More on MCMC diagnostics

Bayesian Data Analysis

Steve Buyske

More on MCMC diagnostics

- Because computational Bayes relies on MCMC processes, unlike say, a least squares fit, we can't assume that the model fit even worked.
- If the chains run long enough, the MCMC sample will converge to the target distribution, so the question is have they run long enough?

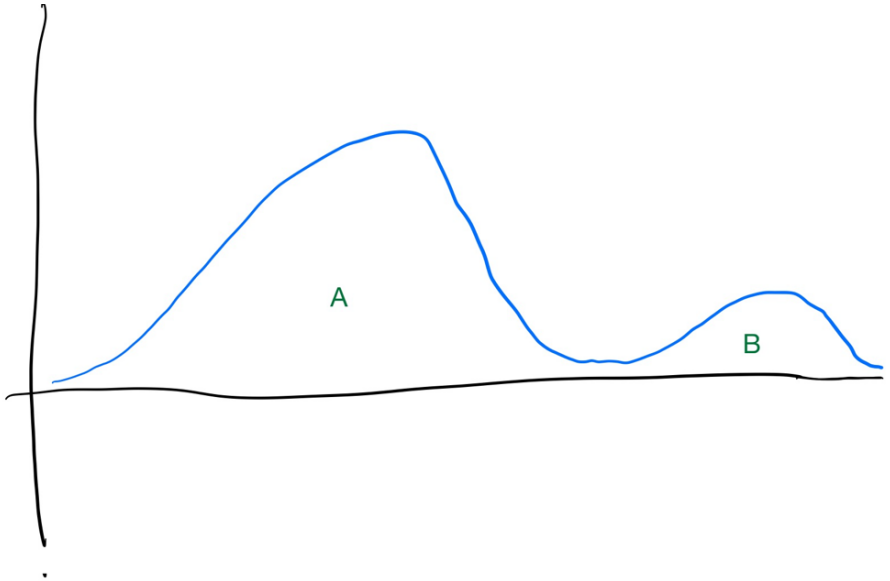
More on MCMC diagnostics

- Because computational Bayes relies on MCMC processes, unlike say, a least squares fit, we can't assume that the model fit even worked.
- If the chains run long enough, the MCMC sample will converge to the target distribution, so the question is have they run long enough?
- We can get information from observing the chains themselves.

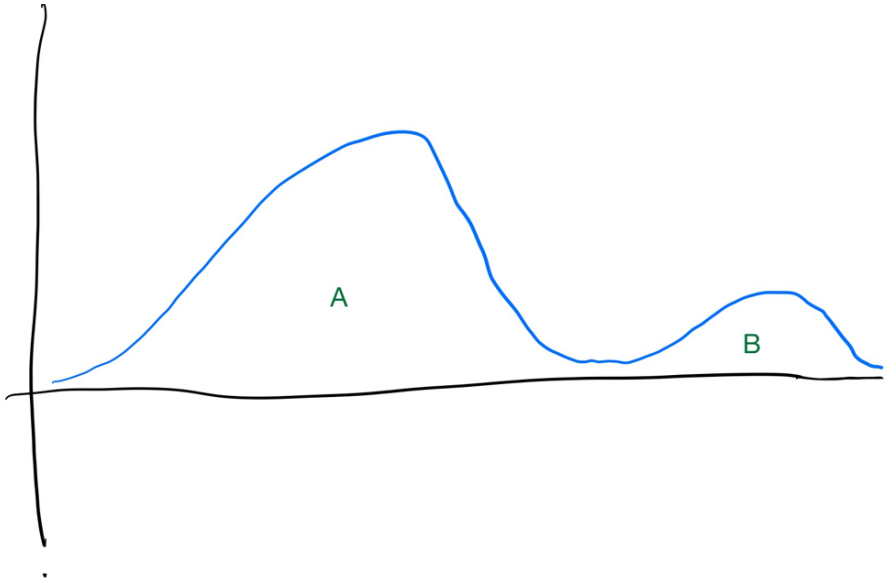
- Potential problem 1: the chains are not sampling from the full distribution
  - Maybe they are moving about really slowly
  - Maybe they are having trouble hoping across a “barrier” of low probability
- Potential problem 2: the effective sample size is too small
  - Draws from an MCMC process like the Metropolis algorithm are not independent—there’s some serial correlation.
  - If that serial correlation is high, you may have 4,000 samples from your chain, but it only has the same information as maybe 1,000 independent samples.

Chains not sampling from the full distribution

- For the distribution below, a chain starting in region A might take a long time before it jumped to region B.
  - Remember we don't know what the distribution looks like, just what samples we are drawing from it.



- By running multiple chains, with random starting positions,
  - we are more likely to have some chains start in A and some chains start in B
  - no matter what, though, we want each chain to run long enough that it moves back and forth between them many times.
  - the critical point is that *by running multiple chains, we are more likely to notice that there may be a problem*.



Chain convergence criteria

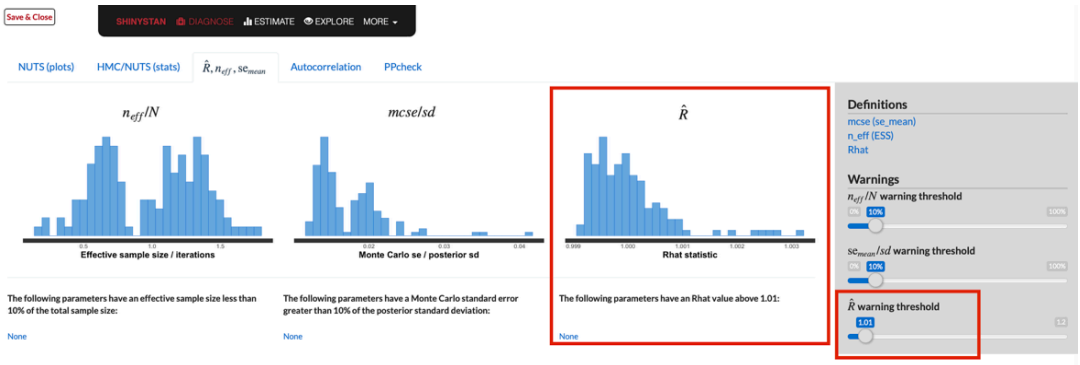
- We've already talked about visually checking the traceplots of the chains.

- A popular numeric measure is the *Gelman-Rubin Diagnostic*, often written *Rhat*.
  - Remember how we said that the chains should look more or less alike?
  - That implies that the variability across the chains should be the same as the variability within each chain
    - so let's compare the two kinds of variability.
- Rhat, very roughly, is the variability across chains divided by the variability within chains—we don't need to know the exact calculation.
- Andrew Gelman recommended in 2004 that Rhat should be less than or equal to 1.1 (as the chains become infinitely long, Rhat will approach 1.0).
  - The cutoff of 1.1 now seems too high—I suggest 1.01, but we will see a better approach later in this segment.

- Once you have a fitted model, both `describe_posterior()` and `summary()` will give `Rhat`, and you can find it in `shinystan` as well
  - Remember, you get the `shinystan` window with `launch_shinystan()`.

```
describe_posterior(exam_hier, ci = .9, rope_ci = .9)
```

```
## # Description of Posterior Distributions
##
## Parameter   | Median |          90% CI |      pd |        90% ROPE | % in ROPE |  Rhat |      ESS
## ------------------------------------------------------------------------------------------------
## (Intercept) | -0.010 | [-0.079, 0.051] |  60.42% | [-0.100, 0.100] |       100 | 1.003 |  592.319
## standLRT    |  0.557 | [ 0.520, 0.589] | 100.00% | [-0.100, 0.100] |         0 | 1.003 | 1363.431
```



```
summary(exam_hier, digits = 3)
```

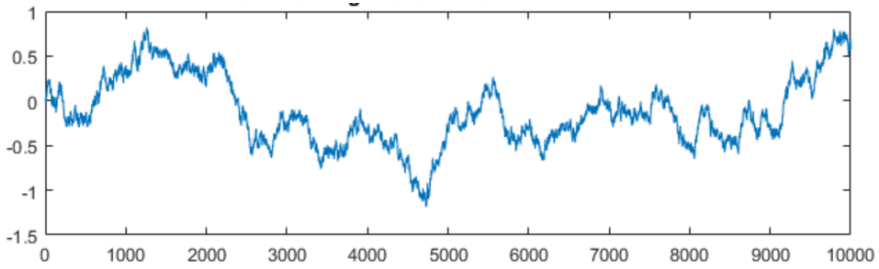
```
MCMC diagnostics
                                  mcse  Rhat  n_eff
(Intercept)                       0.002 1.003  592
standLRT                          0.001 1.003 1363
b[(Intercept) school:1]           0.002 1.000 2291
b[standLRT school:1]              0.001 1.000 4839

For each parameter, mcse is Monte Carlo standard error,
n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at convergence Rhat=1).
```

- The *Monte Carlo standard error* is the standard error for the mean of the parameter (with respect to the posterior distribution).
    - You'd like it to be less than 1/10 of that parameter's posterior's standard deviation.

Too small an effective sample size

- We typically want lots of samples from the posterior distribution so that we have good accuracy in a statement like $\text{Prob}(\theta > 5) = \text{something}$.
- However,
  - the nature of MCMC methods, including the Metropolis algorithm, is that there each draw is correlated with the previous draw
    - commonly seen in time series
    - known as *serial correlation*



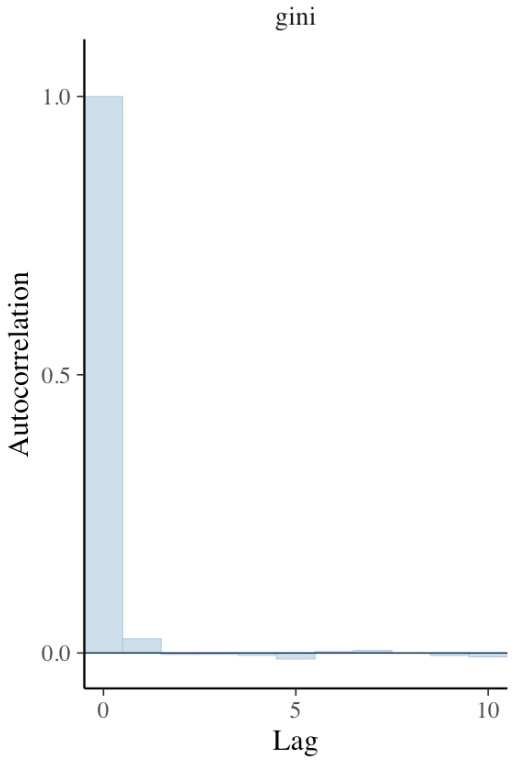
Effective sample size

- When there is a lot of serial correlation, the sample has the information equivalent of a smaller sample—the size of that smaller sample is the *effective sample size*, *ESS*, or `n_eff`.
- If you've seen some time series,

$$\text{ESS} = \frac{N}{1 + 2\sum_{k=1}^{\infty} \text{ACF}(k)},$$

where $\text{ACF}(k)$ is the autocorrelation at lag k

- The figure below shows the autocorrelation for `gini` in `gini_stan`; there is almost none (ignore lag = 0) and so `n_eff` is almost as large as the number of draws.



$$\text{ESS} = \frac{N}{1 + 2\sum_{k=1}^{\infty} \text{ACF}(k)}.$$

The Cure

- Longer chains will help with both Rhat and a low effective sample size
- You can get the equivalent by running more chains, instead of longer chains.
  - In R, for `stan_glm()` and `stan_glmer()` you can do so with the argument `chains = 8` say.

```
`stan_glm(life_expectancy ~ gini, data = mini_gapminder, chains = 8)
```

```
+ The default is 4.
```

- With more than a few chains, you might want to run them in parallel on several cores.
- If the chains run quickly, it will take longer to set up the cores than it's worth.

```
stan_glm(life_expectancy ~ gini, data = mini_gapminder, chains = 8, cores = 8)
```

State of the art for how long to run chains

- The Rhat cutoff of 1.1 is almost certainly too high
- The value of Rhat calcululated from actual chains is not super accurate.
- There is a newer measure called the *Stable Gelman-Rubin Diagnostic*.
- It is not yet incorporated into standard tools, but only in a package called `stableGR`.
- The R code is therefore a little clunky:

```
gini_stan %>%
  as.array(pars = "gini") %>%
  as.matrix %>%
  n.eff()
```

```
gini_stan %>%
  as.array(pars = "gini") %>%
  as.matrix %>%
  n.eff()
```

```
## $n.eff
##   se
## 4000
##
## $converged
## [1] FALSE
##
## $n.target
##   se
## 6147
```

- Notice that it tells you whether your chains have converged—apparently they did not in this example—along with an estimate of how long your chain should be in `n.target`.
- To reach that target, you can use the `iter =` argument in `stan_glm()` or `stan_glmer()`.
  - You have to remember
    - Half of each chain is thrown away as “warmup”
    - `n.target` is the total length, not the per chain length
    - Put both together, and `iter` should be greater than $\texttt{n.target} \times 2 \big/ (\text{number of chains})$

```
gini_stan_big <- stan_glm(formula = life_expectancy ~ gini, data = mini_gapminder, iter = 3500)
```

```
gini_stan_big %>% as.array(pars = "gini") %>% as.matrix %>% n.eff()
```

```
## $n.eff
##   se
## 7000
##
## $converged
## [1] TRUE
##
## $n.target
## NULL
```

```
describe_posterior(gini_stan_big, ci = 0.90, rope_ci = 0.90)
```

```
## # Description of Posterior Distributions
##
## Parameter   | Median |           90% CI |      pd |        90% ROPE | % in ROPE |  Rhat |      ESS
## ------------------------------------------------------------------------------------------------
## (Intercept) | 87.937 | [83.668, 92.099] | 100.00% | [-0.717, 0.717] |         0 | 1.001 | 6609.581
## gini        | -0.387 | [-0.497, -0.284] | 100.00% | [-0.717, 0.717] |       100 | 1.001 | 6843.536
```

Not actually that different from what we had before:

```
describe_posterior(gini_stan, ci = 0.90, rope_ci = 0.90)
```

```
## # Description of Posterior Distributions
##
## Parameter   | Median |           90% CI |      pd |        90% ROPE | % in ROPE |  Rhat |      ESS
## ------------------------------------------------------------------------------------------------
## (Intercept) | 87.986 | [83.621, 92.202] | 100.00% | [-0.717, 0.717] |         0 | 1.001 | 3894.816
## gini        | -0.388 | [-0.492, -0.275] | 100.00% | [-0.717, 0.717] |       100 | 1.001 | 3981.026
```