

Introduction to Bayesian Inference with PyMC

Lara Kattan

Nov. 2022

Questions We'll (Begin to) Answer

- What is Bayesian estimation and inference?
- What is probabilistic programming?
- What is PyMC and how do I begin to use it?
- GitHub link: [https://github.com/larakattan/ODSC PyMC](https://github.com/larakattan/ODSC_PyMC)

Bayesian Inference

- Bayesian inference was originally called “[inverse probability](#)”, now called the posterior distribution
 - Inverse probability = posterior distribution =
distribution of an unobserved parameter/variable, conditional on data and prior distro
- Inverse probability: name came from assigning a probability distro to an unknown parameter
 - Reason from effects (observations) to causes (parameter distribution)
 - What’s the most likely value of our parameter of interest, conditioning on the data we observe?
- “Direct probability” is now called the likelihood (not a probability distribution, technically)

Bayesian Inference

- Bayesian inference was originally called “inverse probability”, now called the posterior distribution
 - Inverse probability = posterior distribution =
distribution of an unobserved parameter/variable, conditional on data and prior distribution
- Inverse probability: name came from assigning a probability distribution to an unknown parameter
 - Reason from effects (observations) to causes (parameter distribution)
 - What's the most likely value of our parameter of interest, conditioning on the data we observe?
- “Direct probability” is now called the likelihood (not a probability distribution, technically)
- A parameter is any measured quantity of a statistical population that summarises or describes an aspect of the population
- Estimating an unknown parameter is **inferential statistics**, hence Bayesian inference

Bayesian Inference

- Outputs differ from traditional Frequentist statistics
 - Frequentist: point estimates of parameters and confidence intervals
 - Bayesian: posterior probability distributions of parameters
- Why? Because what about things that happen only once; e.g. elections
- Now: how do we go about getting these distributions?

Mechanics: Updating Beliefs with Evidence

- Example: You look at the sidewalk outside your window and see that it's wet. Question: Where did the water come from?
- Prior: Start with a prior belief, for example that it was raining earlier

Mechanics: Updating Beliefs with Evidence

- Example: You look at the sidewalk outside your window and see that it's wet. Question: Where did the water come from?
- Prior: Start with a prior belief, for example that it was raining earlier
- Collect data: Look up at the sky and see that it's perfectly clear, there are no clouds. What's your posterior for whether the water came from rain now? Is it higher or lower than before?
- Collect more data: There is a water hose out next to the sidewalk.

Mechanics: Updating Beliefs with Evidence

- Example: You look at the sidewalk outside your window and see that it's wet. Question: Where did the water come from?
- Prior: Start with a prior belief, for example that it was raining earlier
- Collect data: Look up at the sky and see that it's perfectly clear, there are no clouds. What's your posterior for whether the water came from rain now? Is it higher or lower than before?
- Collect more data: There is a water hose out next to the sidewalk.
- What's your posterior now on whether rain caused the water? Probably fairly low! And lower than it was when you first looked out the window
 - That's Bayesian inference
 - Updating a prior using data to arrive at a posterior
- Usually not 100% sure, but we can get pretty close

Bayesian Inference



```
graph LR; A[Prior assumptions] --> B[Evidence (data)]; B --> C[Update prior]; C --> D[Posterior distribution];
```

Prior assumptions

Evidence (data)

Update prior

Posterior distribution

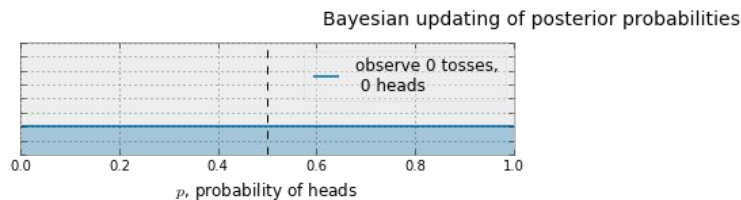
Updating Priors

Question: What is the probability that a coin we have in front of us will come up heads?

Updating Priors

Bayesian answer:

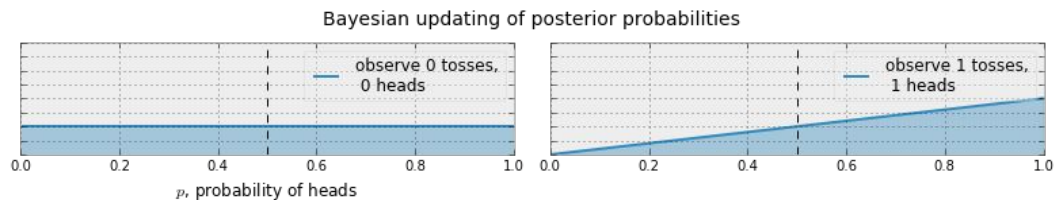
- **Prior:** start with a **prior distribution**. Here we use a uniform distribution: the coin is just as likely to always come up heads as it is to never come up heads. It's also equally likely that it'll be heads 40% of the time (slightly weighted coin).



Updating Priors

Bayesian answer:

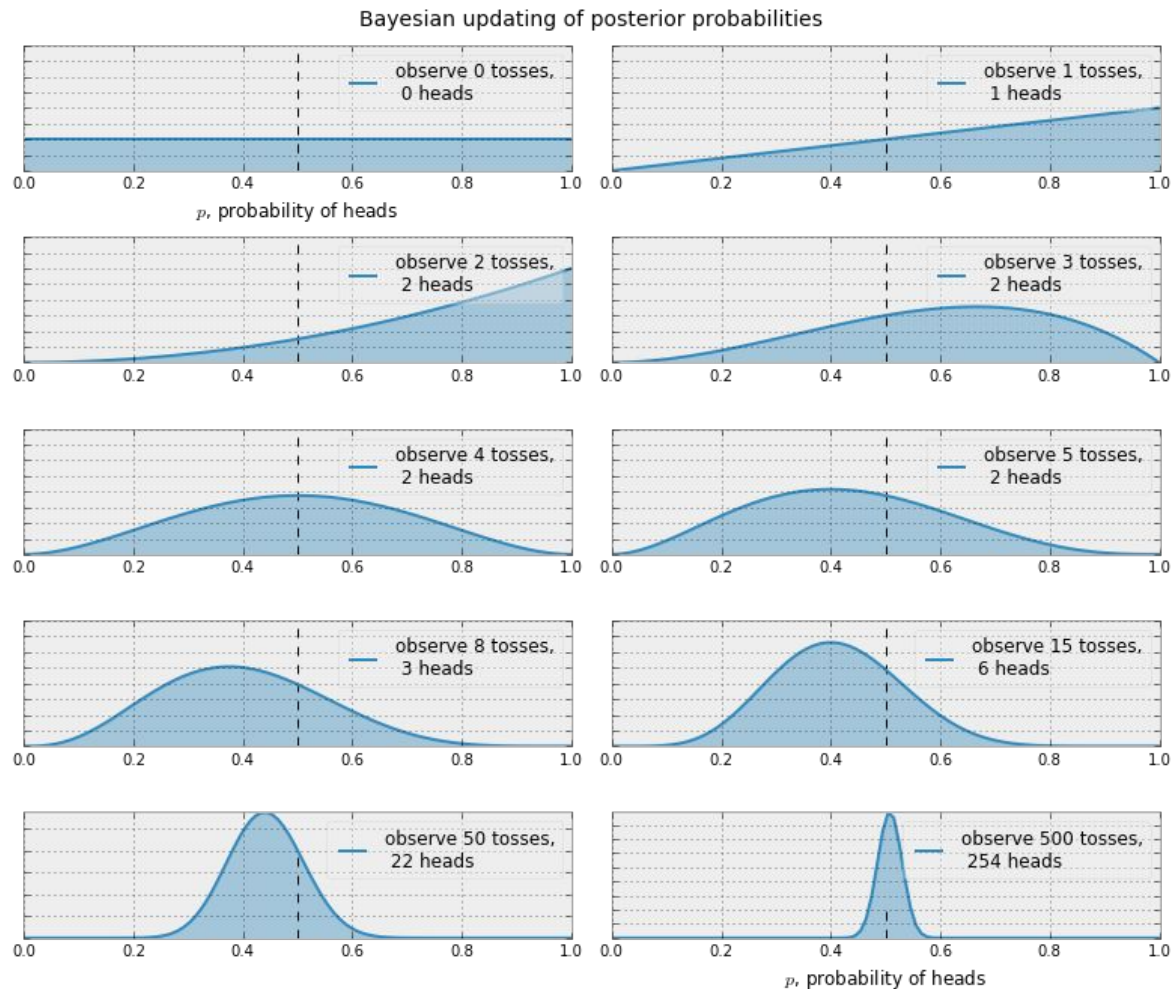
- **Prior:** start with a **prior distribution**. Here we use a uniform distribution: the coin is just as likely to always come up heads as it is to never come up heads. It's also equally likely that it'll be heads 40% of the time (slightly weighted coin).
- **Data:** toss the coin and record whether heads.
- **Posterior:** update the distribution to reflect that heads is more/less likely depending on the outcome of the toss.



Updating Priors

Bayesian answer:

- The old posterior becomes the new prior. Continue to collect data and update the prior into the new posterior.



Bayesian Inference vs Frequentist

As we gather an *infinite* amount of evidence, say as $N \rightarrow \infty$, our Bayesian results (often) align with frequentist results. Hence for large N , statistical inference is more or less objective. On the other hand, for small N , inference is much more *unstable*: frequentist estimates have more variance and larger confidence intervals.

This is where Bayesian analysis excels. By introducing a prior, and returning probabilities (instead of a scalar estimate), we *preserve the uncertainty* that reflects the instability of statistical inference of a small N dataset.

https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter1_Introduction/Ch1_Introduction_PyMC3.ipynb

Bayes' Theorem

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$\Rightarrow P(A \cap B) = P(B \cap A) = P(A|B)P(B) = P(B|A)P(A)$$

$$\Rightarrow P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Bayes' Theorem

Likelihood
(prev. direct probability)

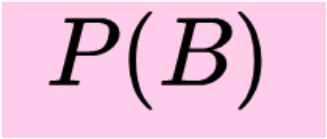

$$P(A|B) = \frac{\overbrace{P(B|A)P(A)}^{\text{Likelihood (prev. direct probability)}}}{P(B)}$$

Bayes' Theorem

$$P(A|B) = \frac{P(B|A) \overbrace{P(A)}^{\text{Prior}}}{P(B)}$$

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$



Evidence

Bayes' Theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Posterior
(prev. Inverse probability)

Probabilistic Programming

- General term for using the high-level programming language of your choice to define and estimate a probability model and run inference
 - Language needs to have random/stochastic events as primitives
 - We can treat the distributions as actual objects and investigate them (e.g. mean or std dev)
 - Primitives can be random variables (e.g. normal distribution) or a DGP (data generating process)
 - Programming language will abstract away complexities of writing/running models
- “Programming” here means writing code, not optimization
 - From Wikipedia: “Programming in this context [optimization] ... comes from the use of program by the United States military to refer to proposed training and logistics schedules, which were the problems Dantzig studied at that time”

Probabilistic Programming

- Been around for decades, with roots in WinBUGS (Bayesian inference Using Gibbs Sampling), first released in 1997
 - Still around in the form of OpenBUGS
- More transparent than a lot of ML models (e.g. random forests) because the modeler has control over the distributions
- Better than a simulation
 - Simulation moves in one direction: get data input and move it according to assumptions of parameters and get a prediction
 - Unknown params are not distributions
 - Bayesian modeling via PP adds another direction: use the data to go back and pick one of many possible parameters as the most likely to have created the data (posterior distros)
- A way to make Bayesian inference tractable (less math)

Probabilistic Programming

[U]nlike a traditional program, which only runs in the forward directions, a probabilistic program is run in both the forward and backward direction.

It runs forward \rightarrow to compute the consequences of the assumptions it contains about the world (i.e., the model space it represents), but it also runs backward \leftarrow from the data to constrain the possible explanations.

In practice, many probabilistic programming systems will cleverly interleave these forward and backward operations to efficiently home in on the best explanations.

Cronin, Beau. "Why Probabilistic Programming Matters." 24 Mar 2013. Google, Online Posting to Google forum. 24 Mar. 2013. <https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1>

Bayesian Inference via Probabilistic Programming

- Solving Bayes' theorem analytically requires taking integrals (denominator)
- If we don't want to do that, we need to use numerical solution methods
- Lots of development in terms of new methods of sampling
 - Markov Chain Monte Carlo and Hamiltonian Monte Carlo
 - If some math satisfied (detailed balance equation), guaranteed to get the posterior distro in the limit
 - Then [NUTS](#), No U-Turn Sampler, which removes having to specify a step size
 - Gelman: "NUTS uses a recursive algorithm to build a set of likely candidate points that spans a wide swath of the target distribution, stopping automatically when it starts to double back and retrace its steps"
 - Variational inference
 - Make distributions similar to each other
 - [Optimization, not sampling, so more appropriate for big data](#)

Solutions via Markov Chain Monte Carlo

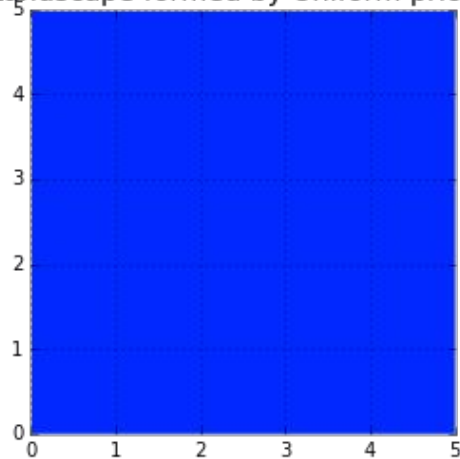
An inference problem with N unknowns/parameters is a problem in N -dimensional space (all the possible combinations of values for each parameter).

Within this space, imagine a surface or curve. The value of the **surface** at a given point is that point's **prior probability**. The surface then is defined by prior distributions.

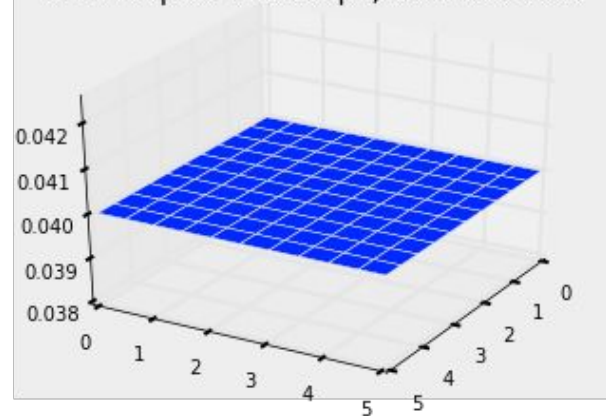
Solutions via Markov Chain Monte Carlo

E.g. If we have two unknowns p_1 and p_2 , and the prior for each is $\text{Uniform}(0,5)$, the curve is a square of side 5 and the surface is a flat plane that sits on top of the square (meaning every point is equally likely)

Landscape formed by Uniform priors.



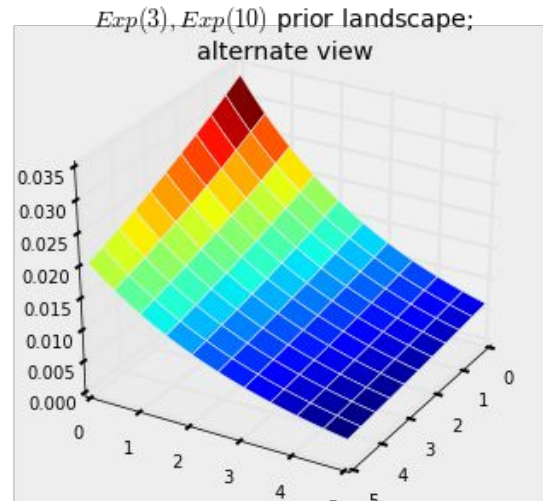
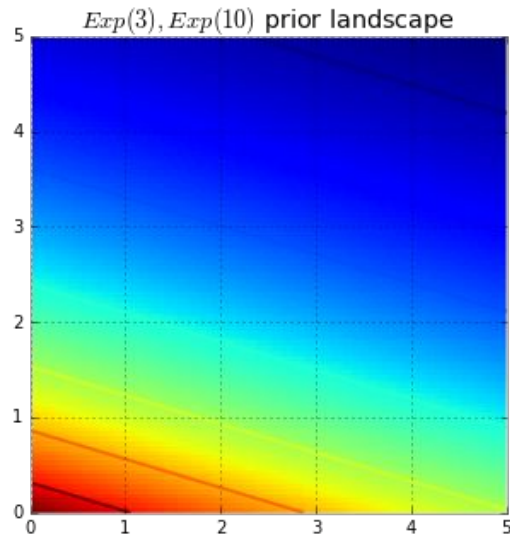
Uniform prior landscape; alternate view



Source:

https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter3_MCMC/C_h3_IntroMCMC_PyMC3.ipynb

Solutions via Markov Chain Monte Carlo



Source:

https://nbviewer.jupyter.org/github/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter3_MCMC/C_h3_IntroMCMC_PyMC3.ipynb

Solutions via Markov Chain Monte Carlo: A random walk

- MCMC explores this space and returns to us samples from the posterior distribution
 - In PyMC, samples are “traces”
- MCMC traverses the space, comparing the probability at that point, then moving to areas of higher probability
- At each new point, either accept or reject
- If the new point has higher probability, definitely move to that point (accept)
- If the new point has lower probability, accept sometimes and reject sometimes
 - The acceptance/rejection criteria is based on a ratio of the target distribution at the proposed new point relative to the value of the target distribution at our current position
 - Move probabilistically: move with probability $p(\theta_{\text{proposed}}) / p(\theta_{\text{current}})$

A random walk example

Source of this example is
the wonderful book [*Doing Bayesian Data Analysis*](#) by
[*John K. Kruschke*](#)

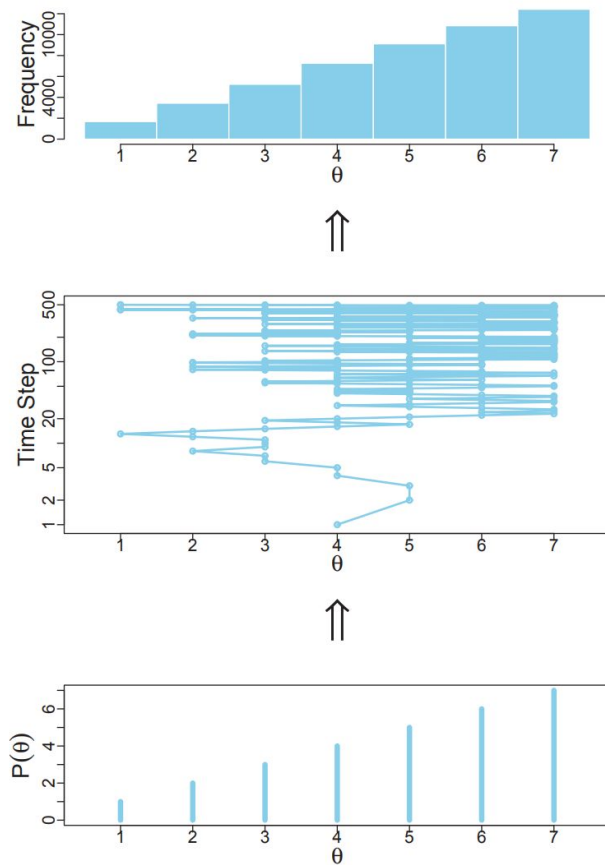
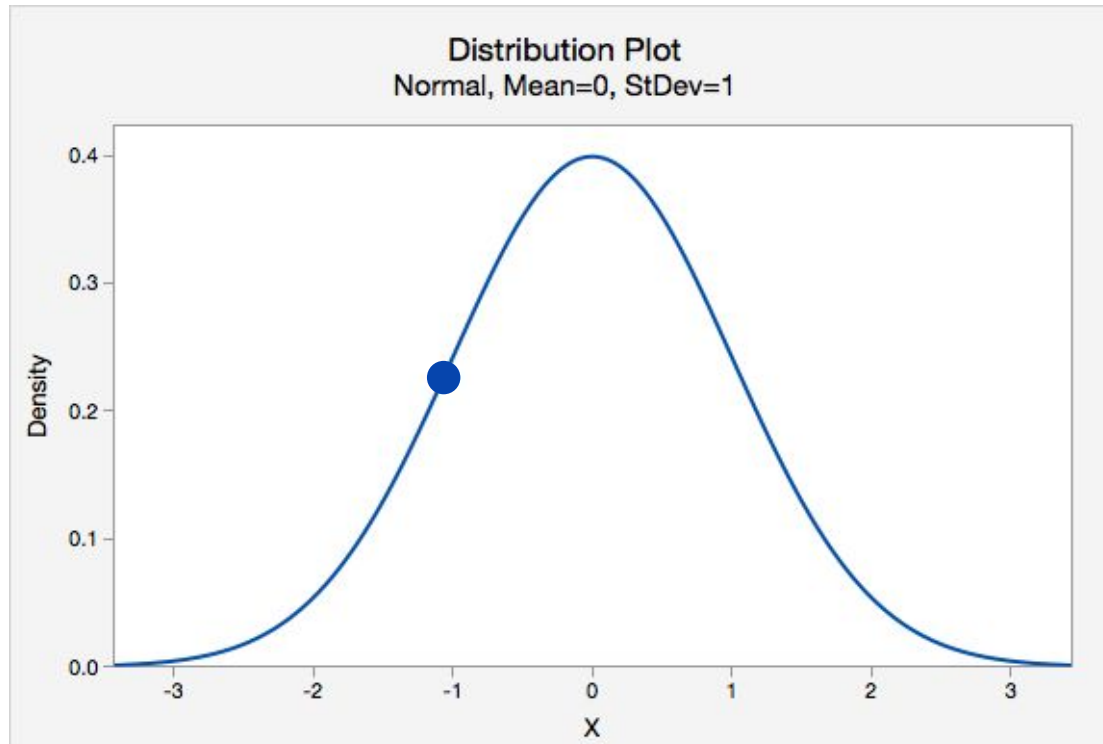
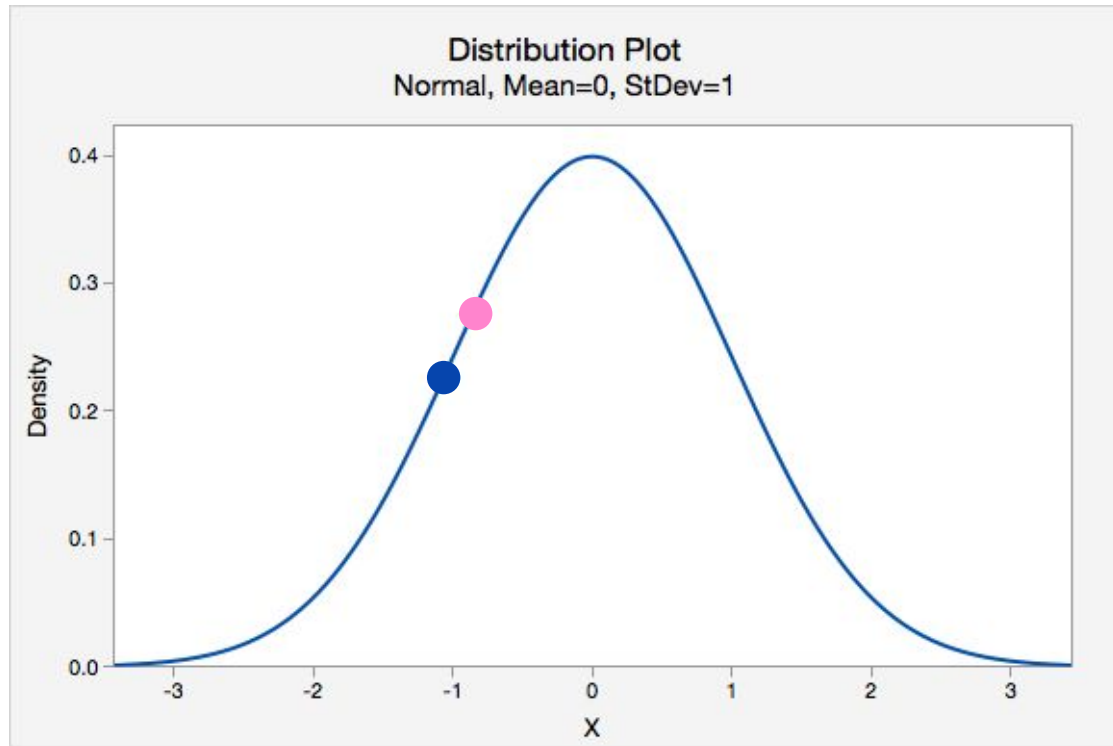


Figure 7.2: Illustration of a simple Metropolis algorithm. The bottom panel shows the values of the target distribution. The middle panel shows one random walk, at each time step proposing to move either one unit right or one unit left, and accepting the proposed move according to the heuristic described in the main text. The top panel shows the frequency distribution of the positions in the walk. Copyright © Kruschke, J. K. (2014). *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan*. 2nd Edition. Academic Press / Elsevier.

Solutions via Markov Chain Monte Carlo

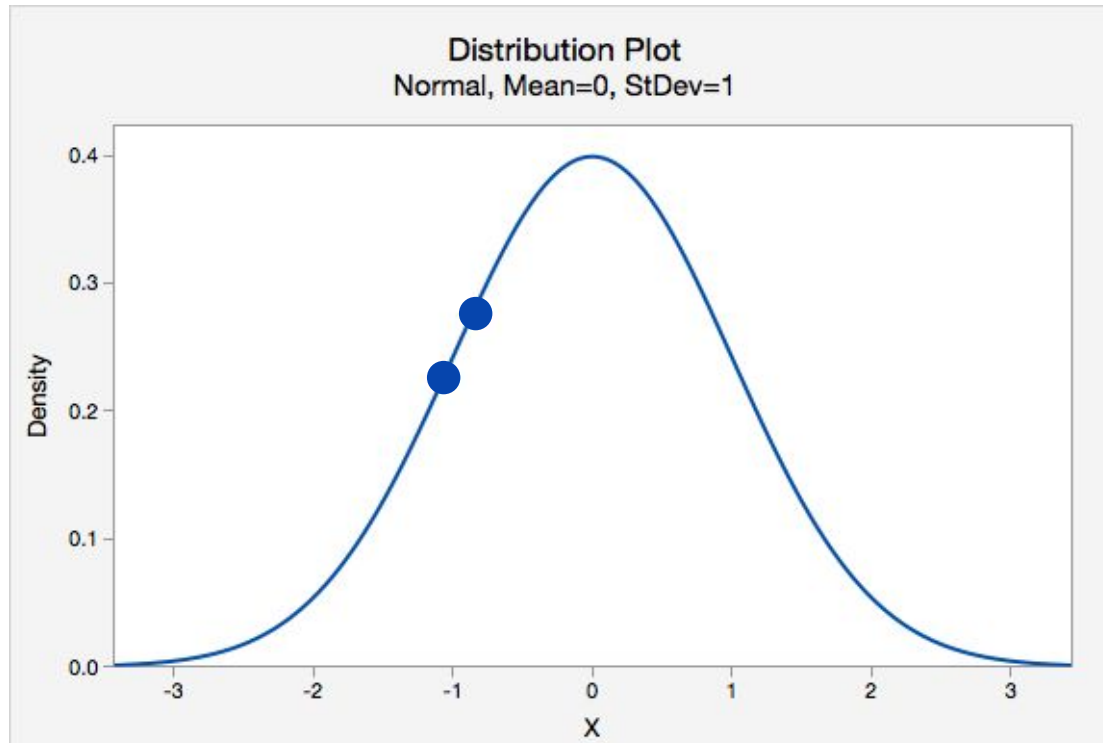


Solutions via Markov Chain Monte Carlo



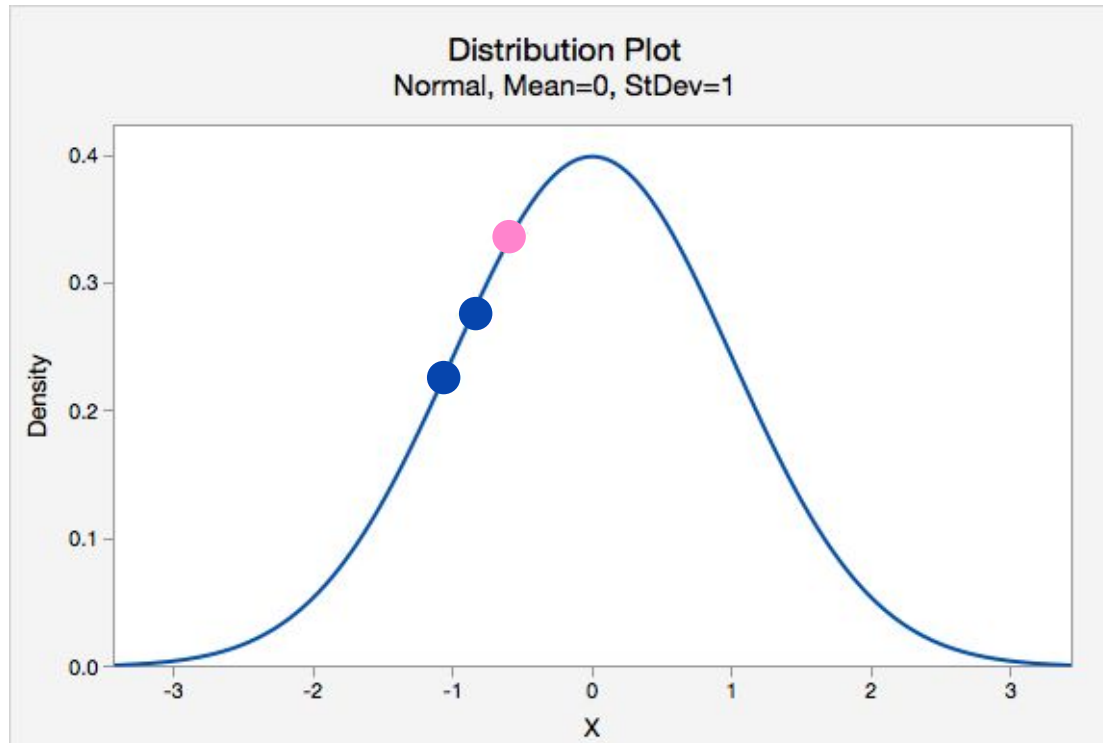
As you move to new point, either reject or accept based on prior and data; markov chain comes from the fact that we don't care about history: only prior position matters (memoryless)

Solutions via Markov Chain Monte Carlo

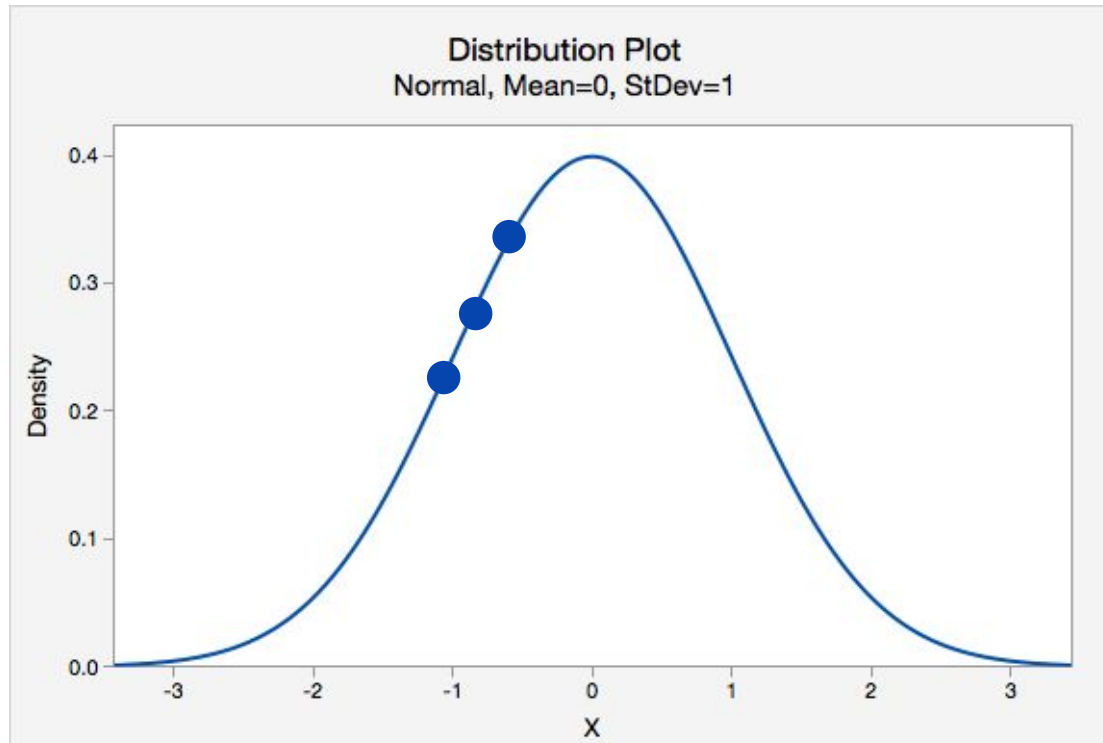


As you move to new point, either reject or accept based on prior and data; markov chain comes from the fact that we don't care about history: only prior position matters (memoryless)

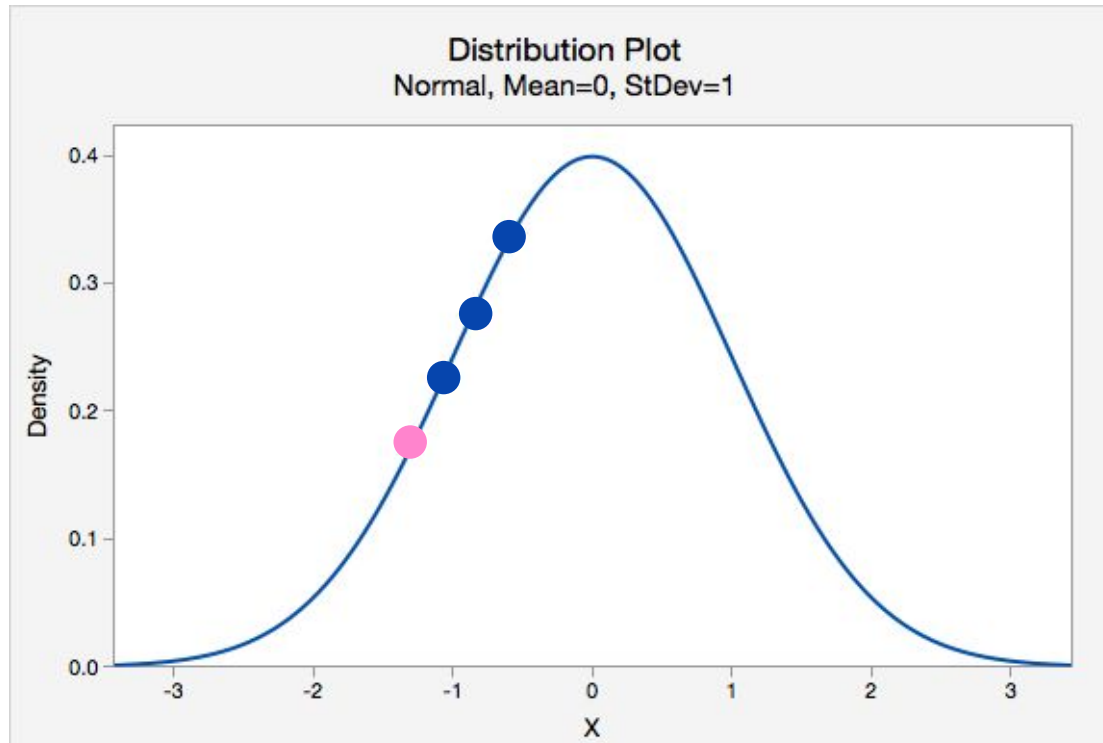
Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo

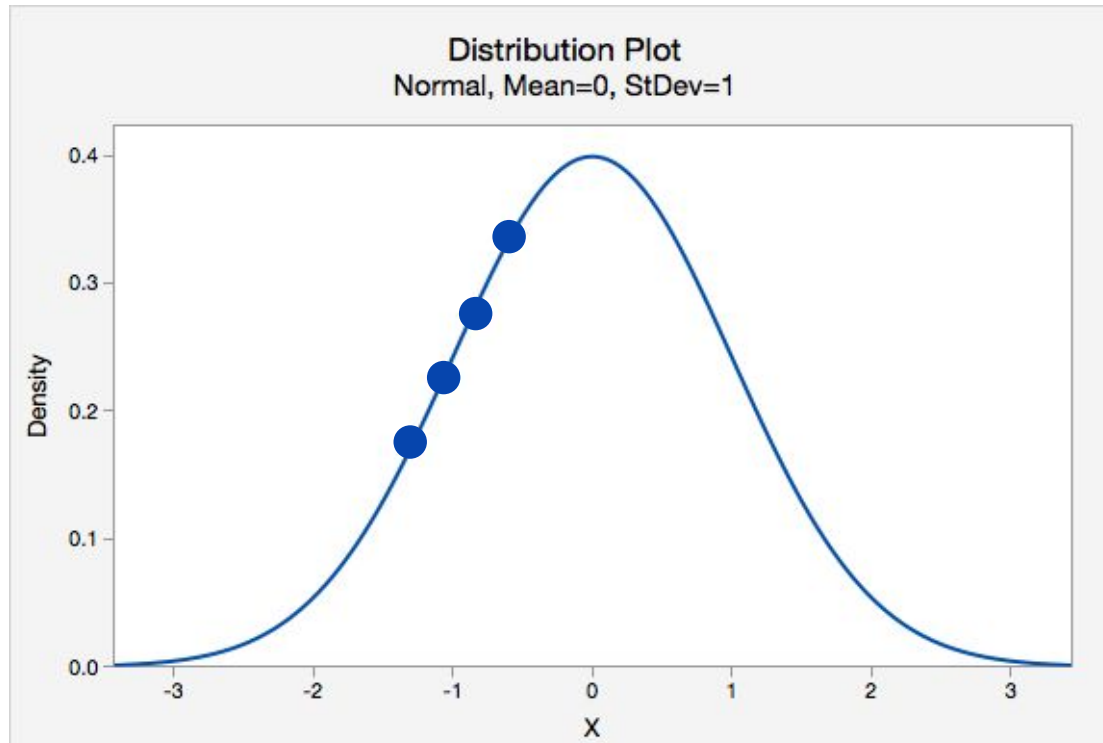


Solutions via Markov Chain Monte Carlo

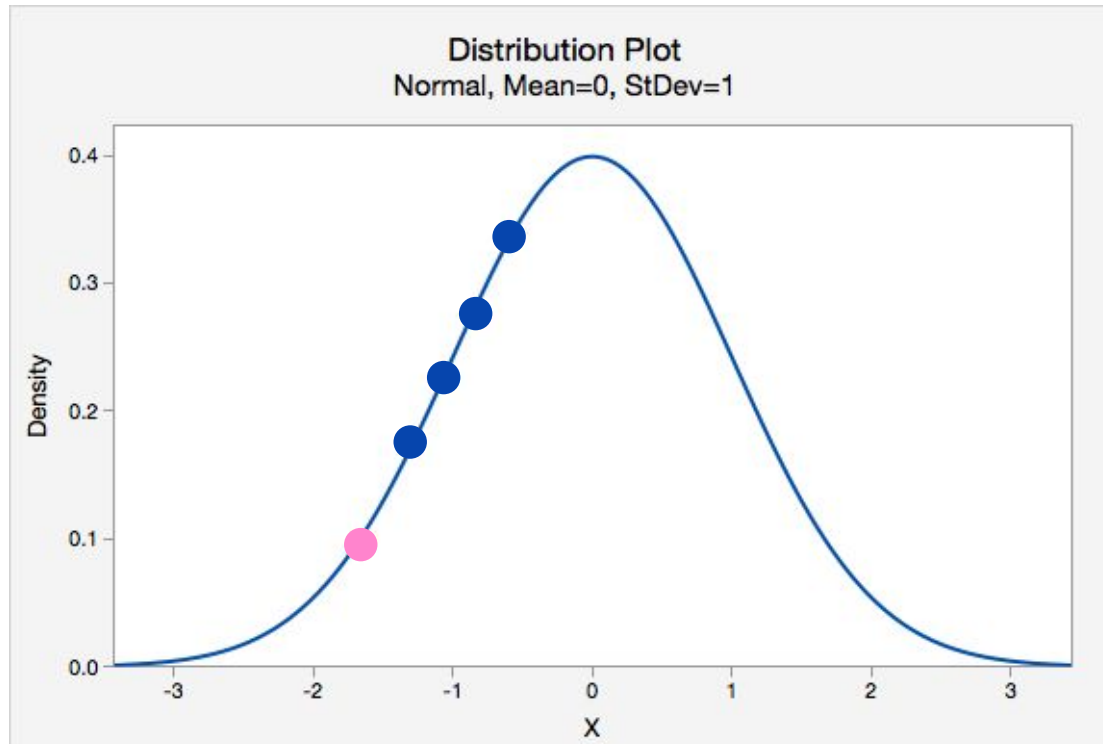


If the new point is at
lower probability,
move only
probabilistically:
move with
probability
 $\frac{p(\theta_{\text{proposed}})}{p(\theta_{\text{current}})}$

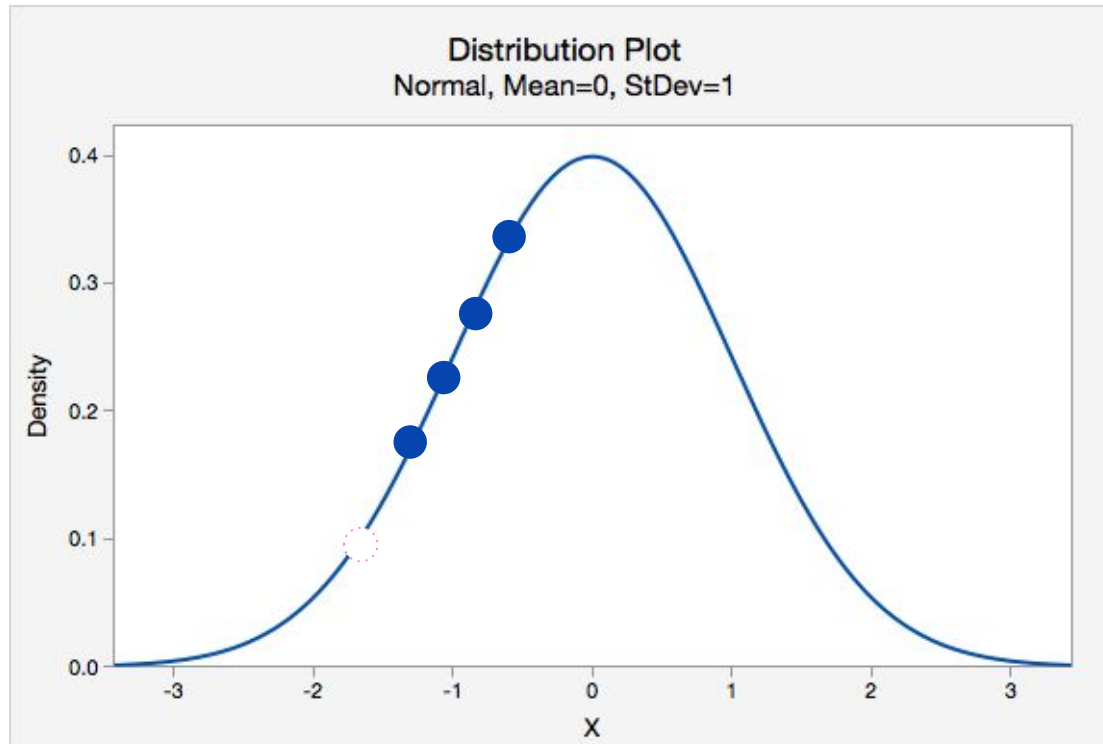
Solutions via Markov Chain Monte Carlo



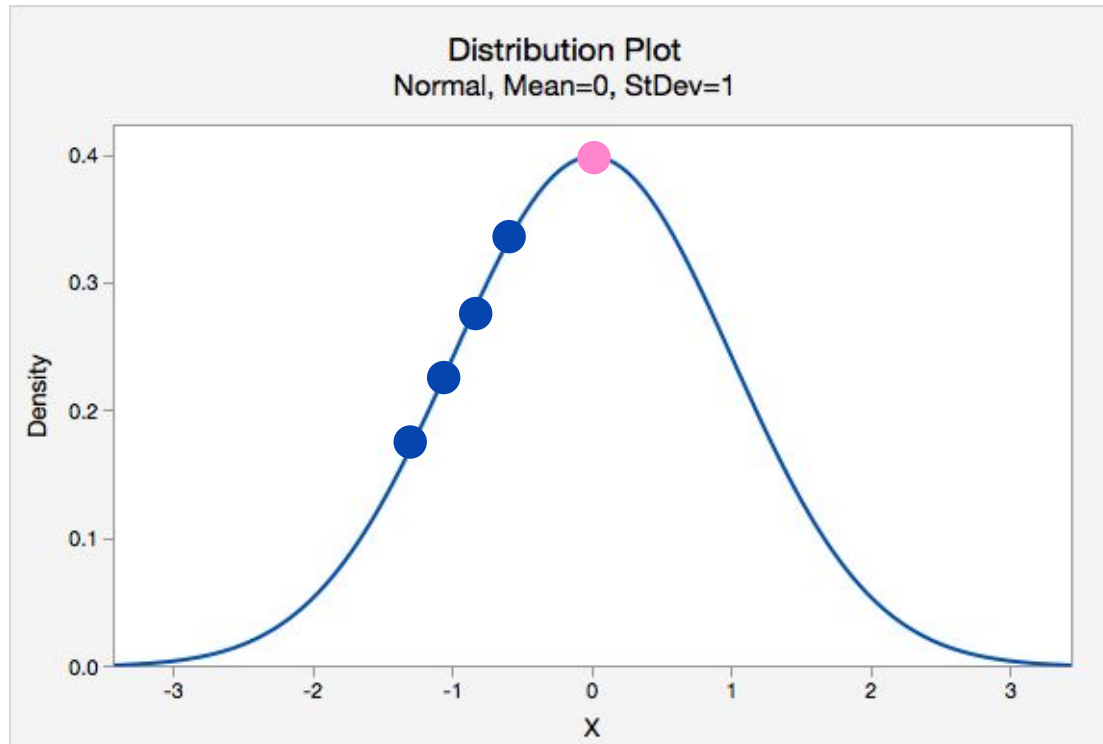
Solutions via Markov Chain Monte Carlo



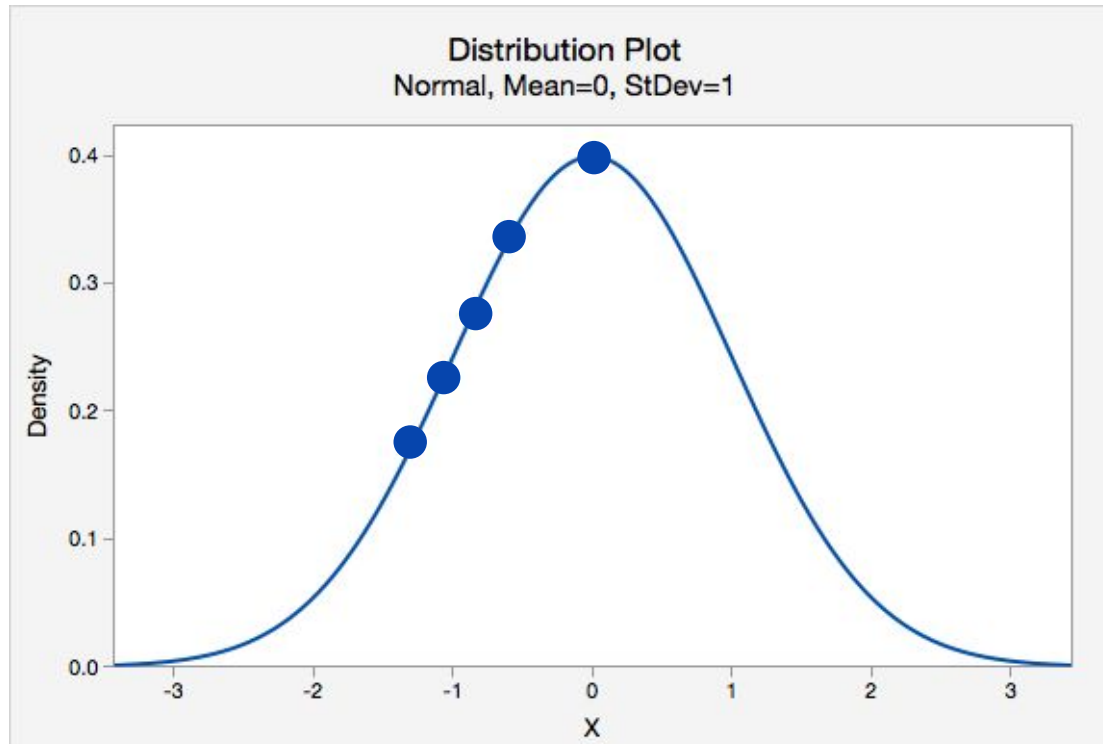
Solutions via Markov Chain Monte Carlo



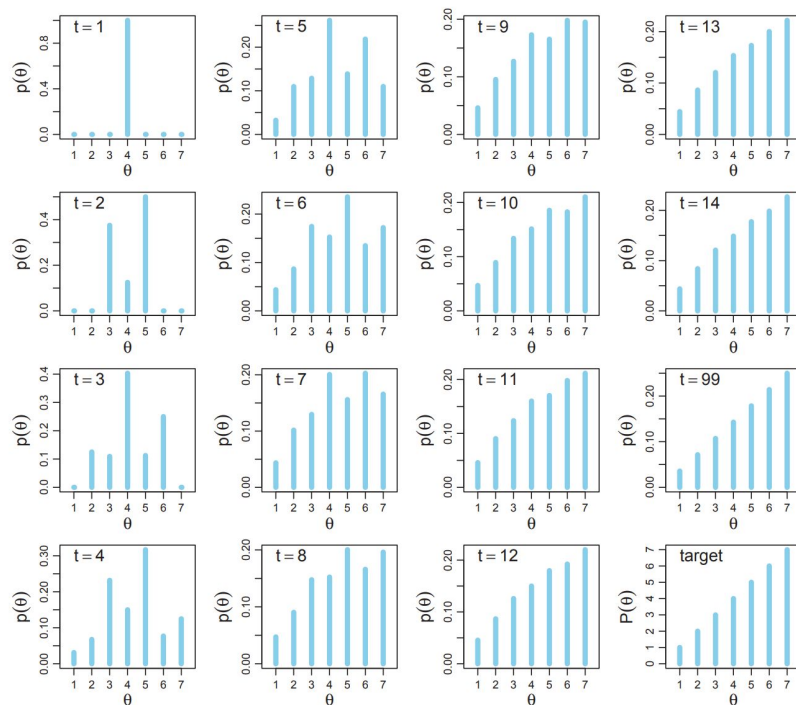
Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo



Solutions via Markov Chain Monte Carlo



Source of this example is
the wonderful book [*Doing Bayesian Data Analysis*](#) by
[*John K. Kruschke*](#)

Figure 7.3: The probability of being at position θ , as a function of time t , when a simple Metropolis algorithm is applied to the target distribution in the lower right panel. The time in each panel corresponds to the step in a random walk, an example of which is shown in Figure 7.2. The target distribution is shown in the lower right panel. Copyright © Kruschke, J. K. (2014). *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan. 2nd Edition.* Academic Press / Elsevier.

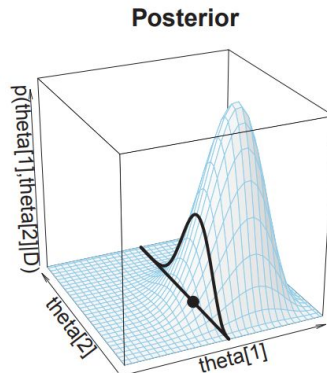
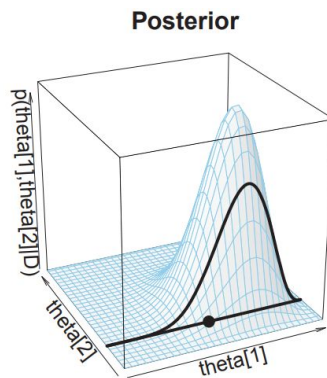
Solutions via Markov Chain Monte Carlo

- Better than trying to analytically solve the N-dimensional distributions
- The example we demonstrated is called the **Metropolis algorithm**: a random walk through parameter space that starts at a random point
- Can be used to reconstruct the shape of the distribution, not just the peak
- Do we need “burn-in” (to discard the first several hundred or so draws)?
 - Opinions differ! <http://users.stat.umn.edu/~geyer/mcmc/burn.html>
 - Instead of burn-in, use “better” starting values

Solutions via Markov Chain Monte Carlo

- Many different types, particularly **Gibbs sampling**
- Gibbs sampling: given a multivariate distribution it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution
- Gibbs sampling is more efficient than the Metropolis algorithm we saw previously. Gibbs sampling is a specialized form of the Metropolis-Hastings algorithm (itself a generalized form of the Metropolis algorithm)
- In Gibbs sampling, still a random walk with no memory
 - But, at each step select one of the model parameters
 - Choose a new value for that one parameter θ_i from the conditional probability distribution of the parameter conditioned on all the other parameter values and the data
 - This is the new position in the random walk
- Hamiltonian -- uses information from gradient to take steps
- NUTS (no u-turn sampler) -- eliminates need to set number of steps
- Downside: slow
 - Other approaches: **variational inference**, which is optimization, not simulation

Gibbs Sampling



Source of this example is
the wonderful book [*Doing Bayesian Data Analysis*](#) by
[*John K. Kruschke*](#)

Figure 7.7: Two steps in a Gibbs sampling. In the upper panel, the heavy lines show a slice through the posterior conditionalized on a particular value of θ_2 , and the large dot shows a random value of θ_1 sampled from the conditional density. The lower panel shows a random generation of a value for θ_2 , conditional on the value for θ_1 determined by the previous step. The heavy lines show a slice through the posterior at the conditional value of θ_1 , and the large dot shows the random value of θ_2 sampled from the conditional density. Copyright © Kruschke, J. K. (2014). *Doing Bayesian Data Analysis: A Tutorial with R, JAGS, and Stan*. 2nd Edition. Academic Press / Elsevier.

PyMC

- From the package [authors](#): “PyMC3 is a new open source probabilistic programming framework written in Python that uses Theano to compute gradients via automatic differentiation as well as compile probabilistic programs on-the-fly to C for increased speed”
 - Theano: Python library for expressing math in terms of tensors (multi-dimensional arrays); fast

PyMC Examples

1. Fitting a distribution to data
2. Linear regression
3. Hierarchical linear regression
 - a. Helpful for small data, especially when the number of observations per group is low (e.g. students per school)
 - b. Not efficient to estimate each group individually... but if you pool, you lose idiosyncratic group-level information

Hierarchical Linear Regression

- Related to **fixed effects** and **random effects** models in traditional metrics
- Fixed effects are constant across groups, random effects vary (pretty much)
 - E.g. If you build a model with a fixed slope but random intercept for each group, you'd have a bunch of parallel lines with different intercepts

Mixed Effects Modeling

- Canonical example is often looking at student test scores:
 - Students have different achievement test scores
 - ... but, students within a school (group) have more similar test scores than across the population
 - You could estimate a student-level model, with a school effect (μ) and student-level epsilon
 - $$Y_{ij} = \mu_j + \epsilon_{ij} \quad \epsilon_{ij} \sim N(0, \sigma^2)$$

σ^2 is how much each individual student deviates from the school mean
- We also know about the distribution of each school's mean:
$$\mu_j \sim N(\mu, \sigma_\mu^2)$$
 where μ is the overall population mean
- μ is the population param (fixed effect); σ^2 is the variance within a group (school); σ_μ^2 is the between-group variance

Mixed Effects Modeling

- Notice that μ_j was the group-level (school) fixed effect in the prior model
- Now we can add a random effect for each student where g is the group-level effect (each school is a group) $\mu_j = \mu + g_j$
- Rewrite the student-level model as

$$Y_{ij} = \mu + g_j + \epsilon_{ij}$$

- This is our mixed effects model where μ is the fixed effect, g is the group-level random effect, and epsilon is individual random effects
- We estimate the following parameters: $\mu, \sigma^2, \sigma_\mu^2, \mu_j$
- Shrinkage: each group's parameters are pulled to the population mean so robust to outliers

Hierarchical Modeling

- Hierarchical: In-between un-pooled and pooled
- Unpooled: every school has its own intercept and slope
- Pooled: single intercept and slope for entire population
- Hierarchical:
 - Intercept for each school comes from a population distribution (normal distrib) of intercepts
 - Slope for each school comes from a population distribution (normal distrib) of slopes

When do we use Bayesian inference?

- You want to incorporate prior beliefs into your model
- Little data, lots of parameters
 - If you had lots of data, you might use deep learning instead
- You'd like to report your results as uncertainty (vs. the point estimate of Frequentist methods)
- Often used in, e.g., marketing and pharma
- Things standing in the way of wider adoption:
 - Still computationally intensive
 - Hard to verify the accuracy of the model

Topics we won't cover but are important

- How to choose a prior
- How to choose a solution method (e.g. MCMC vs variational inference)
- Bayesian point estimates
- ... lots of other things! Keep exploring!

Resources to continue learning

- [*Bayesian Methods for Hackers* by Cam Davidson](#)
- [PyMC3 documentation and tutorials](#)
- [Probabilistic programming primer \(Adrian Sampson\)](#)
- [Overview of No U-Turn Sampler](#)
- [Overview of prior predictive checks](#)
- [Overview of variational inference](#)