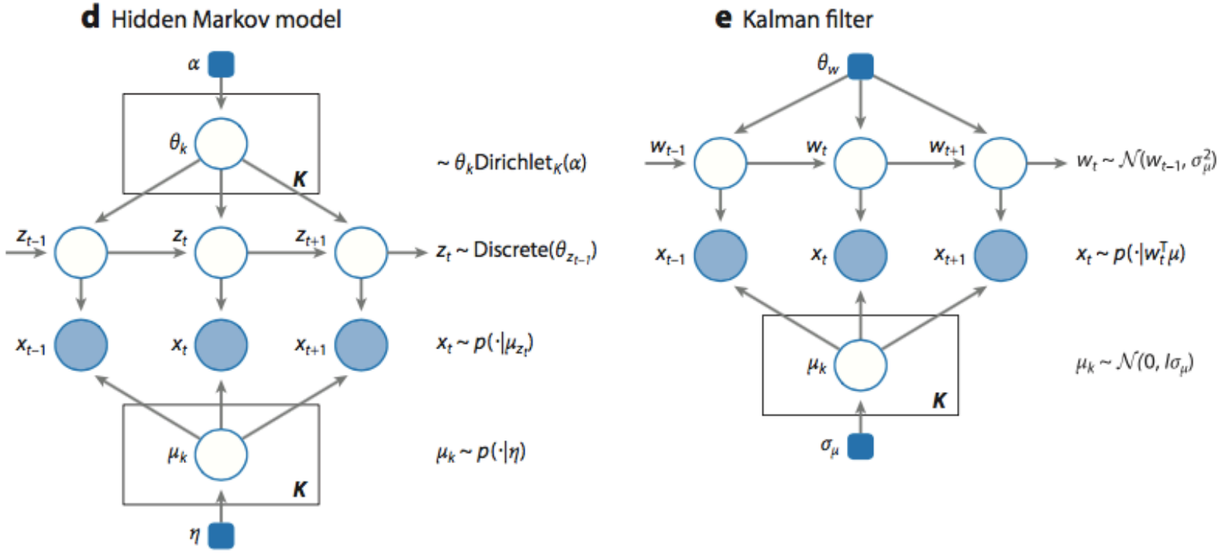
More generally posit hidden structure $\rightarrow$

e.g. **Ratings Latent Factor Model**:

$$Y_{um} = \mu + \theta_u[0] + \gamma_m[0] + \theta_u[1:]^\top \gamma_m[1:] + \epsilon_{um}$$

where $\epsilon_{um} \sim N(0, \sigma)$ and γ_m is an item-specific with first element item-specific bias and remaining latent factors for item m; θ_u is ditto for users; μ overall ratings mean and σ is residual variance of ratings.

See http://multithreaded.stitchfix.com/blog/2015/07/14/glmms/ .



Bayesian

- sample is the data, and is fixed
- parameter is stochastic, has prior and posterior distribution
- posterior: $p(\theta|y) = \frac{p(y|\theta)\,p(\theta)}{p(y)}$, can summarize via MAP
- just bayes rule: $posterior = \frac{likelihood \times prior}{evidence}$

From `edwardlib`: $p(\mathbf{x} \mid \mathbf{z})$

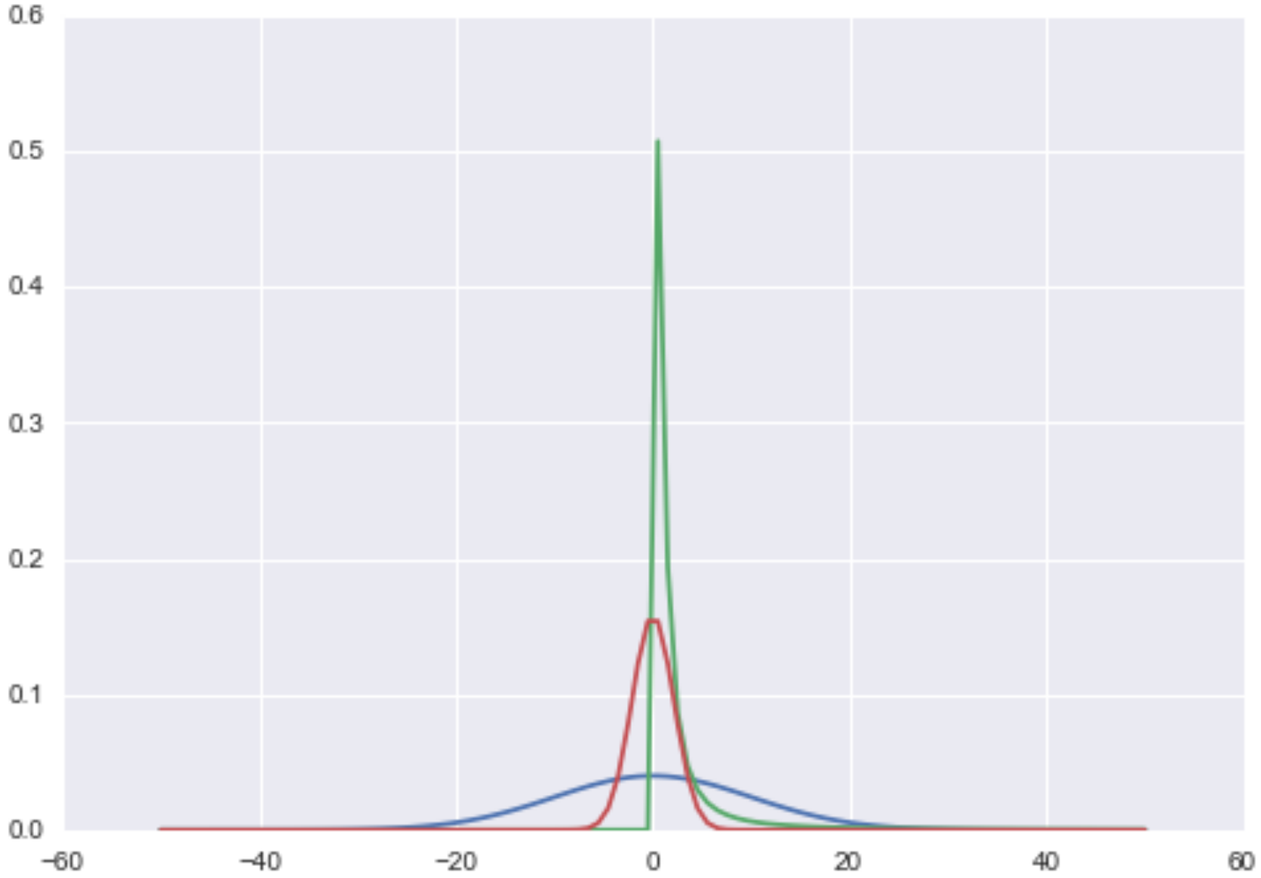
describes how any data $\mathbf{x}$ depend on the latent variables $\mathbf{z}$.

- **The likelihood posits a data generating process**, where the data $\mathbf{x}$ are assumed drawn from the likelihood conditioned on a particular hidden pattern described by $\mathbf{z}$.
- The *prior* $p(\mathbf{z})$ is a probability distribution that describes the latent variables present in the data. **The prior posits a generating process of the hidden structure**.

- prior-predictive = evidence: $p(y) = E_{p(\theta)}[\mathcal{L}] = \int d\theta p(y|\theta)p(\theta)$ a normalization, irrelevant for sampling, useful for EB
- What if θ is multidimensional? Marginal posterior: $p(\theta_1|D) = \int d\theta_{-1} p(\theta|D).$
- posterior predictive: the distribution of a future data point y^*:

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

priors



- choose likelihoods with MAXENT
- choose priors as non-informative, e.g. uniform or Jeffreys
- better still: choose priors as weakly informative/regularizing
- helps with sampler performance
- see https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations and Stan Manual

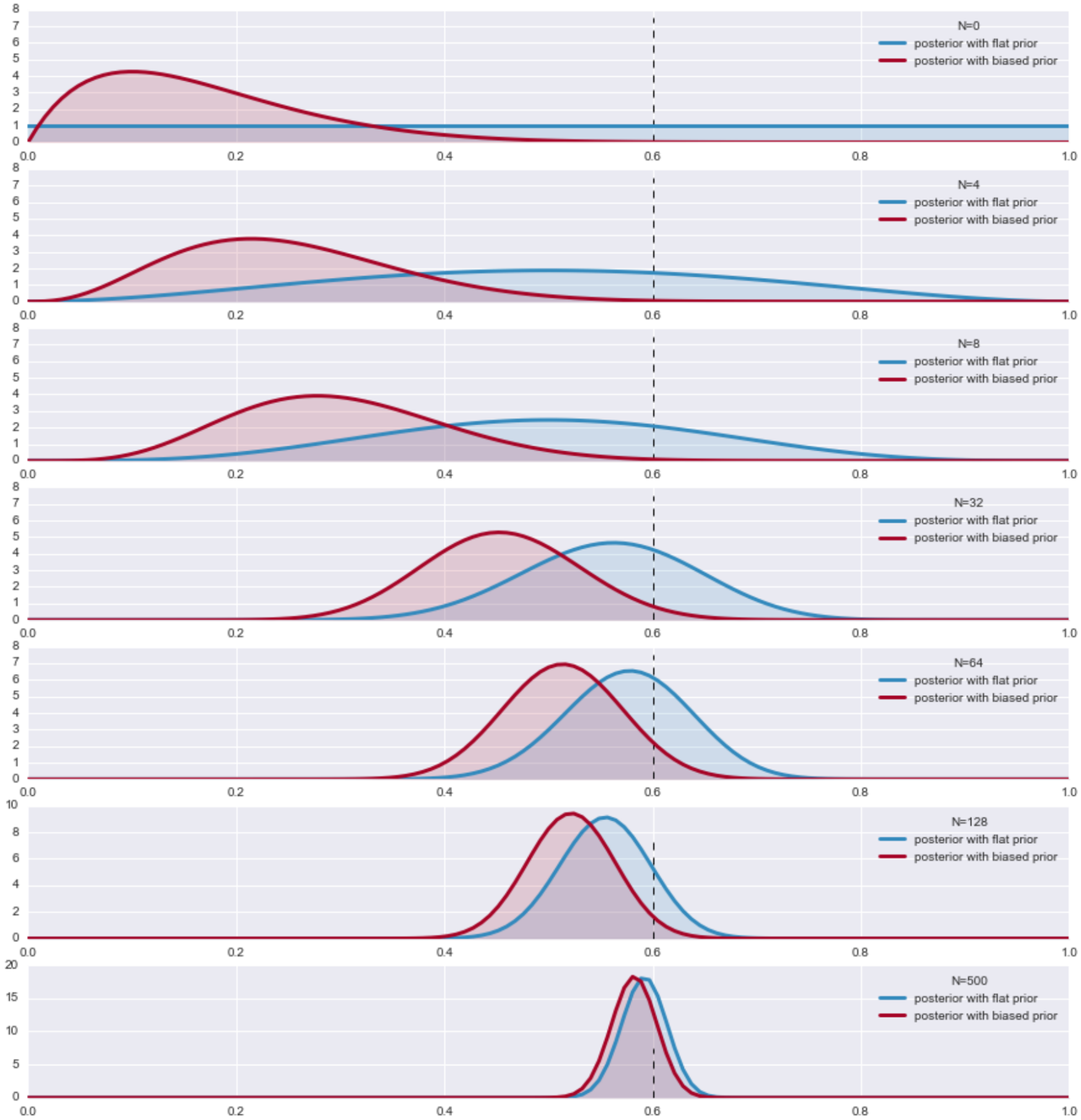
Data Overwhelms priors

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

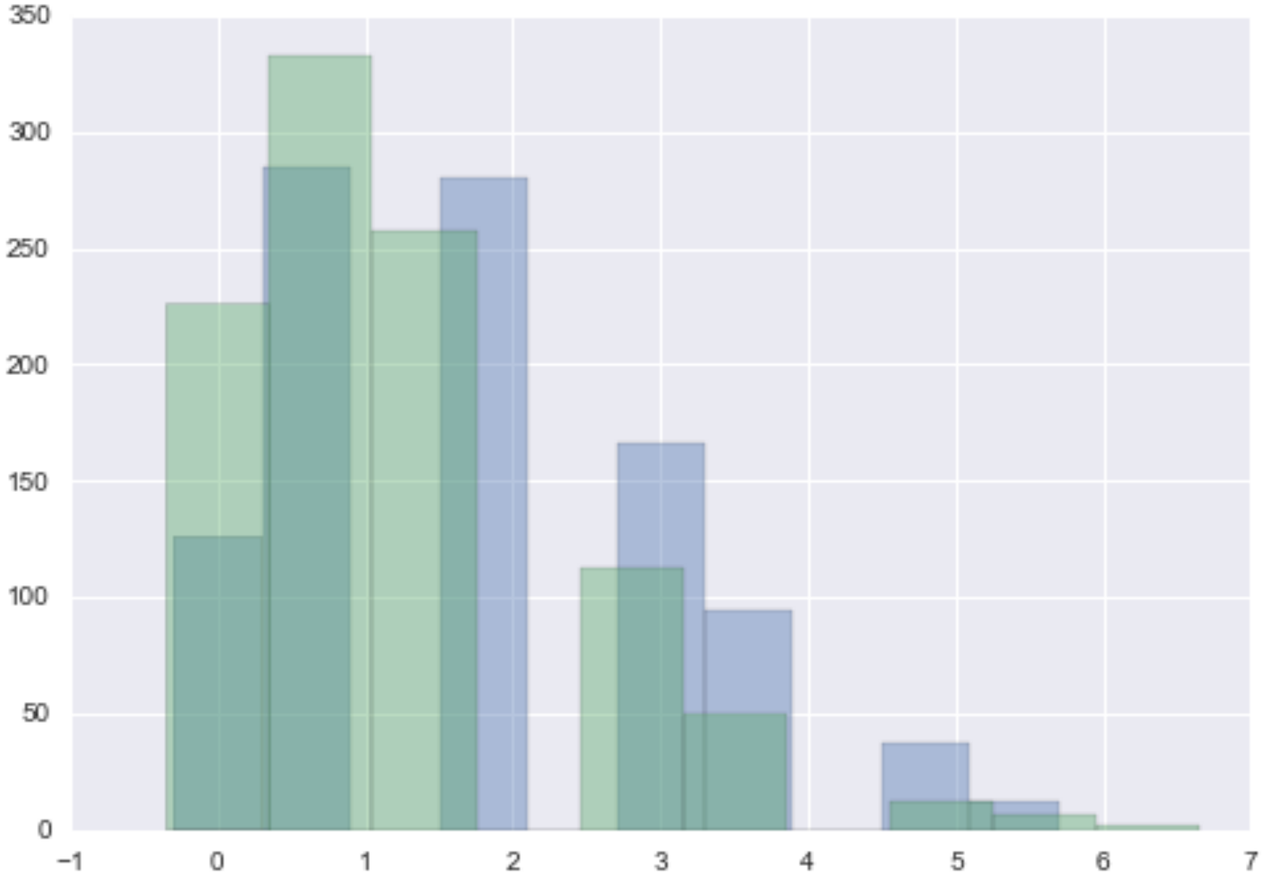
$$\mu_p = \frac{b}{a} = \frac{\kappa}{\kappa+n}\hat{\mu} + \frac{n}{\kappa+n}\bar{y}$$

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

- priors regularize data for small data
- but large data overwhelms priors



Posterior Predictives



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

Sampling easy (mothers poisson-gamma):

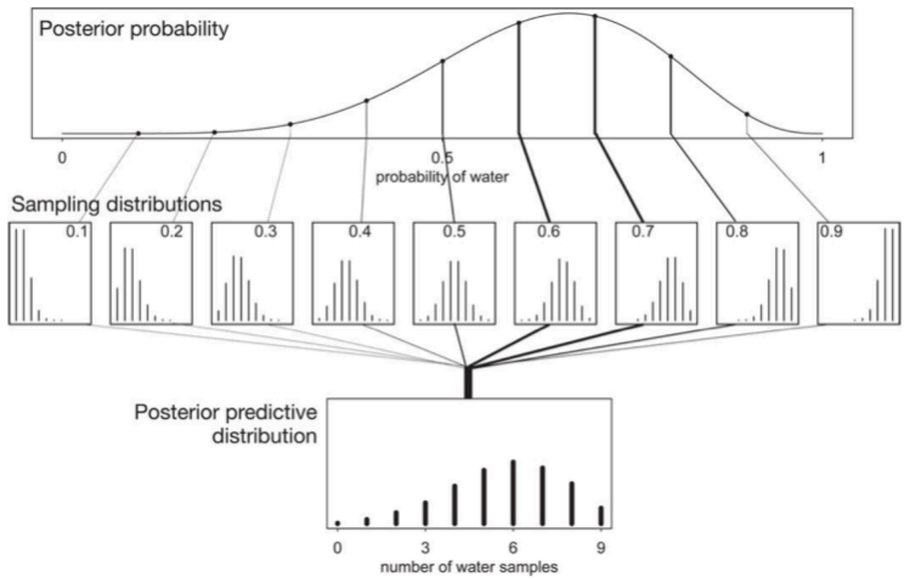
```
postpred1 = poisson.rvs(theta1trace)
postpred2 = poisson.rvs(theta2trace)
```

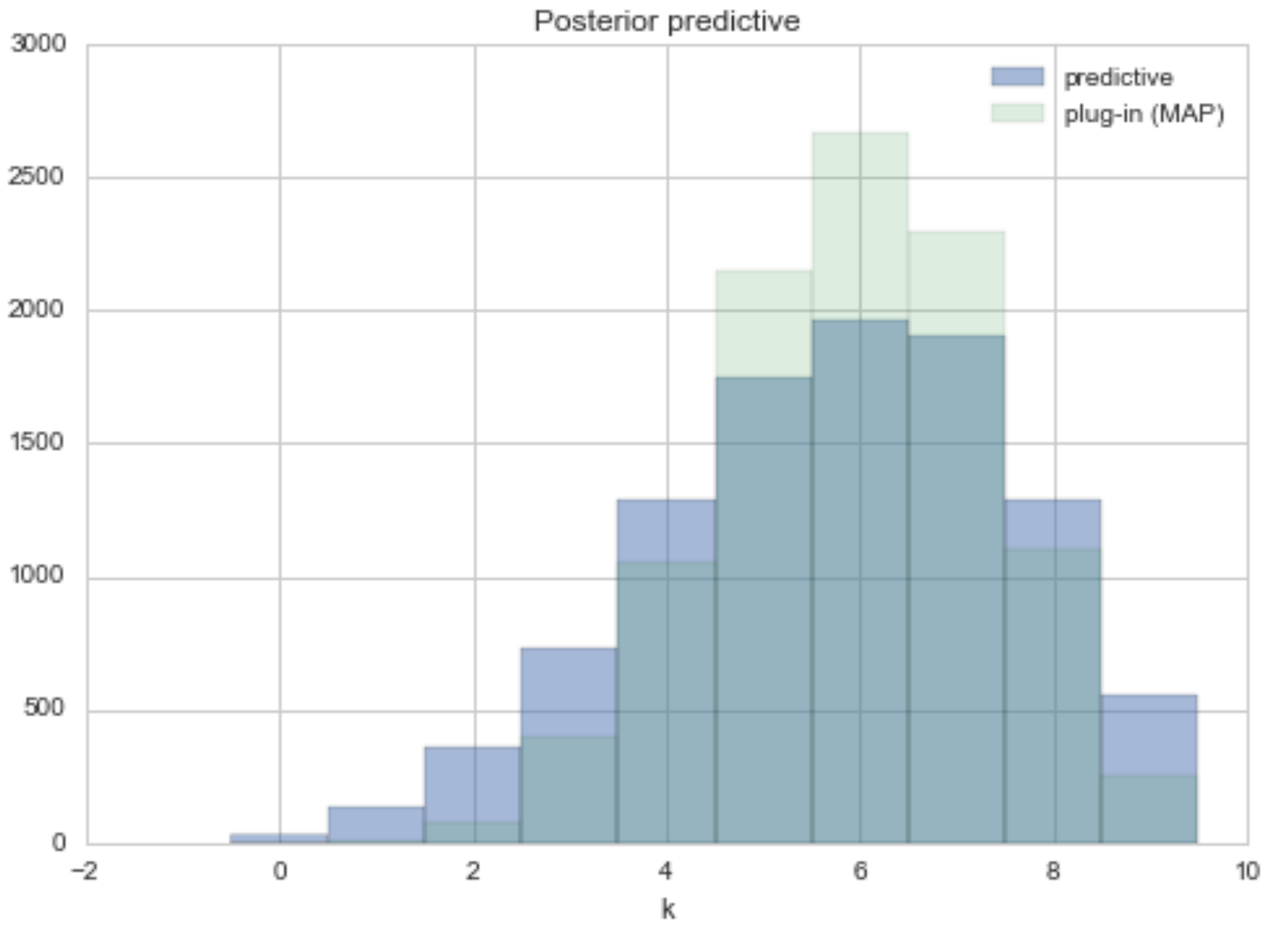
Exact: Negative Binomial (requires math):

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

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

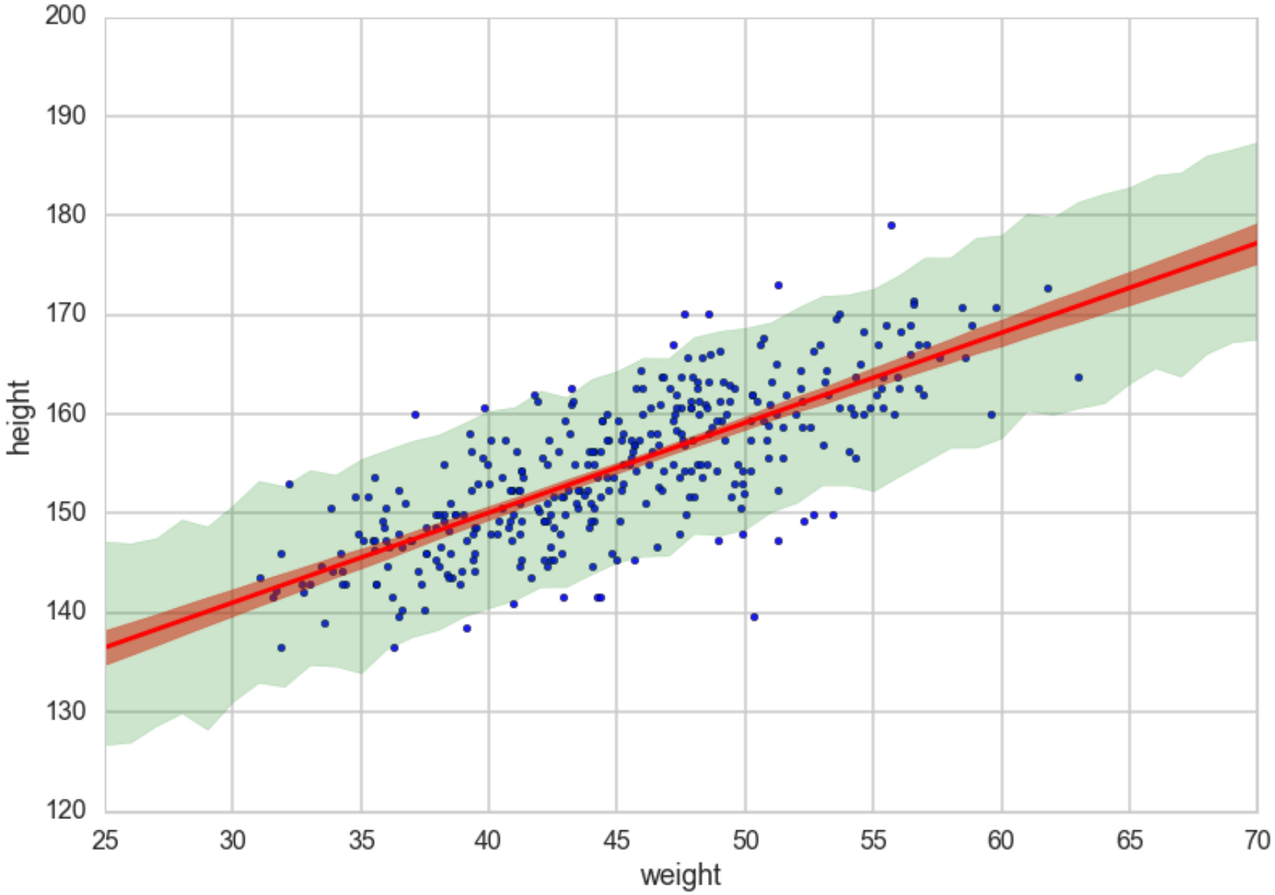
Posterior Predictive Smear



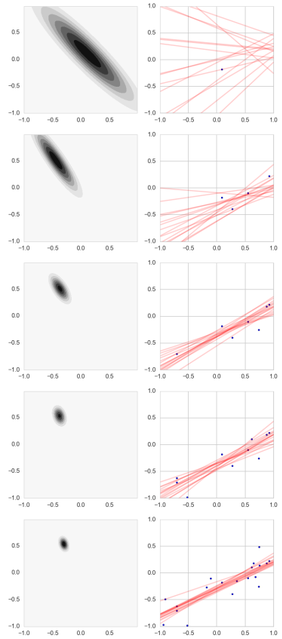


pp vs sampling distrib at MAP →

Bayesian Regression



- posterior narrower (μ spread) than PP
- supervised learning, a distrib at each x



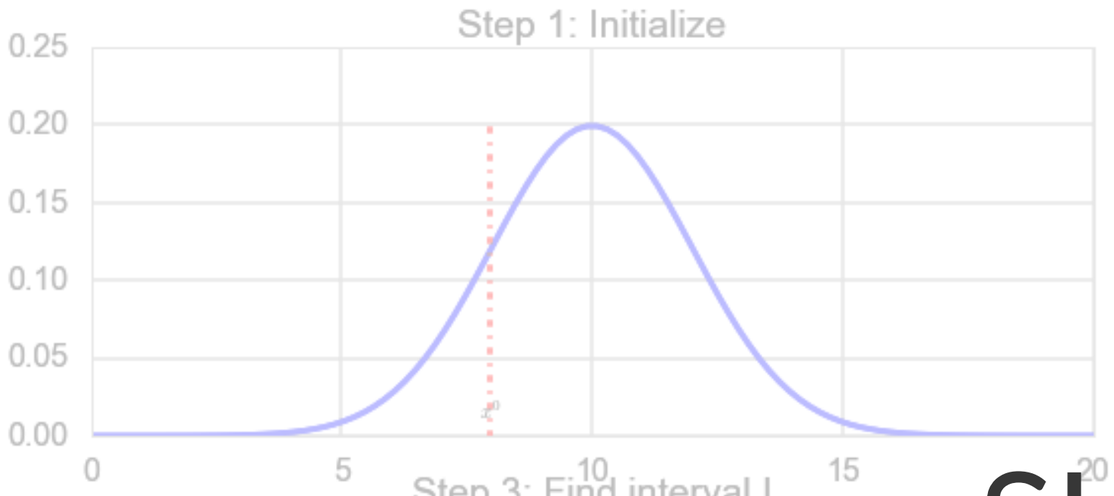
SAMPLING ON STEROIDS

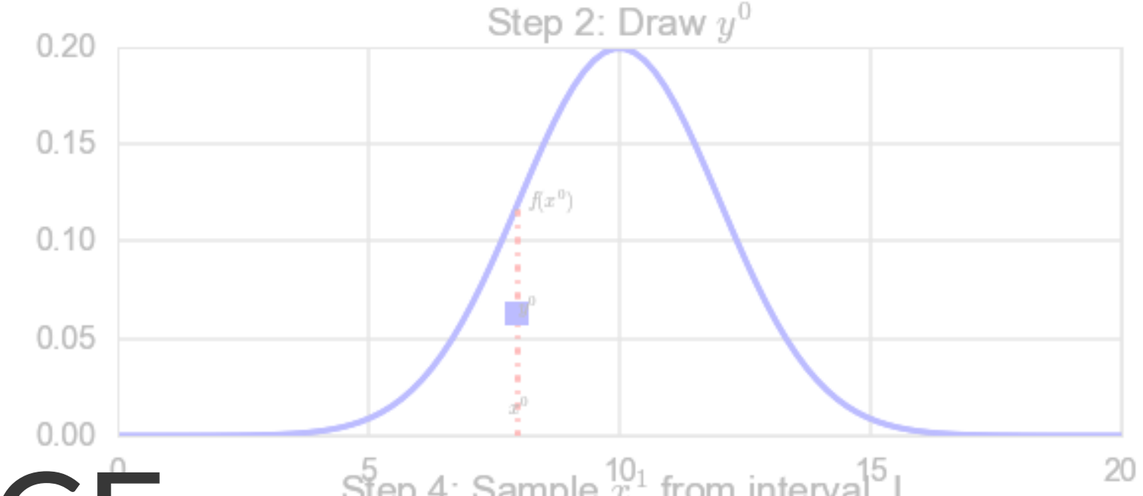
SLICE and HMC

Issues and Monitors

**We need to test for convergence* *

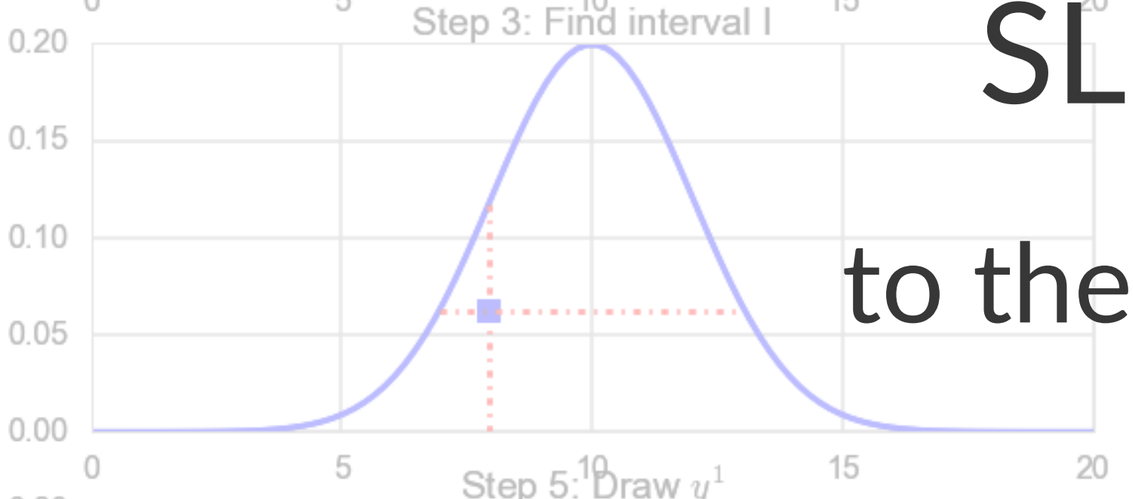
- traceplots, autocorrelation, before and after burnin, thinning
- persistent correlation, trace badness indicates problems with model
- run multiple chains, see n_{eff}
- compute Gewecke and Gelman-Rubin

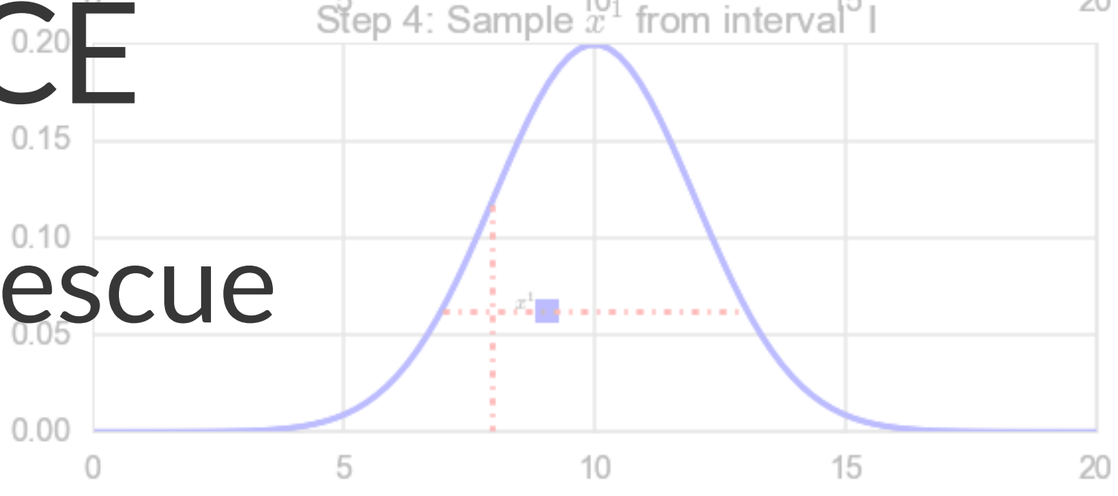


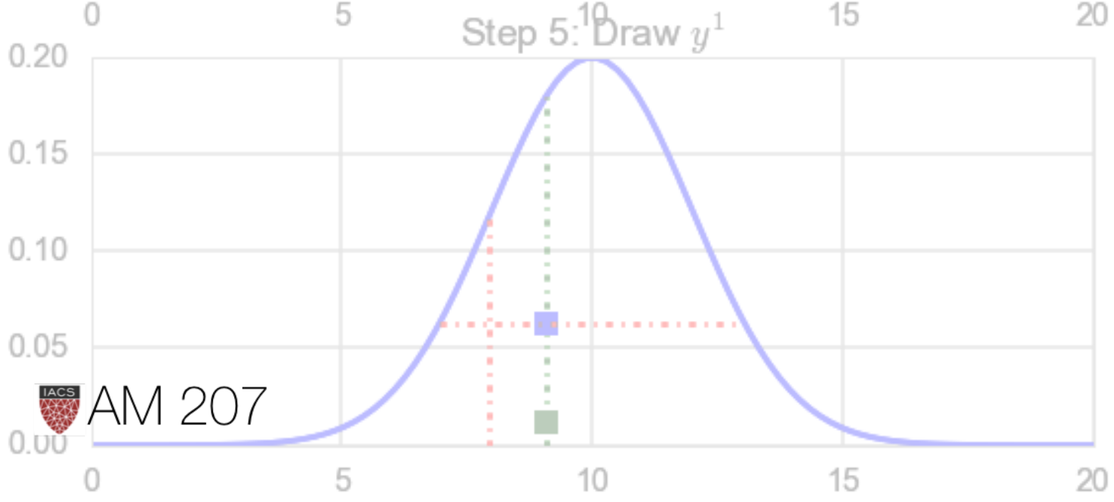


SLICE

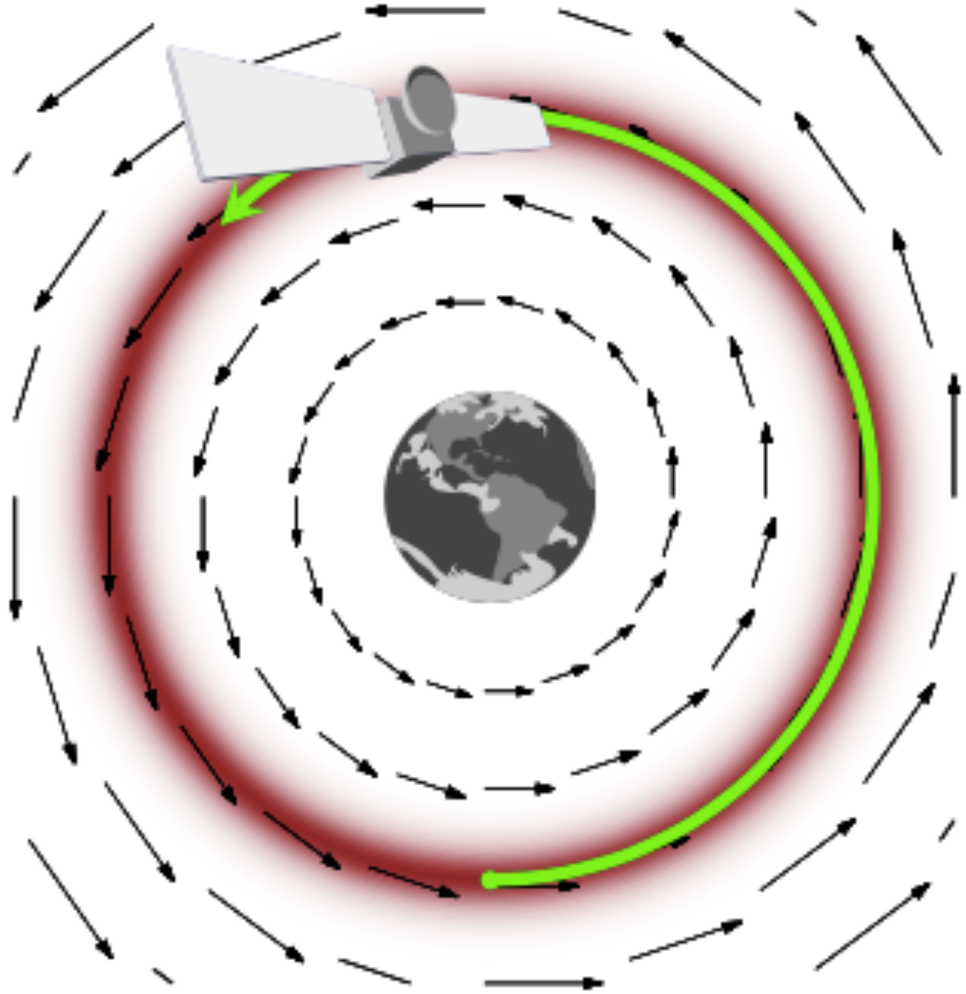
to the rescue

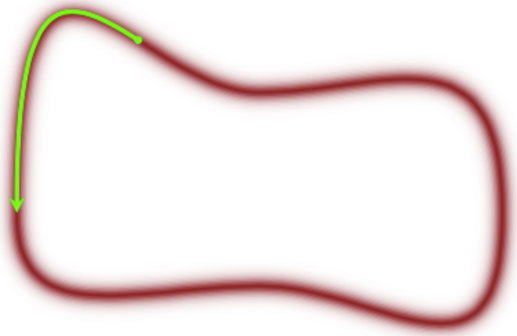






HMC to the rescue: need glide





Now, like in annealing, let $p(p,q) = e^{-Energy}$

DA with an additional momentum gives energy **Hamiltonian** $H(p,q) = \frac{p^2}{2m} + V(q)$

Hamiltonian flow: reversible, time-invariant, volume-preserving

Thrusters fire away

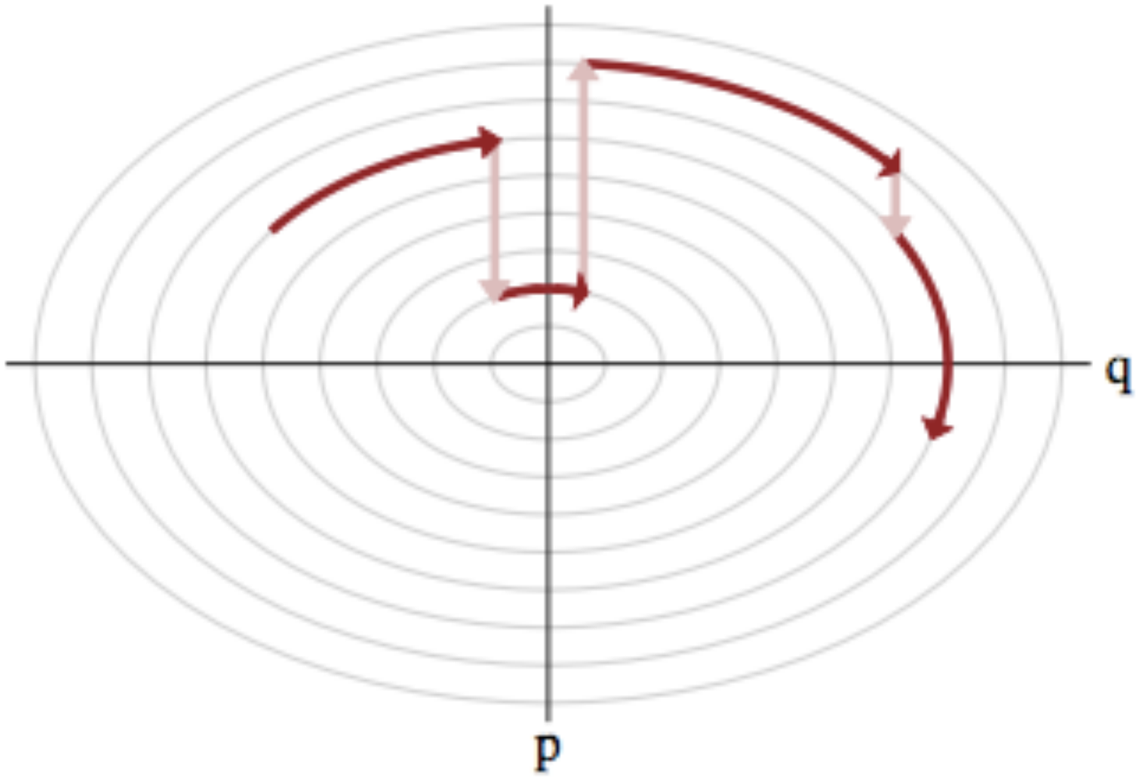
$$p(p,q) = e^{-H(p,q)} = e^{-K(p,q)}e^{-V(q)} = p(p|q)p(q)$$

$$H(p,q) = -log(p(p,q)) = -logp(p|q) - logp(q)$$

Choice of a kinetic energy term is choice of a conditional probability distribution over the "augmented" momentum such that:

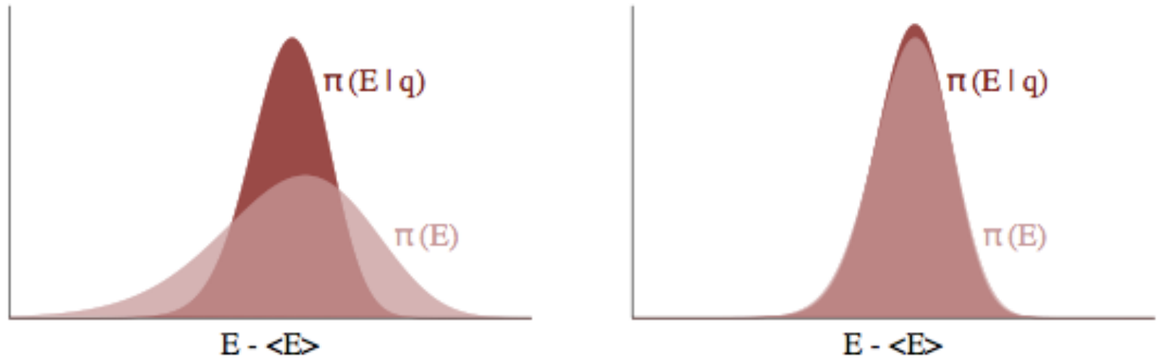
$$\int dpp(p,q) = \int dpp(p|q)p(q) = p(q)\int p(p|q)dp = p(q)$$

.



Tuning:

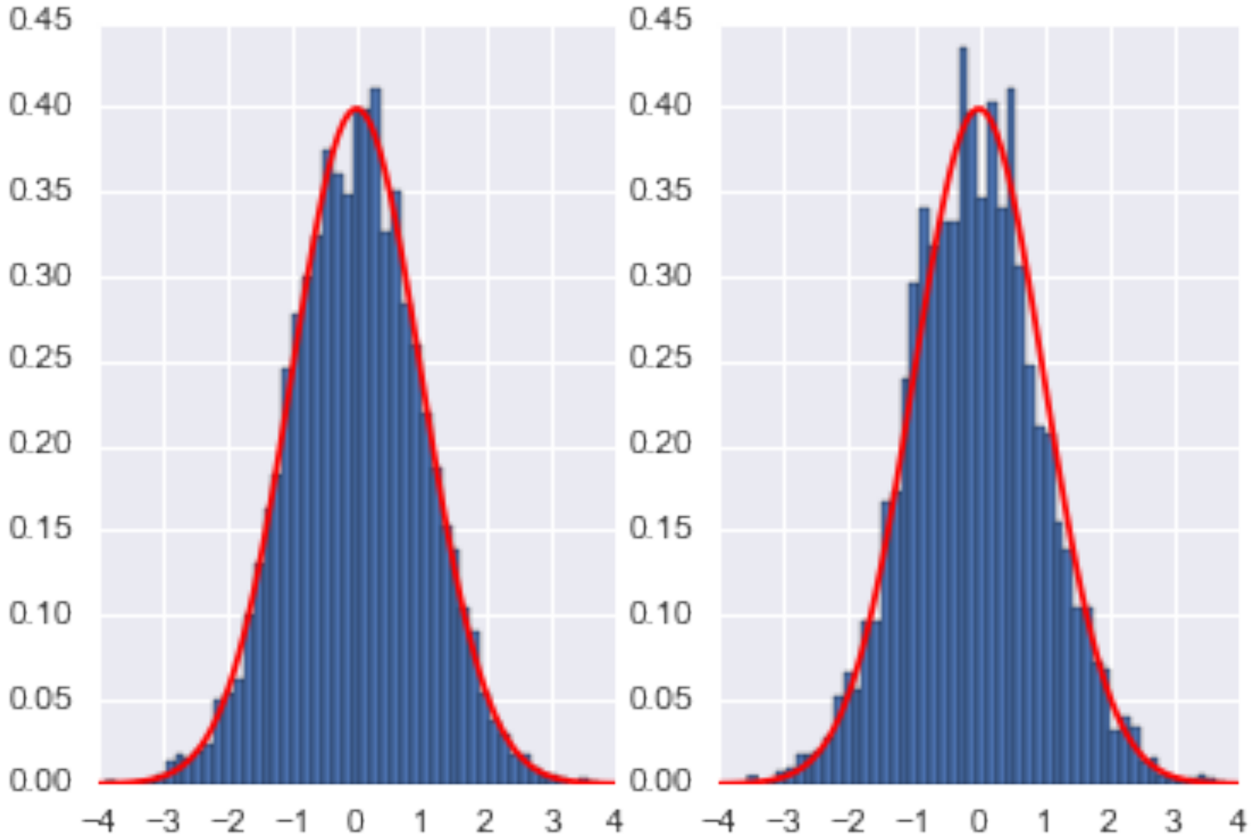
- The ideal kinetic energy interacts with target, in practice we often use $K(p) = p' M^{-1} p$
- Set inverse mass matrix to the covariance of the target distribution: maximally decorrelate the target. Do in warmup phase.
- use symplectic integration
- need to determine L and ϵ.
- generally static not good, under samples tails (high-energy microcanonicals). Estimate dynamically: NUTS (pymc3 and Stan)



Acceptance probability

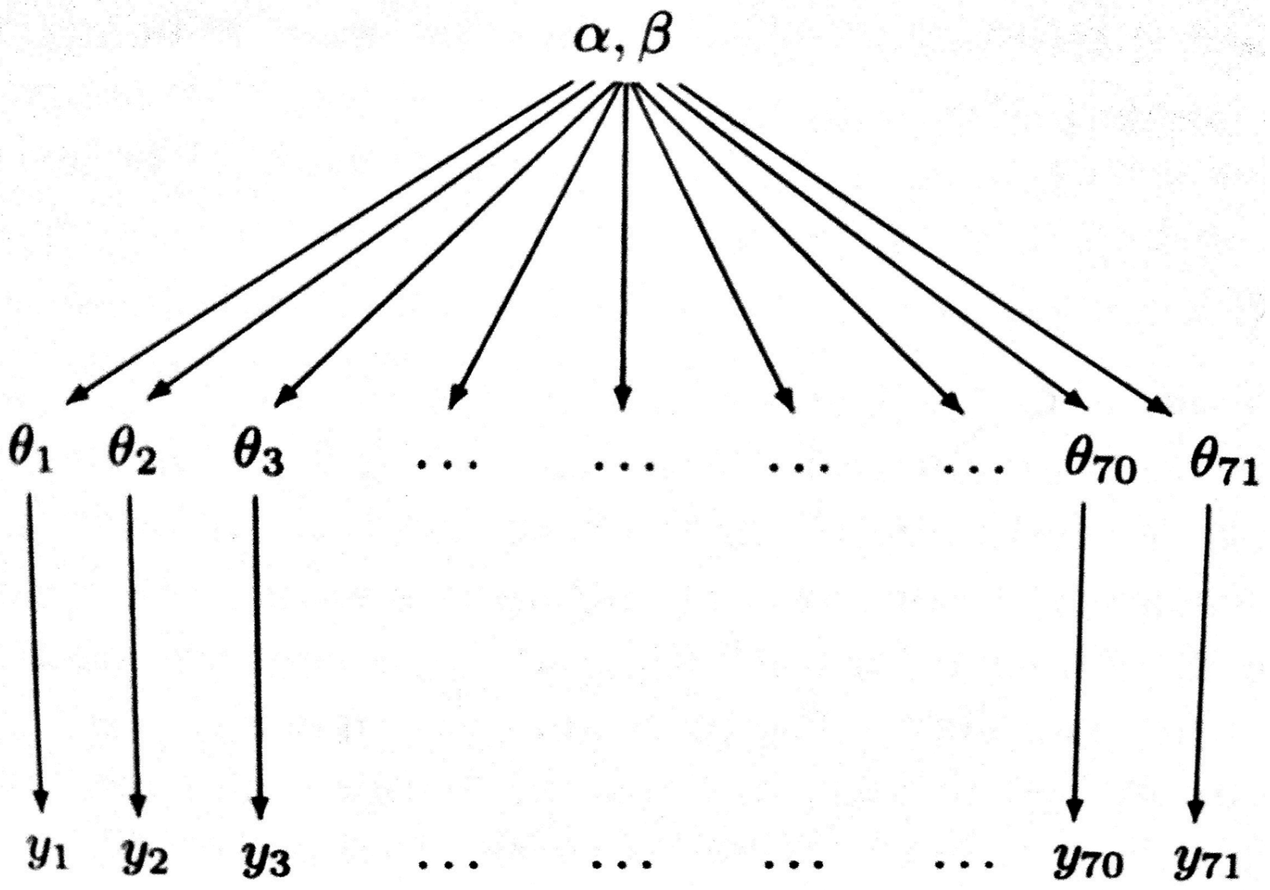
- small symplectic errors means H evolution only forward in time
- tack on sign change $(q, p) \rightarrow (q_L, -p_L)$. Superman to the rescue!
- Acceptance: $A = \mathbf{min}[1, \frac{p(q_L, -p_L)\delta(q_L - q_L)\delta(-p_L + p_L)}{p(q, p)\delta(q - q)\delta(p - p)}]$
- More general acceptance in NUTS, sum over all points in orbit

```python
def HMC(U,K,dUdq,N,q_0, p_0, epsilon=0.01, L=100):
    current_q = q_0
    current_p = p_0
    H = np.zeros(N)
    qall = np.zeros(N)
    accept=0
    for j in range(N):
        q = current_q
        p = current_p
        #draw a new p
        p = np.random.normal(0,1)
        current_p=p
        # leap frog
        # Make a half step for momentum at the beginning
        p = p - epsilon*dUdq(q)/2.0
        # alternate full steps for position and momentum
        for i in range(L):
            q = q + epsilon*p
            if (i != L-1):
                p = p - epsilon*dUdq(q)
        #make a half step at the end
        p = p - epsilon*dUdq(q)/2.
        # negate the momentum
        p= -p;
        current_U = U(current_q)
        current_K = K(current_p)
        proposed_U = U(q)
        proposed_K = K(p)
        A=np.exp( current_U-proposed_U+current_K-proposed_K)
        # accept/reject
        if np.random.rand() < A:
            current_q = q
            qall[j]=q
            accept+=1
        else:
            qall[j] = current_q
        H[j] = U(current_q)+K(current_p)
    print("accept=",accept/np.double(N))
    return H, qall
```



HIERARCHICAL MODELS
AND GLMs

Partial Pooling



Key Idea: Share statistical strength

- Some **units** (experiments) statistically more robust
- Non-robust experiments have smaller samples or outlier like behavior
- Borrow strength from all the data as a whole through the estimation of the hyperparameters
- **regularized partial pooling model** in which the "lower" parameters (θs) tied together by "upper level" hyperparameters.

Empirical Bayes or Type-2 Likelihood

Posterior-predictive distribution, as a function of upper level parameters $\eta = (\alpha, \beta)$.

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

A likelihood with parameters η and simply use maximum-likelihood with respect to η to estimate these η using our "data" y^*

Used in GPS, even can be sampled from

Howto Sampling

- a DAG, with observations at the bottom of a tree, next layer intermediate parameters, upper layers hyper-parameters
- sample conditionals from parents up the tree.
- general structure is sampling steps inside Gibbs
- stan, pymc3 all have this structure

Levels of Bayes

Method	Definition
Maximum Likelihood	$\hat{\theta} = argmax_\theta p(D\|\theta)$
MAP estimation	$\hat{\theta} = argmax_\theta p(D\|\theta)p(\theta\|\eta)$
ML-2 (Empirical Bayes)	$\hat{\eta} = argmax_\eta \int d\theta\, p(D\|\theta)p(\theta\|\eta) = argmax_\eta p(D\|\eta)$
MAP-2	$\hat{\eta} = argmax_\eta \int d\theta\, p(D\|\theta)p(\theta\|\eta)p(\eta) = argmax_\eta p(D\|\eta)p(\eta)$
Full Bayes	$p(\theta, \eta\|D) \propto p(D\|\theta)p(\theta\|\eta)p(\eta)$

Normal-Normal Hierarchical Model

J independent experiments, experiment j estimating the parameter θ_j from n_j independent normally distributed data points, y_{ij}, each with known error variance σ^2; that is,

$$y_{ij}|\theta_j \sim N(\theta_j, \sigma^2),\ i = 1, \ldots, n_j; j = 1, \ldots, J.$$

Gelman 8-schools problem: estimated coaching effects $\bar{y}_j$ to improve SAT scores for school j, with sampling variances, σ_j^2.

Sample mean of each group j

$\bar{y}_j = \frac{1}{n_j}\sum_{i=1}^{n_j} y_{ij}$ with sampling variance

$\sigma_j^2 = \sigma^2/n_j.$

Likelihood for θ_j using suff-stats, $\bar{y}_j$:

$$\bar{y}_j|\theta_j \sim N(\theta_j, \sigma_j^2).$$

Notation flexible in allowing a separate variance σ_j^2 for the mean of each group j.

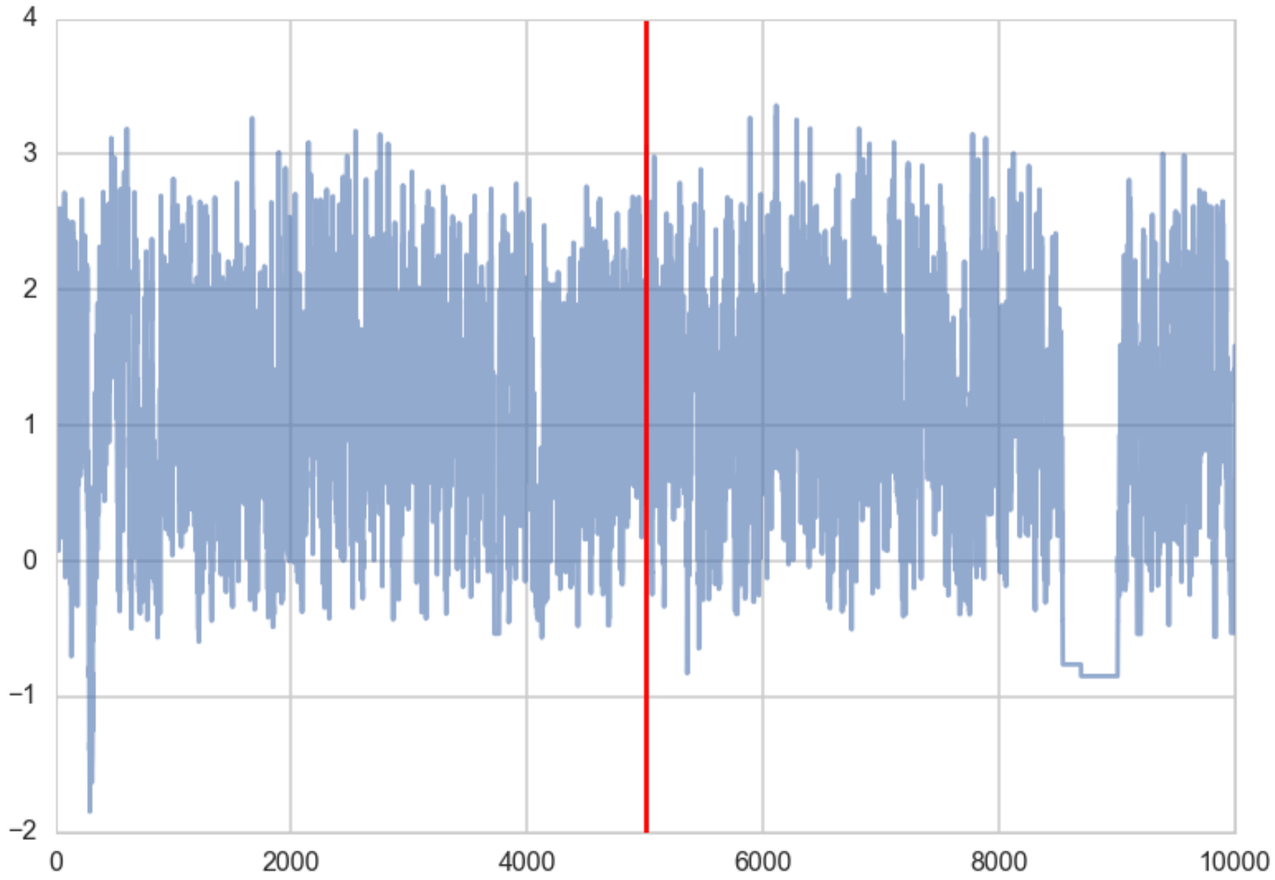
Appropriate when the variances differ for reasons other than number of data pts.

School	Estimated treatment effect, y_j	Standard error of effect estimate, σ_j
A	28	15
B	8	10
C	−3	16
D	7	11
E	−1	9
F	1	11
G	18	10
H	12	18

Centered Hierarchical Model

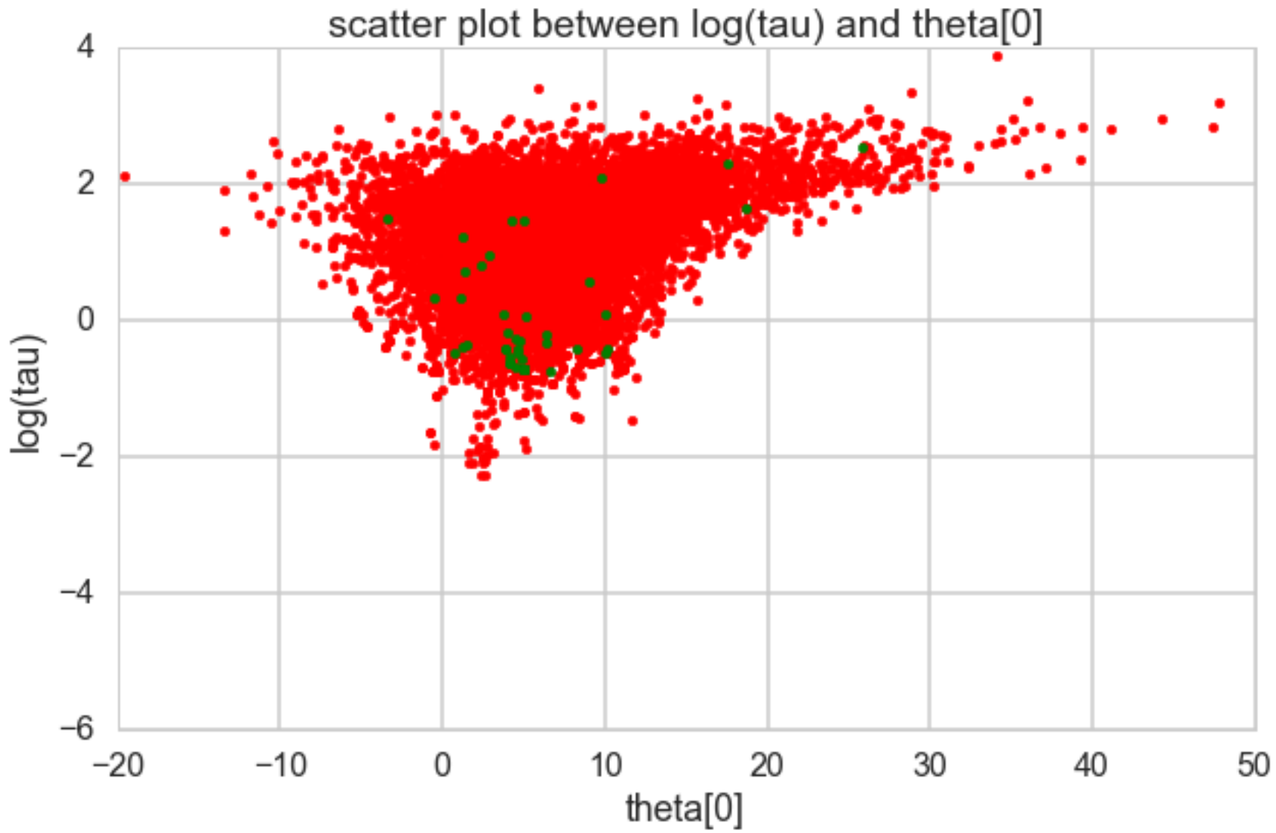
$$\mu \sim \mathcal{N}(0, 5)$$
$$\tau \sim \text{Half-Cauchy}(0, 5)$$
$$\theta_j \sim \mathcal{N}(\mu, \tau)$$
$$\bar{y_j} \sim \mathcal{N}(\theta_j, \sigma_j)$$

problem: Small n_{eff}. Poor sampling.

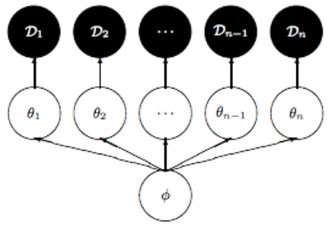


High Curvature Issues

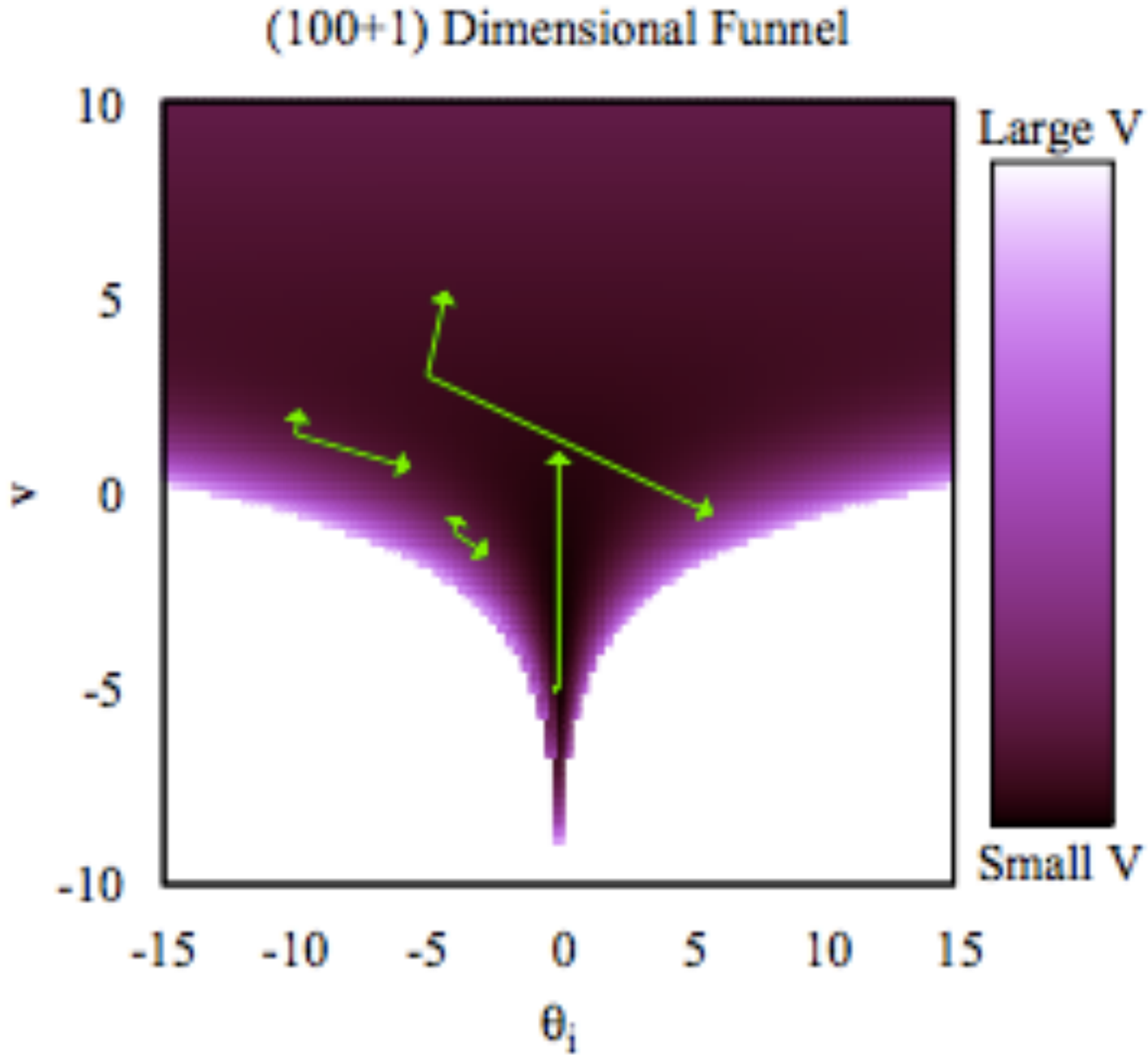
- symplectic integration diverges: good diagnostic
- sampler needs to have real small steps to not diverge, but then becomes sticky
- regions of high curvature often have high energy differences, causing trouble for microcanonical jump transitions.



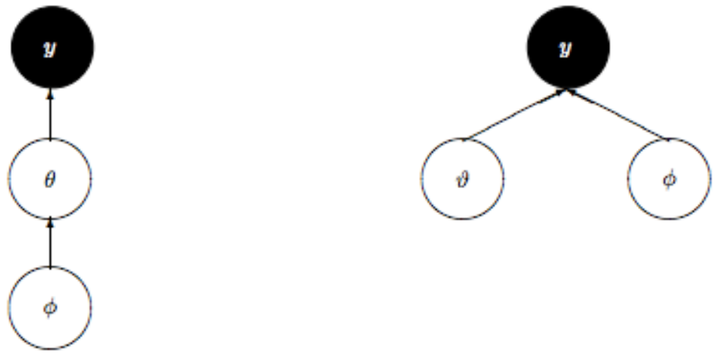
Hierarchical Models have high curvature



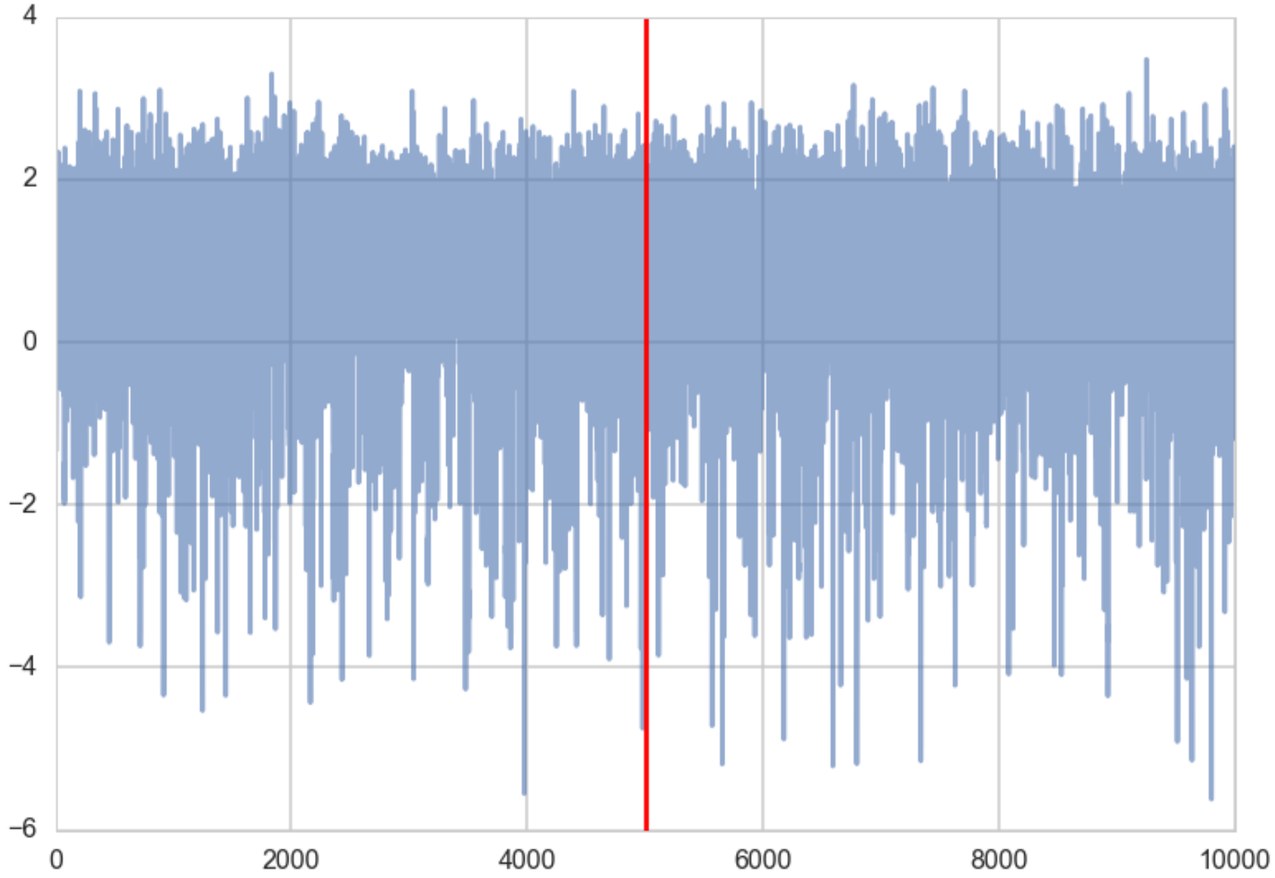
- characteristic funnel, also there in MH and gibbs
- reflects high correlation between levels in tree
- divergences occur in neck



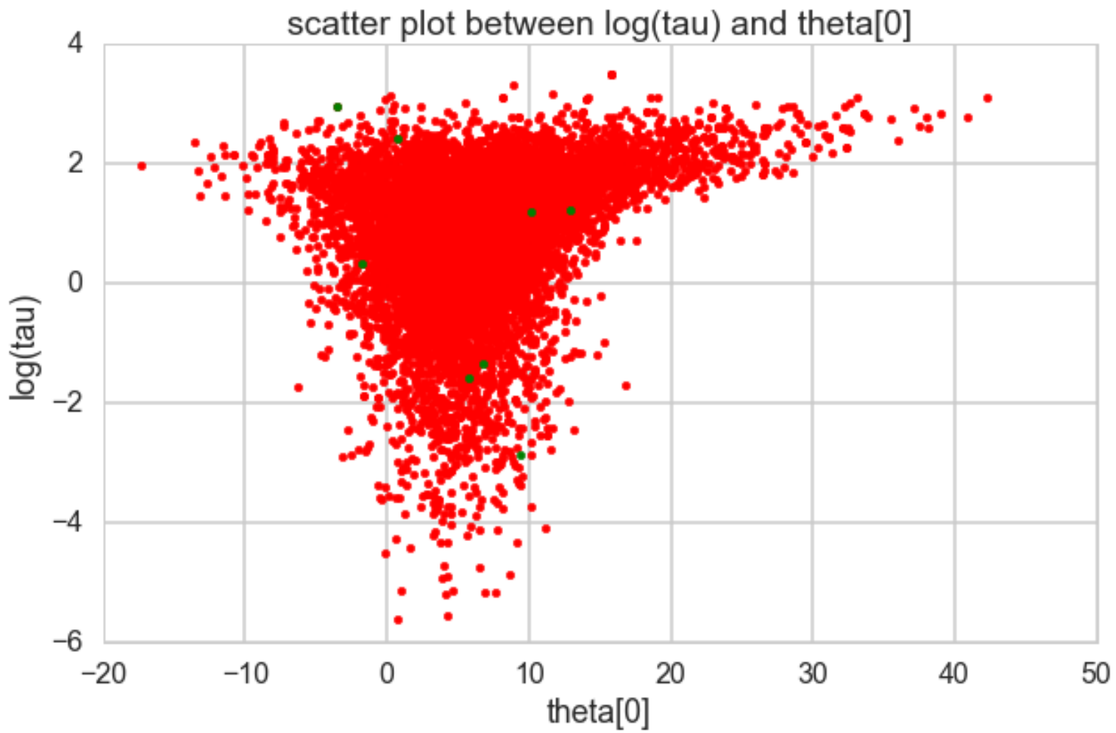
Non-centered model

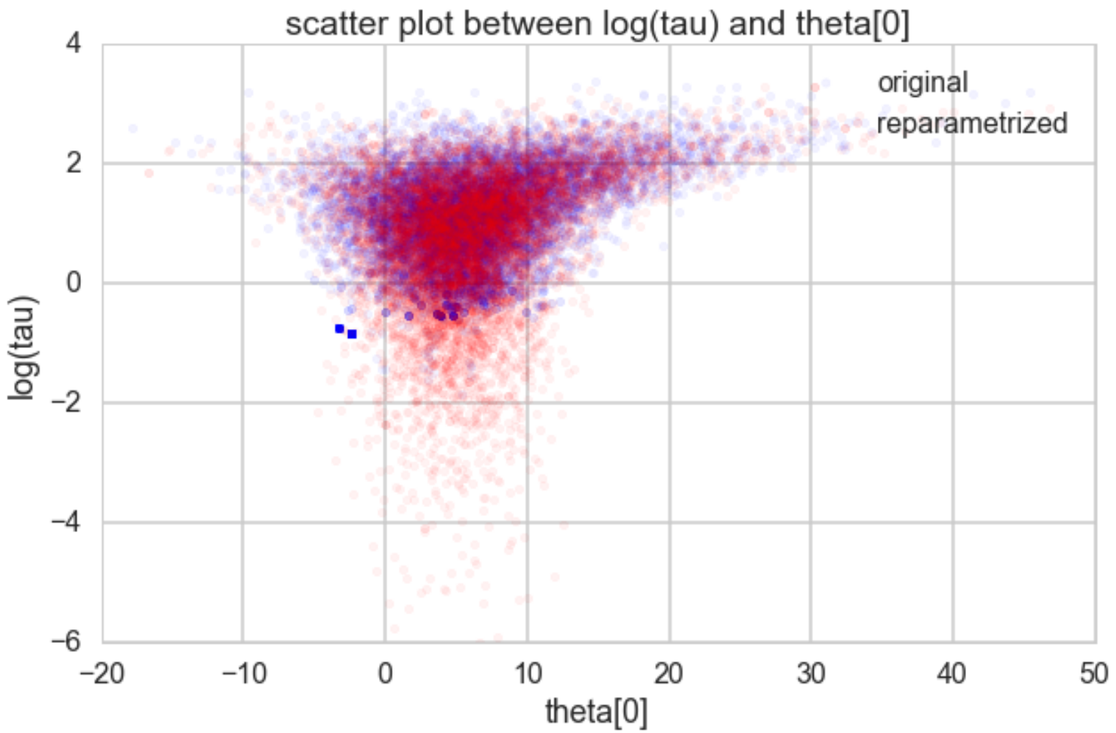


$$
\begin{aligned}
\mu &\sim \mathcal{N}(0, 5) \\
\tau &\sim \text{Half-Cauchy}(0, 5) \\
\nu_j &\sim \mathcal{N}(0, 1) \\
\theta_j &= \mu + \tau\nu_j \\
\bar{y}_j &\sim \mathcal{N}(\theta_j, \sigma_j)
\end{aligned}
$$

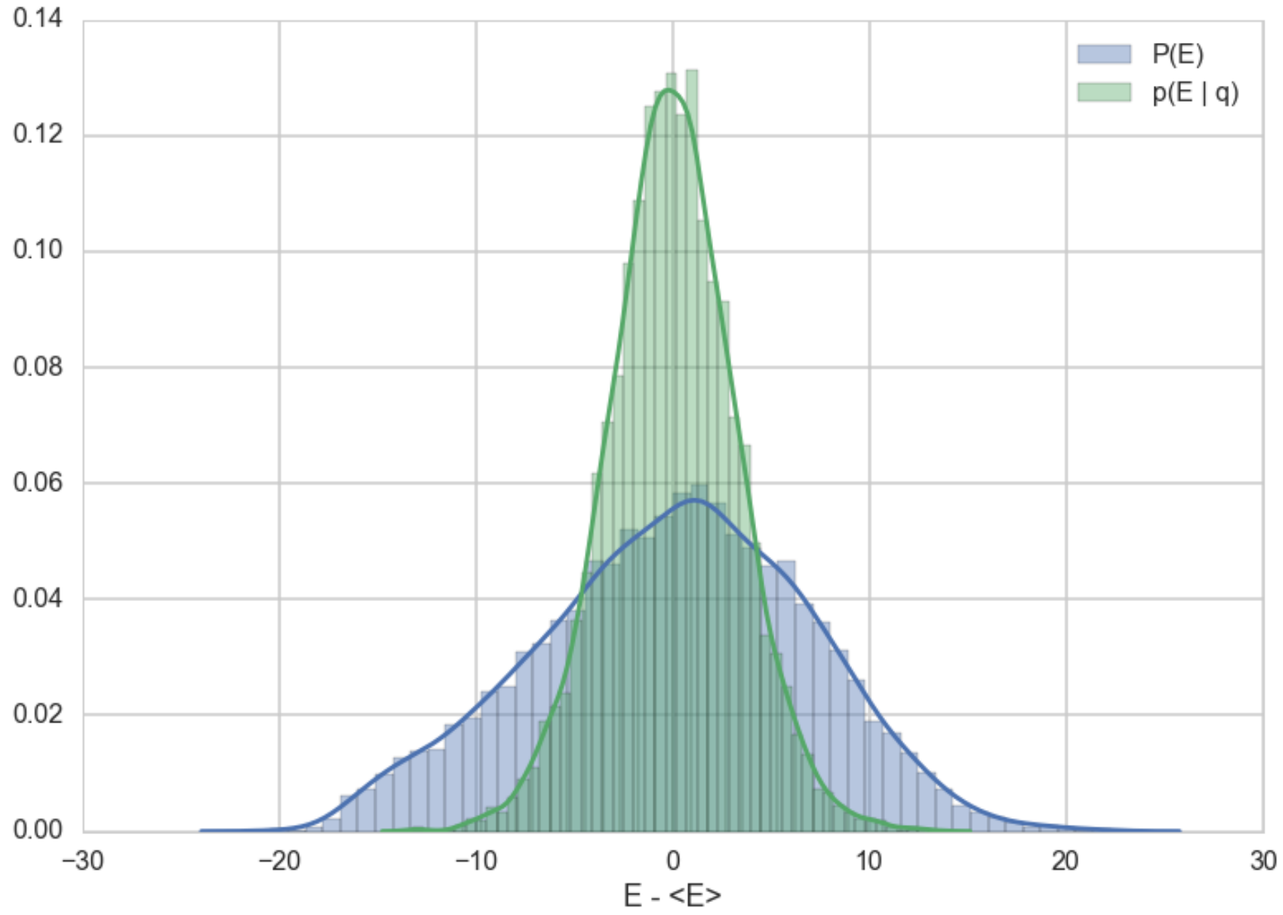


Divergences and true length of funnel





Step size effect



- lower step size ϵ better for symplectic integrators, especially in high curvature regions, but too small: return of the random walk
- if divergences persist on lowering step sizes, we are still too curved
- If Divergences infrequent, and all over. Mostly false positives. Lowering step sizes should make them go away
- check marginal energy: if has bigger tails, indicative of big energy changes in high-curvature regions not possible to boost to.

glms

- linear regression with a link. likelihoods chosen MAXENT

  $f(p_i) = \alpha + \beta x_i$ where p_i is the parameter at the ith data point.

For most GLMs, the common links we use are the *logit* link to model the space of probabilities, and the *log* link which you will use here to enforce positiveness on a parameter.

	days	monastery	y
0	1	0	0
1	1	0	1
2	1	0	1
3	1	0	2
4	1	0	0
5	1	0	1
6	1	0	2
7	1	0	1
8	1	0	1
9	1	0	0
10	1	0	4
11	1	0	1
12	1	0	4
13	1	0	3
14	1	0	1
15	1	0	0
16	1	0	2
17	1	0	1
18	1	0	1
19	1	0	1
20	1	0	1
21	1	0	1
22	1	0	1
23	1	0	1
24	1	0	0
25	1	0	1
26	1	0	1
27	1	0	1
28	1	0	2
29	1	0	2
30	7	1	6
31	7	1	2
32	7	1	7
33	7	1	3

$$y_i \sim Poisson(\lambda_i)$$

$$log(\lambda_i) = log(\frac{\mu_i}{\tau_i}) = \alpha + \beta x_i$$

λ_i is rate, μ_i is counts, τ_i is exposure.

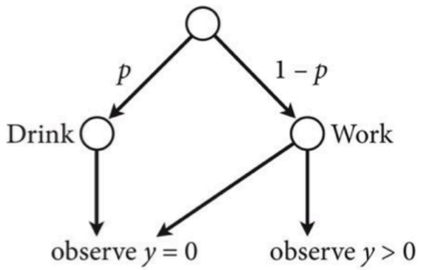
μ_i or λ_i constrained to be positive.

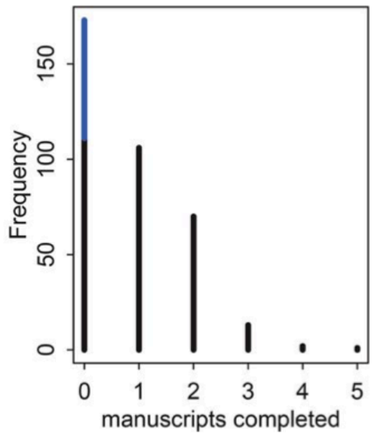
```
import theano.tensor as t
with pm.Model() as model1:
    alpha=pm.Normal("alpha", 0,100)
    beta=pm.Normal("beta", 0,1)
    logmu = t.log(df.days)+alpha+beta*df.monastery
    y = pm.Poisson("obsv", mu=t.exp(logmu), observed=df.y)
    lambda0 = pm.Deterministic("lambda0", t.exp(alpha))
    lambda1 = pm.Deterministic("lambda1", t.exp(alpha + beta))
```

Zero Inflated Poisson Mixture model

$$\mathcal{L}(y = 0) = p + (1 - p)e^{-\lambda},$$

$$\mathcal{L}(y \neq 0) = (1 - p)\frac{\lambda^y e^{-\lambda}}{y!}$$

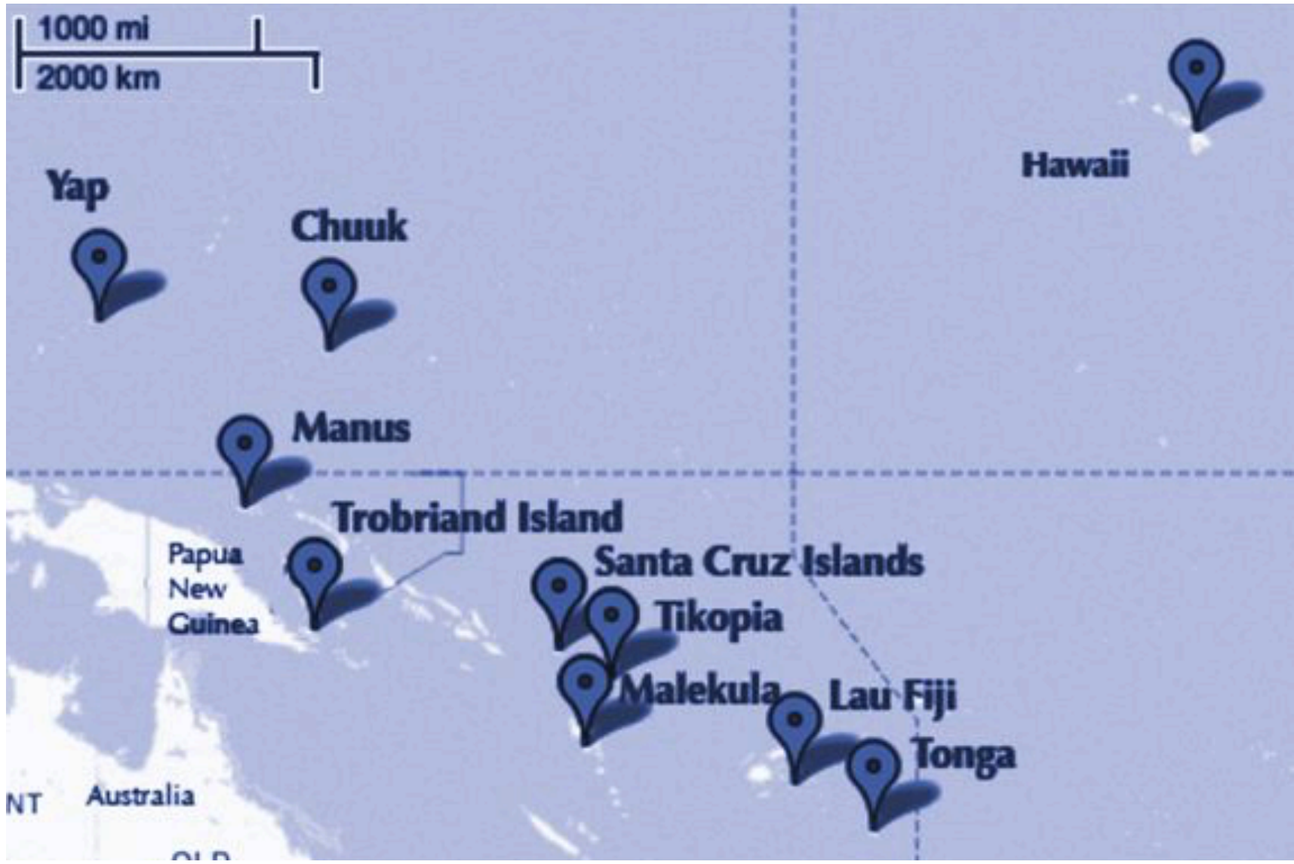




Oceanic tools

From Mcelreath:

The island societies of Oceania provide a natural experiment in technological evolution. Different historical island populations possessed tool kits of different size. These kits include fish hooks, axes, boats, hand plows, and many other types of tools. A number of theories predict that larger populations will both develop and sustain more complex tool kits. So the natural variation in population size induced by natural variation in island size in Oceania provides a natural experiment to test these ideas. It's also suggested that contact rates among populations effectively increase population size, as it's relevant to technological evolution. So variation in contact rates among Oceanic societies is also relevant. (McElreath 313)



Model M1

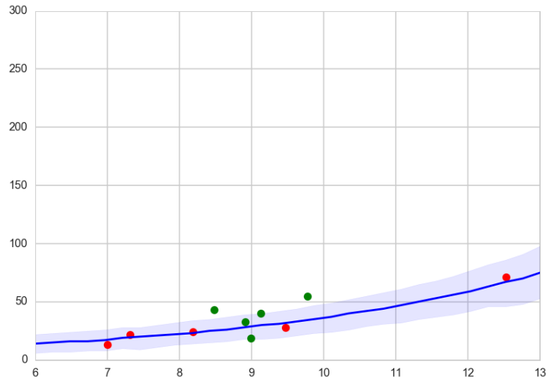
	culture	population	contact	total_tools	mean_TU	logpop	clevel
0	Malekula	1100	low	13	3.2	7.003065	0
1	Tikopia	1500	low	22	4.7	7.313220	0
2	Santa Cruz	3600	low	24	4.0	8.188689	0
3	Yap	4791	high	43	5.0	8.474494	1
4	Lau Fiji	7400	high	33	5.0	8.909235	1
5	Trobriand	8000	high	19	4.0	8.987197	1
6	Chuuk	9200	high	40	3.8	9.126959	1
7	Manus	13000	low	28	6.6	9.472705	0
8	Tonga	17500	high	55	5.4	9.769956	1
9	Hawaii	275000	low	71	6.6	12.524526	0

$$
\begin{aligned}
T_i &\sim Poisson(\lambda_i) \\
log(\lambda_i) &= \alpha + \beta_P log(P_i) + \beta_C C_i + \beta_{PC} C_i log(P_i) \\
\alpha &\sim N(0, 100) \\
\beta_P &\sim N(0, 1) \\
\beta_C &\sim N(0, 1) \\
\beta_{PC} &\sim N(0, 1)
\end{aligned}
$$

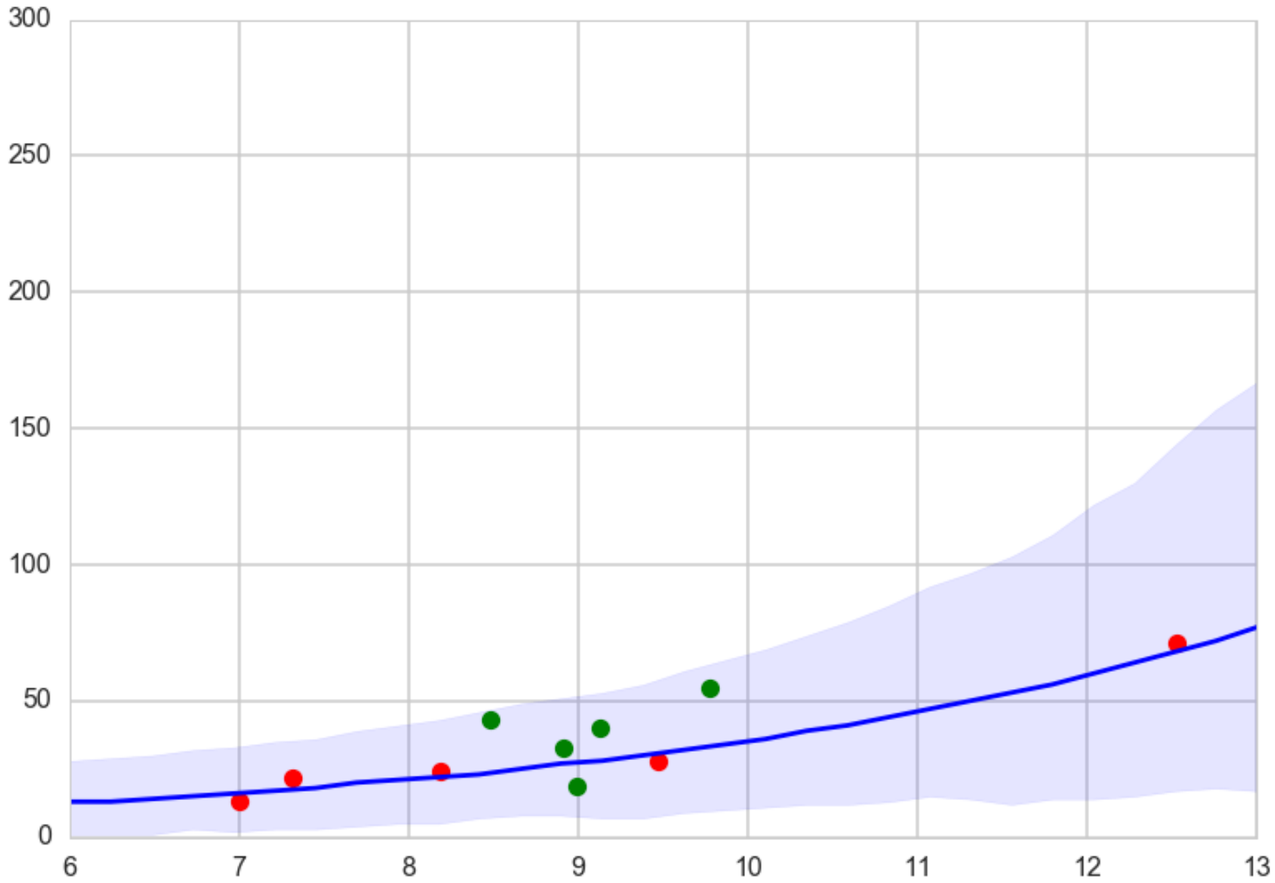
```
with pm.Model() as m1:
    betap = pm.Normal("betap", 0, 1)
    betac = pm.Normal("betac", 0, 1)
    betapc = pm.Normal("betapc", 0, 1)
    alpha = pm.Normal("alpha", 0, 100)
    loglam = alpha + betap*df.logpop +
        betac*df.clevel + betapc*df.clevel*df.logpop
    y = pm.Poisson("ntools", mu=t.exp(loglam), observed=df.total_tools)
```

```
with m1:
    trace=pm.sample(10000, njobs=2)
Average ELBO = -55.784:
100%|██████████| 200000/200000 [00:15<00:00, 13019.16it/s]   12683.03it/s]
100%|██████████| 10000/10000 [01:59<00:00, 83.80it/s]
```

Overdispersion fixed by varying intercepts



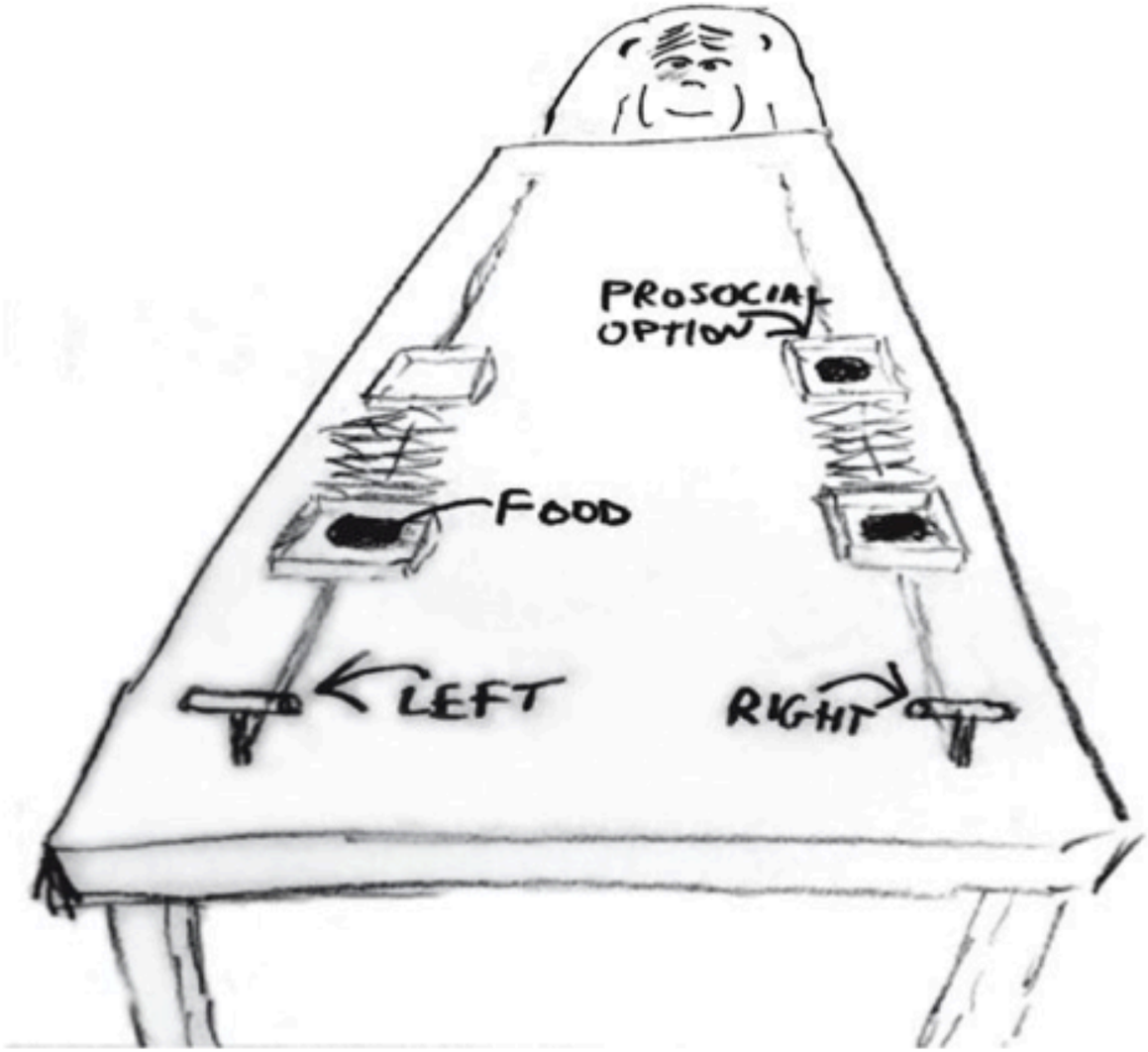

```
with pm.Model() as m3c:
    betap = pm.Normal("betap", 0, 1)
    alpha = pm.Normal("alpha", 0, 100)
    sigmasoc = pm.HalfCauchy("sigmasoc", 1)
    alphasoc = pm.Normal("alphasoc", 0, sigmasoc, shape=df.shape[0])
    loglam = alpha + alphasoc + betap*df.logpop_c
    y = pm.Poisson("ntools", mu=t.exp(loglam), observed=df.total_tools)
```



Prosocial chimps

$$
\begin{aligned}
L_i &\sim \text{Binomial}(1, p_i) \\
\text{logit}(p_i) &= \alpha + \alpha_{\text{ACTOR}[i]} + \alpha_{\text{BLOCK}[i]} + (\beta_P + \beta_{PC} C_i) P_i \\
\alpha_{\text{ACTOR}} &\sim \text{Normal}(0, \sigma_{\text{ACTOR}}) \\
\alpha_{\text{BLOCK}} &\sim \text{Normal}(0, \sigma_{\text{BLOCK}}) \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta_P &\sim \text{Normal}(0, 10) \\
\beta_{PC} &\sim \text{Normal}(0, 10) \\
\sigma_{\text{ACTOR}} &\sim \text{HalfCauchy}(0, 1) \\
\sigma_{\text{BLOCK}} &\sim \text{HalfCauchy}(0, 1)
\end{aligned}
$$

```
with pm.Model() as vipb:
        betapc = pm.Normal("betapc", 0, 10)
        betap = pm.Normal("betap", 0, 10)
        alpha = pm.Normal('alpha', 0, 10)
        sigma_actor = pm.HalfCauchy("sigma_actor", 1)
        sigma_block = pm.HalfCauchy("sigma_block", 1)
        alpha_actor = pm.Normal('alpha_actor', 0, sigma_actor, shape=7)
        alpha_block = pm.Normal('alpha_block', 0, sigma_block, shape=6)
        logitpi = alpha + alpha_actor[df.index//72] +
            alpha_block[df.block.values -1]+ (betap +
             betapc*df.condition)*df.prosoc_left
        o = pm.Bernoulli("pulled_left", p=pm.math.invlogit(logitpi), observed=df.pulled_left)
```



MODEL CHECKING

Multiple replications of the posterior predictive

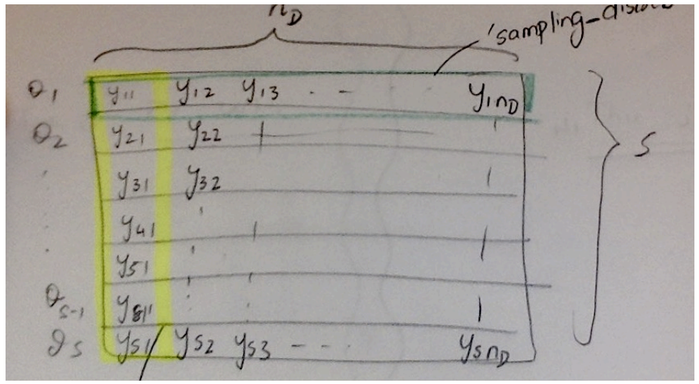
$$p(\{y^*\}) = \int p(\{y^*\}|\theta)p(\theta|\mathcal{D})d\theta, \text{ observed data: } \mathcal{D} = \{y\}$$

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

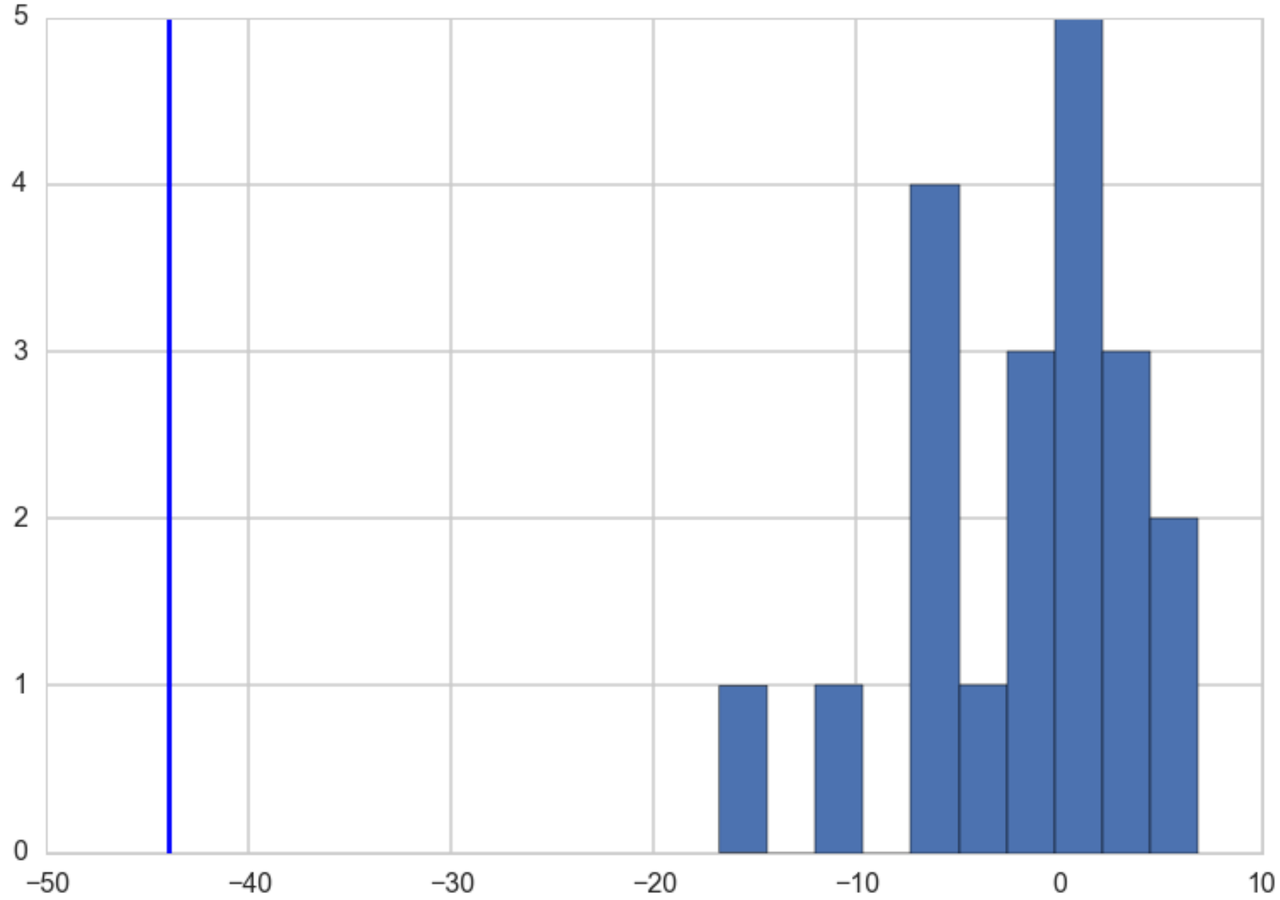
$\{y_r\}$ comes from posterior predictive, and if there are covariates $\{x^*\}$, then $\{y_r\}$ is calculated at those covariates only (`sample_ppc`).

Departure from usual predictive sampling

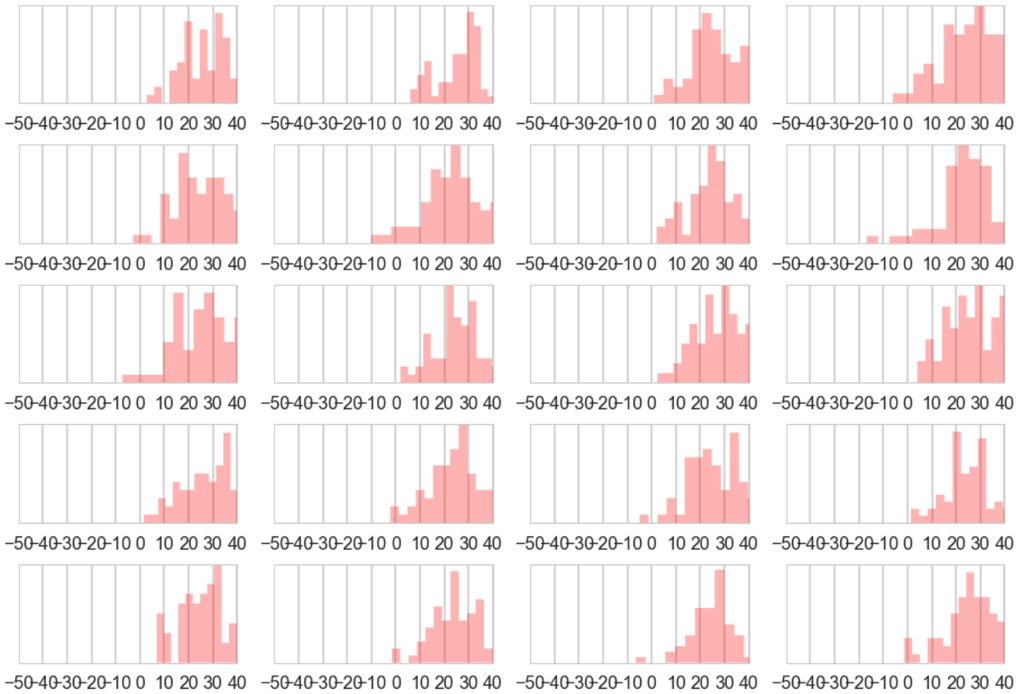
Sample an entire $\{y_r\}$ at each θ from trace.

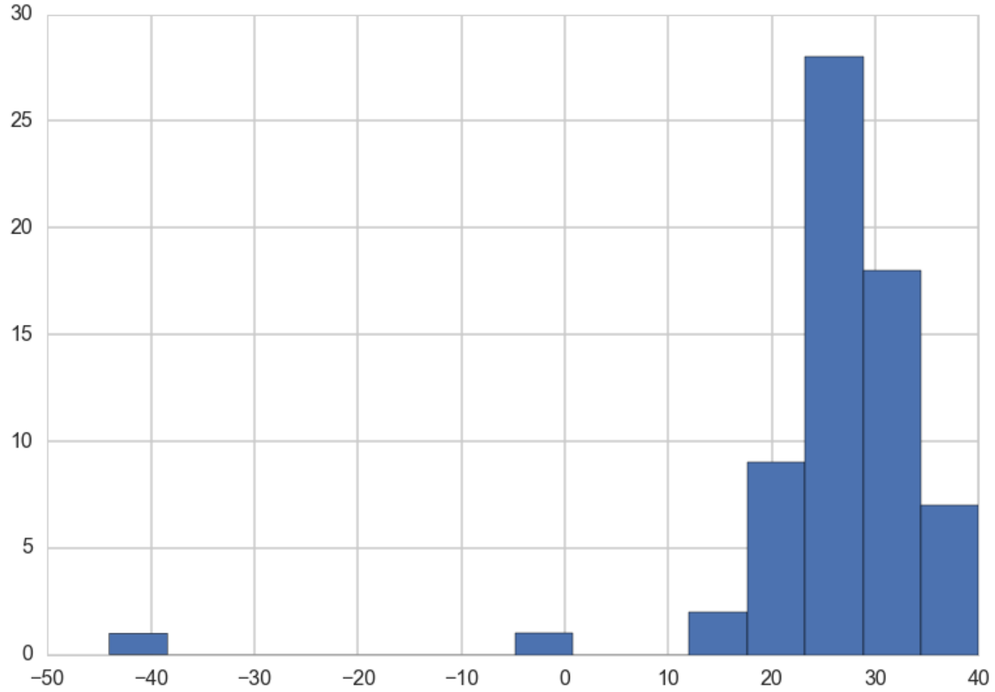


For example the minimum value of speed of light in 20 predictive replications.

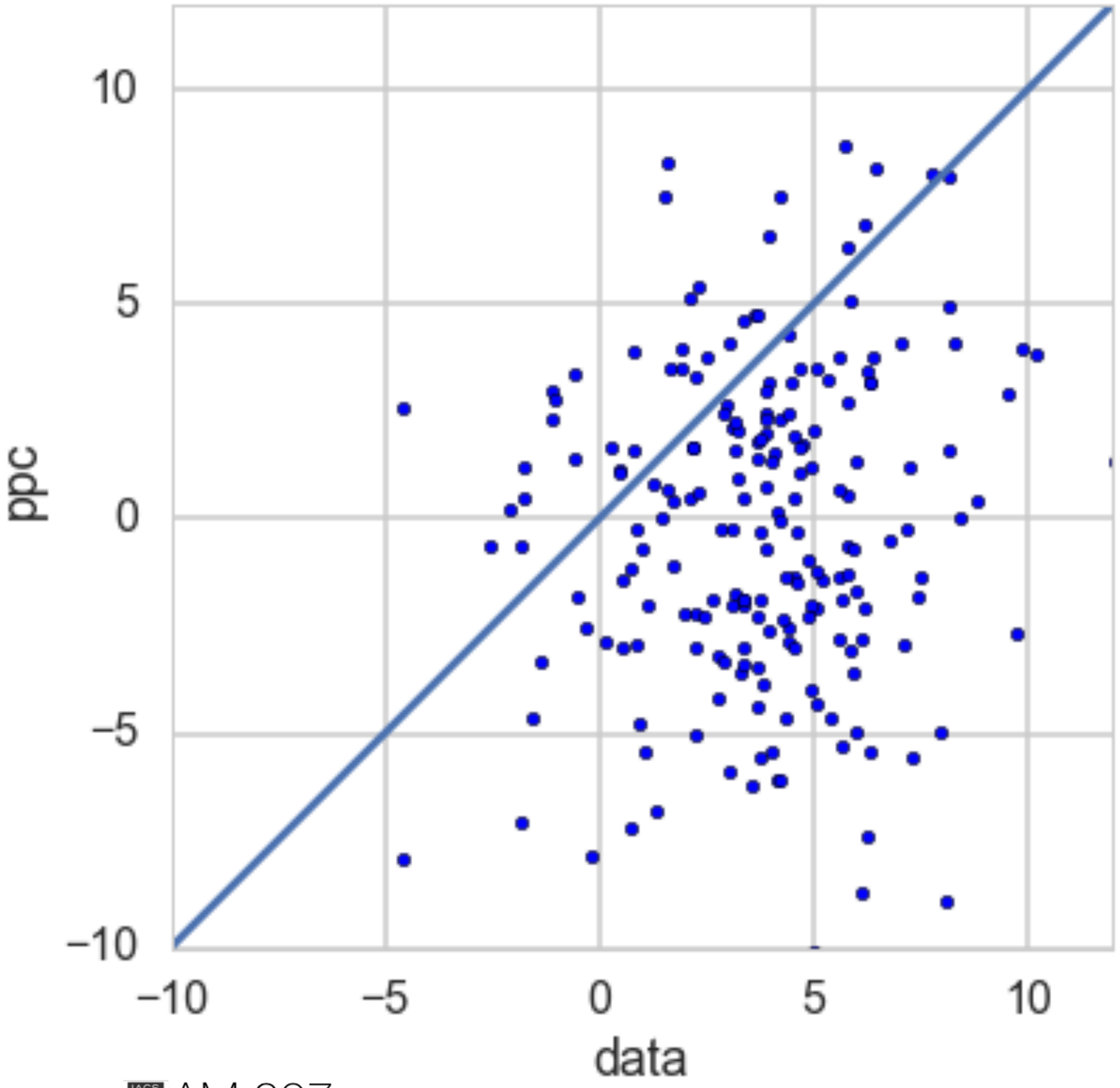


Visual Checking





Do these even look similar??



Bayesian p-values

$$p_B = Pr(T(\{y_r\}, \theta) \geq T(\{y\}, \theta)|\{y\}),$$

probability over the posterior and posterior predictive (that is, the joint distribution, $p(\theta, \{y_r\}|\{y\})$).

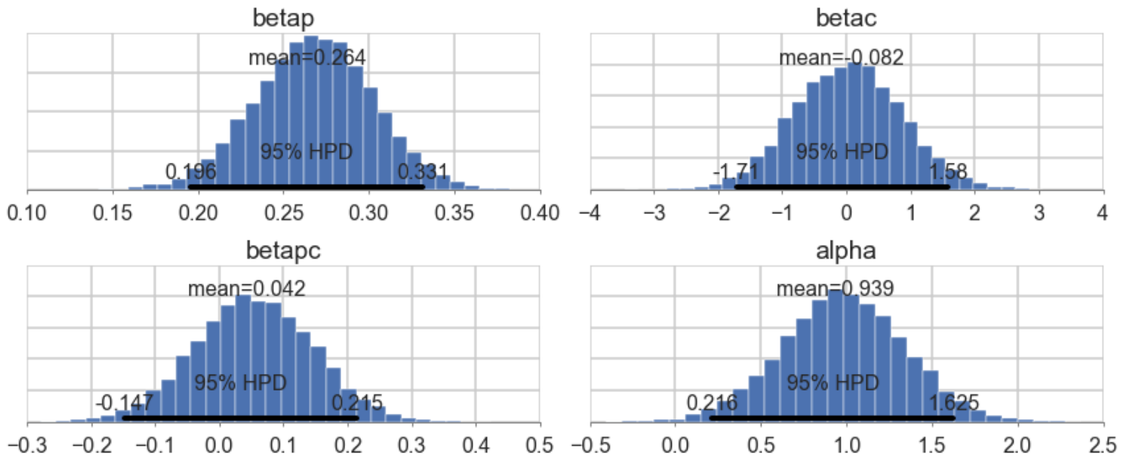
$$p_B = \int d\theta \, d\{y_r\} \, I(T(\{y_r\}, \theta) \geq T(\{y\}, \theta)) \, p(\{y_r\}|\theta) p(\theta|\{y\})$$

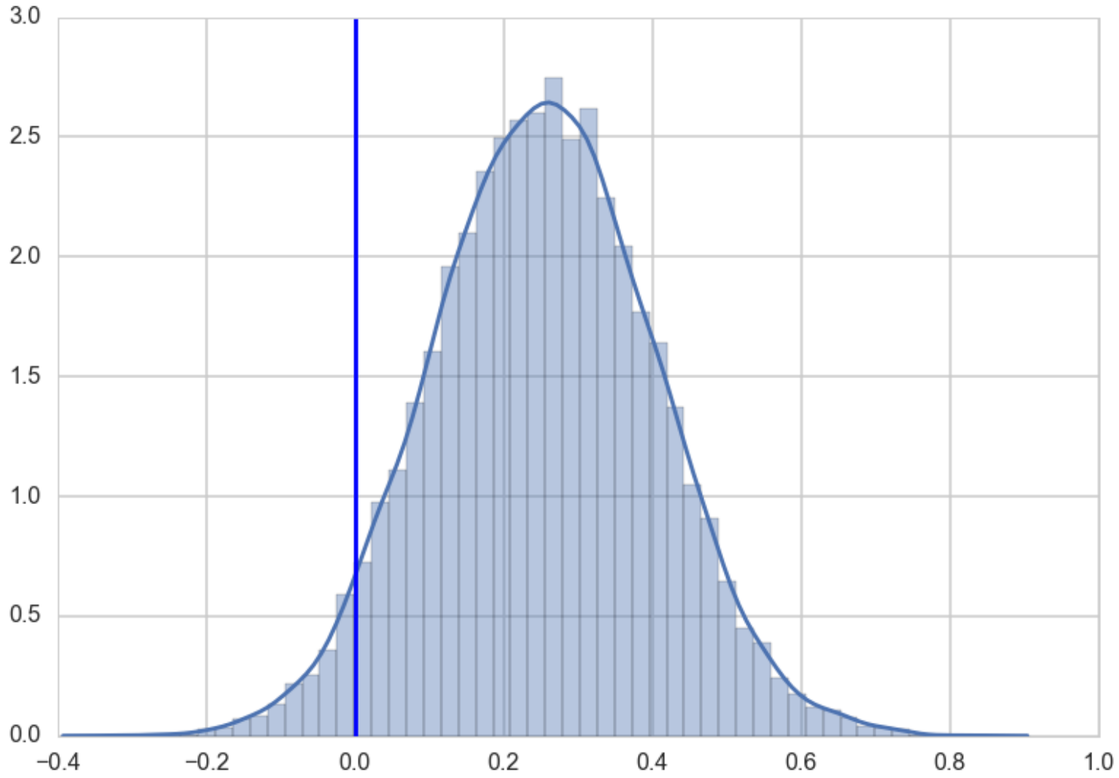
using $p(\{y_r\}|\theta, \{y\}) = p(\{y_r\}|\theta)$.

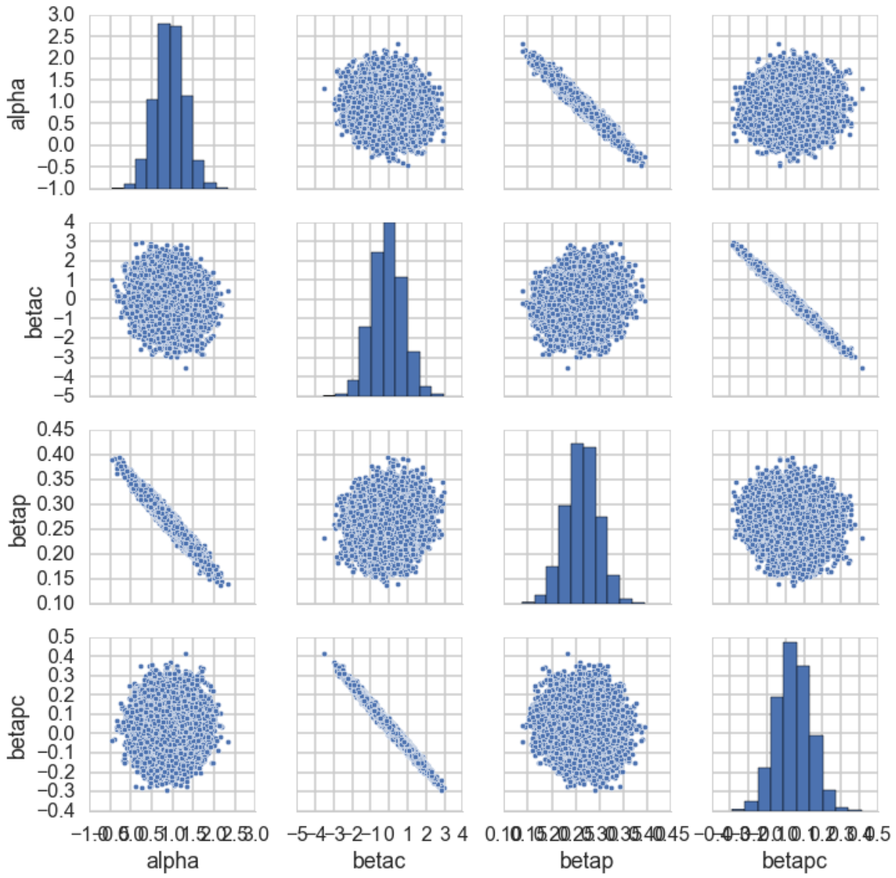
Appropriate usage

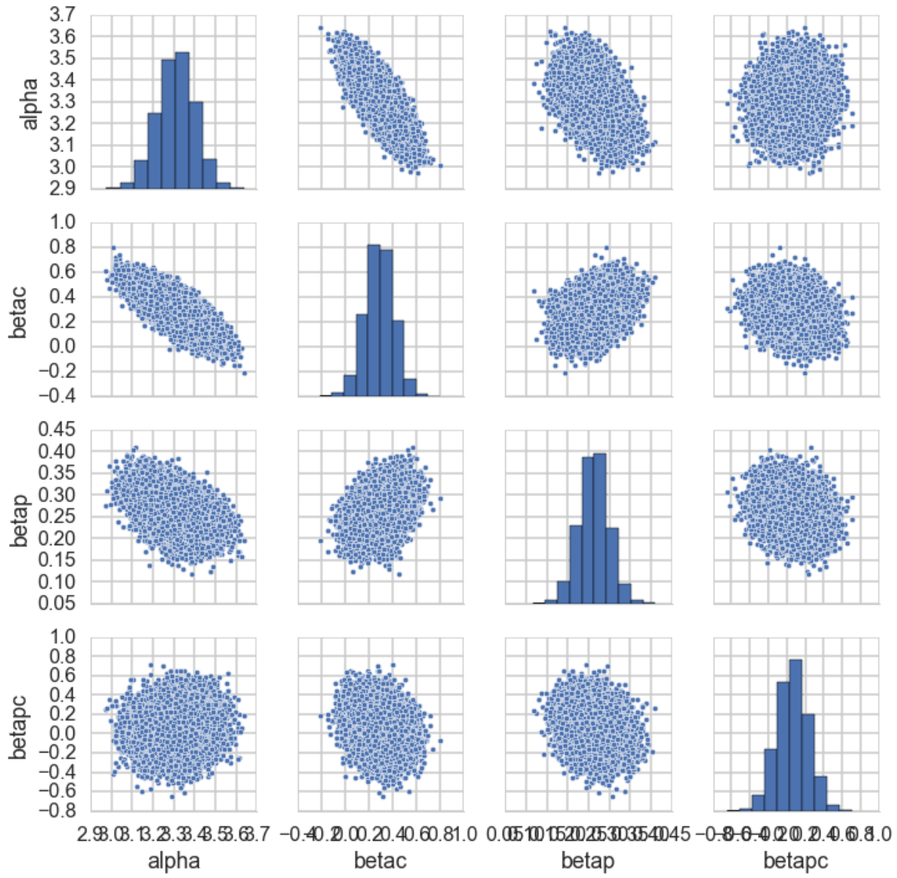
Gelman: *Finding an extreme p-value and thus 'rejecting' a model is never the end of an analysis; the departures of the test quantity in question from its posterior predictive distribution will often suggest improvements of the model or places to check the data, as in the speed of light example. Moreover, even when the current model seems appropriate for drawing inferences (in that no unusual deviations between the model and the data are found), the next scientific step will often be a more rigorous experiment incorporating additional factors, thereby providing better data.*

Oceanic Tools Posteriors and checking









Solution is centering, see in n_{eff}, can use model comparison

MODEL COMPARISON AND ENSEMBLING

Deviance

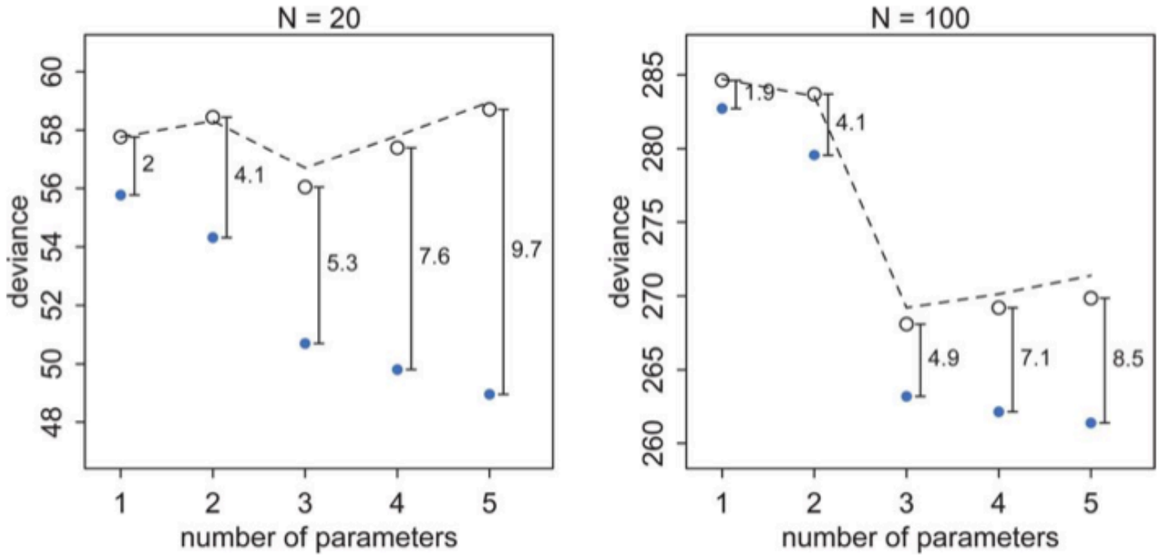
$$D(q) = -2 \sum_i log(q_i),$$

then

$$D_{KL}(p, q) - D_{KL}(p, r) = \frac{2}{N}(D(q) - D(r))$$

More generally: $D(q) = -\frac{N}{2} E_p[log(q)]$

AIC



Akaike Information Criterion, or AIC:

$$AIC = D_{train} + 2p$$

$$D_{train} = -2 * log(p(y|\theta_{mle})$$

- multivariate gaussian posterior
- flat priors
- data >> parameters

DIC

- uses the posterior distribution, calculable from MCMC.
- multivariate gaussian posterior distribution.

$$D_{train} = -2 * log(p(y|\theta_{postmean})$$

$$DIC = D_{train} + 2p_D \text{ where}$$

$$p_{DIC} = 2 * (log(p(y|\theta_{postmean}) - E_{post}[log(p(y|\theta)]) \text{ (by monte carlo)}$$

alternative fomulation for p_D, guaranteed to be positive, is

$$p_D = 2 * Var_{post}[log(p(y|\theta_{postmean}))]$$

Bayesian deviance

- $D(q) = -\frac{N}{2} E_p[log(pp(y))]$ posterior predictive for points y on the test set or future data
- replace joint posterior predictive over new points y by product of marginals: ELPD: $\sum_i E_p[log(pp(y_i))]$
- Since we do not know the true distribution p, replace elpd: $\sum_i E_p[log(pp(y_i))]$ by the computed "log pointwise predictive density" (lppd) **in-sample**

$$\sum_j log \langle p(y_j|\theta) \rangle = \sum_j log \left(\frac{1}{S} \sum_s p(y_j|\theta_s) \right)$$

WAIC

$$WAIC = lppd + 2p_W$$

where

$$p_W = 2\sum_i (log(E_{post}[p(y_i|\theta)] - E_{post}[log(p(y_i|\theta))])$$

Once again this can be estimated by

$$\sum_i Var_{post}[log(p(y_i|\theta))]$$

Centered vs Uncentered

	WAIC	pWAIC	dWAIC	weight	SE	dSE	warning
name							
m2c_nopc	79.3591	4.39013	0	0.846327	11.0543	0	1
m1c	83.8617	6.94776	4.50259	0.0890866	12.2027	4.00079	1
m2c_onlyp	84.5049	3.77558	5.14581	0.0645862	8.91335	19.3282	1
m2c_onlyic	141.327	8.10745	61.9681	2.96038e-14	31.6664	339.158	1
m2c_onlyc	152.975	18.1559	73.6157	8.75158e-17	46.6488	679.109	1

	WAIC	pWAIC	dWAIC	weight	SE	dSE	warning
name							
m2_nopc	79.1059	4.22647	0	0.61959	11.0612	0	1
m1	80.3046	5.03686	1.19871	0.340258	11.3985	0.571957	1
m2_onlyp	84.5787	3.84888	5.47276	0.0401523	8.98146	20.1717	1
m2_onlyic	141.327	8.10745	62.2212	1.90956e-14	31.6664	338.568	1
m2_onlyc	152.975	18.1559	73.8689	5.64512e-17	46.6488	678.014	1

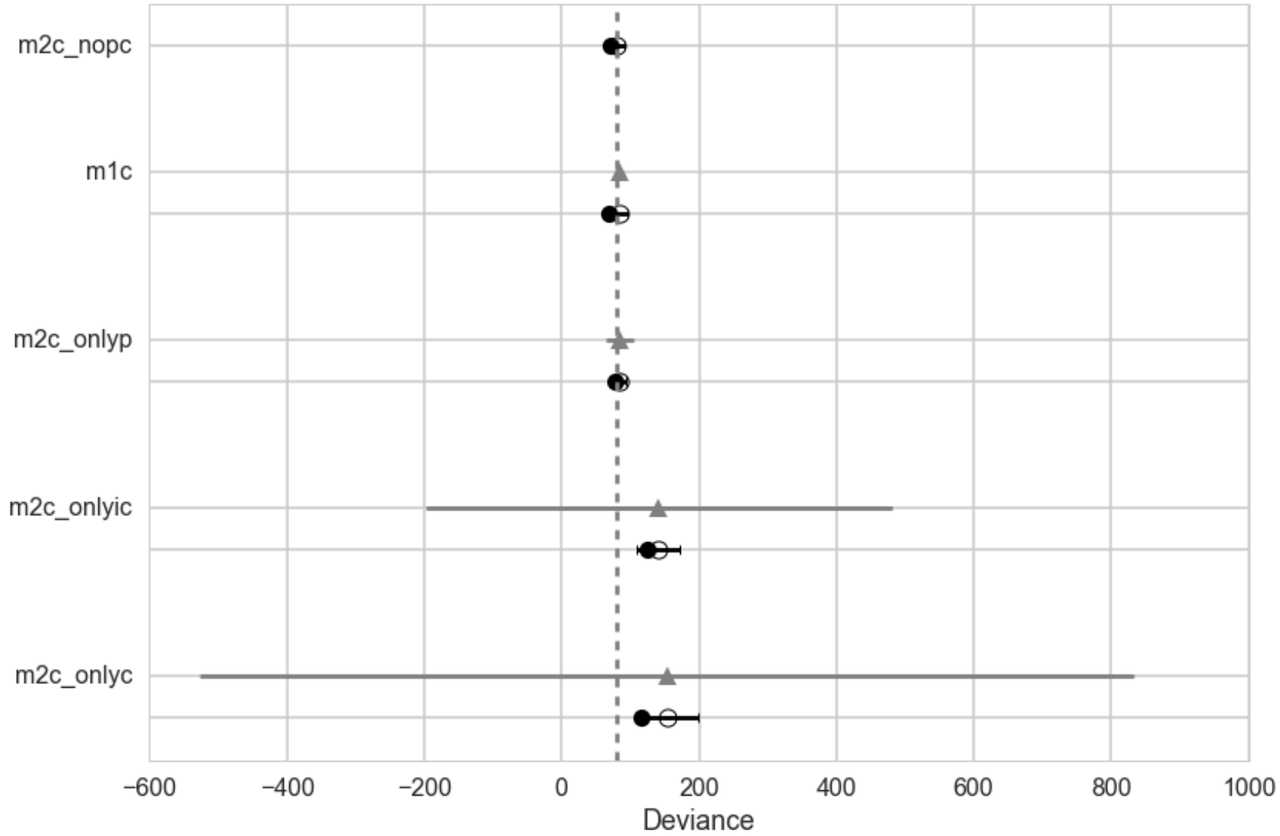
interaction is overfit. centering decorrelates

Definitions

- dWAIC is the difference between each WAIC and the lowest WAIC.
- SE is the standard error of the WAIC estimate.
- dSE is the standard error of the difference in WAIC between each model and the top-ranked model.

$$w_i = \frac{exp(-\frac{1}{2}dWAIC_i)}{\sum_j exp(-\frac{1}{2}dWAIC_j)}$$

read each weight as an estimated probability that each model will perform best on future data.



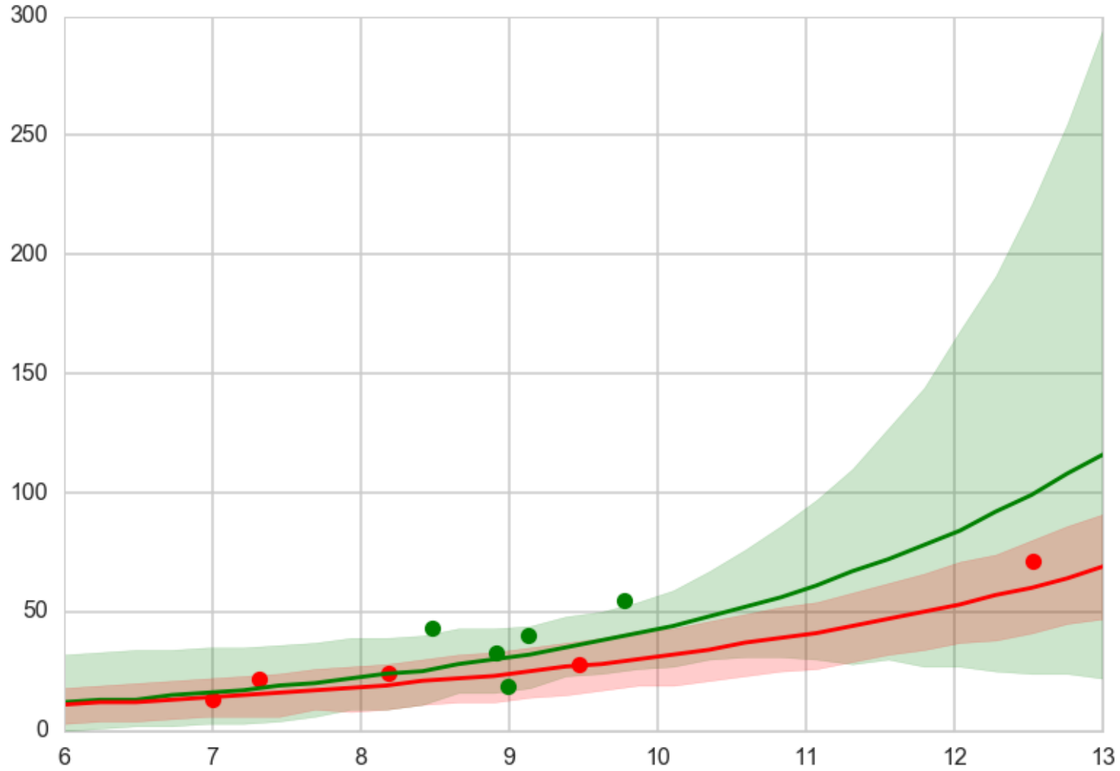
Bayesian Model Averaging

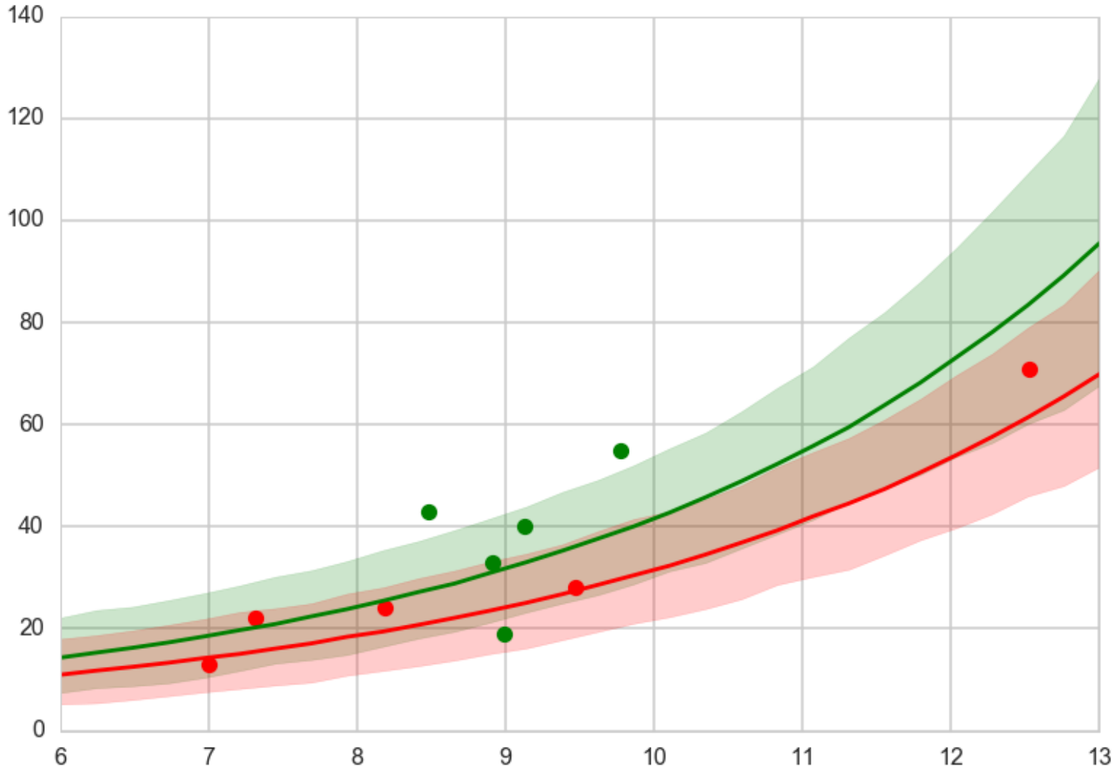
$$p_{BMA}(y^*|x^*, D) = \sum_k p(y^*|x^*, D, M_k)p(M_k|D)$$

where the averaging is with repect to weights $w_k = p(M_k|D)$, the posterior probabilities of the models M_k.

Use the weights from the WAIC

Counterfactual PP and ensembling via weights





Model comparison

The key idea in model comparison is that we will sort our average utilities in some order. The exact values are not important, and may be computed with respect to some true distribution or true-belief distribution M_{tb}.

$$\bar{u}(M_k, \hat{a}_k) = \int dy^* u(\hat{a}_k, y^*) p(y^*|D, M_{tb})$$

where $\hat{a}_k$ is the optimal prediction under the model M_k. Now we compare the actions, that is, we want:

$$\hat{M} = \arg\max_k \bar{u}(M_k, \hat{a}_k)$$

No calibration, but calculating the standard error of the difference can be used to see if the difference is significant, as we did with the WAIC score

it is tempting to use information criteria to compare models with different likelihood functions. Is a Gaussian or binomial better? Can't we just let WAIC sort it out?
Unfortunately, WAIC (or any other information criterion) cannot sort it out. The problem is that deviance is part normalizing constant. The constant affects the absolute magnitude of the deviance, but it doesn't affect fit to data.

— *McElreath*

Use Cross-Validation in such cases

LOOCV

- Fit a model on N-1 data points, and use the Nth point as a validation point.
- the N-point and N-1 point posteriors are likely to be quite similar, use importance sampling. Fit the full posterior once. Then we have

$$w_s = \frac{p(\theta_s|y_{-i})}{p(\theta_s|y)} \propto \frac{1}{p(y_i|\theta_s, y_{-i})}$$

- the importance sampling weights can be unstablein the tails, pymc (`pm.loo`) fits a generalized pareto to the tail (largest 20% importance ratios) for each held out data point i (a MLE fit). Smooths out any large variations.

$$elpd_{loo} = \sum_i log(p(y_i|y_{-i})) = \sum_i log\left(\frac{\sum_s w_{is} p(y_i|\theta_s)}{\sum_s w_{is}}\right)$$

What should you use?

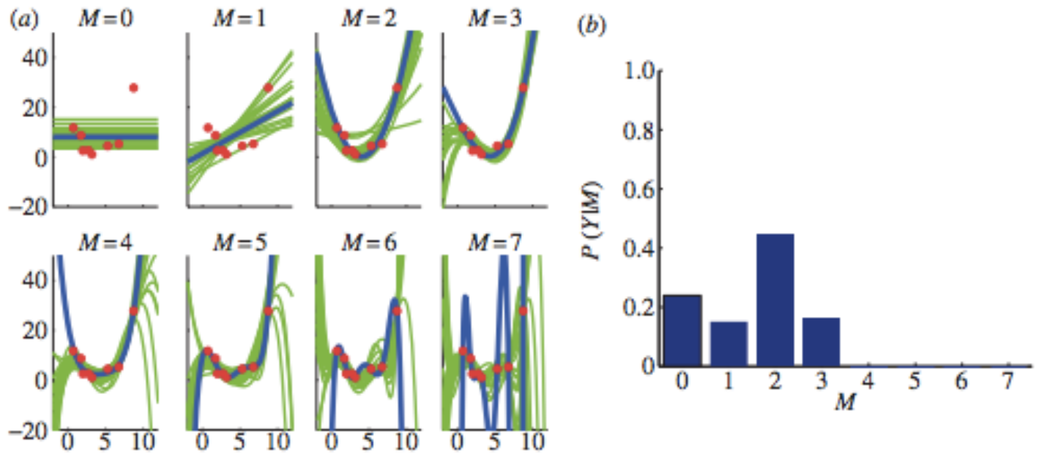
1. LOOCV and WAIC are fine. The former can be used for models not having the same likelihood, the latter can be used with models having the same likelihood.

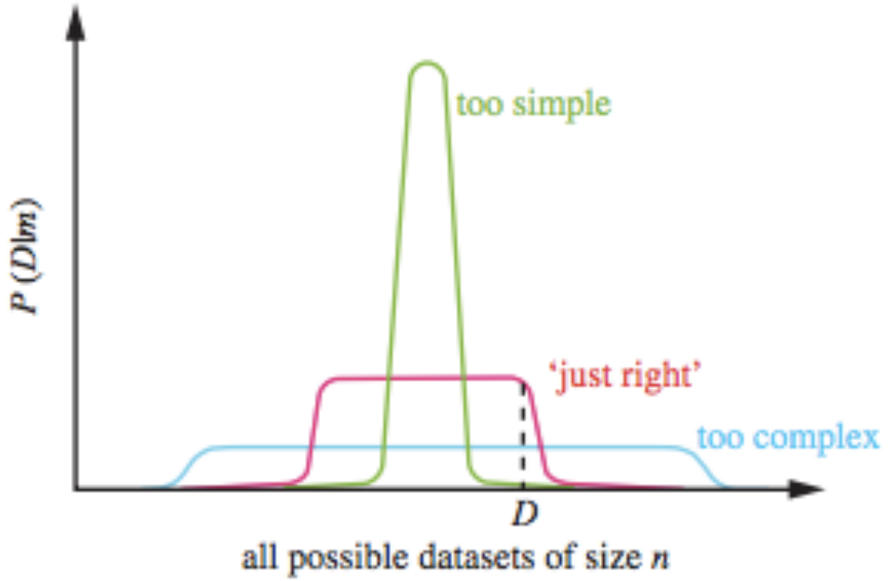
2. WAIC is fast and computationally less intensive, so for same-likelihood models (especially nested models where you are really performing feature selection), it is the first line of attack

3. One does not always have to do model selection. Sometimes just do posterior predictive checks to see how the predictions are, and you might deem it fine.

4. For hierarchical models, WAIC is best for predictive performance within an existing cluster or group. Cross validation is best for new observations from new groups

Evidence for model comparison: Occams Razor





DECISION THEORY

Bayes Action

First define the distribution-averaged utility:

$$\bar{u}(a) = \int d\omega \, u(a, \omega) \, p(\omega|D)$$

We then find the a that maximizes this utility:

$$\hat{a} = \arg\max_a \bar{u}(a)$$

This action is called the **bayes action**.

The resulting maximized expected utility is given by:

$$\bar{u}(\hat{a}, p) = \bar{u}(\hat{a}) = \int d\omega \, u(\hat{a}, \omega) \, p(\omega|D),$$

sometimes referred to as the entropy function, and an associate **divergence** can be defined:

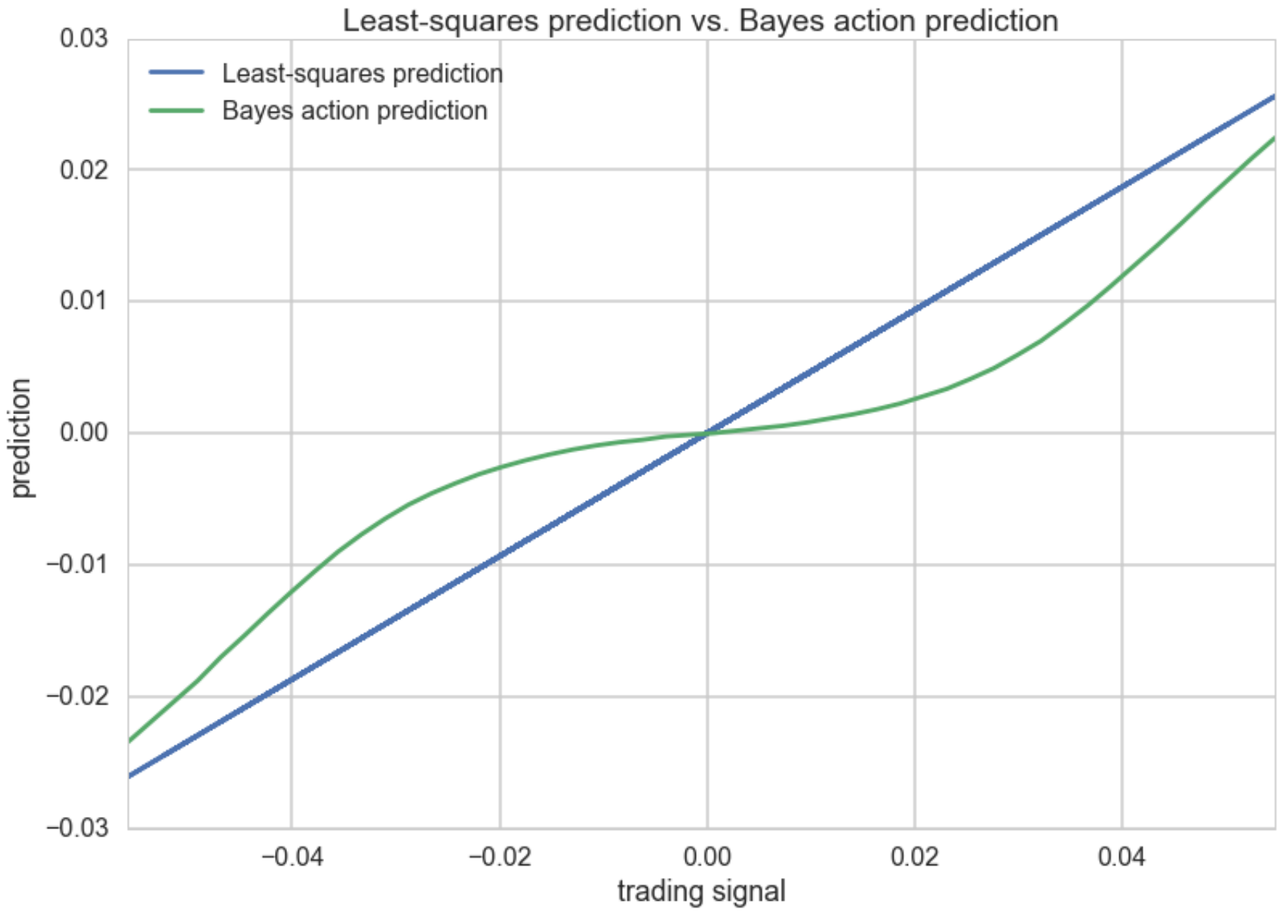
$$d(a, p) = \bar{u}(p, p) - \bar{u}(a, p)$$

Then one can think of minimizing $d(a, p)$ with respect to a to get $\hat{a}$, so that this discrepancy can be thought of as a loss function.

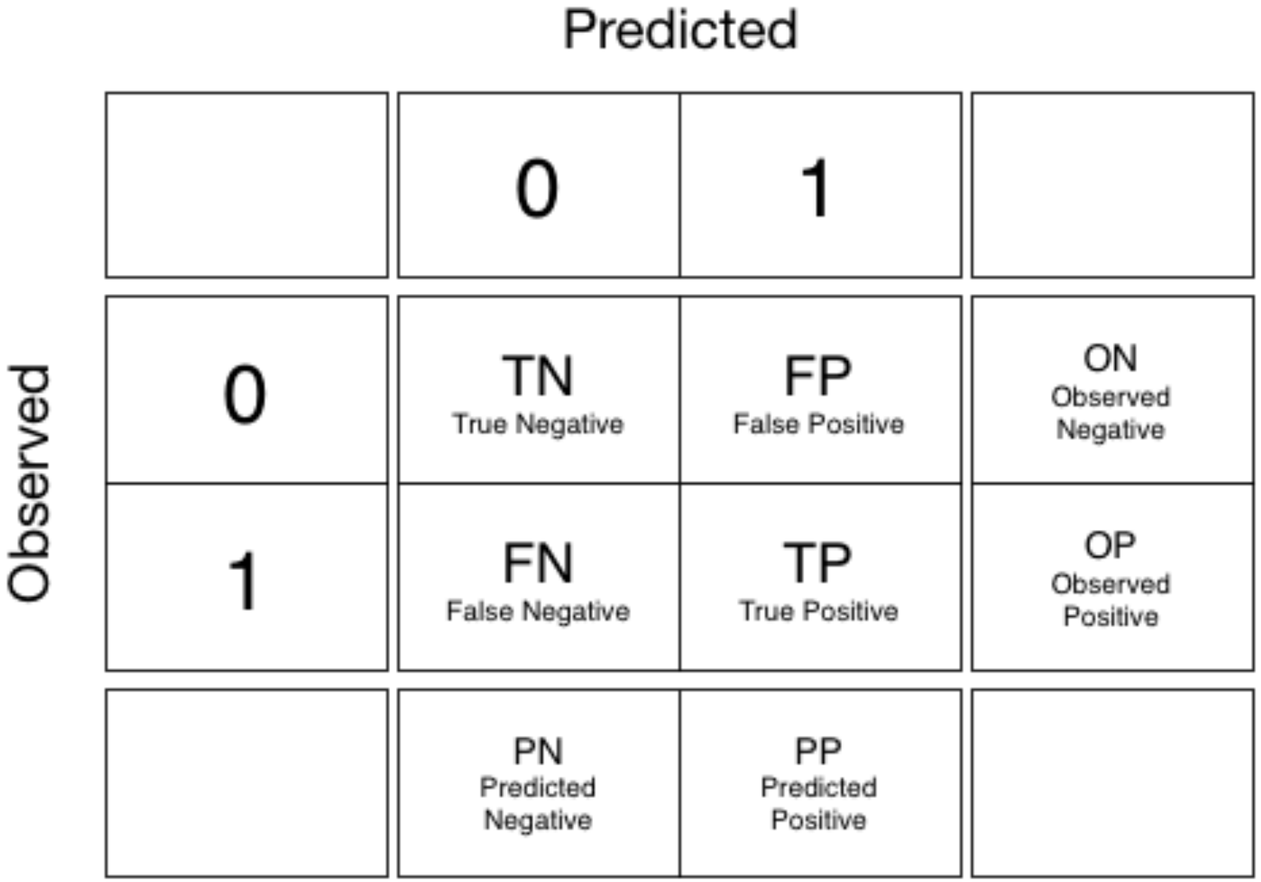
Custom Loss

```
def stock_loss(stock_return, pred, alpha = 100.):
    if stock_return * pred < 0:
        #opposite signs, not good
        return alpha*pred**2 - np.sign(stock_return)*pred \
                        + abs(stock_return)
    else:
        return abs(stock_return - pred)
#posterior predictive samples at every x
possible_outcomes = lambda signal: alpha_samples + \
    beta_samples*signal + noise


opt_predictions = np.zeros(50)
trading_signals =  np.linspace(X.min(), X.max(), 50)
for i, _signal in enumerate(trading_signals):
        _possible_outcomes = possible_outcomes(_signal)
        #expected loss over posterior predictive
        tomin = lambda pred: stock_loss(_possible_outcomes, pred).mean()
        #bayes action minimizes expected loss
        opt_predictions[i] = fmin(tomin, 0, disp = False)
```



Classification Risk (2 classes)



$$R_a(x) = \sum_y l(y, a(x))p(y|x)$$

$$R_1(x) = l(1,1)p(1|x) + l(0,1)p(0|x),$$

$$R_0(x) = l(1,0)p(1|x) + l(0,0)p(0|x).$$

Now, we'd choose 1 for the data point at x if:

$$R_1(x) < R_0(x). \implies P(1|x) > t = \frac{r}{1+r}$$

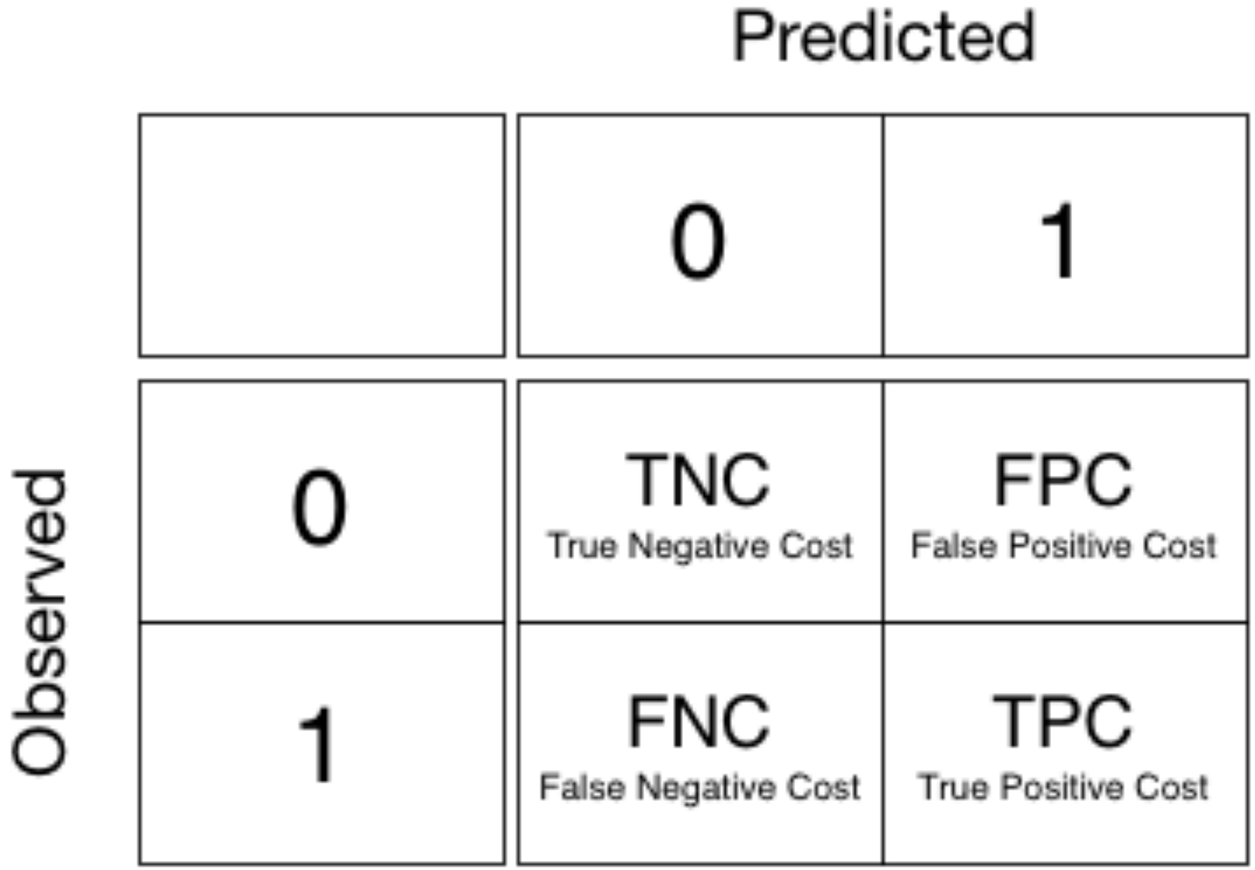
.

This is a decision Risk

One can use the prediction cost matrix corresponding to the consusion matrix

$$r == \frac{c_{FP} - c_{TN}}{c_{FN} - c_{TP}}$$

If you assume that True positives and True negatives have no cost, and the cost of a false positive is equal to that of a false positive, then $r = 1$ and the threshold is the usual intuitive $t = 0.5$.



Log score: probabilistic prediction (estimation risk)

Here we want to find a distribution a.

The utility is defined as: $u(a, y^*) = log\, a(y^*)$,

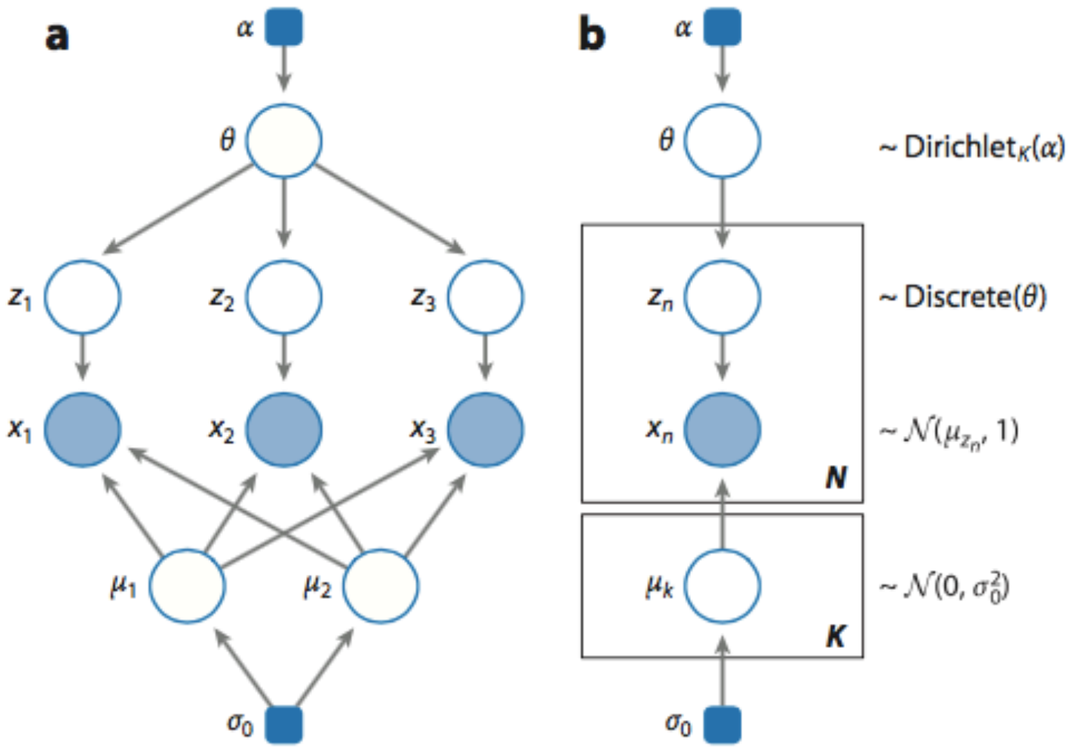
The expected utility then is $\bar{u}(a) = \int dy^*\, log(a(y^*))\, p(y^*|D, M)$.

The a that maximizes this utility is the posterior-predictive itself! $\hat{a}(y^*) = p(y^*|D, M)$

The maximized utility then is: $\bar{u}(a) = \int dy^*\, log(p(y^*|D, M))\, p(y^*|D, M)$.

MIXTURE MODELS
AND THE TYPES OF learning

Mixture Models



A distribution $p(x|\{\theta_k\})$ is a mixture of K component distributions $p_1, p_2, \ldots p_K$ if:

$$p(x|\{\theta_k\}) = \sum_k \lambda_k p_k(x|\theta_k)$$

with the λ_k being mixing weights, $\lambda_k > 0$, $\sum_k \lambda_k = 1$.

Example: Zero Inflated Poisson

Generative Model: How to simulate from it?

$$Z \sim Categorical(\lambda_1, \lambda_2, \ldots, \lambda_K)$$

where Z says which component X is drawn from.

Thus λ_j is the probability that the hidden class variable $z = j$.

Then: $X \sim p_z(x|\theta_z)$ and general structure is:

$$p(x|\theta) = \sum_z p(x, z) = \sum_z p(z)p(x|z, \theta) \text{ where } \theta = \{\theta_k\}.$$

GMM supervised formulation

$$Z \sim \text{Bernoulli}(\lambda)$$
$$X|z=0 \sim \mathcal{N}(\mu_0, \Sigma_0),\ X|z=1 \sim \mathcal{N}(\mu_1, \Sigma_1)$$

Full-data loglike: $l(x, z|\lambda, \mu_0, \mu_1, \Sigma) = -\sum_{i=1}^{m} \log((2\pi)^{n/2}|\Sigma|^{1/2})$

$$-\frac{1}{2}\sum_{i=1}^{m}\sum_{i=1}^{m}(x-\mu_{z_i})^T \Sigma^{-1}(x-\mu_{z_i}) + \sum_{i=1}^{m}[z_i \log\lambda + (1-z_i)\log(1-\lambda)]$$

Concrete Formulation of unsupervised learning

Estimate Parameters by $\mathbf{x}$-MLE:

$$
\begin{aligned}
l(x|\lambda, \mu, \Sigma) &= \sum_{i=1}^{m} \log p(x_i|\lambda, \mu, \Sigma) \\
&= \sum_{i=1}^{m} \log \sum_{z} p(x_i|z_i, \mu, \Sigma)\, p(z_i|\lambda)
\end{aligned}
$$

Not Solvable analytically! EM and Variational. Or do MCMC.

EM ALGORITHM

AND

VARIATIONAL INFERENCE

The EM algorithm

- iterative method for maximizing difficult likelihood (or posterior) problems, first introduced by Dempster, Laird, and Rubin in 1977
- Sorta like, just assign points to clusters to start with and iterate.
- Then, at each iteration, replace the augmented data by its conditional expectation (more precisely, compute the conditional expectation of the augmented or full data likelihood) given current observed data and parameter estimates. (E-step)
- Maximize the full-data likelihood (M-step).

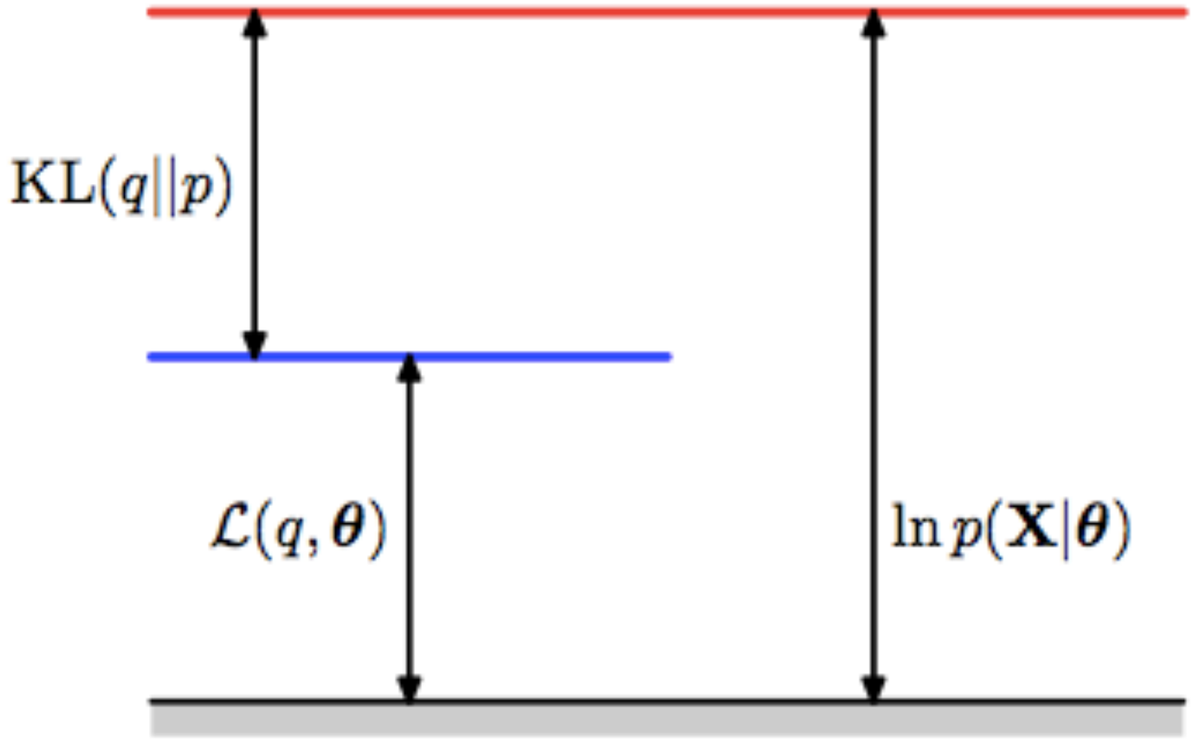
x-data likelihood

$$log\, p(x|\theta) = E_q[log\frac{p(x,z|\theta)}{q}] + D_{KL}(q,p)$$

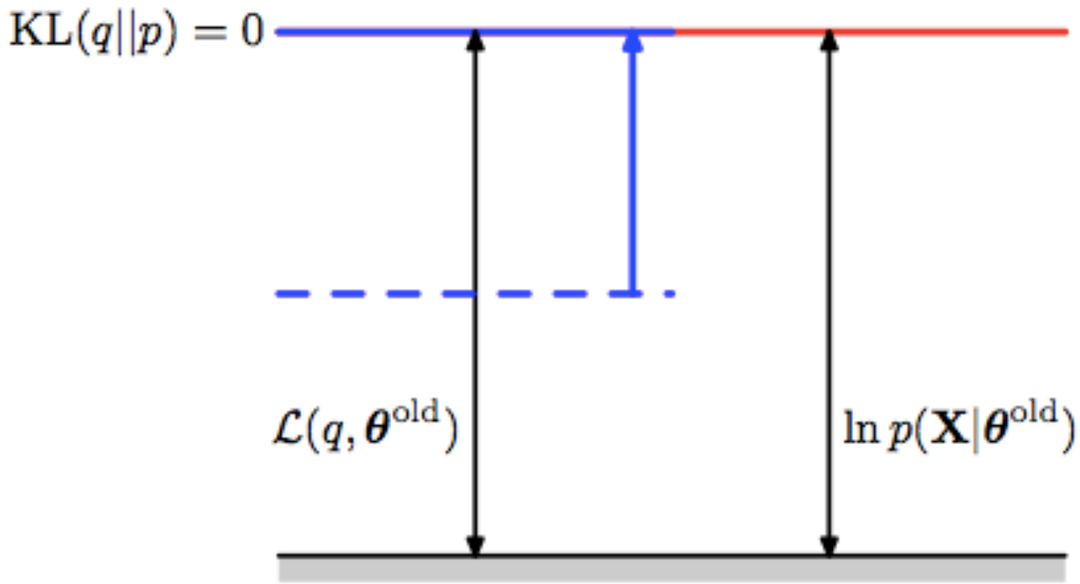
If we define the ELBO or Evidence Lower bound as:

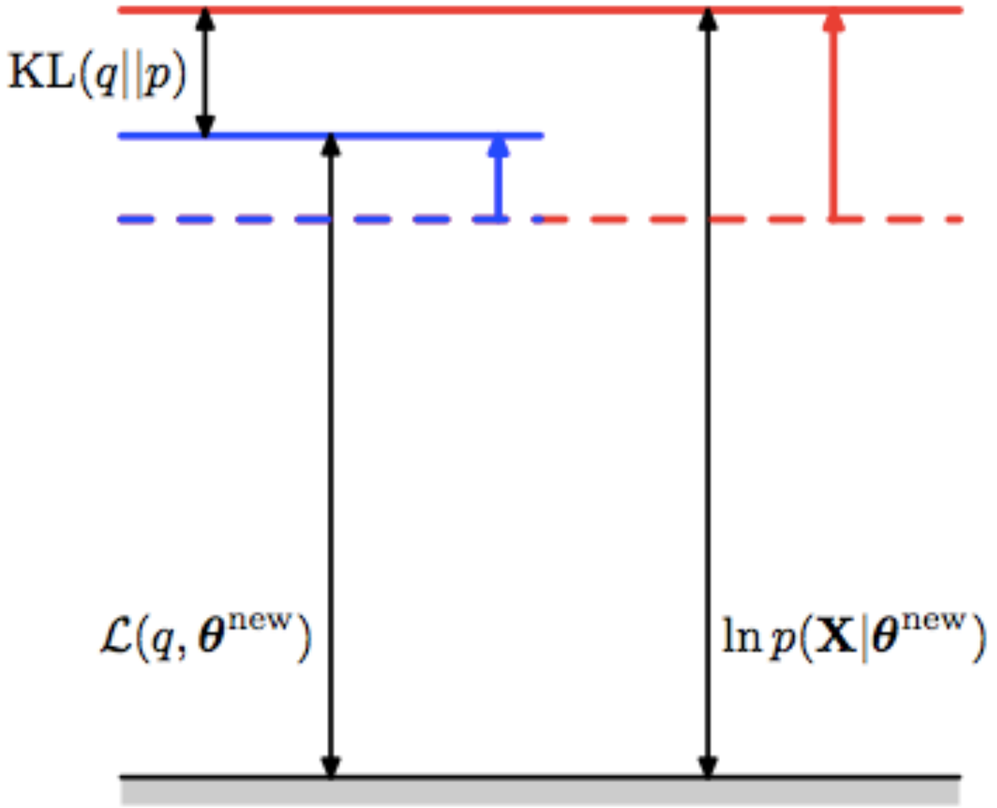
$$\mathcal{L}(q,\theta) = E_q[log\frac{p(x,z|\theta)}{q}]$$

then $log\, p(x|\theta)$ = ELBO + KL-divergence



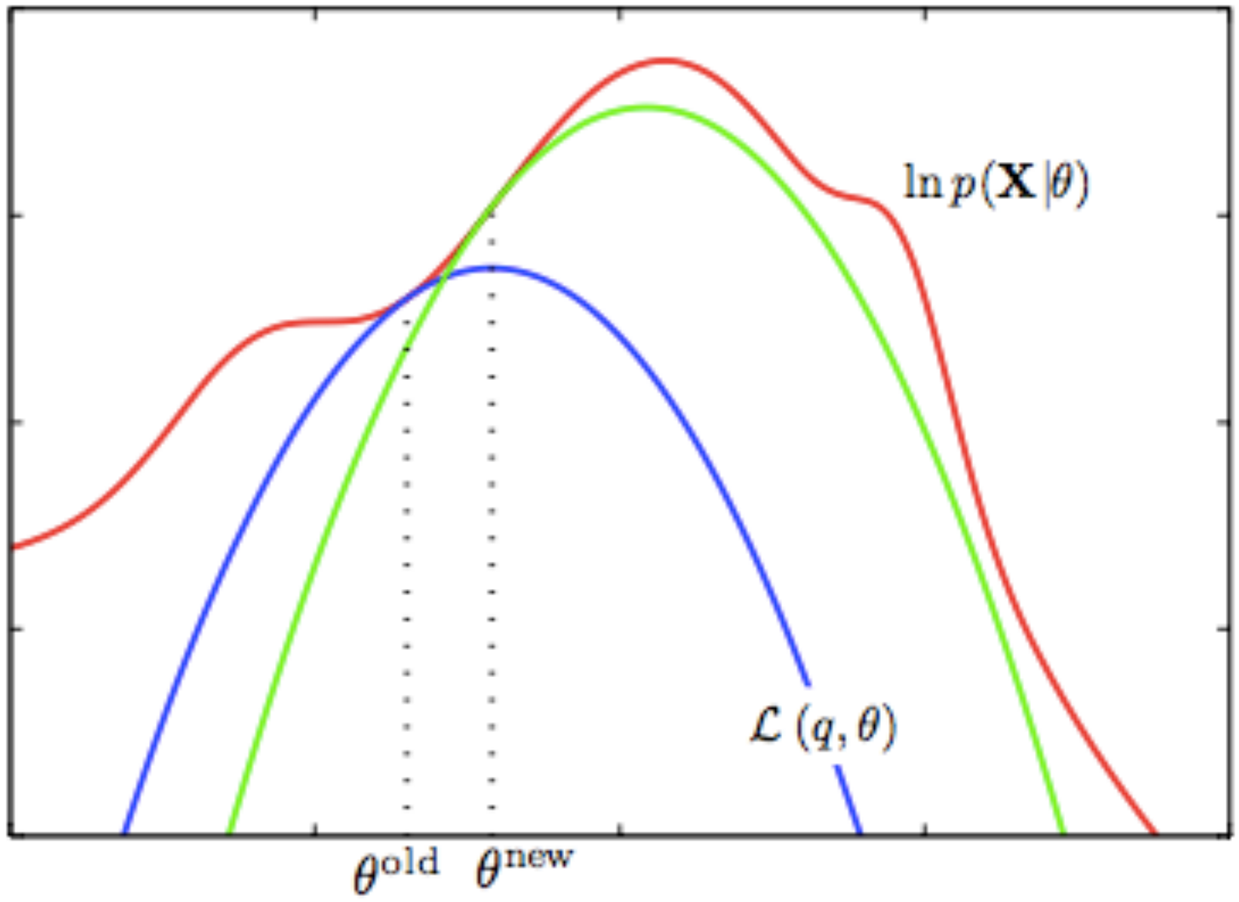
E-step and M-step





Process

1. Start with $p(x|\theta)$(red curve), θ_{old}.
2. Until convergence:
    1. E-step: Evaluate $q(z, \theta_{old}) = p(z|x, \theta_{old})$ which gives rise to ELBO(θ): $\mathcal{L}(q(z, \theta_{old}), \theta)$(blue curve) whose value equals the value of $p(x|\theta)$ at θ_{old}.
    2. M-step: maximize ELBO (or Q func) wrt θ to get θ_{new}.
    3. Set $\theta_{old} = \theta_{new}$

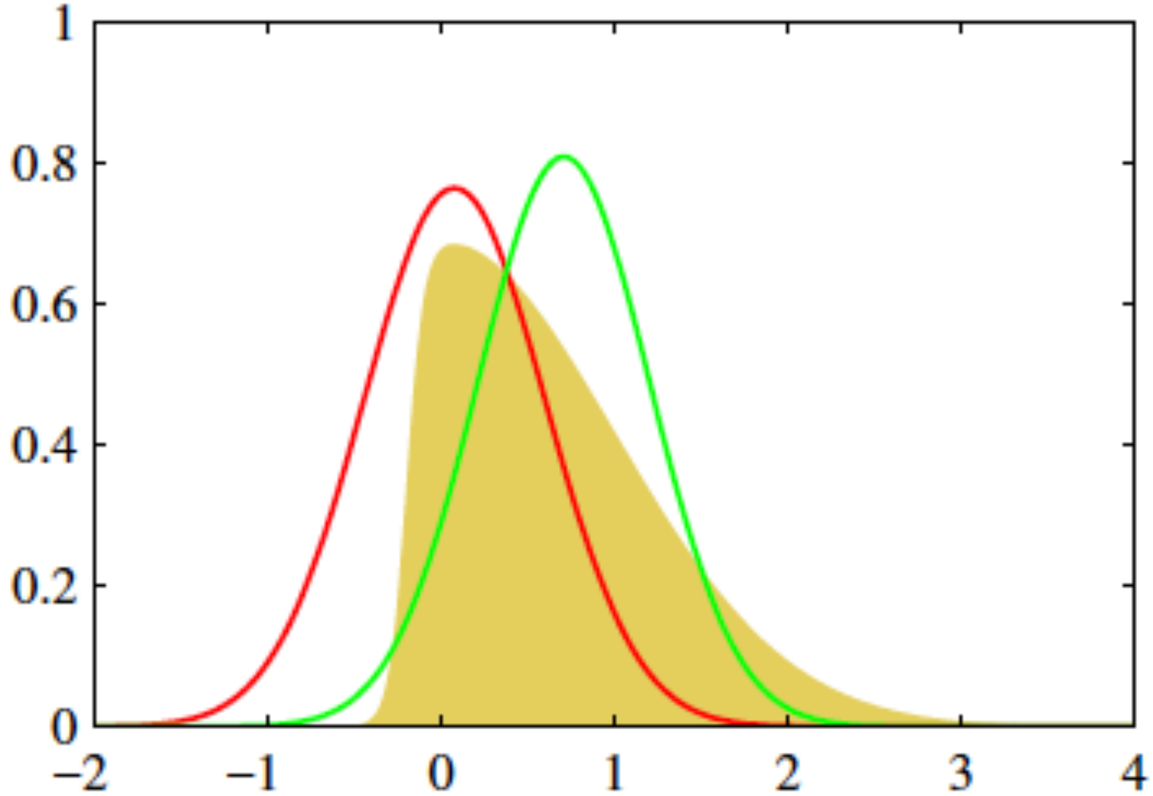


Variational Inference Core Idea

z is now all parameters. Dont distinguish from θ.

Restricting to a family of approximate distributions D over z, find a member of that family that minimizes the KL divergence to the exact posterior. An optimization problem:

$$q^*(z) = \underset{q(z) \in D}{\arg\min} \quad KL(q(z)||p(z|x))$$



Basic Setup in VI

$KL + ELBO = log(p(x))$: ELBO bounds log(evidence)

$$ELBO(q) = E_q[log\,\frac{p(z,x)}{q(z)}] = E_q[log\,\frac{p(x|z)p(z)}{q(z)}] = E_q[log\,p(x|z)] + E_q[log\,\frac{p(z)}{q(z)}]$$

$$\implies ELBO(q) = E_{q(z)}[(log(p(x|z))] - KL(q(z)||p(z))$$

(likelihood-prior balance)

Mean Field: Find a q such that:

$KL + ELBO = log(p(x))$: KL minimized means ELBO maximized.

Choose a "mean-field" q such that:

$$q(z) = \prod_{j=1}^{m} q_j(z_j)$$

Each individual latent factor can take on any paramteric form corresponding to the latent variable.

ADVI

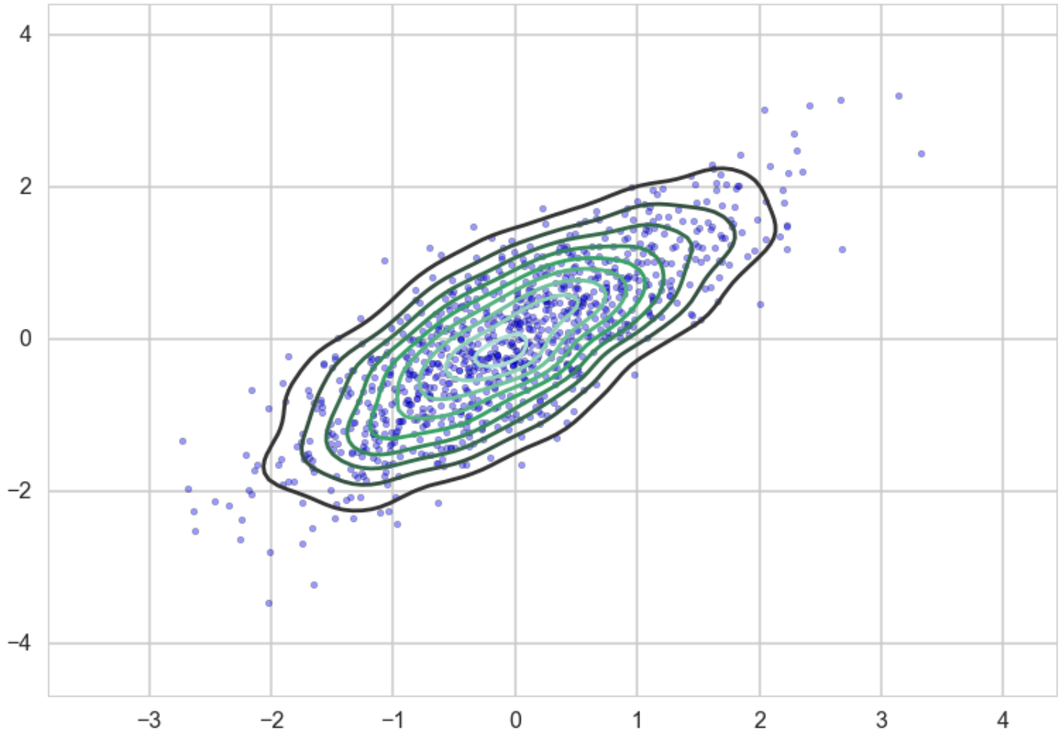
Core Idea:

- CAVI does not scale
- Use gradient based optimization, do it on less data
- do it automatically

What does ADVI do?

1. Transformation of latent parameters
2. Standardization transform for posterior to push gradient inside expectation
3. Monte-Carlo estimate of expectation
4. Hill-climb using automatic differentiation

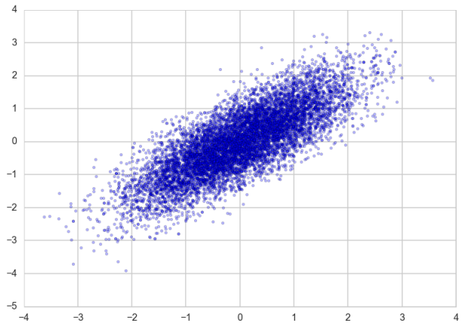
2D gaussian example



High correlation gaussian with sampler

```
cov=np.array([[0,0.8],[0.8,0]], dtype=np.float64)
data = np.random.multivariate_normal([0,0], cov, size=1000)
sns.kdeplot(data);
with pm.Model() as mdensity:
    density = pm.MvNormal('density', mu=[0,0],
    cov=tt.fill_diagonal(cov,1), shape=2)
with mdensity:
    mdtrace=pm.sample(10000)
```

Trace:



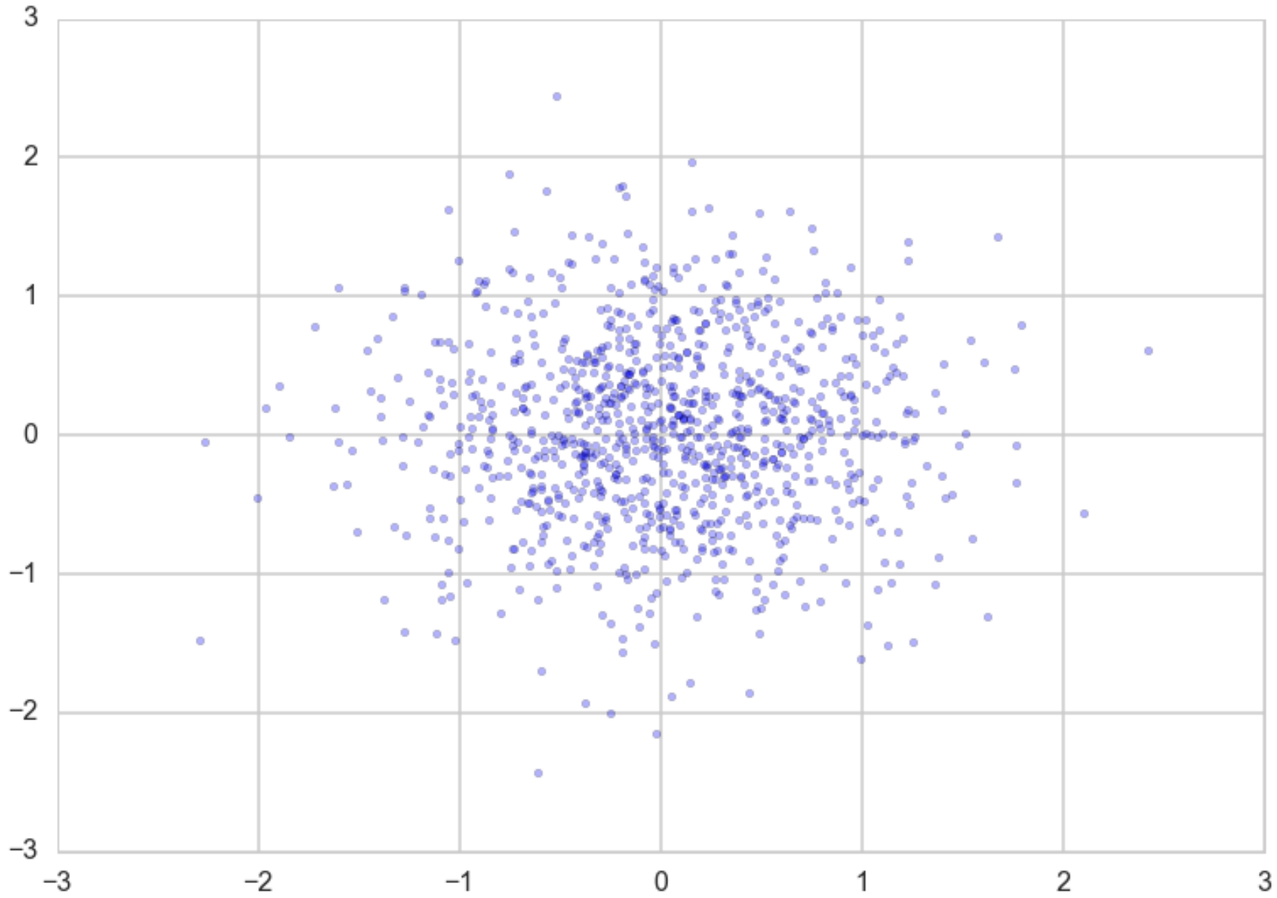
Sampling with ADVI

```
mdvar = pm.variational.advi(model=mdensity, n=100000)
samps=pm.variational.sample_vp(mdvar,model=mdensity)
plt.scatter(samps['density'][:,0],
    samps['density'][:,1], alpha=0.3)
```

ADVI cannot find the correlational structure.

Transform to de-correlate to use ADVI.

You have been doing this for NUTS anyways.



BAYESIAN NONPARAMETRICS
CORRELATION MODELING
GAUSSIAN PROCESSES

Correlation Modeling

```
import theano.tensor as tt
def pm_make_cov(sigpriors, corr_coeffs, ndim):
    sigma_matrix = tt.nlinalg.diag(sigpriors)
    n_elem = int(ndim * (ndim - 1) / 2)
    tri_index = np.zeros([ndim, ndim], dtype=int)
    tri_index[np.triu_indices(ndim, k=1)] = np.arange(n_elem)
    tri_index[np.triu_indices(ndim, k=1)[::-1]] = np.arange(n_elem)
    corr_matrix = corr_coeffs[tri_index]
    corr_matrix = tt.fill_diagonal(corr_matrix, 1)
    return tt.nlinalg.matrix_dot(sigma_matrix, corr_matrix, sigma_matrix)
```

```
sigs=np.array([1,1])
```

```
tri_index = np.zeros([2, 2], dtype=int)
tri_index
```

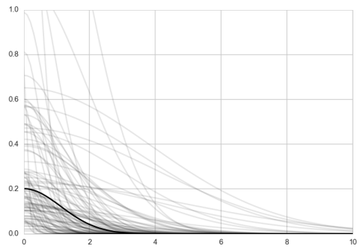
```
array([[0, 0],
       [0, 0]])
```

```
with pm.Model() as modelmvg:
    nu = pm.Uniform('nu', 1, 5)  # prior on how much correlation (0 = uniform pr
    ndim=2
    corr_coeffs = pm.LKJCorr('corr_coeffs', nu, ndim)
    cov = pm_make_cov(sigs, corr_coeffs)
    mvg = pm.MvNormal('mvg', mu=[0,0], cov=cov, shape=2, observed=data)
```

Oceanic Tools Correlations: Example of GP

We modeled society specific intercepts for oceanic tools as draws from a 0 mean multivariate gaussian and correlation function depending on distance: nearer societies have similar intercepts.

Covariance posteriors:



$$
\begin{aligned}
T_i &\sim \text{Poisson}(\lambda_i) \\
\log \lambda_i &= \alpha + \gamma_{\text{SOCIETY}[i]} + \beta_P \log P_i \\
\gamma &\sim \text{MVNormal}\big((0, \ldots, 0), \mathbf{K}\big) \\
\mathbf{K}_{ij} &= \eta^2 \exp(-\rho^2 D_{ij}^2) + \delta_{ij}(0.01) \\
\alpha &\sim \text{Normal}(0, 10) \\
\beta_P &\sim \text{Normal}(0, 1) \\
\eta^2 &\sim \text{HalfCauchy}(0, 1) \\
\rho^2 &\sim \text{HalfCauchy}(0, 1)
\end{aligned}
$$

Gausian Formulae

$$JOINT: \; p(y, f^*) = \mathcal{N}\left(\begin{bmatrix}\mu_y \\ \mu_{f^*}\end{bmatrix}, \begin{bmatrix}\Sigma_{yy} & \Sigma_{yf^*} \\ \Sigma_{yf^*}^T & \Sigma_{f^*f^*}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\mu \\ \mu_*\end{bmatrix}, \begin{bmatrix}K + \sigma^2 I & K_* \\ K_*^T & K_{**}\end{bmatrix}\right)$$

$$MARGINAL: \; p(f^*) = \int p(f^*, y)dy = \mathcal{N}(\mu_*, K_{**})$$

$$CONDITIONAL: \; p(f^* \mid y) = \mathcal{N}\left(\mu_* + K_*(K + \sigma^2 I)^{-1}(y - \mu), \; K_{**} - K_*(K + \sigma^2 I)^{-1}K_*^T\right)$$

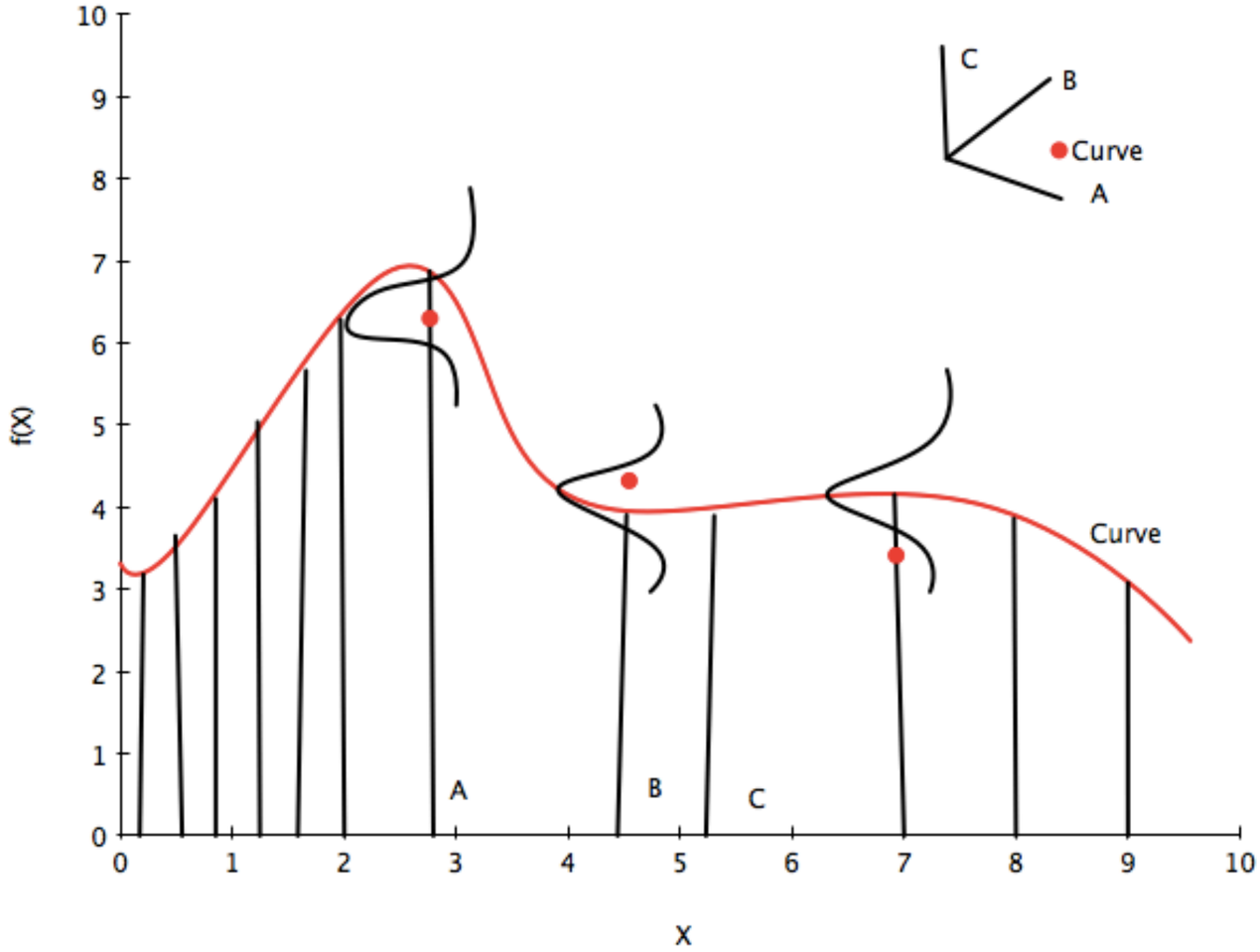
Note here that:

$$K = K(x, x); K_* = K(x, x^*); K_{**} = K(x^*, x^*)$$

Use infinite gaussians

- think of the function as an infinite vector.
- Draw $\overline{f}$ from some 'infinite' gaussian distribution with some mean and some kernel.

  This then is the Gaussian Process, which we use to set a prior on the space of functions.



Key Insight

for the marginal of a gaussian, only the covariance of the block of the matrix involving the unmarginalized dimensions matters! Thus "if you ask only for the properties of the function (you are fitting to the data) at a finite number of points, then inference in the Gaussian process will give you the same answer if you ignore the infinitely many other points, as if you would have taken them all into account!"

-Rasmunnsen

Definition of Gaussian Process

Assume we have this function vector $f = (f(x_1), \dots f(x_n))$. If, for ANY choice of input points, $(x_1, \dots, x_n)$, the marginal distribution over f:

$$P(F) = \int_{f \notin F} P(f) df$$

is multi-variate Gaussian, then the distribution $P(f)$ over the function f is said to be a Gaussian Process.

a Gaussian Process defines a prior distribution over functions!

Once we have seen some data, this prior can be converted to a posterior over functions, thus restricting the set of functions that we can use based on the data.

Since the size of the "other" block of the matrix does not matter, we can do inference from a finite set of points.

Any m observations in an arbitrary data set, $y = y_1, \ldots, y_n = m$ can always be represented as a single point sampled from some m-variate Gaussian distribution. Thus, we can work backwards to 'partner' a GP with a data set, by marginalizing over the infinitely-many variables that we are not interested in, or have not observed.

GP regression

Using a Gaussian process as a prior for our model, and a Gaussian as our data likelihood, then we can construct a Gaussian process posterior.

Likelihood: $y|f(x), x \sim \mathcal{N}(f(x), \sigma^2 I)$

where the infinite $f(x)$ takes the place of the parameters.

Prior: $f(x) \sim \mathcal{GP}(m(x) = 0, k(x, x\prime))$

Infinite normal posterior process: $f(x)|y \sim \mathcal{GP}(m_{post}, \kappa_{post}(x, x\prime))$.

The posterior distribution for f is:

$$m_{post} = k(x\prime, x)[k(x, x) + \sigma^2 I]^{-1} y$$
$$k_{post}(x, x\prime) = k(x\prime, x\prime) - k(x\prime, x)[k(x, x) + \sigma^2 I]^{-1} k(x, x\prime)$$

Posterior predictive distribution for $f(x_*)$ for a test vector input x_*, given a training set X with values y for the GP is:

$$m_* = k(x_*, X)[k(X^T, X) + \sigma^2 I]^{-1} y$$
$$k_* = k(x_*, x_*) - k(x_*, X^T)[k(X^T, X) + \sigma^2 I]^{-1} k(X^T, x_*)$$

The predictive distribution of test targets y_* : add $\sigma^2 I$ to k_*.

INFERENCE

Use the marginal likelihood:

$$p(y|X) = \int_f p(y|f,X)p(f|X)df$$

The Marginal likelihood given a GP prior and a gaussian likelihood is:

$$\log p(y|X) = -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|K+\sigma^2 I| - \frac{1}{2}y^T(K+\sigma^2 I)^{-1}y$$

Fitting in pymc3

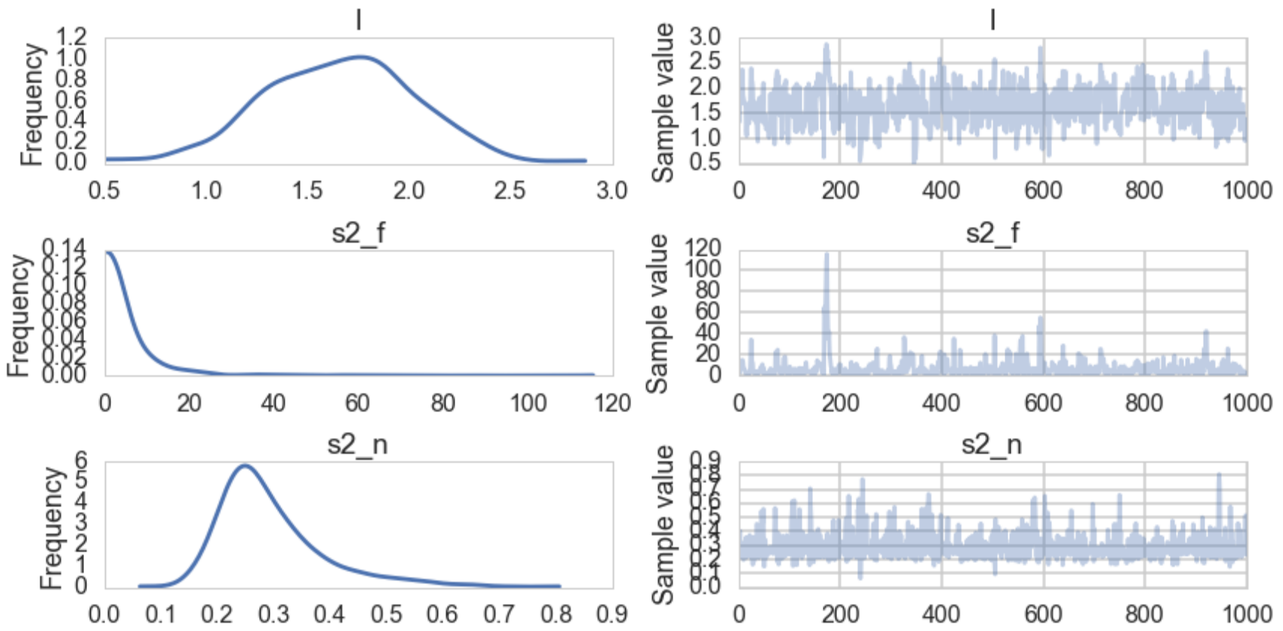
```
import theano.tensor as tt
with pm.Model() as model:
    # priors on the covariance function hyperparameters
    l = pm.Uniform('l', 0, 10)

    # uninformative prior on the function variance
    log_s2_f = pm.Uniform('log_s2_f', lower=-10, upper=5)
    s2_f = pm.Deterministic('s2_f', tt.exp(log_s2_f))

    # uninformative prior on the noise variance
    log_s2_n = pm.Uniform('log_s2_n', lower=-10, upper=3)
    s2_n = pm.Deterministic('s2_n', tt.exp(log_s2_n))

    # covariance functions for the function f and the noise
    f_cov = s2_f * pm.gp.cov.ExpQuad(1, l)

    y_obs = pm.gp.GP('y_obs', cov_func=f_cov, sigma=s2_n,
        observed={'X':xtrain.reshape(-1,1), 'Y':ytrain})
with model:
    trace = pm.sample(2000)
with model:
    gp_samples = pm.gp.sample_gp(trace,
        y_obs, x_pred.reshape(-1,1), samples=100)
```



Posterior (predictive) curves