Bayesian Modeling, Inference, Prediction and Decision-Making

3: Simulation-Based Bayesian Computation

David Draper

Department of Applied Mathematics and Statistics
University of California, Santa Cruz

draper@ams.ucsc.edu
www.ams.ucsc.edu/∼draper

eBay/Google

10 Fridays, 11 Jan–22 Mar 2013 (except 25 Jan)

Short course web page:
www.ams.ucsc.edu/∼draper/eBay-Google-2013.html

Introduction to Markov Chain Monte Carlo (MCMC) methods

Computation via conjugate analysis (parts 2a–2c) produces **closed-form results** (good) but is **limited in scope** to a fairly small set of models for which straightforward conjugate results are possible (bad).

This was a **severe limitation** for Bayesians for almost 250 years (from the 1750s to the 1980s).

Over the past 20 years the Bayesian community has "discovered" and developed an entirely new computing method, **Markov chain Monte Carlo (MCMC)** ("discovered" because the physicists first figured it out about 60 years ago: Metropolis and Ulam, 1949; Metropolis et al., 1953).

We've seen that the **central Bayesian practical challenge** is the **computation of high-dimensional integrals**.

People working on the first atom bomb in World War II faced a **similar challenge**, and noticed that **digital computers** (which were then passing from theory (Turing 1943) to reality) offered an **entirely new approach** to solving the problem.

The idea (Metropolis and Ulam, 1949) was based on the observation that **anything you want to know about a probability distribution** can be learned to arbitrary accuracy by **sampling from it**.

Suppose, for example, that you're interested in a posterior distribution $p(\theta|y)$ that **cannot be worked with (easily) in closed form**, and initially (to keep things simple) think of θ as a **scalar** (real number) rather than vector.

Simulation-Based Computation

Four things of direct interest to you about $p(\theta|y)$ would be

- its **mean** $\mu = E(\theta|y)$ and **standard deviation** $\sigma = \sqrt{V(\theta|y)}$,

- its **shape** (basically you'd like to be able to trace out (an estimate of) the entire **density curve**), and

- one or more of its **quantiles** (e.g., to construct a 95% central posterior interval for θ you need to know the **2.5% and 97.5% quantiles**, and sometimes the **posterior median** (the **50th percentile**) is of interest too).

Suppose you could take an **arbitrarily large random sample** from $p(\theta|y)$, say $\theta_1^*, \ldots, \theta_m^*$.

Then each of the above four aspects of $p(\theta|y)$ can be **estimated** from the θ^* sample:

- $\widehat{E}(\theta|y) = \bar{\theta}^* = \frac{1}{m}\sum_{j=1}^{m} \theta_j^*$,

- $\sqrt{\widehat{V}(\theta|y)} = \sqrt{\frac{1}{m-1}\sum_{j=1}^{m} \left(\theta_j^* - \bar{\theta}^*\right)^2}$,

- the density curve can be estimated by a **histogram** or **kernel density estimate**, and

- percentiles can be estimated by **counting** how many of the θ^* values fall below a series of specified points—e.g., to find an estimate of the 2.5% quantile you solve the equation

$$\widehat{F}_\theta(t) = \frac{1}{m}\sum_{j=1}^{m} I(\theta_j^* \le t) = 0.025 \qquad (1)$$

for t, where $I(A)$ is the **indicator function** (1 if A is true, otherwise 0).

IID Sampling; Rejection Sampling

These are called **Monte Carlo** estimates of the true summaries of $p(\theta|y)$ because they're based on the **controlled use of chance**.

Theory shows that with large enough m, each of the Monte Carlo (or **simulation-based**) estimates can be made arbitrarily close to the truth with arbitrarily high probability, under some reasonable assumptions about the **nature of the random sampling**.

One way to achieve this, of course, is to make the sampling **IID** (this is **sufficient** but **not necessary** — see below).

If, for example, $\bar{\theta}^* = \frac{1}{m}\sum_{j=1}^{m}\theta_j^*$ is based on an IID sample of size m from $p(\theta|y)$, we can use the **frequentist fact** that in repeated sampling $V\left(\bar{\theta}^*\right) = \frac{\sigma^2}{m}$, where (as above) σ^2 is the variance of $p(\theta|y)$, to construct a **Monte Carlo standard error** (MCSE) for $\bar{\theta}^*$:

$$\widehat{SE}\left(\bar{\theta}^*\right) = \frac{\hat{\sigma}}{\sqrt{m}}, \tag{2}$$

where $\hat{\sigma}$ is the **sample SD** of the θ^* values.

This can be used, possibly after some **preliminary experimentation**, to decide on m, the Monte Carlo **sample size**, which later we'll call the length of the **monitoring run**.

An IID example. Consider the posterior distribution $p(\lambda|y) = \Gamma(29.001, 14.001)$ in the **LOS example** in part 2b.

We already know that the **posterior mean** of λ in this example is $\frac{29.001}{14.001} \doteq 2.071$; let's see how well the Monte Carlo method does in estimating this **known truth**.

IID Example (continued)

Here's an `R` function to construct **Monte Carlo estimates** of the **posterior mean** and **MCSE values** for these estimates.

```
gamma.sim <- function( m, alpha, beta, n.sim, seed ) {

  set.seed( seed )

  theta.out <- matrix( 0, n.sim, 2 )

  for ( i in 1:n.sim ) {

    theta.sample <- rgamma( m, shape = alpha, scale = 1 / beta )

    theta.out[ i, 1 ] <- mean( theta.sample )

    theta.out[ i, 2 ] <- sqrt( var( theta.sample ) / m )

 }

  return( theta.out )

}
```

This function simulates, `n.sim` times, the process of **taking** an IID sample of size m from the $\Gamma(\alpha, \beta)$ distribution and **calculating** $\bar{\theta}^*$ and $\widehat{SE}\left(\bar{\theta}^*\right)$.

```
rosalind 296> R

R version 2.15.2 (2012-10-26)
Copyright (C) 2012 The R Foundation for Statistical Computing

> m <- 1000

> alpha <- 29.001

> beta <- 14.001

> n.sim <- 500

> seed <- 9626954
```

IID Example (continued)

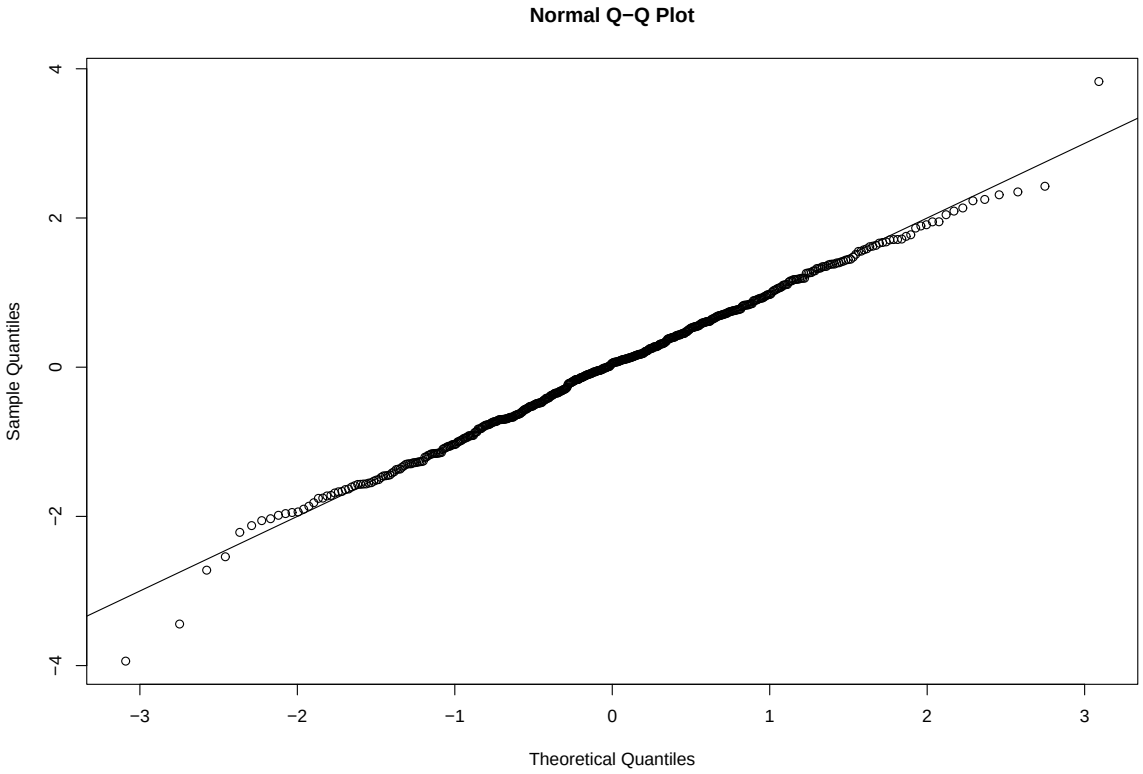
```
> theta.out <- gamma.sim( m, alpha, beta, n.sim, seed )

# This took less than 1 second at 1.6 Unix GHz.

> theta.out[ 1:10, ]

          [,1]       [,2]

 [1,] 2.050852 0.01217430
 [2,] 2.081629 0.01209275
 [3,] 2.067799 0.01243688
 [4,] 2.084047 0.01201150
 [5,] 2.063885 0.01219349
 [6,] 2.093904 0.01214949
 [7,] 2.081116 0.01203641
 [8,] 2.056941 0.01220216
 [9,] 2.086504 0.01242793
[10,] 2.073801 0.01210534
```

The $\bar{\theta}^*$ values fluctuate around the truth with a **give-or-take** of about 0.012, which agrees well with the **theoretical SE** $\frac{\sigma}{\sqrt{m}} = \frac{\sqrt{\alpha}}{\beta\sqrt{m}} \doteq 0.01216$ (recall that the variance of a Gamma distribution is $\frac{\alpha}{\beta^2}$).

```
> postscript( "gamma-sim1.ps" )

> theta.bar <- theta.out[ , 1 ]

> qqnorm( ( theta.bar - mean( theta.bar ) ) /
    sqrt( var( theta.bar ) ) )

> abline( 0, 1 )

> dev.off( )

null device
          1
```

Each of the $\bar{\theta}^*$ values is the mean of $m = 1{,}000$ IID draws, so (by the CLT) the **distribution** of the random variable $\bar{\theta}^*$ should be **closely approximated by a Gaussian**.

IID Example (continued)




```
> truth <- alpha / beta

> theta.bar.SE <- theta.out[ , 2 ]

> qnorm( 0.025 )
[1] -1.959964

> sum( ( theta.bar - 1.96 * theta.bar.SE < truth ) *
    ( truth < theta.bar + 1.96 * theta.bar.SE ) ) / n.sim

[1] 0.972
```

Thus we can use **frequentist ideas** to work out how big m needs to be to have **any desired Monte Carlo accuracy** for $\bar{\theta}^*$ as an estimate of the posterior mean $E(\theta|y)$.

In practice, with $p(\theta|y)$ unknown, you would probably take an **initial sample** (of size $m = 1,000$, say) and look at the MCSE to decide **how big m really needs to be**.

IID Example (continued)

```
> theta.bar <- gamma.sim( m, alpha, beta, 1, seed )

> theta.bar
         [,1]       [,2]
[1,] 2.050852  0.0121743
```

(1) Suppose you wanted the MCSE of $\bar{\theta}^*$ to be (say) $\epsilon = 0.001$. Then you could **solve the equation**

$$\frac{\widehat{\sigma}}{\sqrt{m}} = \epsilon \quad \leftrightarrow \quad m = \frac{\sigma^2}{\epsilon^2}, \tag{3}$$

which says (unhappily) that the required m goes up as the **square** of the posterior SD and as the **inverse square** of ϵ.

The previous calculation shows that $\frac{\widehat{\sigma}}{\sqrt{1000}} \doteq 0.0121743$, from which $\widehat{\sigma} \doteq 0.3849852$, meaning that **to get** $\epsilon = 0.001$ **you need a sample of size** $\frac{0.3849852^2}{0.001^2} \doteq 148,214 \doteq 148$k (!).

(2) Suppose instead that you wanted $\bar{\theta}^*$ to **differ** from the true posterior mean μ by **no more than** ϵ_1 with Monte Carlo probability **at least** $(1 - \epsilon_2)$:

$$P\left(\left|\bar{\theta}^* - \mu\right| \leq \epsilon_1\right) \geq 1 - \epsilon_2, \tag{4}$$

where $P(\cdot)$ here is based on the (frequentist) **Monte Carlo randomness** inherent in $\bar{\theta}^*$.

We know from the CLT and the calculations above that **in repeated sampling** $\bar{\theta}^*$ is approximately **normal** with mean μ and variance $\frac{\sigma^2}{m}$; this leads to the inequality

$$m \geq \frac{\sigma^2 \left[\Phi^{-1}\left(1 - \frac{\epsilon_2}{2}\right)\right]^2}{\epsilon_1^2}, \tag{5}$$

where $\Phi^{-1}(q)$ is the place on the standard normal curve where $100q\%$ of the area is to the left of that place (the qth **quantile** of the standard normal distribution).

A Closer Look at IID Sampling

(5) is like (3) except that the value of m from (3) has to be multiplied by $\left[\Phi^{-1}\left(1-\frac{\epsilon_2}{2}\right)\right]^2$, which typically makes the required sample sizes **even bigger**.

For example, with $\epsilon_1 = 0.001$ and $\epsilon_2 = 0.05$ — i.e., to have at least 95% Monte Carlo confidence that reporting the posterior mean as 2.071 will be correct to about **four significant figures** — (5) says that you would need a monitoring run of at least $148214(1.959964)^2 \doteq 569,358 \doteq 569$k (!).

(On the other hand, this sounds like a long monitoring run but takes less than **1 second** at 1.6 Unix GHz on a `Sun Ultra 45`, yielding $\left[\bar{\theta}^*, \widehat{SE}\left(\bar{\theta}^*\right)\right] = (2.0711, 0.00051)$.)

It's evident from calculations like these that people often report simulation-based answers with numbers of significant figures **far in excess of what is justified** by the actual accuracy of the Monte Carlo estimates.

A Closer Look at IID Sampling. I was able to easily perform the above **simulation study** because `R` has a large variety of built-in functions like `rgamma` for **pseudo-random-number generation**.

How would you go about **writing** such functions **yourself**?

There are a number of **general-purpose** methods for generating random numbers (I won't attempt a survey here); the one we need to look closely at, to understand the algorithms that arise later in this section, is **rejection sampling** (von Neumann 1951), which is often one of the most **computationally efficient** ways to make IID draws from a distribution.

Rejection Sampling

Example. In the spring of 1993 a survey was taken of **bicycle** and other **traffic** in the vicinity of the University of California, Berkeley, campus (Gelman et al. 1995).

As part of this survey 10 city blocks on **residential** streets with **bike routes** were chosen at random from all such blocks at Berkeley; on one of those blocks n vehicles were observed on a randomly chosen **Tuesday afternoon from 3 to 4pm**, and s of them were bicycles.

To draw inferences about the underlying **proportion θ of bicycle traffic** (PBT) on blocks similar to this one at times similar to Tuesday afternoons from 3 to 4pm, it's natural (as in the AMI mortality case study) to employ the **model**

$$\left\{ \begin{array}{c} \theta \sim \text{Beta}(\alpha_0, \beta_0) \\ (S|\theta) \sim \text{Binomial}(n, \theta) \end{array} \right\} \rightarrow (\theta|s) \sim \text{Beta}(\alpha_0 + s, \beta_0 + n - s), \tag{6}$$

provided that whatever **prior information** I have about θ can be meaningfully captured in the **Beta** family.

After **reflection** I realize that I'd be quite surprised if the PBT in residential city blocks with bike routes in Berkeley on Tuesday afternoons from 3 to 4pm was **less than 5%** or **greater than 50%**.

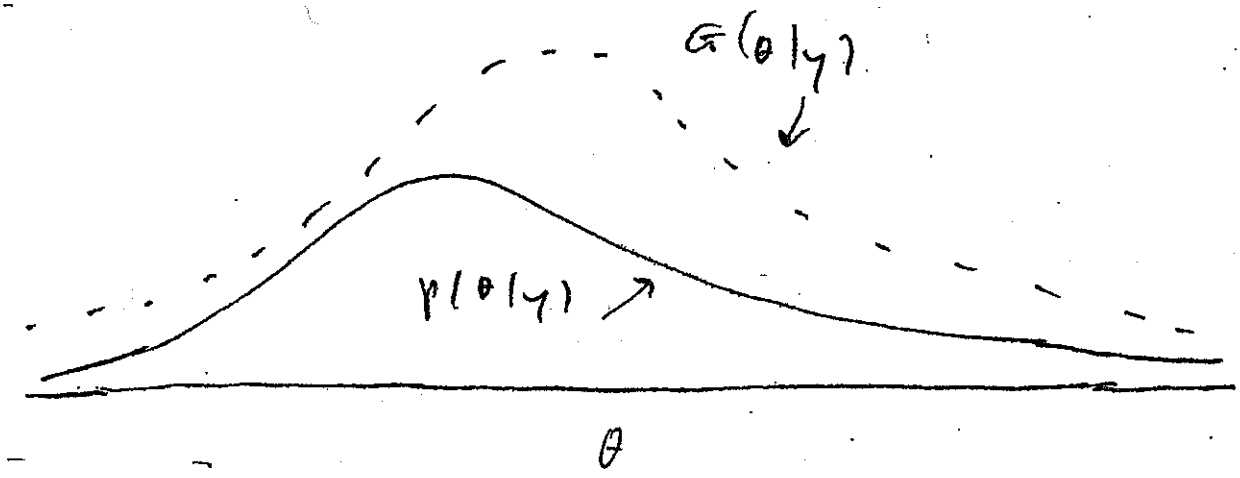
Making this operational by assuming that in the prior $p(0.05 \leq \theta \leq 0.5) =$ **0.9**, and putting **half** of the remaining prior probability in each of the **left** and **right tails** of the Beta distributions, yields (via numerical methods similar to those in the AMI case study) $(\alpha_0, \beta_0) =$ **(2.0, 6.4)** (this Beta distribution has prior mean and SD **0.24** and **0.14**, respectively).

In the city block in question the **data** came out $(n, s) =$ **(74, 16)**, so that the data mean was **0.216**, and the posterior is then $\text{Beta}(\alpha_0 + s, \beta_0 + n - s) =$ **Beta(18.0, 64.4)**.

Rejection Sampling (continued)

Pretend for the sake of illustration of **rejection sampling** that you didn't know the formulas for the mean and SD of a Beta distribution, and suppose that you wanted to use **IID Monte Carlo sampling** from the Beta$(\alpha_0 + s, \beta_0 + n - s)$ posterior to estimate the **posterior mean**.

Here's von Neumann's basic idea: suppose the target density $p(\theta|y)$ is **difficult** to sample from, but you can find an integrable **envelope function** $G(\theta|y)$ such that (a) G **dominates** p in the sense that $G(\theta|y) \geq p(\theta|y) \geq 0$ for all θ and (b) the density g obtained by normalizing G—later to be called the **proposal distribution**—is easy and fast to sample from.



Then to get a **random draw** from p, make a draw θ^* from g instead and **accept** or **reject** it according to an **acceptance probability** $\alpha_R(\theta^*|y)$; if you **reject** the draw, **repeat** this process until you accept.

von Neumann showed that the **choice**

$$\alpha_R(\theta^*|y) = \frac{p(\theta^*|y)}{G(\theta^*|y)} \tag{7}$$

correctly produces IID draws from p, and you can **intuitively** see that he's right by the following argument.

Rejection Sampling (continued)

Making a **draw** from the posterior distribution of interest is like choosing a point **at random** (in two dimensions) under the density curve $p(\theta|y)$ in such a way that **all possible points are equally likely**, and then writing down its θ value.

If you instead draw from G so that all points under G are equally likely, to get **correct** draws from p you'll need to throw away any point that falls between p and G, and this can be accomplished by **accepting** each sampled point θ^* with probability $\frac{p(\theta^*|y)}{G(\theta^*|y)}$, as von Neumann said.

A **summary** of this method is as follows.

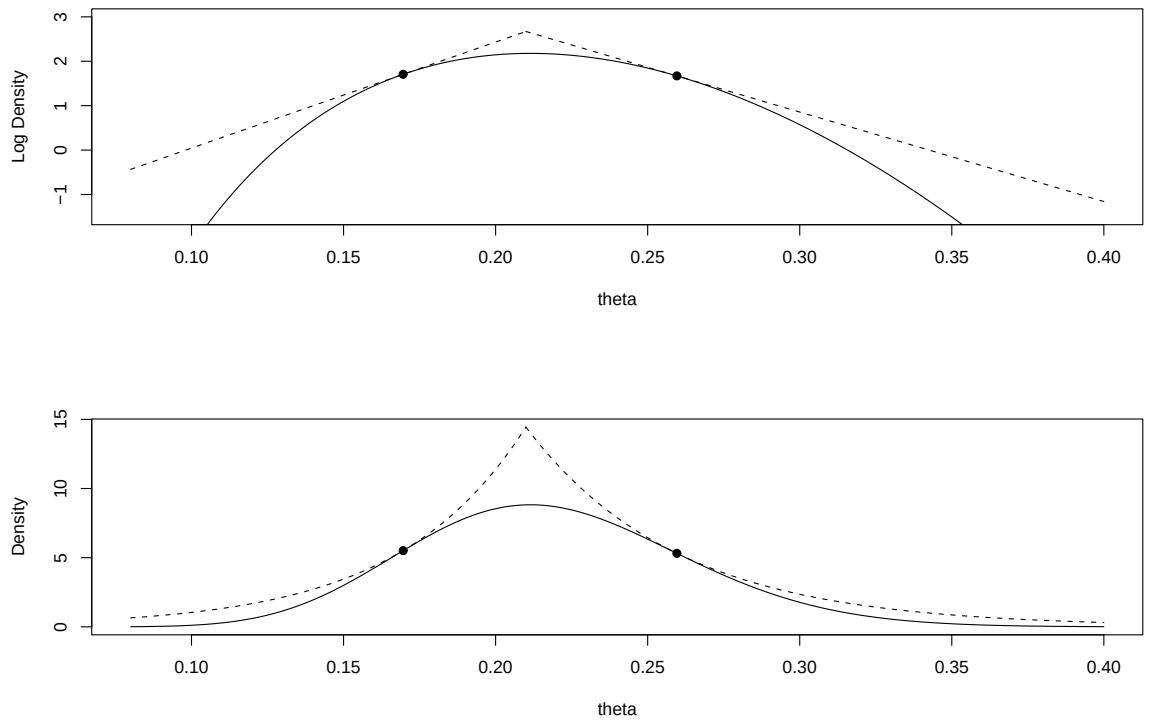
Algorithm (rejection sampling). To make m draws at random from the density $p(\theta|y)$ for real-valued θ, select an integrable **envelope function** G — which when normalized to integrate to 1 is the **proposal distribution** g — such that $G(\theta|y) \geq p(\theta|y) \geq 0$ for all θ; define the acceptance probability $\alpha_R(\theta^*|y) = \frac{p(\theta^*|y)}{G(\theta^*|y)}$; and

(8)

```
Initialize t ← 0
Repeat {
  Sample θ* ~ g(θ|y)
  Sample u ~ Uniform(0, 1)
  If u ≤ α_R(θ*|y) then
    { θ_{t+1} ← θ*;  t ← (t + 1) }
}
until t = m.
```

The **figure** below demonstrates this method on the Beta(18.0, 64.4) density arising in the **Beta-Bernoulli example** above.

Rejection Sampling (continued)



Rejection sampling permits considerable **flexibility** in the choice of **envelope function**; here, borrowing an idea from Gilks and Wild (1992), I've noted that the relevant Beta density is **log concave** (a real-valued function is log concave if its **second derivative** on the log scale is **everywhere non-positive**), meaning that it's easy to construct an envelope on that scale in a **piecewise linear** fashion, by choosing points on the log density and constructing **tangents** to the curve at those points.

The **simplest** possible such envelope involves **two line segments**, one on either side of the **mode**.

The **optimal** choice of the tangent points would maximize the marginal **probability of acceptance** of a draw in the rejection algorithm, which can be shown to be

$$\left[\int G(\theta)\, d\theta\right]^{-1}; \tag{9}$$

Rejection Sampling (continued)

in other words, you should **minimize** the area under the (un-normalized) envelope function subject to the constraint that it **dominates** the target density $p(\theta|y)$ (which makes eminently good sense).

Here this optimum turns out to be attained by locating the two tangent points at about **0.17** and **0.26**, as in the figure above; the resulting acceptance probability of about **0.75** could clearly be **improved** by adding more tangents.

Piecewise linear envelope functions on the log scale are a **good choice** because the resulting envelope density on the raw scale is a piecewise set of **scaled exponential distributions** (see the bottom panel in the figure above), from which random samples can be taken **quickly**.

A **preliminary** sample of $m_0 = 500$ IID draws from the Beta$(18.0, 64.4)$ distribution using the above rejection sampling method yields $\bar{\theta}^* = \mathbf{0.2197}$ and $\hat{\sigma} = \mathbf{0.04505}$, meaning that the posterior mean has already been estimated with an **MCSE** of only $\frac{\hat{\sigma}}{\sqrt{m_0}} = 0.002$ even with just **500** draws.

Suppose, however, that — as in equation (4) above — I want $\bar{\theta}^*$ to **differ** from the true posterior mean μ by no more than some (perhaps even smaller) **tolerance** ϵ_1 with Monte Carlo probability at least $(1 - \epsilon_2)$; then equation (5) tells me how long to **monitor** the simulation output.

For instance, to pin down **three significant figures** (sigfigs) in the posterior mean in this example with high Monte Carlo accuracy I might take $\epsilon_1 = 0.0005$ and $\epsilon_2 = 0.05$, which yields a **recommended IID sample size** of $\frac{(0.04505^2)(1.96)^2}{0.0005^2} \doteq 31,200.$

Rejection Sampling (continued)

So I take another sample of **30,700** (which is virtually instantaneous at 1.6 Unix GHz) and **merge** it with the 500 draws I already have; this yields $\bar{\theta}^* = 0.21827$ and $\hat{\sigma} = 0.04528$, meaning that the **MCSE** of this estimate of μ is $\frac{0.04528}{\sqrt{31200}} \doteq 0.00026$.

I might **announce** that I think $E(\theta|y)$ is about **0.2183**, give or take about **0.0003**, which accords well with the true value **0.2184**.

Of course, **other aspects** of $p(\theta|y)$ are equally easy to monitor; for example, if I want a Monte Carlo estimate of $p(\theta \leq q|y)$ for some q, as noted above I just work out the **proportion** of the sampled θ^* values that are no larger than q.

Or, even better, I recall that $P(A) = E[I(A)]$ for any event or proposition A, so to the **Monte Carlo dataset** (see p. 26 below) consisting of 31,200 rows and one column (the θ_t^*) I add a column monitoring the values of the **derived variable** that is 1 whenever $\theta_t^* \leq q$ and 0 otherwise; the **mean** of this derived variable is the Monte Carlo estimate of $p(\theta \leq q|y)$, and I can attach an **MCSE** to it in the same way I did with $\bar{\theta}^*$.

By this approach, for instance, the **Monte Carlo estimate** of $p(\theta \leq 0.15|y)$ based on the 31,200 draws examined above comes out $\hat{p} =$ **0.0556** with an MCSE of **0.0013**.

Percentiles are typically harder to pin down with equal Monte Carlo accuracy (in terms of sigfigs) than means or SDs, because the 0/1 scale on which they're based is **less information-rich** than the θ^* scale itself; if I wanted an MCSE for $\hat{p}$ of 0.0001 I would need an IID sample of more than **5 million draws** (which would still only take a **few seconds** at contemporary workstation speeds).

Beyond Rejection Sampling

IID sampling is not necessary. Nothing in the Metropolis-Ulam idea of Monte Carlo estimates of posterior summaries requires that these estimates be based on **IID samples from the posterior**.

This is lucky, because in practice it's often difficult, particularly when θ is a **vector of high dimension** (say k), to figure out how to make such an IID sample, via rejection sampling or other methods (e.g., imagine trying to find an **envelope function** for $p(\theta|y)$ when k is 10 or 100 or **1,000**).

Thus it's necessary to **relax** the assumption that $\theta_j^* \overset{\text{IID}}{\sim} p(\theta|y)$, and to consider samples $\theta_1^*, \ldots, \theta_m^*$ that form a **time series**: a series of draws from $p(\theta|y)$ in which θ_j^* may **depend on** $\theta_{j'}^*$ for $j' < j$.

In their pioneering paper Metropolis et al. (1953) allowed for **serial dependence** of the θ_j^* by combining von Neumann's idea of rejection sampling (which had itself only been published a few years earlier in 1951) with concepts from **Markov chains**, a subject in the theory of **stochastic processes**.

Combining **Monte Carlo sampling** with **Markov chains** gives rise to the name now used for this technique for solving the Bayesian high-dimensional integration problem: **Markov chain Monte Carlo** (MCMC).

Markov Chains

Markov chains. A **stochastic process** is just a collection of random variables $\{\theta_t^*, t \in T\}$ for some **index set** T, usually meant to stand for **time**.

In practice T can be either **discrete**, e.g., $\{0, 1, \ldots\}$, or **continuous**, e.g., $[0, \infty)$.

Markov chains are a special kind of stochastic process that can either unfold in discrete or continuous time — we'll talk here about **discrete-time Markov chains**, which is all you need for MCMC.

The **possible values** that a stochastic process can take on are collectively called the **state space** S of the process — in the simplest case S is **real-valued** and can also either be discrete or continuous.

Intuitively speaking, a Markov chain (e.g., Feller, 1968; Roberts, 1996; Gamerman, 1997) is a stochastic process unfolding in time in such a way that the **past and future states of the process are independent given the present state** — in other words, to figure out where the chain is likely to go next you don't need to pay attention to where it's been, you just need to consider **where it is now**.

More formally, a stochastic process $\{\theta_t^*, t \in T\}$, $T = \{0, 1, \ldots\}$, with state space S is a **Markov chain** if, for any set $A \in S$,

$$P(\theta_{t+1}^* \in A | \theta_0^*, \ldots, \theta_t^*) = P(\theta_{t+1}^* \in A | \theta_t^*). \qquad (10)$$

The theory of Markov chains is **harder mathematically** if S is continuous (e.g., Tierney, 1996), which is what we need for MCMC with real-valued parameters, but **most of the main ideas emerge with discrete state spaces**, and I'll assume discrete S in the intuitive discussion here.

Markov Chains (continued)

Example. For a simple example of a **discrete-time Markov chain** with a **discrete state space**, imagine a **particle** that moves around on the integers $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$, starting at 0 (say).

Wherever it's at time t — say at i — it **tosses a (3-sided) coin** and moves to $(i-1)$ with probability p_1, stays at i with probability p_2, and moves to $(i+1)$ with probability p_3, for some $0 < p_1, p_2, p_3 < 1$ with $p_1 + p_2 + p_3 = 1$ — these are the **transition probabilities** for the process.

This is a **random walk** (on the integers), and it's **clearly a Markov chain**.

Nice behavior. The most **nicely-behaved** Markov chains satisfy **three properties**:

• They're **irreducible**, which basically means that no matter where it starts the chain has to be able to reach any other state in a finite number of iterations with positive probability;

• They're **aperiodic**, meaning that for all states i the set of possible **sojourn times**, to get back to i having just left it, can have no divisor bigger than 1 (this is a **technical** condition; periodic chains still have some nice properties, but the nicest chains are aperiodic).

• They're **positive recurrent**, meaning that (a) for all states i, if the process starts at i it will return to i with probability 1, and (b) the expected length of waiting time til the first return to i is finite.

Notice that this is a bit delicate: wherever the chain is now, we insist that it **must certainly come back here**, but we don't expect to have to **wait forever** for this to happen.

Markov Chains (continued)

The random walk defined above is clearly **irreducible** and **aperiodic**, but it may not be **positive recurrent** (depending on the p_i): it's true that it has positive probability of returning to wherever it started, but (because S is **unbounded**) this probability may not be 1, and on average you may have to wait forever for it to return.

We can fix this by **bounding** S: suppose instead that $S = \{-k, -(k-1), \ldots, -1, 0, 1, \ldots, k\}$, keeping the same transition probabilities except **rejecting** any moves **outside the boundaries** of S.

This bounded random walk now satisfies **all three of the nice properties**.

The value of nice behavior. Imagine running the bounded random walk for a long time, and look at the **distribution** of the **states** it visits — over time this distribution should **settle down** (converge) to a kind of limiting, **steady-state** behavior.

This can be demonstrated by **simulation**, for instance in `R`, and using the **bounded random walk** as an example:

```
rw.sim <- function( k, p, theta.start, n.sim, seed ) {

  set.seed( seed )

  theta <- rep( 0, n.sim + 1 )

  theta[ 1 ] <- theta.start

  for ( i in 1:n.sim ) {

    theta[ i + 1 ] <- move( k, p, theta[ i ] )

  }

  return( table( theta ) )

}
```

Markov Chain Simulation

```
move <- function( k, p, theta ) {

  repeat {

    increment <- sample( x = c( -1, 0, 1 ), size = 1, prob = p )

    theta.next <- theta + increment

    if ( abs( theta.next ) <= k ) {

      return( theta.next )

      break

    }

  }

}

rosalind 17> R

R version 2.15.2 (2012-10-26)
Copyright (C) 2012 The R Foundation for Statistical Computing

> p <- c( 1, 1, 1 ) / 3

> k <- 5

> theta.start <- 0

> seed <- 9626954

> rw.sim( k, p, theta.start, 10, seed )

theta
-2 -1  0
 3  4  4

> rw.sim( k, p, theta.start, 100, seed )

-5 -4 -3 -2 -1  0  1  2  3
15 22 18 11 11  8  2  9  5
```

Simulation (continued)

```
> rw.sim( k, p, theta.start, 1000, seed )

 -5  -4  -3  -2  -1   0   1   2   3   4   5
100 113  87  65  66  68  83  95 130 124  70

> rw.sim( k, p, theta.start, 10000, seed )

  -5   -4   -3   -2   -1    0    1    2    3    4    5
 768 1052 1054 1013  980  909  877  916  939  906  587

> rw.sim( k, p, theta.start, 100000, seed )

  -5    -4    -3    -2    -1     0     1     2     3     4     5
6663 10111 10201  9864  9841  9587  9380  9305  9535  9337  6177

> rw.sim( k, p, theta.start, 1000000, seed )

   -5    -4    -3    -2    -1     0     1     2     3     4     5
65232 97285 97862 97707 96845 96007 96624 96173 96415 96317 63534
```

You can see that the distribution of where the chain has visited is **converging** to something close to **uniform** on $\{-5, -4, \ldots, 4, 5\}$, except for the effects of the **boundaries**.

Letting q_1 denote the **limiting** probability of being in one of the 9 **non-boundary** states $(-4, -3, \ldots, 3, 4)$ and q_2 be the **long-run** probability of being in one of the 2 **boundary** states $(-5, 5)$, on grounds of **symmetry** you can guess that q_1 and q_2 should satisfy

$$9q_1 + 2q_2 = 1 \quad \text{and} \quad q_1 = \frac{3}{2}q_2, \tag{11}$$

from which $(q_1, q_2) = \left(\frac{3}{31}, \frac{2}{31}\right) \doteq (0.096774, 0.064516)$.

Based on the run of **1,000,001 iterations** above we would estimate these probabilities **empirically** as $\left[\frac{97285+\ldots+96317}{(9)(1000001)}, \frac{65232+63534}{(2)(1000001)}\right] \doteq (0.0968038, 0.06438294)$.

Simulation (continued)

It should also be clear that the limiting distribution **does not depend** on the initial value of the chain:

```
> rw.sim( k, p, 5, 100000, seed )

  -5    -4    -3    -2    -1     0     1     2     3     4     5
6661 10109 10197  9859  9839  9586  9382  9307  9539  9341  6181
```

Of course, you get a **different limiting distribution** with a **different choice of** (p_1, p_2, p_3):

```
> p <- c( 0.2, 0.3, 0.5 )

> rw.sim( k, p, 0, 10, seed )

-2 -1  0
 5  5  1

> rw.sim( k, p, 0, 100, seed )

-5 -4 -3 -2 -1  0  1  2  3  4  5
 1  7 13  9 13  4  4  6  9 18 17

> rw.sim( k, p, 0, 1000, seed )

-5  -4  -3  -2  -1   0   1   2   3   4   5
 1   7  13   9  16  17  55  88 150 296 349

> rw.sim( k, p, 0, 10000, seed )

-5   -4   -3   -2   -1    0    1    2    3    4    5
 1    7   17   25   61  117  257  564 1395 3370 4187

> rw.sim( k, p, 0, 100000, seed )

-5    -4    -3    -2    -1     0     1     2     3     4     5
13    28    64   140   382   946  2236  5509 13945 34454 42284

> rw.sim( k, p, 0, 1000000, seed )
-5     -4     -3     -2     -1      0      1      2      3      4      5
77    247    584   1413   3579   8991  22214  54773 137010 342069 429044
```

Stationary Distributions

A positive recurrent and aperiodic chain is called **ergodic**, and it turns out that such chains possess a unique **stationary** (or **equilibrium**, or **invariant**) distribution π, characterized by the relation

$$\pi(j) = \sum_i \pi(i) P_{ij}(t) \tag{12}$$

for all states j and times $t \geq 0$, where $P_{ij}(t) = P(\theta_t^* = j | \theta_{t-1}^* = i)$ is the **transition matrix** of the chain.

Informally, the stationary distribution characterizes the **behavior that the chain will settle into** after it's been run for a long time, regardless of its initial state.

The point of all of this. Given a parameter vector θ and a data vector y, the Metropolis et al. (1953) idea is to **simulate** random draws from the posterior distribution $p(\theta|y)$, by constructing a **Markov chain** with the following four properties:

- It should have the **same state space** as θ,
- It should be **easy to simulate from**,
- Its **equilibrium distribution** should be $p(\theta|y)$, and
- You don't need to know the **normalizing constant** $p(\theta|y)$ when building the chain.

If you can do this, you can run the Markov chain for a long time, generating a huge sample from the posterior, and then use **simple descriptive summaries** (means, SDs, correlations, histograms or kernel density estimates) to extract any features of the posterior you want.

The Ergodic Theorem

The mathematical fact that underpins this strategy is the **ergodic theorem**: if the Markov chain $\{\theta_t^*\}$ is ergodic and f is any real-valued function for which $E_\pi|f(\theta)|$ is finite, then with probability 1 as $m \to \infty$

$$\frac{1}{m}\sum_{t=1}^{m} f(\theta_t^*) \to E_\pi[f(\theta)] = \sum_i f(i)\,\pi(i), \tag{13}$$

in which the right side is just the **expectation** of $f(\theta)$ under the stationary distribution π.

In plain English this means that — as long as the stationary distribution **is** $p(\theta|y)$ — you can learn (to arbitrary accuracy) about things like posterior means, SDs, and so on just by **waiting for stationarity to kick in** and **monitoring** thereafter **for a long enough period**.

Of course, as Roberts (1996) notes, the theorem is **silent** on the two key practical questions it raises: **how long you have to wait** for stationarity, and **how long to monitor** after that.

A third practical issue is what to use for the **initial value** θ_0^*: intuitively the **closer** θ_0^* is to the **center** of $p(\theta|y)$ the **less time** you should have to wait for stationarity.

The standard way to deal with **waiting for stationarity** is to (a) run the chain from a **good starting value** θ_0^* for b iterations, until **equilibrium** has been reached, and (b) **discard** this initial **burn-in** period.

All of this motivates the topic of **MCMC diagnostics**, which are intended to answer the following questions:

- What should I use for the **initial value** θ_0^*?
- How do I know when I've reached **equilibrium**? (This is equivalent to asking **how big** b should be.)
- Once I've reached equilibrium, how big should m be, i.e., how long should I **monitor the chain** to get posterior summaries with **decent accuracy**?

The Monte Carlo and MCMC Datasets

The basis of the Monte Carlo approach to obtaining **numerical approximations** to posterior summaries like means and SDs is the (weak) **Law of Large Numbers**: with IID sampling the **Monte Carlo estimates** of the true summaries of $p(\theta|y)$ are **consistent**, meaning that they can be made arbitrarily close to the truth with arbitrarily high probability as the number of monitoring iterations $m \to \infty$.

Before we look at how Metropolis et al. attempted to achieve the same goal with a **non-IID Monte Carlo approach**, let's look at the **practical consequences** of switching from IID to Markovian sampling.

Running the **IID rejection sampler** on the Berkeley PBT example above for a total of m monitoring iterations would produce something that might be called the **Monte Carlo dataset**, with one **row** for each **iteration** and one **column** for each **monitored quantity**; in that example it might look like this (MCSEs in parenthesis):

Iteration	θ	$I(\theta \leq 0.15)$
1	$\theta_1^* = 0.244$	$I_1^* = 0$
2	$\theta_2^* = 0.137$	$I_2^* = 1$
$\vdots$	$\vdots$	$\vdots$
$m = 31,200$	$\theta_m^* = 0.320$	$I_m^* = 0$
Mean	0.2183 (0.003)	0.0556 (0.0013)
SD	0.04528	—
Density Trace	(like the bottom plot on p. 13)	—

Running the **Metropolis sampler** on the same example would produce something that might be called the **MCMC dataset**.

It would have a **similar structure** as far as the **columns** are concerned, but the rows would be divided into **three phases**:

The MCMC Dataset (continued)

• Iteration 0 would be the value(s) used to **initialize** the Markov chain;

• Iterations 1 through b would be the **burn-in** period, during which the chain reaches its **equilibrium** or **stationary** distribution (as mentioned above, iterations 0 through b are generally **discarded**); and

• Iterations $(b+1)$ through $(b+m)$ would be the **monitoring** run, on which **summaries** of the posterior (means, SDs, density traces, ...) will be based.

In the Berkeley PBT example the **MCMC dataset** might look like this:

Iteration	Phase	θ	$I(\theta \leq 0.15)$
0	Initialization	$\theta_0^* = 0.200$	—
1	Burn-in	$\theta_1^* = 0.244$	—
⋮	⋮	⋮	⋮
$b = 500$	Burn-in	$\theta_b^* = 0.098$	—
$(b+1) = 501$	Monitoring	$\theta_{b+1}^* = 0.275$	$I_{b+1}^* = 0$
⋮	⋮	⋮	⋮
$(b+m) = 31,700$	Monitoring	$\theta_{b+m}^* = 0.120$	$I_{b+m}^* = 1$
Mean	(Monitoring Phase Only)	0.2177 (0.009)	0.0538 (0.004)
SD		0.04615	—
Density Trace		(like the bottom plot on p. 14)	—

Think of iteration number i in the Monte Carlo sampling process as a discrete **index of time** t, so that the columns of the MC and MCMC datasets can be viewed as **time series**.

An important concept from time series analysis is **autocorrelation**: the autocorrelation ρ_k of a **stationary** time series θ_t^* at **lag** k (see, e.g., Chatfield (1996)) is $\frac{\gamma_k}{\gamma_0}$, where γ_k is $C(\theta_t^*, \theta_{t-k}^*)$, the covariance of the series with itself k iterations in the past — this measures the degree to which the time series at any given moment **depends on its past history**.

The MCMC Dataset (continued)

IID draws from $p(\theta|y)$ correspond to **white noise**: a time series with **zero autocorrelations** at all lags.

This is the behavior of the columns in the **MC data set** on p. 25, produced by **ordinary rejection sampling**.

Because of the **Markov character** of the columns of the MCMC data set on p. 26, each column, when considered as a time series, will typically have **non-zero autocorrelations**, and because Markov chains use their present values to decide where to go next it shouldn't surprise you to hear that the typical behavior will be **(substantial) positive autocorrelations** — in other words, every time you get another draw from the Markov chain you get some **new information** about the posterior and a rehash of **old information** mixed in.

It's a **marvelous result** from time series analysis (the Ergodic Theorem for Markov chains on p. 24 is an example of this fact) that all of the usual descriptive summaries of the posterior are still **consistent** as long as the columns of the MCMC data set form **stationary** time series.

In other words, provided that you can achieve the **four goals** back on p. 23 that Metropolis et al. set for themselves, and provided that you only do your monitoring **after the Markov chain has reached equilibrium**, the MCMC approach and the IID Monte Carlo approach are **equally valid** (they both get the right answers), but they may well differ on their **efficiency** (the rate per iteration, or per CPU second, at which they learn about the posterior may not be the same); and if, as is typically true, the columns of the MCMC dataset have **positive autocorrelations**, this will translate into **slower learning** (larger MCSEs) than with IID sampling (compare the MCSEs on pages 25 and 26).

The Metropolis Algorithm

Metropolis et al. were able to create what people would now call a **successful MCMC algorithm** by the following means (see the excellent book edited by Gilks et al. (1996) for many more details about the MCMC approach).

Consider the **rejection sampling method** given above in (8) as a mechanism for generating realizations of a time series (where as above time indexes iteration number).

At any time t in this process you make a draw θ^* from the **proposal distribution** $g(\theta|y)$ (the normalized version of the envelope function G) and either accept a **"move"** to θ^* or reject it, according to the **acceptance probability** $\frac{p(\theta^*|y)}{G(\theta^*|y)}$; if accepted the process moves to θ^*, if not you **draw again** and discard the rejected draws until you do make a successful move.

As noted above, the stochastic process thus generated is an **IID (white noise)** series of draws from the **target distribution** $p(\theta|y)$.

Metropolis et al. had the following **beautifully simple idea** for how this may be generalized to situations where IID sampling is difficult: **they allowed the proposal distribution at time t to depend on the current value θ_t of the process**, and then — to get the right stationary distribution — if a proposed move is rejected, instead of discarding it **the process is forced to stay where it is for one iteration before trying again**.

The resulting process is a **Markov chain**, because (a) the draws are now dependent but (b) all you need to know in determining where to go next is **where you are now**.

Metropolis-Hastings

Letting θ_t stand for **where you are now** and θ^* for **where you're thinking of going**, in this approach there's **enormous flexibility** in the choice of the proposal distribution $g(\theta^*|\theta_t, y)$, even more so than in ordinary rejection sampling.

The original Metropolis et al. idea was to work with **symmetric** proposal distributions, in the sense that $g(\theta^*|\theta_t, y) = g(\theta_t|\theta^*, y)$, but Hastings (1970) pointed out that this could easily be **generalized**; the resulting method is the **Metropolis-Hastings** (MH) algorithm.

Building on the Metropolis et al. results, Hastings showed that you'll get the **correct stationary distribution** $p(\theta|y)$ for your Markov chain by making the following choice for the **acceptance probability**:

$$\alpha_{MH}(\theta^*|\theta_t, y) = \min \left\{ 1, \frac{\frac{p(\theta^*|y)}{g(\theta^*|\theta_t, y)}}{\frac{p(\theta_t|y)}{g(\theta_t|\theta^*, y)}} \right\}. \qquad (14)$$

It turns out that the proposal distribution $g(\theta^*|\theta_t, y)$ can be **virtually anything** and you'll get the **right equilibrium distribution** using the acceptance probability (14); see, e.g., Roberts (1996) and Tierney (1996) for the mild regularity conditions necessary to support this statement.

A **summary** of the method is on the next page.

It's instructive to compare (15) with (8) to see how heavily the MH algorithm **borrows from ordinary rejection sampling**, with the key difference that the proposal distribution is allowed to **change over time**.

Notice how (14) **generalizes** von Neumann's acceptance probability ratio $\frac{p(\theta^*|y)}{G(\theta^*|y)}$ for ordinary rejection sampling: the crucial part of the new MH acceptance probability becomes the **ratio of two von-Neumann-like ratios**, one for **where you are now** and one for **where you're thinking of going** (it's equivalent to work with g or G since the normalizing constant **cancels** in the ratio).

Metropolis-Hastings (continued)

Algorithm (Metropolis-Hastings sampling). To construct a **Markov chain** whose **equilibrium distribution** is $p(\theta|y)$, choose a **proposal distribution** $g(\theta^*|\theta_t, y)$, define the **acceptance probability** $\alpha_{MH}(\theta^*|\theta_t, y)$ by (14), and

```
Initialize θ_0; t ← 0
Repeat {
  Sample θ* ~ g(θ|θ_t, y)
  Sample u ~ Uniform(0, 1)
  If u ≤ α_MH(θ*|θ_t, y) then θ_{t+1} ← θ*
    else θ_{t+1} ← θ_t
  t ← (t + 1)
}
```

(15)

When the proposal distribution is **symmetric** in the Metropolis et al. sense, the acceptance probability ratio reduces to $\frac{p(\theta^*|y)}{p(\theta_t|y)}$, which is easy to **motivate intuitively**: whatever the target density is at the current point θ_t, you want to visit points of **higher density more often** and points of **lower density less often**, and it turns out that (14) does this for you in the natural and appropriate way.

As an example of the **MH algorithm in action**, consider a Gaussian model with **known mean** μ and **unknown variance** σ^2 applied to the NB10 data back in part 2a.

The **likelihood function** for σ^2, derived from the sampling model $(Y_i|\sigma^2) \overset{\text{IID}}{\sim} N(\mu, \sigma^2)$ for $i = 1, \ldots, n$, is

$$\begin{aligned} l(\sigma^2|y) &= c \prod_{i=1}^{n} (\sigma^2)^{-\frac{1}{2}} \exp\left[-\frac{(y_i-\mu)^2}{2\sigma^2}\right] \\ &= c\,(\sigma^2)^{-\frac{n}{2}} \exp\left[-\frac{\sum_{i=1}^{n}(y_i-\mu)^2}{2\sigma^2}\right]. \end{aligned} \tag{16}$$

MH Sampling (continued)

This is recognizable as a member of the **Scaled Inverse** χ^2 family $\chi^{-2}(\nu, s^2)$ (e.g., Gelman, Carlin, et al. (2003)) of distributions, which (as we saw in part 2) is a **rescaled version** of the Inverse Gamma family chosen so that s^2 is an estimate of σ^2 based upon ν "observations."

You can now convince yourself that if the **prior** for σ^2 in this model is taken to be $\chi^{-2}(\nu, s^2)$, then the **posterior** for σ^2 will also be Scaled Inverse χ^2: with this choice of prior

$$p(\sigma^2|y) = \chi^{-2}\left[\nu + n, \frac{\nu s^2 + \sum_{i=1}^n (y_i - \mu)^2}{\nu + n}\right]. \qquad (17)$$

This makes **good intuitive sense**: the **prior estimate** s^2 of σ^2 receives ν votes and the **sample estimate** $\widehat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^n (y_i - \mu)^2$ receives n votes in the **posterior weighted average estimate** $\frac{\nu s^2 + n\widehat{\sigma}^2}{\nu + n}$.

Equation (17) provides a satisfying **closed-form solution** to the Bayesian updating problem in this model (e.g., it's easy to compute posterior moments **analytically**, and you can use numerical integration or well-known approximations to the CDF of the Gamma distribution to compute percentiles).

For **illustration purposes** suppose instead that you want to use **MH sampling** to summarize this posterior.

Then your main **choice** as a user of the algorithm is the specification of the **proposal distribution** (PD) $g\left(\sigma^2|\sigma_t^2, y\right)$.

The goal in choosing the PD is getting a chain that **mixes well** (moves freely and fluidly among all of the possible values of $\theta = \sigma^2$), and nobody has (yet) come up with a **sure-fire strategy** for always succeeding at this task.

Having said that, here are **two basic ideas** that often tend to **promote good mixing**:

MH Sampling (continued)

(1) Pick a PD that looks like a **somewhat overdispersed version** of the posterior you're trying to sample from (e.g., Tierney (1996)).

Some work is naturally required to overcome the **circularity inherent** in this choice (if I fully knew $p(\theta|y)$ and all of its properties, why would I be using this algorithm in the first place?).

(2) Set up the PD so that the expected value of **where you're going to move to** (θ^*), given that you **accept a move away from where you are now** (θ_t), is to **stay where you are now**: $E_g(\theta^*|\theta_t, y) = \theta_t$.

That way, when you do make a move, there will be an **approximate left-right balance**, so to speak, in the direction you move away from θ_t, which will encourage **rapid exploration of the whole space**.

Using idea (1), a decent choice for the PD in the Gaussian model with unknown variance might well be the **Scaled Inverse χ^2 distribution**: $g\left(\sigma^2|\sigma_t^2, y\right) = \chi^{-2}\left(\nu_*, \sigma_*^2\right)$.

This distribution has **mean** $\frac{\nu_*}{\nu_*-2}\,\sigma_*^2$ for $\nu_* > 2$.

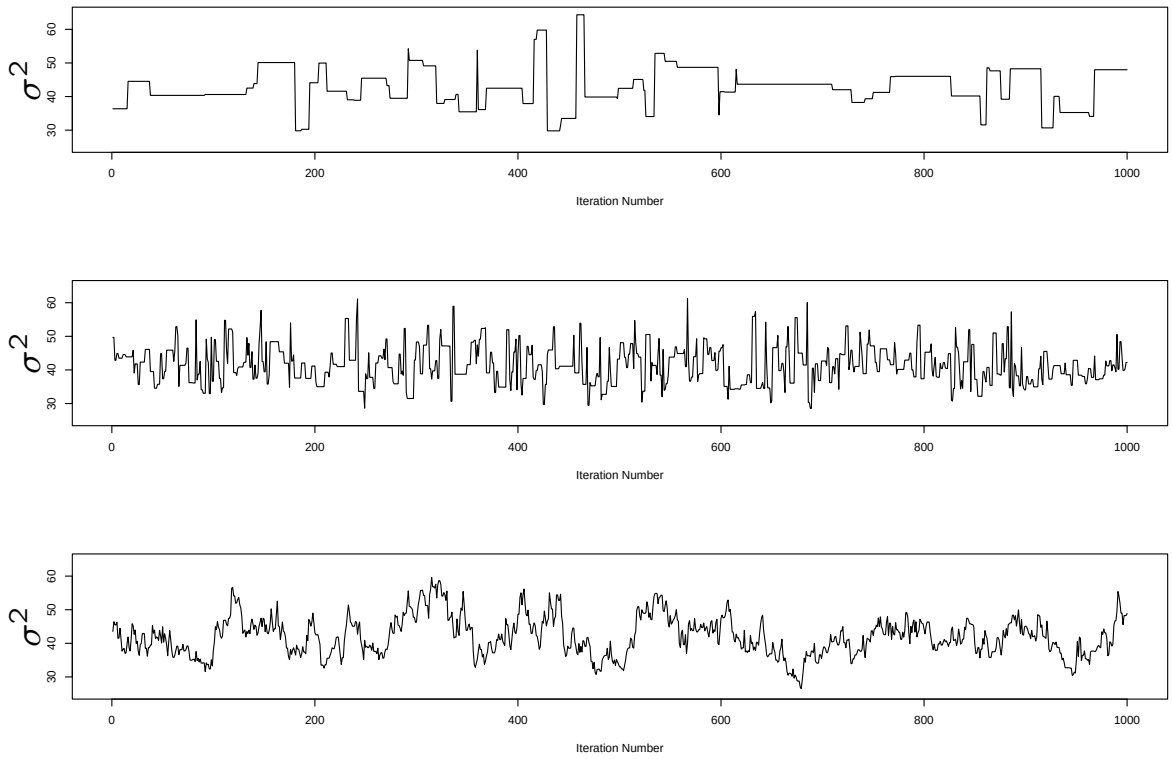
To use idea (2), then, I can **choose** any ν_* greater than 2 that I want, and as long as I take $\sigma_*^2 = \frac{\nu_*-2}{\nu_*}\sigma_t^2$ that will **center** the PD at σ_t^2 as desired.

So I'll **use**

$$g\left(\sigma^2|\sigma_t^2, y\right) = \chi^{-2}\left(\nu_*, \frac{\nu_*-2}{\nu_*}\sigma_t^2\right). \tag{18}$$

This leaves ν_* as a kind of potential **tuning constant** — the hope is that I can vary ν_* to improve the **mixing** of the chain.

MH Sampling (continued)



The above figure (motivated by an analogous plot in Gilks et al. (1996)) presents **time series traces** of some **typical output of the MH sampler** with $\nu_* = (2.5, 20, 500)$.

The **acceptance probabilities** with these values of ν_* are $(0.07, 0.44, 0.86)$, respectively.

The **SD** of the $\chi^{-2}\left(\nu_*, \frac{\nu_*-2}{\nu_*}\sigma_t^2\right)$ distribution is proportional to $\frac{\nu_*^2}{(\nu_*^2-2)^2\sqrt{\nu_*-4}}$, which decreases as ν_* increases, and this turns out to be **crucial**: when the proposal distribution SD is too **large** (small ν_*, as in the top panel in the figure), the algorithm tries to make **big jumps** around θ space (good), but almost all of them get **rejected** (bad), so there are long periods of no movement at all, whereas when the PD SD is too **small** (large ν_*; see the bottom panel of the figure), the algorithm **accepts** most of its proposed moves (good), but they're so tiny that it takes a **long time to fully explore the space** (bad).

MH Sampling (continued)

Gelman, Roberts, et al. (1995) have shown that in simple canonical problems with **approximately normal target distributions** the **optimal acceptance rate** for MH samplers like the one illustrated here is about **44%** when the vector of unknowns is one-dimensional, and this can serve as a rough guide: you can **modify the proposal distribution SD** until the acceptance rate is around the Gelman et al. target figure.

The central panel of the figure displays the **best possible MH behavior** in this problem in the family of PDs chosen.

Even with this optimization you can see that the **mixing is not wonderful**, but contemporary computing speeds enable huge numbers of draws to be collected in a short period of time, compensating for the **comparatively slow rate** at which the MH algorithm learns about the posterior distribution of interest.

In this example the unknown quantity $\theta = \sigma^2$ was **real-valued**, but there's nothing in the MH method that requires this; in principle it works equally well when θ is a **vector of any finite dimension** (look back at the algorithm in (15) to verify this).

Notice, crucially, that to **implement** this algorithm you only need to know how to calculate $p(\theta|y)$ up to a **constant multiple**, since any such constant will **cancel** in computing the acceptance probability (15) — thus you're free to work with **unnormalized versions** of $p(\theta|y)$, which is a **great advantage** in practice.

MH Sampling (continued)

There's even **more flexibility** in this algorithm than might first appear: it's often possible to identify a set A of **auxiliary variables** — typically these are **latent** (unobserved) quantities — to be sampled along with the parameters, which have the property that they **improve the mixing** of the MCMC output (even though extra time is spent in sampling them).

When the set (θ, A) of quantities to be sampled is a **vector of length** k, there's additional flexibility: you can **block update** all of (θ, A) at once, or with appropriate modifications of the acceptance probability you can divide (θ, A) up into **components**, say $(\theta, A) = (\lambda_1, \ldots, \lambda_l)$, and **update the components one at a time** (as Metropolis et al. originally proposed in 1953).

The idea in this **component-by-component** version of the algorithm, which Gilks et al. (1996) call **single-component** MH sampling, is to have k **different** proposal distributions, one for each component of θ.

Each **iteration** of the algorithm (indexed as usual by t) has k steps, indexed by i; at the beginning of iteration t you **scan** along, updating λ_1 first, then λ_2, and so on until you've updated λ_k, which **concludes** iteration t.

Let $\lambda_{t,i}$ stand for the **current state** of component i at the end of iteration t, and let λ_{-i} stand for the (θ, A) vector with component i **omitted** (the notation gets awkward here; it can't be helped).

The proposal distribution $g_i(\lambda_i^* | \lambda_{t,i}, \lambda_{t,-i}, y)$ for component i is allowed to **depend** on the most recent versions of all components of (θ, A); here $\lambda_{t,-i}$ is the **current state** of λ_{-i} after step $(i-1)$ of iteration t is finished, so that components 1 through $(i-1)$ have been updated **but not the rest**.

Gibbs Sampling

The **acceptance probability** for the proposed move to λ_i^* that creates the **correct equilibrium distribution** turns out to be

$$\alpha_{MH}(\lambda_i^*|\lambda_{t,-i},\lambda_{t,i},y) = \min\left[1, \frac{p(\lambda_i^*|\lambda_{t,-i},y)\, g_i(\lambda_{t,i}|\lambda_i^*,\lambda_{t,-i},y)}{p(\lambda_{t,i}|\lambda_{t,-i},y)\, g_i(\lambda_i^*|\lambda_{t,i},\lambda_{t,-i},y)}\right]. \tag{19}$$

The distribution $p(\lambda_i|\lambda_{-i},y)$ appearing in (19), which is called the **full conditional** distribution for λ_i, has a natural interpretation: it represents the posterior distribution for the relevant portion of (θ, A) **given** y **and the rest of** (θ, A).

The full conditional distributions act like **building blocks** in constructing the **complete posterior distribution** $p(\theta|y)$, in the sense that **any multivariate distribution is uniquely determined by its set of full conditionals** (Besag (1974)).

An **important special case** of **single-component MH sampling** arises when the **proposal distribution** $g_i(\lambda_i^*|\lambda_{t,i},\lambda_{t,-i},y)$ for component i is chosen to be the **full conditional** $p(\lambda_i^*|\lambda_{t,-i},y)$ for λ_i: you can see from (19) that when this choice is made a **glorious cancellation** occurs and the **acceptance probability is 1**.

This is **Gibbs sampling**, independently (re)discovered by Geman and Geman (1984): the Gibbs recipe is to **sample from the full conditionals** and **accept all proposed moves**.

Even though it's **just a version of MH**, Gibbs sampling is important enough to merit a **summary** of its own.

Single-element Gibbs sampling, in which each real-valued coordinate $(\theta_1,\ldots,\theta_k)$ gets updated in turn, is probably the **most frequent** way Gibbs sampling gets used, so that's what I'll summarize ((20) details Gibbs sampling in the case with **no auxiliary variables** A, but the algorithm **works equally well** when θ is replaced by (θ, A) in the summary).

Gibbs Sampling (continued)

Algorithm (Single-element Gibbs sampling). To construct a **Markov chain** whose **equilibrium distribution** is $p(\theta|y)$ with $\theta = (\theta_1, \ldots, \theta_k)$,

Initialize $\theta^*_{0,1}, \ldots, \theta^*_{0,k}$; $t \leftarrow 0$
Repeat {
  Sample $\theta^*_{t+1,1} \sim p(\theta_1|y, \theta^*_{t,2}, \theta^*_{t,3}, \theta^*_{t,4}, \ldots, \theta^*_{t,k})$
  Sample $\theta^*_{t+1,2} \sim p(\theta_2|y, \theta^*_{t+1,1}, \theta^*_{t,3}, \theta^*_{t,4}, \ldots, \theta^*_{t,k})$
  Sample $\theta^*_{t+1,3} \sim p(\theta_3|y, \theta^*_{t+1,1}, \theta^*_{t+1,2}, \theta^*_{t,4}, \ldots, \theta^*_{t,k})$
  ⋮ ⋮ ⋮ ⋮ ⋮ ⋮
  Sample $\theta^*_{t+1,k} \sim p(\theta_k|y, \theta^*_{t+1,1}, \theta^*_{t+1,2}, \theta^*_{t+1,3}, \ldots, \theta^*_{t+1,k-1})$
  $t \leftarrow (t+1)$
}

(20)

Example: the **NB10 Data**. Recall from the posterior predictive plot toward the end of part 2a that the Gaussian model for the NB10 data was inadequate: the tails of the data distribution are **too heavy** for the Gaussian.

It was also clear from the normal qqplot that the data are **symmetric**.

This suggests thinking of the NB10 data values y_i as like draws from a t **distribution** with fairly small degrees of freedom ν.

One way to write this model is

$$\begin{aligned} (\mu, \sigma^2, \nu) &\sim p(\mu, \sigma^2, \nu) \\ (y_i|\mu, \sigma^2, \nu) &\overset{\text{IID}}{\sim} t_\nu(\mu, \sigma^2), \end{aligned} \tag{21}$$

where $t_\nu(\mu, \sigma^2)$ denotes the **scaled** t**-distribution** with mean μ, scale parameter σ^2, and shape parameter ν.

Model Expansion

This distribution has variance $\sigma^2 \left(\frac{\nu}{\nu-2}\right)$ for $\nu > 2$ (so that shape and scale are mixed up, or **confounded**, in $t_\nu(\mu, \sigma^2)$) and may be thought of as the distribution of the quantity $\mu + \sigma\, e$, where e is a draw from the **standard** t distribution that is tabled at the back of all introductory statistics books.

However, a **better way** to think about model (21) is as follows.

It's a fact from **basic distribution theory**, probably of more interest to Bayesians than frequentists, that the t distribution is an

Inverse Gamma mixture of Gaussians.

This just means that to generate a t random quantity you can first draw from an Inverse Gamma distribution and then draw from a Gaussian **conditional** on what you got from the Inverse Gamma.

($\lambda \sim \Gamma^{-1}(\alpha, \beta)$ just means that $\lambda^{-1} = \frac{1}{\lambda} \sim \Gamma(\alpha, \beta)$).

In more detail, $(y|\mu, \sigma^2, \nu) \sim t_\nu(\mu, \sigma^2)$ is the same as the **hierarchical model**

$$\begin{aligned} (\lambda|\nu) &\sim \Gamma^{-1}\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \\ (y|\mu, \sigma^2, \lambda) &\sim N\left(\mu, \lambda\, \sigma^2\right). \end{aligned} \tag{22}$$

Putting this together with the **conjugate prior** for μ and σ^2 we looked at earlier in the Gaussian model gives the following HM for the NB10 data:

$$\begin{aligned} \nu &\sim p(\nu) \\ \sigma^2 &\sim \text{SI-}\chi^2\left(\nu_0, \sigma_0^2\right) \\ \left(\mu|\sigma^2\right) &\sim N\left(\mu_0, \frac{\sigma^2}{\kappa_0}\right) \\ (\lambda_i|\nu) &\overset{\text{IID}}{\sim} \Gamma^{-1}\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \\ \left(y_i|\mu, \sigma^2, \lambda_i\right) &\overset{\text{indep}}{\sim} N\left(\mu, \lambda_i\, \sigma^2\right). \end{aligned} \tag{23}$$

Remembering also from introductory statistics that the Gaussian distribution is the **limit** of the t family as $\nu \to \infty$, you can see that the idea here has been to **expand** the Gaussian model by embedding it in the richer t family, of which it's a special case with $\nu = \infty$.

Implementing Gibbs

Model expansion is often the best way to deal with **uncertainty in the modeling process**: when you find deficiencies of the current model, **embed it in a richer class**, with the model expansion in directions suggested by the deficiencies (we'll also see this method in action again later).

The MCMC Dataset. Imagine trying to do **Gibbs sampling** on model (21), with the parameter vector $\theta = (\mu, \sigma^2, \nu)$.

Carrying out the iterative program described in (20) above would produce the following **MCMC Dataset**:

Iteration	Phase	μ	σ^2	ν
0	Initializing	μ_0	σ_0^2	ν_0
1	Burn-In	$\mu_1(y, \sigma_0^2, \nu_0)$	$\sigma_1^2(y, \mu_1, \nu_0)$	$\nu_1(y, \mu_1, \sigma_1^2)$
2	Burn-In	$\mu_2(y, \sigma_1^2, \nu_1)$	$\sigma_2^2(y, \mu_2, \nu_1)$	$\nu_1(y, \mu_2, \sigma_2^2)$
.	.	.	.	.
b	Burn-In	μ_b	σ_b^2	ν_b
$(b+1)$	Monitoring	μ_{b+1}	σ_{b+1}^2	ν_{b+1}
$(b+2)$	Monitoring	μ_{b+2}	σ_{b+2}^2	ν_{b+2}
.	.	.	.	.
$(b+m)$	Monitoring	μ_{b+m}	σ_{b+m}^2	ν_{b+m}

Looking at iterations 1 and 2 you can see that, in addition to y, the sampler makes use **only of parameter values in the current row and the previous row** (this illustrates the Markov character of the samples).

As we've seen above, at the end of the $(b+m)$ iterations, if you want (say) the **marginal posterior** for μ, $p(\mu|y)$, all you have to do is take the m values $\mu_{b+1}, \ldots, \mu_{b+m}$ and summarize them in any ways that interest you: their sample mean is your **simulation estimate** of the posterior mean of μ, their sample histogram (or, better, their **kernel density trace**) is your simulation estimate of $p(\mu|y)$, and so on.

Practical Issues

Implementation Details. (1) How do you figure out the **full conditionals**, and how do you sample from them?

(2) What should you use for **initial values**?

(3) How **large** should b and m be?

(4) More generally, how do you know when the chain has **reached equilibrium**?

Questions (3–4) fall under the heading of **MCMC diagnostics**, which I'll cover a bit later, and I'll address question (2) in the **case studies** below.

Computing the full conditionals. For a simple example of working out the full conditional distributions, consider the **conjugate Gaussian model** we looked at earlier:

$$\begin{aligned} \sigma^2 &\sim \text{SI-}\chi^2(\nu_0, \sigma_0^2) \\ (\mu|\sigma^2) &\sim N\left(\mu_0, \frac{\sigma^2}{\kappa_0}\right) \\ (Y_i|\mu,\sigma^2) &\overset{\text{IID}}{\sim} N\left(\mu, \sigma^2\right). \end{aligned} \tag{24}$$

The full conditional distribution for μ in this model is $p(\mu|\sigma^2, y)$, **considered as a function of μ for fixed σ^2 and y** — but this is just

$$\begin{aligned} p(\mu|\sigma^2, y) &= \frac{p(\mu, \sigma^2, y)}{p(\sigma^2, y)} \\ &= c\, p(\mu, \sigma^2, y) \\ &= c\, p(\sigma^2)\, p(\mu|\sigma^2)\, p(y|\mu, \sigma^2) \\ &= c\, \exp\left[-\frac{\kappa_0}{2\sigma^2}(\mu-\mu_0)^2\right] \prod_{i=1}^{n} \exp\left[-\frac{1}{2\sigma^2}(y_i-\mu)^2\right]. \end{aligned} \tag{25}$$

Full Conditionals

From this

$$p(\mu|\sigma^2, y) = c \exp\Big[-\frac{\kappa_0}{2\sigma^2}(\mu-\mu_0)^2\Big] \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right].$$

Expanding out the squares, **collecting** powers of μ, and **completing the square** in μ gives

$$p(\mu|\sigma^2, y) = c \exp\left[-\frac{\kappa_0+n}{2\sigma^2}\left(\mu - \frac{\kappa_0\mu_0 + n\bar{y}}{\kappa_0+n}\right)^2\right], \qquad (26)$$

from which it's clear that the **full conditional for μ in model (24)** is

$$(\mu|\sigma^2, y) \sim N\left(\frac{\kappa_0\mu_0 + n\bar{y}}{\kappa_0+n}, \frac{\sigma^2}{\kappa_0+n}\right). \qquad (27)$$

Similarly, the full conditional for σ^2 in this model, $p(\sigma^2|\mu, y)$, **considered as a function of σ^2 for fixed μ and y**, is just

$$\begin{aligned}
p(\sigma^2|\mu, y) &= \frac{p(\sigma^2, \mu, y)}{p(\mu, y)} \\
&= c\, p(\sigma^2, \mu, y) \qquad (28)\\
&= c\, p(\sigma^2)\, p(\mu|\sigma^2)\, p(y|\mu, \sigma^2) \\
&= c\, \left(\sigma^2\right)^{-(1+\frac{1}{2}\nu_0)} \exp\left(\frac{-\nu_0\,\sigma_0^2}{2\sigma^2}\right) \cdot \\
&\qquad \left(\sigma^2\right)^{-\frac{1}{2}} \exp\Big[-\frac{\kappa_0}{2\sigma^2}(\mu-\mu_0)^2\Big] \cdot \\
&\qquad \left(\sigma^2\right)^{-\frac{n}{2}} \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2\right].
\end{aligned}$$

When this is **simplified** you get

Full Conditionals (continued)

$$p(\sigma^2|\mu, y) = c\,\left(\sigma^2\right)^{-(1+\frac{\nu_0+1+n}{2})}\,\exp\left[-\frac{\nu_0\sigma_0^2+\kappa_0(\mu-\mu_0)^2+ns_\mu^2}{2\sigma^2}\right],$$

where $s_\mu^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i-\mu)^2$.

From the **form** of this distribution it becomes clear that

$$(\sigma^2|\mu, y) \sim \text{SI-}\chi^2\left(\nu_0+1+n, \frac{\nu_0\sigma_0^2+\kappa_0(\mu-\mu_0)^2+ns_\mu^2}{\nu_0+1+n}\right). \quad (29)$$

Thus in **conjugate situations** the full conditional distributions have **conjugate forms**, which are **tedious but straightforward** to compute.

Both the directness and the tedium of this calculation suggest that it should be possible to write a **computer program** to work out the full conditionals for you, and indeed at least **three such programs now exist**:

- `WinBUGS`, a fairly **general-purpose MCMC program** produced by David Spiegelhalter and others at the MRC Biostatistics Unit in Cambridge, UK (Spiegelhalter et al., 1997), intended for use on `Windows` machines;
- `JAGS`, with **syntax modeled** after `WinBUGS` and **available** via `Windows`, `Mac OS X` and `Linux` **binaries**; and
- `MLwiN`, a program that does both **maximum-likelihood** and Bayesian calculations in **hierarchical (multilevel) models** (Rasbash et al. 2010).

`WinBUGS` and `JAGS` are available for **free downloading** at

`www.mrc-bsu.cam.ac.uk/bugs` and

`mcmc-jags.sourceforge.net`;

`MLwiN` has a nominal charge and can be downloaded from the web page of the **Multilevel Models Project**,

`multilevel.ioe.ac.uk`.

Why the Metropolis Algorithm Works

Here's a sketch of the crucial part of the **proof**, based on an argument in Gamerman (1997), of the **validity** of the Metropolis algorithm, in the case of a **discrete** (finite or countably infinite) state space S (see chapter 1 in Gilks et al. 1996 for a proof sketch when S is **continuous**).

I see now that my **Markov chain notation** up until this point has not been consistent enough to keep the proof from becoming confusing, so let's start over again with the following **notation**.

A **stochastic process** $\{\theta_t^*, t \in T\}, T = \{0, 1, \ldots\}$ on a discrete state space S is a **Markov chain** iff

$$P(\theta_{t+1}^* = y | \theta_t^* = x, \theta_{t-1}^* = x_{n-1}, \ldots, \theta_0^* = x_0) = P(\theta_{t+1}^* = y | \theta_t^* = x) \tag{30}$$

for all $x_0, \ldots, x_{t-1}, x, y \in S$.

In general $P(\theta_{t+1}^* = y | \theta_t^* = x)$ depends on x, y, and t, but if the probability of transitioning from x to y at time t is **constant** in t things will clearly be simpler; such chains are called **homogeneous** (confusingly, some sources call them **stationary**, but that terminology seems well worth avoiding).

The **random walk** described earlier is obviously a homogeneous Markov chain, and so are any Markov chains generated by the **MH algorithm**; I'll **assume homogeneity** in what follows.

Under **homogeneity** it makes sense to talk about the **transition probability**

$$P(x, y) = P(\theta_{t+1}^* = y | \theta_t^* = x) \quad \text{for all } t, \tag{31}$$

which **satisfies**

$$P(x, y) \geq 0 \text{ for all } x, y \in S \quad \text{and} \quad \sum_{y \in S} P(x, y) = 1 \text{ for all } x \in S. \tag{32}$$

Metropolis Proof Sketch

When S is discrete a **transition matrix** P can be defined with element (i, j) given by $P(x_i, x_j)$, where x_i is the ith element in S according to whatever **numbering convention** you want to use (the second part of (32) implies that the row sums of such a matrix are always 1; this is the defining condition for a **stochastic matrix**).

Suppose the chain is **initialized** at time 0 by making a **draw** from a probability distribution $\pi_0(x) = P(\theta_0^* = x)$ on S (**deterministically** starting it at some point x_0 is a special case of this); then the probability distribution $\pi_1(y)$ for where it will be at time 1 is

$$\begin{aligned} \pi_1(y) &= P(\theta_1^* = y) \\ &= \sum_{x \in S} P(\theta_0^* = x, \theta_1^* = y) \\ &= \sum_{x \in S} P(\theta_0^* = x)\, P(\theta_1^* = y | \theta_0^* = x) \qquad (33) \\ &= \sum_{x \in S} \pi_0(x)\, P(x, y), \end{aligned}$$

which can be written in **vector** and **matrix** notation as

$$\pi_1 = \pi_0\, P, \qquad (34)$$

where π_0 and π_1 are regarded as **row vectors**.

Then by the **same reasoning**

$$\pi_2 = \pi_1\, P = (\pi_0\, P)P = \pi_0\, P^2, \qquad (35)$$

and **in general**

$$\pi_t = \pi_0\, P^t. \qquad (36)$$

For **simple** Markov chains this can be used to work out the **long-run** behavior of the chain as $t \to \infty$, but this becomes **algebraically prohibitive** as the **transition behavior** of the chain increases in **complexity**.

Proof Sketch (continued)

In any case for **ergodic** Markov chains the limiting behavior $\pi(y)$ is **independent** of π_0 and turns out to be characterized by the relation

$$\pi(y) = \sum_{x \in S} \pi(x)\, P(x, y), \quad \text{or} \quad \pi = \pi P, \tag{37}$$

which defines the **stationary distribution** π of the chain.

As we've seen above, the hard bit in verifying the **validity** of the Metropolis algorithm is demonstrating that the Markov chain created by running the algorithm has the **correct stationary distribution**, namely the target posterior $p(\theta|y)$; one way to do this is the following.

It's possible to imagine running any **homogeneous** Markov chain $\{\theta_t^*, t = 0, 1, \ldots\}$ with transition probabilities $P(x, y)$ **backwards** in time.

This new **reverse-time** stochastic process can be shown also to be a **Markov chain**, although it may not be **homogeneous**.

If it **is** homogeneous, and if in addition the reverse-time process has the **same transition probabilities** as the original process, the Markov chain is said to be **reversible**; all such chains satisfy the **detailed balance equation**

$$\pi(x)\, P(x, y) = \pi(y)\, P(y, x) \text{ for all } x, y \in S. \tag{38}$$

It turns out that if there's a distribution π satisfying (38) for an **irreducible** Markov chain, then the chain is **positive recurrent** (and therefore **ergodic**) and **reversible**, and its **stationary distribution** is π (sum (38) over y to get (37)).

Proof Sketch (continued)

In other words, if you're trying to create an **ergodic Markov chain** and you want it to have some **target stationary distribution** π, one way to achieve this goal is to ensure that the chain is **irreducible** and that its **transition probabilities** $P(x, y)$ satisfy **detailed balance** with respect to the target π.

Any **reasonable** proposal distribution in the Metropolis algorithm will yield an **irreducible** Markov chain, so the interesting bit is to **verify detailed balance**; the argument proceeds as follows.

Consider a given **target distribution** p_x on S; we're trying to construct a Markov chain with **stationary distribution** π such that $\pi(x) = p_x$ for all $x \in S$.

The **Metropolis algorithm** — (15), with the special case of the **acceptance probabilities** (14) reducing to the **simpler form** $\min\left[1, \frac{p(\theta^*|y)}{p(\theta_t|y)}\right]$ by the assumption of a **symmetric** proposal distribution — actually involves **two related Markov chains**: the (**less interesting**) chain that you could create by **accepting all proposed moves**, and the (**more interesting**) chain created by the **actual algorithm**.

Let $Q(x, y)$ be any **irreducible** transition matrix on S such that $Q(x, y) = Q(y, x)$ for all $x, y \in S$; this is the transition matrix for the (**less interesting**) chain induced by the proposal distribution.

Define the (**more interesting**) chain $\{\theta_t^*, t = 0, 1, \ldots\}$ (the **actual** Metropolis chain) as having transitions from x to y proposed according to $Q(x, y)$, except that the proposed value for θ_{t+1}^* is **accepted** with probability $\min\left(1, \frac{p_y}{p_x}\right)$ and **rejected** otherwise, **leaving** the chain in state x.

Proof Sketch (continued)

The **transition probabilities** $P(x,y)$ for the **Metropolis chain** are as follows: for $y \neq x$, and denoting by A_{xy} the event that the proposed move from x to y is **accepted**,

$$\begin{aligned} P(x,y) &= P\big(\theta^*_{t+1} = y | \theta^*_t = x\big) \\ &= P\big(\theta^*_{t+1} = y, A_{xy} | \theta^*_t = x\big) + P\big(\theta^*_{t+1} = y, \text{not } A_{xy} | \theta^*_t = x\big) \\ &= P\big(\theta^*_{t+1} = y | A_{xy}, \theta^*_t = x\big)\ P(A_{xy} | \theta^*_t = x) \qquad (39) \\ &= Q(x,y) \min\left(1, \frac{p_y}{p_x}\right). \end{aligned}$$

A **similar calculation** shows that for $y = x$

$$P(x,x) = Q(x,x) + \sum_{y \neq x} Q(x,y) \left[1 - \min\left(1, \frac{p_y}{p_x}\right)\right], \qquad (40)$$

but this is not needed to show **detailed balance** because (38) is **trivially satisfied** when $y = x$.

When $y \neq x$ there are **two cases**: $p_y \geq p_x > 0$ (I'll give details in **this case**) and $0 < p_y < p_x$ (the other case follows **analogously**).

If $p_y \geq p_x$, **note** that $\min\left(1, \frac{p_y}{p_x}\right) = 1$ and $\min\left(1, \frac{p_x}{p_y}\right)\ p_y = \min\left(p_y, \frac{p_x}{p_y} p_y\right) = \min(p_y, p_x) = p_x$; **then**

$$\begin{aligned} p_x\, P(x,y) &= p_x\, Q(x,y) \min\left(1, \frac{p_y}{p_x}\right) = p_x\, Q(x,y) \\ &= p_x\, Q(y,x) = Q(y,x)\ \min\left(1, \frac{p_x}{p_y}\right)\ p_y \qquad (41) \\ &= p_y\, P(y,x) \end{aligned}$$

and the proof of **detailed balance**, and with it the **validity** of the **Metropolis algorithm**, is **complete**.

Directed Acyclic Graphs

WinBUGS achieves its **generality** by means of two ideas:

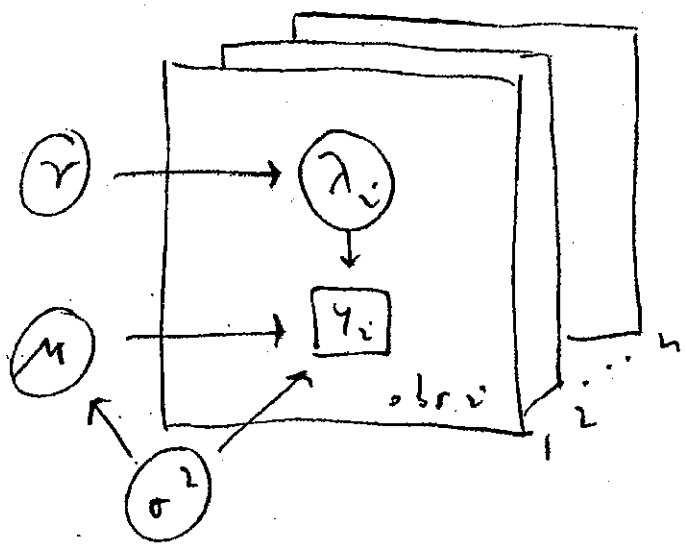
(1) Viewing Bayesian models as
directed (acyclic) graphs (DAGs).

The **conditional independence** nature of **Bayesian hierarchical models**—in which quantities in the model depend on things one layer higher in the hierarchy but no higher (e.g., in the NB10 t model (23) the y_i depend on $(\mu, \sigma^2, \lambda_i)$ but not on ν)—lends itself to thinking of all quantities in such models as **nodes** in a **directed graph**.

A DAG can be thought of as a **picture** in which known and unknown quantities are represented either by **squares** (for knowns) or **circles** (for unknowns), connected by **arrows** (from the **parents** to the **children**) that indicate the direction of the stochastic dependence.

The **acyclic** assumption means that by following the directions of the arrows it's impossible to return to a node once you've left it, and **stacked sheets** indicate **repetition** (e.g., across conditionally IID data values).

Here's a DAG for the **NB10 model** based on the t distribution.



Adaptive Rejection Sampling

(2) Employing **adaptive-rejection sampling** (Gilks and Wild, 1992) to generate the random draws from the full conditional distributions, when they don't have **simple recognizable forms**.

As we've seen, **rejection sampling** is a general method for sampling from a given density $p(\theta|y)$, which requires an **envelope function** G that dominates p (chosen so that $G(\theta|y) \geq p(\theta|y)$ for all θ).

A restatement of the **algorithm** for normalized G (e.g., Ripley 1987) is

```
Repeat {
  Sample a point theta from G ( . | y );
  Sample a Uniform( 0, 1 ) random variable U;
  If U <= p ( theta | y ) / G ( theta | y ) accept theta;
  }
until one theta is accepted.
```

If $p(\theta|y)$ is **expensive** to evaluate, time can be saved by identifying **squeezing functions** $a(\theta|y)$ and $b(\theta|y)$ with $b(\theta|y) \leq p(\theta|y) \leq a(\theta|y)$; to use these, replace the **acceptance step** above (line 4 in the algorithm) by

```
If U > a( theta | y ) / G( theta | y ) reject theta;
  else if U <= b( theta | y ) / G( theta | y ) accept theta;
  else if U <= p( theta | y ) / G( theta | y ) accept theta.
```

Adaptive rejection sampling (ARS; Gilks and Wild 1992) is a method of **adaptive envelope construction** that works as a basis for Gibbs sampling if all of the full conditional densities are **log concave** (formally, a function $p(\theta|y)$ of a vector argument θ is log concave if the **determinant** of

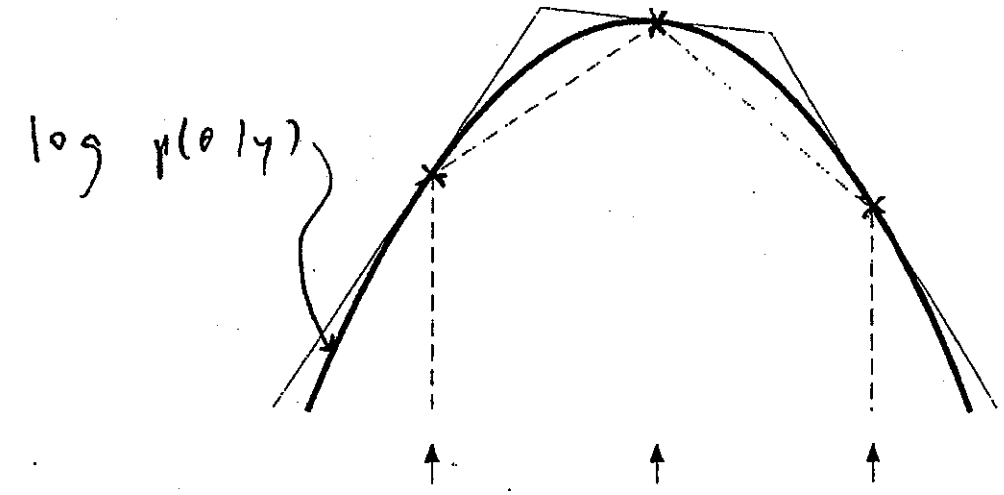
$$\frac{d^2 \log g}{dy\, dy^T} \tag{42}$$

is **non-positive**).

ARS (continued)

For univariate θ the idea (see the figure on p. 13) is that an envelope function $\log G_S(\theta|y)$ can be constructed on the log scale by drawing **tangents** to $\log p(\theta|y)$ at each point in a given set of θ values S.

An **envelope** between any two adjacent points is then constructed from the tangents at each end of the interval defined by the points:



the tangents form the envelope function on the log scale, & the secants form a lower squeezing function

The envelope is **linear on the log scale**, so rejection sampling on the original scale is performed with **scaled exponential** distributions (as noted earlier, this can be done efficiently), and you get a **lower squeezing function** for free.

The useful thing about this idea is that the envelope can be constructed **adaptively**, by adding points to S as new θ are sampled—thus the envelope **improves** as more samples are drawn.

WinBUGS (continued)

WinBUGS uses a **hierarchy** of methods to sample from the full conditionals: it first tries to verify **conjugacy**; if that fails it then tries to verify **log concavity** of the full conditionals and uses **ARS** if so; and if that fails winBUGS **switches over to (non-Gibbs) Metropolis-Hastings sampling**.

Log concavity includes many, **but not all**, distributions occurring in standard models, e.g., a uniform $U(a, b)$ prior on the degrees of freedom parameter ν in the NB10 t model **fails** log-concavity.

In classic BUGS such distributions must be **discretized** (BUGS allows discrete variables to have 500 possible values, which generally leads to **quite accurate approximations**).

Running classic BUGS. You make **four kinds** of files:

(1) a **program** file, with suffix .bug, containing the specification of your **model**;

(2) one or more **data** files, with suffix .dat;

(3) an **initial values** file, with suffix .in; and

(4) a **command** file with suffix .cmd, containing instructions that specify the burn-in and monitoring phases.

Here's the **data file** in the NB10 example.

```
list( y = c(409., 400., 406., 399., 402., 406., 401., 403.,
  401., 403., 398., 403., 407., 402., 401., 399., 400., 401.,
                    [ several lines omitted ]
  401., 407., 412., 375., 409., 406., 398., 406., 403., 404.),
 grid = c(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                    [ several lines omitted ]
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 )
```

BUGS (continued)

And here are the BUGS **program** (.bug) and **initial values** (.in) files in the NB10 example.

```
model nb10;

const

  n = 100, g = 100;

var

  mu, tau, u, grid[ g ], nu, y[ n ], sigma;

data in "nb10.dat";
inits in "nb10.in";

{

  mu ~ dnorm( 0.0, 1.0E-6 );
  tau ~ dgamma( 0.001, 0.001 );  # specifying the
  u ~ dcat( grid[ ] );           # prior distributions
  nu <- 2.0 + u / 10.0;

  for ( i in 1:n ) {
                                 # specifying the
    y[ i ] ~ dt( mu, tau, nu );  # likelihood

  }
                                 # defining any other
  sigma <- 1.0 / sqrt( tau );    # quantities to be
                                 # monitored
}
             Initial values

list( mu = 404.59, u = 30, tau = 0.04,
  seed = 90915314 )
```

Implementation Details

Here are two BUGS **command** (.cmd) files in the NB10 example.

```
compile( "nb10-1.bug" )    |   compile( "nb10-1.bug" )
update( 1000 )              |   update( 2000 )
monitor( mu )               |   monitor( mu, 8 )
monitor( sigma )            |   monitor( sigma, 8 )
monitor( nu )               |   monitor( nu, 8 )
update( 5000 )              |   update( 40000 )
q( )                        |   q( )
```

Some Details. (1) The priors: (a) I want to use a **diffuse** prior for μ, since I don't know anything about the true weight of NB10 *a priori*.

The phrase `mu ~ dnorm( 0.0, 1.0E-6 )` in BUGS-speak means that μ has a Gaussian prior with mean 0 and **precision** 10^{-6}, i.e., SD $= 1/\sqrt{\text{precision}} = 1,000$, i.e., as far as I'm concerned *a priori* μ could be **just about anywhere** between $-3,000$ and $3,000$.

(b) Similarly I want a **diffuse** prior for σ^2, or equivalently for the **precision** $\tau = \frac{1}{\sigma^2}$.

As we saw in the Poisson LOS case study, one popular **conventional** choice is $\tau \sim \Gamma(\epsilon, \epsilon)$ for a small ϵ like 0.001, which in BUGS-speak is said `tau ~ dgamma( 0.001, 0.001 )`.

This distribution is **very close to flat** over an extremely wide range of the interval $(0, \infty)$, although it does have a **nasty spike** at 0 (as $\tau \downarrow 0, \Gamma(\epsilon, \epsilon)(\tau) \uparrow \infty$).

As noted earlier, the idea behind diffuse priors is to make them approximately **constant in the region in which the likelihood is appreciable**.

For this purpose it's useful to remember what the **frequentist answers** for μ and σ would be, at least in the Gaussian model we looked at earlier.

Recall that the 95% confidence interval (CI) for μ came out $(403.3, 405.9)$ — so you can guess that the **likelihood** for μ would be **non-negligible** in the range from (say) 402 to 407.

Diffuse Priors

As for σ (or σ^2 or τ), in the model $(Y_i|\mu,\sigma^2) \overset{\text{IID}}{\sim} N(\mu,\sigma^2)$, it's a standard result from **frequentist distribution theory** that in repeated sampling

$$\frac{(n-1)s^2}{\sigma^2} \sim \chi^2_{n-1}, \tag{43}$$

where $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$ is **random** and σ^2 is **fixed**, from which

$$P_f\left[A \leq \frac{(n-1)s^2}{\sigma^2} \leq B\right] = 0.99 \tag{44}$$

for A, B **such that**

$$P_f\left(\chi^2_{n-1} \leq A\right) = P_f\left(\chi^2_{n-1} \geq B\right) = 0.005. \tag{45}$$

Thus, using **Neyman's confidence trick**,

$$P_f\left[\frac{(n-1)s^2}{B} \leq \sigma^2 \leq \frac{(n-1)s^2}{A}\right] = 0.99; \tag{46}$$

in other words, $\left[\frac{(n-1)s^2}{B}, \frac{(n-1)s^2}{A}\right]$ is a 99% **confidence interval** for σ^2.

With the **NB10 data** $n = 100$ and $s^2 = 41.82$, and you can use `R` to do this analysis:

```
> y
 [1] 409 400 406 399 402 406 401 403 401 403 398 403 407 402 401 399 400 401
[19] 405 402 408 399 399 402 399 397 407 401 399 401 403 400 410 401 407 423
[37] 406 406 402 405 405 409 399 402 407 406 413 409 404 402 404 406 407 405
[55] 411 410 410 410 401 402 404 405 392 407 406 404 403 408 404 407 412 406
[73] 409 400 408 404 401 404 408 406 408 406 401 412 393 437 418 415 404 401
[91] 401 407 412 375 409 406 398 406 403 404
```

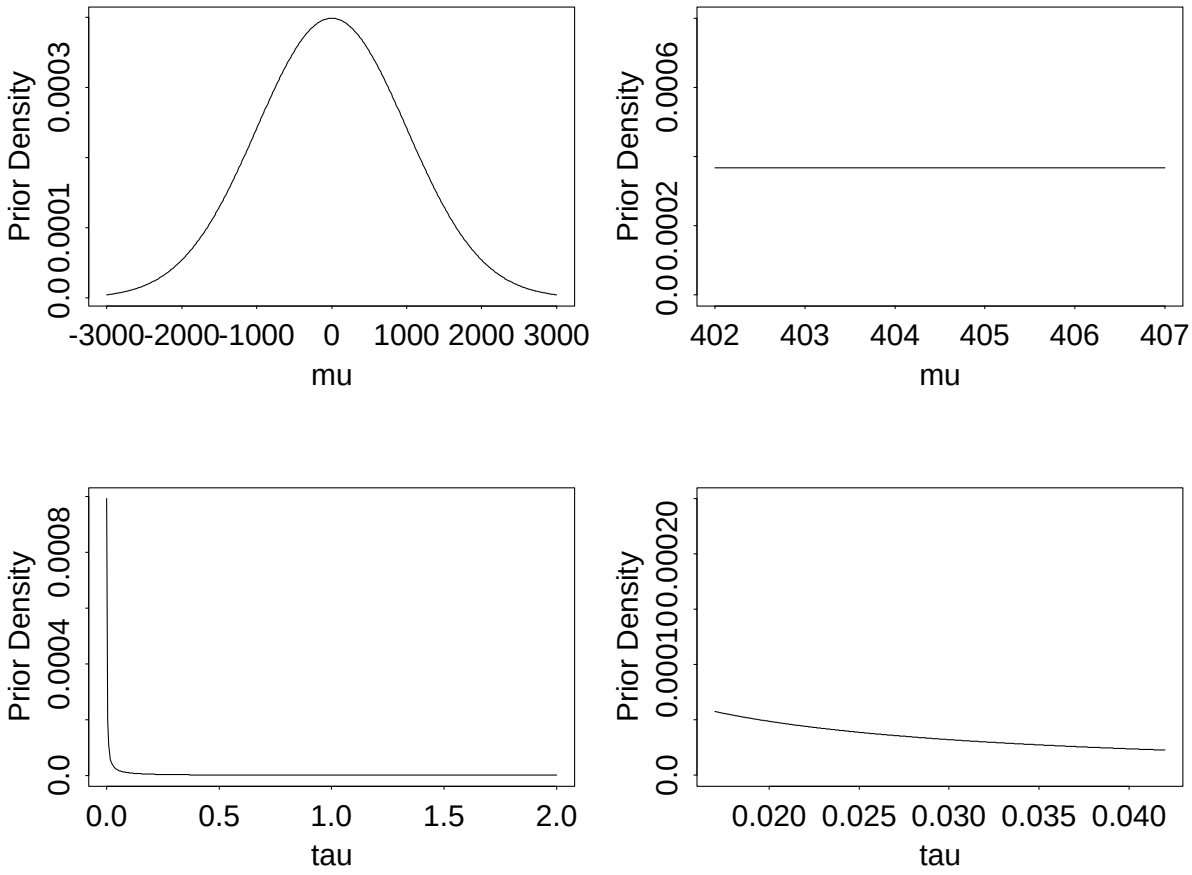
More Details

```
> print( n <- length( y ) )

[1] 100

> print( s2 <- var( y ) )

[1] 41.8201

> qchisq( 0.005, 99 )

[1] 66.5101

> qchisq( 0.995, 99 )

[1] 138.9868

> ( n - 1 ) * s2 / qchisq( 0.995, 99 )

[1] 29.78837

> ( n - 1 ) * s2 / qchisq( 0.005, 99 )

[1] 62.24904

> qchisq( 0.005, 99 ) / ( ( n - 1 ) * s2 )

[1] 0.01606451

> qchisq( 0.995, 99 ) / ( ( n - 1 ) * s2 )

[1] 0.03357015
```

So the conclusion is that the **likelihood** for $\tau = \frac{1}{\sigma^2}$ should be **non-negligible** roughly in the region from about 0.015 to 0.035.

The figure below plots the prior distributions for μ and τ and **verifies their diffuseness** in the relevant regions.

More Details



1. (c) As for the **prior** on ν, you can tell from the normal qqplot of the NB10 data that the degrees of freedom parameter in the underlying t distribution is **fairly small**.

I'm going to use a **uniform** $U(c_1, c_2)$ **prior**, where c_1 is small but not too small (as noted earlier, with $\nu < 2$ the variance is infinite, which is **not realistic** as a model for actual data) and c_2 is big enough not to **truncate** the likelihood function (experience tells me that $c_2 = 12$ will suffice; this can also be determined via MCMC **experimentation**).

Classic `BUGS` can't figure out how to sample from a continuous $U(c_1, c_2)$ prior on ν, however, so instead I've used a **discrete uniform prior** on a $g = 100$–point grid from 2.1 to 12.0 in steps of 0.1 (that's what `u ~ dcat( grid[ ] ); nu <- 2.0 + u / 10.0;` does when `grid[ ]` is a vector of 100 1s).

More Details

WinBUGS has a **more elegant solution** to this problem that we'll look at later.

(2) **Initial Values.** I can make fairly decent guesses at all the parameters as **starting values** for the Markov chain:

(a) The **sample mean** is 404.59, which should be close to the posterior mean for μ in the t model;

(b) I'm just going to **guess** that ν is around 5, which is specified by taking $u = 30$.

(c) Earlier I said that $V\left[t_\nu(\mu, \sigma^2)\right] = \sigma^2 \left(\frac{\nu}{\nu-2}\right)$, so with $\nu \doteq 5$ and a **sample variance** of 41.82 you get $\tau = \frac{1}{\sigma^2} \doteq 0.04$.

A Running Strategy. With a problem like this with **relatively few parameters**, I often start off with a burn-in of 1,000 and a monitoring run of 5,000 and then look at the **MCMC diagnostics** (to be covered below).

The left-hand part of the table at the top of page 53 shows the BUGS **commands** that carry out this run.

You can either type in these commands **interactively** one at a time at the keyboard or put them in a `.cmd` file and run BUGS in the **background** (this is useful when you're interested in **simulating** the Bayesian analysis of many similar datasets for research purposes; the latest release of WinBUGS now also has this capability).

This run took about **5 minutes** on a not particularly fast workstation (a SunBlade 150 running Solaris Unix at 600 Mhz), which is actually fairly slow for a 3-parameter problem (the **discrete grid sampling** for ν slows things down a lot).

Classic BUGS Run

```
rosalind 61> bugs

Welcome to BUGS on 20 th Feb 2003  at 16:38:29
BUGS : Copyright (c) 1992 .. 1995 MRC Biostatistics Unit.
All rights reserved.
Version 0.603 for unix systems.
For general release: please see documentation for disclaimer.
The support of the Economic and Social Research Council (UK)
is gratefully acknowledged.

Bugs>compile( "nb10-1.bug" )

model nb10;

  [here BUGS just echoes the model shown on page 53]

}

Parsing model declarations.
Loading data value file(s).
Loading initial value file(s).
Parsing model specification.
Checking model graph for directed cycles.
Generating code.
Generating sampling distributions.
Checking model specification.
Choosing update methods.
compilation took  00:00:00

Bugs> update( 1000 )

      time for    1000   updates was  00:00:47

Bugs>monitor( mu )

Bugs>monitor( sigma )

Bugs>monitor( nu )

Bugs>update( 5000 )

      time for    5000   updates was  00:03:56
Bugs>q( )       # (output file created; more about this later)
```

Practical MCMC monitoring and convergence diagnostics

Remember questions (3) and (4) awhile ago? — (3) **How large** should b and m be? (4) More generally, how do you know when the chain has reached **equilibrium**?

A **large body of research** has grown up just in the last eight years or so to answer these questions (some **good reviews** are available in Gelman et al. 2003, Gilks et al. 1995, and Cowles and Carlin 1996).

The theoretical bottom line is unpleasant: you **can't ever be sure you've reached equilibrium**, in the sense that every **MCMC diagnostic** invented so far has at least one example in which it failed to diagnose problems.

However, a collection of **four of the best diagnostics** has been brought together in a set of `R` functions called `CODA` by Best, Cowles, and Vines (1995) (downloadable from the `R` web site).

I will briefly discuss each of these in the context of the **NB10 analysis**.

Geweke (1992) proposed a simple diagnostic based on **time series** ideas.

Thinking of each column of the MCMC dataset as a **time series** (with iterations indexing time), he reasoned that, if the chain were in equilibrium, the **means** of the first (say) 10% and the last (say) 50% of the iterations should be **nearly equal**.

His diagnostic is a z-score for testing this equality, with a separate value for each quantity being monitored: Geweke z-scores **a lot bigger than 2 in absolute value** indicate that the mean level of the time series is **still drifting**, even after whatever burn-in you've already done.

MCMC Diagnostics (continued)

```
GEWEKE CONVERGENCE DIAGNOSTIC (Z-score):
========================================

Iterations used = 1002:6001     Fraction in
Thinning interval = 1             1st window = 0.1
Sample size per chain = 5000    Fraction in
                                  2nd window = 0.5
-+----------+-------------+-
 | VARIABLE |   bugs1     |
 | ======== |   =====     |
 |          |             |
 | mu       |   2.39      |
 | nu       |   1.78      |
 | sigma    |   1.14      |
 |          |             |
-+----------+-------------+-
```

Here for run 1 with the NB10 data (the left-hand set of commands in the table on p. 53) there's **some evidence of nonstationarity** with a burn-in of only 1,000 (although a z-value of 2.4 is **not overwhelming**).

Gelman-Rubin (1992) have suggested a diagnostic that looks for **multimodality** of the posterior distribution.

If the posterior has (say) two major modes that are far away from each other in parameter space, and you initialize the chain near one of the modes,
you may never find the other one.

The idea is to run the chain two or more times from widely-dispersed starting points and **see if you always converge to the same place**.

Gelman and Rubin do what amounts to an **analysis of variance** within and between the chains, looking for evidence of **large variability between them**.

Gelman-Rubin Shrink Factors

"This comparison is used to estimate the factor by which the scale parameter of the marginal posterior distribution of each [quantity being monitored] might [**shrink**] if the chain were run to infinity" (Best et al., 1995).

The output is the 50% and 97.5% quantiles of the distributions of **shrink factors**, one for each quantity monitored.

If these quantiles are both close to 1.0 then there's **little evidence of dispersion** between the distributions to which the chains are converging.

```
GELMAN AND RUBIN 50% AND 97.5% SHRINK FACTORS:
==============================================

Iterations used for diagnostic  = 2501:5000
Thinning interval = 1
Sample size per chain = 5000

-+----------+-----------------------------+-
 | VARIABLE |  Point est. 97.5% quantile  |
 | ======== |  ========== ==============  |
 |          |                             |
 | mu       |  1.00       1.00            |
 | nu       |  1.00       1.01            |
 | sigma    |  1.00       1.00            |
 |          |                             |
-+----------+-----------------------------+-
```

Here, with initial values as different as $(\mu, \tau, \nu) = (405.0, 0.1823, 5.0)$ and $(402.0, 0.03, 11.0)$ there's **no evidence of multimodality** at all.

Raftery-Lewis Dependence Factors

(To be really safe I should run a **number of additional chains** — Gelman and Rubin (1992) give advice on how to generate the set of initial values to try — but with even modest sample sizes (like $n = 100$) the posterior in t models is **unimodal** so there would be no point in this case.)

Raftery-Lewis (1992) suggested a **diagnostic** that directly helps to answer question (3) — How do you pick b and m?

The answer to this question depends on how **accurate** you want your posterior summaries to be, so Raftery and Lewis require you to **input** three values:

(a) **Which quantiles** of the marginal posteriors are you most interested in?

Usually the answer is the **2.5%** and **97.5%** points, since they're the basis of a 95% interval estimate.

(b) How close to the **nominal levels** would you like the **estimated quantiles** to be?

The CODA default is **0.5%**, e.g., if the left-hand value of your 95% interval is supposed to be at the **2.5%** point of the distribution, CODA will recommend a length of monitoring run so that the actual level of this quantile will be between **2.0%** and **3.0%**.

(**NB** This is sometimes more, and often less, Monte Carlo accuracy than you really need.)

(c) With what minimum **probability** do you want to achieve these accuracy goals? The default is **95%**.

Having input these values, the output is of **five kinds** for each quantity monitored:

(a) A recommended **thinning interval**. When the Gibbs sampler is performing poorly people say the output is not **mixing well**, and what they mean is that the Markovian nature of the time series for each quantity has led to **large positive serial autocorrelations** in time, e.g., μ_{1000} depends highly on μ_{999}, μ_{998}, and so on.

Raftery-Lewis (continued)

This is another way to say that the **random draws** in the simulation process are **not moving around the parameter space quickly**.

When this happens, one way to reduce the autocorrelation is to **run the chain a lot longer and only record every kth iteration** — this is the **thinning interval**.

(b) A recommended length of **burn-in** to use, above and beyond whatever you've already done.

(c) A recommended **total length of run** N (including burn-in) to achieve the desired accuracy.

(d) A **lower bound** N_{min} on run length — what the minimum would have needed to be if the quantity in question had an **IID** time series instead of an autocorrelated series.

(e) And finally, the ratio $I = N/N_{min}$, which Raftery and Lewis call the **dependence factor** — values of I near 1 indicate good mixing.

```
RAFTERY AND LEWIS CONVERGENCE DIAGNOSTIC:
=========================================

Iterations used = 1001:6000
Thinning interval = 1
Sample size per chain = 5000

Quantile = 0.025
Accuracy = +/- 0.005
Probability = 0.95
```

VARIABLE	Thin (k)	Burn-in (M)	Total (N)	Lower bound (Nmin)	Dependence factor (I)
mu	1	3	4533	3746	1.21
nu	3	18	39720	3746	10.6
sigma	3	12	13308	3746	3.55

Heidelberger-Welch Diagnostic

Here μ is mixing well — 5,000 iterations are sufficient to achieve the default accuracy goal — but σ **and (especially)** ν **require longer monitoring periods**: the recommendation is to run for about 40,000 iterations and store every third.

Heidelberger-Welch (1983) propose a diagnostic approach that uses the **Cramér-von Mises statistic** to test for stationarity.

If **overall stationarity** fails for a given quantity being monitored, CODA **discards** the first 10% of the series for that quantity and recomputes the **C-vonM** statistic, continuing in this manner until only the final 50% of the data remain.

If stationarity still fails with the last half of the data then CODA reports **overall failure** of the stationarity test.

CODA also computes a **half-width** test, which tries to judge whether the portion of the series that passed the stationarity test is sufficient to estimate the posterior mean with a particular default accuracy (**NB** this default is often not stringent enough for careful numerical work).

Here the table below shows that the first run with the NB10 data **clears the Heidelberger-Welch hurdle** with ease.

Autocorrelations and Cross-correlations. CODA also computes the **autocorrelations** for each monitored quantity at lags from 1 to 50 and the **cross-correlations** between all of the variables.

As mentioned previously, the **autocorrelation** at **lag** k of a time series $\{\theta_t^*, t = 1, \ldots, m\}$ (e.g., Chatfield 1996) measures the extent to which the series at time $(t + k)$ and at time t are **linearly related**, for $k = 1, 2, \ldots$.

MCMC Diagnostics (continued)

```
HEIDELBERGER AND WELCH STATIONARITY AND INTERVAL HALFWIDTH TESTS:
================================================================

Precision of halfwidth test = 0.1

-+----------+-------------------------------------------------------+-
 |          |  Stationarity   # of iters.  # of iters.  C-vonM   |
 | VARIABLE |      test         to keep    to discard    stat.   |
 | ======== |  ============   ==========  ==========  ======   |
 |          |                                                       |
 | mu       |  passed         5000        0           0.126    |
 | nu       |  passed         5000        0           0.349    |
 | sigma    |  passed         5000        0           0.176    |
 |          |                                                       |
-+----------+-------------------------------------------------------+-
 |          |  Halfwidth                          |
 | VARIABLE |    test      Mean    Halfwidth      |
 | ======== |  =========   ====    =========      |
 |          |                                     |
 | mu       |  passed      404.00  0.0160         |
 | nu       |  passed        3.75  0.1500         |
 | sigma    |  passed        3.89  0.0344         |
 |          |                                     |
-+----------+-------------------------------------+-
```

The usual **sample estimate** of this quantity is

$$r_k = \frac{c_k}{c_0}, \text{ where } c_k = \frac{1}{m-k}\sum_{t=1}^{m-k}\left(\theta_t^* - \bar{\theta^*}\right)\left(\theta_{t+k}^* - \bar{\theta^*}\right) \qquad (47)$$

$$\text{and } \bar{\theta^*} = \tfrac{1}{m}\textstyle\sum_{t=1}^{m}\theta_t^*.$$

The **cross-correlation** at **lag** k of two time series $\{\theta_t^*, t = 1, \ldots, m\}$ and $\{\eta_t^*, t = 1, \ldots, m\}$ measures the extent to which the first series at time $(t + k)$ and the second at time t are **linearly related**, for $k = 1, 2, \ldots$.

A **natural sample estimate** of this quantity is

$$r_{\theta\eta}(k) = \frac{c_{\theta\eta}(k)}{\sqrt{c_{\theta\theta}(0)c_{\eta\eta}(0)}}, \quad \text{where}$$

$$c_{\theta\eta}(k) = \frac{1}{m-k}\sum_{t=1}^{m-k}\left(\theta_t^* - \bar{\theta^*}\right)\left(\eta_{t+k}^* - \bar{\eta^*}\right). \qquad (48)$$

MCMC Diagnostics (continued)

LAGS AND AUTOCORRELATIONS WITHIN EACH CHAIN:

Chain	Variable	Lag 1	Lag 10	Lag 50
bugs1	mu	0.29400	0.00118	-0.01010
	nu	0.97200	0.78900	0.32100
	sigma	0.62100	0.30300	0.10800

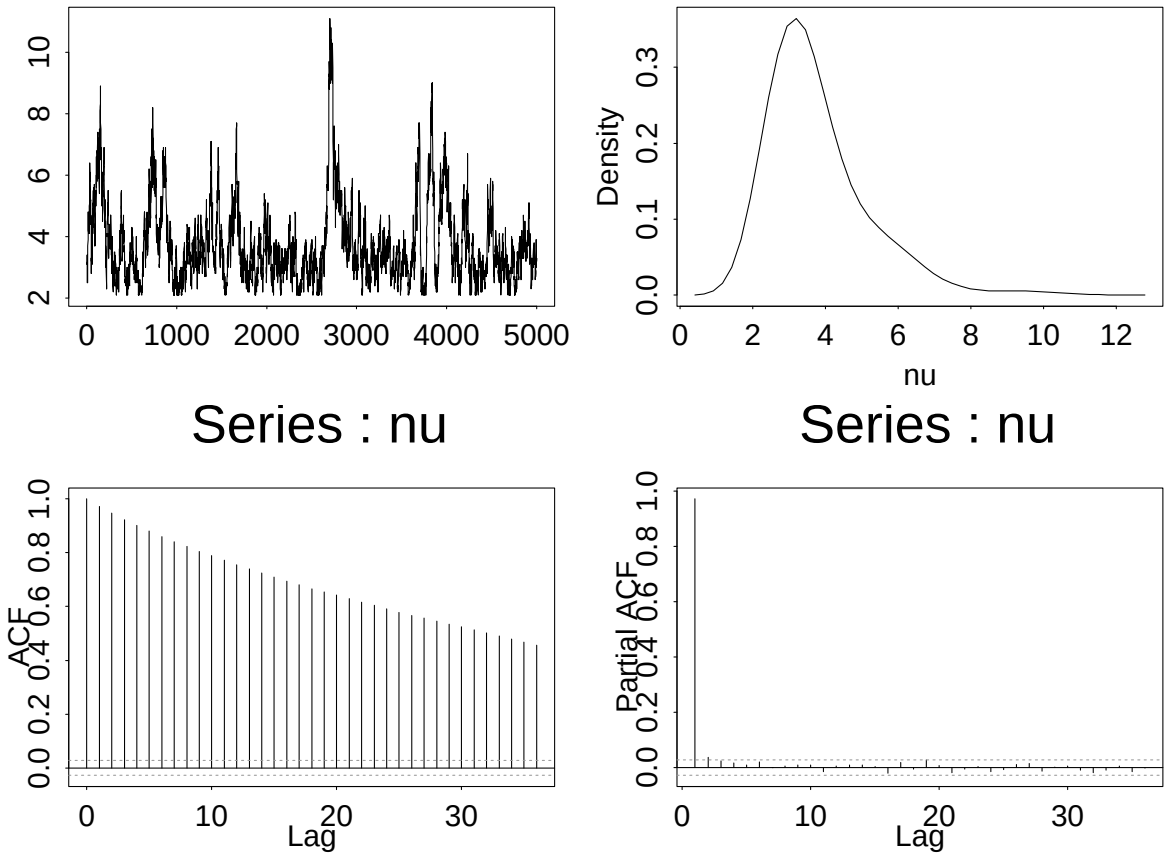
CROSS-CORRELATION MATRIX:

VARIABLE	mu	nu	sigma
mu	1.0000		
nu	0.0946	1.0000	
sigma	0.0534	0.5540	1.0000

You can see (a) that the series for ν is **especially strongly autocorrelated**, and (b) that ν and σ are **fairly strongly positively correlated**, which connects with the observation earlier about confounding of scale and shape in the t family.

Diagnostic and Summary Plots. The figure below presents four plots that are useful as **MCMC diagnostics** and for **graphical summaries of posterior distributions**, in the case of the parameter ν with run 1 from the NB10 data.

Diagnostic and Summary Plots



The upper left panel is a **time series trace**, which documents the poor mixing that has been evident from several of the numerical diagnostics.

The lower left panel is a plot of the **autocorrelation** function (ACF) for ν, and the lower right panel plots the **partial autocorrelation** function (PACF).

One of the most common behaviors observed in time series in general, and in the output of MCMC samplers in particular, is that of an **autoregressive process**.

Letting e_t denote an IID (or **white-noise** or **purely random**) process with mean 0 and variance σ_e^2, the time series θ_t^* is said to be an **autoregressive process of order** p (AR_p) if

$$\theta_t^* = \alpha_1 \theta_{t-1}^* + \ldots + \alpha_p \theta_{t-p}^* + e_t. \tag{49}$$

Diagnostic and Summary Plots

Equation (49) is like a **multiple regression model** except that θ_t^* is being regressed on **past values of itself** instead of on other predictor variables; this gives rise to the term **autoregressive**.

The **partial autocorrelation function** (PACF) measures the excess correlation between θ_t^* and θ_{t+k}^* not accounted for by the autocorrelations $r_1, \ldots, r_{k-1}$, and is useful in **diagnosing the order** of an AR_p process: if θ_t^* is AR_p then the PACF at lags $1, \ldots, p$ will be significantly different from 0 and then close to 0 at lags larger than p.

The lower right-hand plot above shows the characteristic **single spike at lag 1** that diagnoses an AR_1 series (the dotted lines in the ACF and PACF plots represent **2 standard error traces around 0**, indicating how big an ACF or PACF value needs to be to be significantly different from 0).

This is reinforced by the ACF plot: if θ_t^* is AR_1 with positive first-order autocorrelation ρ_1 then the autocorrelation function should show a **slow geometric decay** (a **ski-slope** shape), which it clearly does in this case.

We would conclude that the Gibbs sampling output for ν, when thought of as a **time series**, behaves like an AR_1 process with **first-order autocorrelation** roughly $r_1 = 0.972$ (from the table above).

MCMC Accuracy. Suppose that θ_t^* is a **stationary time series** with underlying true mean μ_θ and variance σ_θ^2.

MCMC Accuracy

It can be shown that if $\{\theta_t^*, t = 1, \ldots, m\}$ is AR_1 with first-order autocorrelation ρ_1 then **in repeated sampling** the **uncertainty about** μ_θ on the basis of the sample mean $\bar{\theta}^*$ is quantified by

$$V(\bar{\theta}^*) = \frac{\sigma_\theta^2}{m}\left(\frac{1+\rho_1}{1-\rho_1}\right). \tag{50}$$

Thus if you want to use MCMC to estimate the posterior mean of a given quantity θ with **sufficient accuracy** that the standard error of the Monte Carlo mean estimate $\bar{\theta}^*$ based on a monitoring run of length m is **no larger than a specified tolerance** T, and the MCMC output θ^* behaves like an AR_1 series with first-order autocorrelation ρ_1, you would need m to satisfy

$$\widehat{SE}(\bar{\theta}^*) = \frac{\widehat{\sigma}_\theta}{\sqrt{m}}\sqrt{\frac{1+\widehat{\rho}_1}{1-\widehat{\rho}_1}} \leq T, \tag{51}$$

from which

$$m \geq \frac{\widehat{\sigma}_\theta^2}{T^2}\left(\frac{1+\widehat{\rho}_1}{1-\widehat{\rho}_1}\right). \tag{52}$$

This formula explains why monitoring runs with MCMC often need to be **quite long**: as $\rho_1 \to 1$ the required $m \to \infty$.

For example, we have seen that $\widehat{\rho}_1 = r_1$ for ν in the NB10 t model is +0.972, and we will see below that the **sample mean and SD** based on the output for ν are roughly 3.628 and 1.161, respectively.

If you wanted to be able to report the posterior mean of ν to **3–significant-figure accuracy** (3.63) with reasonably high Monte Carlo probability, you would want T to be **on the order of 0.01**, giving an enormous monitoring run:

Diagnostic Plots (continued)

$$m \geq \left(\frac{1.161}{0.01}\right)^2 \left(\frac{1+0.972}{1-0.972}\right) \doteq (13,479)(70.4) \doteq 949,322 \tag{53}$$

This is much larger than the **Raftery-Lewis default recommendation** above (there's no conflict in this fact; the two diagnostics are focusing on **different posterior summaries**).

Note from (52) that if you could figure out how to sample **in an IID manner** from the posterior for θ you would only need $m_{\text{IID}} \geq \frac{\widehat{\sigma}_\theta^2}{T^2}$, which in this case is about 13,500 draws.

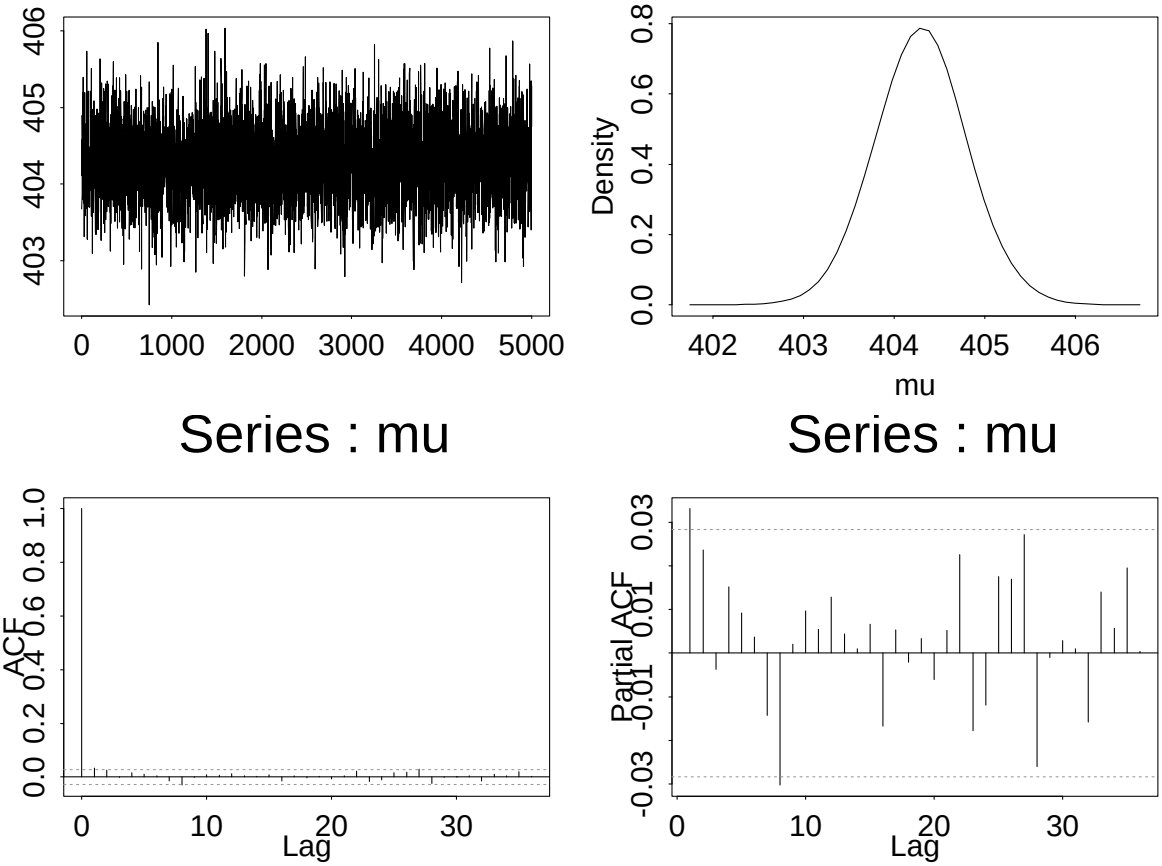
The term $\left(\frac{1+\widehat{\rho}_1}{1-\widehat{\rho}_1}\right)$ in (52) represents the amount by which m_{IID} would need to be multiplied to get the same accuracy from MCMC output — it's natural to call this the **sample size inflation factor** (SSIF), which for ν comes out a whopping 70.4.

The upper right panel in the diagnostic plots above gives a **density trace** for ν, which shows a mode at about 3 degrees of freedom and a long right-hand tail.

Round 2. From all of this I decided to run the chain again with the BUGS commands in the right-hand part of the table on page 54: a **burn-in** of 2,000 and a **monitoring run** of 40,000, thinning the output by writing out to disk only every **8th draw** (thus ending up with 5,000 stored values).

The MCMC diagnostics were **much better**: Raftery-Lewis total N recommendations all less than 5,000, all other summaries fine.

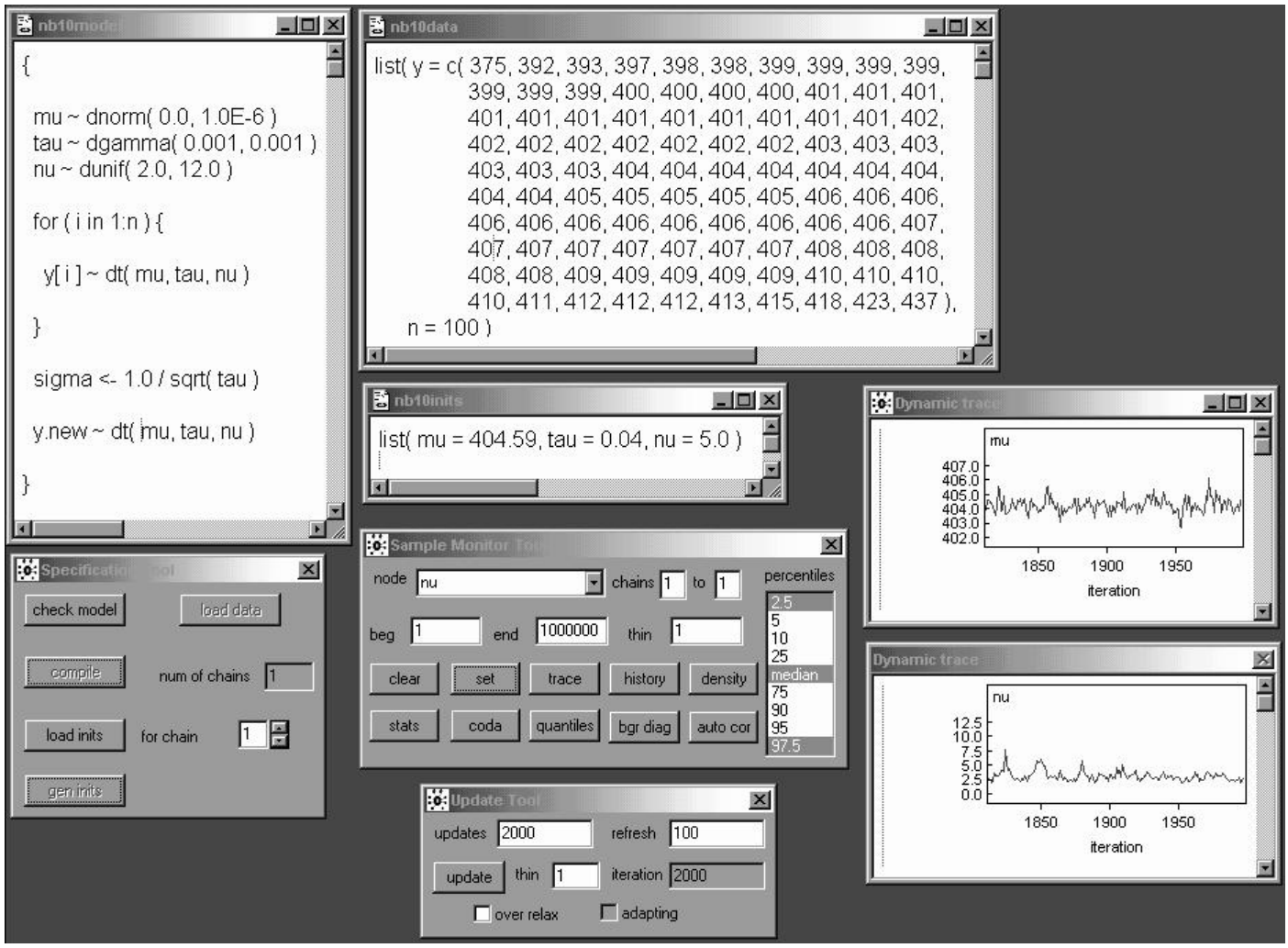
Numerical Summaries



All the parameters are **mixing well now**, so numerical posterior summaries are worth making, as in the table below.

Parameter	Posterior Mean	Posterior SD	95% Interval
μ	404.3	0.4641	(403.4, 405.2)
ν	3.63	1.16	(2.2, 6.6)
σ	3.873	0.4341	(3.100, 4.778)

WinBUGS Implementation

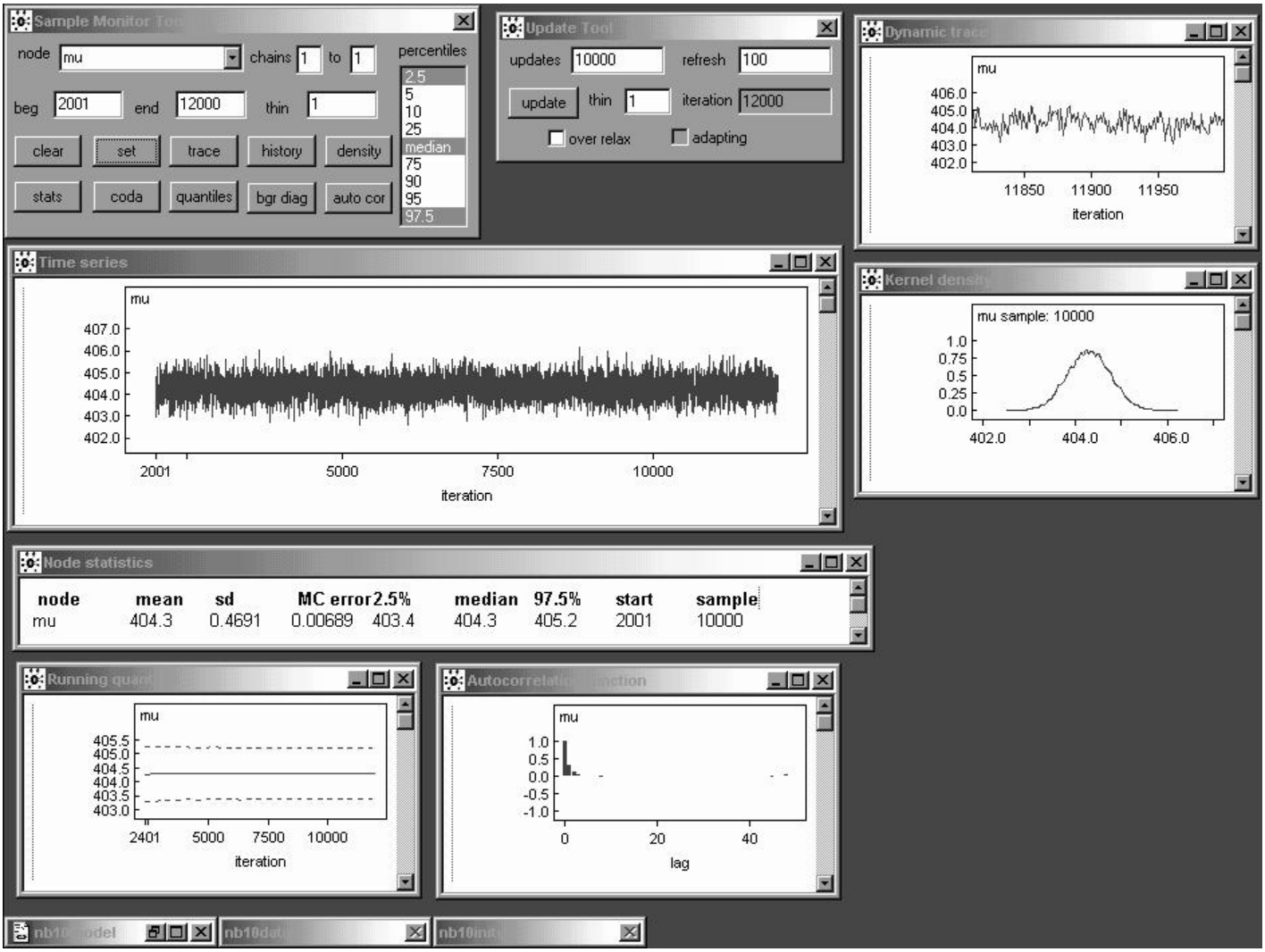


I read in three files — the **model**, the **data**, and the **initial values** — and used the `Specification Tool` from the `Model` menu to `check` the model, `load` the data, `compile` the model, `load` the initial values, and `generate` additional initial values for uninitialized nodes in the graph.

I then used the `Sample Monitor Tool` from the `Inference` menu to `set` the `mu`, `sigma`, `nu`, and `y.new` nodes, and clicked on `Dynamic Trace` **plots** for `mu` and `nu`.

Then choosing the `Update Tool` from the `Model` menu, specifying 2000 in the `updates` box, and clicking `update` permitted a **burn-in** of 2,000 iterations to occur with the **time series traces** of the two parameters displayed in **real time**.

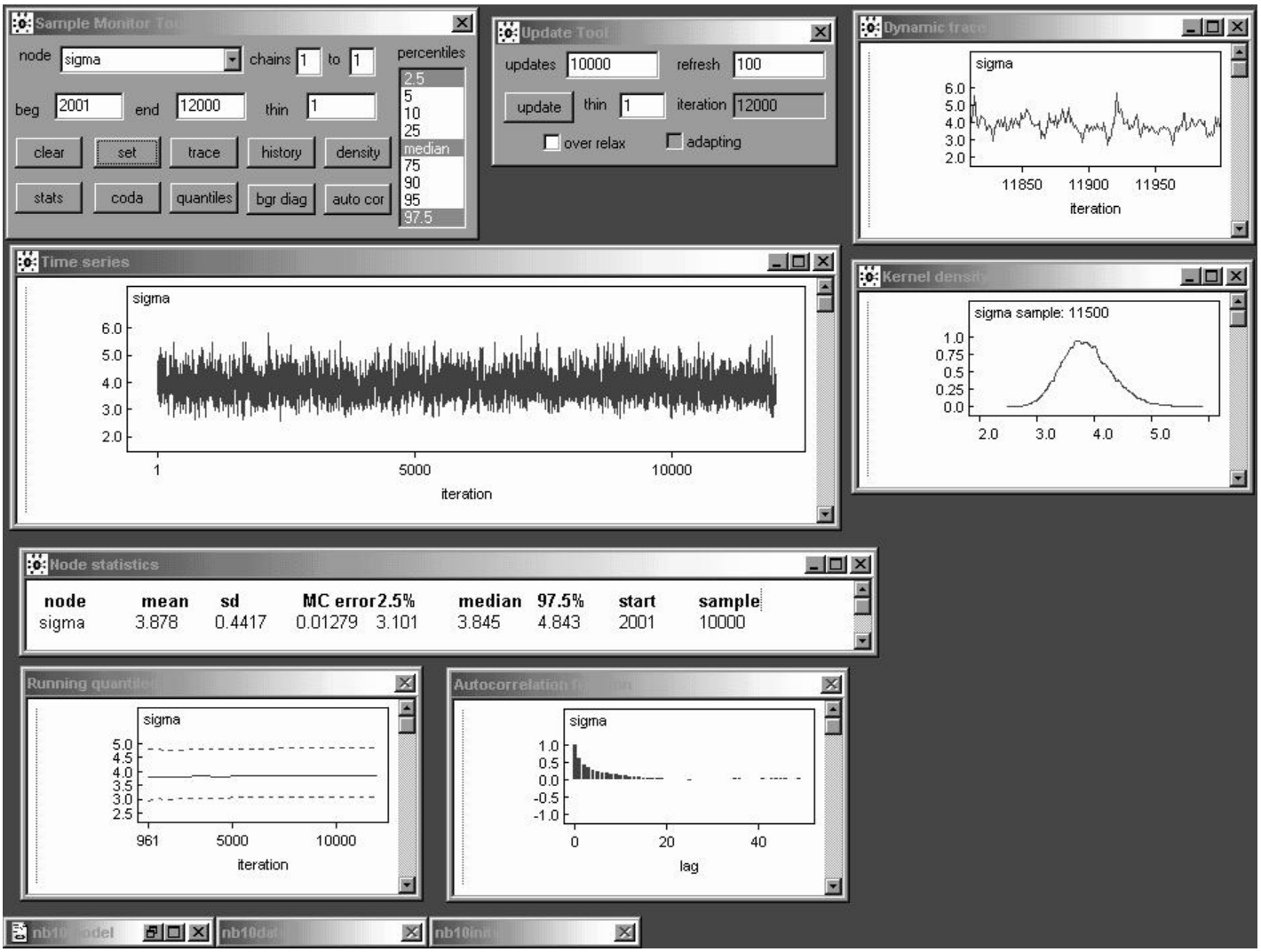
WinBUGS Implementation (continued)



After **minimizing** the `model`, `data`, and `inits` windows and **killing** the `Specification Tool` (which are no longer needed until the model is respecified), I typed 10000 in the `updates` box of the `Update Tool` and clicked `update` to generate a **monitoring run** of 10,000 iterations (you can watch the updating of `mu` and `nu` dynamically to get an idea of the **mixing**, but this slows down the sampling).

After **killing** the `Dynamic Trace` window for `nu` (to concentrate on `mu` for now), in the `Sample Monitor Tool` I selected `mu` from the pull-down menu, set the `beg` and `end` boxes to 2001 and 12000, respectively (to summarize only the **monitoring** part of the run), and clicked on `history` to get the **time series trace** of the monitoring run, `density` to get a **kernel density trace** of the 10,000 iterations, `stats` to get **numerical summaries** of the monitored iterations, `quantiles` to get a trace of the **cumulative estimates** of the 2.5%, 50% and 97.5% points in the estimated posterior, and `autoC` to get the **autocorrelation function**.

WinBUGS Implementation (continued)

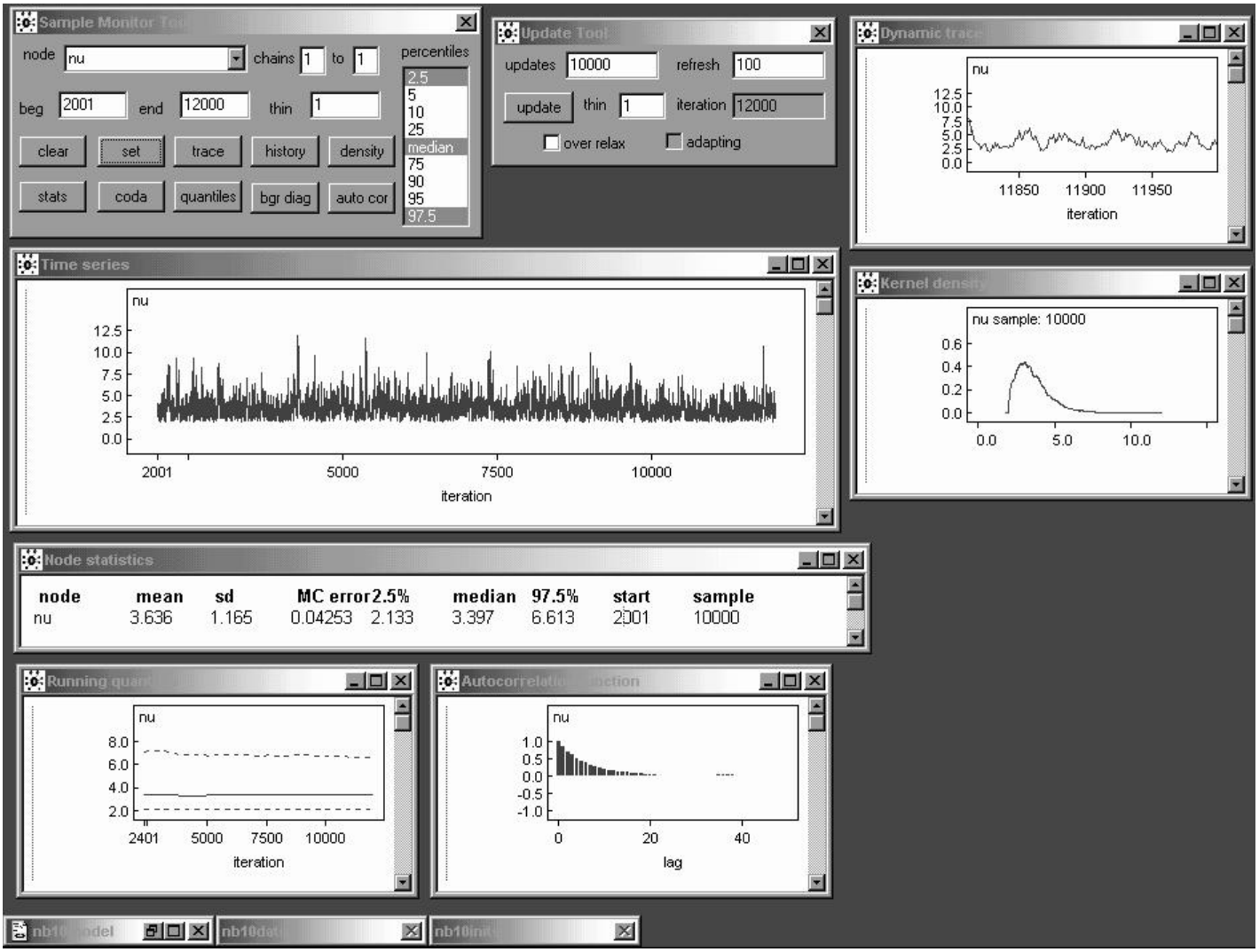


You can see that the output for μ is **mixing fairly well** — the ACF looks like that of an AR_1 series with first-order **serial correlation** of only about **0.3**.

σ is mixing less well: its ACF looks like that of an AR_1 series with first-order **serial correlation** of about **0.6**.

This means that a monitoring run of 10,000 would probably **not be enough** to satisfy **minimal Monte Carlo accuracy goals** — for example, from the `Node statistics` window the estimated posterior mean is **3.878** with an estimated MC error of **0.0128**, meaning that we've not yet achieved **three-significant-figure accuracy** in this posterior summary.

WinBUGS Implementation (continued)



And ν'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** $\widehat{\rho}_1$ 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

$$\widehat{SE}\left(\bar{\theta}^*\right) = \frac{\widehat{\sigma}_\theta}{\sqrt{m}} \sqrt{\frac{1+\widehat{\rho}_1}{1-\widehat{\rho}_1}}, \tag{54}$$

you can **solve** this equation for $\widehat{\rho}_1$ to get

$$\widehat{\rho}_1 = \frac{m\left[\widehat{SE}\left(\bar{\theta}^*\right)\right]^2 - \widehat{\sigma}_\theta^2}{m\left[\widehat{SE}\left(\bar{\theta}^*\right)\right]^2 + \widehat{\sigma}_\theta^2}. \tag{55}$$

WinBUGS Implementation (continued)

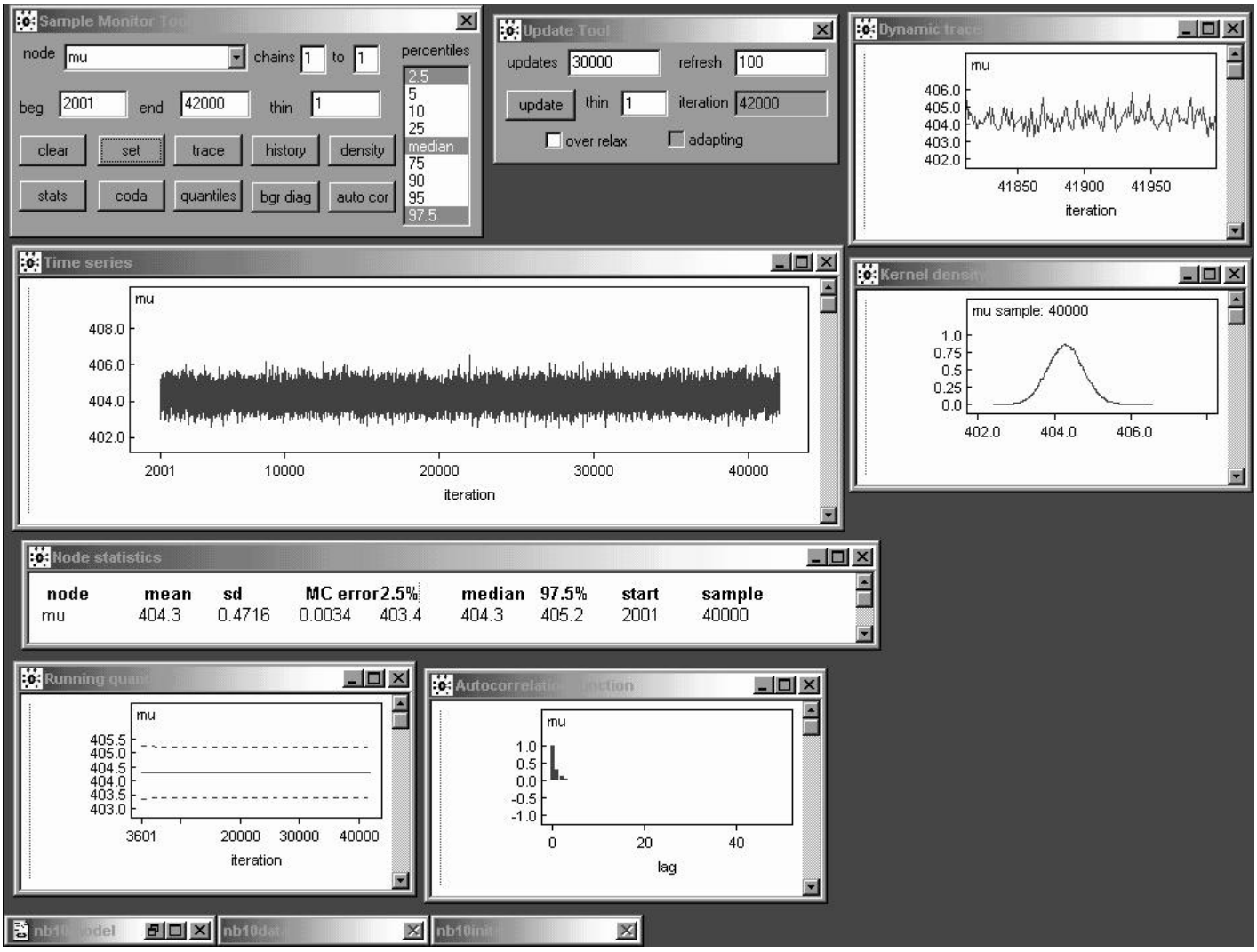
Plugging in the relevant values here gives

$$\widehat{\rho}_1 = \frac{(10,000)(0.04253)^2 - (1.165)^2}{(10,000)(0.04253)^2 + (1.165)^2} \doteq 0.860, \qquad (56)$$

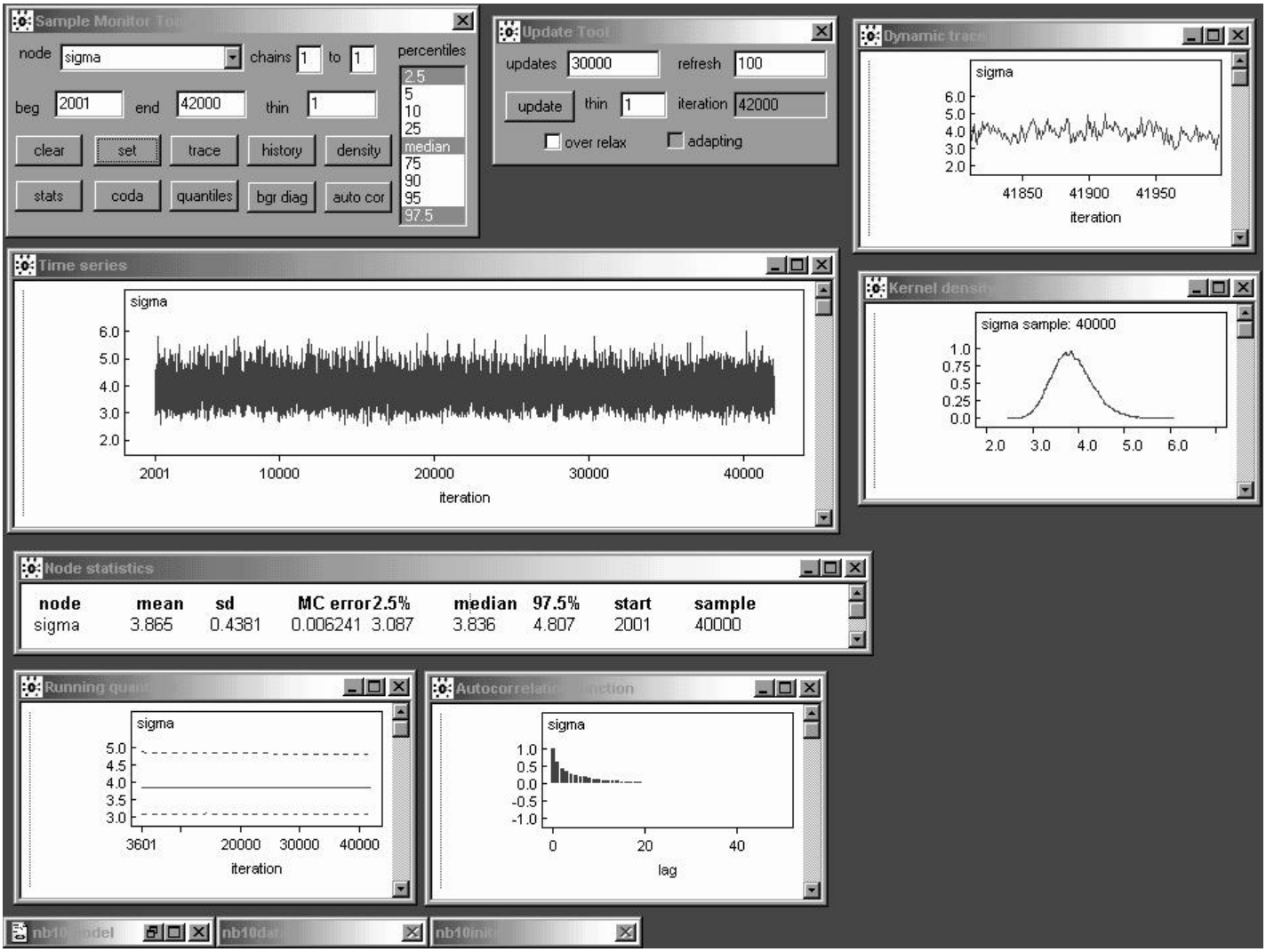
which is smaller than the corresponding value of **0.972** generated by the `classicBUGS` sampling method (from `CODA`, page 66).

To match the `classicBUGS` strategy outlined above (page 70) 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 μ are **satisfactory in every way**.

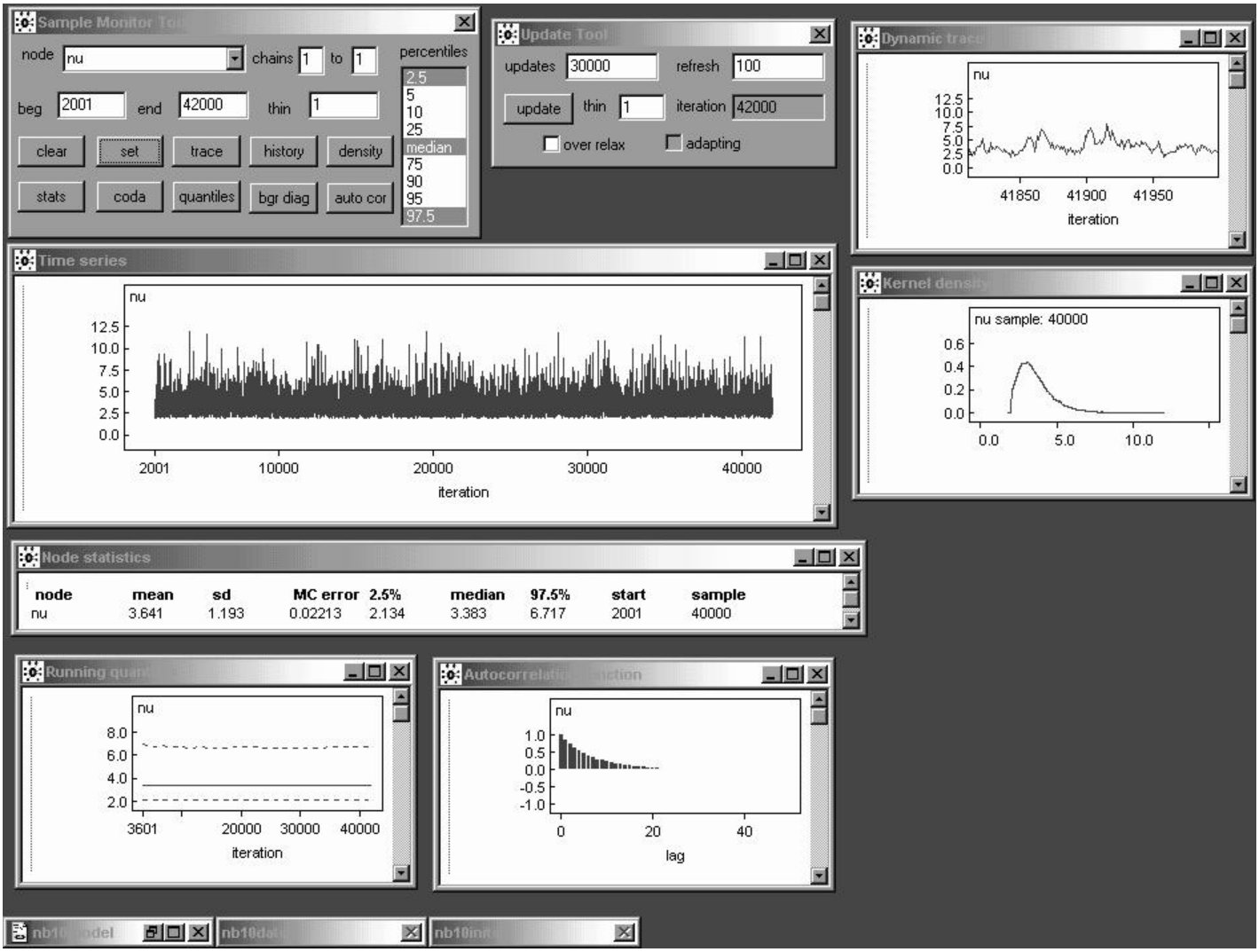


WinBUGS **Implementation (continued)**



A monitoring run of **40,000** also looks good for σ: on this basis, and **conditional on this model and prior**, I think σ 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 σ runs from about **3.09** to about **4.81** (the distribution has a bit of **skewness** to the right, which makes sense given that σ is a **scale parameter**).

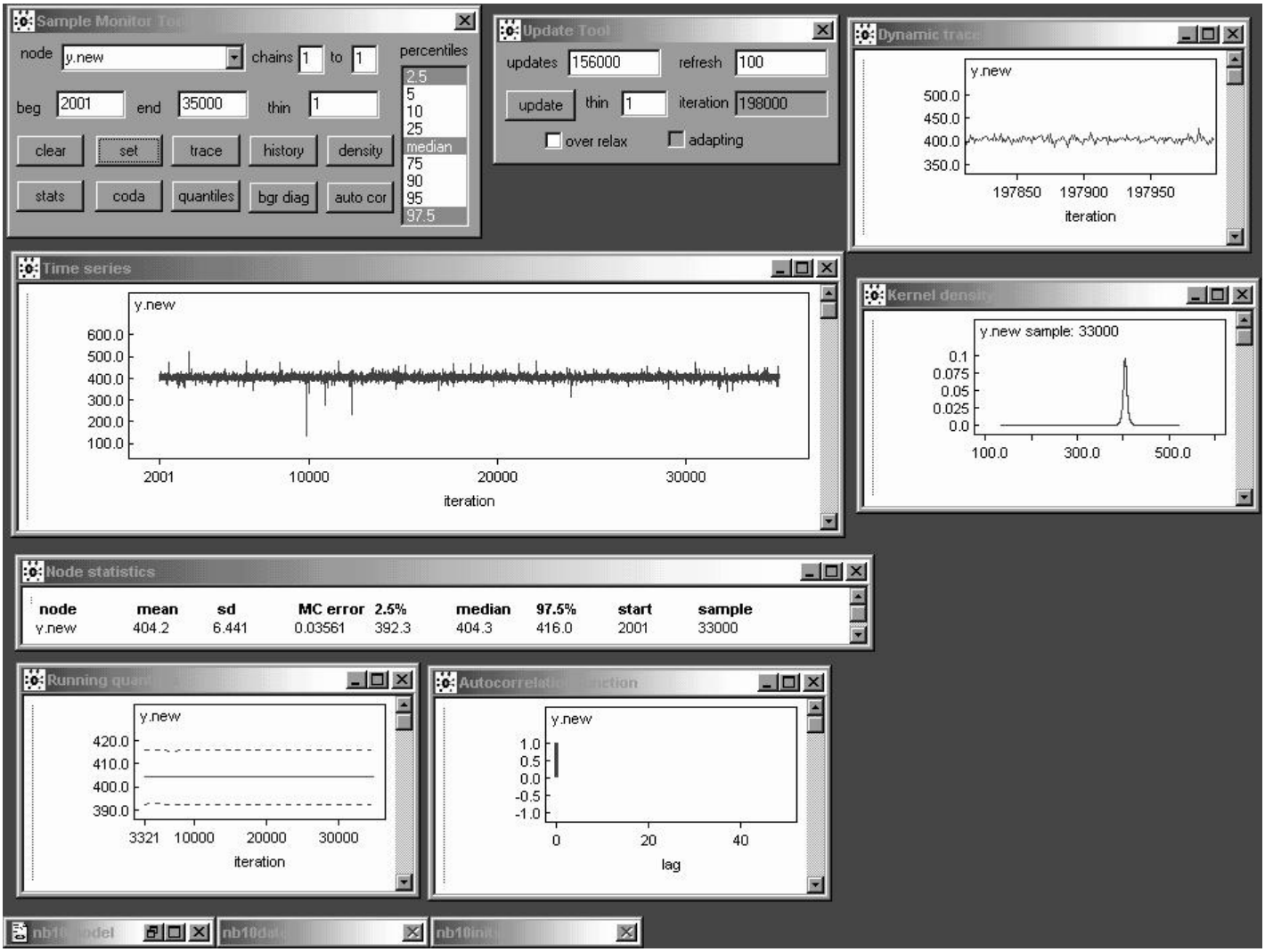
WinBUGS Implementation (continued)



If the **real goal** were ν I would use a **longer monitoring run**, but the main point here is μ, and we saw back on p. 66 that μ and ν are **close to uncorrelated in the posterior**, so this is good enough.

If you wanted to report the **posterior mean** of ν 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 $\left(\frac{0.02213}{0.01}\right)^2 \doteq 4.9$, 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**).

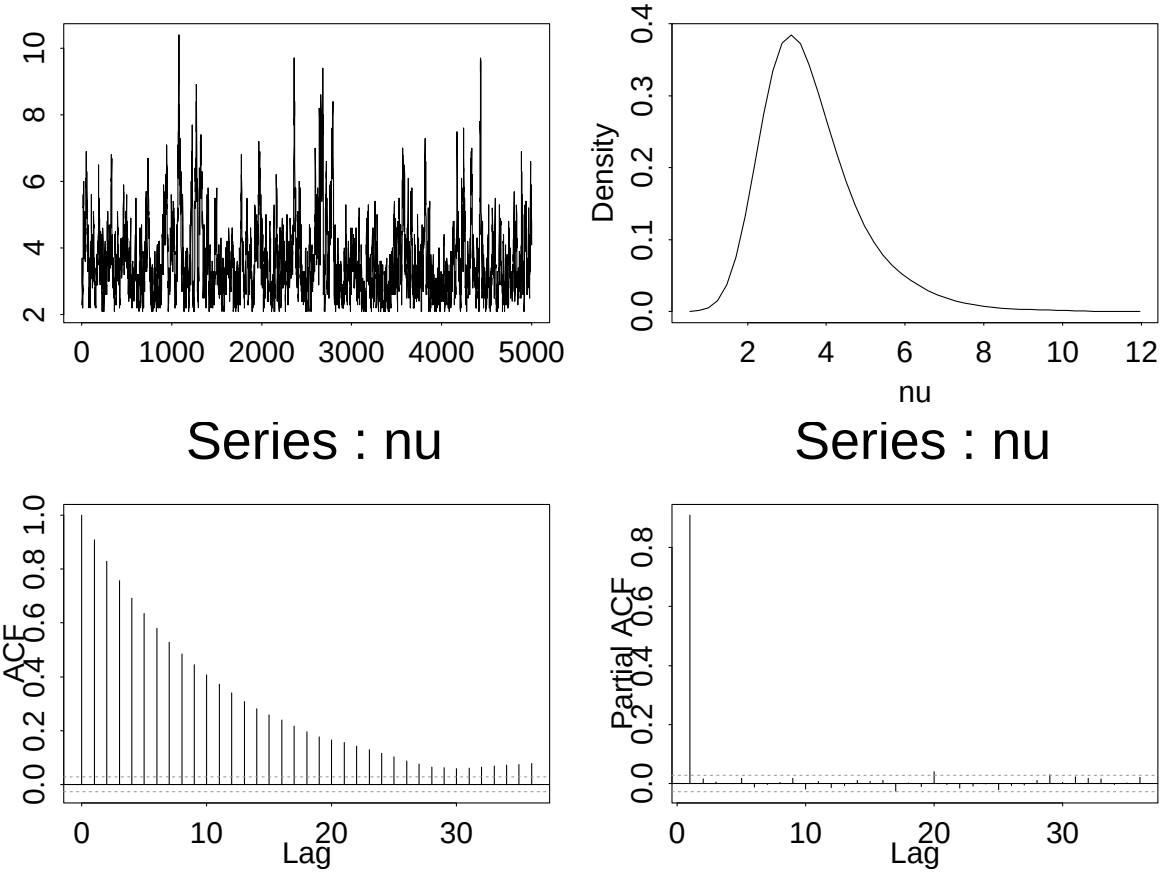
WinBUGS Implementation (continued)



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 ν 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

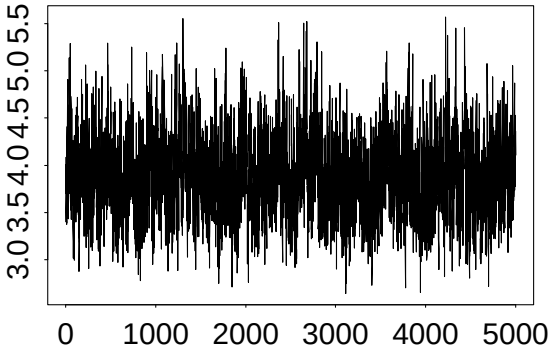
The posterior SD for μ, the only parameter directly comparable across the Gaussian and t models for the NB10 data, came out **0.47** from the t modeling, versus 0.65 with the Gaussian, i.e., the interval estimate for μ from the (incorrect) Gaussian model is about **40% wider** that that from the (much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

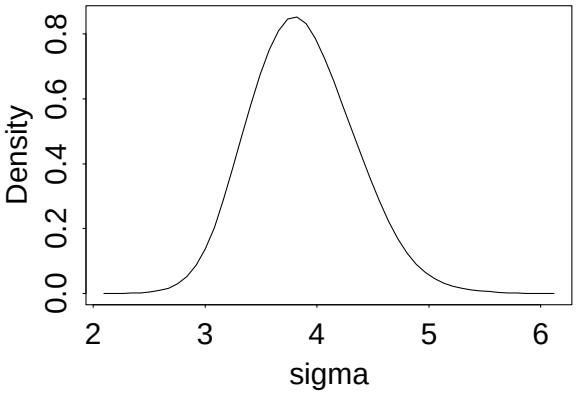


A Model Uncertainty Anomaly?

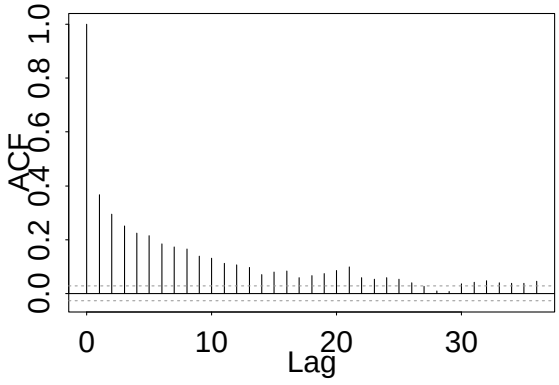
NB Moving from the Gaussian to the t model involves a net increase in **model uncertainty**, because when you assume the Gaussian you're in effect saying that you know the t degrees of freedom are ∞, whereas with the t model you're treating ν as unknown. And yet, even though there's been an increase in model uncertainty, the inferential uncertainty about μ 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.

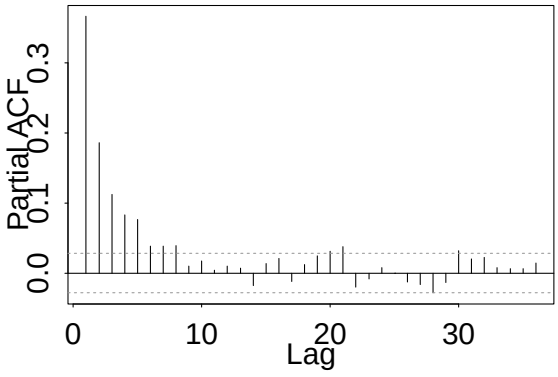




Series : sigma



Series : sigma



CODA in R

If you go to `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/`, 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

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

If everything goes smoothly this will **automatically install** R-CODA on your machine.

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

```
library( coda )
```

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

```
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** described above are available to you with this approach.

References

Best NG, Cowles MK, Vines SK (1995). *CODA Manual version 0.30*. MRC Biostatistics Unit, Cambridge, UK.

Cowles MK, Carlin BP (1996). Markov chain Monte Carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, **91**, 883–904.

Draper D (2004). On the relationship between model uncertainty and inferential/predictive uncertainty. Under revision.

Gelfand AE, Smith AFM (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American Statistical Association*, **85**, 398–409.

Gelman A, Carlin JB, Stern HS, Rubin DB (2003). *Bayesian Data Analysis*, second edition. London: Chapman & Hall.

Gelman A, Rubin DB (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, **7**, 457–472.

Geweke J (1992). Evaluating the accuracy of sampling-based approaches to calculating posterior moments. In *Bayesian Statistics 4*, JM Bernardo, JO Berger, AP Dawid, AFM Smith (eds.). Oxford: Clarendon Press.

Gilks WR, Wild P (1992). Adaptive rejection sampling for Gibbs sampling. *Applied Statistics*, **41**, 337–348.

Gilks WR, Clayton DG, Spiegelhalter DJ, Best NG, McNeil AJ, Sharples LD, Kirby AJ (1993). Modeling complexity: Applications of Gibbs sampling in medicine. *Journal of the Royal Statistical Society, Series B*, **55**, 39–52.

Gilks WR, Richardson S, Spiegelhalter DJ (eds.) (1995). *Markov Chain Monte Carlo in Practice*. London: Chapman & Hall.

Heidelberger P, Welch P (1983). Simulation run length control in the presence of an initial transient. *Operations Research*, **31**, 1109–1144.

Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.

Raftery AL, Lewis S (1992). How many iterations in the Gibbs sampler? In *Bayesian Statistics 4*, JM Bernardo, JO Berger, AP Dawid, AFM Smith (eds.). Oxford: Clarendon Press, 763–774.