MCMC Diagnostics

Patrick Breheny

March 5

Introduction

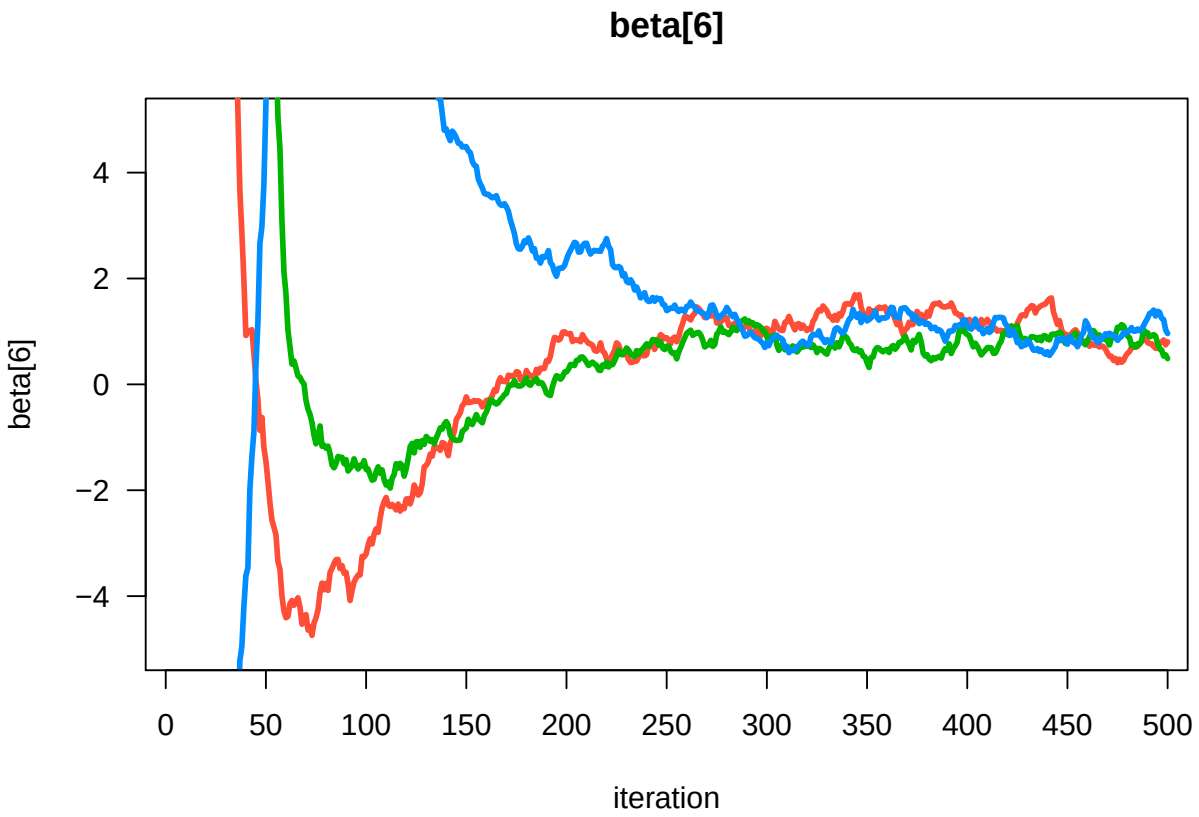
- As we have seen, there are two chief concerns with the use of MCMC methods for posterior inference:
  - Has our chain converged in distribution to the posterior?
  - How much information about the posterior does our chain contain?
- Today's lecture is about methods – both informal (graphical/visual checks) and formal – for assessing each of the above concerns
- The `R2OpenBUGS` and `R2jags` packages provide some of these tools, but we will also be using the package `coda`, which provides many additional diagnostics for assessing convergence and accuracy
- We will illustrate these methods on basic linear regression model for a classic data set on Swiss Fertility (full details in supplementary code)

Multiple chains

- Most approaches for detecting convergence, both formal and informal, rest on the idea of starting multiple Markov chains and observing whether they come together and start to behave similarly (if they do, we can pool the results from each chain)
- In `bugs`/`jags`, the number of chains is set by the `n.chains` argument (the default is 3 chains)
- The results are contained in a $T \times M \times p$ array called `sims.array`, where T is the number of draws/iterations, M is the number of chains, and p is the number of parameters we are monitoring, although most diagnostic functions operate at a higher level and do not require interacting with the array itself

Multiple chains: Traceplot

Trace plots, as in the last lecture, can be obtained via `traceplot(fit)`



Initial values

- The previous plot indicates that the three chains converge to the posterior after around $T = 300$ iterations; certainly, however, the number of iterations required to reach convergence depends on the initial values
- It is typically recommended (*e.g.*, Gelman and Rubin, 1992) to use *overdispersed* initial values, meaning "more variable than the target distribution" *i.e.*, the posterior
- In `bugs`/`jags`, initial values may be specified in one of two ways:
  - As a list of M lists, each specifying the initial values for each parameter for that chain
  - As a function generating a list of (random) initial values

Quantifying convergence

- Although looking at trace plots is certainly useful, it is also desirable to obtain an objective, quantifiable measure of convergence
- Numerous methods exist, although we will focus on the measure originally proposed in Gelman and Rubin (1992), which is the method used in `R2OpenBUGS` and `R2jags`
- The basic idea is to quantify the between-chain and the within-chain variability of a quantity of interest – if the chains have converged, these measures will be similar; otherwise, the between-chain variability will be larger

$\hat{R}$

- The basic idea of the estimator is as follows (the actual estimator makes a number of modifications to account for degrees of freedom):
  - Let B denote the standard deviation of the pooled sample of all MT iterations (the between-chain variability)
  - Let W denote the average of the within-chain standard deviations
  - Quantify convergence with

$$\hat{R} = \frac{B}{W}$$

- If $\hat{R} \gg 1$, this is clear evidence that the chains have not converged
- As $T \to \infty$, $\hat{R} \to 1$; $\hat{R} < 1.05$ is widely accepted as implying convergence for practical purposes

Obtaining $\hat{R}$ in R

- $\hat{R}$ is displayed for `bugs` and `jags` objects using both the `print` and `plot` methods, which we will discuss later
- More details (along with other convergence measures) are given in the `coda` package, whose `gelman.diag` function provides, in addition to $\hat{R}$ itself,
  - An upper confidence interval for $\hat{R}$
  - A multivariate extension of $\hat{R}$ for quantifying convergence of the entire posterior

gelman.diag output

	$\hat{R}$	Upper CI
β_1	1.18	1.25
β_2	1.02	1.06
β_3	1.11	1.27
β_4	1.06	1.07
β_5	1.04	1.05
β_6	1.03	1.10
τ	1.00	1.01

Multivariate $\hat{R}$: 1.04

Updating a chain

- It is worth pointing out that if a chain has not converged, one does not have to start over again from the initial values, but may simply resume the chain from where we left off
- In BUGS, unfortunately, this feature is only available through the OpenBUGS GUI (*i.e.*, cannot be accessed from within `R`, at least as far as I know)
- In JAGS, however, "updating" a chain in `R` is straightforward:

```
fit <- update(fit, 1000)
```

replaces `fit` with 1000 new draws (per chain), treating the values in the original `fit` as burn-in

Multiple chains: Pros and Cons

- The obvious downside to running multiple chains is that it is inefficient: we intentionally force our sampler to spend extra time in a non-converged state, which in turn requires much more burn-in
- The obvious upside, however, is that it provides us with some measure of confidence that we are actually drawing samples from the posterior
- It should be stressed, however, that (without additional assumptions about the posterior) no method can truly prove convergence; diagnostics can only detect failure to converge

How many iterations?

- We may use $\hat{R}$, then, as a guide to how long we must run our chains until convergence
- The obvious next question is: how long must we run our chains to obtain reasonably accurate estimates of the posterior?
- This issue is complicated by the fact that we cannot obtain independent draws from the posterior distribution and must settle instead for correlated samples

Accuracy of the Monte Carlo approach

- If we could obtain iid draws from the posterior, estimating the Monte Carlo standard error (at least, of the posterior mean) is straightforward: letting σ_p denote the posterior standard deviation, the MCSE is $\sigma_p/\sqrt{T}$
- A reasonable rule of thumb, then, would be to use $T = 400$; this is the point at which the Monte Carlo error is less than 5% of the overall uncertainty about the mean
- Another rule of thumb, studied in Raftery and Lewis (1992), is that $T = 4,000$ iterations are required for reasonable accuracy concerning the 2.5th (and 97.5th) percentiles

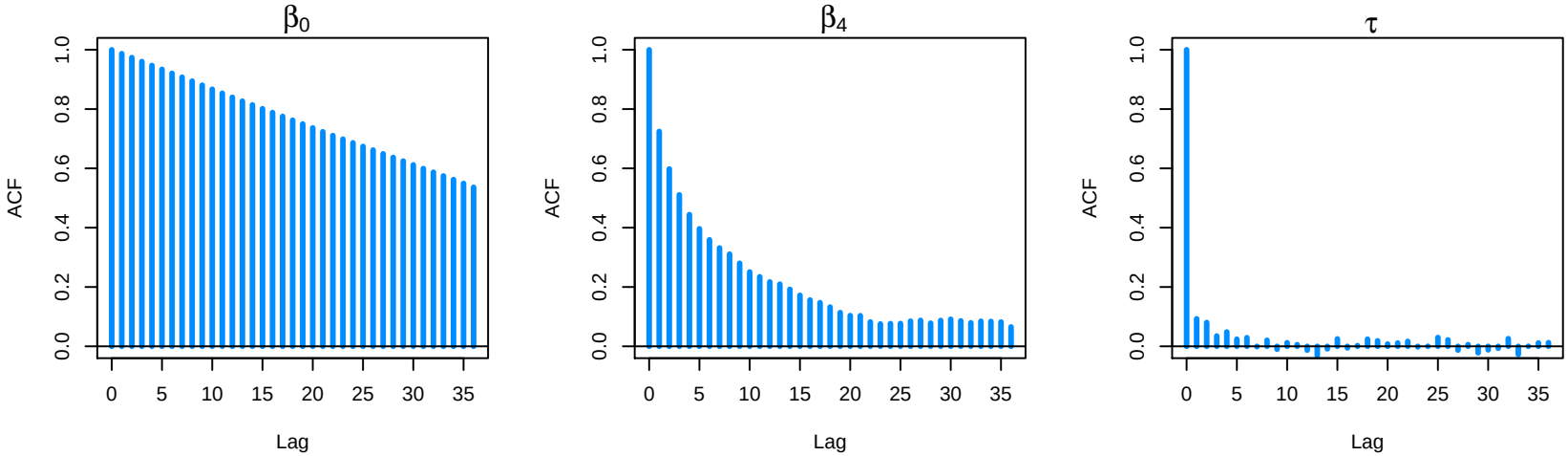
Efficiency

- In the presence of autocorrelation, however, we may obtain $T = 4,000$ samples from the posterior, but those samples contain less (possibly much less) information about the 2.5th and 97.5th percentiles than 4,000 independent draws would
- The lower the autocorrelation, the greater the amount of information contained in a given number of draws from the posterior; this is referred to as the *efficiency* or *mixing* of the chain

Autocorrelation

- The autocorrelation between two states s and t of a Markov chain is defined, simply, as the correlation between $X^{(s)}$ and $X^{(t)}$
- If the chain is *stationary*, in the sense that its mean and variance are not changing with time, then the correlation between $X^{(t)}$ and $X^{(t+k)}$ does not depend on t; this is known as the *lag-k autocorrelation*
- To calculate and plot the autocorrelation function, one may use the `acfplot` function in `coda`, or the `acf` function on elements of `sims.array` directly

Autocorrelation



Thinning

- A somewhat crude, yet reasonably effective, method dealing with autocorrelation is to only keep every k draws from the posterior and discard the rest; this is known as *thinning* the chain
- Both `R2OpenBUGS` and `R2jags` allow you to control thinning through the `n.thin` option, although it is important to note a difference between the two with regard to their defaults:
  - By default, `R2OpenBUGS` sets `n.thin=1`; *i.e.*, no thinning
  - By default, `R2jags` sets `n.thin` so that the chain is thinned down to 1,000 samples per chain from the posterior

Thinning: Pros and Cons

- The advantages of thinning are (a) simplicity and (b) a reduction in memory usage – saving and working with large chains can be burdensome, especially when p and T are large
- The disadvantage is that we are clearly throwing away information; thinning can never be as efficient as using all the iterations
- If we decide not to thin, we must estimate the MCSE for dependent samples; this is not trivial

Markov chain CLTs

- Although the proof of such theorems is beyond the scope of this course, there exist central limit theorems for dependent samples that can be applied to Markov chains
- In particular, suppose we are interested the posterior mean of a quantity ω; it is still true that our MCMC estimate, $\bar{\omega}$, tends in distribution to $\mathrm{N}(\mathrm{E}(\omega|\mathbf{y}), \rho/T)$ for some positive constant ρ
- Note, however, that ρ is not the posterior variance, as it would be if we had iid samples; in the presence of autocorrelation, $\rho > \sigma_p^2$

Batch means

- How can we estimate the MCSE, $\sqrt{\rho/T}$?
- One simple method is to use non-overlapping *batch means*
- Suppose we divide up our sample into Q batches, each with a iterations, and let $\bar{\omega}_q$ denote the mean in batch q
- If a is sufficiently large that the batch means are approximately uncorrelated, $\mathrm{Var}(\bar{\omega}_q) \approx \rho/a$
- Thus, a reasonable estimator is

$$\hat{\rho} = a\widehat{\mathrm{Var}}(\bar{\omega}),$$

where $\widehat{\mathrm{Var}}(\bar{\omega})$ is the sample variance of the $\{\bar{\omega}_q\}$

Effective sample size

- We may define, then, an *effective sample size* of the Markov chain as follows:

$$T^* = T\frac{\hat{\sigma}_p^2}{\hat{\rho}}$$

- One may then apply the iid rules of thumb analogously, using T^* in place of T: 400 (effective) iterations is enough for a reasonable estimate of the posterior mean, and 4,000 iterations is required for a reasonable 95% posterior interval

Time series methods

- More sophisticated methods for estimating MCSE have been proposed based on applying ideas from the time series literature to MCMC chains
- This is the approach used by `coda`, which fits an autoregressive (AR) model to the data and then estimates its spectral density; the results are available via the functions `summary` (which provides the MCSE estimate) and `effectiveSize` (which estimates T^*)
- BUGS itself (if you run it through its stand-alone GUI) estimates MCSE and T^* using batch means
- `R2OpenBUGS` and `R2jags` use yet another, even cruder method based on within- and between- chain variances

Effective sample size measures

For the swiss fertility data, with 10,000 iterations and a burn-in of 5,000 and no thinning, we have the following effective sample size estimates

	Time series	Batch means	Between/Within
β_1	91	214	35
β_2	475	416	63
β_3	440	457	190
β_4	1112	788	980
β_5	1059	1030	15000
β_6	167	246	60
τ	7419	4905	810

Summary

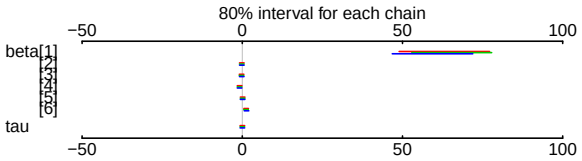
- In summary, then, and at the risk of oversimplifying things, any responsible Bayesian analysis involving MCMC should check to ensure that, for all quantities of interest,
  - $\hat{R} < 1.05$
  - $T^* > 4,000$
- Obviously, neither of these diagnostics provide proof that your MCMC approach has adequately estimated the posterior, and further plots and diagnostics are useful, but in practice, these two checks do tend to detect a lot of errors
- Fortunately, these quantities are easily accessible in `bugs`/`jags` with the `print` and `plot` methods

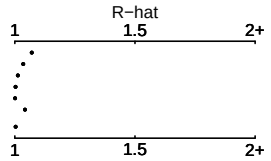
print(fit)

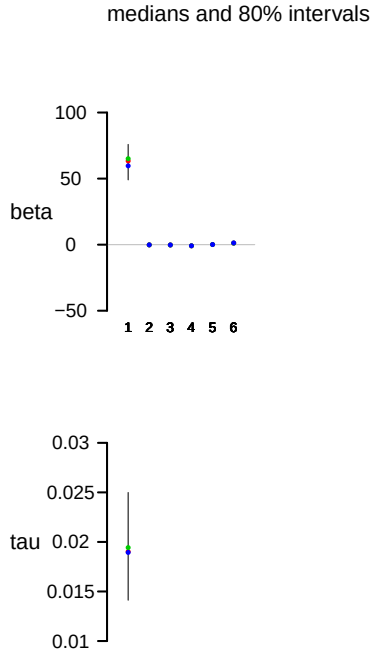
```
> fit
Current: 3 chains, each with 10000 iterations
         (first 5000 discarded)
Cumulative: n.sims = 15000 iterations saved
        mean   sd 2.5%  25%  50%  75% 97.5% Rhat n.eff
beta[1] 62.5 10.3 42.1 55.4 62.8 69.6  82.3  1.1    35
beta[2] -0.2  0.1 -0.3 -0.2 -0.2 -0.1   0.0  1.0    63
beta[3] -0.2  0.3 -0.7 -0.4 -0.2 -0.1   0.3  1.0   190
beta[4] -0.9  0.2 -1.2 -1.0 -0.9 -0.7  -0.5  1.0   980
beta[5]  0.1  0.0  0.0  0.1  0.1  0.1   0.2  1.0 15000
beta[6]  1.2  0.4  0.5  1.0  1.2  1.5   2.0  1.0    60
tau      0.0  0.0  0.0  0.0  0.0  0.0   0.0  1.0   810
```

(Obviously, we need to increase T here)

plot(fit)


80% interval for each chain
−50
0
50
100
beta[1]
[2]
[3]
[4]
[5]
[6]
tau
−50
0
50
100


R−hat
1
1.5
2+
1
1.5
2+


medians and 80% intervals
100
50
beta
0
−50
1 2 3 4 5 6
0.03
0.025
tau
0.02
0.015
0.01