# DATA 582: Bayesian Inference

## Lecture 5: MCMC methods
## Metropolis-Hastings

Dr. Irene Vrbik

UBCO MDS

## Introduction

- Thus far we have studied some classic problems in Bayesian Inference which can be used on a wide variety of problems.
- In the developing those conjugate models we have selected convenient prior distributions that led us to posterior distributions with a recognizable from.
- In such cases, the posterior was easy summarized since many of the quantitative measures related to the distribution (eg. mean and variance) have a closed-form solution.

- While these formulas were a simple google-search away, what we have indirectly been relying on was integrating the posterior, eg.

$$\text{mean} = \mathbb{E}[\theta] = \int \theta p(\theta \mid y) d\theta. \tag{1}$$

- Before we can evaluate the above, we need to ensure that $p(\theta \mid y)$ is a proper distribution (which also requires computing an integral!)

- For some problems, integration may be difficult (or impossible!) to compute, especially in multidimensions.

- In these situations, efficient *approximation* strategies are required . . .

- Recall that the posterior distribution is given by:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{\int p(x \mid \theta)p(\theta)d\theta} = \frac{p(x \mid \theta)p(\theta)}{p(x)}$$

- In most of our examples, we have written the posterior up to a constant of proportionality and only calculated:

$$p(\theta \mid x) \propto p(x \mid \theta)p(\theta). \tag{2}$$

- Remember, this means that $p(x \mid \theta)p(\theta)$ a *not* proper pdf that integrates to 1.

- When we do *not* recognize the functional form[1] of the right-hand side of (2) we need to work out the normalizing constant given by

$$\frac{1}{\int_{\Theta} p(x \mid \theta) h(\theta) d\theta}. \tag{3}$$

- This is typically not available in closed form, i.e. it can not be expressed analytically in terms of a finite number of standard operations.

- Today we'll see how we can use *simulation-based* techniques to perform integrations necessary to summarize the posterior density.

[1]the Bayes Rules! textbook refers to these as Kernels

- Simulation-based techniques are a central part of Bayesian inference.
- While there are a variety of approximation techniques (e.g. grid approximation BR 6.1) we will focus on a subset of simulation-based methods.
- Implementations can get somewhat sophisticated[2] the basic idea is quite simple: draw samples from the distribution of interest and use those samples to estimate characteristics of distributions.
- Given a large enough sample from our distribution, a histogram can provide essentially all the information we need about the density.

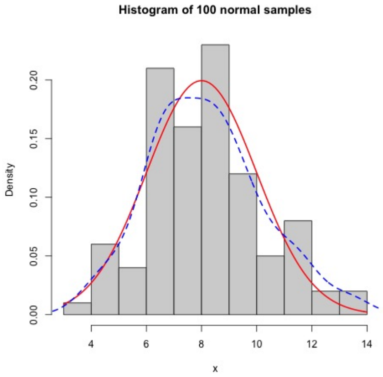
[2]there are packages and programs to help us with this

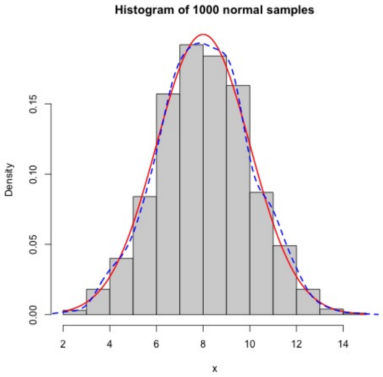
- For instance, if we didn't know the theoretical mean and variance of the normal distribution, but had a way of generating samples from it, we could approximate these quantities of interest to an arbitrary degree of precision.

- Let demo this by drawing samples from a Normal($\mu$, $\sigma^2$) in R using the the `rnorm` function ...

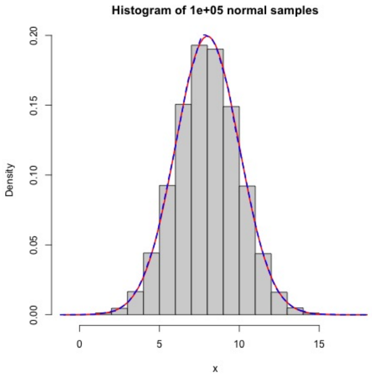
- The Kernel Density Plot[3] given in dashed blue is a smoothed version of a histogram that provides the distribution of data over a continuous interval.

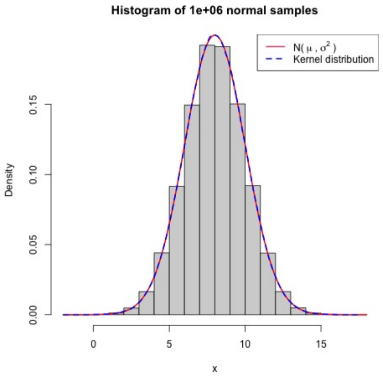
[3]this uses something called kernel smoothing–this is not related to the use of Kernel when discussing the functional form of the distributions from previous lectures.





Above gives a histogram of 100 (left) and 1000 (right) samples from a Normal($\mu$ =8, $\sigma^2$=4). The true density is plotted in red.





As the number of samples increases, we get closer and closer to approximating this curve.

- Using the `mean` and `var` functions in R we see that these estimates are pretty close to the true values, especially as the number of samples increase (recall $\mu$ =8, $\sigma^2$=4)

| Sample size | Estimate for mean | Estimate for Variance |
|---|---|---|
| 100 | 8.138345 | 4.241300 |
| 1000 | 8.055305 | 3.846822 |
| 100000 | 7.993553 | 3.988547 |
| 1000000 | 8.001834 | 3.994677 |

- We could use a similar concept to obtain other summary statistics such as percentiles – which play a crucial role in determining Bayesian credible intervals, for example.

- To obtain a sample of size $S$ we used

$$\texttt{rnorm}(\texttt{n} = S, \texttt{mean} = \mu, \texttt{sd} = \sigma).$$

- In our setting, of course, we don't know $\mu$ and $\sigma$ (if we did, we wouldn't need to be doing Bayesian inference!)
- Furthermore, if we don't recognize the form of the posterior distribution we can certainly not count on R to have an associated `r<dist>` function for drawing samples from it.
- So how *do* we from a complicated density resulting from some Bayesian computation?

- *Markov Chain Monte Carlo* (or *MCMC*) methods allow us to sample from the posterior for these non-standard cases.
- While not specific to Bayesian Statistics, MCMC methods have become the predominant computational strategy for fitting Bayesian models and have contributed greatly to the revival of Bayesian methods.
- Today we do a quick review on MCMC and note that are a large number of MCMC algorithms (there are textbooks dedicated solely to this topic[4]).

[4] see for example this one

## Markov Chain Monte Carlo

- The name *Monte Carlo* comes from the famous Monte Carlo Casino in Monaco since—like games in a casino—this approach involves the element of chance.
- Recall that *Monte Carlo Integration* is a numerical integration technique which uses random numbers.
- We say that Monte Carlo methods are *stochastic* (as opposed to *deterministic*) since the same set of parameter values and initial conditions will produce a different output each time we run it.

## Monte Carlo Integration

### Monte Carlo Integration

If $x_1, \ldots, x_n$ are idependent samples drawn from $f(x)$:

$$\mathbb{E}_f[\phi(x)] = \int \phi(x)\ f(x) dx$$

can be approximated by:

$$\mathbb{E}_f[\phi(x)] \approx \overline{\phi}_n := \frac{1}{n} \sum_{i=1}^{n} \phi(x_i) \tag{4}$$

## Monte Carlo Integration

- **Main idea**: collect a sampled sequence from $p(\theta \mid y)$, the posterior distribution denoted: $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(S)}$
- Given a large enough sample size $S$, these data will be used to provide an approximation to the posterior.
- These samples can then be used to approximate integrals of interest, like the mean and variance, for example:

$$\mathbb{E}[\theta] = \int \theta p(\theta \mid y) d\theta \approx \frac{1}{S} \sum_{1}^{S} \theta^{(i)}$$

$$\mathbb{V}[\theta] = \int (\theta - \mathbb{E}[\theta])^2 p(\theta \mid y) d\theta \approx \frac{1}{S} \sum_{1}^{S} (\theta^{(i)} - \mathbb{E}[\theta])^2$$

## Monte Carlo Integration

- We can also use this sampled sequence to estimate things like the percentiles used for defining credible intervals.

- Recall that the $s$-quantile of a distribution is the number $q_s$ such that a proportion $s$ of the values are less than or equal to $q_s$. If the distribution is the posterior, we have:

$$P(\theta \leq q_s) = \int_{-\infty}^{q_s} p(\theta \mid y) d\theta = s$$

- A natural approximation to the value is the *$s^{th}$ ordered statistic* which gives the number for which $100s\%$ of the sampled values from $p(\theta \mid y)$ fall below it.

Returning to our normal samples, we could approximate the 34$^{th}$ percentile or 0.34-quantile using the `quantile` function:

```
qs <- quantile(samp, prob = 0.34)
```

where `samp` is a vector containing our sample. Compare the empirical estimates with the true value $q_{0.34} = 7.175074$

| Sample size | Estimate for $q_{0.34}$ | $\#$ of samples $< \hat{q}_{0.34}$ |
|---|---|---|
| 100 | 7.131680 | 34 |
| 1000 | 7.198219 | 340 |
| 100,000 | 7.172377 | 34,000 |
| 1,000,000 | 7.178714 | 340,000 |

- Again, since our distribution of interest here is the normal, the true value for $q_{0.34}$ can be obtained using the `qnorm` function:

```
> qnorm(0.34, mean = mu, sd = sig)
[1] 7.175074
```

- We can verify, however, that each of these empirical percentiles do indeed have $100s\%$ of samples falling below $q_s$ as expected.

```
> sum(samp<qs)/n
0.34
```

## Markov Chains

- A *Markov chain* is a sequence of dependent random variables $\{X^{(0)}, X^{(1)}, X^{(2)}, \ldots, X^{(t)}, \ldots, X^{(S)}\}$.
- A Markov chain is generated by sampling the new state of the chain, based *only* on information regarding the current state i.e., at time $t$.
- That is we generate the new state of the chain, $X^{(t+1)}$ from a density dependent only upon $x^{(t)}$.

## Markov Chains for Bayesian Inference

- Under certain regularity conditions, Markov chains will converge to a *stationary distribution*.
- In the context of Bayesian inference we want to construct a Markov chain whose stationary distribution is our **posterior** of interest.
- In other words, if left to run long enough under these regularity conditions, once "convergence" occurs then subsequent iterates act like draws from the posterior.
- We can then use the converged chain to approximate the posterior distribution (eg. histogram) or approximate integrals (eg. mean).

## Ergodic Theorem

### Ergodic Theorem

Suppose that we have an irreducible, aperiodic, positive recurrent Markov chain with stationary distribution, $f(x)$. Then the Ergodic Theorem states that

$$\overline{\phi}_n = \frac{1}{S}\sum_{t=1}^{S}\phi(x^t) \to E_f[\phi(x)]$$

We call $\overline{\phi}_n$ the ergodic average.

- MCMC is an example of the *Monte Carlo* method, which we implement through the use of *Markov Chains*.

- In essence, these all following the same general algorithm as shown on the next slide.

- The difference between MCMC alogorithms is determined by:
    1. proposal distribution, i.e. how they propose/generate new positions, $\theta_{new}$
    2. acceptance probabilities, i.e. how they determine if the proposal is accepted

## MCMC Algorithms

0. At $t = 1$, initialize your chain at value, $\theta^t$
1. Given current position, $\theta^{(t)}$, generate/propose a new value

$$\theta_{new} \sim q(\cdot \mid \theta^{(t)})$$

2. Accept or reject $\theta_{new}$ according to some probability $\alpha$.
3. Update: $\begin{cases} \text{If accepted, move to } \theta^{(t+1)} = \theta_{new} \\ \text{If rejected, stay where we are, } \theta^{(t+1)} = \theta^{(t)} \end{cases}$
4. Set $t = t + 1$
5. Return to step 1

Repeat for a set number of iterations.

## Metropolis Algorithm

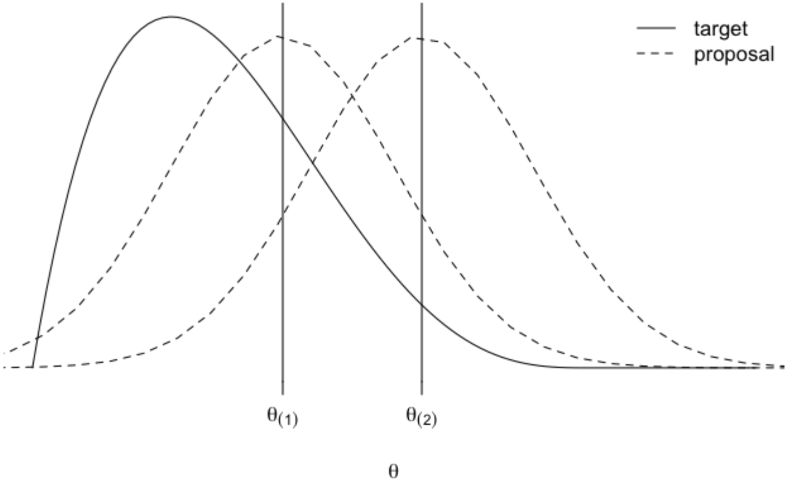
- One simple MCMC method is the *Metropolis Algorithm* [5]
- This MCMC algorithm proposes new positions according to a symmetric proposal distribution $q(\theta_{new} \mid \theta^{(t)})$.
- For example one might use:

$$\theta_{new} \sim \mathcal{N}(\theta^{(t)}, \sigma^2) \tag{5}$$

  with mean centered at the current state of our chain, $\theta^{(t)}$.

- The user sets standard deviation $\sigma$ called the proposal width.

[5] a nice summary video can be viewed here


target
proposal
$\theta_{(1)}$
$\theta_{(2)}$
$\theta$

## Proposal Distribution

- In theory we have a great deal of flexibility in our choice of $q$
- If $q$ is poorly chosen, then the number of rejections can be high, so that the efficiency of the procedure can be low.
- Choosing $q$ well is therefore key to successfully implementing a Metropolis-Hastings MCMC alogorithm.

## Acceptance Probability

- In order to determine if a new proposed value is accepted, this algorithm uses:

$$\alpha(\theta_{new} \mid \theta^{(t)}) = \min\left(\frac{f(\theta_{new})}{f(\theta_t)}, 1\right) = \min\left(r(\theta_{new}, \theta_t), 1\right) \quad (6)$$

- We then generate a random number $u \sim \mathcal{U}(0,1)$ and if

$$\begin{cases} u \leq \alpha & \text{accept move} \\ \text{otherwise} & \text{reject} \end{cases}$$

*Note: $f(x)$ is called the target density, i.e. the stationary distribution that our Markov chain will eventually converge to.*

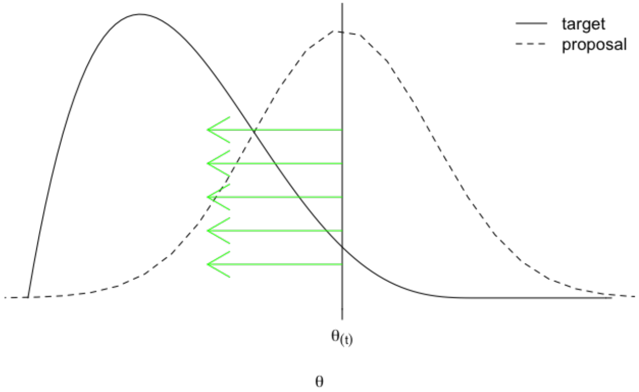
In the context of Bayesian inference $f(x)$ is the posterior $p(\theta \mid x)$. Notice that any constant w.r.t $\theta$ will cancel out!

$$r(\theta_{new}, \theta_t) = \frac{p(\theta_{new} \mid x)}{p(\theta_t \mid x)} = \frac{\frac{p(x \mid \theta_{new}) p(\theta_{new})}{\cancel{p(x)}}}{\frac{p(x \mid \theta^{(t)}) p(\theta^{(t)})}{\cancel{p(x)}}} = \frac{p(x \mid \theta_{new}) p(\theta_{new})}{p(x \mid \theta^{(t)}) p(\theta^{(t)})}$$

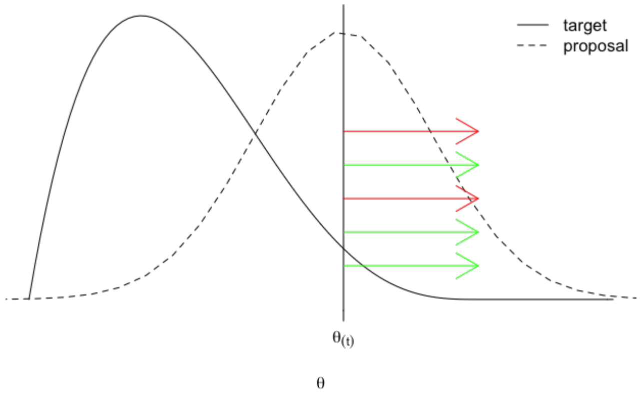
*We can generate samples from the target distribution even when the target distribution is not normalized!*

Hence this algorithm only relies on the likelihoods and the priors, both of which we can usually calculate easily.

If $p(\theta_{new} \mid x) \geq p(\theta^{(t)} \mid x)$ then we will definitely accept the proposed state, i.e. we will set $\theta^{(t+1)} = \theta_{new}$



If $p(\theta_{new} \mid x) < p(\theta^{(t)} \mid x)$ , we will move to the proposed $\theta_{new}$ with probability $\alpha(\theta_{new} \mid \theta^{(t)})$.



- Defining $\alpha$ in this way means that the chain is more likely to move towards values with higher posterior probability.

- So the idea is that we are sampling from regions having higher posterior probability more often.

- Furthermore, the time the chain spends in any state will be proportional to the posterior probability at that state.

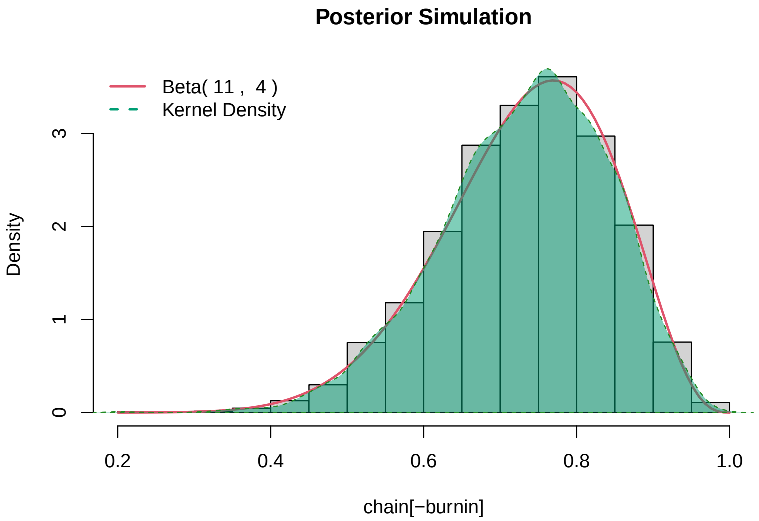
- If you haven't already, I recommend again that you watch a nice summary video here[6]

[6] this video uses a software that we won't be implementing but the explanation is great

- The resulting chain acts like a sample from the posterior distribution.

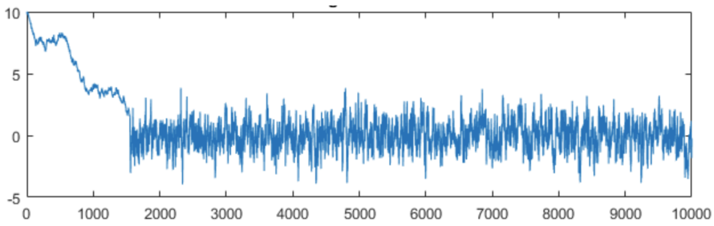
- To put another way, it serves as a result of r¡dist¿ when we don't know what `<dist>` actually looks like.

- We can use those samples to approximate the distribution (eg. histograms of kernel density plots) or estimate the mean $E[X]$


Posterior Simulation
Beta( 11 , 4 )
Kernel Density
Density
0
1
2
3
0.2
0.4
0.6
0.8
1.0
chain[−burnin]

- Remember, only after convergence do the samples that we draw from this process mimic draws from the stationary distribution.
- Hence there may be a considerable number of samples we need to throw away since they were generated before stationarity was reached.
- The preliminary iterations, during which the chain may be unrepresentative of the target posterior distribution, is called the *burn-in* period.

- *Traceplots* are a very useful in 1) checking convergence 2) determining burn-in, and 3) assessing our proposal width.
- With regards to checking convergence, a chain might have reached convergence when we see a relatively constant mean and variance in our trace plot.
- A converged well-mixed traceplot will look like a fat and hairy caterpillar.
- The meandering behaviour of the chain before it reaches this stage is known as burn-in and those samples are usually discarded.



Source: statlect.com

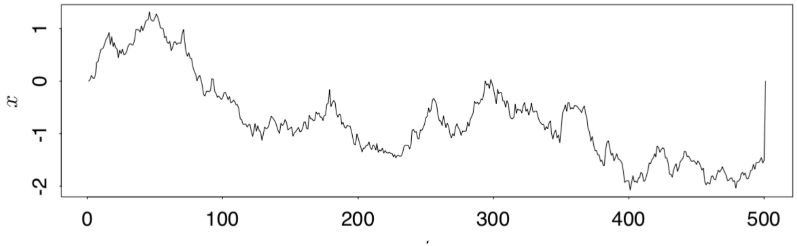
- Here the burnin is about 1500 iterations.
- After the initial burnin, this chain appears to have “achieve stationarity”, aka “reached convergence”, i.e. it has converged to its stationary distribution.
- That is to say, the sequence $\{x_{1501}, x_{1502}, \ldots, x_{10000}\}$ can be viewed as samples drawn from the target distribution.

- The proposal width[7] $\sigma$, plays a crucial roll in convergence and may need to be adjusted or *tuned* so that our chain "mixes well".
  - If $\sigma$ is too big, you'll rarely accept a new $\theta_{new}$ and you get trapped in certain states for too long.
  - If $\sigma$ is two small you'll accept $\theta_{new}$ too often. This is inefficient as it will take a long time to explore the parameter space
- Poor (resp. good) mixing $\implies$ slow (resp. quick) convergence).
- This is more an art than a science.

[7] referred to as a tuning parameter

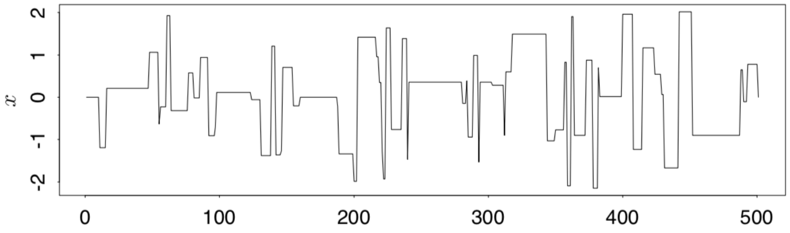
## Bad Mixing

The typical random-walk behaviour of a chain have a proposal width that is too small. Chains of this nature are said not to “explore” the parameter space efficiently.



## Bad Mixing

The typical “steps” behaviour of an chain have a proposal width that is too big. Since this Markov chain traverses through the parameter space slowly, it is said to be mixing poorly.



## Metroplis Algorithm: An example

- We return to our example for inferring a Binomial proportion.
- Aassuming a Beta($a, b$) prior on $\theta$ we know should end up with a $Beta(a + y, b + n - y)$ posterior.
- Let's pretend we didn't know this and use MCMC to estimate the posterior and compare the approximated mean and variance with the theoretical quantities of a Beta($\alpha, \beta$):

$$\mu = \frac{\alpha}{\alpha + \beta} \qquad \sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

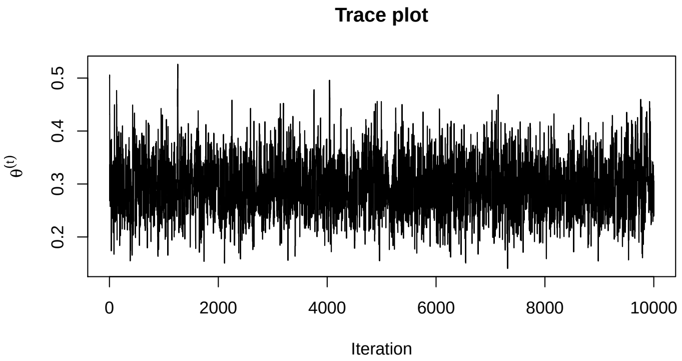
- Suppose we perform 50 random flips of a coin and observe 10 heads and ssuming a Beta(12,12) prior on $\theta$.

- The Metropolis Algorithm relies on the product of the prior and likelihood (up to a constant of proportionality) to be easily computed for any given value of $\theta$.

- As you'll see in lab, the **MetropolisAlgorithm.R**[8], defines a `for` loop through this algorithm to obtain our chain $\{\theta_S\}$ which we store in the object `chain`.

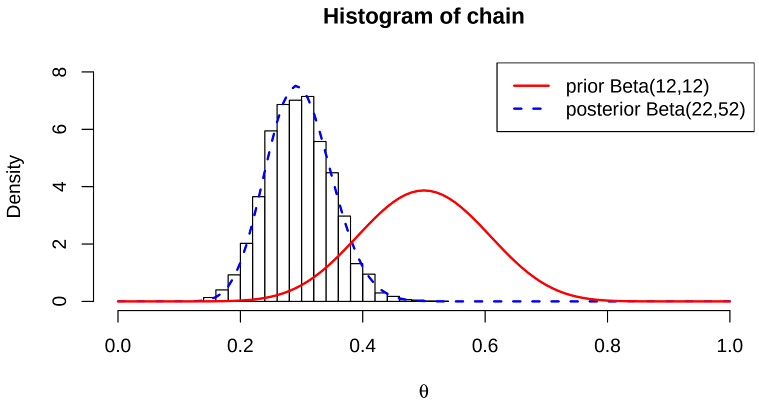
- The trace plot based on these samples is shows that we have reached stationarity with no significant burnin which will need to be removed.

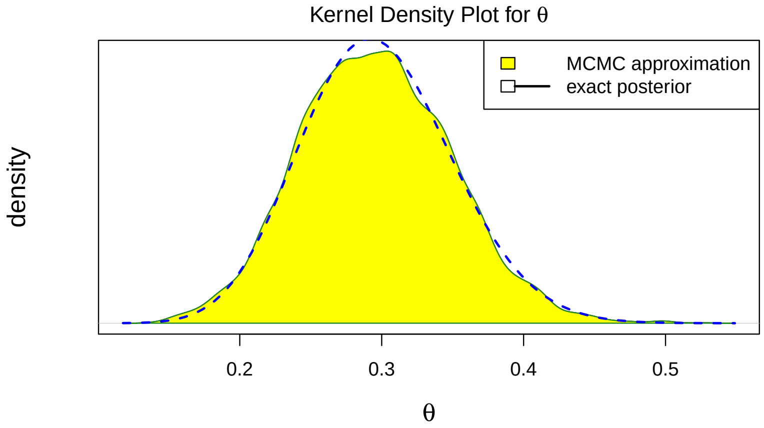
[8]the **Beta-BinomialMCMC.R** script does this on the log-scale instead



This is an example of a “good trace plot”.

- Through conjugacy we know that the posterior should be a Beta$(22, 52)$ with mean 0.297 and variance 0.003.
- On slide 44 we can compare the empirical distribution obtained by plotting the histogram of our chain with the theoretical posterior, i.e. the `debta` curve of a Beta(22,52).
- As an alternative to plotting a histogram, we can use a smoothed kernel density estimate of the posterior via the `density` function (see slide 45)


Histogram of chain
Density
θ
prior Beta(12,12)
posterior Beta(22,52)
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0


Kernel Density Plot for θ
MCMC approximation
exact posterior
density
0.2
0.3
0.4
0.5
θ

- Finally we compare the theoretical mean and variance of the posterior given by:

$$\frac{22}{22+52} = 0.297 \qquad \sqrt{\frac{2252}{(22+52)^2(22+52+1)}} = 0.003$$

with the corresponding values calculated using the Monte Carlo integration:

$$\text{Ergodic mean} = 0.296 \qquad \text{Ergodic variance} \qquad = 0.003$$

- As you can see, the results are pretty darn close!

## Metropolis Hastings Algorithm

- While this algorithm is relatively easy to understand and implement there have been improvements to it over the years.
- The Metropolis Algorithm was generalized by Hastings in 1970 to form the *Metropolis-Hastings (MH) Algorithm*
- That advantage is that they are in a sense 'intelligent' or 'dynamic' and are thus much better at dealing with very high dimensional problems.

## Metropolis Hastings Algorithm

Suppose that a Markov chain is in position $x$;

1. Propose a move to $y$ with probability $q(y \mid x)$
2. Accept move with probability $\alpha = \min\left\{1, r = \dfrac{f(y)q(x \mid y)}{f(x)q(y \mid x)}\right\}$
3. $\begin{cases} \text{If accepted, move to } y \\ \text{If rejected, stay where we are (i.e., } X^{(t+1)} = X^{(t)}\text{)} \end{cases}$
4. Return to step 1
5. Repeat for a set number of iterations

## Metropolis Hastings Algorithm

- The main difference is that the MH algorithm does not have the symmetric distribution requirement

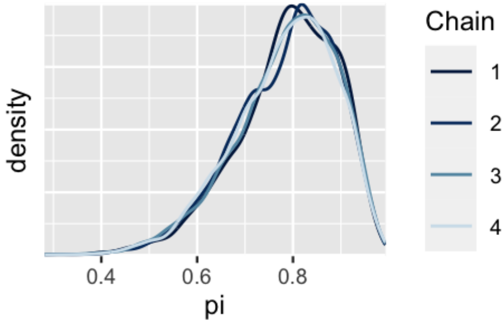
- Hence the Metropolis algorithm is a special case of MH algorithm, that is, if proposal is symmetric, i.e., $q(y \mid x) = q(x \mid y)$, the ratio is simply $r = p(y)/p(x)$ and we get the Metropolis algorithm.

- The Metropolis algorithm can be slow, especially if your initial starting point is way off target so using asymmetric proposal distributions can speed up the process.

- Only mathematical analysis can guarantee that a Markov chain has converged to its target distribution, and such analysis is generally prohibitively difficult for realistically complex models.

- Practical procedures that are commonly used to attempt to assess MCMC convergence, include things like running more than one chain for each model and examining if the all converge to the same place.

- While none of these methods offer guarantees, they can help to identify problems with MCMC samplers and provide some protection against invalid inference.



Density plot of the four parallel Markov chains for $\pi$. [BR 6.3.2]

## Comments

- While this exercise served well as a proof of concept, the power of this method is that we can use it *when we do not know the functional form of the posterior*!

- While these simple models are not too difficult to code up there are programs and packages that we can leverage to do some of the heavy-lifting for us :)