

Computational Statistics

Computational Cognitive Science 2011

Dan Navarro

“A somewhat random bag of tricks,
mostly Monte Carlo methods”

Computational Cognitive Science 2011

Dan Navarro

The general problem to solve

How do I generate random samples from an arbitrarily chosen probability distribution $z \sim P(z)$, especially when my knowledge of $P(z)$ itself may be limited?

[illegible]

$$P(h|x) = \frac{P(x|h)P(h)}{\sum_{h' \in \mathcal{H}} P(x|h')P(h')}$$

Sampling from $P(h \mid x)$ might be a very nice thing to be able to do!

Overview

- The conjugacy trick
 - Something that we can use to avoid having to resort to computational methods
 - Also something that is a building block for a lot of models that get used in real life
- Importance sampling
 - A very simple Monte Carlo method that is rarely useful, but provides the foundations for things that actually are useful

Overview

- Markov chain Monte Carlo
 - A clever trick that solves a lot of problems
 - Detailed discussion of Metropolis-Hastings
 - Quick comments: Gibbs sampling & simulated annealing
- Particle filtering
 - A digression on likelihood weighting.
 - How to make importance sampling actually useful for problems that have a sequential structure.

Notation

- Lecture goes back and forth between:
 - Sampling from some generic distribution
 - Sampling from the posterior distribution
- The ideas are quite general, but I'll distinguish between the two cases notationally:
 - Generic distribution: $P(z)$
 - Posterior distribution: $P(h | x)$
- I've gotten sloppy about “ p versus P ” notation

WARNING: Computational statistics can be hard.

It's worth the effort though. The ideas in these lectures (a) are critical for later parts of the course, and (b) formed the foundation for nearly every major advance in Bayesian statistics and a solid chunk of probabilistic machine learning in the last 20 years.

THE CONJUGACY TRICK

Actually, this first part has nothing to do with our “sample from $P(z)$ ” problem, but it’s so very common in statistics, cognitive science and machine learning that I really ought to talk about it a bit...

I hate coding

- As a rule... if I can find any trick to avoid or minimise the amount of time I spend coding up models, I'll use it.
- So what I want to know is... when can I avoid having to write lots of code to solve my problems?

Small hypothesis spaces

- “There’s a fireworks display tonight.”
- What day of the year is it?”



Small hypothesis spaces

- Solution is simple...
 - Only 366 possible hypotheses.
 - $P(h)$ is nearly uniform (except Feb 29th)
 - $P(x | h)$ is mostly low (except Jan 1st, Jan 26th etc)
 - So $P(h | x)$ involves a sum over 366 simple things.

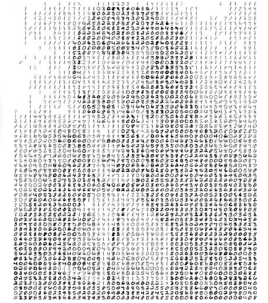


Analytic solutions

- Email prioritisation. I have observed...
 - $n = 100$ emails with the [allstaff] tag.
 - $k = 0$ of them are interesting.
- What is the probability that the next email that has the [allstaff] tag turns out to be interesting?
 - Can we solve this without coding?



Bayes



- θ is the probability that [allstaff] is useful...

$$p(\theta|k) = \frac{P(k|\theta) p(\theta)}{\int_0^1 P(k|\theta') p(\theta') d\theta'}$$

- The **posterior** $p(\theta|k)$ is a density over θ
- Need the **likelihood** $P(k|\theta)$ and the **prior** $p(\theta)$.

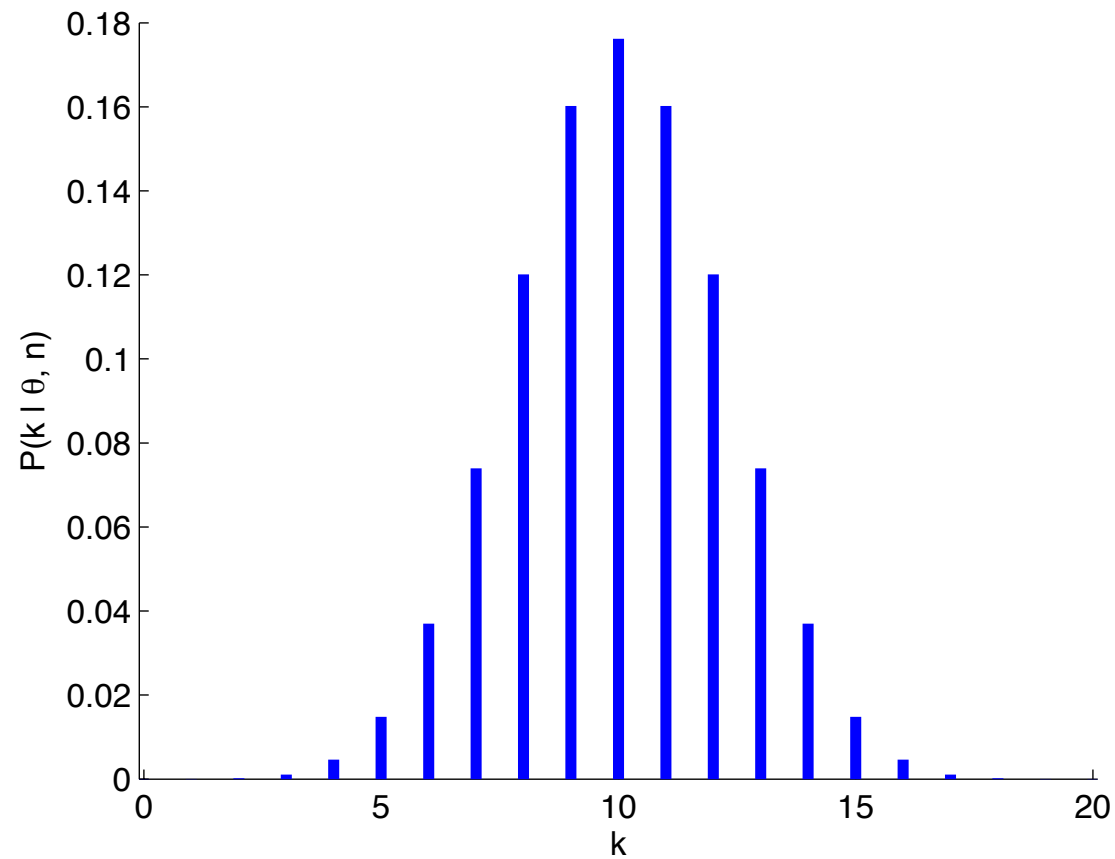
Likelihood is binomial

- Binomials:
 - The probability we see exactly k “successes” from n independent trials, where the probability of individual success is θ
 - The formula:

$$P(k|\theta, n) = \frac{n!}{k!(n-k)!} \theta^k (1 - \theta)^{n-k}$$

Likelihood is binomial

$$\theta = 0.5, n = 20$$



Prior is beta

- Beta distribution:
 - Intuitive introduction to the beta distribution is the same as Laplace smoothing...
 - Let's pretend that we've seen β_1 “prior successes”, and β_2 “prior failures”

$$p(\theta) \propto \theta^{\beta_1-1} (1 - \theta)^{\beta_2-1}$$

Prior is beta

- Beta distribution:
 - Intuitive introduction to the beta distribution is the same as Laplace smoothing...
 - Let's pretend that we've seen β_1 “prior successes”, and β_2 “prior failures”

$$p(\theta) = \frac{\theta^{\beta_1-1} (1 - \theta)^{\beta_2-1}}{\int_0^1 \theta'^{\beta_1-1} (1 - \theta')^{\beta_2-1} d\theta'}$$

Prior is beta

- Beta distribution:
 - Intuitive introduction to the beta distribution is the same as Laplace smoothing...
 - Let's pretend that we've seen β_1 “prior successes”, and β_2 “prior failures”

$$p(\theta) = \frac{\Gamma(\beta_1 + \beta_2)}{\Gamma(\beta_1)\Gamma(\beta_2)} \theta^{\beta_1-1} (1 - \theta)^{\beta_2-1}$$

(Gamma function?)

- $\Gamma(x) = (x-1)!$ for integer valued x
- $\Gamma(x) = (x-1) \Gamma(x-1)$ for all x
- $\Gamma(x)$
 - has easily computed approximations for all x (e.g., Lanczos approx – see the tech note),
 - is a built in function in lots of languages

Wait... what tech note?

An introduction to the Beta-Binomial model

COMPSCI 3016: Computational Cognitive Science
Dan Navarro & Amy Perfors
University of Adelaide

This one!

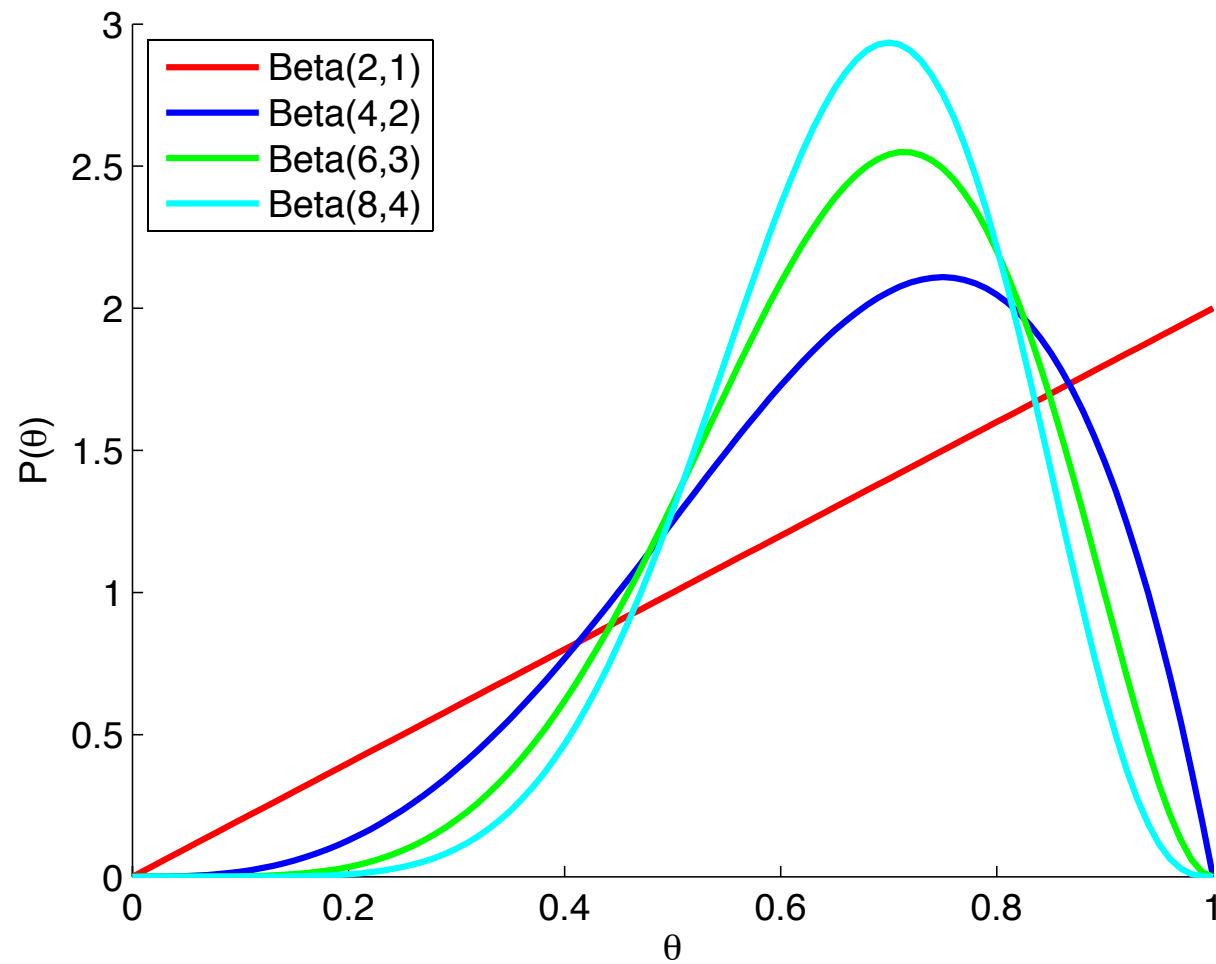
Abstract

This note provides a detailed discussion of the “beta-binomial model” described in lecture 4, and pops up in several places in the lecture series. The goal is to (a) give you a more comprehensive resource to work from, and (b) to highlight some of the interesting technical issues that will arise when you try to build models of your own. Additionally, the notes aim to be self-contained – very little knowledge of statistics will be assumed. If anything the notes doesn’t make sense please contact me (these notes were written by Dan: daniel.navarro@adelaide.edu.au) and I’ll try to fix them!

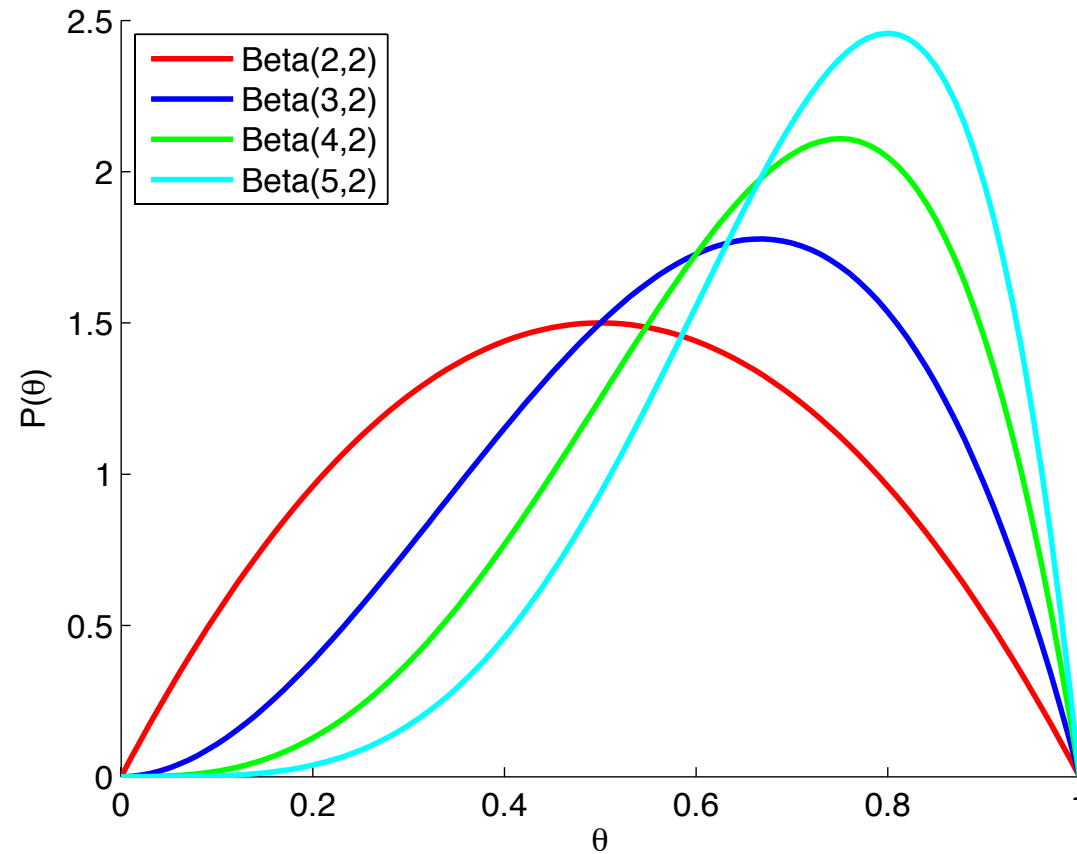
Introduction

The beta-binomial model (along with the closely-related beta-Bernoulli model) is probably the simplest interesting Bayesian model. It is tractable, useful for simple situations (e.g., the coins problem from lecture 4), and is easy to extend to more interesting examples (e.g., the AI survey problem from lectures 11-13). As such, it forms one of the basic building blocks for Bayesian modelling in cognitive science as well as other disciplines. This note describes the model in some detail. Much of this material should be very familiar to you

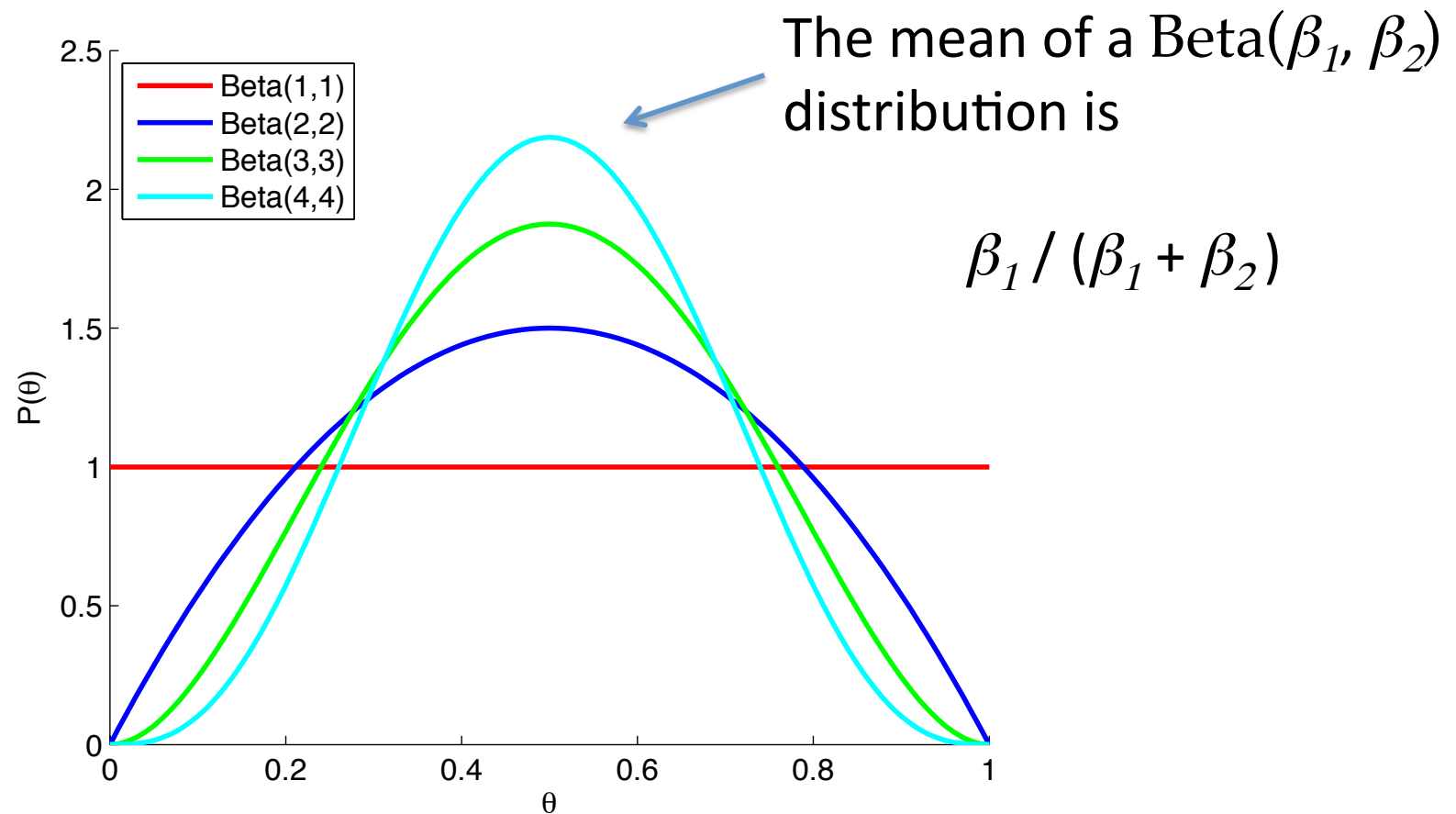
Betas are flexible, handy things



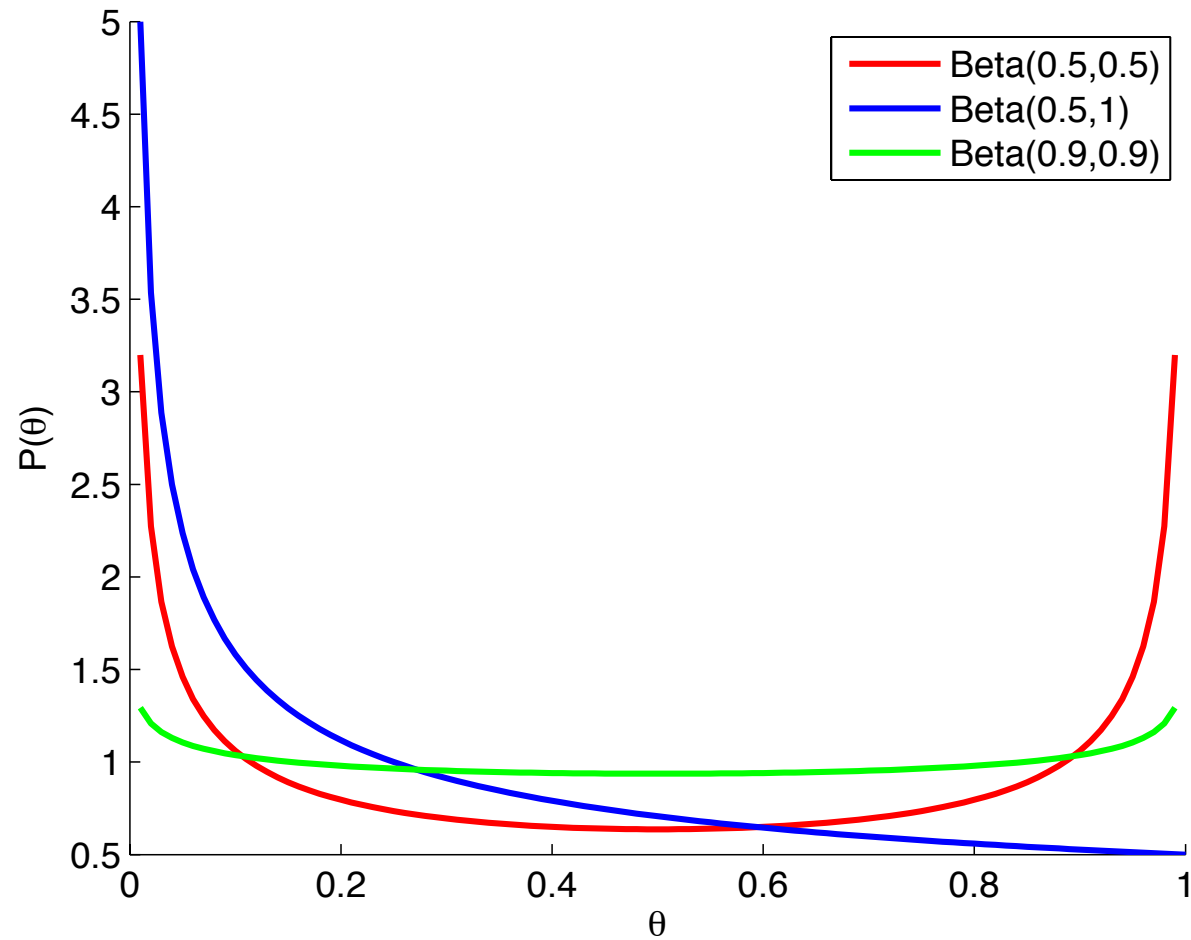
Betas are flexible, handy things



Betas are flexible, handy things



Betas are flexible, handy things



Betas and binomials go together nicely

$$p(\theta|k) = \frac{P(k|\theta)p(\theta)}{P(k)}$$

$$\propto P(k|\theta)p(\theta)$$



Let's apply Bayes' rule to find the posterior distribution over θ , assuming that the prior is Beta and the likelihood is Binomial

[illegible]

Betas and binomials go together nicely

$$p(\theta|k) = \frac{P(k|\theta)p(\theta)}{P(k)}$$

$$\propto P(k|\theta)p(\theta)$$

$$= \underbrace{\frac{n!}{k!(n-k)!} \theta^k (1-\theta)^{n-k}}_{\text{Binomial likelihood}} \times \underbrace{\frac{\Gamma(\beta_1)\Gamma(\beta_2)}{\Gamma(\beta_1 + \beta_2)} \theta^{\beta_1-1} (1-\theta)^{\beta_2-1}}_{\text{Beta prior}}$$

Binomial likelihood

Beta prior

Betas and binomials go together nicely

$$p(\theta|k) = \frac{P(k|\theta)p(\theta)}{P(k)}$$

$$\propto P(k|\theta)p(\theta)$$

$$= \left(\frac{n!}{k!(n-k)!} \theta^k (1-\theta)^{n-k} \right) \times \left(\frac{\Gamma(\beta_1)\Gamma(\beta_2)}{\Gamma(\beta_1 + \beta_2)} \theta^{\beta_1-1} (1-\theta)^{\beta_2-1} \right)$$

These terms don't depend
on θ , so they're just
constants of proportionality

Betas and binomials go together nicely

$$p(\theta|k) = \frac{P(k|\theta)p(\theta)}{P(k)}$$

$$\propto P(k|\theta)p(\theta)$$

$$= \frac{n!}{k!(n-k)!} \theta^k (1-\theta)^{n-k} \times \frac{\Gamma(\beta_1)\Gamma(\beta_2)}{\Gamma(\beta_1 + \beta_2)} \theta^{\beta_1-1} (1-\theta)^{\beta_2-1}$$

$$\propto \theta^k (1-\theta)^{n-k} \times \theta^{\beta_1-1} (1-\theta)^{\beta_2-1}$$

So we can basically ignore them... we're only interested in $p(\theta|k)$ up to a constant of proportionality anyway!

Betas and binomials go together nicely

$$p(\theta|k) = \frac{P(k|\theta)p(\theta)}{P(k)}$$

$$\propto P(k|\theta)p(\theta)$$

$$= \frac{n!}{k!(n-k)!} \theta^k (1-\theta)^{n-k} \times \frac{\Gamma(\beta_1)\Gamma(\beta_2)}{\Gamma(\beta_1 + \beta_2)} \theta^{\beta_1-1} (1-\theta)^{\beta_2-1}$$

$$\propto \theta^k (1-\theta)^{n-k} \times \theta^{\beta_1-1} (1-\theta)^{\beta_2-1}$$

$$= \theta^{k+\beta_1-1} (1-\theta)^{n-k+\beta_2-1}$$

Finally, if we collect terms, we get this... which is basically the formula for a beta distribution again, just with different parameter values.

The conjugacy property...

- If $p(\theta)$ is Beta... $\theta \sim \text{Beta}(\beta_1, \beta_2)$
and $P(k)$ is Binomial... $k \sim \text{Binomial}(\theta)$

The conjugacy property...

- If $p(\theta)$ is Beta... $\theta \sim \text{Beta}(\beta_1, \beta_2)$
and $P(k)$ is Binomial... $k \sim \text{Binomial}(\theta)$
- Then $p(\theta | k)$ is also Beta...

$$\theta | k \sim \text{Beta}(\beta_1 + k, \beta_2 + n - k)$$

Back to the email problem

- Email problem:
 - Data are binomial
 - $k = 0$ good emails
 - $n = 100$ emails total
- Priors:
 - I have a bias to think emails often useless
 - Beta prior with, say.... $\beta_1 = 1, \beta_2 = 3$



The posteriors, and my expectations about the future of my email...

- Using the conjugacy property:
 - Posterior is $\text{Beta}(\beta_1 + k, \beta_2 + n-k)$
 - i.e., $\text{Beta}(1,103)$
- Probability that new email will be good?
 - Mean of $\text{Beta}(x,y)$ distribution is $x / (x+y)$
 - $1 / 104 = .0096$

Notice that this is a similar result to Laplace smoothing... which is what you'd hope to see.

Postscript

- Why such a fuss over beta-binomial models?
 - It's basically just Laplace smoothing right?
 - Sort of: beta-binomials give you the theoretical foundations to Laplace smoothing, but they're more general.
 - The full probabilistic formalism gives you access to other measures besides the posterior mean.

Postscript

- Why such a fuss over beta-binomial models?
 - Beta-binomials (and their multivariate generalisation, Dirichlet-multinomials) are a basic building block for lots of very cool methods.
 - See, e.g., **latent Dirichlet allocation** (look it up on Google). It's a machine learning tool popular for the last 8 years or so; relies fundamentally on the conjugacy trick that I've just described.

Postscript

- The conjugacy trick is quite general...
 - beta-binomial
 - Dirichlet-multinomial
 - Poisson-gamma
 - normal-gamma
 - multivariate normal-inverse Wishart
 - Dirichlet process–i.i.d. sampling

(any likelihood function in
the “exponential family”
has a conjugate prior)

This is good spot to ask questions.

**A BILLION WAYS THAT STATISTICS
CAN BE VERY HARD.**

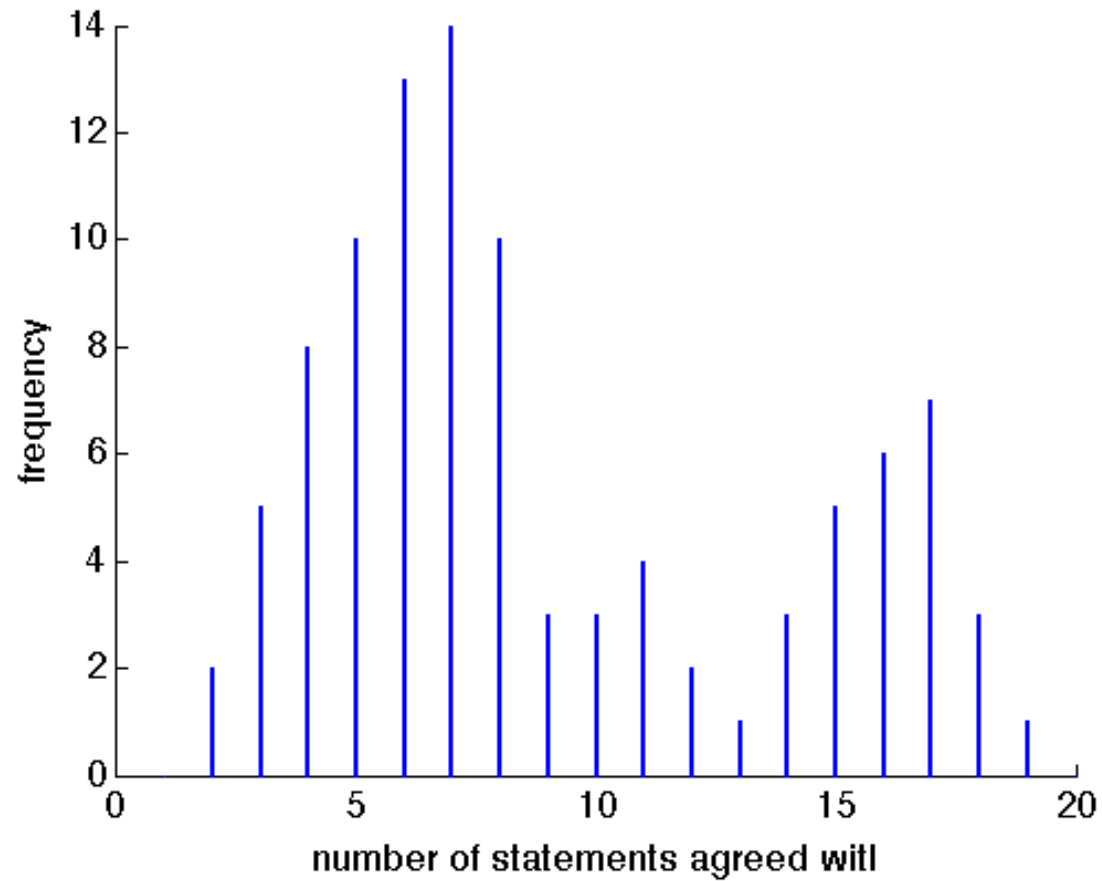
A very simple inference problem

- A survey of 100 people
 - Asks whether they agree or disagree with 20 statements about “the strong AI hypothesis” (that the mind is a Turing machine)

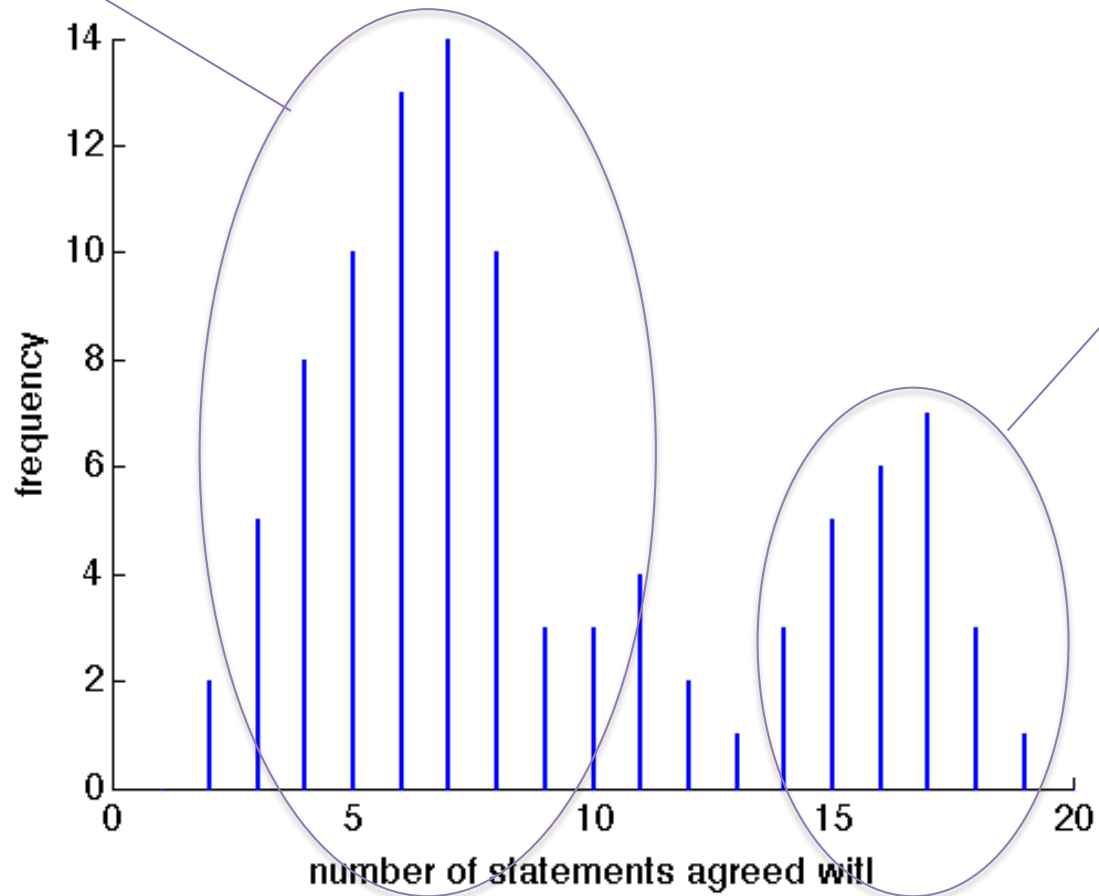
A very simple inference problem

- A survey of 100 people
 - Asks whether they agree or disagree with 20 statements about “the strong AI hypothesis” (that the mind is a Turing machine)
- The reponses:
 - Most humans are pretty similar to each other, and they don’t agree with many of your survey questions
 - But as it turns out, some of your respondents are actually robots, and they have a different opinion...

The AI survey “data”



The AI survey “data”



A model for these data

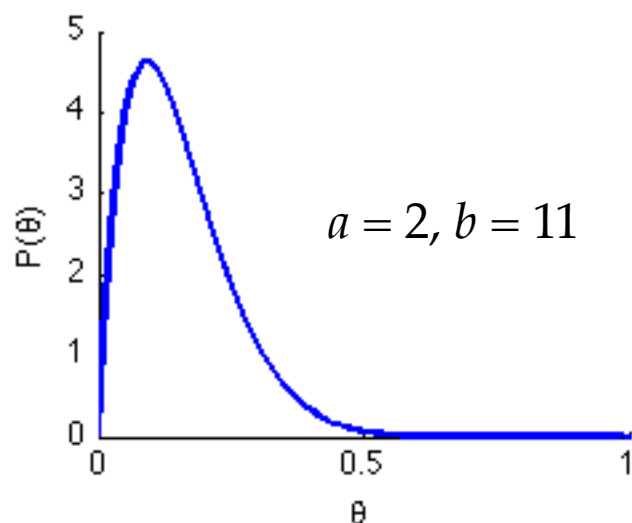
- With probability ϕ , the respondent is human
- For robots, the number of agreements is binomial with rate θ_0
- For humans, the number of agreements is binomial with rate θ_1
- Three unknown parameters ϕ , θ_0 and θ_1

Priors for this model?

- Beta distribution:

$$P(\theta|a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{\int_0^1 \theta^{a-1}(1-\theta)^{b-1} d\theta}$$

$$\propto \theta^{a-1}(1-\theta)^{b-1}$$



Use beta priors for all three parameters, θ_o , θ_1 and ϕ , but with different values of a and b for each case

Likelihoods for this model

- Mixture of two binomials:

$$P(x|\theta_0, \theta_1, \phi) = \phi P(x|\theta_1) + (1 - \phi)P(x|\theta_0)$$

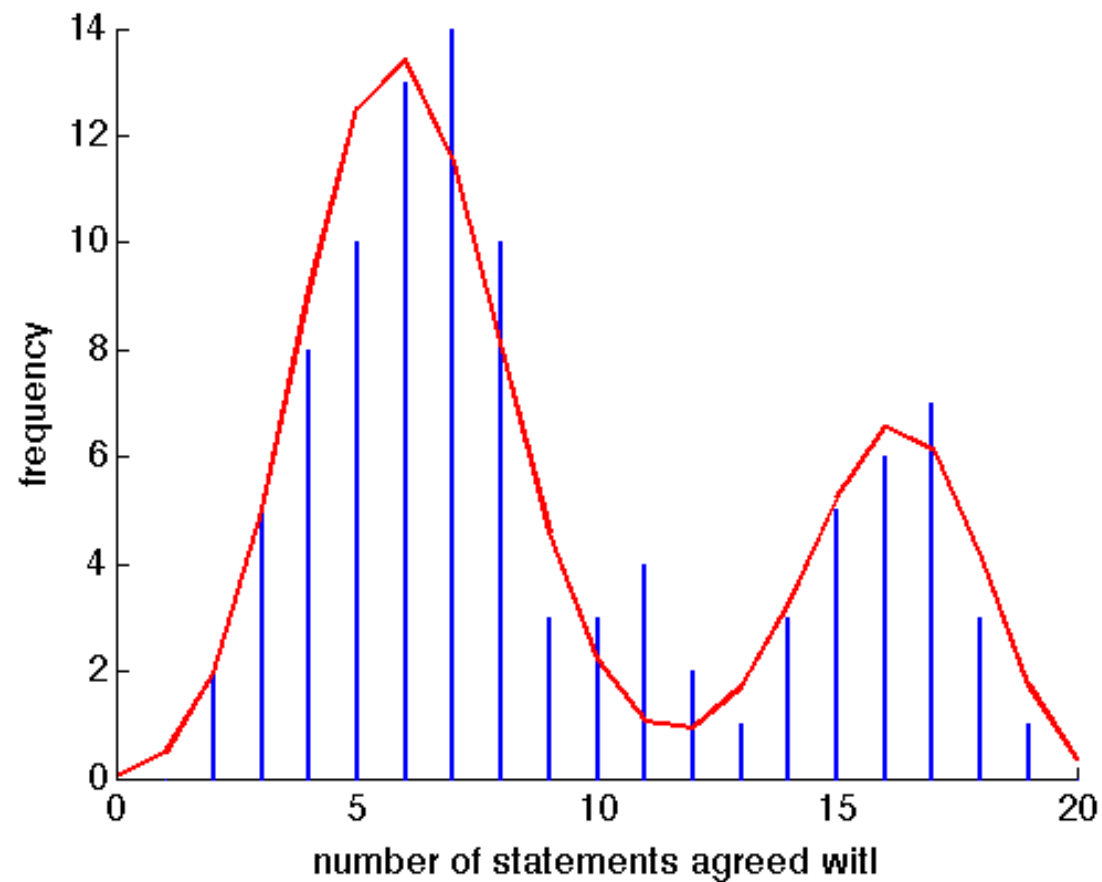
where:

$$\begin{aligned} P(x|\theta) &= \frac{N!}{(N-x)!x!} \theta^x (1-\theta)^{N-x} \\ &\propto \theta^x (1-\theta)^{N-x} \end{aligned}$$

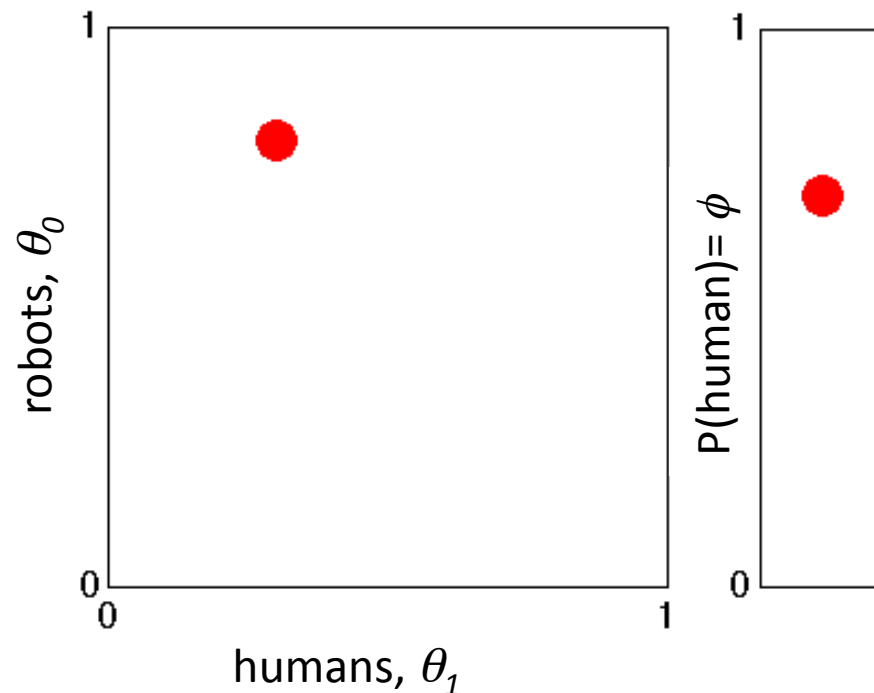
and $N = 20$

The true distribution:

- $\theta_1 = .3, \theta_0 = .8, \phi = .7$:



Here's the true answer & the hypothesis space that it belongs to...



**TWO VERY BAD IDEAS, IF YOU WANT TO
KNOW ABOUT THE POSTERIOR
DISTRIBUTION...**

Trying to solve it analytically?

- Here's the mathematical expression:

$$P(\theta_1, \theta_2, \phi | \mathbf{x}) = \frac{P(\mathbf{x} | \theta_1, \theta_2, \phi) P(\theta_1, \theta_2, \phi)}{\int_0^1 \int_0^1 \int_0^1 P(\mathbf{x} | \theta_1, \theta_2, \phi) P(\theta_1, \theta_2, \phi) d\theta_1 d\theta_2 d\phi}$$

- The triple-integral there looks nasty
 - and this problem is **much** simpler than anything you might need to solve in real life!

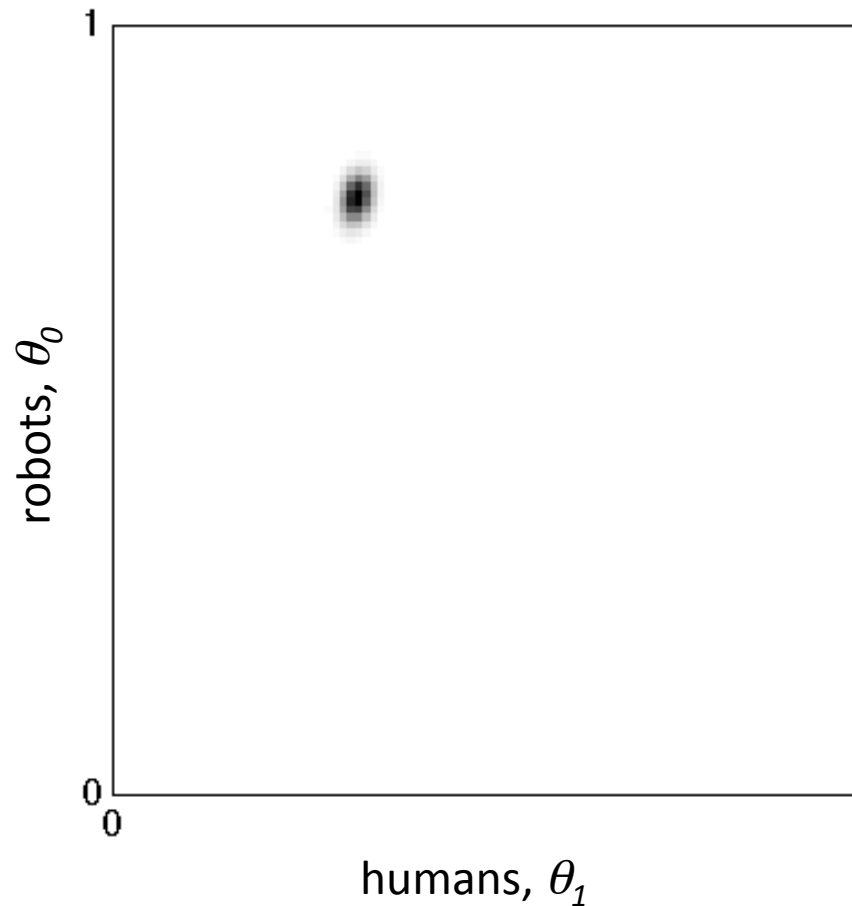
Trying to solve it by brute force?

- We could discretise the hypothesis space
 - 1000 values for θ_0 , 1000 values for θ_1 and 1000 values for ϕ means we need to sum over 10^9 hypotheses

Trying to solve it by brute force?

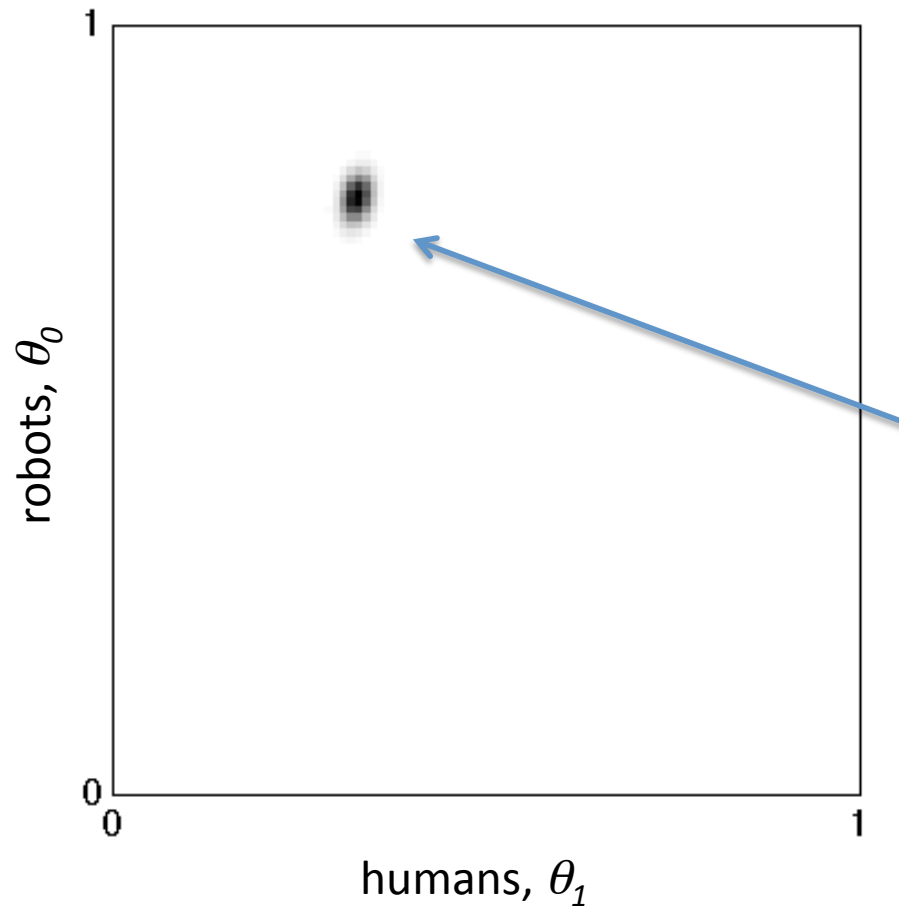
- We could discretise the hypothesis space
 - 1000 values for θ_0 , 1000 values for θ_1 and 1000 values for ϕ means we need to sum over 10^9 hypotheses
 - In general, Q values for K parameters is going to require you to compute predictions for Q^K hypotheses.
 - Interesting models can have large K , so this isn't going to scale very well!

The fundamental problem: the space is big and the distribution is sparse



This is a thin slice through a “brute force” evaluation for a 100 by 100 by 100 grid over the parameters θ_0 , θ_1 and ϕ . ($\phi = .7$ in this slice).

The fundamental problem: the space is big and the distribution is sparse

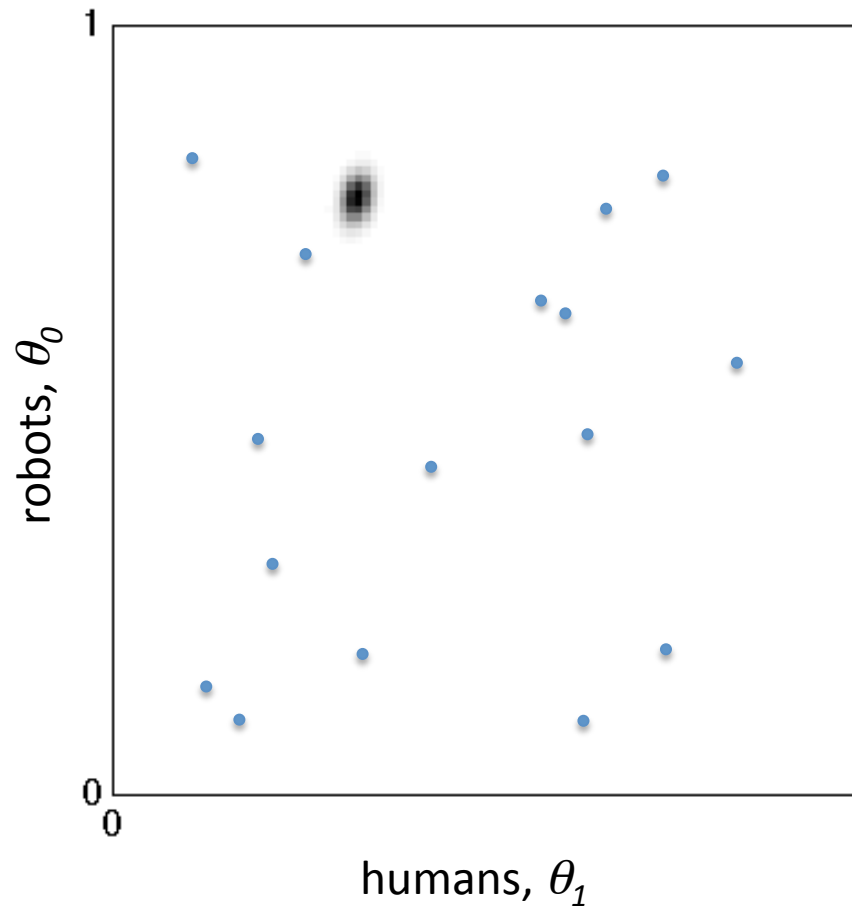


This is a thin slice through a “brute force” evaluation for a 100 by 100 by 100 grid over the parameters θ_0 , θ_1 and ϕ . ($\phi = .7$ in this slice).

The posterior is “sparse”. 2243 of the 1030301 parameter sets (a mere 0.22%) make up 95% of the posterior probability

Very wasteful simulation.

The fundamental problem: the space is big and the distribution is sparse



Sparsity means that randomly chosen parameter values aren't very likely to be useful.

We need to do something smarter than brute force evaluation (or random generation)

Interim summary

- When the number of parameters gets large, the posterior distribution
 - is usually analytically intractable
 - is defined over a high-dimensional space
 - has near-zero probability across the vast majority of that space (esp. when you have a lot of data)

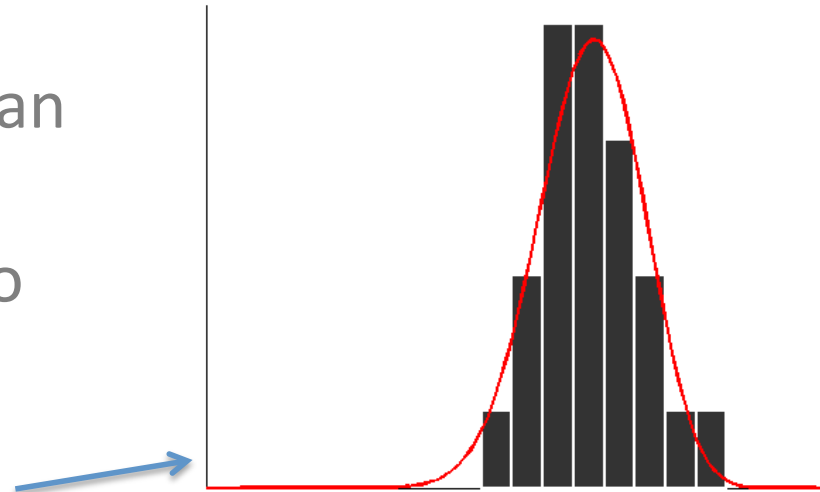
**PROPOSED SOLUTION... GENERATE
SOLUTIONS BY SAMPLING “DIRECTLY”
FROM THIS POSTERIOR**

Using samples for estimation

- If we have a set of **samples from the distribution**, we can use that set of samples to construct approximations to that distribution

Using samples for estimation

- If we have a set of **samples from the distribution**, we can use that set of samples to construct approximations to that distribution
- e.g., approximating the Beta(20,12) with 100 samples

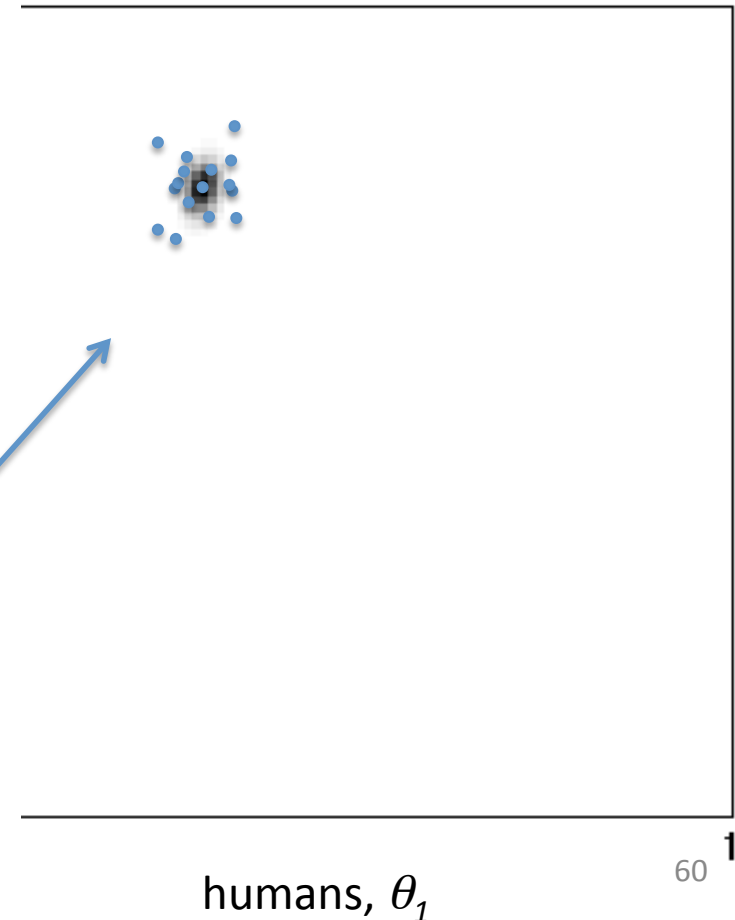


Note that what we're doing is treating the set of samples as if it were the actual distribution!

This “**Monte Carlo**” idea is a trick we are going to use a lot.

Using samples for estimation

- If we have a set of **samples from the distribution**, we can use that set of samples to construct approximations to that distribution
- e.g., approximating the Beta(20,12) with 100 samples
- Sparsity of the distribution doesn't matter as much, because the samples will be clustered in the right part of the space!



“Monte Carlo approximation” to a distribution $P(z)$

- Treat the distribution of the samples as an approximation to $P(z)$
- Formally:

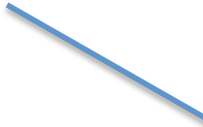
$$P(z) \approx \frac{1}{n} \sum_{i=1}^n \delta(z_i)$$

for $z_i \sim P(z)$

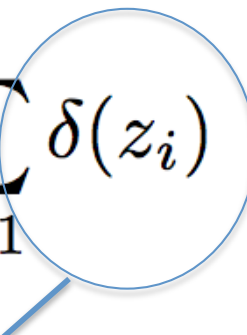
Sort of dumb to have so much notation for something so simple, but it's hand.

“Monte Carlo approximation” to a distribution $P(z)$

All it means is that we’re approximating the distribution of interest $P(z)$ by taking an average...


$$P(z) \approx \frac{1}{n} \sum_{i=1}^n \delta(z_i)$$

“Monte Carlo approximation” to a distribution $P(z)$

$$P(z) \approx \frac{1}{n} \sum_{i=1}^n \delta(z_i)$$


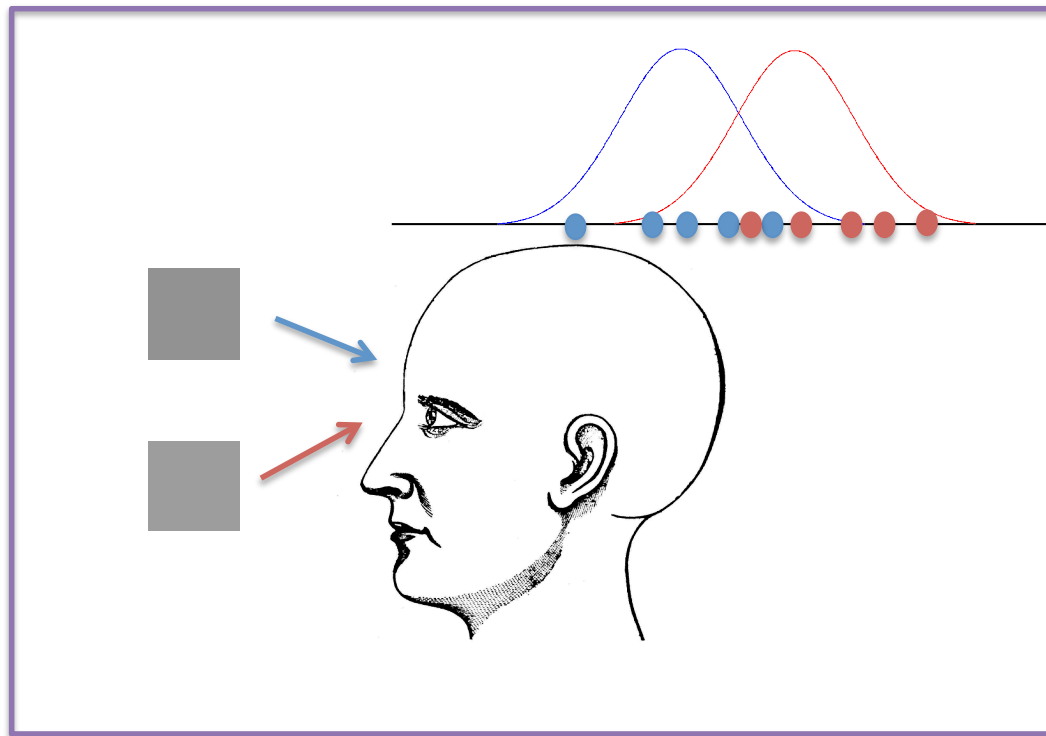
... the average of lots of “delta functions”; which assign probability 1 to the value z_i and probability 0 to all others

“Monte Carlo approximation” to a distribution $P(z)$

In short... the sample itself is being used as an approximation to the **target distribution** $P()$

$$P(z) \approx \frac{1}{n} \sum_{i=1}^n \delta(z_i)$$

The big question: where do these samples come from?



Sometimes, the world will do
the work for you!

There are problems in psychology where the world generates “samples” from a target distribution for you.

The big question: where do these samples come from?

$t \sim \text{Poisson}(\lambda)$

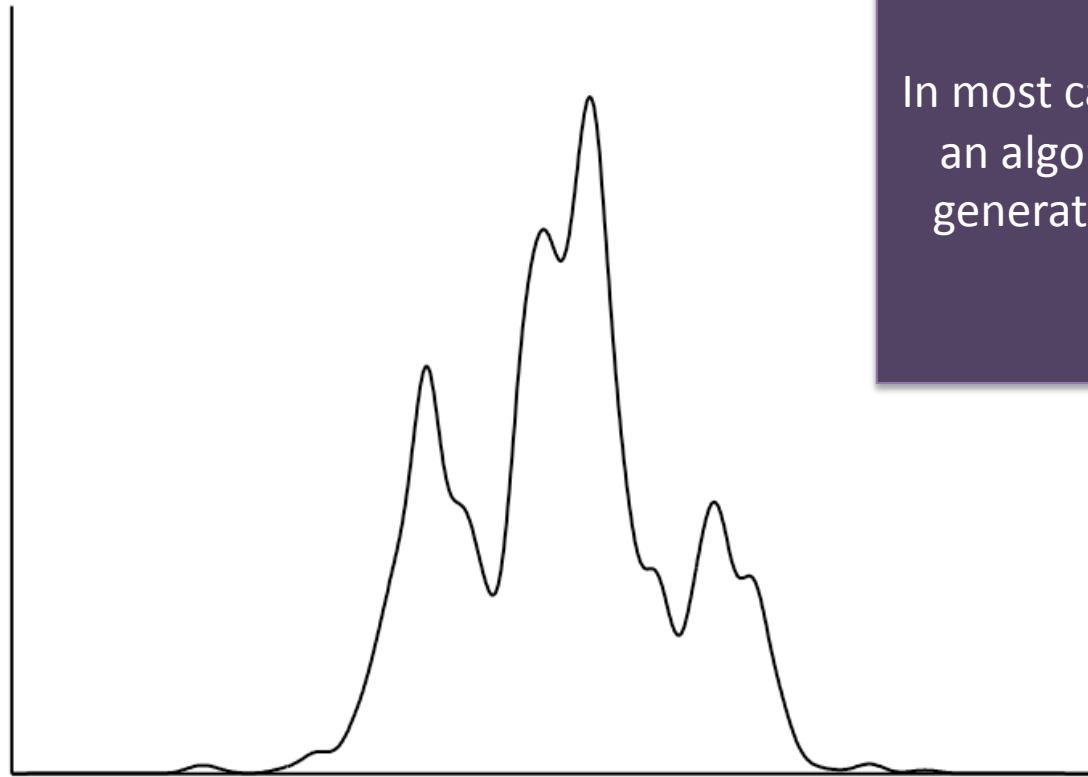
```
set  $L = \exp(-\lambda)$ ;  $t=0$ ;  $p=1$ ;  
do while  $p > L$   
     $t = t+1$ ;  
    generate  $u \sim \text{Uniform}([0,1])$   
     $p = p * u$ ;  
 $t = t-1$ ;
```

Sometimes there is a well-known algorithm for sampling from the distribution

$$P(t|\lambda) = \frac{\lambda^t \exp(-\lambda)}{t!}$$

* hint. You'll need this algorithm during the decision making lectures

The big question: where do these samples come from?



In most cases, we don't have an algorithm designed to generate samples directly from $P(z)$

Three suggestions

- Importance sampling
- Markov chain Monte Carlo (MCMC)
 - Metropolis-Hastings
 - Gibbs sampling
- Particle filtering
 - aka. Sequential importance samplers

Let's start with a method that is totally useless in real life!

IMPORTANCE SAMPLING WHEN WE CAN EVALUATE THE EQUATION FOR $P(Z)$

What do we know about $P(z)$?

- Obviously, if the answer is “nothing whatsoever”, then we’re screwed...
- Suppose we know the equation for $P(z)$.
 - So we can *evaluate* $P(z)$, but we don’t know for sure (ahead of time) which values of z correspond to big or small values of $P(z)$.

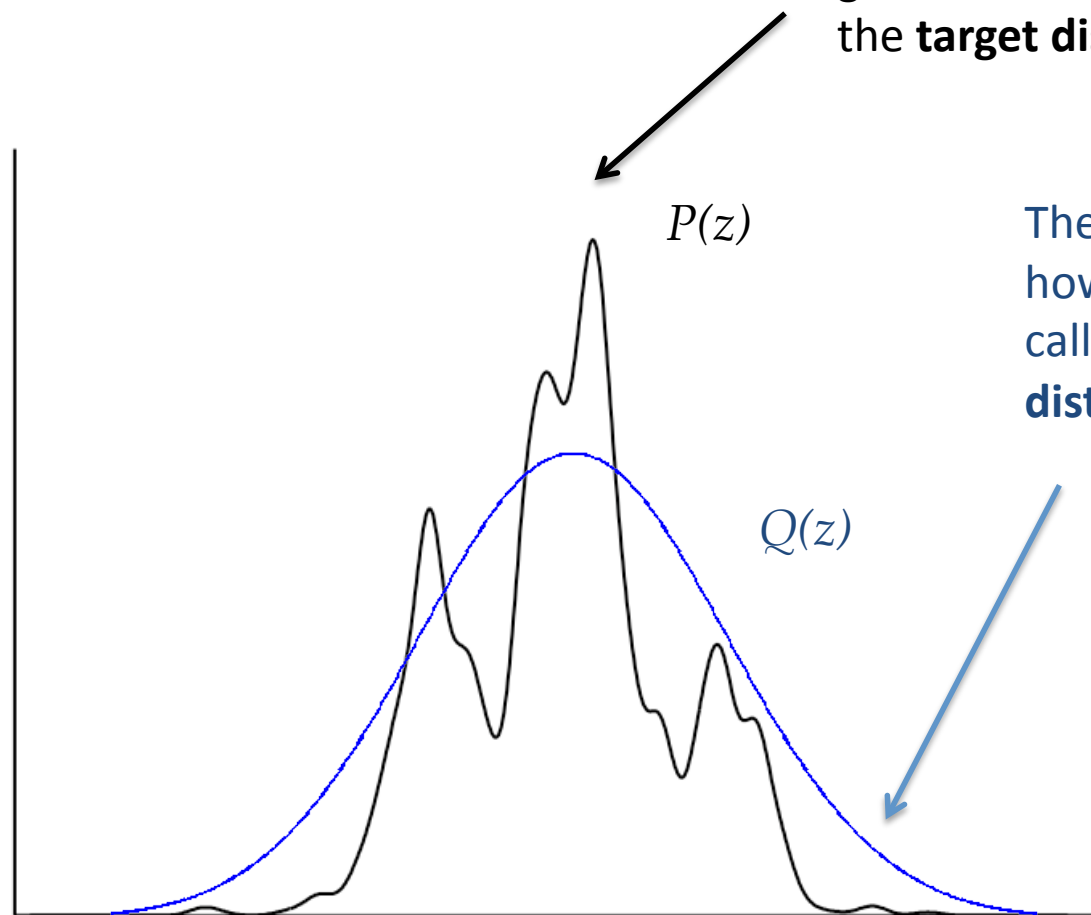
Importance sampling

- The idea...
 - I can't sample directly from $P(z)$. So I'll make up some other distribution $Q(z)$ which looks similar, but which I do know how to sample from....
- Specifically...
 - I'll sample $z \sim Q(z)$ first, and then make some "corrections" to deal with the fact that $P \neq Q$

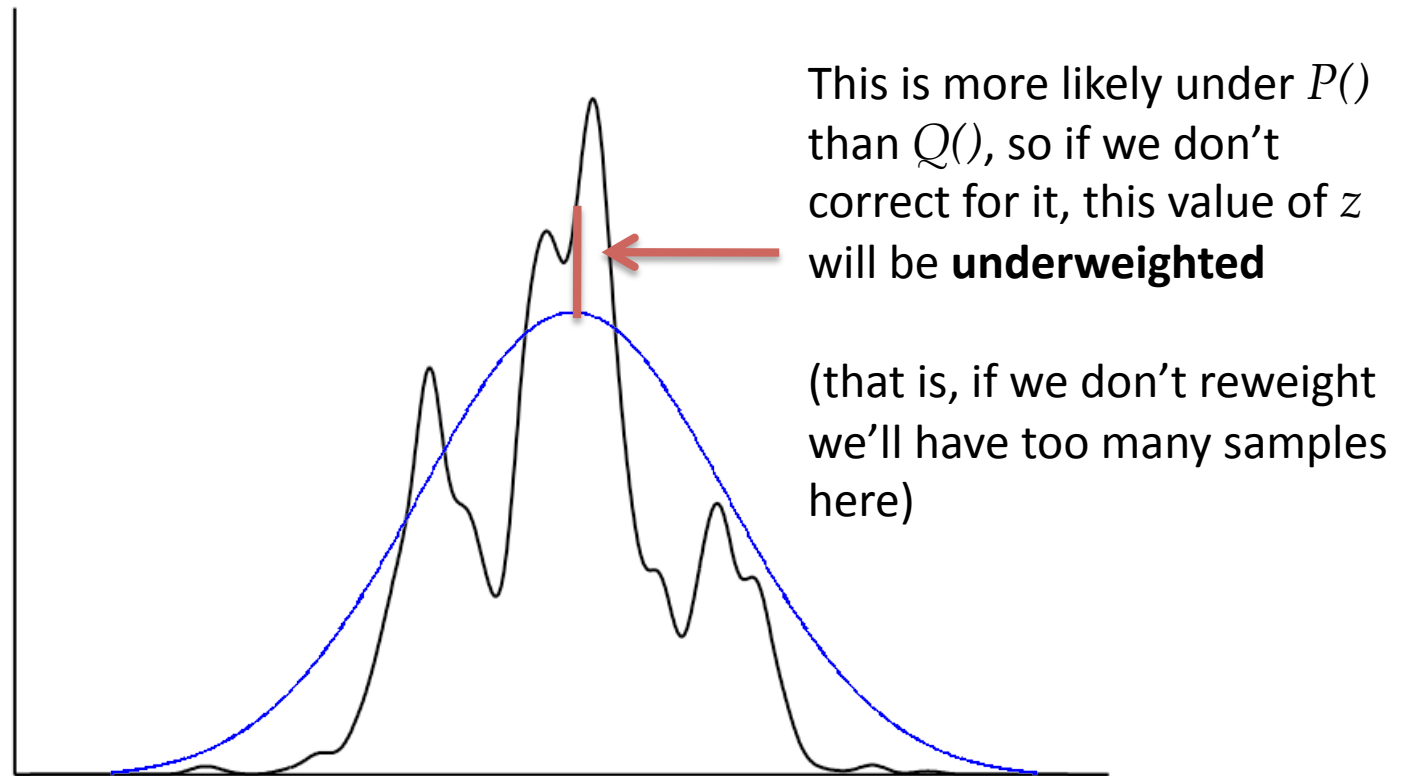
Importance sampling

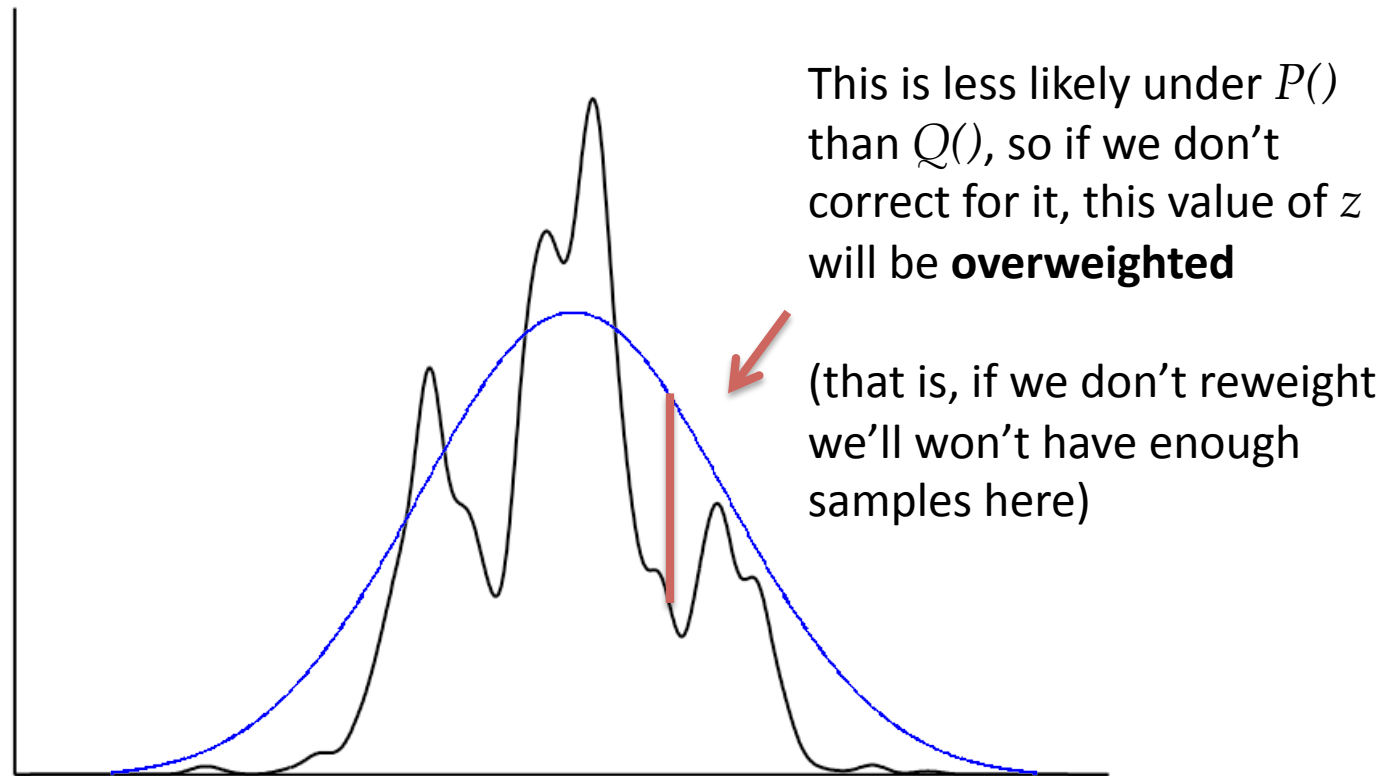
- Even more specifically...
 - Generate the samples from the wrong distribution $Q()$, and then **re-weight** them to correct for the fact that $Q() \neq P()$.
 - That is, the unweighted samples approximate $Q()$
 - But the **weighted samples** approximate $P()$.

The thing we want to generate samples from, called the **target distribution**



The thing we know how to sample from, called the **importance distribution**





How much to reweight:

- We want value z to appear with prob. $P(z)$
- It actually appears with prob. $Q(z)$
- Therefore, the **importance weight** that we assign to a sampled value of z needs to be

$$w(z) = P(z) / Q(z)$$

The Monte Carlo approximation to $P(z)$ based on importance sampling...

$$P(z) \approx \frac{1}{n} \sum_{i=1}^n w(z_i) \delta(z_i)$$

where $z_i \sim Q(z)$

and $w(z) = \frac{P(z)}{Q(z)}$

IMPORTANCE SAMPLING “ALGORITHM”

for $i = 1 : n$

 generate sample z_i from the importance distribution $Q(z)$

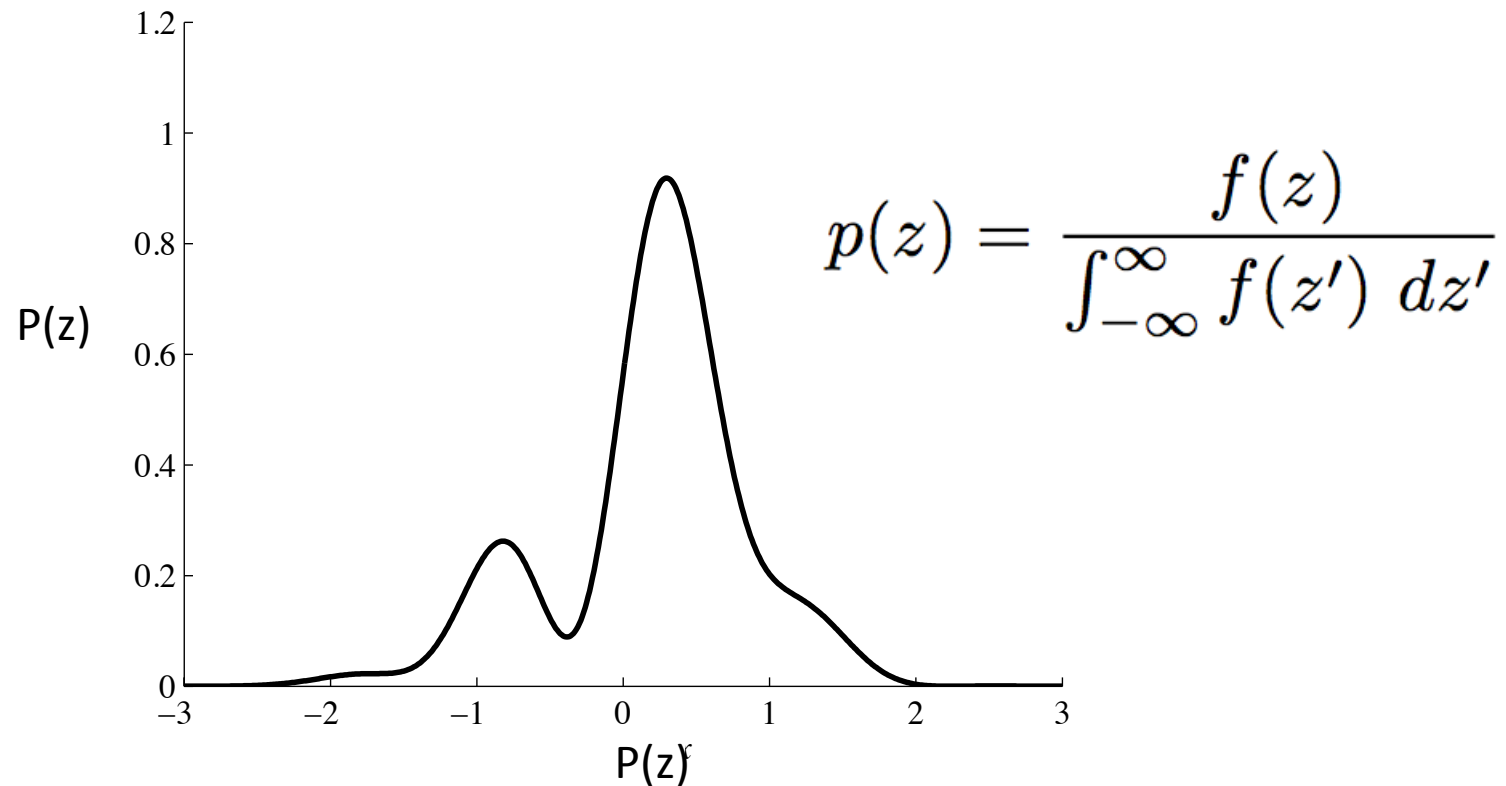
 calculate the importance weight $w(z_i) = P(z_i) / Q(z_i)$

end

treat the set of weighted samples z_i as the approximation to $P(z)$,
and calculate what ever you want; e.g. distribution mean is
approximated as the weighted sample mean, etc.

**IN REAL LIFE, IT'S RARE TO HAVE
THE *WHOLE* EQUATION...**

A very common situation



$$f(z) = \exp(-z^2)(2 + \sin(5z) + \sin(2z))$$

Okay, so we only know $f(z)$

- No problem...
 - We know $f(z)$, where $P(z) = k f(z)$ for unknown constant k
 - So if we calculate $w^*(z) = f(z) / Q(z)$, then:

$$P(z) \approx k \times \frac{1}{n} \sum_{i=1}^n w^*(z_i) \delta(z_i)$$

But $P(z)$ is a probability distribution...

- So the weights ought to sum to 1, right?
Therefore we can approximate k like this...

$$k \approx n \frac{1}{\sum_{i=1}^n w^*(z_i)}$$

- And so our distribution approximation is:

$$P(z) \approx \frac{\sum_{i=1}^n w^*(z_i) \delta(z_i)}{\sum_{i=1}^n w^*(z_i)}$$

- Or, to put it in a simpler way...

Importance weights...

- If we only know $f(z)$, then the importance weights are:

$$w(z_i) = \frac{f(z_i)/Q(z_i)}{\sum_{j=1}^n f(z_j)/Q(z_j)}$$

- Otherwise, everything else is the same as in the earlier version when we knew $P(z)$.
 - same pseudo code, but with this formula for $w(z)$

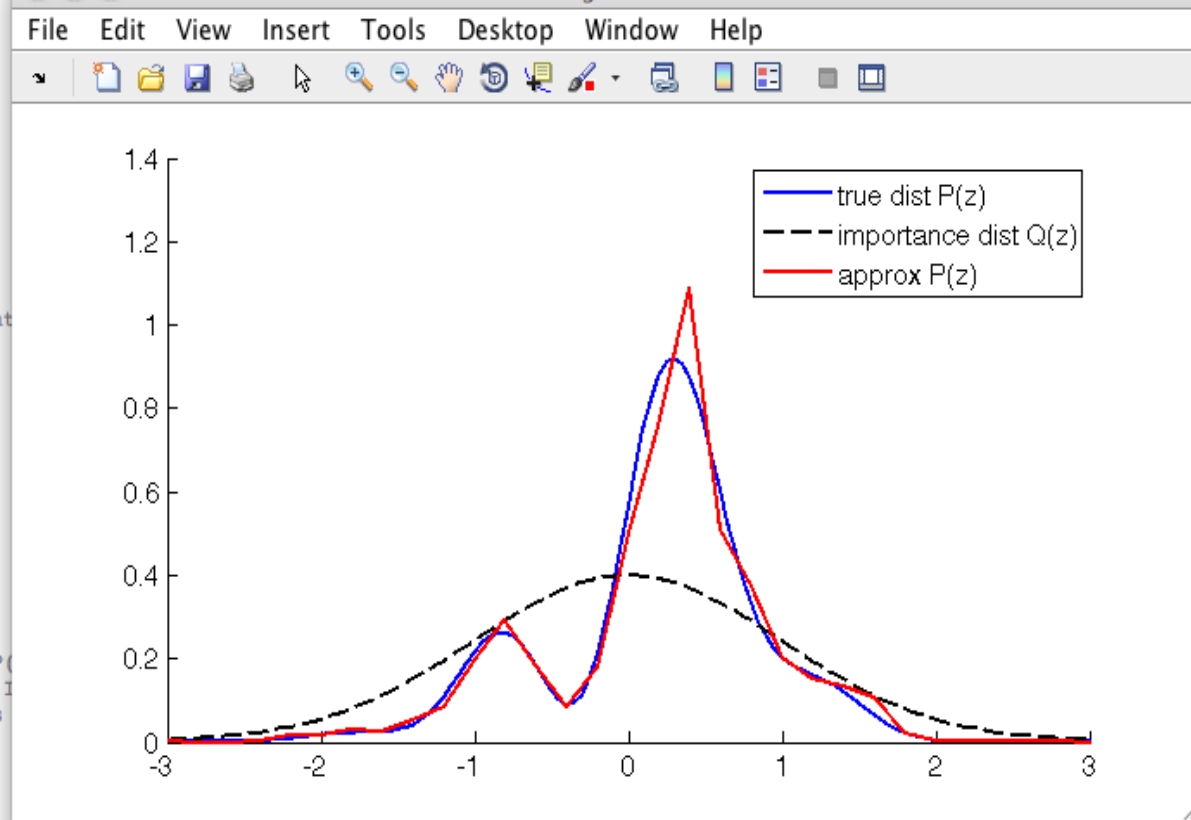
[Demo]

```

>>
>>
>> importancesampler(-0.1,1,1000)
>>

```

Figure 1

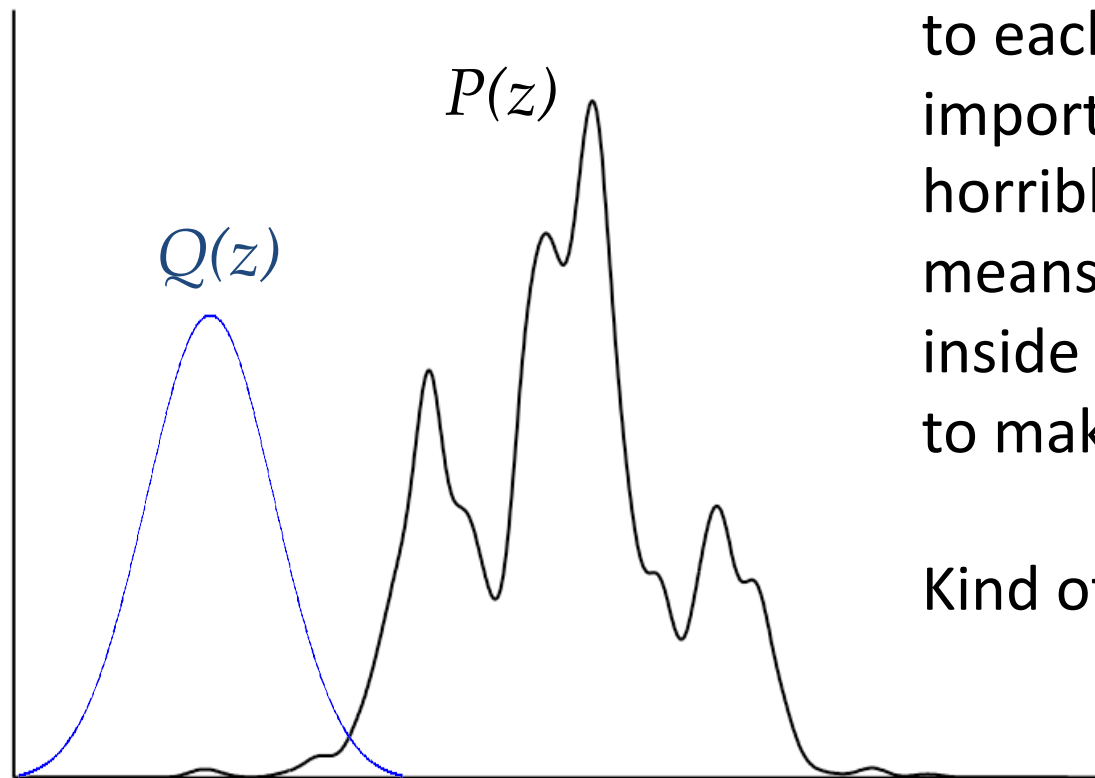


```

3 function importancesampler(mu,sig,n)
4
5 % run an importance sampler for n samples using
6 % a normal distribution as the importance sampler
7 % with mean mu and standard deviation sig. makes
8 % use of the stats toolbox functions normpdf
9 % and normrnd.
10 %
11 % (btw: if your language of preference doesn't
12 % have a "generate from normal dist" random
13 % number generator, google the "Box-Muller"
14 % transform; if it doesn't have a random
15 % number generator for uniform distribution over
16 % [0,1], change languages; or google "Mersenne
17 % twister")
18
19 % n by 1 array of normally distributed numbers
20 % i.e. samples from Q()
21 z = normrnd(mu,sig,n,1);
22
23 % calculate the probability density function
24 % i.e. values of Q(z) for all z
25 qz = normpdf(z,mu,sig);
26
27 % now evaluate the function fz
28 fz = targetfunction(z);
29
30 % calculate the w* weights
31 w = fz ./ qz;
32
33 % normalise w* so that we get importance weight
34 % that actually sum to 1
35 w = w / sum(w);
36
37 % draw a picture
38 drawpicture(z,w,mu,sig)
39
40
41
42
43
44 function fz = targetfunction(z)
45
46 % evaluate the function f(z), where f(z) = k P
47 % is proportional to the target distribution. I
48 % set this up to be able to take vector inputs
49
50 % in this case, f(z) is simple
51 fz = exp(-z.^2) .* (2 + sin(5*z) + sin(2*z));
52
53

```

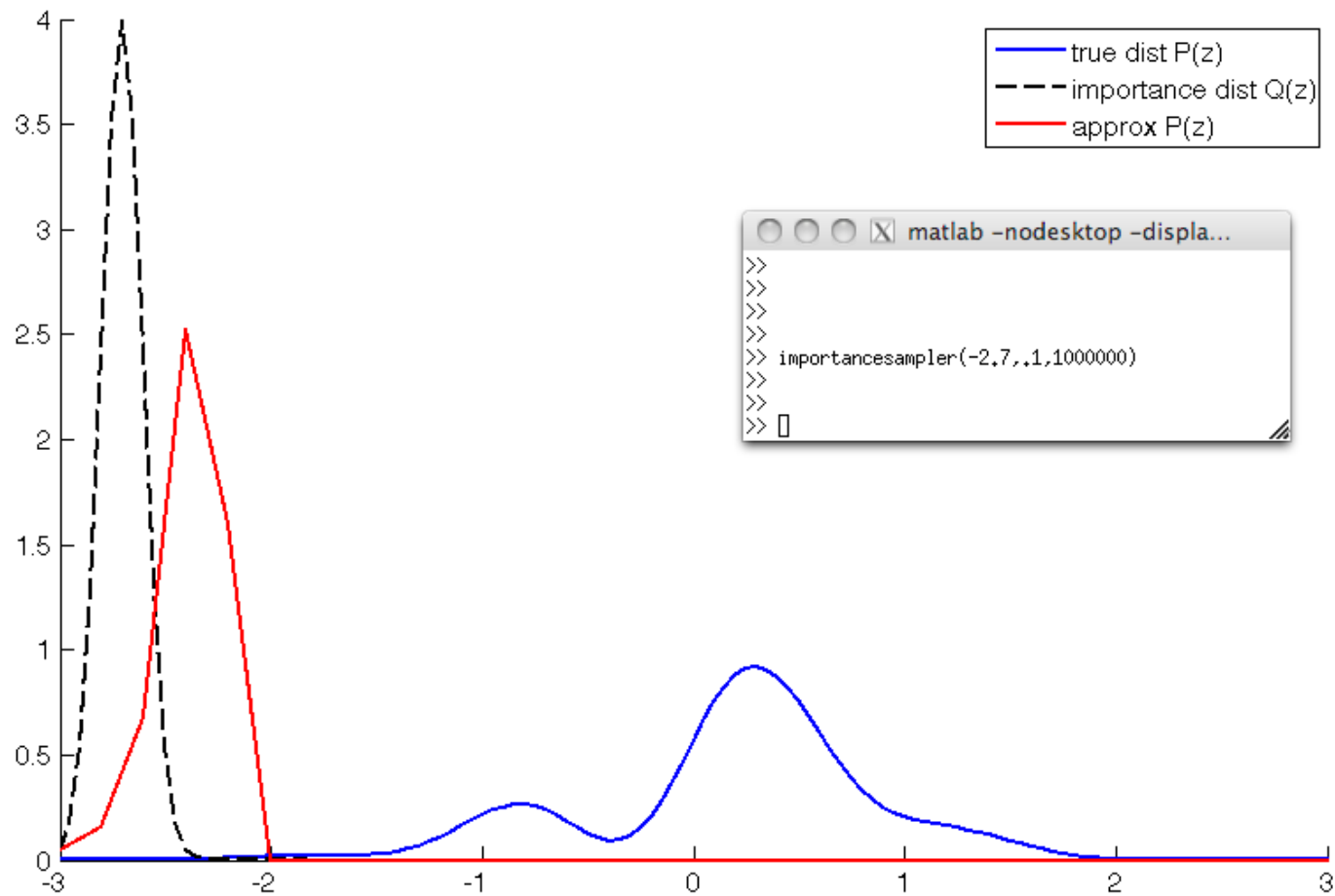
The problem with importance sampling



If Q and P are very different to each other, then importance sampling is horribly inefficient ... which means that you need a lot of inside knowledge about $P()$ to make it work

Kind of defeats the point.

A million samples and it's still complete rubbish.



Interim summary

- Why do importance sampling?
 - In rare cases, it is actually useful (i.e., when you almost know the exact answer)
 - More realistically... importance sampling is a useful precursor to MCMC and particle filtering, which actually are useful

This is good spot to ask questions.

MARKOV CHAIN MONTE CARLO



The problem...

- As before the problem is:
 - Generate samples from $P(z)$
 - All we only know is $f(z)$, where $P(z) = k f(z)$
 - $f(z)$ is a function whose values we can calculate
- What we'd like is:
 - For the method to actually be useful this time.

The tech note...

Gives you a simple walk through of the “standard” MCMC algorithm, with an illustration of how it needs to be tweaked in practice.

The Metropolis-Hastings Algorithm

COMPSCI 3016: Computational Cognitive Science
Dan Navarro & Amy Perfors
University of Adelaide

Abstract

This note provides a discussion of the Metropolis-Hastings algorithm. If anything the notes doesn't make sense please contact me (these notes were written by Dan: daniel.navarro@adelaide.edu.au) and I'll try to fix them!

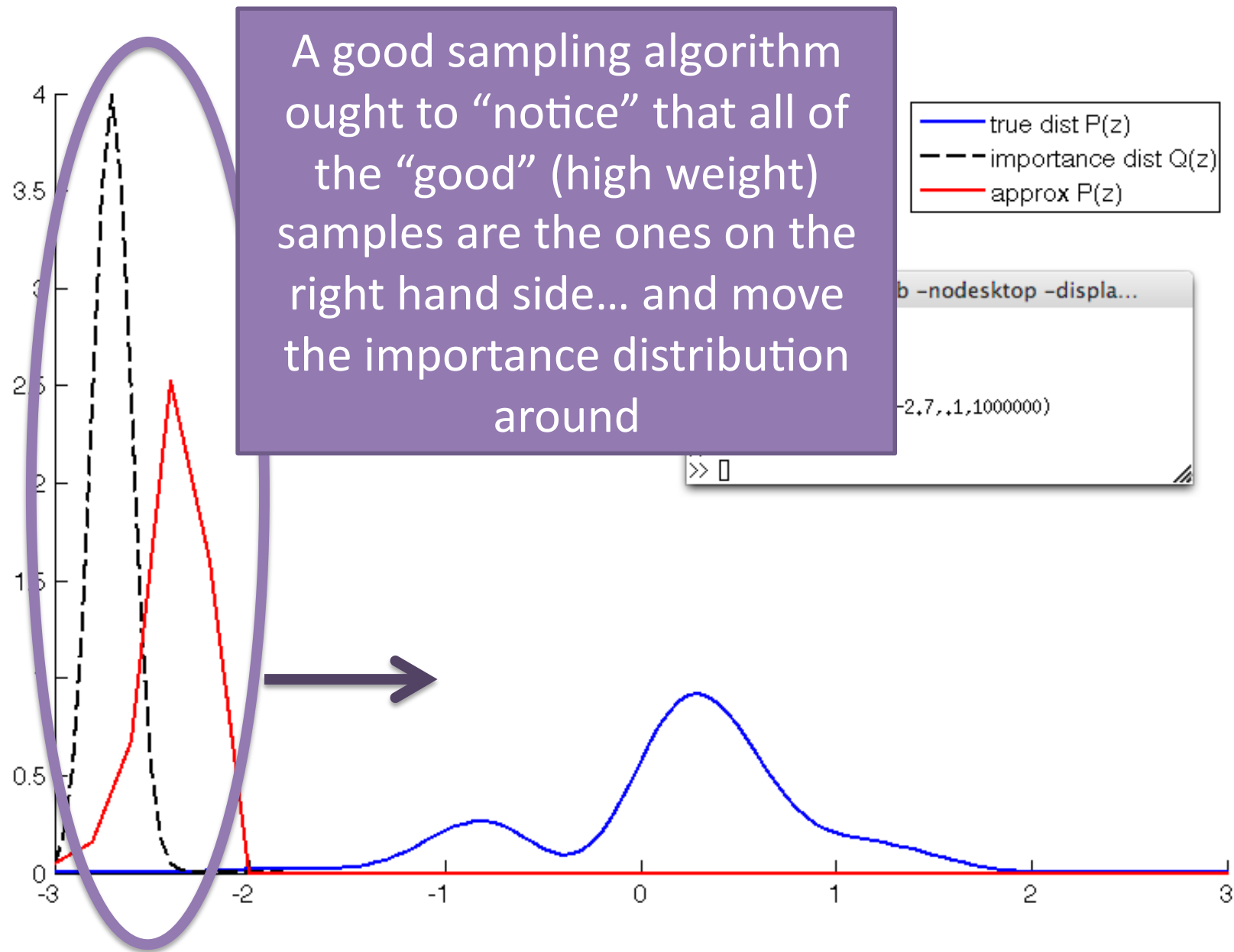
The problem to be solved

The Metropolis-Hastings algorithm is the most popular example of a *Markov chain Monte Carlo* (MCMC) method. The basic problem that it solves is to provide a method for sampling from some generic distribution, $P(x)$. The idea is that in many cases, you know how to write out the equation for the probability $P(x)$, but you don't know how to generate a random number from this distribution, $x \sim P(x)$. This is the situation where MCMC is handy. In fact, for the Metropolis-Hastings algorithm we don't even need to know how to

The idea...

- Can we do something kind of like importance sampling, but where we allow the importance distribution to adapt*, and move around on its own?
- With any luck, it would move to the right spot, and generate good samples, without us needing to do too much fine tuning.

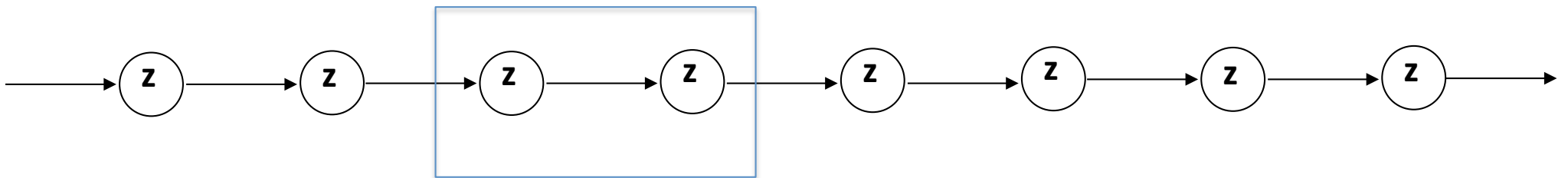
* “adaptive importance sampling” is something different, by the way.



Hold onto that thought, because we'll
come back to it. In the meantime...

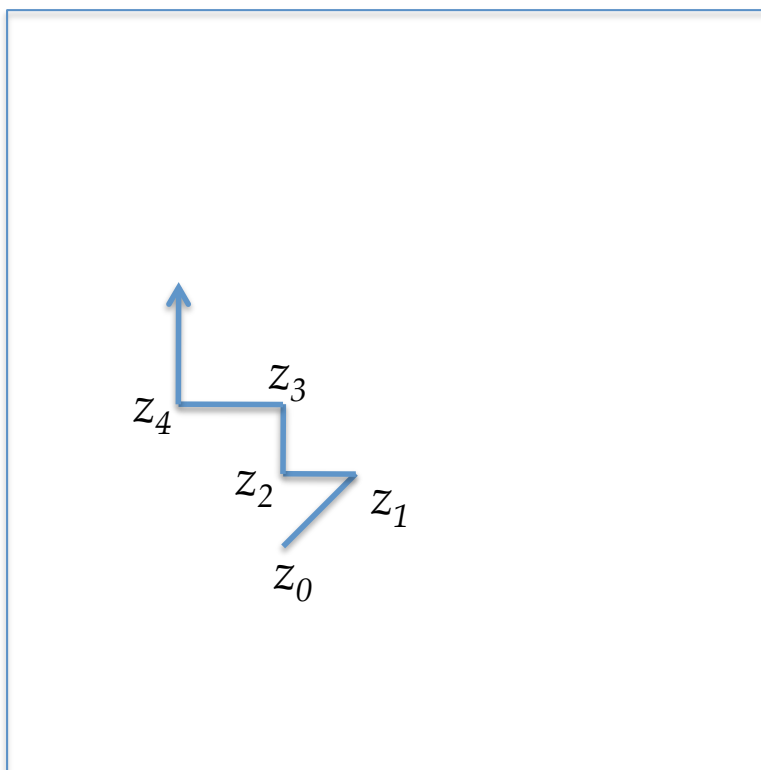
Markov chain Monte Carlo

- **Monte Carlo:** because it involves generating samples from some probability distribution
- **Markov chain:** because the sample z_t depends on z_{t-1} , but is otherwise independent of z_0, z_1, \dots, z_{t-2}



We make use of the information in z_{t-1} to help us make a good choice for z_t

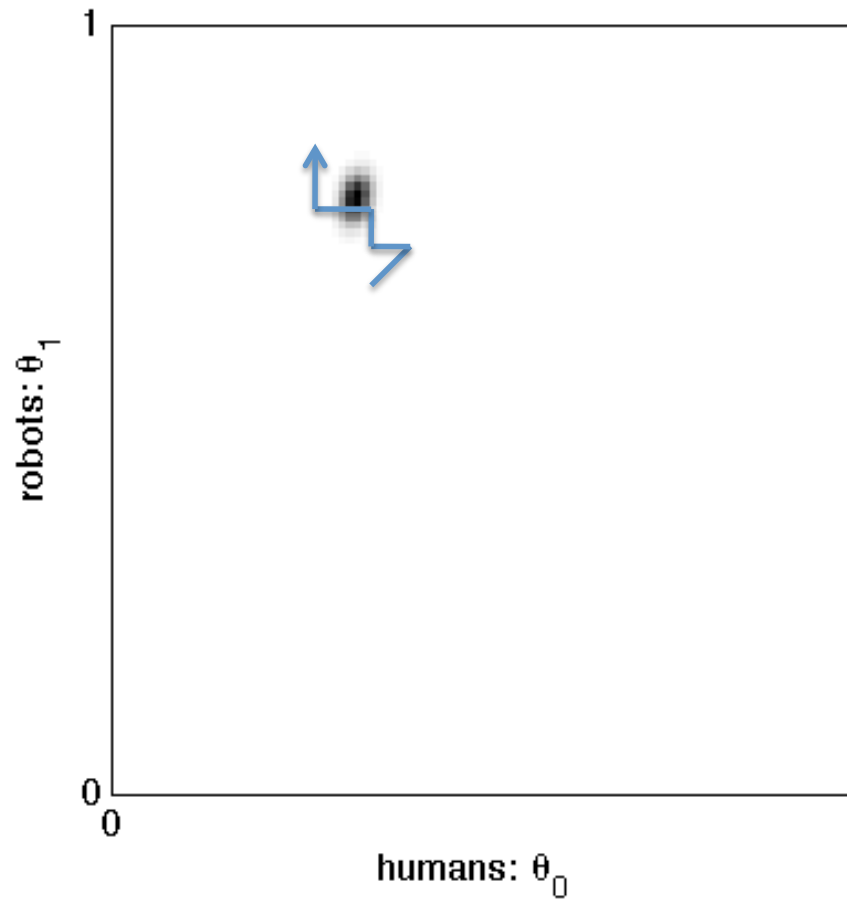
Markov chain Monte Carlo



At each step, we have some rule that describes the **transition probability** of moving from z_{t-1} to z_t

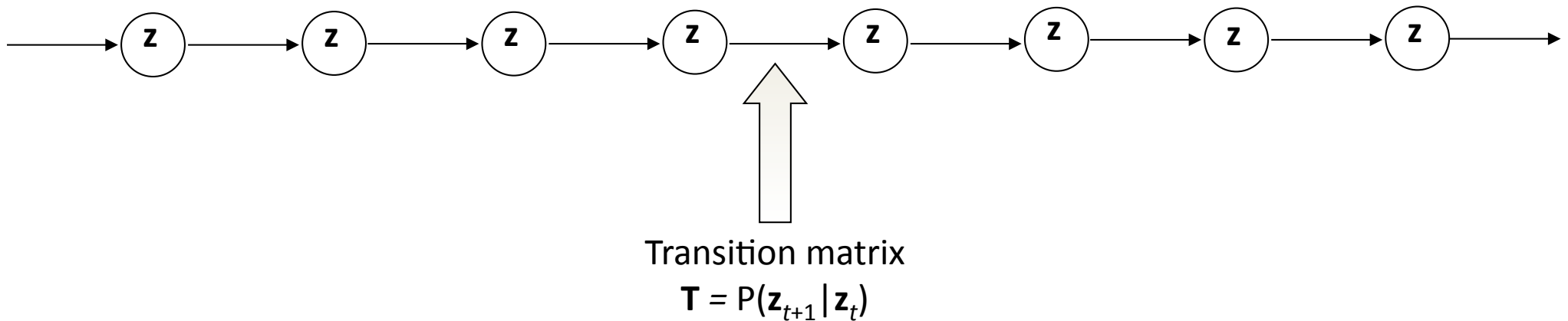
The desired behaviour

It ought to “jitter” around the high probability region of the posterior distribution



Markov chain Monte Carlo

- **THE TRICK:** rig the Markov chain so that it will converge to the target distribution, and draw samples from that chain



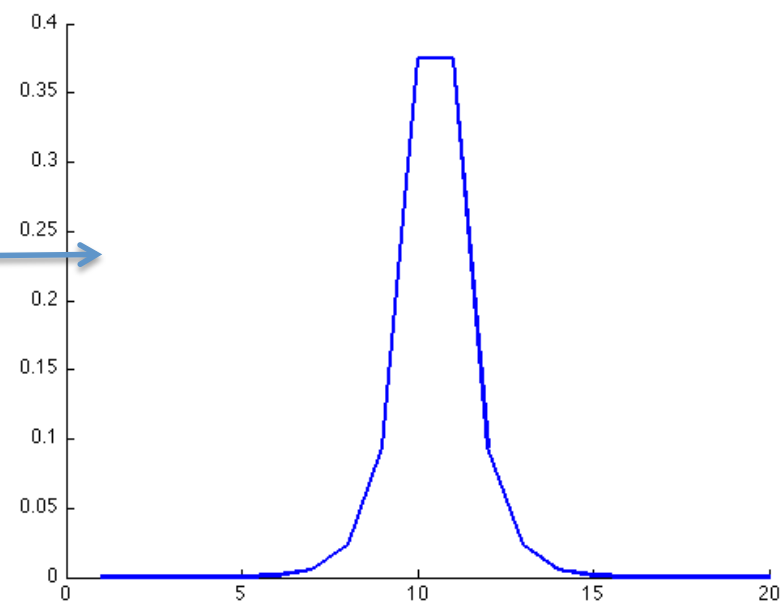
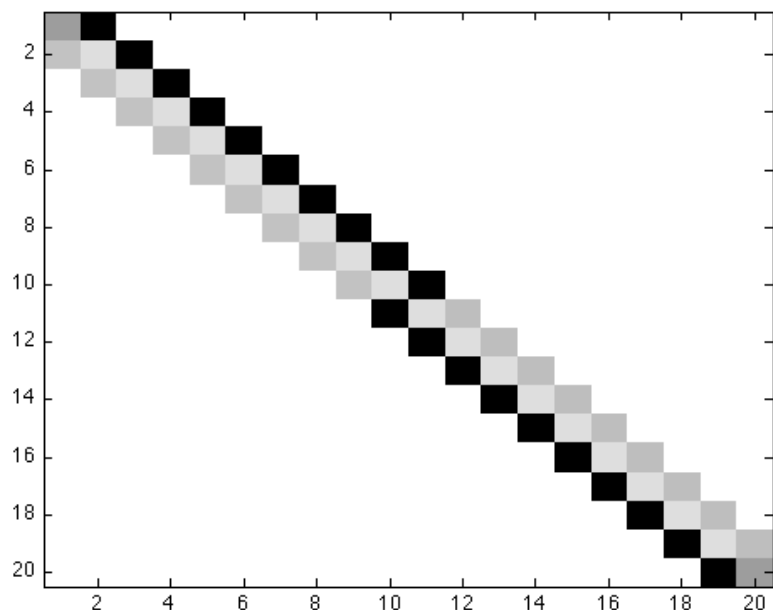
Convergence?

- Ergodicity of (some) Markov chains...
 - Suppose that I draw my initial state z_0 from some probability distribution \mathbf{P}_0 .
 - I then sample all future states from a Markov chain with transition matrix \mathbf{T}
 - What is the probability distribution over the n -th sample in the chain?

$$\mathbf{P}_n = \mathbf{T}^n \mathbf{P}_0$$

What happens when n gets large?

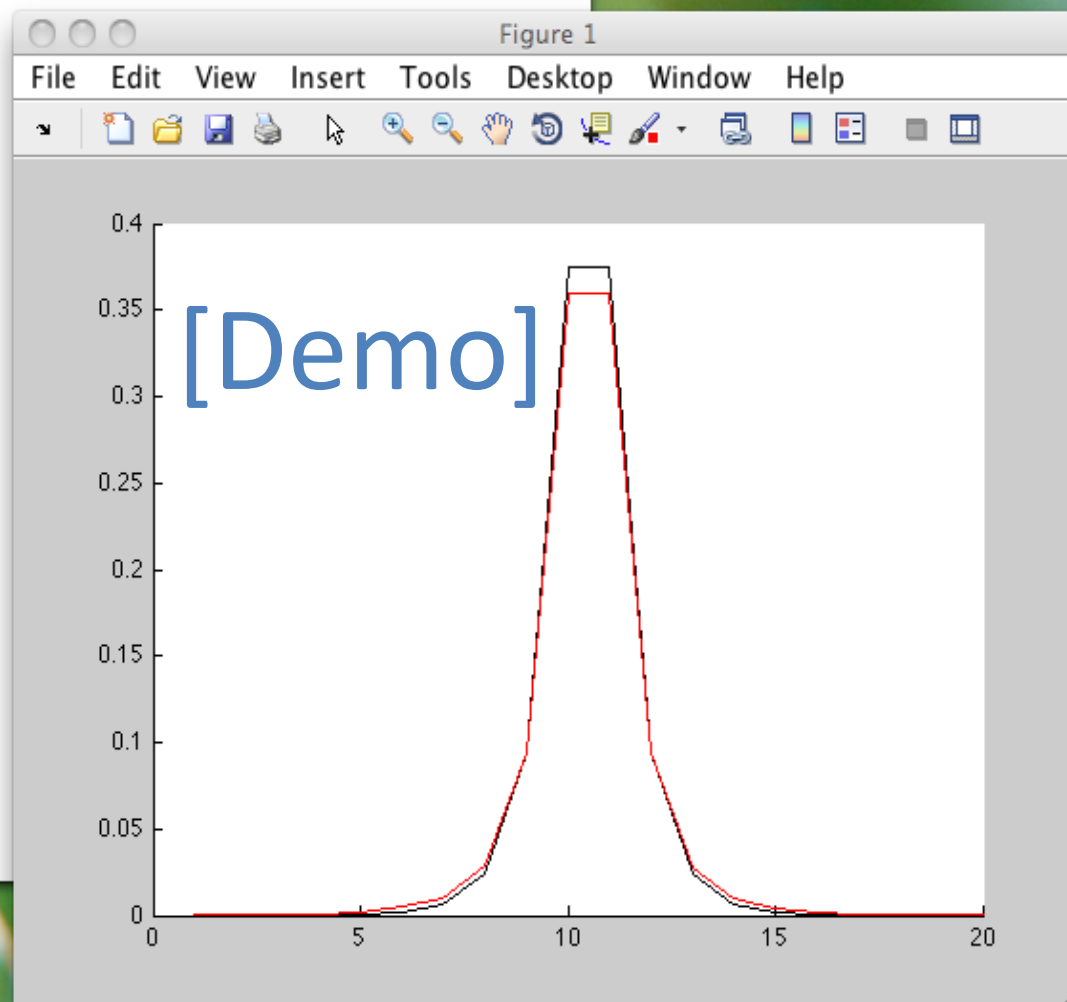
- Eventually the chain becomes independent of the start point, and only depends on the transition matrix...



```
markovchainconvergence.m
1 function markovchainconvergence(P0,n)
2
3     plength = .1;
4
5     P0 = reshape(P0,[],1); % make P0 a column vector
6     m = length(P0); % count the states
7
8     % create a transition matrix T
9     ind = floor(m*.5);
10    p = [repmat(.5,ind,ind), repmat(.5,m-ind,ind)];
11    T = diag(p(1:ind));
12    T = T + diag(p(ind+1:m));
13    T(1,1) = 1-p(1);
14    T(m,m) = p(m);
15    T = T*.9 + .1*ones(m,m);
16    T = T'; % <- transpose
17
18    % stationary distribution
19    S = T^10000;
20    S = S(:,1);
21
22    % simulate the chain
23    Pt = P0;
24    figure(1); clear;
25    plot(1:m,S,'k');
26    h = plot(1:m,Pt,'r');
27    pause(plength);
28
29    for t = 1:n
30        Pt = T * Pt;
31        set(h,'ydata',Pt);
32        pause(plength);
33    end
34
35
```

Line: 16 Column: 68

matlab -nodesktop -display \$DISPLAY



A Markov chain with $P(z)$ as the stationary distribution

- Metropolis-Hastings... the transitions are broken into two distinct parts...
 - A “proposal distribution” $Q(z^* | z_{t-1})$ for generating new candidate z values; depends on old z value.
 - An “acceptance rule” $A(z^* | z_{t-1})$ that depends on the target distribution $P(z)$ (just $f(z)$ actually).
 - If candidate is accepted, set $z_t = z^*$.
 - If candidate is rejected, set $z_t = z_{t-1}$.

METROPOLIS-HASTINGS ALGORITHM, FOR GENERATING SAMPLES FROM AN ARBITRARY DISTRIBUTION, $z \sim P(\cdot)$

set $t = 0$

choose a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* \mid z)$ that you can sample from

do while $t < \infty$

 increment: $t = t + 1$

 generate a “candidate” $z^* \sim Q(\cdot \mid z_{t-1})$ from the proposal

 calculate the “acceptance probability”, $A(z^* \mid z_{t-1})$

 generate $u \sim U([0,1])$ from a uniform distribution

 if $u \leq A$, accept the candidate: set $z_t = z^*$

 elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$

METROPOLIS-HASTINGS ALGORITHM, FOR GENERATING SAMPLES FROM AN ARBITRARY DISTRIBUTION, $z \sim P(\cdot)$

Notation: the “dot” just means that we’re talking about the distribution $P(\cdot)$, not the probability $P(z)$ that the distribution assigns to some event

set $t = 0$

choose a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* | z)$ that you can sample from
do while $t < \infty$

increment: $t = t + 1$

generate a “candidate” $z^* \sim Q(\cdot | z_{t-1})$ from the proposal

calculate the “acceptance probability”, $A(z^* | z_{t-1})$

generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z_t = z^*$

elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$

METROPOLIS-HASTINGS ALGORITHM, FOR GENERATING SAMPLES FROM AN ARBITRARY DISTRIBUTION, $z \sim P(\cdot)$

set $t = 0$

choose a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* \mid z)$ that

do while $t < \infty$

increment: $t = t + 1$

generate a “candidate” $z^* \sim Q(\cdot \mid z_{t-1})$ from the proposal

calculate the “acceptance probability”, $A(z^* \mid z_{t-1})$


generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z_t = z^*$

elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$

$P(\cdot)$ is called the “target” distribution.



The acceptance probability is:

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

(we'll come back to this in a minute...)

CHOOSE a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* | z)$ that you can sample from
do while $t < \infty$

increment: $t = t + 1$

generate a “candidate” $z^* \sim Q(\cdot | z_{t-1})$ from the proposal

calculate the “acceptance probability”, $A(z^* | z_{t-1})$

generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z_t = z^*$

elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$

METROPOLIS-HASTINGS ALGORITHM, FOR GENERATING SAMPLES FROM AN ARBITRARY DISTRIBUTION, $z \sim P(\cdot)$

set $t = 0$

choose a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* | z)$ that you can sample from
do while $t < \infty$

increment: $t = t + 1$

generate a “candidate” $z^* \sim Q(\cdot | z)$


calculate the “acceptance probability” $A = \min\left(1, \frac{P(z^*)Q(z | z^*)}{P(z)Q(z^* | z)}\right)$

generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z = z^*$

elseif $u > A$, reject the candidate: set $z = z$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$



The algorithm is VERY sensitive to the choice of the proposal distribution

METROPOLIS-HASTINGS ALGORITHM, FOR GENERATING SAMPLES FROM AN ARBITRARY DISTRIBUTION, $z \sim P(\cdot)$

set $t = 0$

choose a initial value z_0 arbitrarily

choose a “proposal distribution” $Q(z^* | z)$

do while $t < \infty$

increment: $t = t + 1$

generate a “candidate” $z^* \sim Q(\cdot | z_{t-1})$ from the proposal

calculate the “acceptance probability”, $A(z^* | z_{t-1})$

generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z_t = z^*$

elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

as $t \rightarrow \infty$, the “state” of the algorithm converges $z_t \sim P(\cdot)$

You can't run it for an infinite number of iterations, so you need to make decisions about how long to wait!


The acceptance probability

- Here's the equation again:

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

- Note that it depends on both the target distribution P , and the proposal distribution Q

The “min” part isn’t interesting: it’s just there to ensure that A is actually a probability.


$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

Probability of the candidate z^* under the target distribution, relative to the probability of the current state z_{t-1} . In most cases, this is something we can calculate (see later)

A correction factor. If your proposal makes it more likely that you try the move $z_{t-1} \rightarrow z^*$ than the reverse move, (i.e., if z^* had been the current state and you were proposing to move to the new location z_{t-1}) you end up with a bias that needs to be corrected

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

Suppose we chose Q like this:

$$z^* = z_{t-1} + y + 1/3$$

where $y \sim \text{beta}(2,4)$

A correction factor. If your proposal makes it more likely that you try the move $z_{t-1} \rightarrow z^*$ than the reverse move, (i.e., if z^* had been the current state and you were proposing to move to the new location z_{t-1}) you end up with a bias that needs to be corrected

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

Suppose we chose Q like this:

$$z^* = z_{t-1} + y + 1/3$$

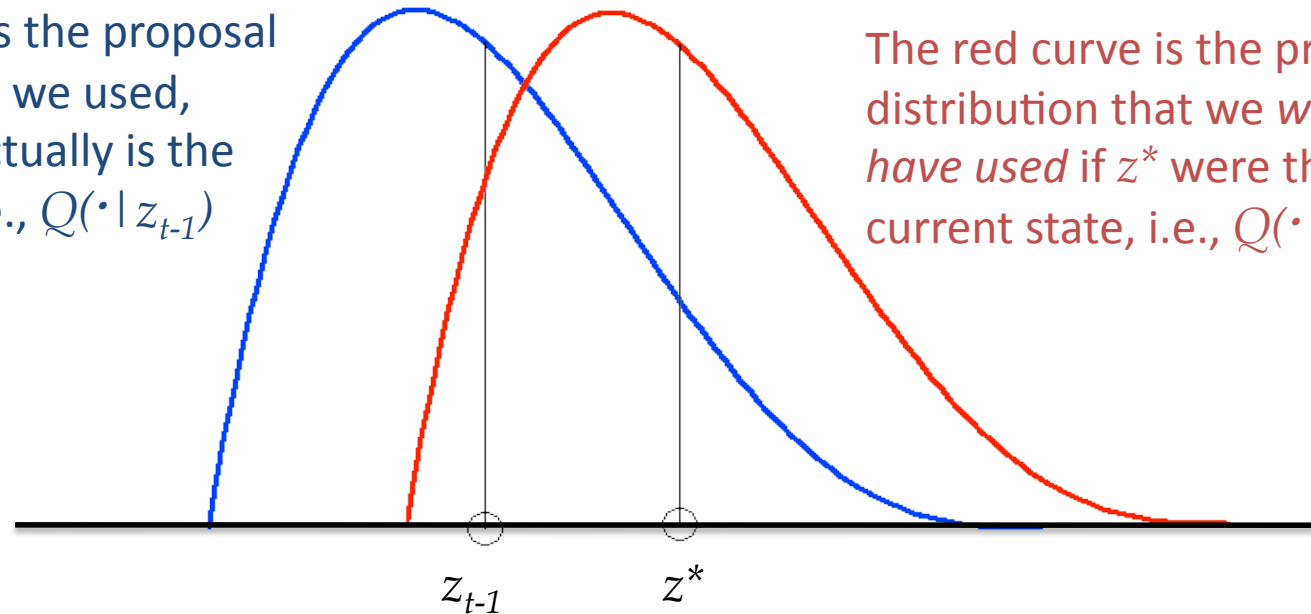
where $y \sim \text{beta}(2,4)$

A correction factor. If your proposal makes it more likely that you try the move $z_{t-1} \rightarrow z^*$ than the reverse move, (i.e., if z^* had been the current state and you were proposing to move to the new location z_{t-1}) you end up with a bias that needs to be corrected

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

The blue curve is the proposal distribution that we used, given that z_{t-1} actually is the current state, i.e., $Q(\cdot | z_{t-1})$

The red curve is the proposal distribution that we *would have used* if z^* were the current state, i.e., $Q(\cdot | z^*)$



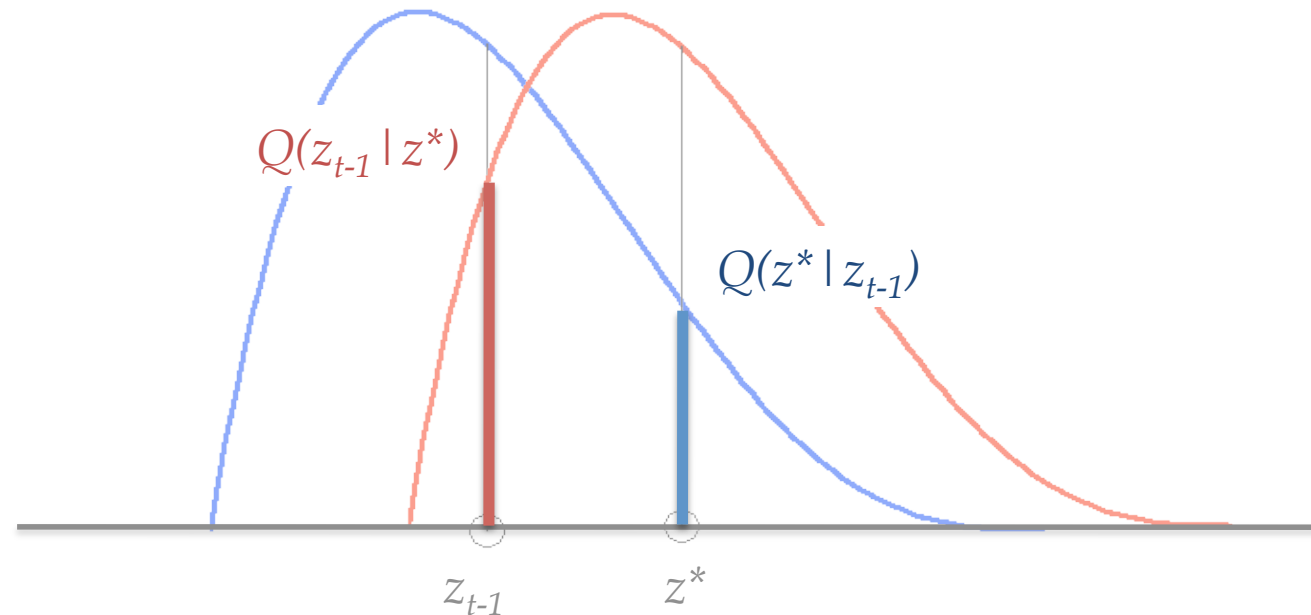
Suppose we chose Q like this:

$$z^* = z_{t-1} + y + 1/3$$

where $y \sim \text{beta}(2,4)$

A correction factor. If your proposal makes it more likely that you try the move $z_{t-1} \rightarrow z^*$ than the reverse move, (i.e., if z^* had been the current state and you were proposing to move to the new location z_{t-1}) you end up with a bias that needs to be corrected

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$



If the proposal distribution is **ASYMMETRIC**, then in general you will find that $Q(z_{t-1} | z^*) \neq Q(z^* | z_{t-1})$

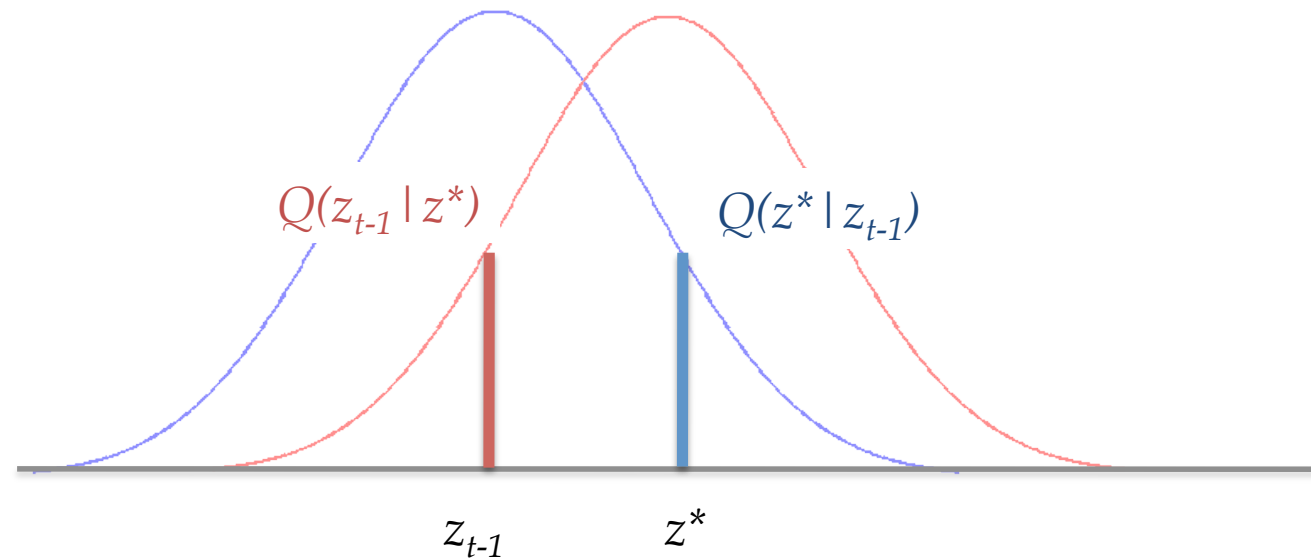
Suppose we chose Q like this:

$$z^* = z_{t-1} + y$$

where $y \sim \text{normal}(0,1)$

A correction factor. If your proposal makes it more likely that you try the move $z_{t-1} \rightarrow z^*$ than the reverse move, (i.e., if z^* had been the current state and you were proposing to move to the new location z_{t-1}) you end up with a bias that needs to be corrected

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

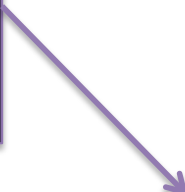


If the proposal distribution is **SYMMETRIC**, then

$$Q(z_{t-1} | z^*) = Q(z^* | z_{t-1}).$$

This is called a “Metropolis sampler”

Metropolis – Hastings allows asymmetric proposal distributions, so you need to calculate the correction factor


$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \times \frac{Q(z_{t-1}|z^*)}{Q(z^*|z_{t-1})} \right)$$

Metropolis algorithm only allows symmetric proposal distributions, so we can ignore Q when calculating the acceptance probability!

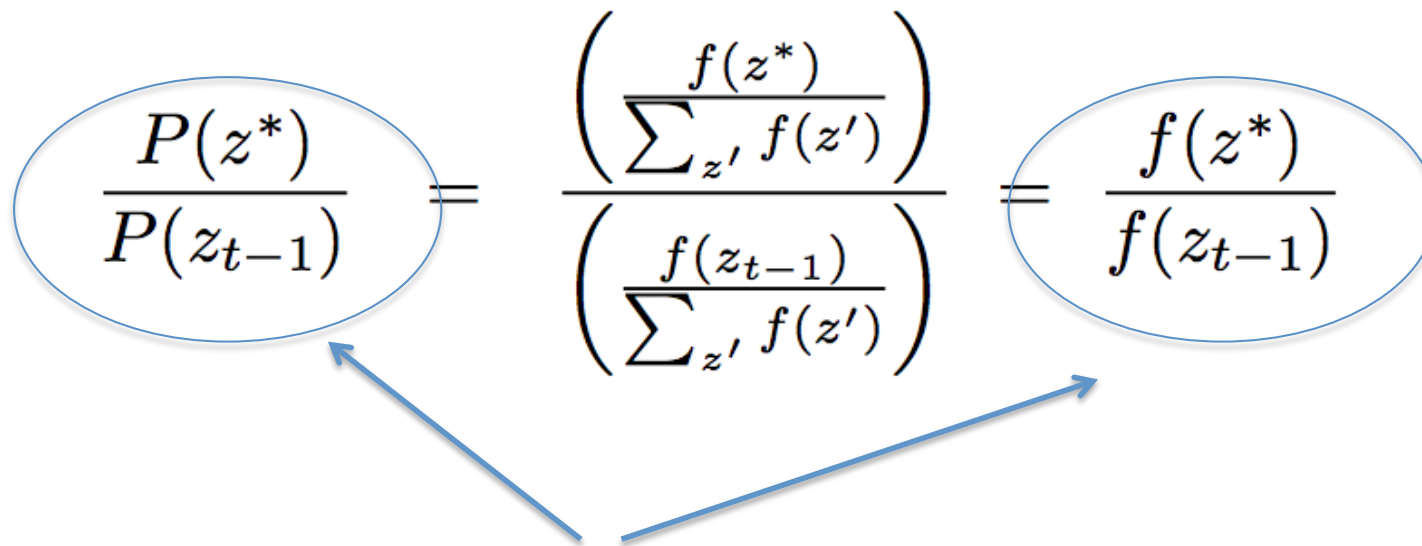
$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

Why is this awesome?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

Why is this awesome?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

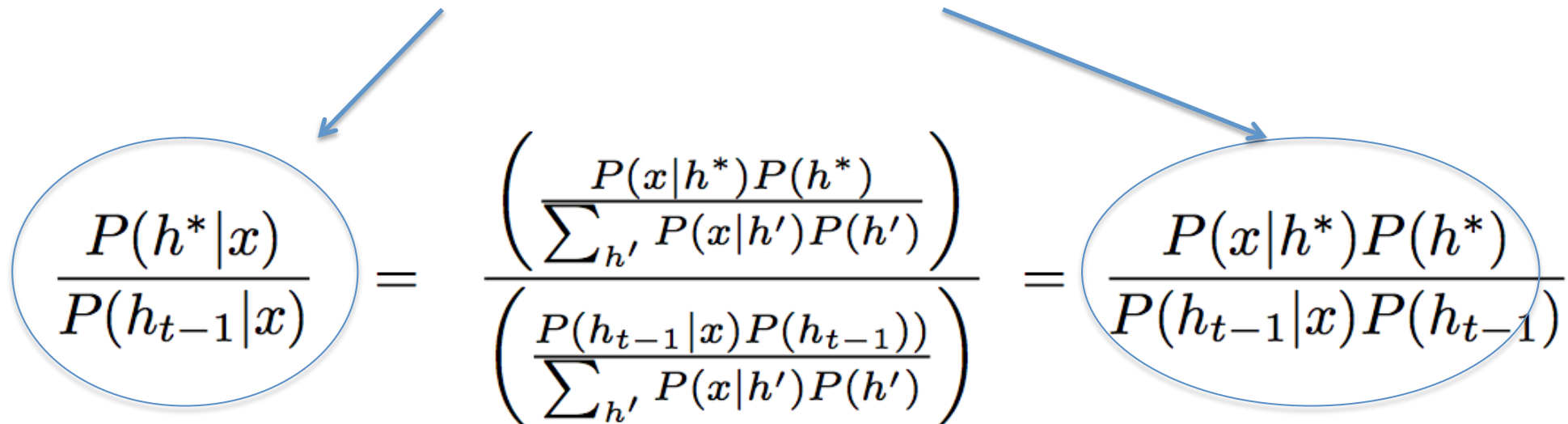
$$\frac{P(z^*)}{P(z_{t-1})} = \frac{\left(\frac{f(z^*)}{\sum_{z'} f(z')} \right)}{\left(\frac{f(z_{t-1})}{\sum_{z'} f(z')} \right)} = \frac{f(z^*)}{f(z_{t-1})}$$


In the generic case.... we
only need to use $f(z)$.

Why is this awesome?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

In the Bayesian special case, we don't need to compute the sum over the hypothesis space!



The diagram illustrates the simplification of a Bayesian formula. It shows a sequence of three expressions connected by equals signs. The first expression, $\frac{P(h^*|x)}{P(h_{t-1}|x)}$, is enclosed in a blue oval and has a blue arrow pointing to it from the text above. The second expression is a fraction of two sums over the hypothesis space h' : $\frac{\left(\frac{P(x|h^*)P(h^*)}{\sum_{h'} P(x|h')P(h')} \right)}{\left(\frac{P(h_{t-1}|x)P(h_{t-1})}{\sum_{h'} P(x|h')P(h')} \right)}$. The third expression, $\frac{P(x|h^*)P(h^*)}{P(h_{t-1}|x)P(h_{t-1})}$, is also enclosed in a blue oval and has a blue arrow pointing to it from the text above. The common denominator sum is canceled out, resulting in the simplified expression.

$$\frac{P(h^*|x)}{P(h_{t-1}|x)} = \frac{\left(\frac{P(x|h^*)P(h^*)}{\sum_{h'} P(x|h')P(h')} \right)}{\left(\frac{P(h_{t-1}|x)P(h_{t-1})}{\sum_{h'} P(x|h')P(h')} \right)} = \frac{P(x|h^*)P(h^*)}{P(h_{t-1}|x)P(h_{t-1})}$$

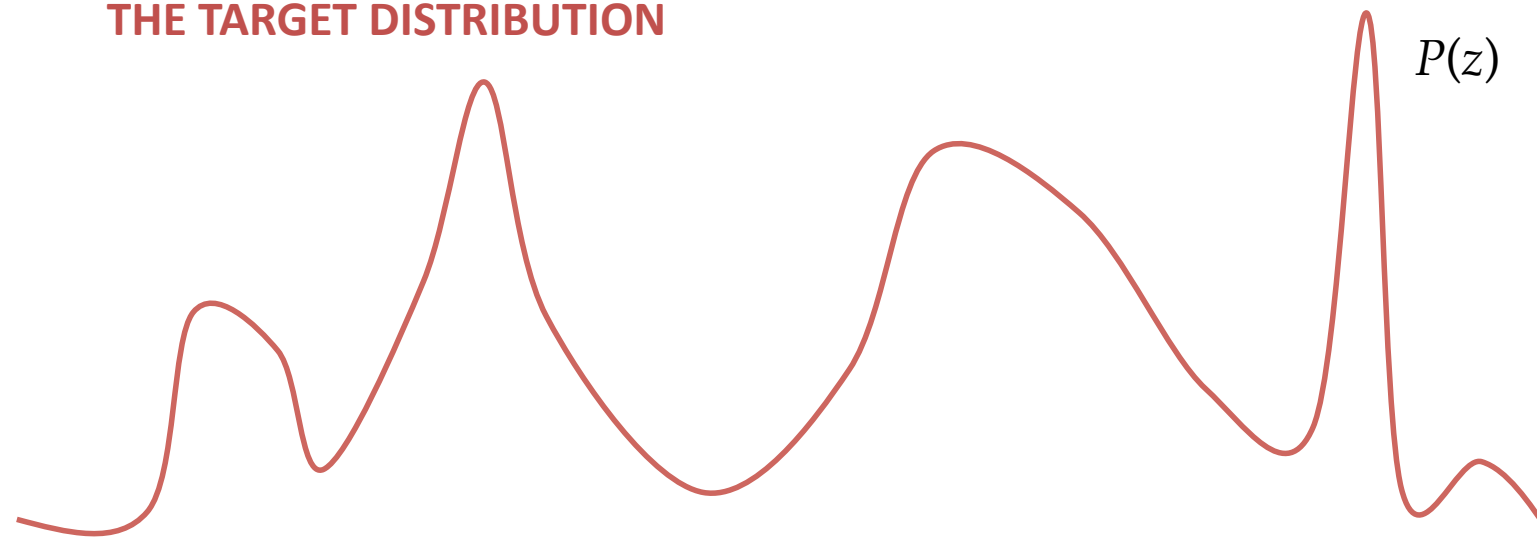
So, how does the Metropolis algorithm work?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

So, how does the Metropolis algorithm work?

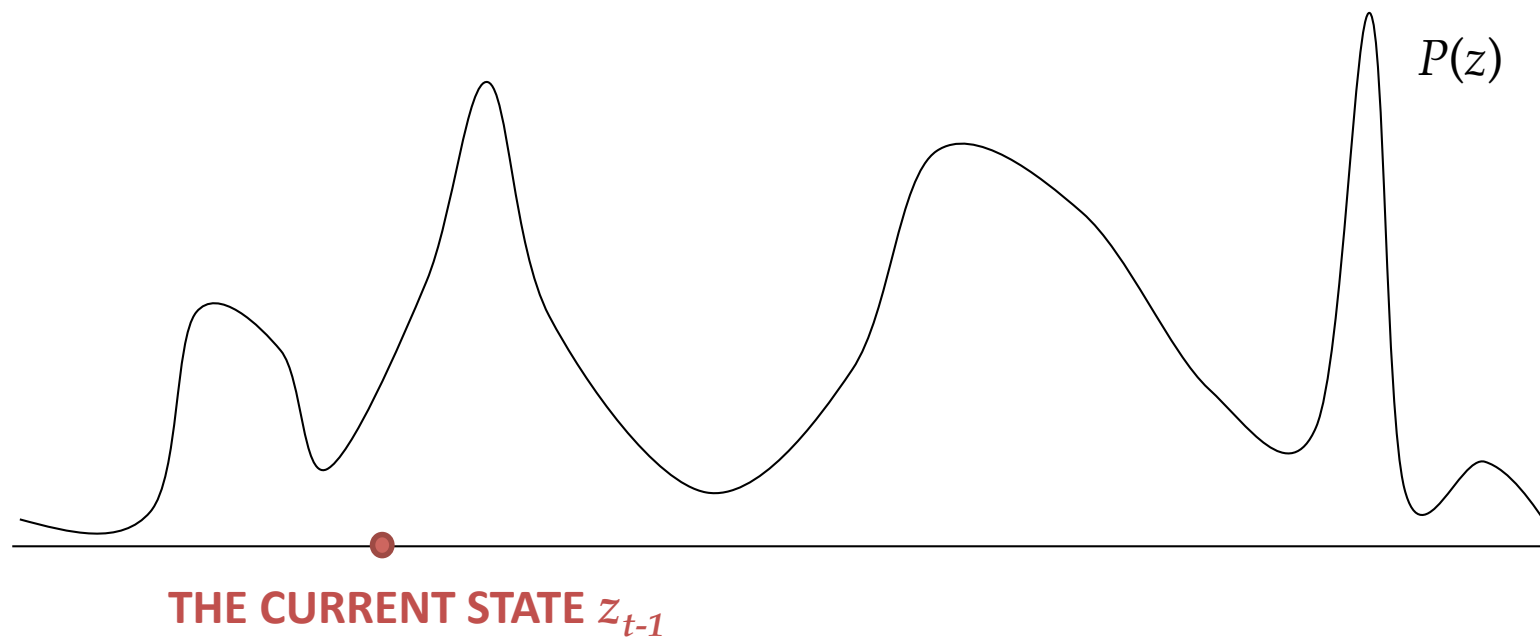
$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$

THE TARGET DISTRIBUTION



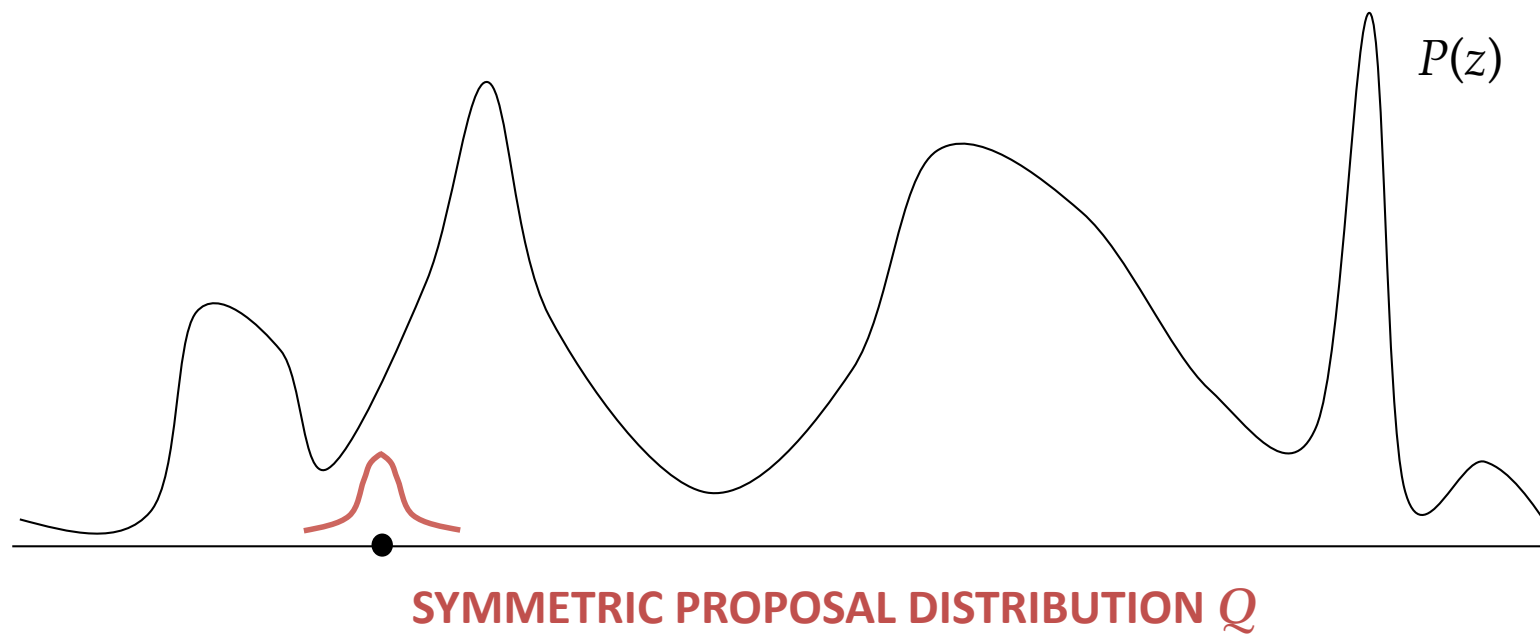
So, how does the Metropolis algorithm work?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



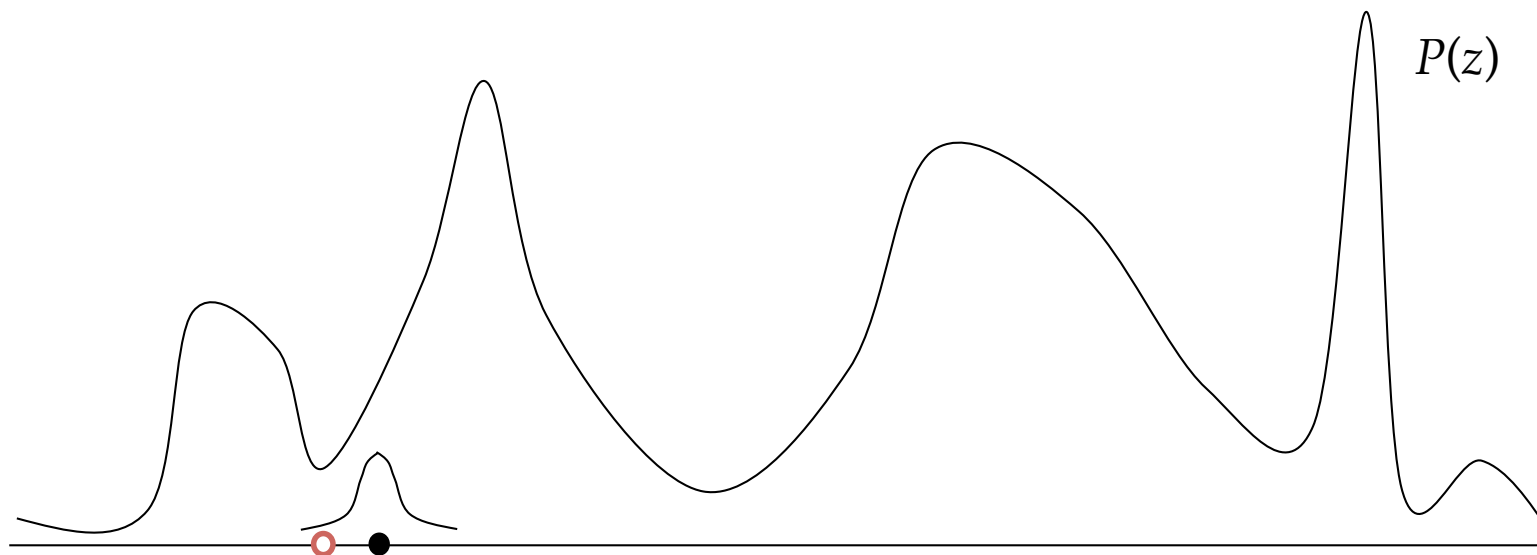
So, how does the Metropolis algorithm work?

$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



So, how does the Metropolis algorithm work?

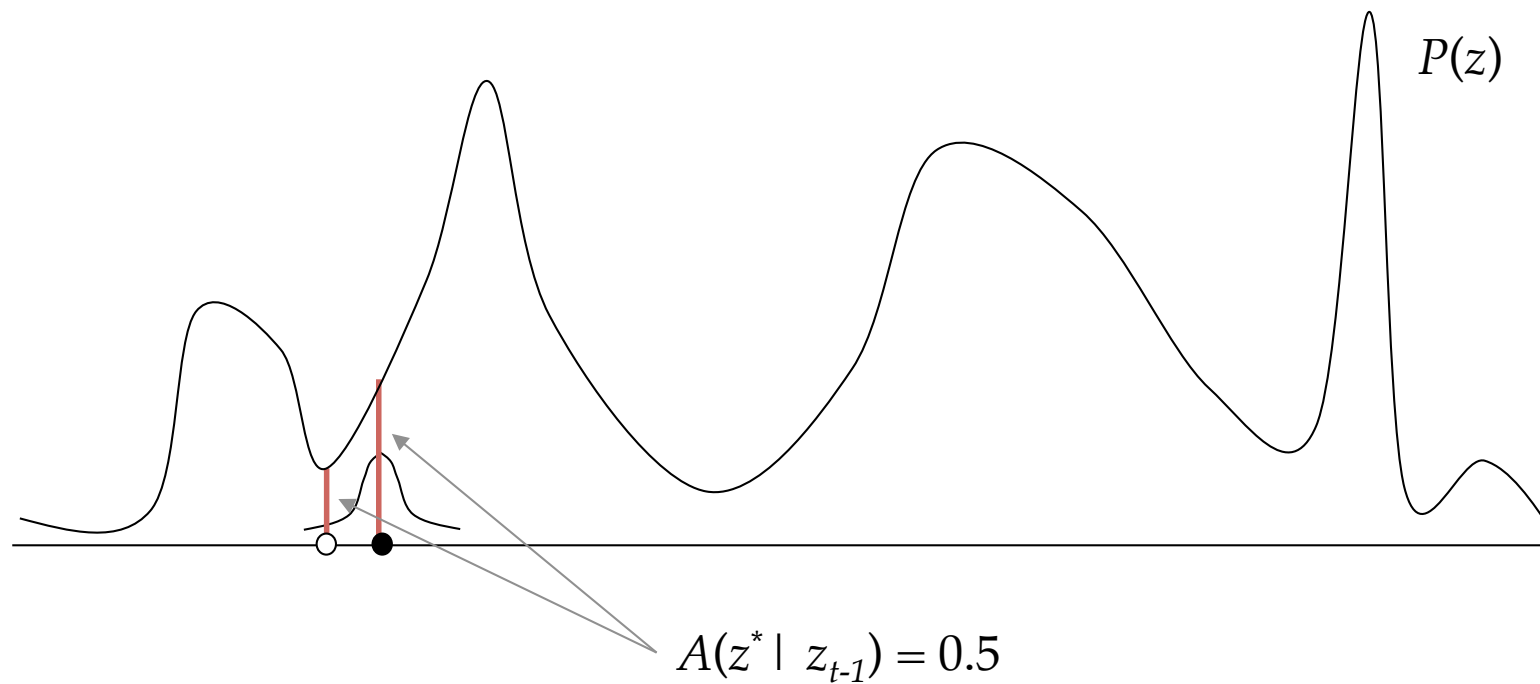
$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



**CANDIDATE z^* GENERATED FROM
THE PROPOSAL DISTRIBUTION**

So, how does the Metropolis algorithm work?

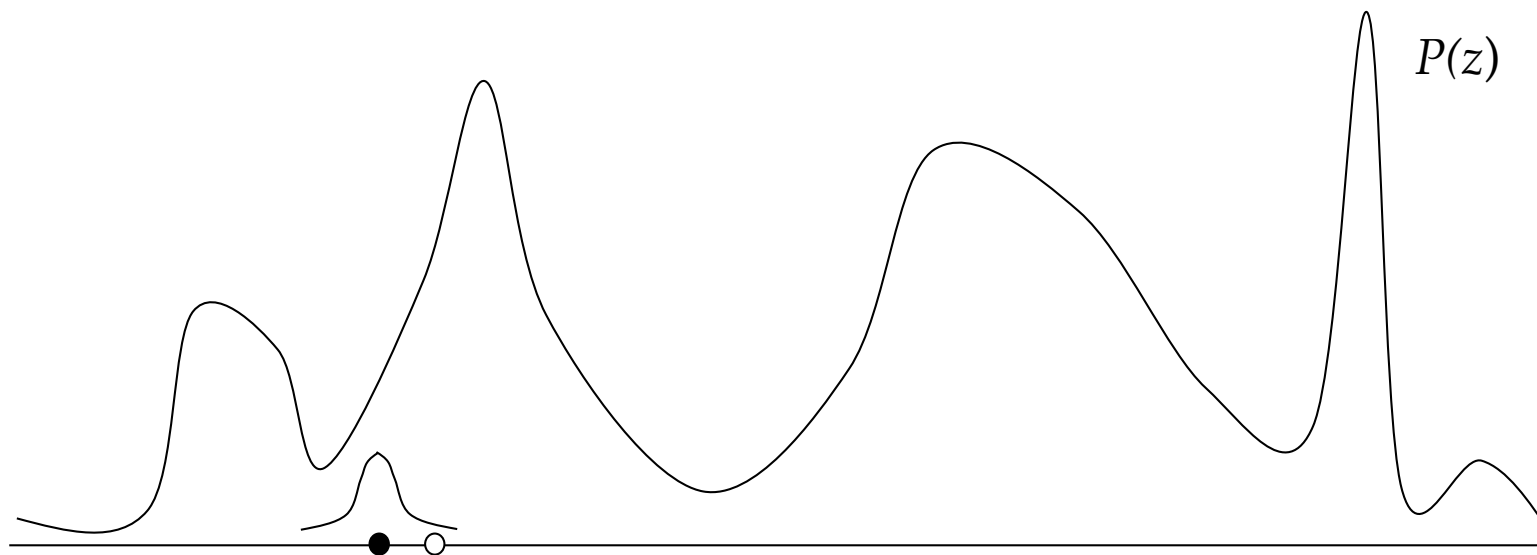
$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



**THE CANDIDATE z^* IS HALF AS
LIKELY AS z_{t-1} SO THE ACCEPTANCE
PROBABILITY IS 0.5**

So, how does the Metropolis algorithm work?

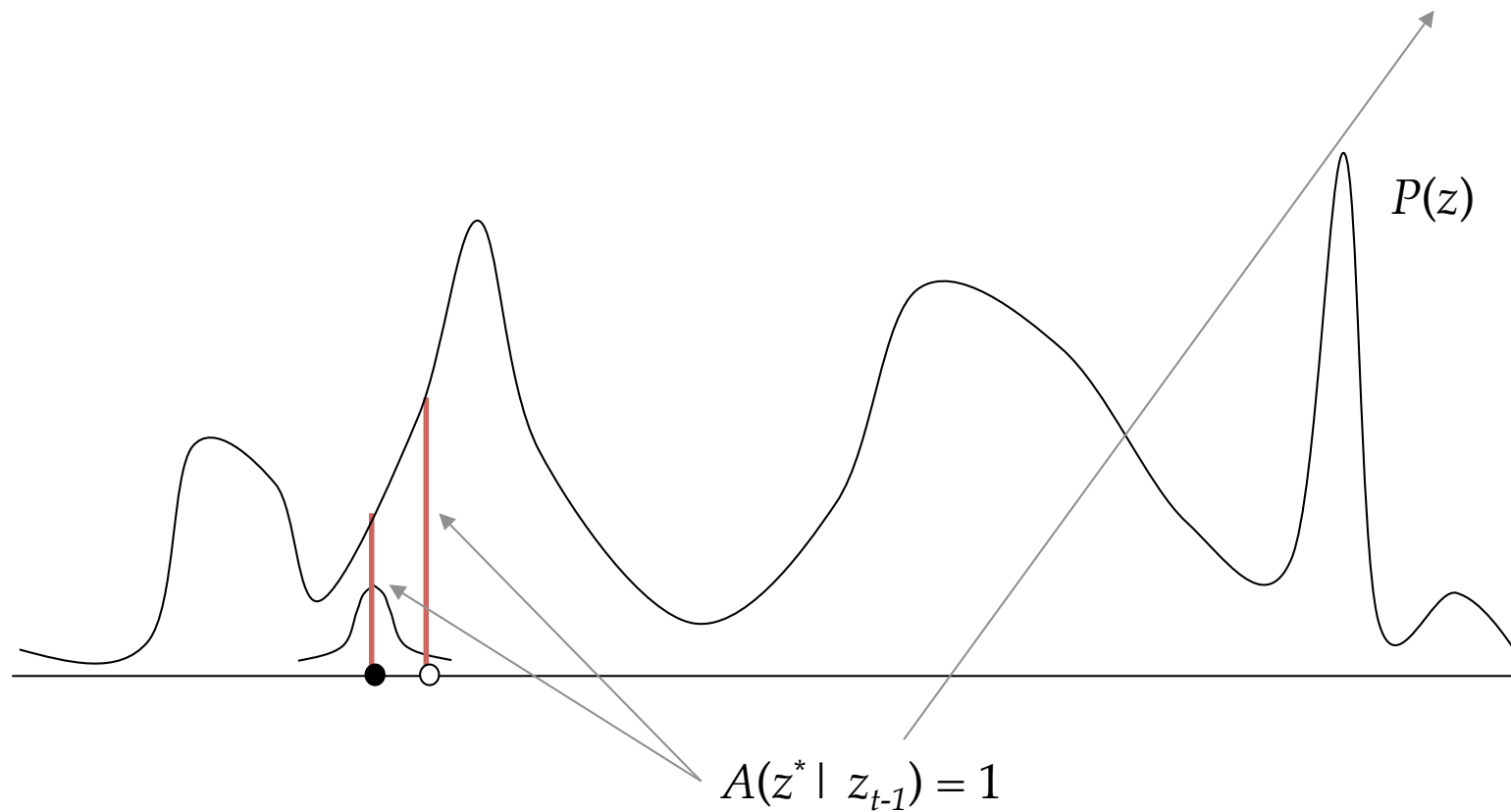
$$A(z^*|z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



**WHAT HAPPENS IF THE
CANDIDATE IS MORE LIKELY?**

So, how does the Metropolis algorithm work?

$$A(z^* | z_{t-1}) = \min \left(1, \frac{P(z^*)}{P(z_{t-1})} \right)$$



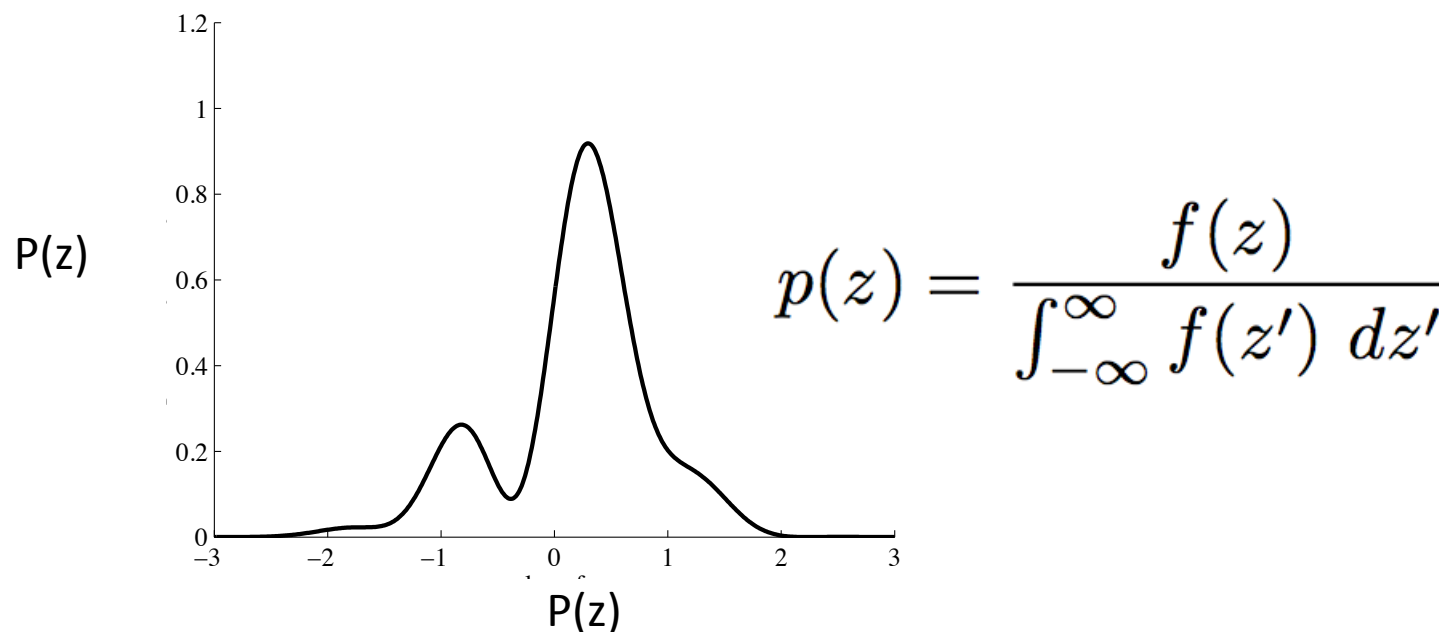
ACCEPT WITH PROBABILITY 1

The Metropolis algorithm is actually very simple

- If the candidate is an “uphill” move, always accept
- If it’s a “downhill” move, sometimes accept (based on exactly how far downhill it is).

Try it yourself.

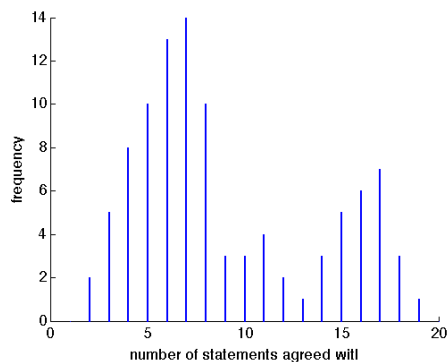
- Construct a Metropolis algorithm to sample from this distribution (see tech note if stuck)



$$f(z) = \exp(-z^2)(2 + \sin(5z) + \sin(2z))$$

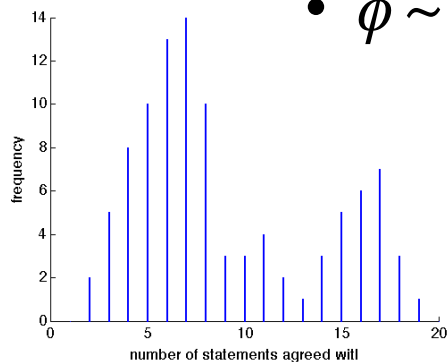
Application to the “AI survey data”

- Reminder:
 - Data are generated from a mixture of two binomials, one with parameter θ_0 and the other θ_1 . Total proportion of the data from θ_0 is ϕ .



Application to the “AI survey data”

- Reminder:
 - Data are generated from a mixture of two binomials, one with parameter θ_0 and the other θ_1 . Total proportion of the data from θ_0 is ϕ .
 - My prior is:
 - $\theta_1 \sim \text{beta}(5,50)$ (the humans don't believe in AI)
 - $\theta_0 \sim \text{beta}(50,5)$ (the robots do believe in AI)
 - $\phi \sim \text{beta}(2,20)$ (more humans than robots in the survey)



Application to the “AI survey data”

- Reminder:

- Data

- bino

- Total

- My p

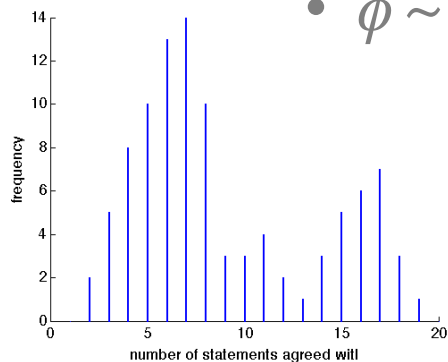
- θ_1

- $\theta_0 \sim \text{beta}(50,5)$ (the robots do believe in AI)

- $\phi \sim \text{beta}(2,20)$ (more humans than robots in the survey)

Our Metropolis sampler will use a proposal distribution that is a normal distribution with mean located at the current value and std. dev. 0.05. It will only resample one parameter at a time...

er θ_1 .



METROPOLIS ALGORITHM FOR THE “AI SURVEY” DATA

set $t = 0$.

choose value for σ^2

choose initial value $\theta_{00}, \theta_{10}, \phi_0$ arbitrarily

separate Gaussian-proposals for each parameter

do while $t < \infty$

 increment: $t = t + 1$

$\theta_{0t} = \text{resample}(\theta_{0,t-1}, \sigma^2)$

$\theta_{1t} = \text{resample}(\theta_{1,t-1}, \sigma^2)$

$\phi_t = \text{resample}(\phi_{t-1}, \sigma^2)$

as $t \rightarrow \infty$, the “state” of the algorithm converges $x_t \sim P(\cdot)$

$z_t = \text{resample}(z_{t-1}, \sigma^2)$

generate a “candidate” $z^* \sim \text{normal}(z_{t-1}, \sigma^2)$

calculate the “acceptance probability”, $A(z^* | z_{t-1})$

generate $u \sim U([0,1])$ from a uniform distribution

if $u \leq A$, accept the candidate: set $z_t = z^*$

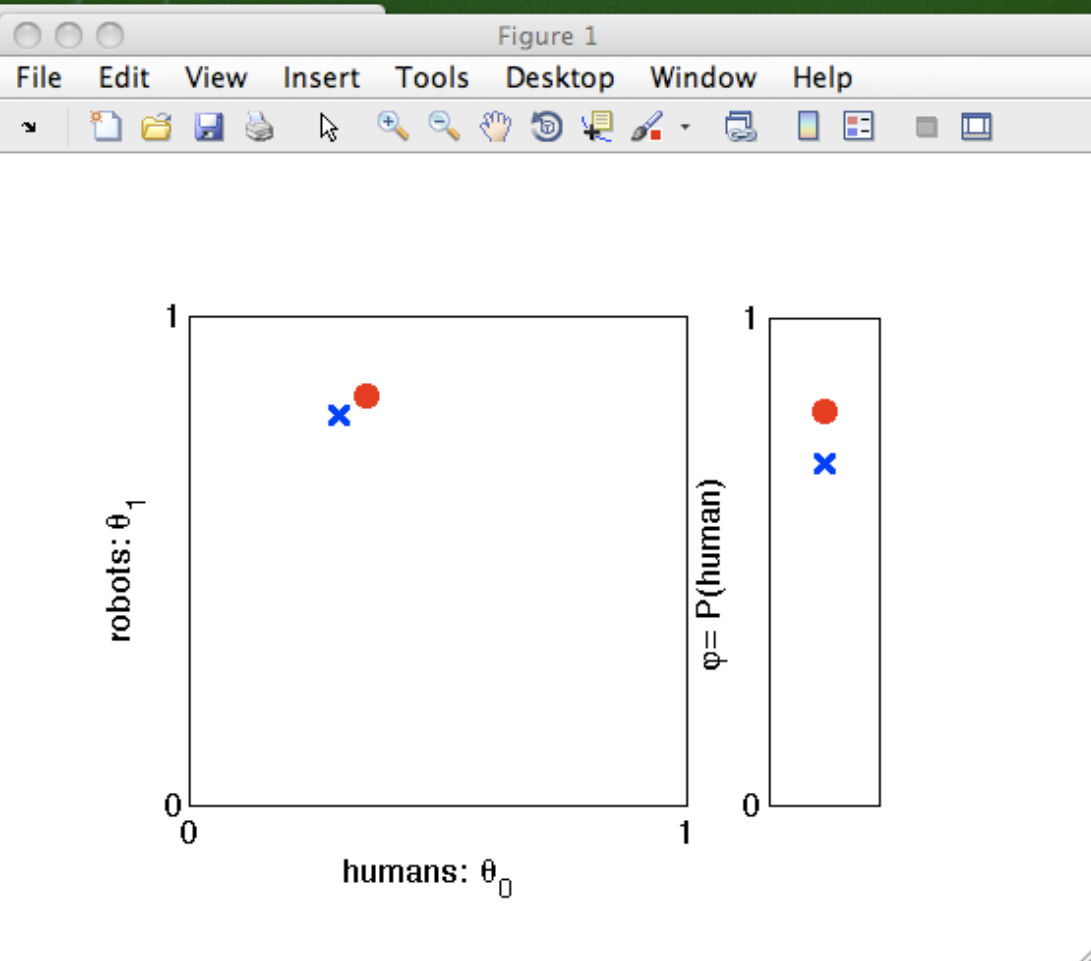
elseif $u > A$, reject the candidate: set $z_t = z_{t-1}$

```

1 function metropolisdemo
2
3     % load the AI data
4     load aidata; M=m1+m2;
5
6     % reformat the AI data: The raw data lists
7     % agreements n for each participant. Since
8     % value of n is represented at least once,
9     % efficient to reformat as a vector of counts
10    % counts the number of participants who agree
11    % statements (remember, Matlab indexing starts at 1)
12    x=zeros(N+1,1);
13    for i=1:M
14        n=numberofagreements(i);
15        x(n+1)=x(n+1)+1;
16    end
17
18    % specify parameters for the prior
19    a_th0 = 5; b_th0 = 50;
20    a_th1 = 50; b_th1 = 5;
21    a_phi = 20; b_phi = 2;
22
23    % initialise variables randomly
24    th0=rand;
25    th1=rand;
26    phi=rand;
27    [h_th,h_phi]=setupplots; % set up figure
28
29    % iterate sampler
30    nits=1000;
31    sig2=.05;
32    for t=1:nits
33
34        % redraw parameters
35        th0=redraw(1,[th0 th1 phi],sig2,a_th0,b_th0,x);
36        th1=redraw(2,[th0 th1 phi],sig2,a_th1,b_th1,x);
37        phi=redraw(3,[th0 th1 phi],sig2,a_phi,b_phi,x);
38
39        % draw me a pretty picture
40        disp([th0 th1 phi])
41        set(h_th,'xdata',th0,'ydata',th1); % update theta location
42        set(h_phi,'ydata',phi); % update theta location
43        pause(.01)
44    end
45
46
47

```

RUN DEMO



Shortcuts	How to Add	What's New
0.3547	0.8443	0.7990
0.3547	0.8135	0.7945
0.3547	0.8412	0.7756
0.3547	0.8412	0.7765
0.3547	0.8412	0.7833
0.3547	0.8412	0.8058

>>

Start 138

This is a good spot to ask questions...

(e.g. “What is the relationship between Metropolis-Hastings and simulated annealing?”)

Notes on MCMC convergence

From the tech note....

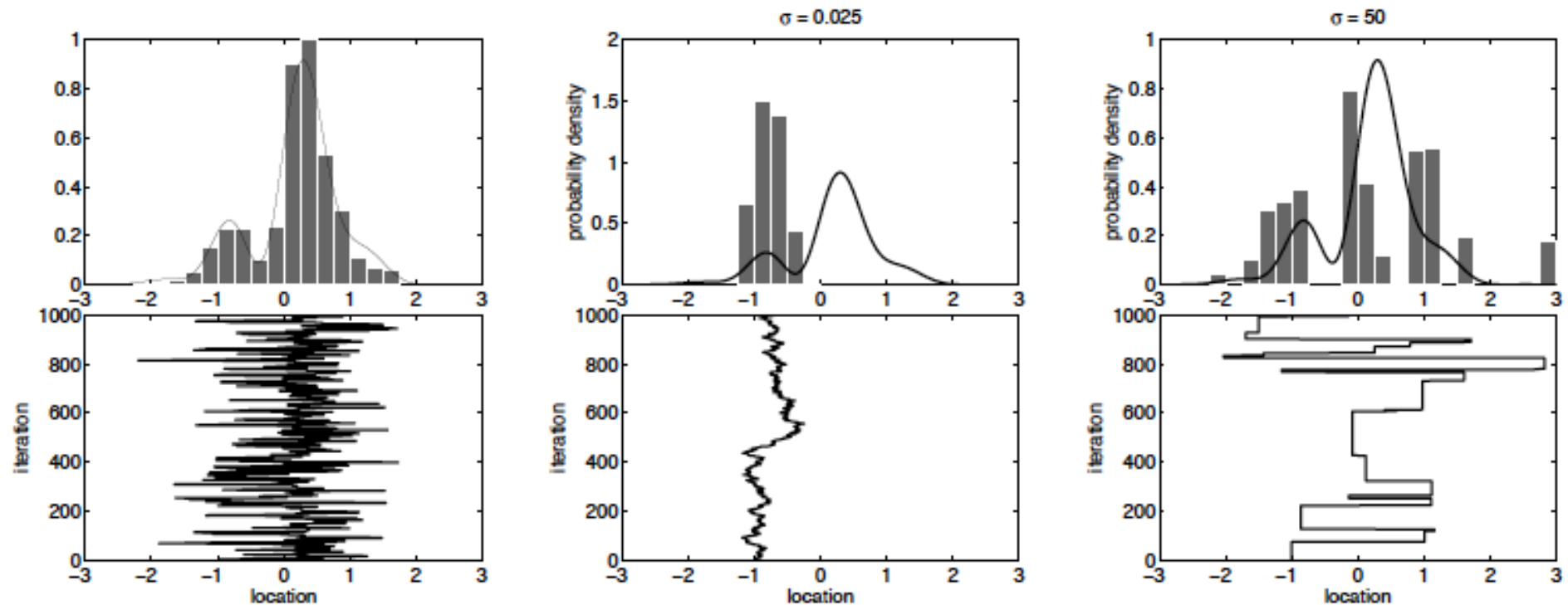
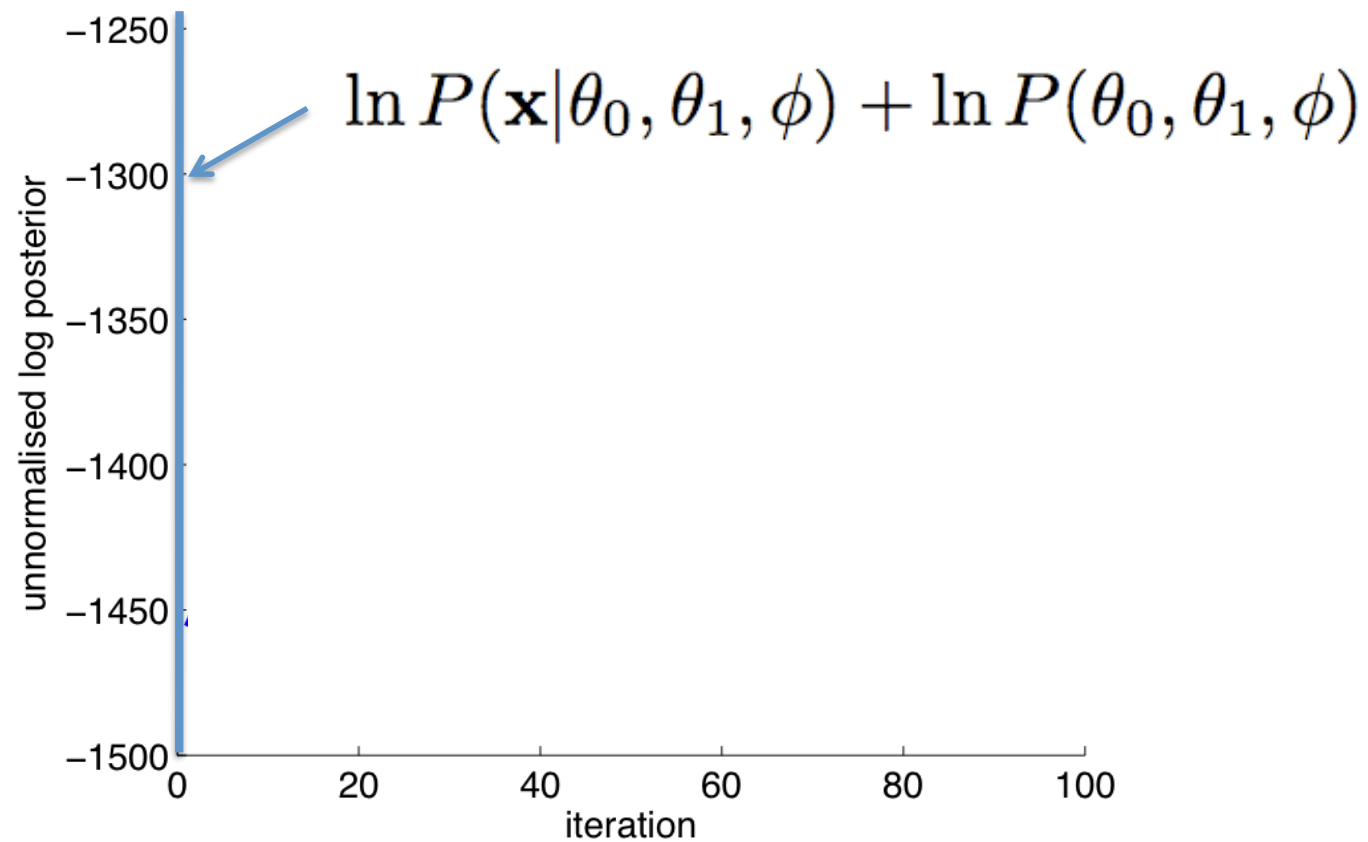
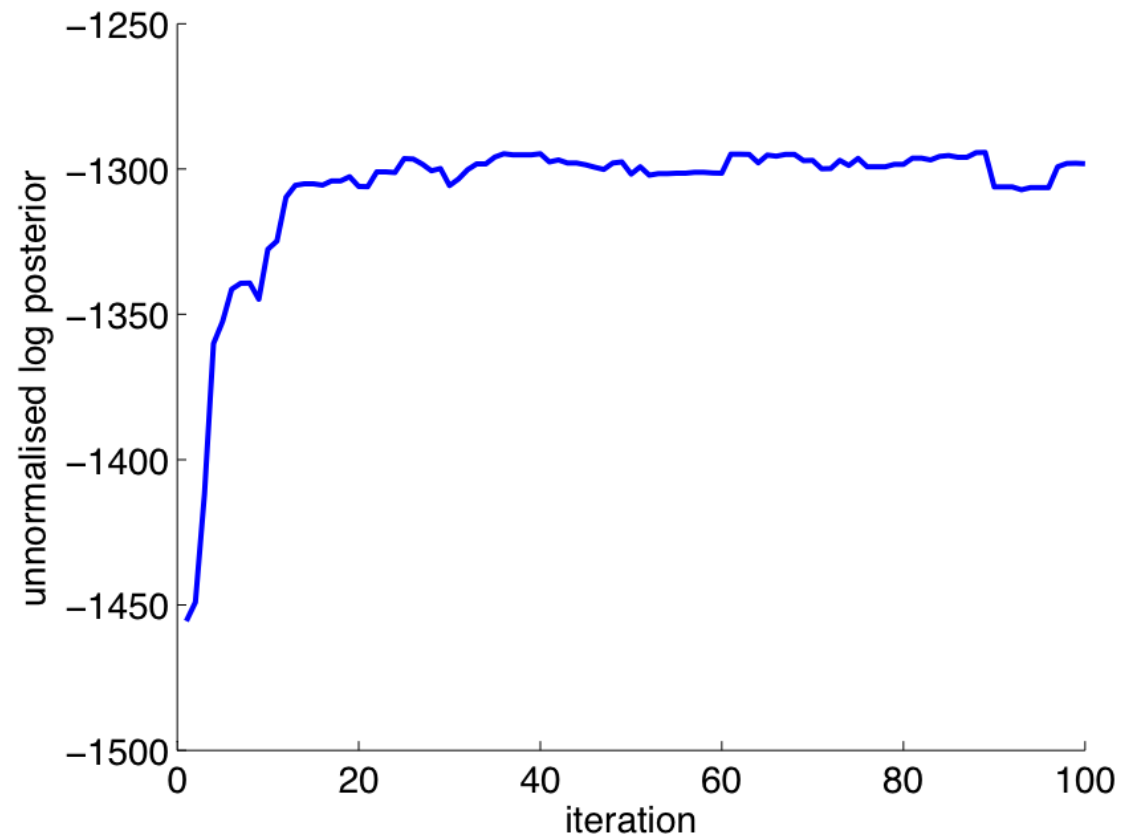


Figure 3. Metropolis samplers for the toy problem, with a good choice of proposal distribution ($\sigma = 1$, left panel), a proposal distribution that is too narrow ($\sigma = .025$, middle panel), and a proposal distribution that is too wide ($\sigma = 50$, right panel).

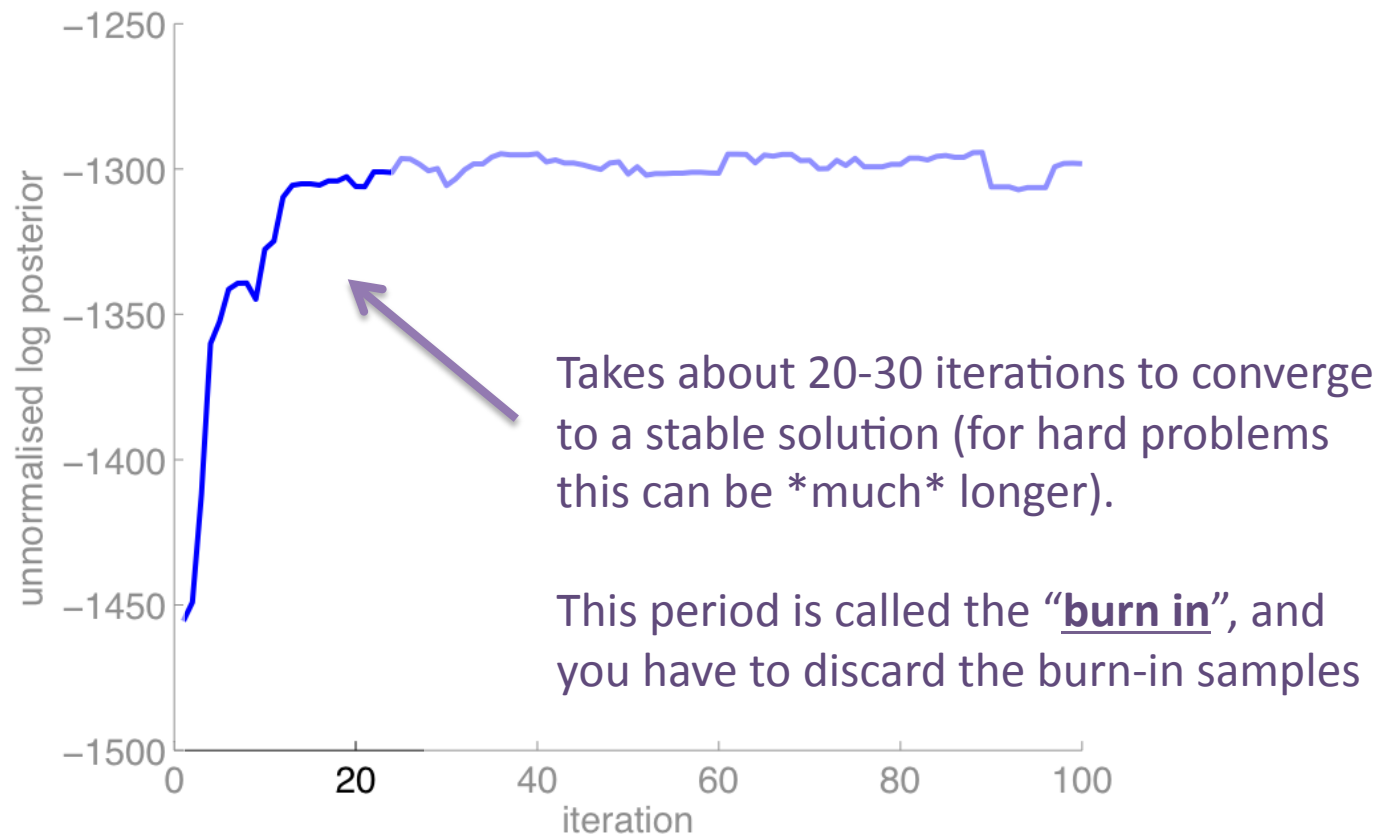
Monitoring the sampler



Monitoring the sampler

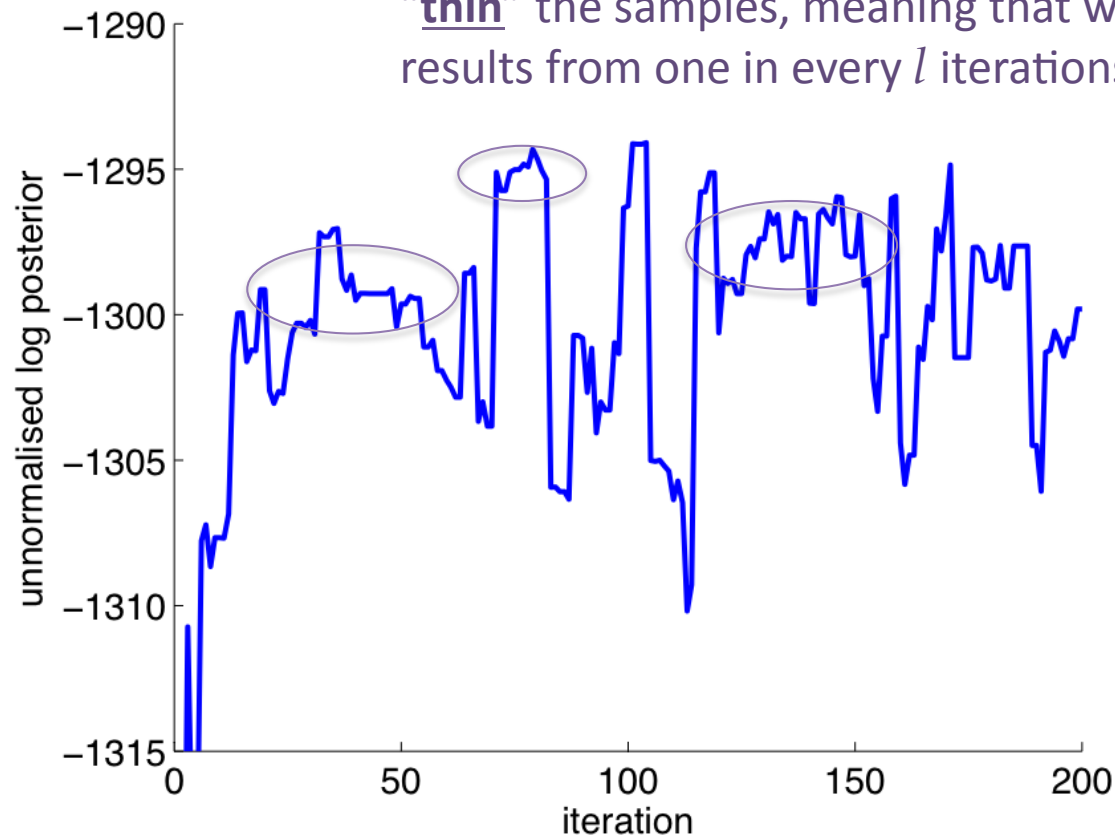


Monitoring the sampler

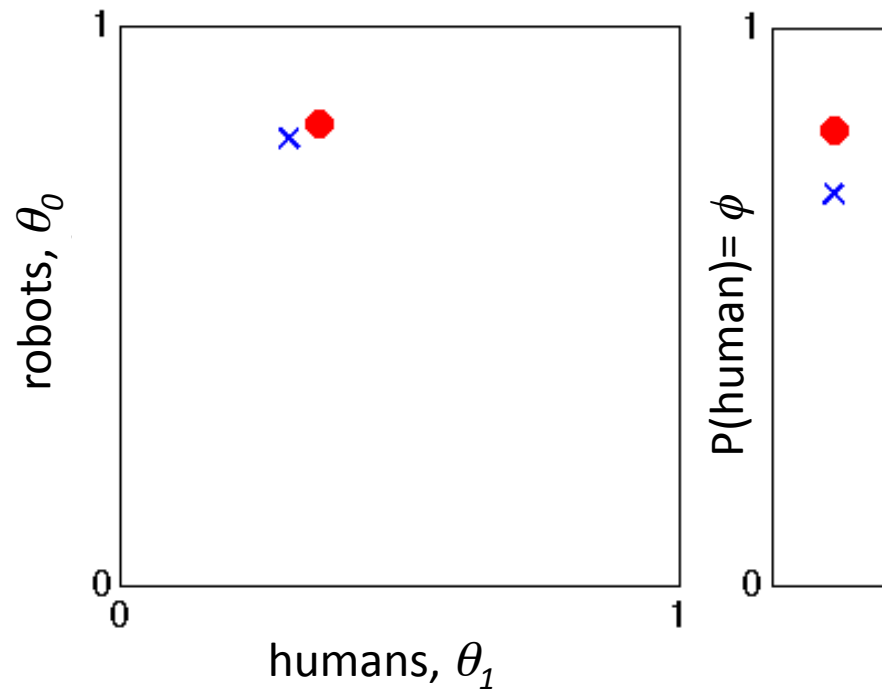


Monitoring the sampler

The samples are autocorrelated: each iteration is somewhat dependent on the last one. Typically, we “thin” the samples, meaning that we only “save” the results from one in every l iterations, where l is the lag

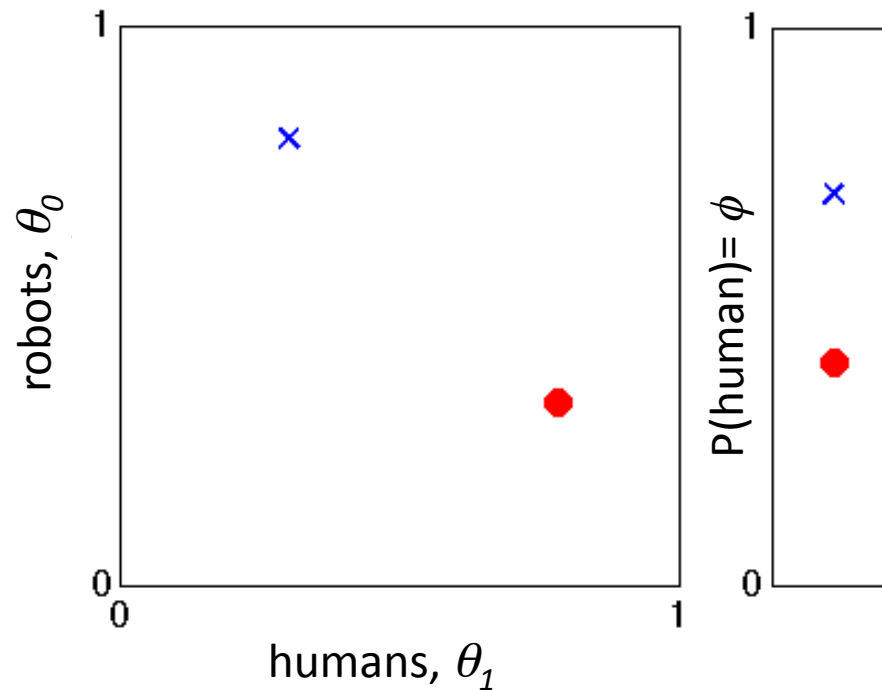


Two stable solutions

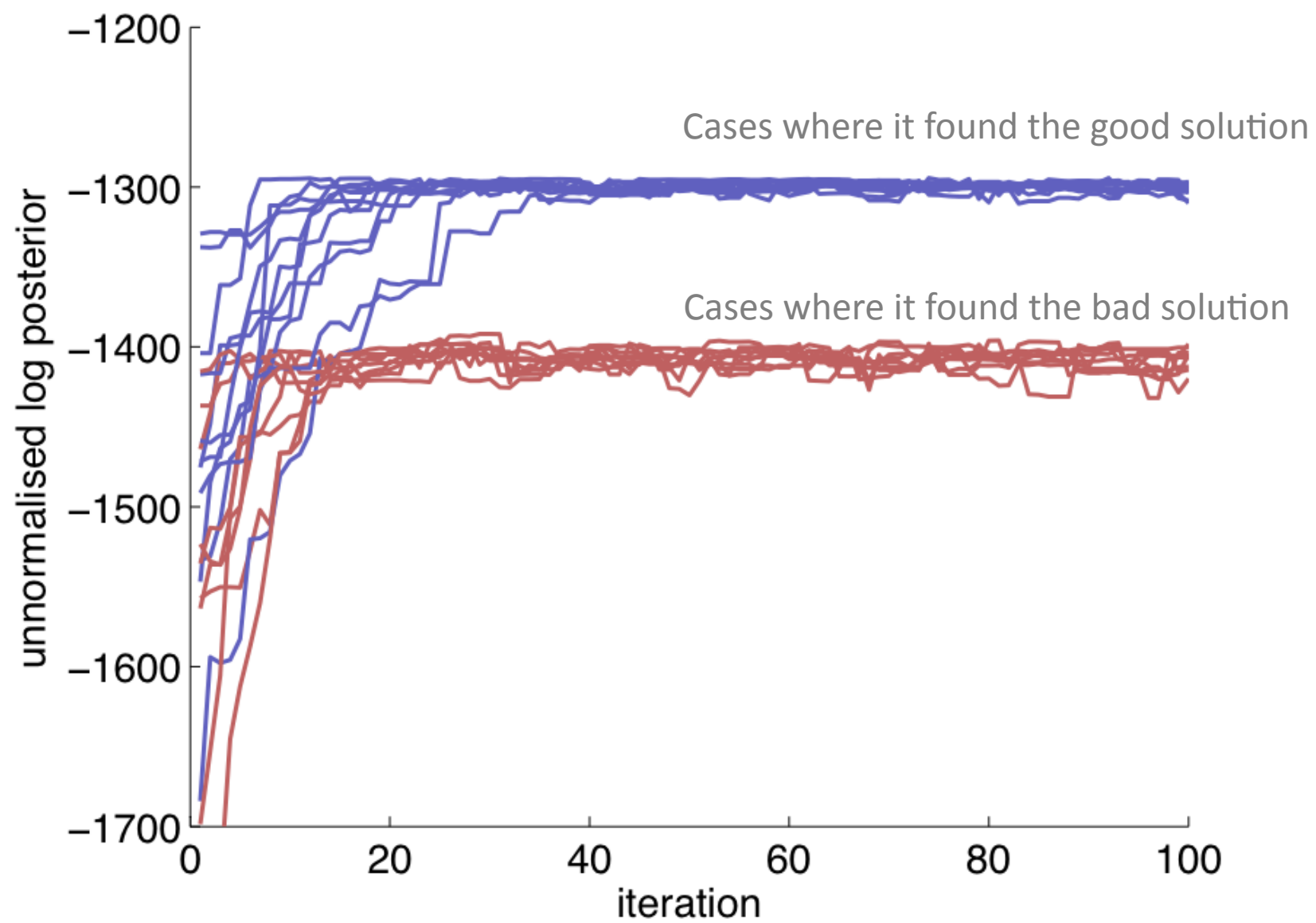


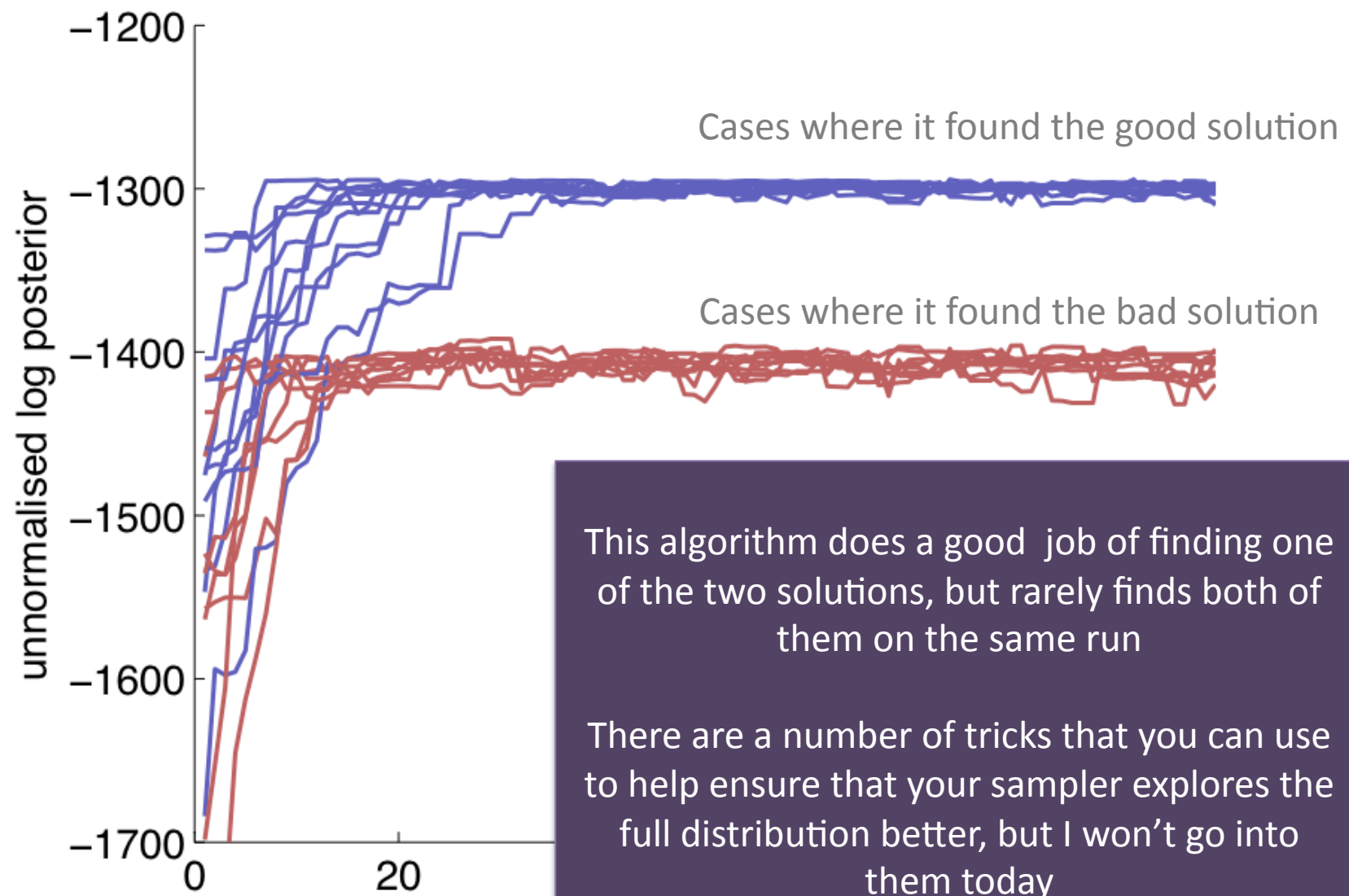
The “correct” solution: there’s a few robots, and they believe in AI.

Two stable solutions



This solution says that most of the respondents are robots, and that robots are more skeptical of strong-AI than humans (lower prior probability, but not by enough, apparently...)

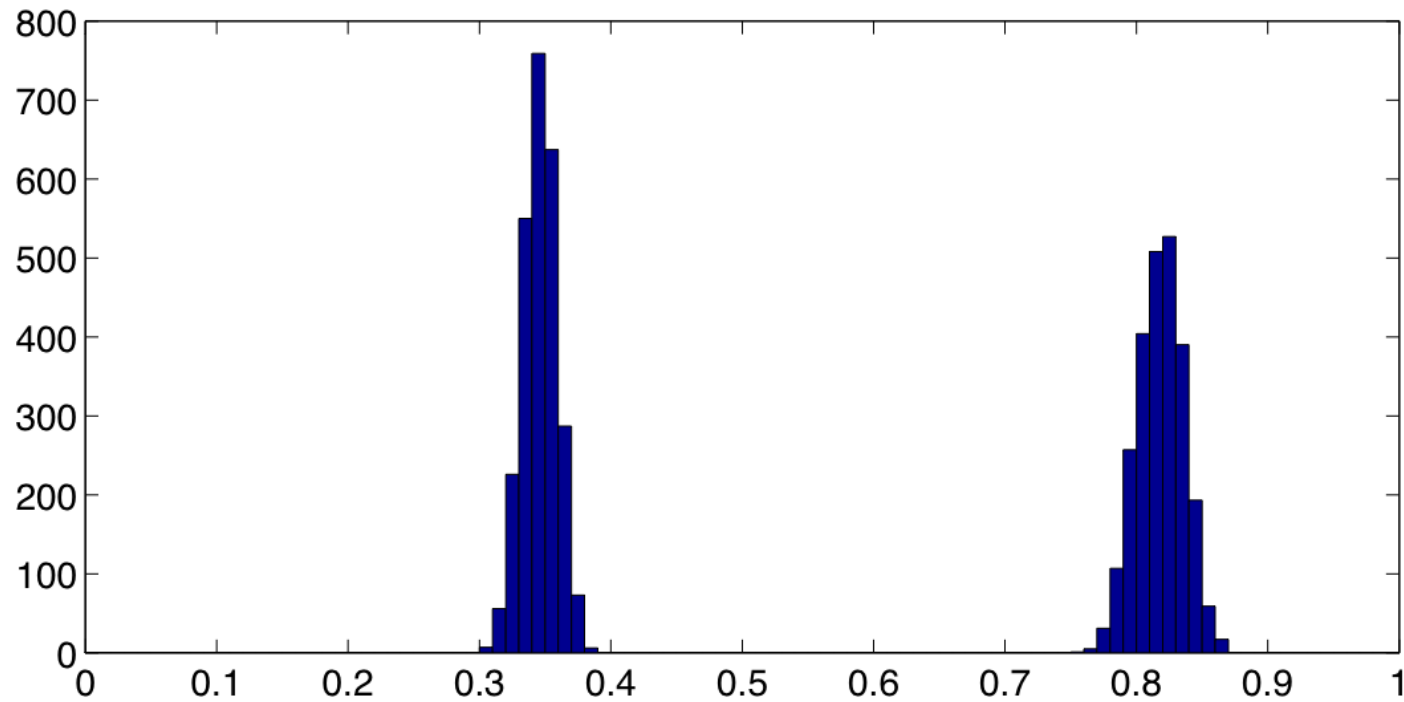




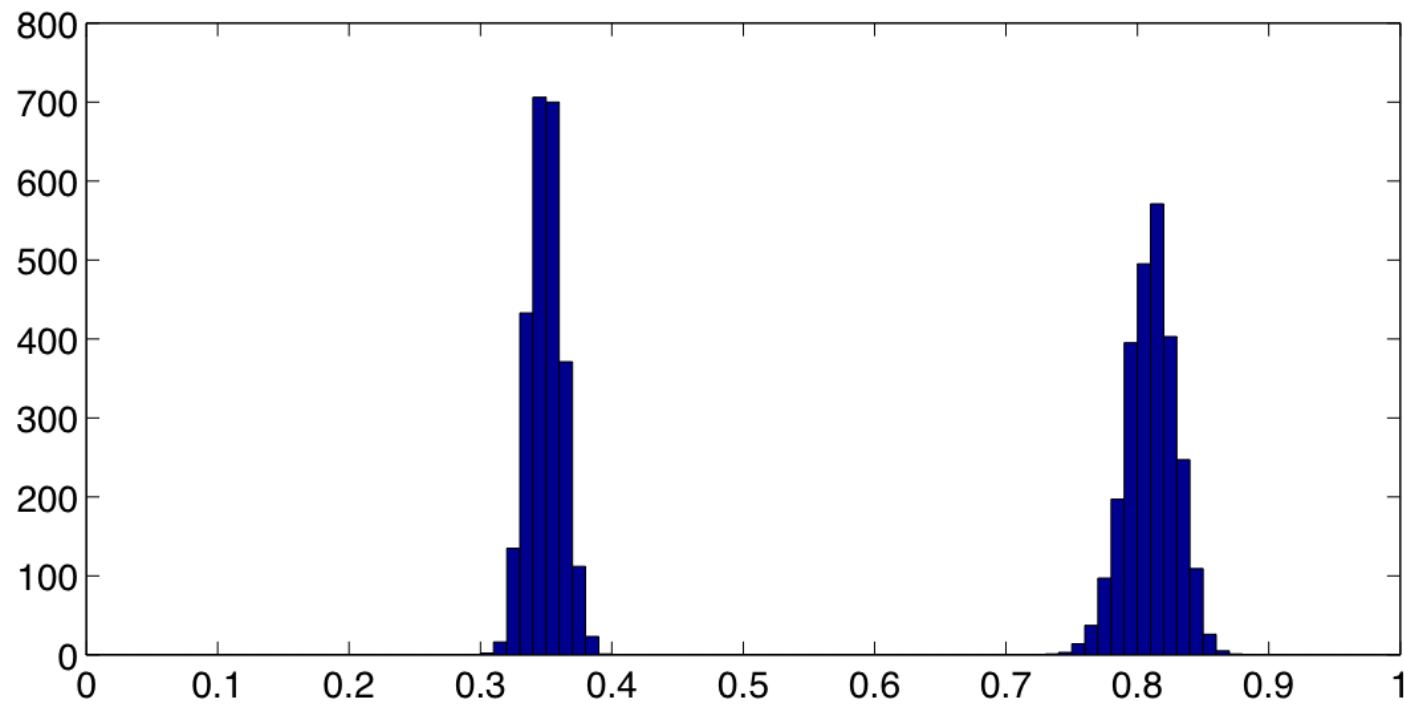
Try it yourself.

- I've put the AI data up on *MyUni*.
- Implement your own Metropolis algorithm for solving this problem
- Try varying the variance of the proposal distribution, σ^2 . See what happens!
 - A lot of the art to MCMC is in playing around with the proposal distribution until it starts to work.

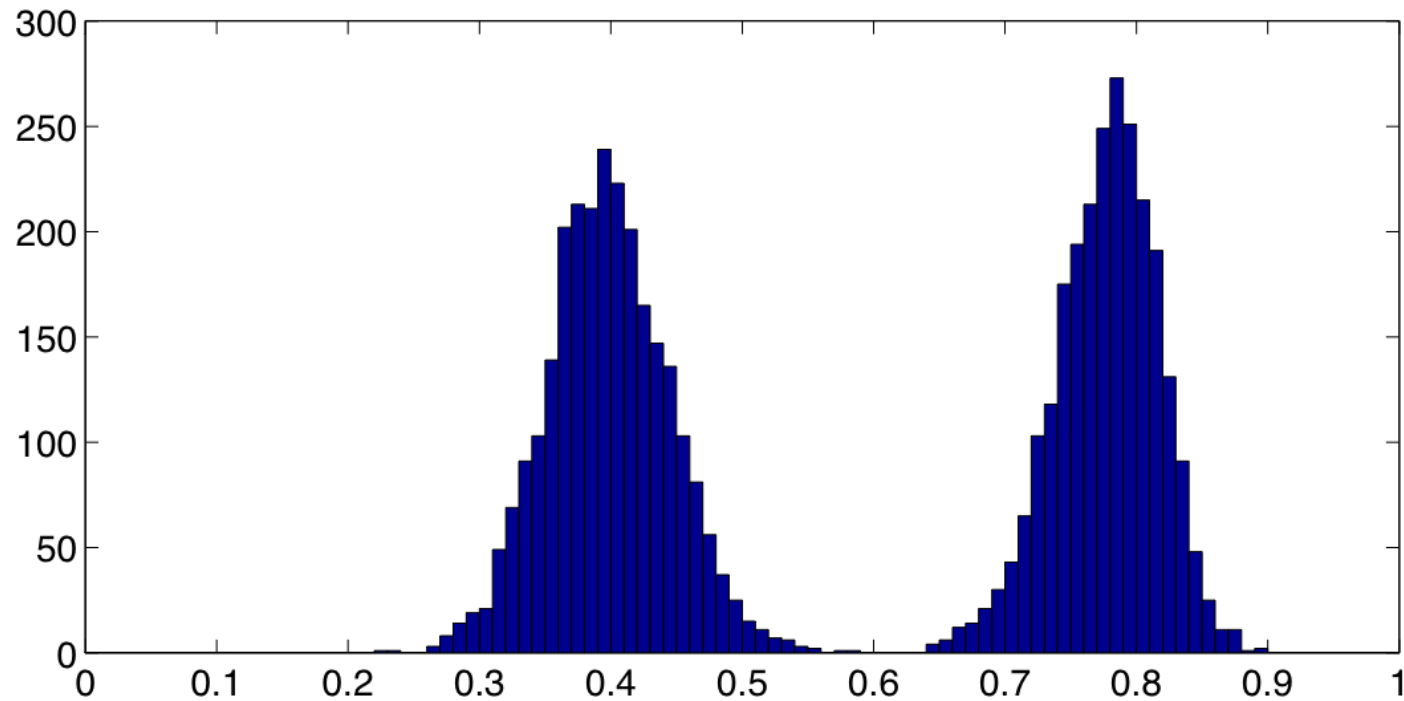
Over lots of chains... θ_0



Over lots of chains... θ_1



Over lots of chains... ϕ



51% of the solutions are “good”. But **cannot** trust this, since **none** of the chains managed to “visit” both of the peaks in the posterior distribution!!!

This is a good spot for questions.

LIKELIHOOD WEIGHTING...
ANOTHER USELESS FORM OF
IMPORTANCE SAMPLING

Likelihood weighting

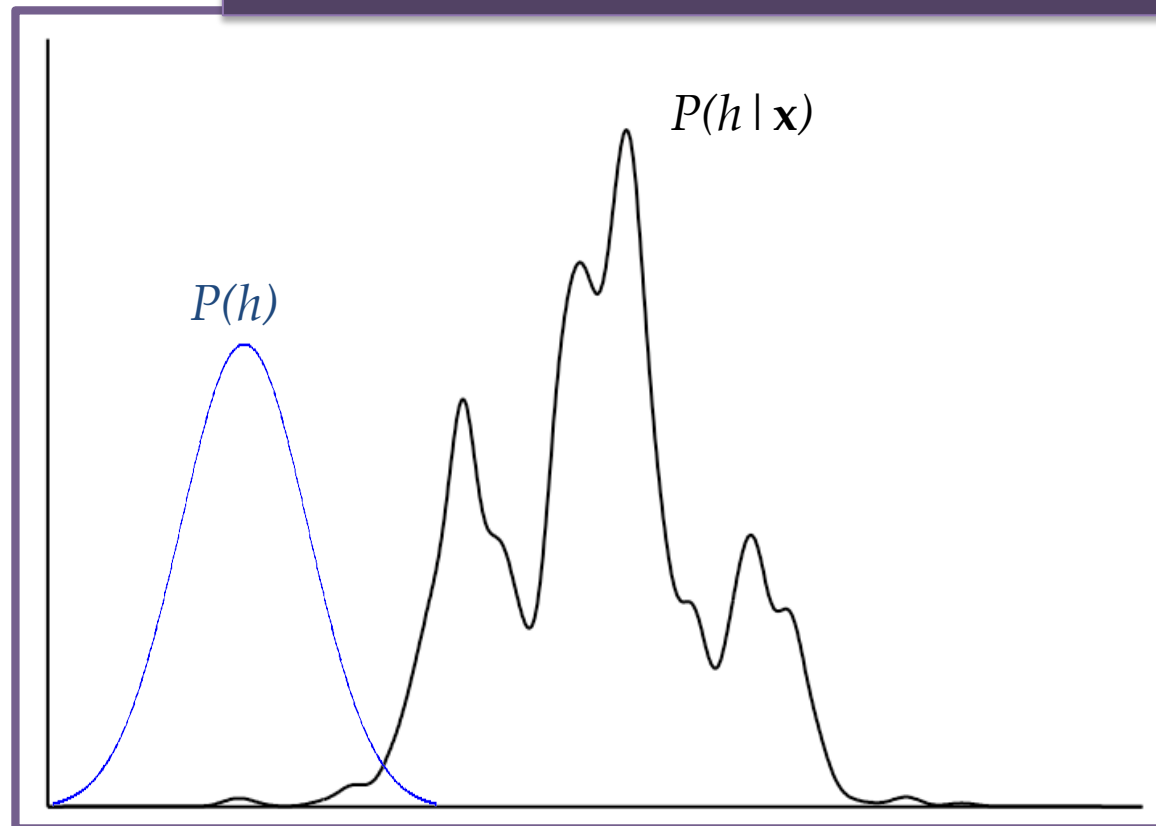
- In the Bayesian context:
 - A particularly simple form of importance sampling for posterior distributions is use the prior as the importance distribution
 - set $Q(h) = P(h)$

Likelihood weighting

- Simplifies the calculation of the weights $w(h_i)$:

$$\begin{aligned}w(h_i) &= \frac{P(x|h_i)P(h_i)}{Q(h_i)} \\&= \frac{P(x|h_i)P(h_i)}{P(h_i)} \\&= P(x|h_i)\end{aligned}$$

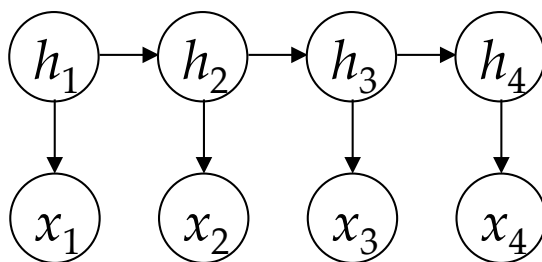
The problem (again!) is that if the prior $P(h)$ is sufficiently different from the posterior $P(h | \mathbf{x})$, then it fails



PARTICLE FILTERS

The particle filtering trick

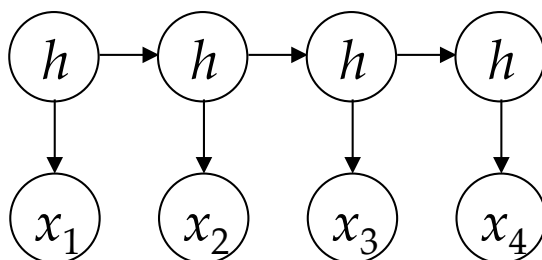
- Sequential data:



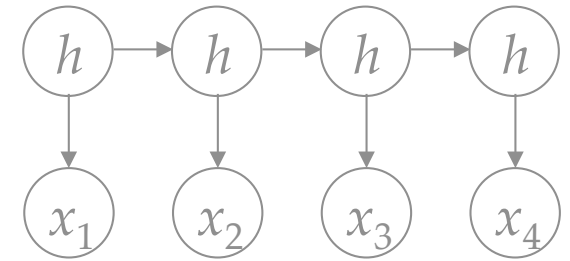
- The datum x_t is generated from hypothesis h_t
 - The new hypothesis h_{t+1} is generated from h_t (in some fashion)
- We want to sample from: $P(h_4 \mid x_1 \dots x_4)$

The particle filtering trick

- Non-sequential data:

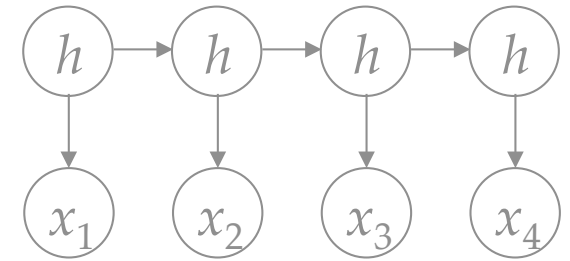


- Corresponds to the special case where $h_{t+1} = h_t$
 - I'll start with this special case because it's simple to describe
 - But in practice, it's actually the hardest situation to make the particle filter work!
- We want to sample from: $P(h \mid x_1 \dots x_4)$



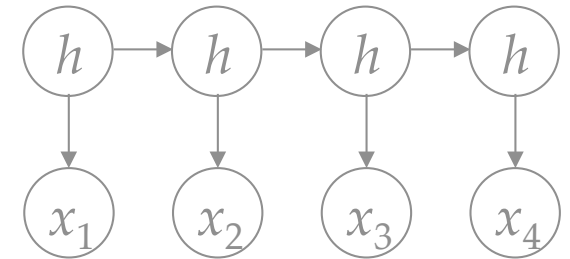
$$P(h|x_1 \dots x_t) \propto P(x_t|h) \underbrace{P(h|x_1, \dots, x_{t-1})}$$

Let's suppose, for the moment, that we actually do know how to sample from this

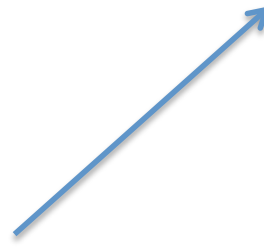


$$\underline{P(h|x_1 \dots x_t)} \propto P(x_t|h)P(h|x_1, \dots, x_{t-1})$$

If so, how would we
sample from this?



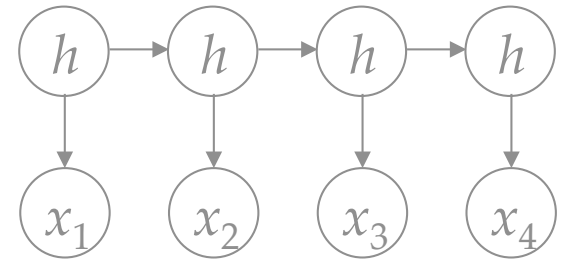
$$P(h|x_1 \dots x_t) \propto \underline{P(x_t|h)P(h|x_1, \dots, x_{t-1})}$$



Answer: importance sampling, with likelihood weighting!

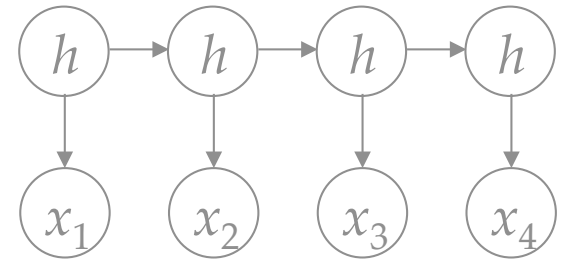
That is: generate lots of h_i values from $P(h|x_1, \dots, x_{t-1})$, then assign each one a weight proportional to $P(x_t|h)$. Recall, the weight of the i -th such “particle” is:

$$w_i = \frac{P(x_t|h_i)}{\sum_{j=1}^n P(x_t|h_j)}$$



$$P(h|x_1 \dots x_t) \propto P(x_t|h) \underline{P(h|x_1, \dots, x_{t-1})}$$

So we have a recursion: if we can
sample from this...



$$\underline{P(h|x_1 \dots x_t)} \propto P(x_t|h)P(h|x_1, \dots, x_{t-1})$$

... then we can sample from this

PARTICLE FILTERING

for $i = 1 : n$

$$h_{i,0} \sim P(h)$$

$$w_i = 1 / n$$

set $\mathbf{A}_0 = (\mathbf{w}, \mathbf{h}_0)$

for $t = 1 : m$

set $v = 0$

for $i = 1 : n$

$$w_i = P(x_t \mid h_{i,t-1})$$

$$v = v + w_i$$

for $i = 1 : n$

$$w_i = w_i / v$$

set $\mathbf{A}_t = (\mathbf{w}, \mathbf{h}_{t-1})$

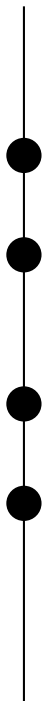
for $i = 1 : n$

sample $h_{i,t} \sim \mathbf{A}_t$

$$\text{set } w_i = 1 / n$$

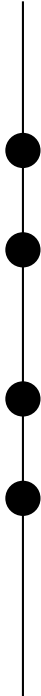
set $\mathbf{A}_t = (\mathbf{w}, \mathbf{h}_t)$

Accurate, but not helpful!

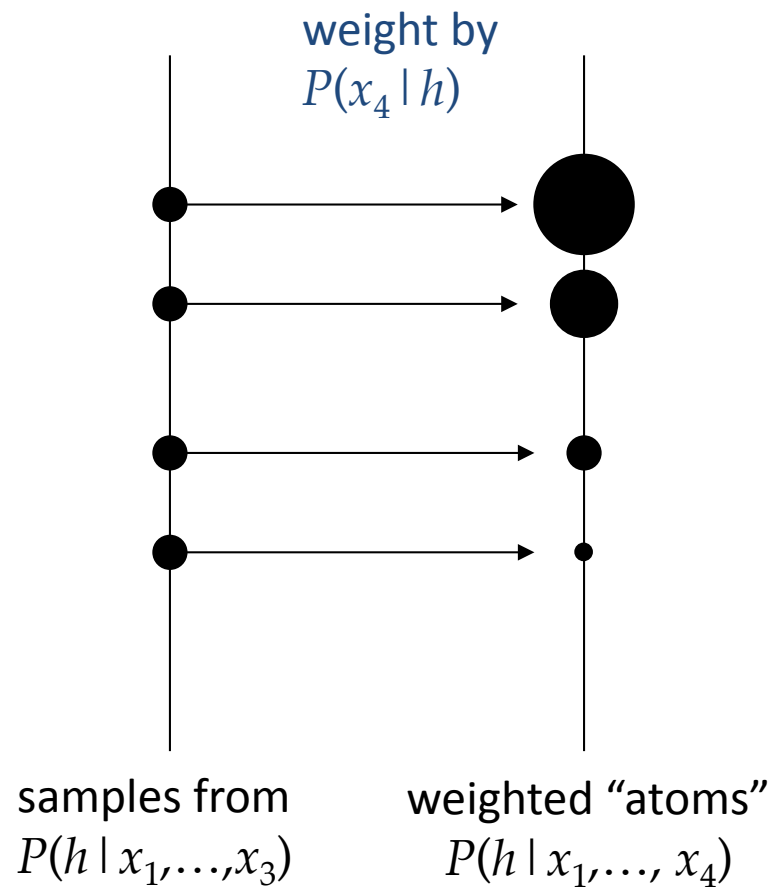


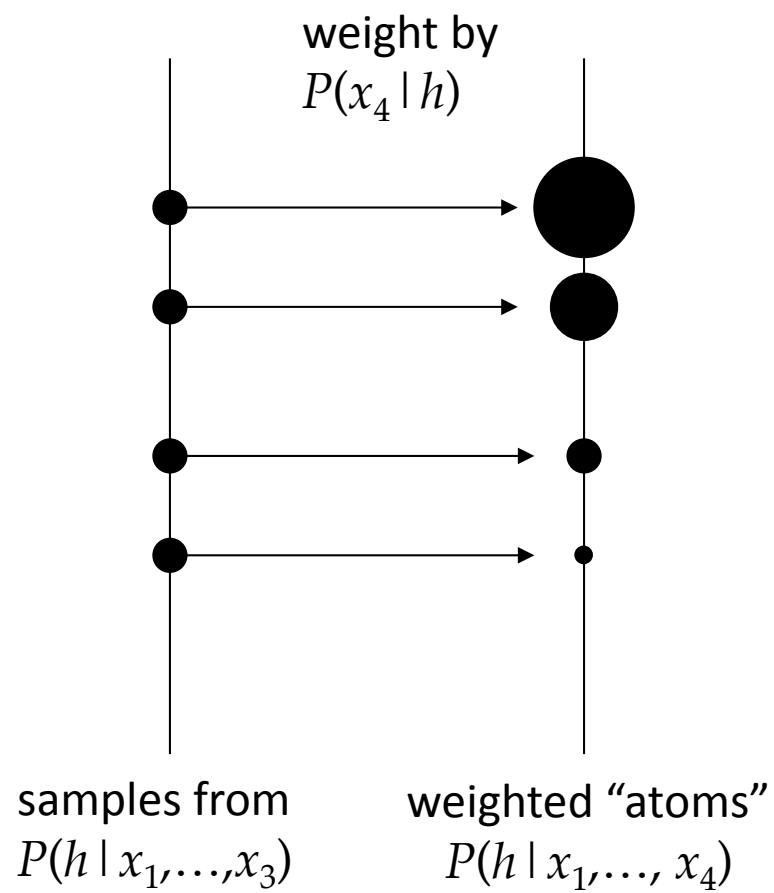
samples from
 $P(h \mid x_1, \dots, x_3)$

What do we do when x_4 arrives?

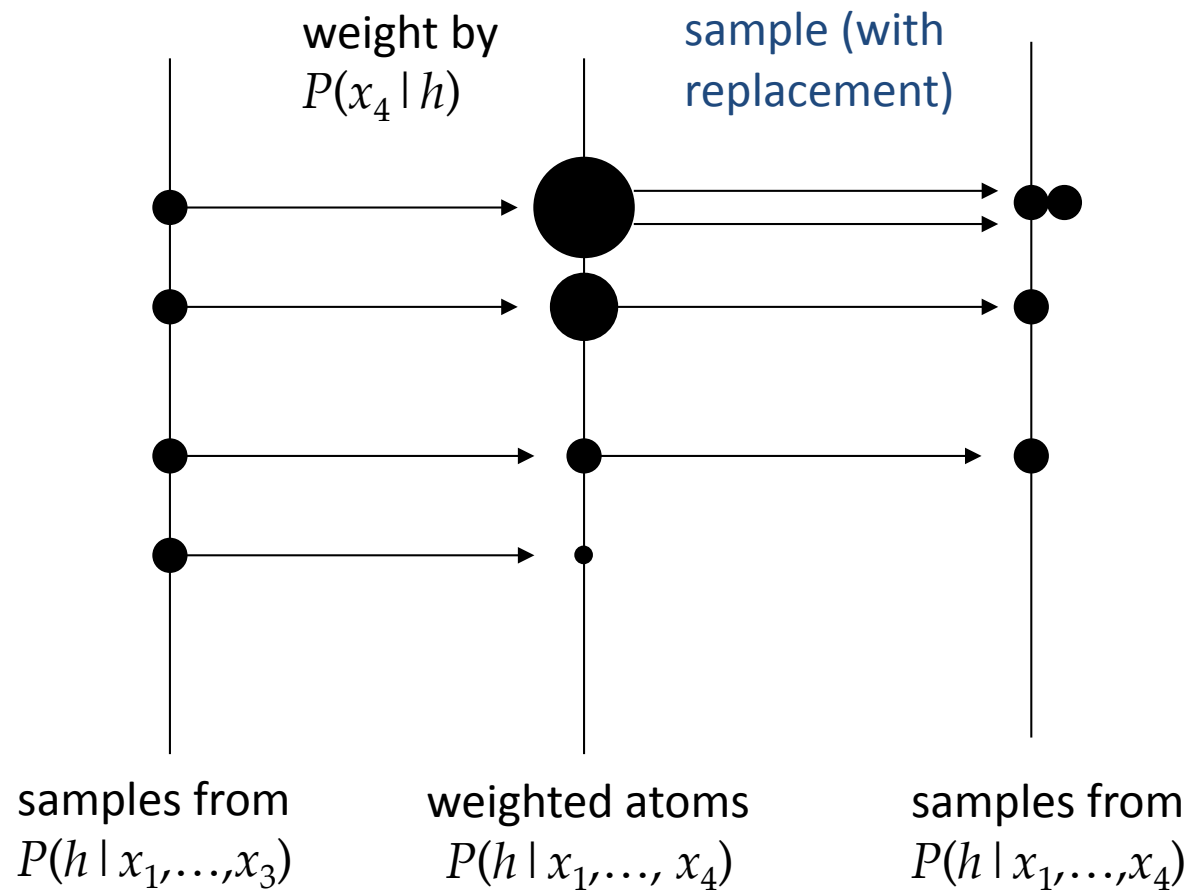


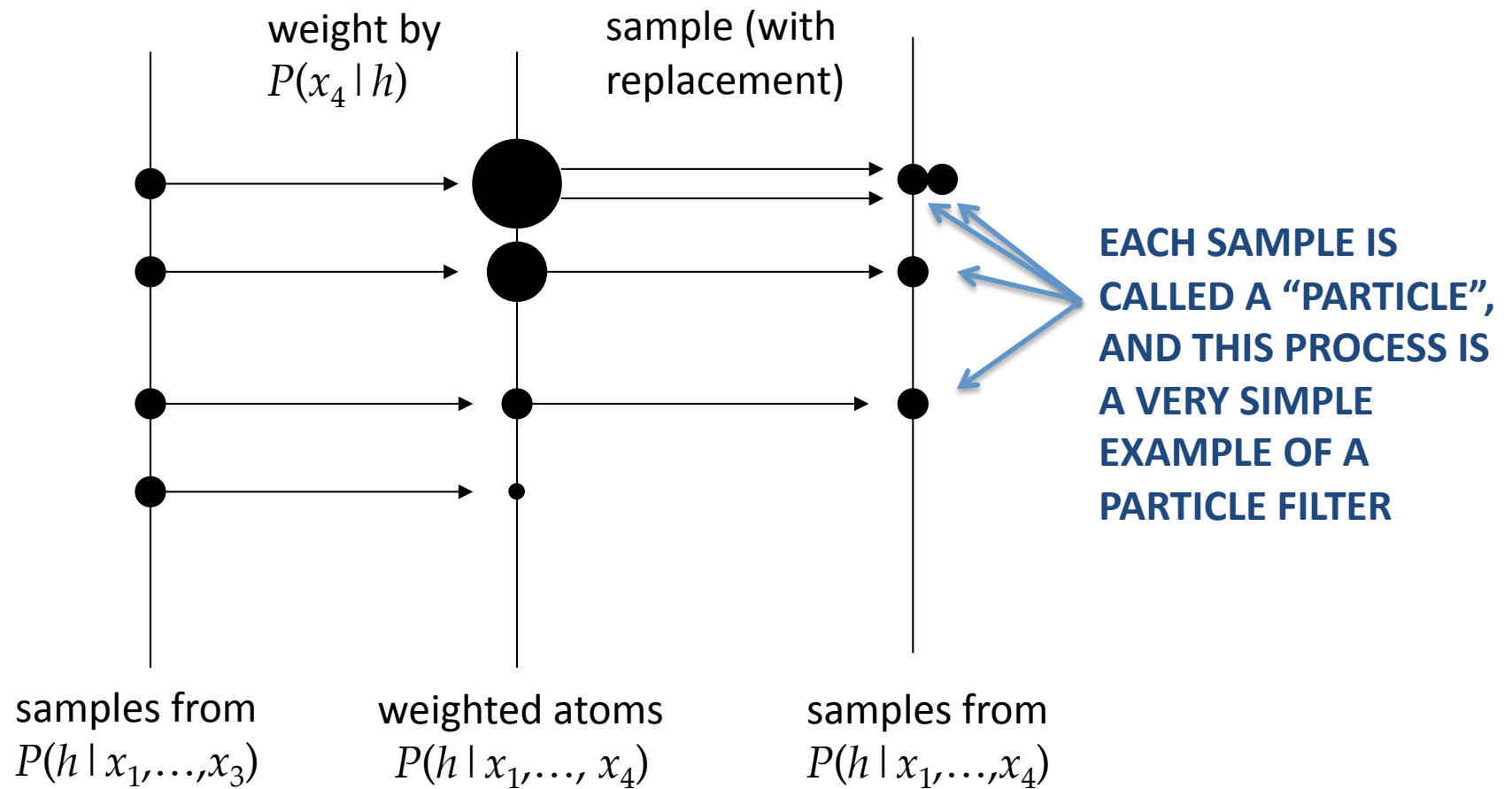
samples from
 $P(h \mid x_1, \dots, x_3)$





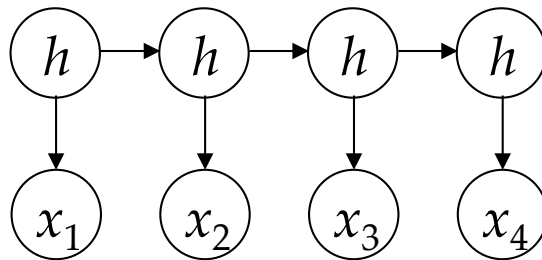
How do we turn these "atoms"
into a proper sample from P
 $(h | x_1, \dots, x_4)$?

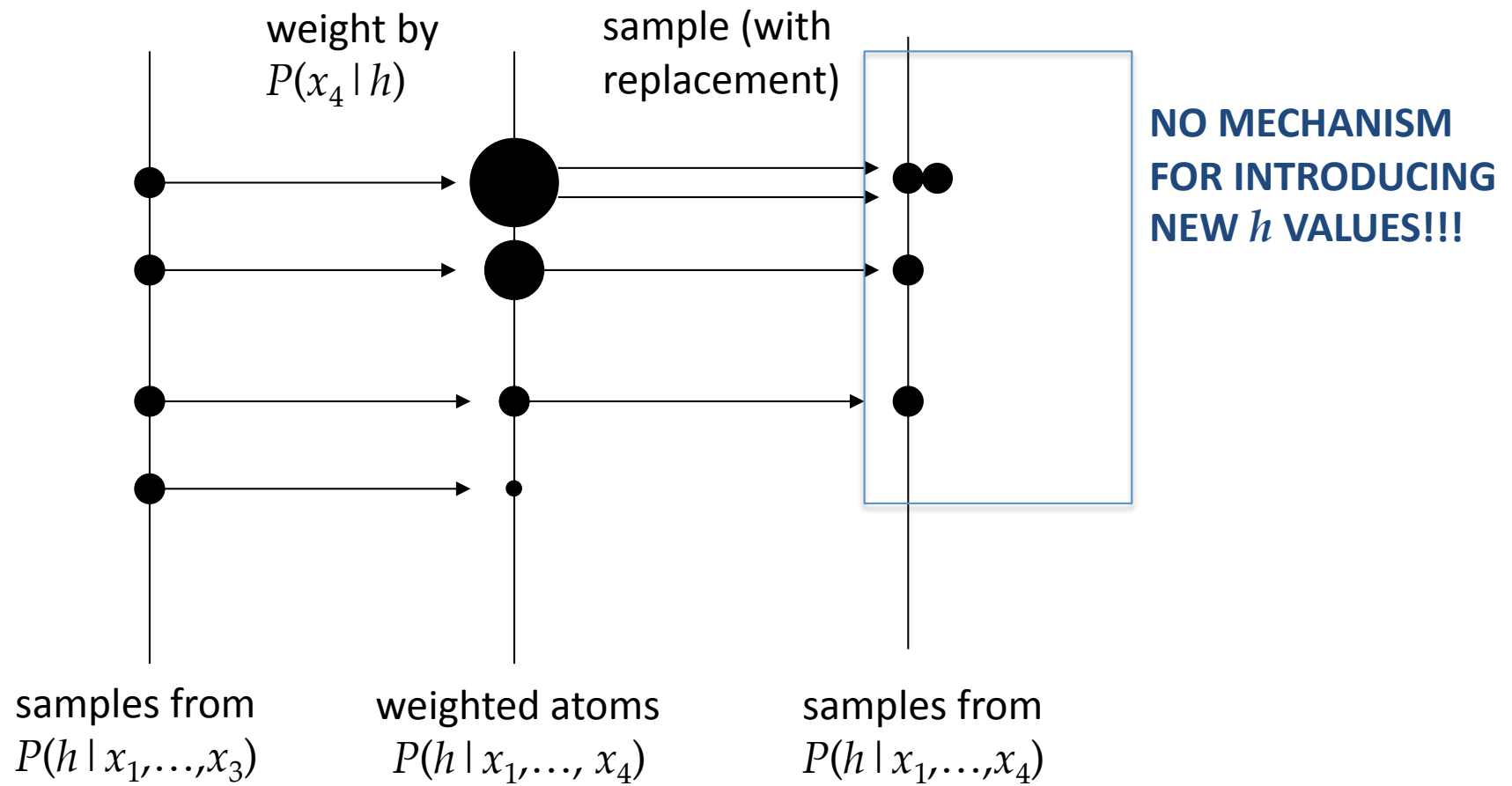




A problem...

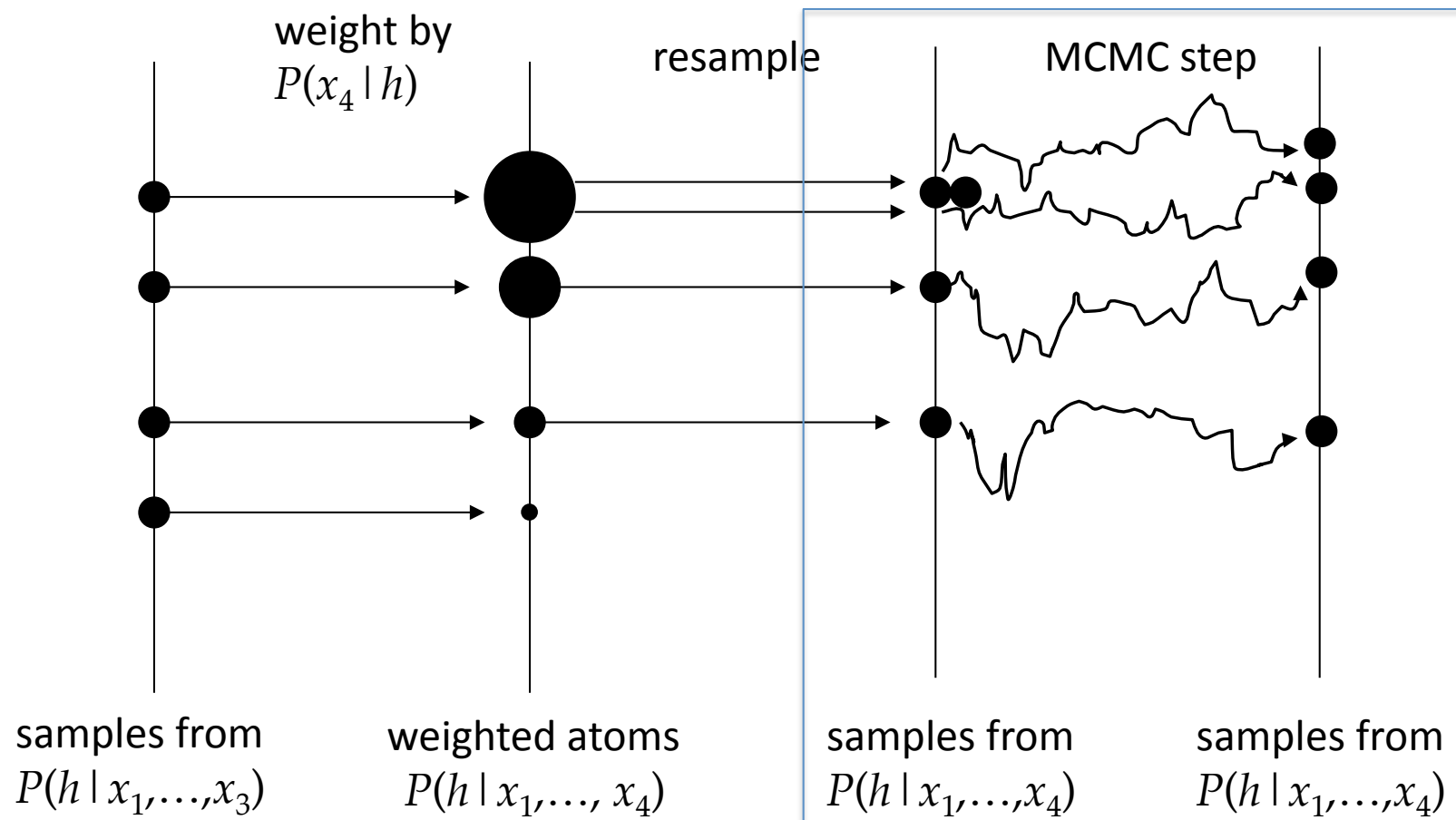
- If hypothesis h wasn't present at the very beginning – in the initial draw from $P(h)$ – it can't ever appear in the approximation to $P(h \mid x_1 \dots x_t)$





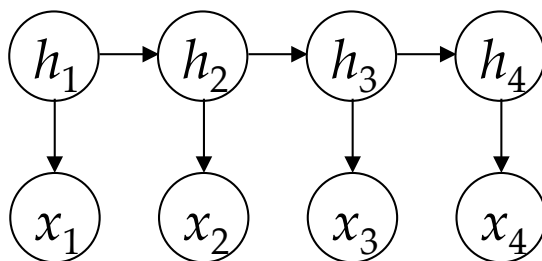
Solution #1

- Run a few iterations of an MCMC sampler on the particles every now and then!



Dynamic models avoid the problem

- Reminder:

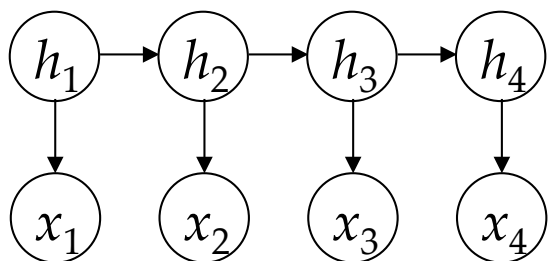


- The datum x_t is generated from hypothesis h_t
- The new hypothesis h_{t+1} is generated from h_t (in some fashion)

Dynamic models avoid the problem

- **Why** does the problem disappear?
 - Because the maths is subtly different:

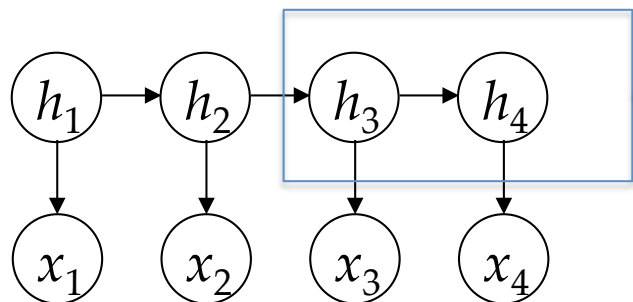
$$P(h_t|x_1, \dots, x_t) \propto P(x_t|h_t) \sum_{h_{t-1}} P(h_t|h_{t-1})P(h_{t-1}|x_1, \dots, x_{t-1})$$



Dynamic models avoid the problem

- **Why** does the problem disappear?
 - Because the maths is subtly different:


$$P(h_t|x_1, \dots, x_t) \propto P(x_t|h_t) \sum_{h_{t-1}} \boxed{P(h_t|h_{t-1})} P(h_{t-1}|x_1, \dots, x_{t-1})$$



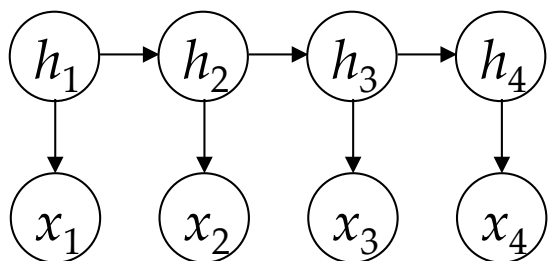
The model makes an explicit, **probabilistic** transitions between hypotheses as data arrive.

Dynamic models avoid the problem

- **Why** does the problem disappear?
 - Because the maths is subtly different:

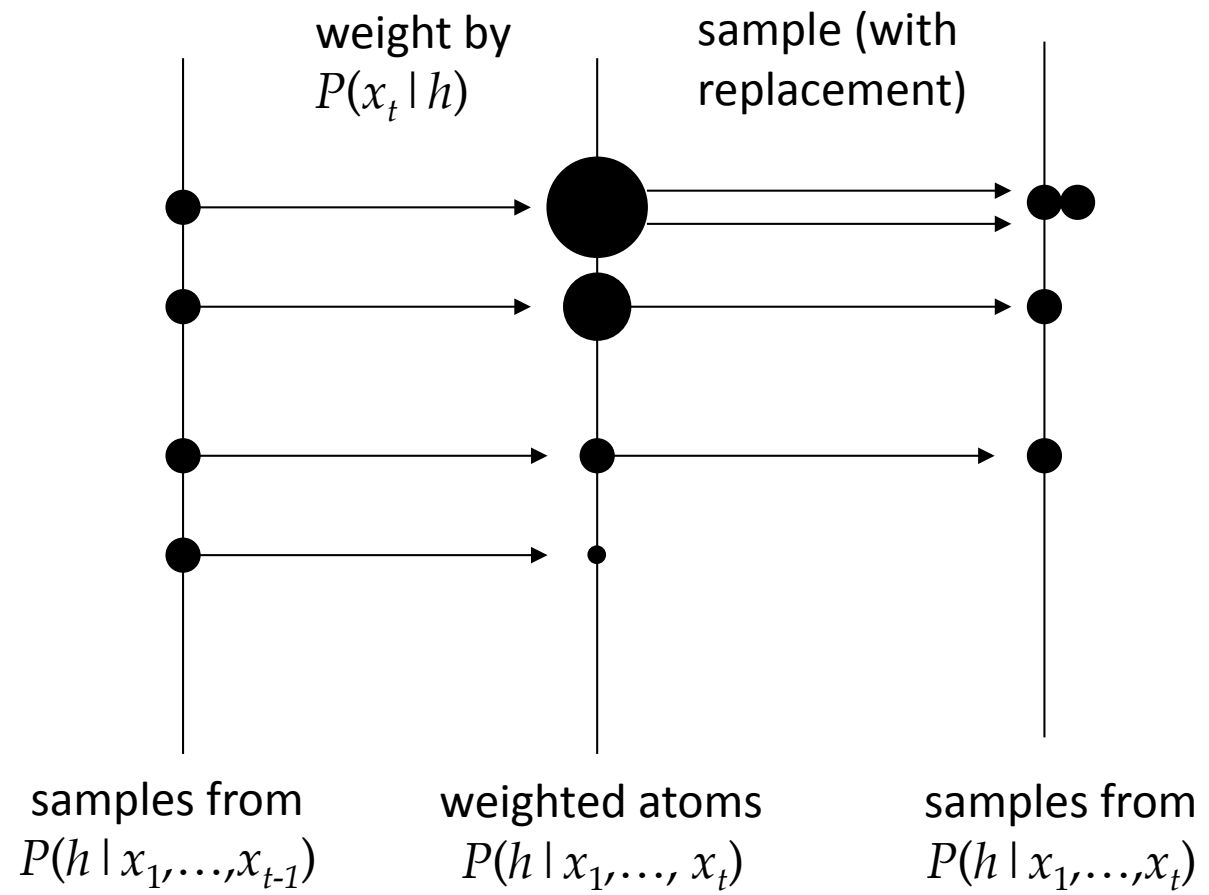
$$P(h_t | x_1, \dots, x_t) \propto P(x_t | h_t) \sum_{h_{t-1}} \boxed{P(h_t | h_{t-1})} P(h_{t-1} | x_1, \dots, x_{t-1})$$


The consequence is that the particle filtering algorithm involves an extra step when the model is dynamic.

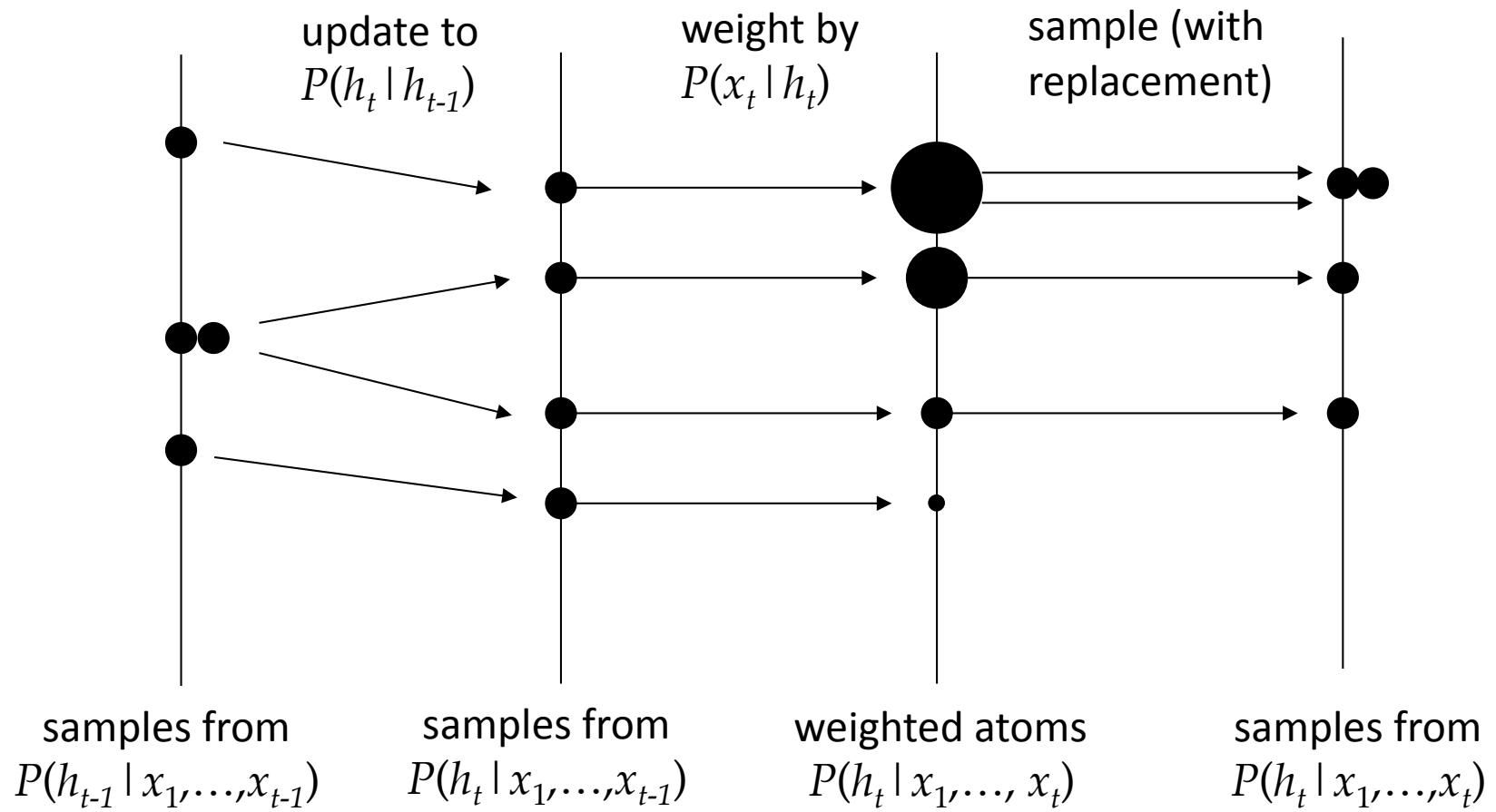


As before, we sample h_{t-1} from $P(h_{t-1} | x_1 \dots x_{t-1})$ using our particle filter approximation, but now we update the sampled h_{t-1} to h_t by drawing from $P(h_t | h_{t-1})$

STATIC:

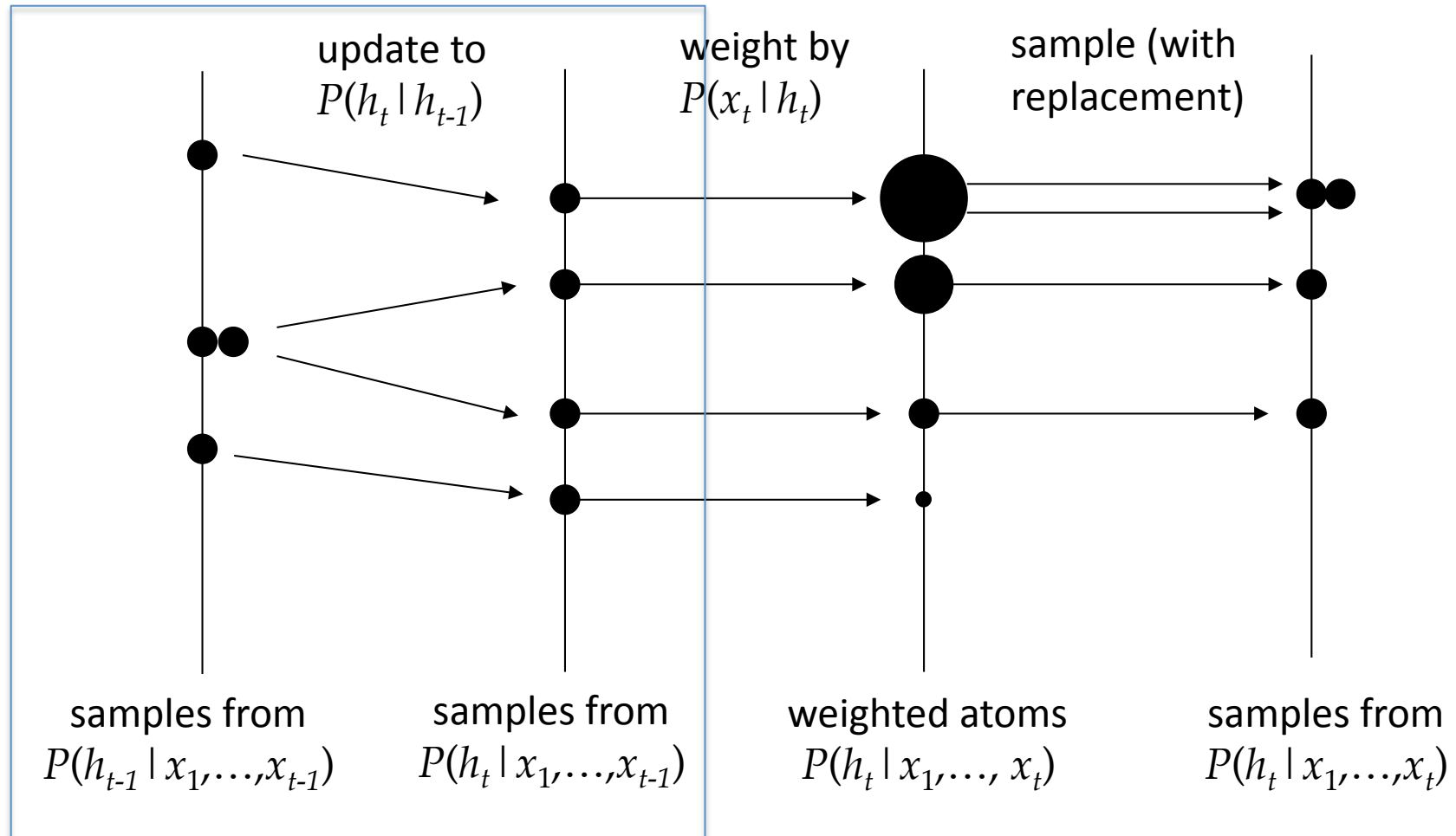


DYNAMIC:



DYNAMIC:

THE MODEL ITSELF HAS A MECHANISM FOR INTRODUCING NEW h VALUES!!!



Solution #2

- Turn your static model into a dynamic one!
- We'll use the AI survey data as an example...
 - In the original problem description, the model is static, and described by the parameters θ_0 , θ_1 and ϕ .

Let $\mathbf{x} = (x_1, \dots, x_M)$ denote the data from the $M = 100$ respondents, where x_i is the number of statements endorsed by the i th survey respondent. Let s_i be an (unobserved) binary “species assignment” variable that indicates which species each respondent belongs to, such that

$$s_i = \begin{cases} 1 & \text{if the } i\text{th participant is human} \\ 0 & \text{if the } i\text{th participant is a robot} \end{cases}$$

Where we let $\phi = P(s = 1)$ denote the probability that a randomly selected survey respondent turns out to be human. Let θ_0 denote the probability that a robot will endorse a statement about the strong AI hypothesis, and let θ_1 denote the corresponding probability for a human. Then x_i has distribution:

$$x_i \sim \begin{cases} \text{Binomial}(\theta_1, N) & \text{if } s_i = 1 \\ \text{Binomial}(\theta_0, N) & \text{if } s_i = 0 \end{cases}$$

where there are $N = 20$ questions, and the $\text{Binomial}(\theta, N)$ distribution is

$$P(x|\theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}$$

The values of ϕ , θ_1 and θ_2 are unknown! The problem is to infer $P(\theta_1, \theta_2, \phi | \mathbf{x}, M, N)$.

Let $\mathbf{x} = (x_1, \dots, x_M)$ denote the data from the $M = 100$ respondents, where x_i is the number of statements endorsed by the i th survey respondent. Let s_i be an (unobserved) binary “species assignment” variable that indicates which species each respondent belongs to, such that

$$s_i = \begin{cases} 1 & \text{if the } i\text{th participant is human} \\ 0 & \text{if the } i\text{th participant is a robot} \end{cases}$$

Where we let $\phi = P(s = 1)$ denote the probability that a randomly selected survey respondent turns out to be human. Let θ_0 denote the probability that a robot will endorse a statement about the strong AI hypothesis, and let θ_1 denote the corresponding probability for a human. Then x_i has distribution:

$$x_i \sim \begin{cases} \text{Binomial}(\theta_1, N) & \text{if } s_i = 1 \\ \text{Binomial}(\theta_0, N) & \text{if } s_i = 0 \end{cases}$$

where there are $N = 20$ questions, and the $\text{Binomial}(\theta, N)$ distribution is

$$P(x|\theta, N) = \frac{N!}{x!(N-x)!} \theta^x (1-\theta)^{N-x}$$

The values of ϕ , θ_1 and θ_2 are unknown! The problem is to infer $P(\theta_1, \theta_2, \phi | \mathbf{x}, M, N)$.

Solution #2

- Turn your static model into a dynamic one!
- We'll use the AI survey data as an example...
 - In the original problem description, the model is static, and described by the parameters θ_0 , θ_1 and ϕ .
 - What happens if we use the “species assignment variables” $\mathbf{s} = (s_1, \dots, s_M)$, and integrate everything else out?

Particle filters for the AI data

- Let the k -th particle (at time t) correspond to a set of species-assignments for the first t respondents $\mathbf{s}_t^{(k)} = (s_1^{(k)}, \dots, s_t^{(k)})$
- Since we have a latent parameter ϕ

$$\begin{aligned} P(s_{t+1}^{(k)} = 1 \mid \mathbf{s}_t^{(k)}) &= \int_0^1 \phi P(\phi \mid \mathbf{s}_t^{(k)}) d\phi \\ &= \frac{m_t^{(k)} + 20}{t + 22} \end{aligned}$$

(This follows from the discussion of the standard beta-binomial model from earlier)

- where $m_t^{(k)}$ is the number of respondents that the k -th particle has classified as human so far.

Particle filters for the AI data

- Having made a species-assignment for respondent $t+1$ (for particle k) the probability $P(x_{t+1} | \mathbf{s}_{t+1}^{(k)})$ that this person endorses x_{t+1} statements is...

$$P(x_{t+1} | \mathbf{s}_t^{(k)}, s_{t+1}^{(k)} = 1) = \int_0^1 P(x_{t+1} | \theta_1) P(\theta_1 | \mathbf{s}_t^{(k)}, \mathbf{x}_t) d\theta_1$$

- which has an analytic solution that I'll make explicit in the auxiliary materials

PARTICLE FILTERING ALGORITHM FOR THE “AI SURVEY” DATA

choose p , the number of particles

for $r = 1 : M$ (i.e. add the respondents one by one)

for $k = 1 : p$ (loop over the particles)

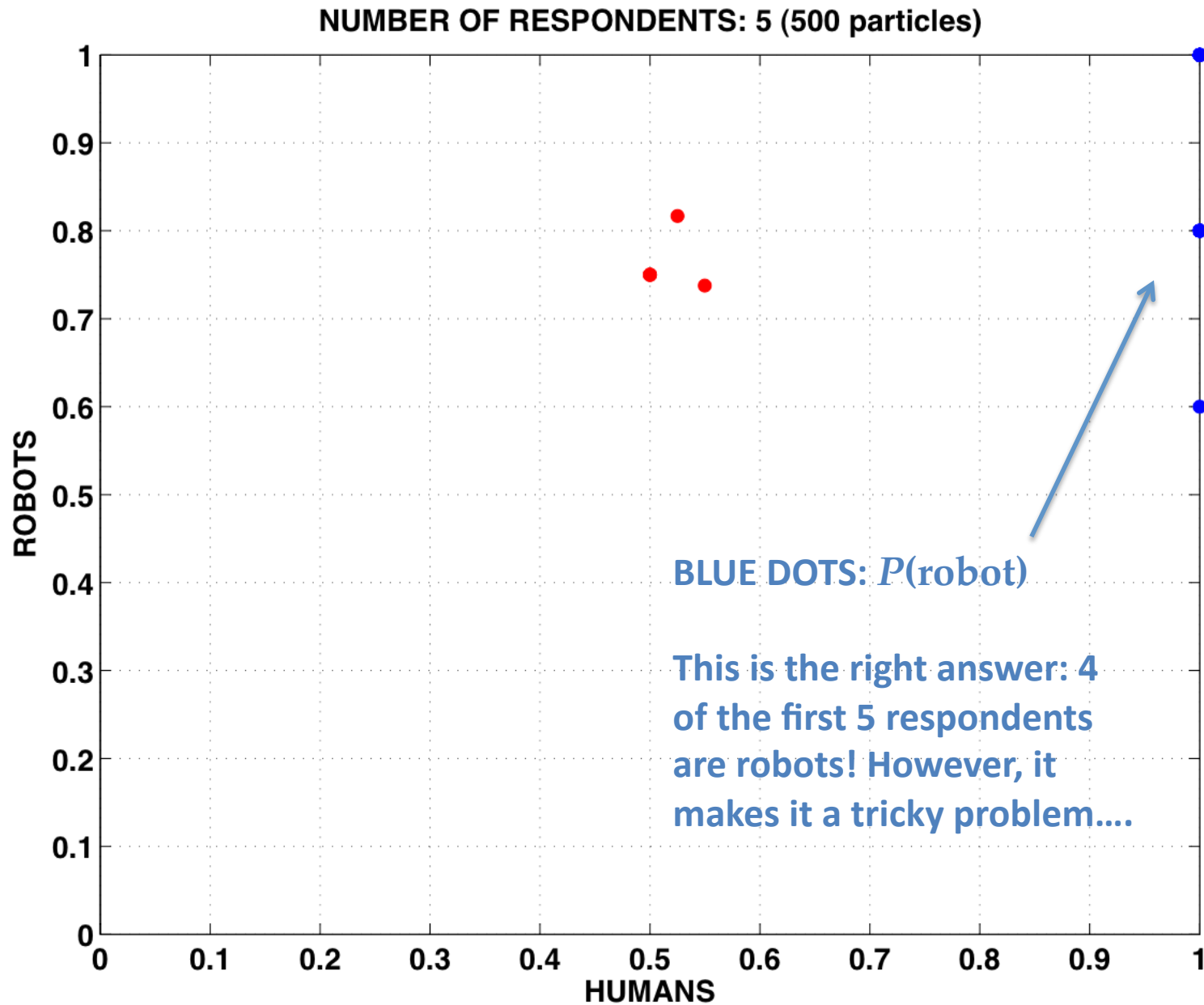
sample species assignment: $s_r^{(k)} \mid s_{r-1}^{(k)}$

calculate unnormalised weight $w^{(k)}$

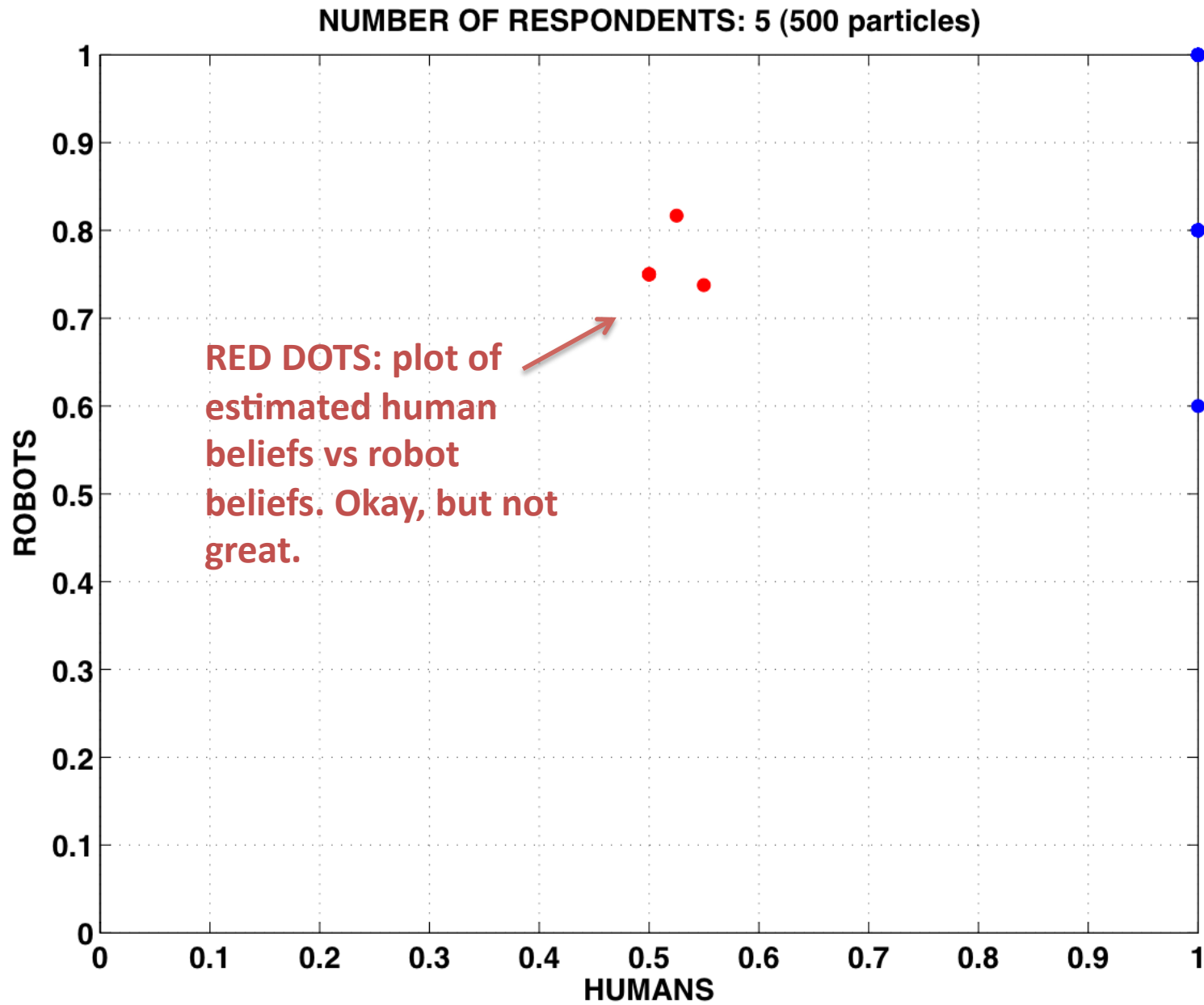
normalise the weights so that they sum to 1

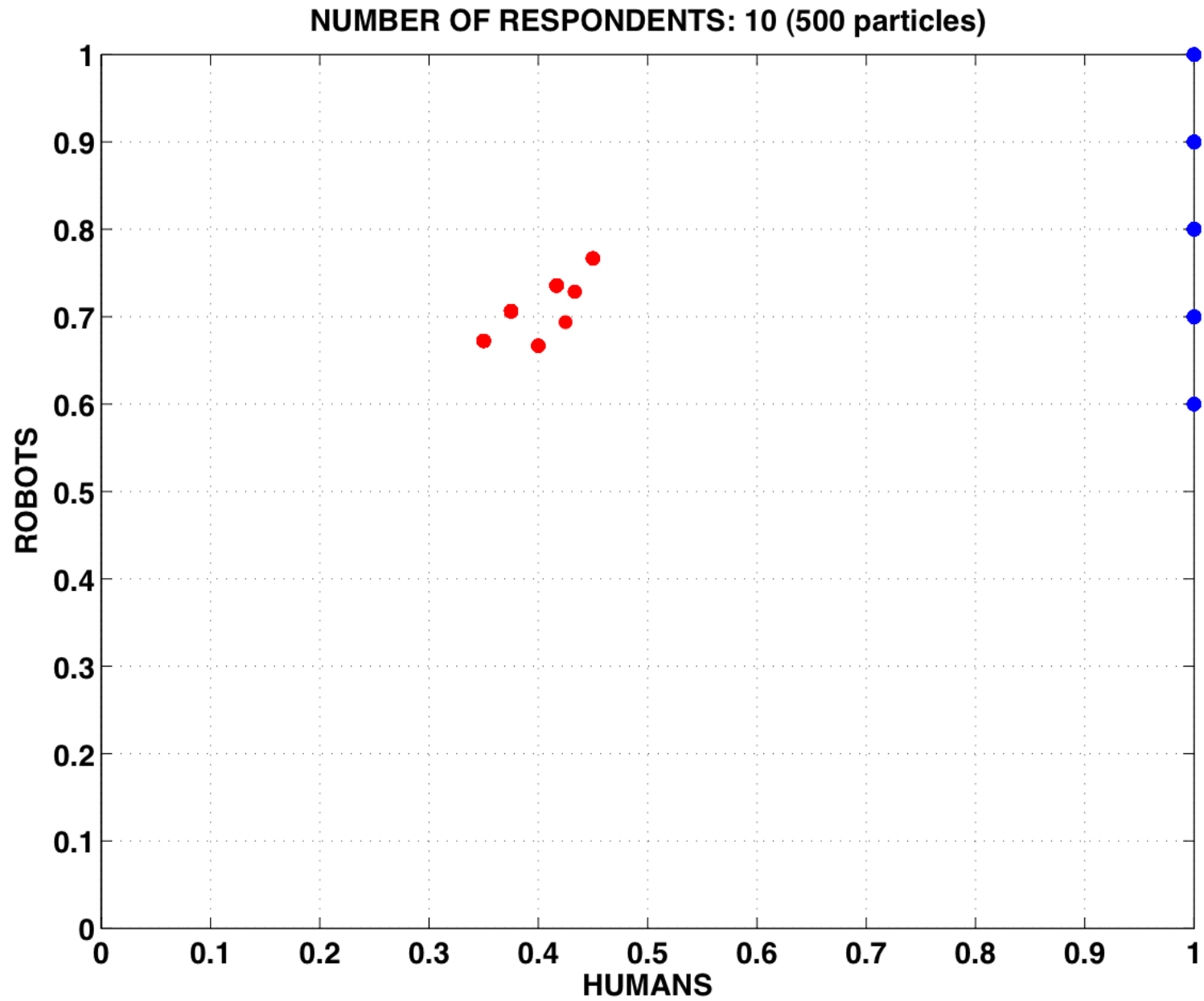
for $k = 1 : p$ (loop over the particles)

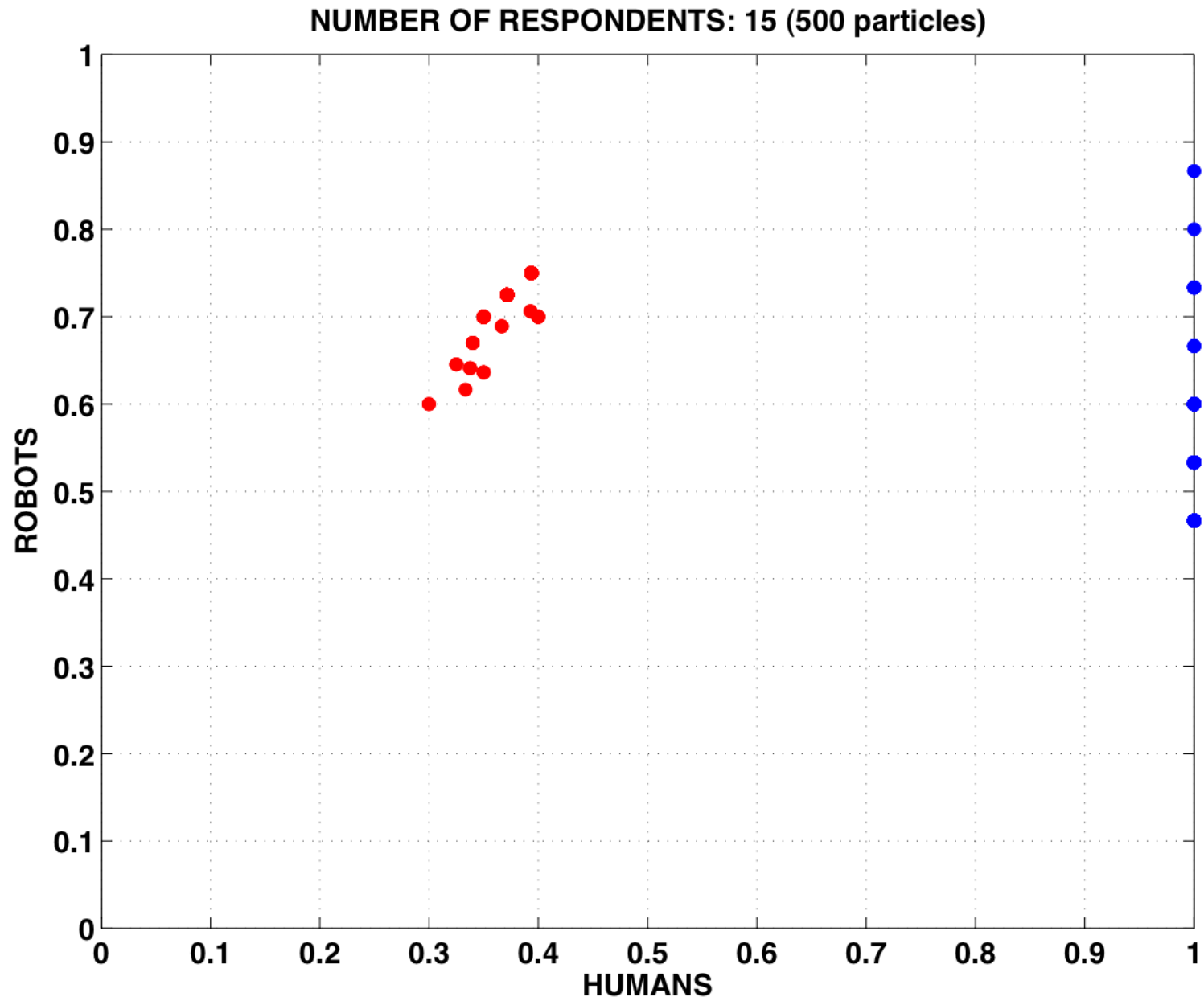
resample particle k : $P(\text{new particle } k = \text{old particle } j) = w^{(j)}$

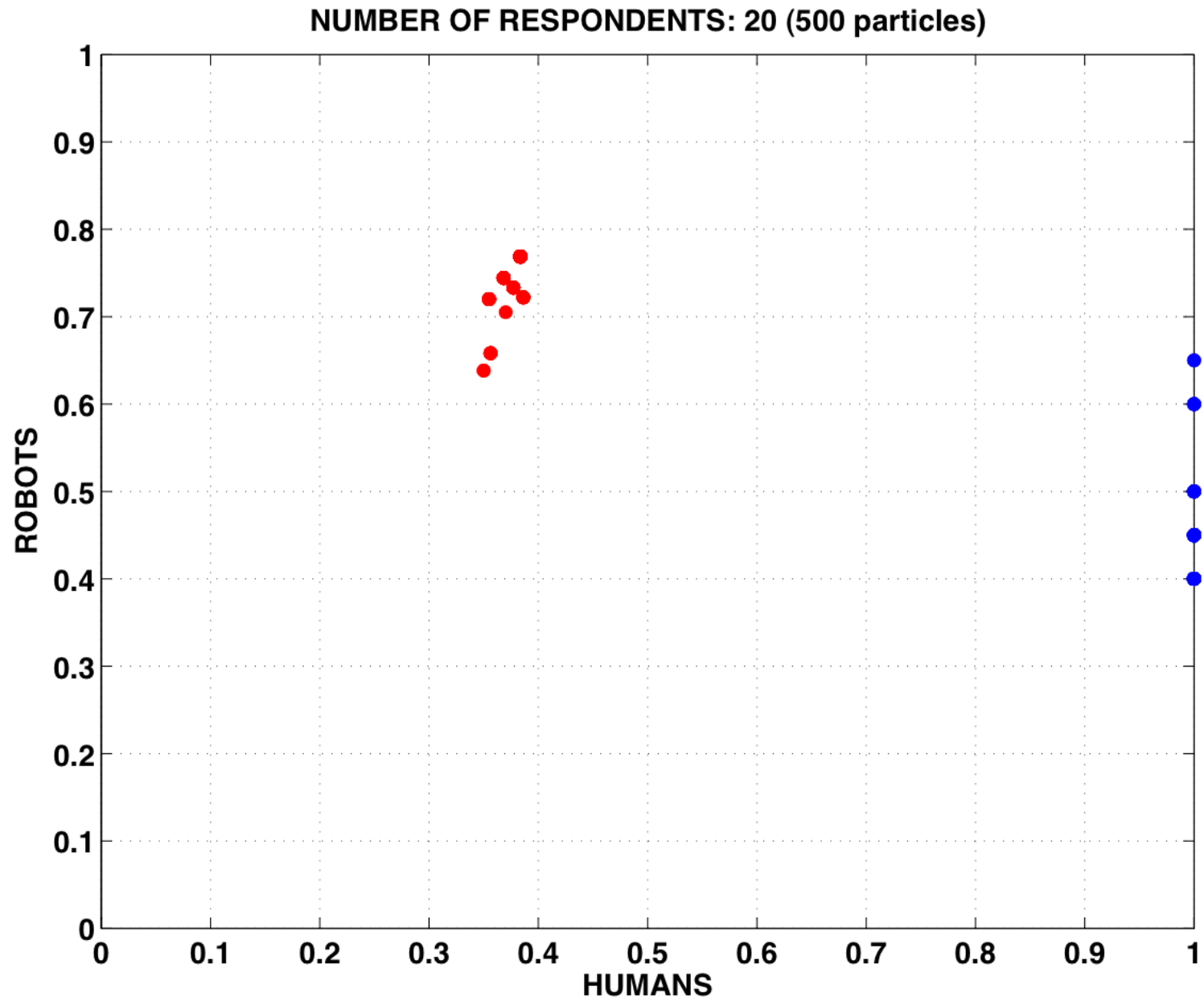


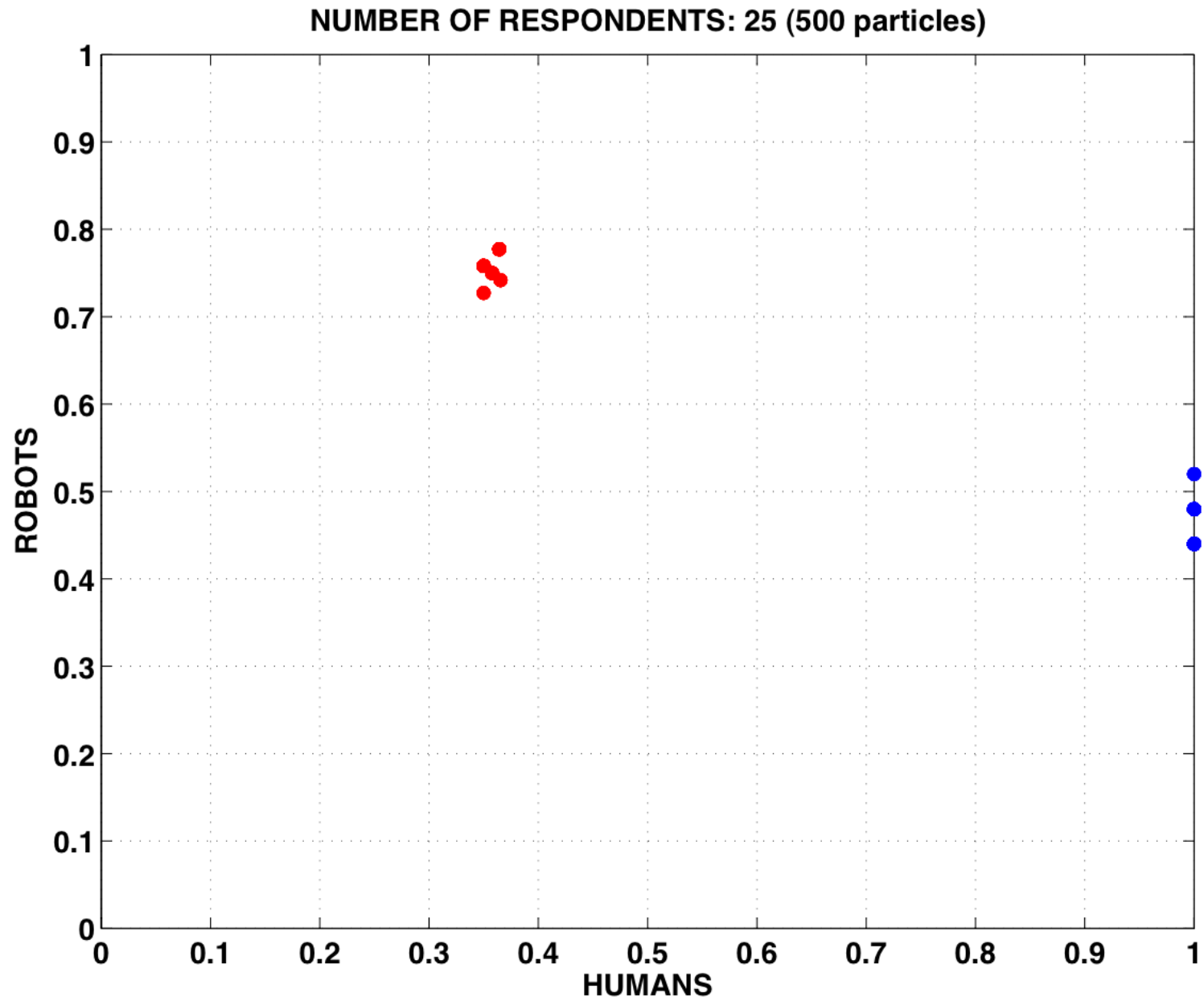
* note that this visual display plots $P(\text{robot})$ rather than $P(\text{human})$











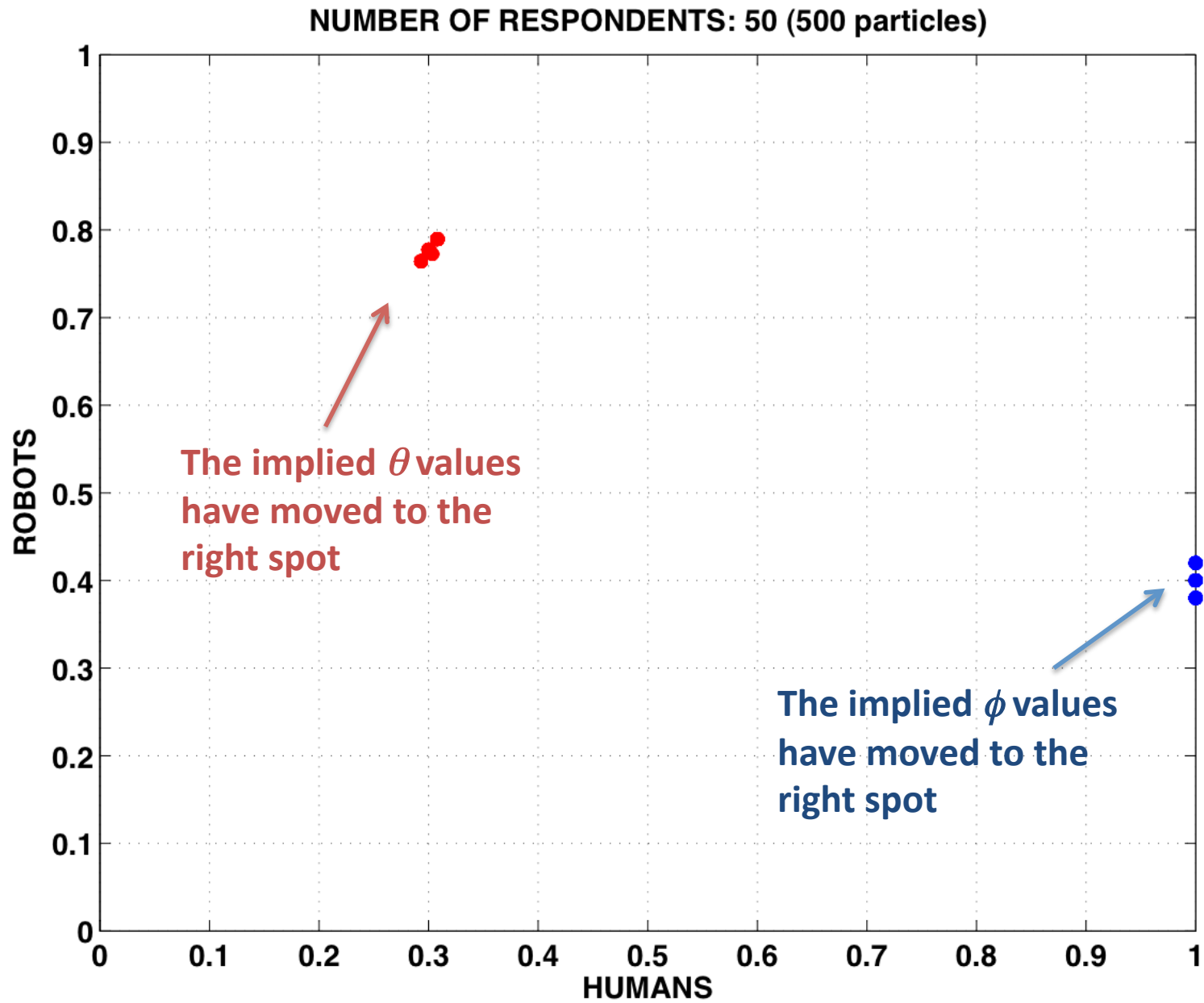
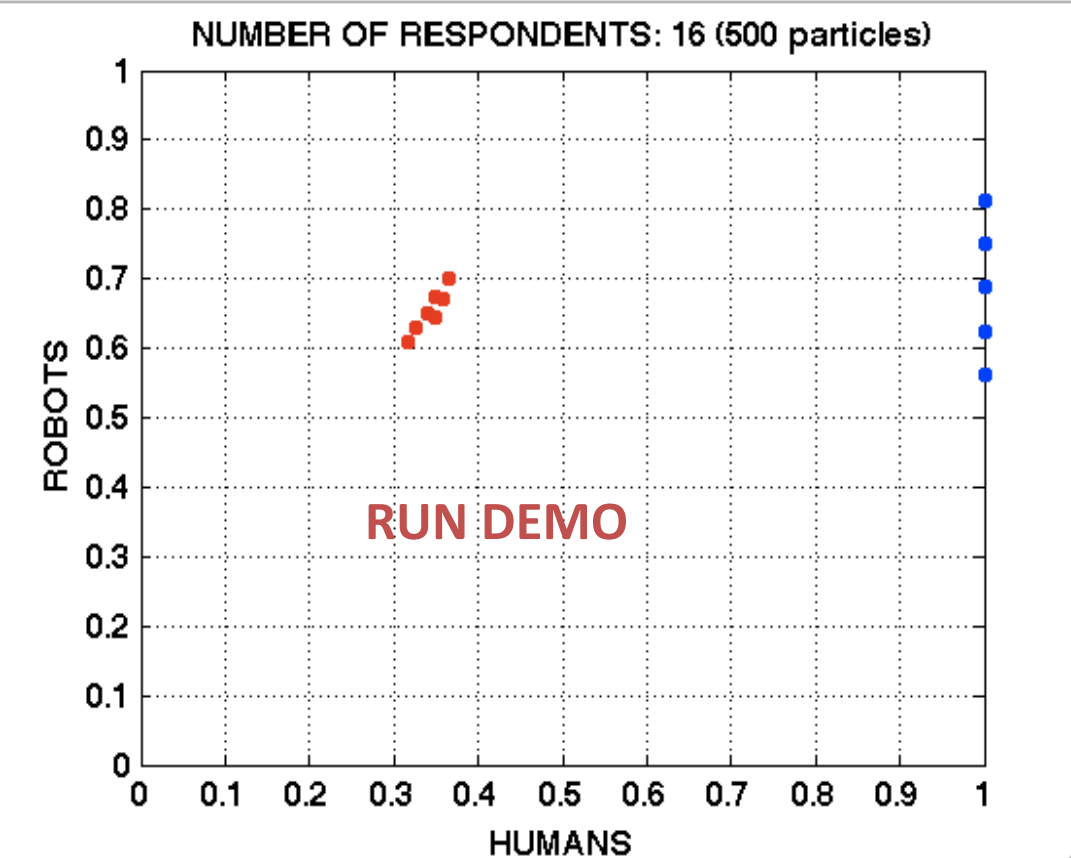


Figure 1



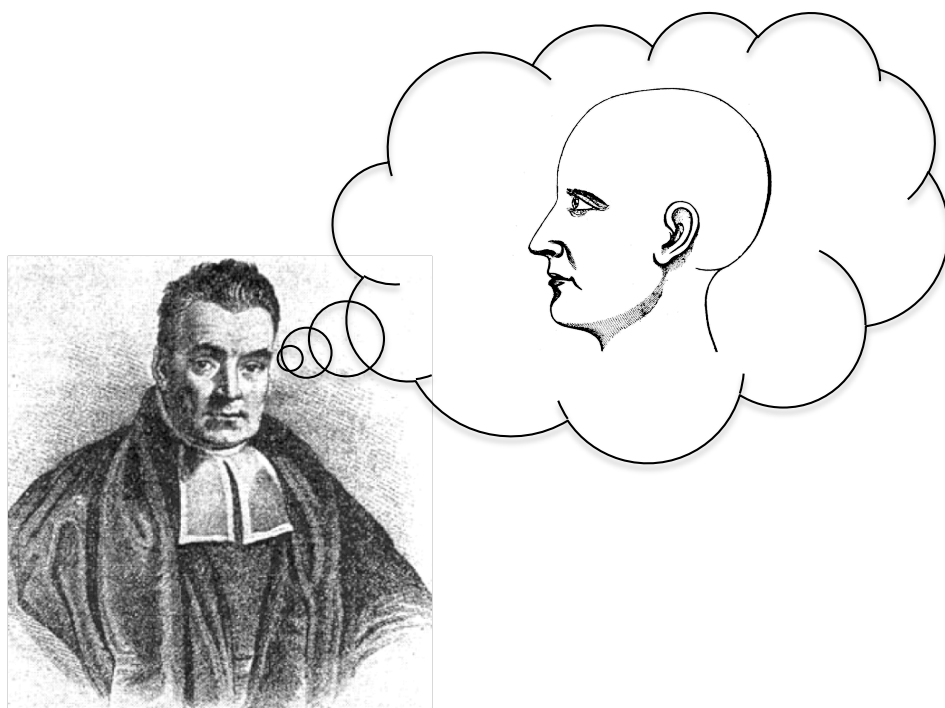
“RATIONAL PROCESS” MODELS

Why are we doing this again?

- Reason #1:
 - MCMC, importance sampling and particle filtering form the basic toolkit of all modern Bayesian computational statistics
 - Without them, you can't solve anything except the simplest of problems
- Reason #2:
 - The mind has to solve problems that aren't simple. Probably has a similar toolkit!

Reason #1

Bayesian data analysis

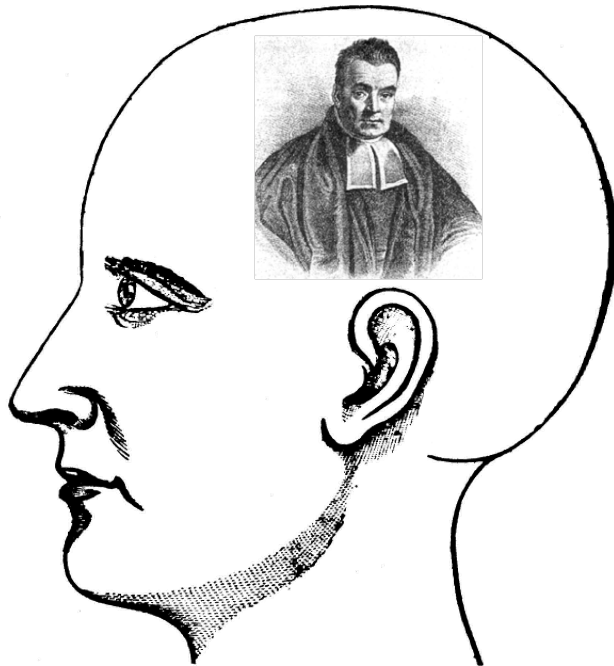


MCMC is a tool:

The AI survey problem is a simple example of how researchers in the social sciences like to use MCMC. That is, we specify a measurement model for the data, and then use MCMC to do the inference.

Reason #2

Bayesian cognition



Computational statistics might also be an explanatory mechanism

People have to deal with the exact same problems... the world is complex, and has lots of variables that you need to draw inferences about. Maybe MCMC & particle filtering is how humans solve the problem!

This idea is called **rational process modelling** – the idea that statistics doesn't just solve problems at Marr's computational level (using Bayes theorem), it also provides solutions at the algorithmic level (via MCMC, etc)

