Introduction to Applied Bayesian Modeling

ICPSR

Day 6

MCMC Diagnostics

1) Two approaches to monitoring convergence
   - Monitoring one chain for a long time
   - Monitoring more than one chain for a shorter period of time.
2) Strategies for improving results

Convergence and MCMC

After the model has **converged**, samples from the conditional distributions are used to summarize the posterior distribution of parameters of interest, in this case β and τ.

Convergence refers to the idea that eventually the Gibbs Sampler or other MCMC technique that we choose will eventually reach a stationary distribution. From this point on it stays in this distribution and moved about (or mixes" throughout the subspace forever.

The general questions for us to ask are:

1) At what point do we know that have we converged to the stationary distribution? (i.e. how long should our "burn-in" period be?

2) After we have reached the stationary distribution, how many iterations will it take to summarize the posterior distribution?

Possible problems with MCMC (Gelman)

1) The assumed model may not be realistic from a substantive point of view or may not fit.

2) Errors in calculation or programming!
   - Often, simple syntax mistakes may be responsible; however, it is possible that the algorithm may not converge to a *proper* distribution.

3) Slow convergence: this is the problem we are most likely to run into. The simulation can remain for many iterations in a region heavily influenced by the starting distribution or in a local maximum. If the iterations are used to summarize the target distribution, they can yield falsely precise estimates.

Traceplots

One intuitive and easily implemented diagnostic tool is a traceplot which plots the parameter value at time t against the iteration number.

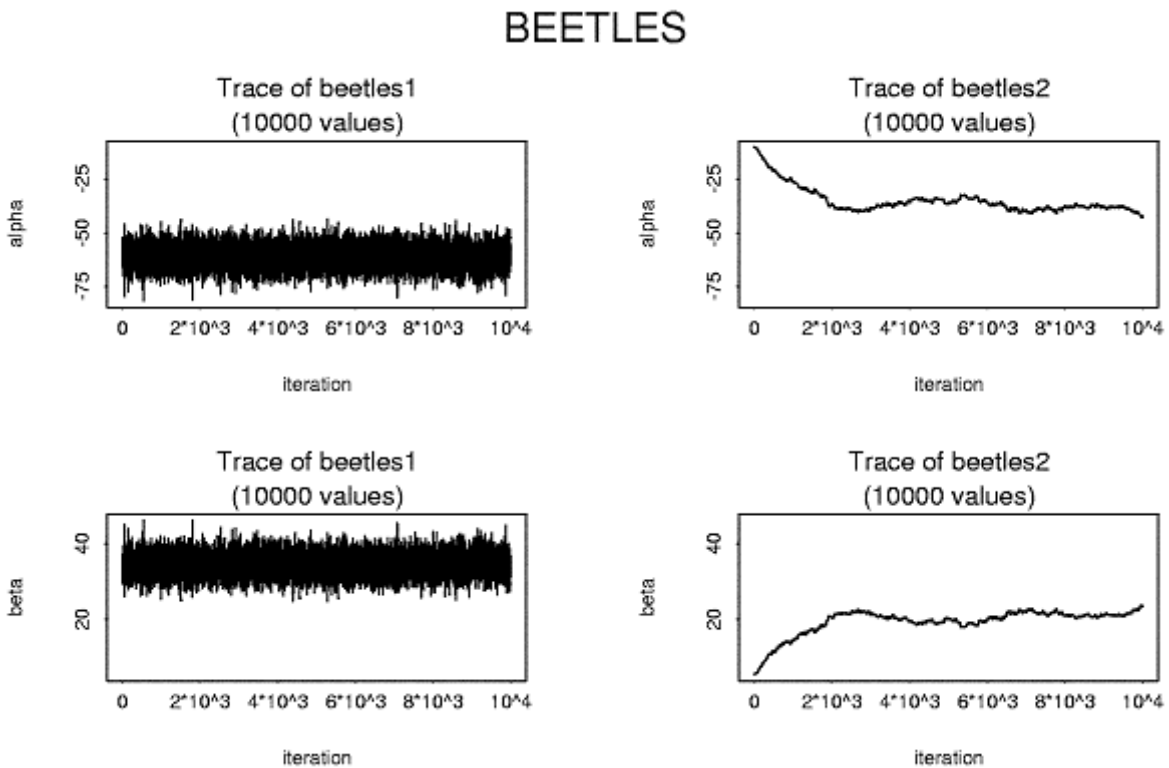
If the model has converged, the traceplot will move around the mode of the distribution.

A clear sign of non-convergence with a traceplot occurs when we observe some trending in the sample space.

The problem with traceplots is that it may appear that we have converged, however, the chain trapped (for a finite time) in a local region rather exploring the full posterior.

In WinBugs, you may setup traceplots to monitor parameters while the program runs.

Examples of Apparent Convergence and Non-Convergence Based on a trace plot



Autocorrelation Diagnostic

Autocorrelation refers to a pattern of serial correlation in the chain, where sequential draws of a parameter, say β_1, from the conditional distribution are correlated.

The cause of autocorrelation is that the parameters in our model may be highly correlated, so the Gibbs Sampler will be slow to explore the entire posterior distribution.

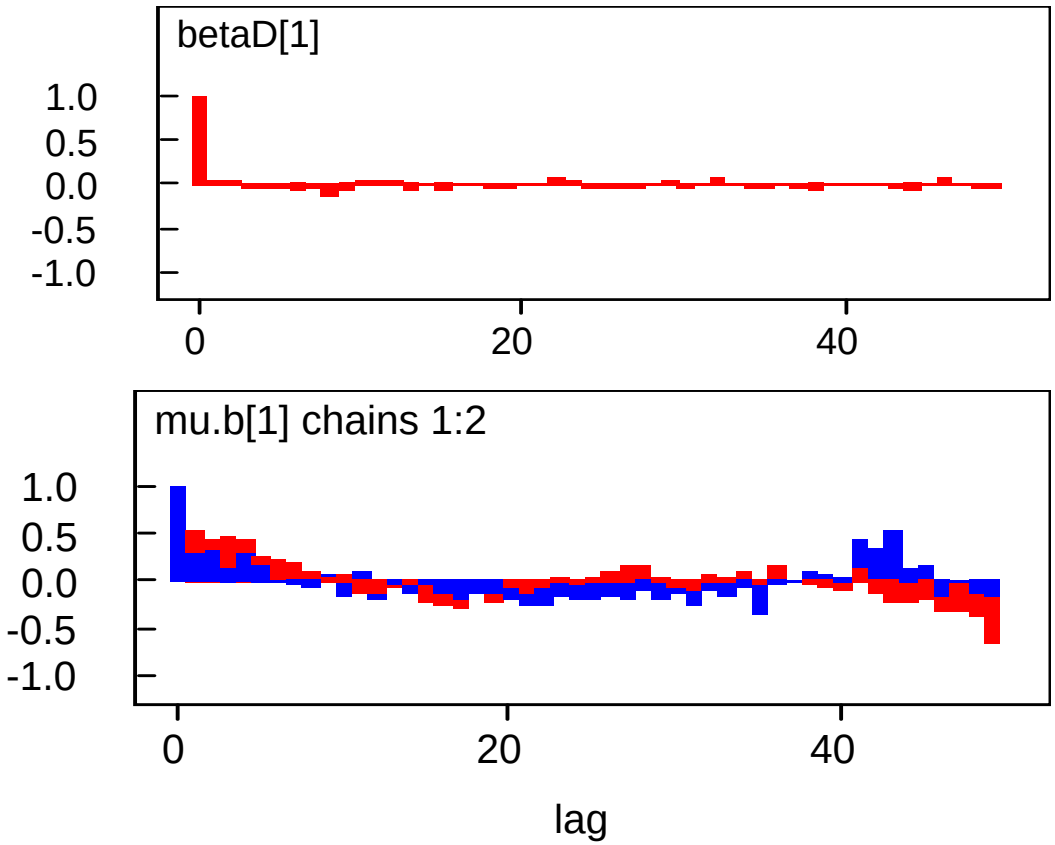
The reason why autocorrelation is important is that it will take a very long time to explore the entire posterior distribution.

- Note that if the level of autocorrelation is high for a parameter of interest, then a traceplot will be a poor diagnostic for convergence.

WinBugs plots the level of autocorrelation out to 50 lags.

Typically, the level of autocorrelation will decline with an increasing number of lags in the chain (e.g. as we go from the 1000^{th} to the 1010^{th} lags, the level of autocorrelation will often decline.) When this dampening doesn't occur, then you have a problem and will probably want to re-parameterize the model (more on this below).

Example of Model without autocorrelation (top) and with autocorrelation (bottom)



In serious (and not uncommon cases), the autocorrelation plot will be a solid bar across the screen.

Running means of parameters

Running means: Once you have taken enough draws to summarize the posterior distribution, then if the model has converged, further samples from a parameter's posterior distribution should not influence the calculation of the mean.

A plot of the average draw from the conditional distribution of draws 1 through t against t is useful for identifying convergence.

Note: that you could probably get the same effect with a traceplot.

Note: WinBugs does not have a canned feature to do this.

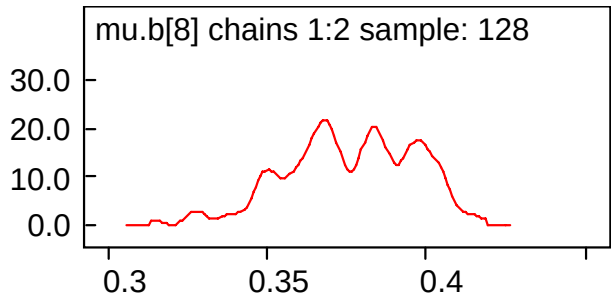
Kernel Density Plots

Kernel density plots (a.k.a. smoothed density; histograms) may be useful diagnostic.

Sometimes non-convergence is reflected in multimodal distributions. This is especially true if the kernel density plot isn't just multi-modal, but lumpy (you'll know what I mean when you see it).

When you get a lumpy posterior, it may be important to let the algorithm run a bit longer. Often, doing this will allow you to get a much more reasonable summary of the posterior.

Example of a problematic kernel density plot



A more satisfactory kernel density plot would look more bell-shaped, though it need not be symmetric

Geweke Time-Series Diagnostic

The Geweke Time-Series Diagnostic: if a model has converged, then if we simulate a large number of draws, the mean (and variance) of a parameter's posterior distribution from the first half of the chain will be equal to the mean (and variance) from the second half of the chain.

Technically, this statistic is based on spectral density functions that are beyond the purview of this course and WinBugs does not estimate this statistic directly, but if you export the CODA chain to R or S-plus the programs CODA and BOA report the Geweke statistic.

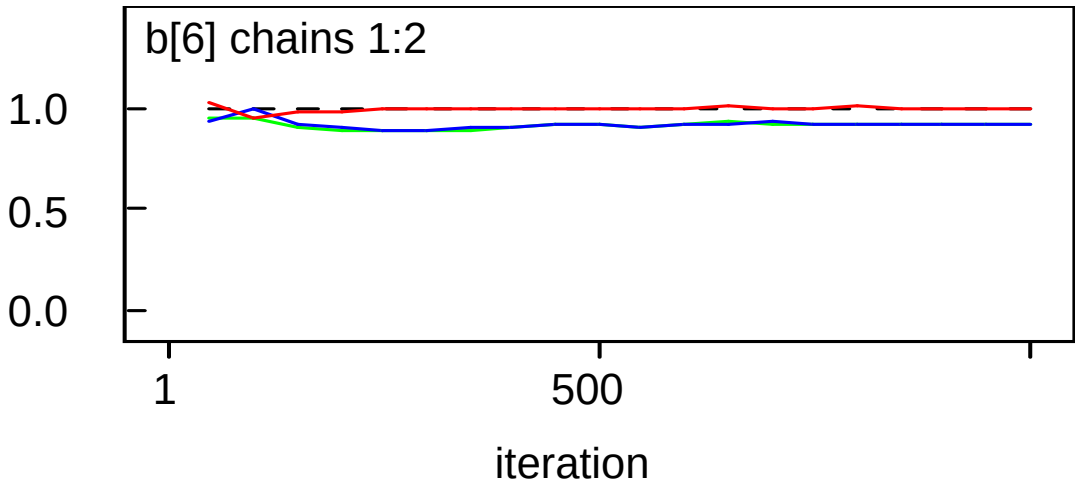
However, a perfectly reasonable way to proceed is look to see whether the posterior means (and variances) of your parameters are approximately the same for different halves of your simulated chain.

The value of this approach is that by allowing the algorithm to run for a very long time, it may reach areas of the posterior distribution that may not otherwise be reached.

Gelman and Rubin

- test based on 2 or more parallel chains , (each started from different initial values). Their method is based on a comparison of the within and between chain variances for each variable (essentially a classical analysis of variance). Best results are obtained for parameters whose marginal posterior densities are approximately normal

Example of BGR Diagnostic Plot from Winbugs consistent with convergence



The normalized width of the central 80% interval of the pooled runs is green

The normalized average width of the 80% intervals within the individual runs is blue

R is red. *R* would generally be expected to be greater than 1 if the starting values are suitably over-dispersed.

Brooks and Gelman (1998) emphasize that one should be concerned both with convergence of *R* to 1, and with convergence of both the pooled and within interval widths to stability.

Heidelberger-Welch

- tests the null hypothesis that the sampled values for each variable are form a stationary process. If the null hypothesis is rejected for a given variable, the test is repeated after discarding the first 10% of iterations. If the hypothesis is again rejected, a further 10% of iterations are discarded. This process is repeated until either a portion of the chain (of length of the total number of iterations) passes the stationarity test, or 50% of the iterations have been discarded and the null hypothesis is still rejected. If the latter occurs, This indicates that a longer BUGS run is needed in order to achieve convergence.

- If the stationarity test is passed, CODA/BOA reports the number of iterations to keep (i.e. which are diagnosed to arise from a stationary process), the number of initial iterations to discard and the Cramer-von-Mises statistic.

Raftery-Lewis

- method applies to single chains. It is intended both to detect convergence to the stationary distribution and to provide bounds for the accuracy of the estimated quantiles of functions of variables of interest. The user must specify the quantile to be estimated (the default is the 2.5*th* percentile), the desired degree of accuracy for the estimate of this quantile (the default is 0.005) and the required probability of attaining this degree of accuracy (the default is 0.95). The output then reports Nmin -- the minimum number of iterations that would be needed to estimate the specified quantile to the desired precision if the samples in the chain were independent.

Convergence Diagnostic Summary

1) You can never prove that something has converged, you can only tell when something has not converged.
2) If you're model has not converged and you are confident that **YOU** haven't made a stupid mistake, then the best thing to do may be to just let the model run a long time. CPU time is often "cheaper" than your time.

   → For models with large numbers of parameters you should let the model run for a long time.
3) There are a number of easy to implement tricks (mostly reparamerizations) that will help to speed convergence in most regression-based models.
4) Convergence does not mean that you have a good model!!! Convergence should be the beginning of model assessment, not its end.

Tricks to speed convergence

1) Standardize all of your variables by subtracting them from their sample means and dividing by their sample variances.

   - This speeds convergence by decreasing the posterior correlation between parameters.

   In a simple example, suppose that $Y_i \sim N(a + bX_i, 1)$ and that we chose flat priors for a and b. The posterior correlation between a and b is:

   $$r_{a,b} = - \text{mean}(x) / (\text{mean}(x) + \text{var}(x))^{.5}$$

   If the absolute value of mean(x) is large relative to the sample variance of x, then you will have a large posterior correlation between a and b and therefore slow convergence (due to a high autocorrelation in the parameter simulations).

Tricks to speed convergence

2) If you have multiple indicators of the same concept, create an index or use a latent variable model to decrease correlation among the predictors.

(we will get to latent variable models the last week, but they are very straightforward to implement).

Tricks to speed convergence

3) Use multivariate (normal) priors.

Multivariate normal priors allow for prior correlation among regression coefficients.

```
model  {
  for (i in 1:15)  {
    DemDim1[i] ~ dnorm(muD[i], tauD)
    muD[i] <- betaD[1] + betaD[2]*((DemActId[i])) + betaD[3]*((DemNotActId[i]))
     }

 betaD[1:3] ~ dmnorm( nuD[], v1[ , ] )

nuD[1] <-  -.687
nuD[2] <- .099
nuD[3] <- .212

v1[1,1] <- .1; v1[1,2] <- 0; v1[1,3] <- 0
v1[2,1] <- 0;  v1[2,2] <- .1; v1[2,3] <- 0
v1[3,1] <- 0; v1[3,2] <- 0; v1[3,3] <- 0.1

tauD ~ dchisqr(1)
}
```

Tricks to Speed Convergence

4) Use WinBugs' Over-relax Algorithm.

This generates multiple samples at each iteration and then selects one that is negatively correlated with the current value.

The time per iteration will be increased, but the within-chain correlations should be reduced and hence fewer iterations may be necessary.

However, this method is not always effective and should be used with caution.

The auto-correlation function may be used to check whether the mixing of the chain is improved

Tricks to Speed Convergence

5) Pick good initial values.

If your initial values are near their posterior modes, then convergence should occur relatively quickly, especially if there is not an autocorrelation problem.

Tricks to speed convergence

6) Change sampling algorithms.

This can be a little tricky, but can really speed up the chains and help with mixing.

It requires you to go into the ‘guts’ of the Bugs files and change some default settings.

It’s not hard, but you can certainly make some mistakes here and mess a lot of things up—potentially.

So BE CAREFUL!

Tricks to Speed Convergence

7) Just wait: Sometimes models just take a long time to converge.

In extreme cases, it may take 100,000 iterations of your sampler to fully explore the posterior distribution.

Unfortunately, this tends to occur in models with a large number of parameters that are already slow to run (especially if you also have a large n).

The problem with this is that storing 100,000 samples * m parameters requires considerable memory. This may become particularly important if you try to calculate multiple sample statistics at once after convergence.

A mildly controversial solution to this is to “thin” your chain, by only storing every k^{th} sample of your chain.