450D Political Methodology IV
TA Section 3

Vincent Bauer

October 13, 2017

Estimated time: 1hr, 15min

Goal Today

- Motivate and show the basics of MCMC.

Road map to those goals:

1. Motivation
2. Markov Chains
3. MCMC Algorithms
   - 3.1 Metropolis *
   - 3.2 Metropolis-Hastings *
   - 3.3 Gibbs sampler
   - 3.4 Hamiltonian
4. Application: Standard Normal
   - 4.1 Intuition
   - 4.2 Small step size
   - 4.3 Bigger step size
5. Briefly, next week's topics

Motivation

When we use a conjugate prior, we have two methods for showing how our priors have updated based on new information.

Motivation

When we use a conjugate prior, we have two methods for showing how our priors have updated based on new information.

- *Analytically*: we know what the mean and variance of the posterior distribution is and can state these exactly.

Motivation

When we use a conjugate prior, we have two methods for showing how our priors have updated based on new information.

- *Analytically*: we know what the mean and variance of the posterior distribution is and can state these exactly.
- *Simulation*: we have functions that allow us to draw random samples from these distributions and then compute the mean, variance, or any other quantity we are interested in.

Motivation

But what if we either don't have a conjugate prior for our data or we want to use a different prior.

Motivation

But what if we either don't have a conjugate prior for our data or we want to use a different prior. In this case, our posterior won't be a distribution that we can solve analytically or even sample from easily.

Motivation

But what if we either don't have a conjugate prior for our data or we want to use a different prior. In this case, our posterior won't be a distribution that we can solve analytically or even sample from easily.

Markov Chain Monte Carlo is a simulation technique that allows us to easily sample from any density we want to.

Motivation

But what if we either don't have a conjugate prior for our data or we want to use a different prior. In this case, our posterior won't be a distribution that we can solve analytically or even sample from easily.

Markov Chain Monte Carlo is a simulation technique that allows us to easily sample from any density we want to. Discovering these methods in the 1950s and, in the 1970s better algorithms and faster computers, made Bayesian statistics much more appealing and less constrained.

Motivation

But what if we either don't have a conjugate prior for our data or we want to use a different prior. In this case, our posterior won't be a distribution that we can solve analytically or even sample from easily.

Markov Chain Monte Carlo is a simulation technique that allows us to easily sample from any density we want to. Discovering these methods in the 1950s and, in the 1970s better algorithms and faster computers, made Bayesian statistics much more appealing and less constrained.

What is a Markov Chain?

Markov Chains

A Markov chain $\{ X^t \}$ is a sequence of *dependent* random variables:

$$\{X^t\} = X^0, X^1, \cdots, X^t$$

Markov Chains

A Markov chain $\{ X^t \}$ is a sequence of *dependent* random variables:

$$\{X^t\} = X^0, X^1, \cdots, X^t$$

such that the probability distribution of X^t given past variables depends only on X^{t-1}.

Markov Chains

A Markov chain $\{ X^t \}$ is a sequence of *dependent* random variables:

$$\{X^t\} = X^0, X^1, \cdots, X^t$$

such that the probability distribution of X^t given past variables depends only on X^{t-1}.

The conditional probability distribution is called the transition kernal, or the Markov kernel K.

$$\text{Transition distribution: } K(\theta_t|\theta_0, \cdots, \theta_{t-1}) = K(\theta_t|\theta_{t-1})$$

Markov Chains

To put this in context, for a simple random walk Markov chain, we just add standard normal noise to the previous value.

Markov Chains

To put this in context, for a simple random walk Markov chain, we just add standard normal noise to the previous value.

$$\text{Random walk: } X_{t+1} = X_t + \epsilon_t$$

Markov Chains

To put this in context, for a simple random walk Markov chain, we just add standard normal noise to the previous value.

$$\text{Random walk: } X_{t+1} = X_t + \epsilon_t$$

Then the transition density, K, would be a normal distribution centered on X_t with variance 1.

Markov Chains

To put this in context, for a simple random walk Markov chain, we just add standard normal noise to the previous value.

$$\text{Random walk: } X_{t+1} = X_t + \epsilon_t$$

Then the transition density, K, would be a normal distribution centered on X_t with variance 1.

$$\text{Random walk: } K(\theta_t|\theta_{t-1}) = N(X_t, 1)$$

Markov Chains

Markov chains are by construction *stationary probability distributions* which means that they have the following properties:

Markov Chains

Markov chains are by construction *stationary probability distributions* which means that they have the following properties:

1. *Irreducibility*: No matter the starting value, the sequence has positive probability of reaching any location. A sufficient condition is that $K > 0$ everywhere.

Markov Chains

Markov chains are by construction *stationary probability distributions* which means that they have the following properties:

1. *Irreducibility*: No matter the starting value, the sequence has positive probability of reaching any location. A sufficient condition is that $K > 0$ everywhere.
2. *Recurrent*: the chain will return to any arbitrary location an infinite number of times

Markov Chains

Markov chains are by construction *stationary probability distributions* which means that they have the following properties:

1. *Irreducibility*: No matter the starting value, the sequence has positive probability of reaching any location. A sufficient condition is that $K > 0$ everywhere.
2. *Recurrent*: the chain will return to any arbitrary location an infinite number of times

I presume Doug will talk more about these conditions.

MCMC

The goal of *MCMC* is to create a Markov Chain of the posterior distribution.

MCMC

The goal of *MCMC* is to create a Markov Chain of the posterior distribution.

- MCMC differs from conventional Monte Carlo methods (like taking an integral) because successive sampled parameters are (purposefully) autocorrelated.

MCMC

The goal of *MCMC* is to create a Markov Chain of the posterior distribution.

- MCMC differs from conventional Monte Carlo methods (like taking an integral) because successive sampled parameters are (purposefully) autocorrelated.
- This means that our guesses get sequentially better but also that we need larger sample sizes.

MCMC

The goal of *MCMC* is to create a Markov Chain of the posterior distribution.

- MCMC differs from conventional Monte Carlo methods (like taking an integral) because successive sampled parameters are (purposefully) autocorrelated.
- This means that our guesses get sequentially better but also that we need larger sample sizes.
- We start with initial parameter values θ_0, and then generate a correlated sequence of sampled values $\theta_t, t = 1, 2, 3, \cdots$ where updated values θ_t are drawn from a transition distribution.

MCMC

The goal of *MCMC* is to create a Markov Chain of the posterior distribution.

- MCMC differs from conventional Monte Carlo methods (like taking an integral) because successive sampled parameters are (purposefully) autocorrelated.
- This means that our guesses get sequentially better but also that we need larger sample sizes.
- We start with initial parameter values θ_0, and then generate a correlated sequence of sampled values $\theta_t, t = 1, 2, 3, \cdots$ where updated values θ_t are drawn from a transition distribution.
- This transition distribution is Markovian, which means that it only depends on θ_{t-1} and not any previous values of θ beyond that.

MCMC

There are some conditions on this transition distribution in order to ensure that it converges to the posterior density $p(\theta|y)$.

MCMC

There are some conditions on this transition distribution in order to ensure that it converges to the posterior density $p(\theta|y)$.

If these conditions hold, then the sampled parameters beyond a burn-in phase can be treated as random samples from the target density, which is the posterior density.

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple)

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical.

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple)

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling*

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling* This is a special case of the MH algorithm where each component of is updated sequentially.

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling* This is a special case of the MH algorithm where each component of is updated sequentially. It allows us to break high dimensional target distributions, which would be hard approximate with the MH algorithm, into easier small-dimensional problems.

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling* This is a special case of the MH algorithm where each component of is updated sequentially. It allows us to break high dimensional target distributions, which would be hard approximate with the MH algorithm, into easier small-dimensional problems.
- *Hamiltonian Monte Carlo* (most complicated)

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling* This is a special case of the MH algorithm where each component of is updated sequentially. It allows us to break high dimensional target distributions, which would be hard approximate with the MH algorithm, into easier small-dimensional problems.
- *Hamiltonian Monte Carlo* (most complicated) This is what stan does,

MCMC Algorithms

There are four main algorithms for carrying out MCMC.

- *Metropolis algorithm* (most simple) Nicholas Metropolis laid out the foundations in 1953 but it requires that the proposal distribution is symmetrical. (I will explain what that means).
- *Metropolis-Hastings algorithm* (pretty simple) Wilfred Hastings discovered how to generalize the Metropolis algorithm to use a non-symmetric proposal.
- *Gibbs sampling* This is a special case of the MH algorithm where each component of is updated sequentially. It allows us to break high dimensional target distributions, which would be hard approximate with the MH algorithm, into easier small-dimensional problems.
- *Hamiltonian Monte Carlo* (most complicated) This is what stan does, I do not really understand it.

Metropolis algorithm

The Metropolis algorithm is a set of rule for how to randomly traverse a parameter space.

Metropolis algorithm

The Metropolis algorithm is a set of rule for how to randomly traverse a parameter space. It has the necessary properties to ensure that our estimates converge to (create a Markov chain of) the posterior density.

1. Sample a new estimate, θ^*, from the candidate (proposal/jump) distribution $J_t(\theta^*|\theta^{t-1})$. This proposal distribution should be symmetric, such as a Normal, t, or uniform density.

Metropolis algorithm

The Metropolis algorithm is a set of rule for how to randomly traverse a parameter space. It has the necessary properties to ensure that our estimates converge to (create a Markov chain of) the posterior density.

1. Sample a new estimate, θ^*, from the candidate (proposal/jump) distribution $J_t(\theta^*|\theta^{t-1})$. This proposal distribution should be symmetric, such as a Normal, t, or uniform density.
2. The acceptance probability for this candidate is

$$\alpha = min(1, \underbrace{\frac{\pi(\theta^*)}{\pi(\theta_t)}}_{\text{posterior}}) = min(1, \underbrace{\frac{p(\theta^*|y)}{p(\theta_t|y)}}_{\text{posterior}})$$

Metropolis algorithm

The Metropolis algorithm is a set of rule for how to randomly traverse a parameter space. It has the necessary properties to ensure that our estimates converge to (create a Markov chain of) the posterior density.

1. Sample a new estimate, θ^*, from the candidate (proposal/jump) distribution $J_t(\theta^*|\theta^{t-1})$. This proposal distribution should be symmetric, such as a Normal, t, or uniform density.
2. The acceptance probability for this candidate is

$$\alpha = min(1, \underbrace{\frac{\pi(\theta^*)}{\pi(\theta_t)}}_{\text{posterior}}) = min(1, \underbrace{\frac{p(\theta^*|y)}{p(\theta_t|y)}}_{\text{posterior}})$$

3. Update θ^t to θ^* with probability α, otherwise keep θ^{t-1}.

Metropolis algorithm

The implication is that if a candidate point increases the posteriror density, then it is accepted with probability 1. If it does not increase it, then it is accepted with probablity r $= \frac{\pi(\theta^*)}{\pi(\theta_t)}$.

Metropolis algorithm

The implication is that if a candidate point increases the posteriror density, then it is accepted with probability 1. If it does not increase it, then it is accepted with probablity r $= \frac{\pi(\theta^*)}{\pi(\theta_t)}$.

The reason why this algorithm is so powerful is that the proposal distribution can have any form and the invariant distribution of the resulting Markov Chain will still be the desired posterior distribution, $p(\theta|y)$.

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling.

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling. If the proposal density is symmetric (i.e. $q(\theta^*|\theta_t) = q(\theta_t|\theta^*)$), then the MH algorithm reduces to the Metropolis algorithm.

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling. If the proposal density is symmetric (i.e. $q(\theta^*|\theta_t) = q(\theta_t|\theta^*)$), then the MH algorithm reduces to the Metropolis algorithm.

The MH chain is updated with:

$$\alpha = min(1, \frac{p(\theta^*|y) \cdot q(\theta^t|\theta^*)}{p(\theta_t|y) \cdot q(\theta^*|\theta_t)}$$

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling. If the proposal density is symmetric (i.e. $q(\theta^*|\theta_t) = q(\theta_t|\theta^*)$), then the MH algorithm reduces to the Metropolis algorithm.

The MH chain is updated with:

$$\alpha = min(1, \frac{p(\theta^*|y) \cdot q(\theta^t|\theta^*)}{p(\theta_t|y) \cdot q(\theta^*|\theta_t)}$$

Where q is the proposal density and may now be non-symmetric (which was not allowed in the Metropolis algorithm).

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling. If the proposal density is symmetric (i.e. $q(\theta^*|\theta_t) = q(\theta_t|\theta^*)$), then the MH algorithm reduces to the Metropolis algorithm.

The MH chain is updated with:

$$\alpha = min(1, \frac{p(\theta^*|y) \cdot q(\theta^t|\theta^*)}{p(\theta_t|y) \cdot q(\theta^*|\theta_t)}$$

Where q is the proposal density and may now be non-symmetric (which was not allowed in the Metropolis algorithm). This means that $q(\theta^*|\theta_t) \neq q(\theta_t|\theta^*)$ in all cases.

Metropolis-Hastings algorithm

The MH algorithm is a generalization of Metropolis sampling. If the proposal density is symmetric (i.e. $q(\theta^*|\theta_t) = q(\theta_t|\theta^*)$), then the MH algorithm reduces to the Metropolis algorithm.

The MH chain is updated with:

$$\alpha = min(1, \frac{p(\theta^*|y) \cdot q(\theta^t|\theta^*)}{p(\theta_t|y) \cdot q(\theta^*|\theta_t)}$$

Where q is the proposal density and may now be non-symmetric (which was not allowed in the Metropolis algorithm). This means that $q(\theta^*|\theta_t) \neq q(\theta_t|\theta^*)$ in all cases.

These are equivalent to the probability of moving forward, and moving backwards, respectively.

Okay, that was all very theoretical.

Okay, that was all very theoretical. It will hopefully become much clearer in the following example.

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution.

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator.

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want
2. Add random noise from a uniform number generator, `runif()`, call this updated point x^*

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want
2. Add random noise from a uniform number generator, `runif()`, call this updated point x^*
3. Evaluate whether x_t or x^* is more likely to have come from a standard normal

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want
2. Add random noise from a uniform number generator, `runif()`, call this updated point x^*
3. Evaluate whether x_t or x^* is more likely to have come from a standard normal
4. If x^* is more likely than x_t to have come from a standard normal, always accept this update

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want
2. Add random noise from a uniform number generator, `runif()`, call this updated point x^*
3. Evaluate whether x_t or x^* is more likely to have come from a standard normal
4. If x^* is more likely than x_t to have come from a standard normal, always accept this update
5. If x^* is less likely than x_t, accept this update with probability equal to the ratio of their likelihoods

Drawing from a Standard Normal: Intuition

Our goal in this example is to draw values from a standard normal distribution. But, instead of drawing from a `rnorm()` in `R` we will only draw values from a uniform number generator. Whhhaaaat?

Metropolis algorithm:

1. Start with a random point x_t with whatever value we want
2. Add random noise from a uniform number generator, `runif()`, call this updated point x^*
3. Evaluate whether x_t or x^* is more likely to have come from a standard normal
4. If x^* is more likely than x_t to have come from a standard normal, always accept this update
5. If x^* is less likely than x_t, accept this update with probability equal to the ratio of their likelihoods
6. Update x_t based on the rules above and restart at 2

Drawing from a Standard Normal: Code

```
metro.sampler<-function (n, alpha, init){
  x <- init
  out <- c()  #a vector to hold each value of x
  out[1] <- x #store current value

  #now iterate
  for (i in 2:n) {
    can <- x + runif(1, -alpha, alpha) #step size
    aprob <- min(1, dnorm(can)/dnorm(x)) #prob of update
    x <- ifelse(runif(1) < aprob, can, x) #update x
    out[i] <- x #store current value
  }

  return(out)
}
```

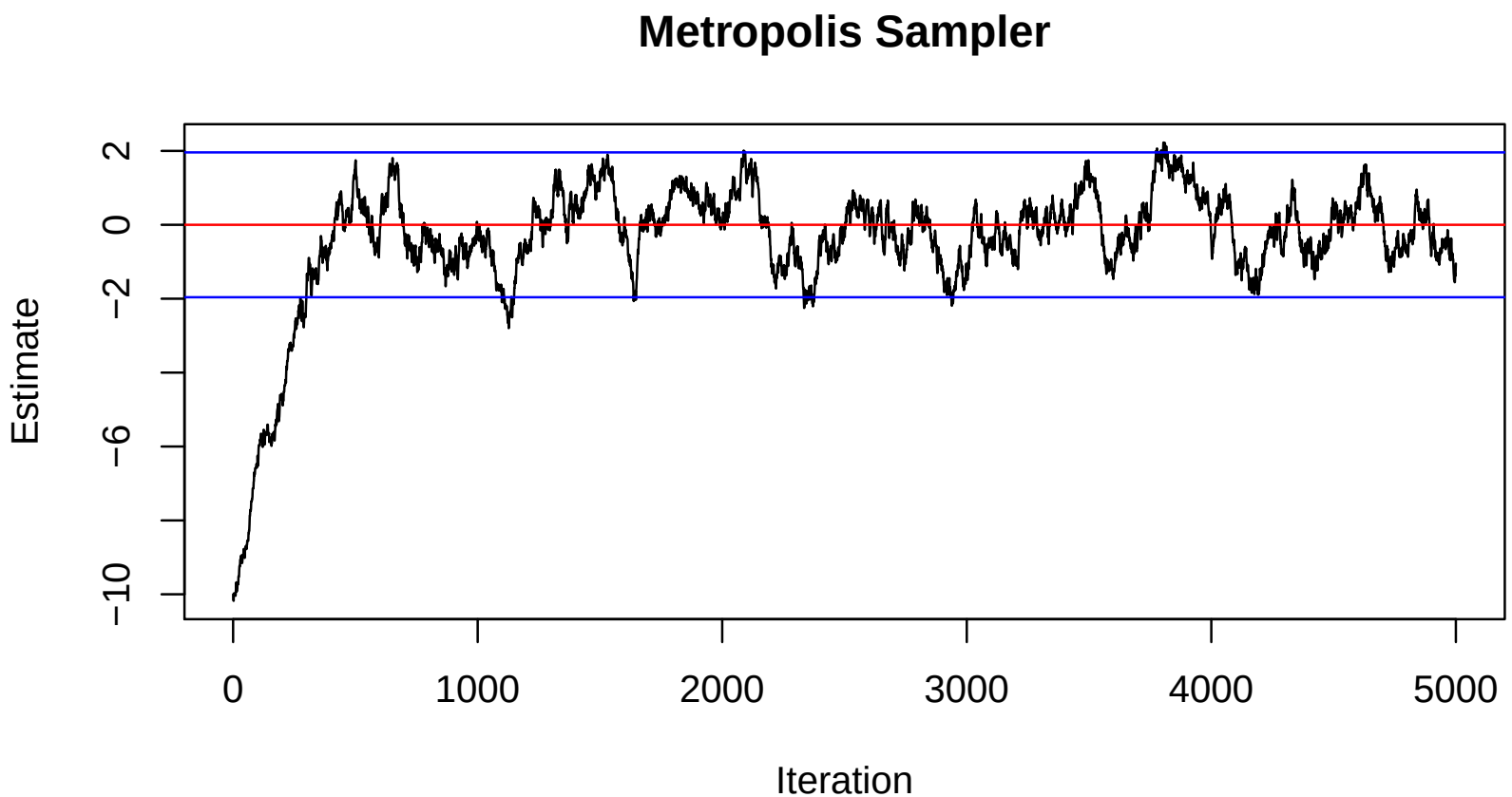
Drawing from a Standard Normal: Code

```
metro.sampler<-function (n, alpha, init){
  x <- init
  out <- c()  #a vector to hold each value of x
  out[1] <- x #store current value

  #now iterate
  for (i in 2:n) {
    can <- x + runif(1, -alpha, alpha) #step size
    aprob <- min(1, dnorm(can)/dnorm(x)) #prob of update
    x <- ifelse(runif(1) < aprob, can, x) #update x
    out[i] <- x #store current value
  }

  return(out)
}
```

We could replace a standard normal with any other distribution that we can write the PDF for.

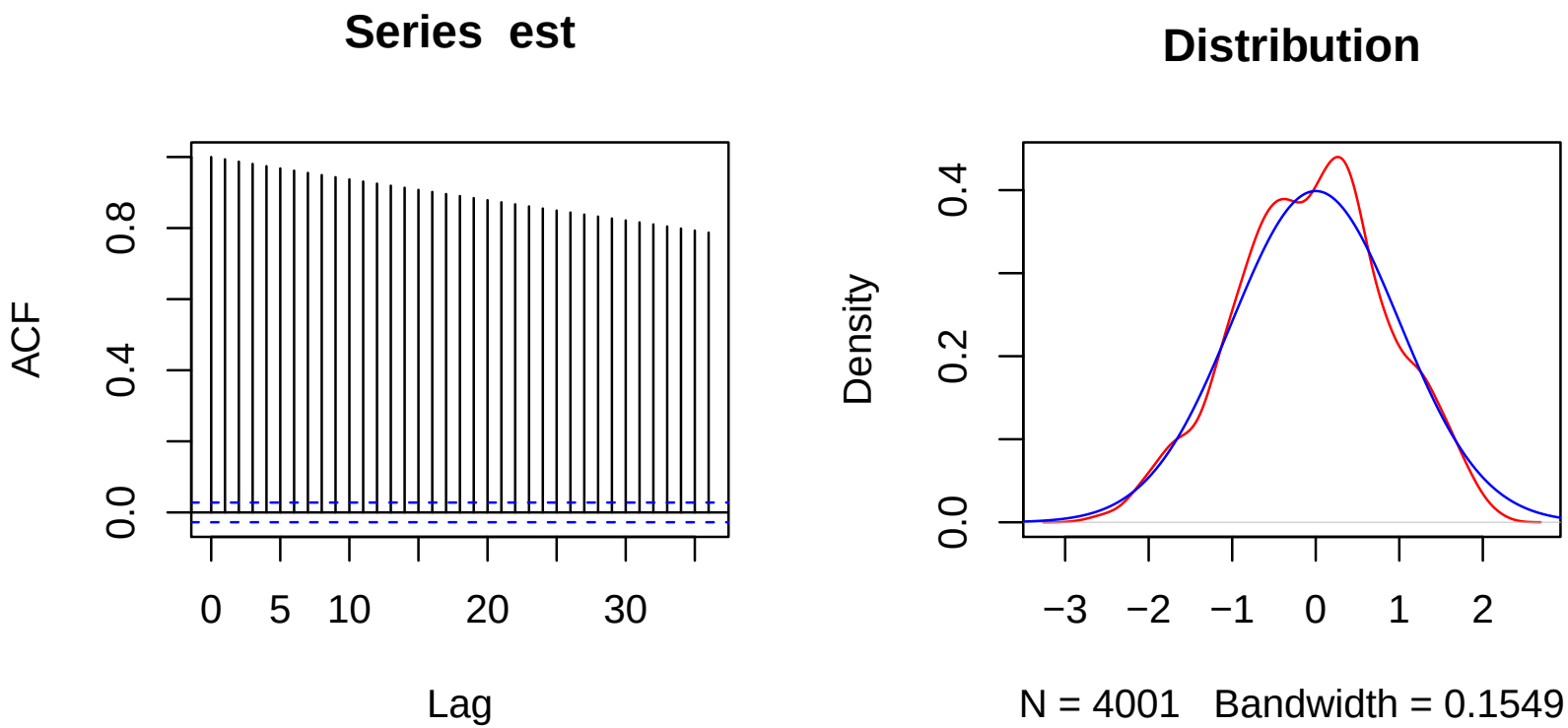
Drawing from a Standard Normal: Implement

```
m <- 5000
est <- metro.sampler(m, alpha=.25, init=-10)
plot(x=1:m, y=est, type="l", main="Metropolis Sampler", xlab="Iteration
abline(h=0, col="red")
abline(h=1.96, col="blue")
abline(h=-1.96, col="blue")
```



Drawing from a Standard Normal: Diagnostics

```
par(mfrow=c(1,2))
acf(est)
plot(density(est[1000:m]), main="Distribution", col="red")
x <- seq(-5, 5, .01)
lines(x=x, y=dnorm(x), col="blue")
```



Notice that I'm dropping the first 1000 samples, *burn-in*.

Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.

Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.
- Once it reaches the target, the sampler stays within the probability bounds for the standard normal.

Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.
- Once it reaches the target, the sampler stays within the probability bounds for the standard normal.
- Our estimates show a large armount of auto-correlation, bigger steps would probably be better.

Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.
- Once it reaches the target, the sampler stays within the probability bounds for the standard normal.
- Our estimates show a large armount of auto-correlation, bigger steps would probably be better.
- As a result of this autocorrelation, the distribution of our estimates is not quite right.

Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.
- Once it reaches the target, the sampler stays within the probability bounds for the standard normal.
- Our estimates show a large armount of auto-correlation, bigger steps would probably be better.
- As a result of this autocorrelation, the distribution of our estimates is not quite right.
- If we got the right distribution, we could say anything about it, not just the mean.

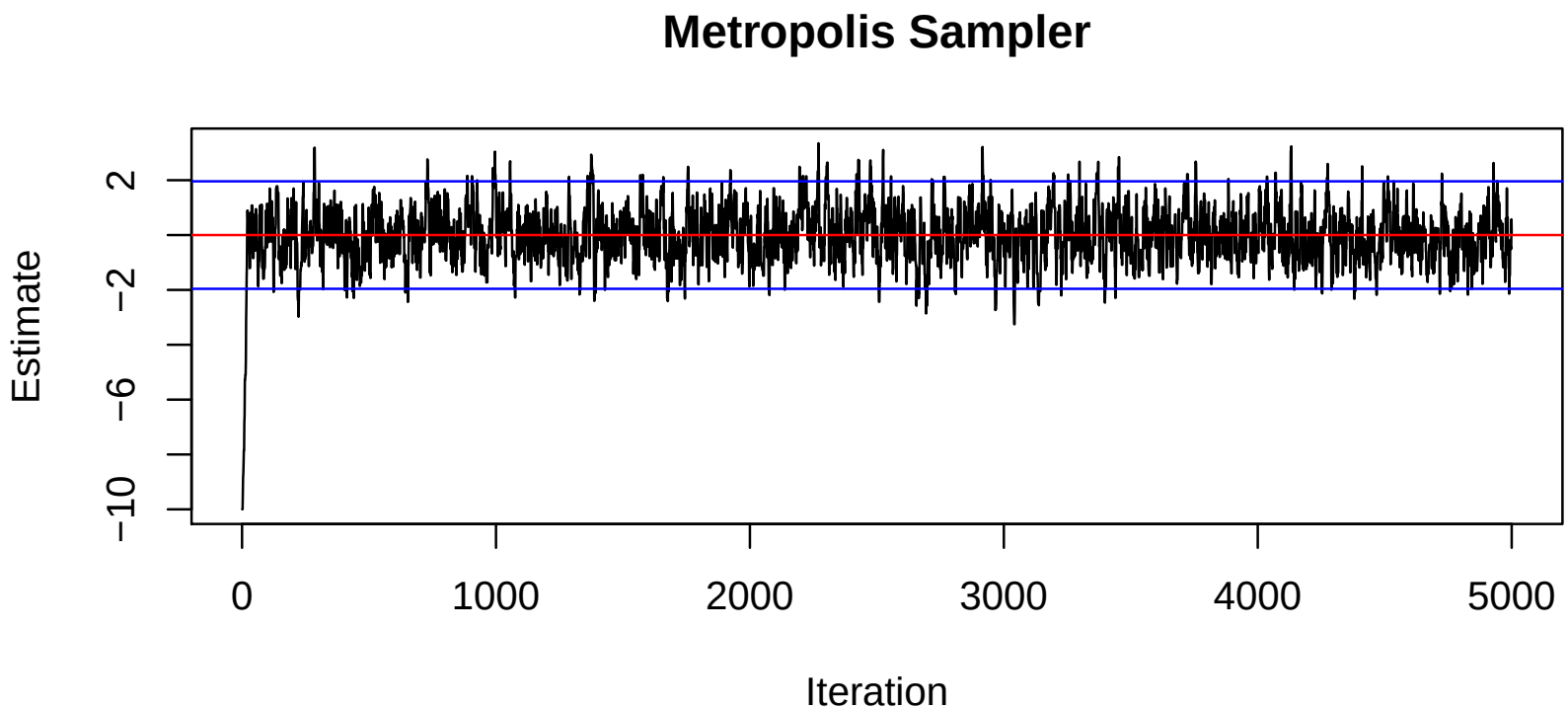
Drawing from a Standard Normal: Takeaways

- Our sampler quickly converges to the target distribution, but not immediately and not monotonoically.
- Once it reaches the target, the sampler stays within the probability bounds for the standard normal.
- Our estimates show a large armount of auto-correlation, bigger steps would probably be better.
- As a result of this autocorrelation, the distribution of our estimates is not quite right.
- If we got the right distribution, we could say anything about it, not just the mean.
- Side note: some autocorrelation is actually helpful, we could have drawn similar values with a simple method called the "Accept-Reject" method which will not have autocorrelation but we will need more samples.

Drawing from a Standard Normal: Bigger steps

Exactly the same code but I changed the step size rom .25 to 1.5.

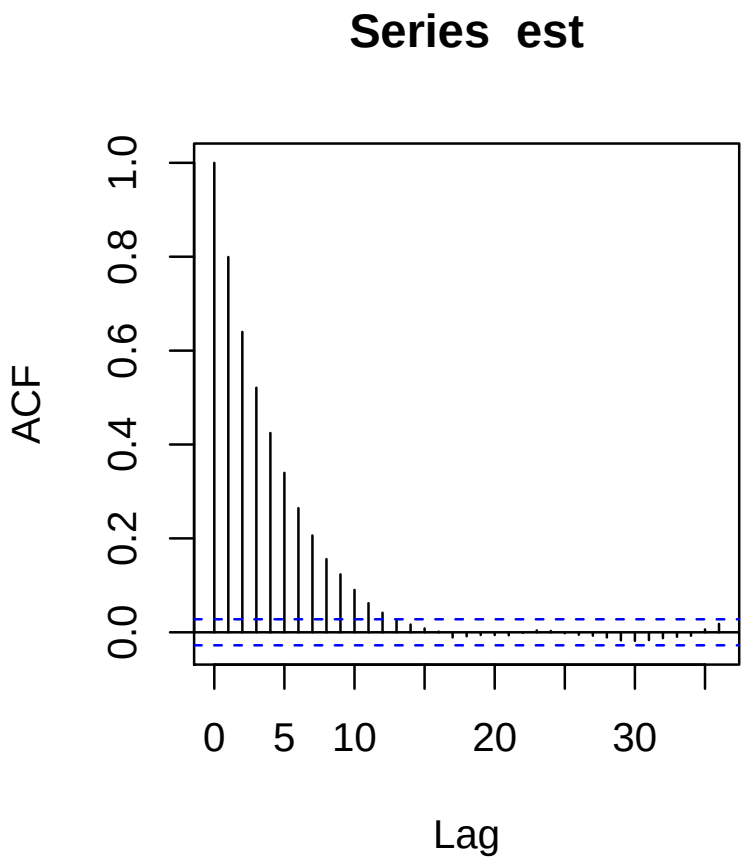
```
est <- metro.sampler(m, alpha=1.5, init=-10)
plot(x=1:m, y=est, type="l", main="Metropolis Sampler", xlab="Iteration
abline(h=0, col="red"); abline(h=1.96, col="blue"); abline(h=-1.96, col
```



This plot looks much better, you want a "caterpillar" effect.

Drawing from a Standard Normal: Bigger steps

```
par(mfrow=c(1,2))
acf(est)
plot(density(est[1000:m]), main="Distribution")
x <- seq(-5, 5, .01)
lines(x=x, y=dnorm(x), col="blue")
```



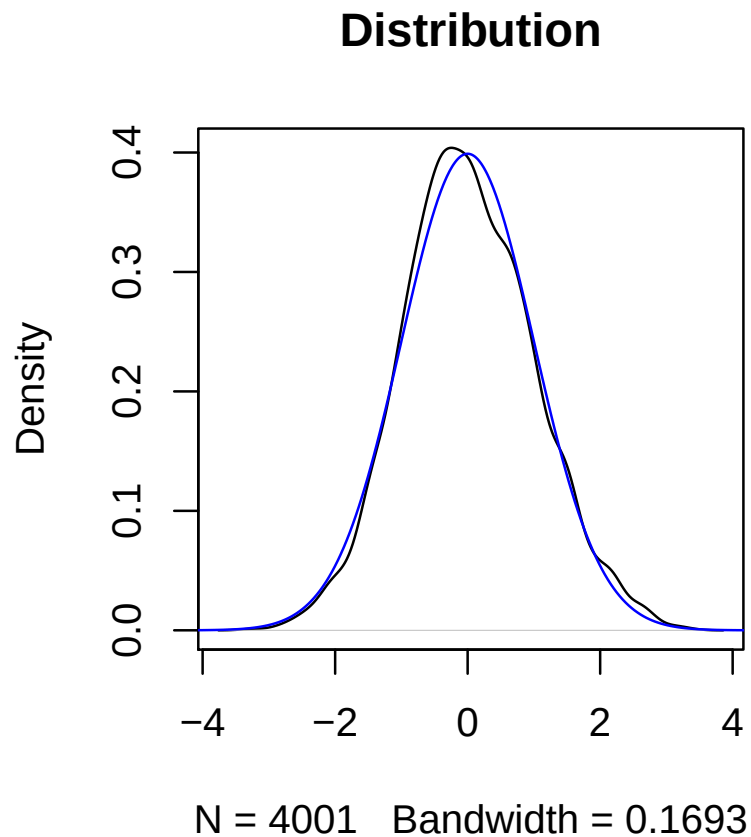
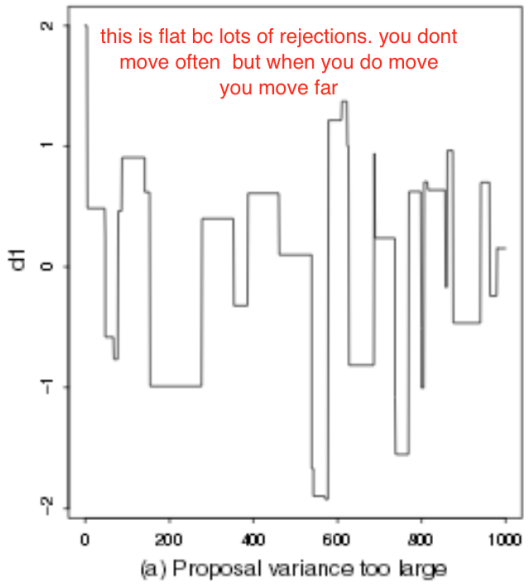
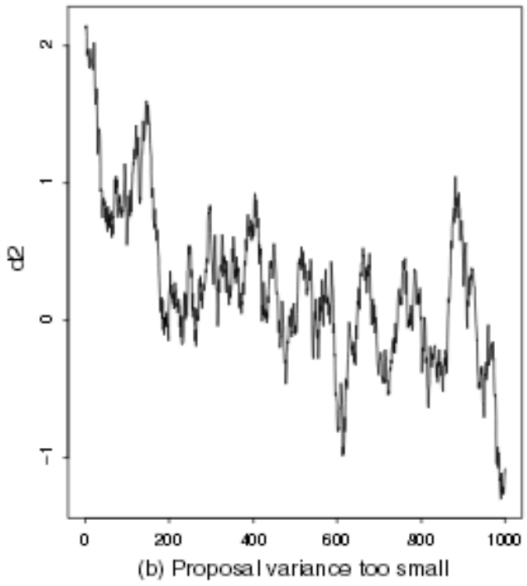


Figure: Step size and autocorrelation

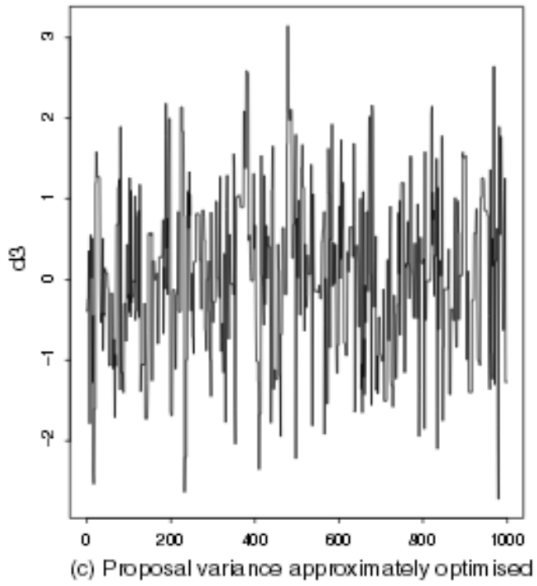
too big!



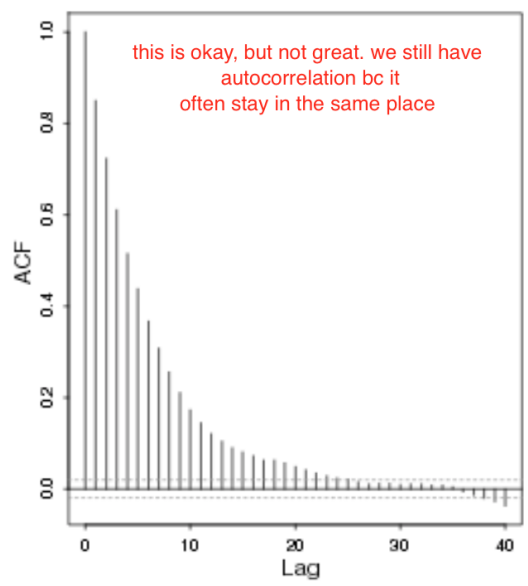
too small!

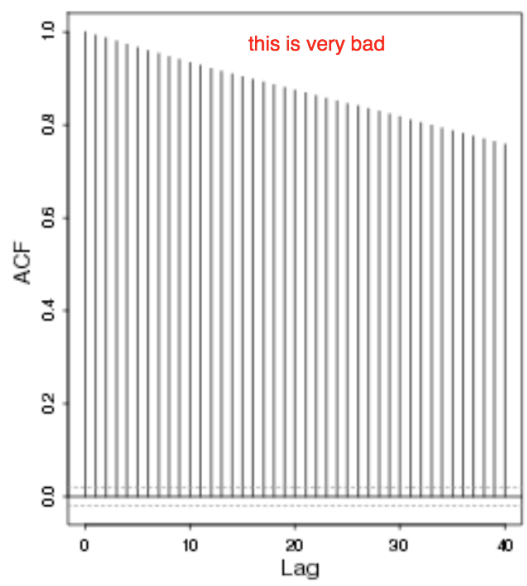


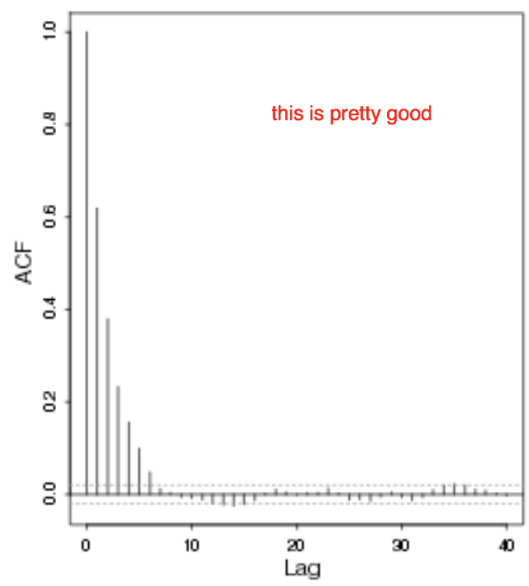
just right



these are autocorrelations between drags, want it to decrease quickly







Next week

Bayesian Poisson regression from scratch

Next week

Bayesian Poisson regression from scratch (and also not from scratch).