And \( \nu \)'s mixing is the worst of the three: its ACF looks like that of an \( AR_1 \) series with first-order serial correlation of a bit less than +0.9.

WinBUGS has a somewhat complicated provision for printing out the autocorrelations; alternately, you can approximately infer \( \hat{\rho}_1 \) from an equation like (51) above: assuming that the WinBUGS people are taking the output of any MCMC chain as (at least approximately) \( AR_1 \) and using the formula

\[
\hat{SE}(\bar{\theta}^*) = \frac{\hat{\sigma}_\theta}{\sqrt{m}} \sqrt{\frac{1}{1 - \hat{\rho}_1}},
\]

(54)

you can solve this equation for \( \hat{\rho}_1 \) to get

\[
\hat{\rho}_1 = \frac{m \left[ \hat{SE}(\bar{\theta}^*) \right]^2 - \hat{\sigma}_\theta^2}{m \left[ \hat{SE}(\bar{\theta}^*) \right]^2 + \hat{\sigma}_\theta^2}.
\]

(55)
WinBUGS Implementation (continued)

Plugging in the relevant values here gives

\[ \hat{p}_1 = \frac{(10,000)(0.04253)^2 - (1.165)^2}{(10,000)(0.04253)^2 + (1.165)^2} = 0.860, \]  

which is smaller than the corresponding value of 0.972 generated by the classicBUGS sampling method (from CODA, page 67).

To match the classicBUGS strategy outlined above (page 71) I typed 30000 in the updates window in the Update Tool and hit update, yielding a total monitoring run of 40,000.

Remembering to type 42000 in the end box in the Sample Monitoring Tool window before going any further, to get a monitoring run of 40,000 after the initial burn-in of 2,000, the summaries below for \( \mu \) are satisfactory in every way.
A monitoring run of 40,000 also looks good for $\sigma$: on this basis, and conditional on this model and prior, I think $\sigma$ is around 3.87 (posterior mean, with an MCSE of 0.006), give or take about 0.44 (posterior SD), and my 95% central posterior interval for $\sigma$ runs from about 3.09 to about 4.81 (the distribution has a bit of skewness to the right, which makes sense given that $\sigma$ is a scale parameter).
If the real goal were \( \nu \) I would use a longer monitoring run, but the main point here is \( \mu \), and we saw back on p. 67 that \( \mu \) and \( \nu \) are close to uncorrelated in the posterior, so this is good enough.

If you wanted to report the posterior mean of \( \nu \) with an MCSE of \( 0.01 \) (to come close to 3-sigfig accuracy) you’d have to increase the length of the monitoring run by a multiplicative factor of \( \left(\frac{0.02213}{0.01}\right)^2 \approx 4.9 \), which would yield a recommended length of monitoring run of about \( 196,000 \) iterations (the entire monitoring phase would take about 3 minutes at 2.0 (PC) GHz).
The posterior predictive distribution for \( y_{n+1} \) given \( (y_1, \ldots, y_n) \) is interesting in the \( t \) model: the predictive mean and SD of 404.3 and 6.44 are not far from the sample mean and SD (404.6 and 6.5, respectively), but the predictive distribution has very heavy tails, consistent with the degrees of freedom parameter \( \nu \) in the \( t \) distribution being so small (the time series trace has a few simulated values less than 300 and greater than 500, much farther from the center of the observed data than the most outlying actual observations).
Gaussian Comparison

The posterior SD for $\mu$, the only parameter directly comparable across the Gaussian and $t$ models for the NB10 data, came out $0.47$ from the $t$ modeling, versus $0.65$ with the Gaussian, i.e., the interval estimate for $\mu$ from the (incorrect) Gaussian model is about 40% wider than that from the (much better-fitting) $t$ model.
A Model Uncertainty Anomaly?

NB Moving from the Gaussian to the t model involves a net increase in **model uncertainty**, because when you assume the Gaussian you’re in effect saying that you know the t degrees of freedom are $\infty$, whereas with the t model you’re treating $\nu$ as unknown. And yet, even though there’s been an increase in model uncertainty, the inferential uncertainty about $\mu$ has **gone down**.

This is relatively rare—**usually when model uncertainty increases so does inferential uncertainty** (Draper 2004)—and arises in this case because of two things: (a) the t model **fits better** than the Gaussian, and (b) the Gaussian is actually a **conservative** model to assume as far as inferential accuracy for location parameters is concerned.
CODA in R

If you go to [http://www.r-project.org/](http://www.r-project.org/), click on CRAN (the Comprehensive R Archive Network), click on one of the CRAN mirror sites, and click on Package Sources, you’ll find a lot of contributed packages, one of which is CODA.

Clicking on coda will get you the source code for CODA (you can also visit [http://www-fis.iarc.fr/coda/](http://www-fis.iarc.fr/coda/), a web site maintained by Martyn Plummer, the guy who ported CODA from S+ to R).

In this way you can download the source for R-CODA and follow the instructions for installing it.

An easier way, if you’re running R on a machine that’s connected to the internet, is to go into R and just type

```r
install.packages( "coda" )
```

If everything goes smoothly this will automatically install R-CODA on your machine (you’ll need to do this on your laptop to get CODA; it’s already installed in the Engineering School version of R).

Once you have it in your local library you can invoke it from inside R with the command

```r
library( coda )
```

and you can find out what it can do with the command

```r
help( package = coda )
```

The idea is to run classicBUGS or WinBUGS, store the MCMC dataset somewhere handy, go into R, and use R-CODA to read the MCMC dataset in and analyze it.

All of the MCMC diagnostics I showed you described above are available to you with this approach.
References


