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):

<table>
<thead>
<tr>
<th>Iteration</th>
<th>( \theta )</th>
<th>( I(\theta \leq 0.15) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1 )</td>
<td>( \theta_1^* = 0.244 )</td>
<td>( I_1^* = 0 )</td>
</tr>
<tr>
<td>( 2 )</td>
<td>( \theta_2^* = 0.137 )</td>
<td>( I_2^* = 1 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( m = 31,200 )</td>
<td>( \theta_m^* = 0.320 )</td>
<td>( I_m^* = 0 )</td>
</tr>
<tr>
<td>Mean</td>
<td>0.2183 (0.003)</td>
<td>0.0556 (0.0013)</td>
</tr>
<tr>
<td>SD</td>
<td>0.04528</td>
<td>—</td>
</tr>
<tr>
<td>Density</td>
<td>(like the bottom)</td>
<td></td>
</tr>
<tr>
<td>Trace</td>
<td>plot on p. 14)</td>
<td>—</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Phase</th>
<th>( \theta )</th>
<th>( I(\theta \leq 0.15) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initialization</td>
<td>( \theta^*_0 = 0.200 )</td>
<td>—</td>
</tr>
<tr>
<td>1</td>
<td>Burn-in</td>
<td>( \theta^*_1 = 0.244 )</td>
<td>—</td>
</tr>
<tr>
<td>( b = 500 )</td>
<td>Burn-in</td>
<td>( \theta^*_b = 0.098 )</td>
<td>—</td>
</tr>
<tr>
<td>( (b + 1) = 501 )</td>
<td>Monitoring</td>
<td>( \theta^*_{b+1} = 0.275 )</td>
<td>( I^*_{b+1} = 0 )</td>
</tr>
<tr>
<td>( (b + m) = 31,700 )</td>
<td>Monitoring</td>
<td>( \theta^*_{b+m} = 0.120 )</td>
<td>( I^*_{b+m} = 1 )</td>
</tr>
</tbody>
</table>

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 \( \rho_k \) of a stationary time series \( \theta_t^* \) at lag \( k \) (see, e.g., Chatfield (1996)) is \( \gamma_k / \gamma_0 \), where \( \gamma_k \) 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. 26, produced by ordinary rejection sampling.

Because of the Markov character of the columns of the MCMC data set on p. 27, 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. 25 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 three goals back on p. 24 which 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 26 and 27).
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 \( \theta^* \) from the **proposal distribution** \( g(\theta|y) \) (the normalized version of the envelope function \( G \)) and either accept a “move” to \( \theta^* \) or reject it, according to the **acceptance probability** \( \frac{p(\theta^*|y)}{G(\theta^*|y)} \); if accepted the process moves to \( \theta^* \), 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 \( \theta_t \) 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 $\theta_t$ stand for where you are now and $\theta^*$ for where you’re thinking of going, in this approach there is 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:

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

(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).
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

\[
\text{Initialize } \theta_0; \; t \leftarrow 0 \\
\text{Repeat } \\
\quad \text{Sample } \theta^* \sim g(\theta|\theta_t, y) \\
\quad \text{Sample } u \sim \text{Uniform}(0,1) \\
\quad \text{If } u \leq \alpha_{MH}(\theta^*|\theta_t, y) \text{ then } \theta_{t+1} \leftarrow \theta^* \\
\quad \text{else } \theta_{t+1} \leftarrow \theta_t \\
\quad t \leftarrow (t + 1)
\]

(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 \( \theta_t \), 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 \( \mu \) and unknown variance \( \sigma^2 \) applied to the NB10 data in part 2 of the lecture notes.

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

\[
l(\sigma^2|y) = c \prod_{i=1}^{n} (\sigma^2)^{-\frac{1}{2}} \exp \left[ -\frac{(y_i - \mu)^2}{2\sigma^2} \right] \\
= c (\sigma^2)^{-\frac{n}{2}} \exp \left[ -\frac{\sum_{i=1}^{n} (y_i - \mu)^2}{2\sigma^2} \right].
\]

(16)
MH Sampling (continued)

As you'll explore further in homework 3, this is recognizable as a member of the **Scaled Inverse \( \chi^2 \) family** \( \chi^{-2}(\nu, s^2) \) (e.g., Gelman, Carlin, et al. (1995)) of distributions, which (as we saw in part 2 of the lecture notes) is a **rescaled version** of the Inverse Gamma family chosen so that \( s^2 \) is an estimate of \( \sigma^2 \) based upon \( \nu \) "observations."

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

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

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

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) \( g(\sigma^2 | \sigma_t^2, y) \).

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 \((\theta^*)\), given that you accept a move away from where you are now \((\theta_t)\), 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 \( \theta_t \), 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 \( \chi^2 \) distribution: \( g(\sigma^2|\sigma_t^2, y) = \chi^{-2}(\nu_*, \sigma^2_*) \).

This distribution has mean \( \frac{\nu_*}{\nu_* - 2} \sigma^2_* \) for \( \nu_* > 2 \).

To use idea (2), then, I can choose any \( \nu_* \) 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 \( \sigma_t^2 \) as desired.

So I’ll use

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

This leaves \( \nu_* \) as a kind of potential tuning constant—the hope is that I can vary \( \nu_* \) 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 $\nu_*$ are $(0.07, 0.44, 0.86)$, respectively.

The **SD** of the $\chi^2(\nu_*, \frac{\nu_*-2}{\nu_*} \sigma_t^2)$ distribution is proportional to $\frac{\nu_*^2}{(\nu_*-2)^2 \sqrt{\nu_*-4}}$, which decreases as $\nu_*$ increases, and this turns out to be **crucial**: when the proposal distribution SD is too **large** (small $\nu_*$, as in the top panel in the figure), the algorithm tries to make **big jumps** around $\theta$ 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 $\nu_*$; 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 \( \theta \) 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 $(\theta, A)$ of quantities to be sampled is a vector of length $k$, there is additional flexibility: you can block update all of $(\theta, A)$ at once, or with appropriate modifications of the acceptance probability you can divide $(\theta, A)$ 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 $\theta$.

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 $\lambda_1$ first, then $\lambda_2$, and so on until you've updated $\lambda_k$, which concludes iteration $t$.

Let $\lambda_{t,i}$ stand for the current state of component $i$ at the end of iteration $t$, and let $\lambda_{-,i}$ stand for the $(\theta, A)$ 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 $(\theta, A)$; here $\lambda_{t,-i}$ is the current state of $\lambda_{-,i}$ 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 \( \lambda_i^* \) that creates the correct equilibrium distribution turns out to be

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

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

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 \( \lambda_i \): 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 \( \theta \) is replaced by \((\theta, A)\) in the summary).
Gibbs Sampling (continued)

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

\[
\text{Initialize } \theta_{0,1}, \ldots, \theta_{0,k}; \; t \leftarrow 0 \\
\text{Repeat } \{ \\
\text{Sample } \theta_{t+1,1} \sim p(\theta_1 | y, \theta_{t,2}, \theta_{t,3}, \theta_{t,4}, \ldots, \theta_{t,k}) \\
\text{Sample } \theta_{t+1,2} \sim p(\theta_2 | y, \theta_{t+1,1}, \theta_{t,3}, \theta_{t,4}, \ldots, \theta_{t,k}) \\
\text{Sample } \theta_{t+1,3} \sim p(\theta_3 | y, \theta_{t+1,1}, \theta_{t+1,2}, \theta_{t,4}, \ldots, \theta_{t,k}) \\
\vdots \\
\text{Sample } \theta_{t+1,k} \sim p(\theta_k | y, \theta_{t+1,1}, \theta_{t+1,2}, \theta_{t+1,3}, \ldots, \theta_{t+1,k-1}) \\
\} \\
\text{t } \leftarrow (t + 1) \\
\tag{20}
\]

Example: the NB10 Data. Recall from the posterior predictive plot toward the end of part 2 of the lecture notes 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 \( \nu \).

One way to write this model is

\[
(\mu, \sigma^2, \nu) \sim p(\mu, \sigma^2, \nu) \\
(y_i | \mu, \sigma^2, \nu) \overset{\text{IID}}{\sim} t_\nu(\mu, \sigma^2), \tag{21}
\]

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

(As noted in **homework 2**, $\lambda \sim \Gamma^{-1}(\alpha, \beta)$ 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**

\[
(\lambda|\nu) \sim \Gamma^{-1}\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \\
(y|\mu, \sigma^2, \lambda) \sim N(\mu, \lambda \sigma^2).
\]  
(22)

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

\[
\nu \sim p(\nu) \\
\sigma^2 \sim \text{SI-}\chi^2(\nu_0, \sigma_0^2) \\
(\mu|\sigma^2) \sim N\left(\mu_0, \frac{\sigma^2}{\kappa_0}\right) \\
(\lambda_i|\nu) \sim \Gamma^{-1}\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \\
(y_i|\mu, \sigma^2, \lambda_i) \sim \text{indep } N(\mu, \lambda_i \sigma^2).
\]  
(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 is a special case with $\nu = \infty$.  

39
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**:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Phase</th>
<th>( \mu )</th>
<th>( \sigma^2 )</th>
<th>( \nu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initializing</td>
<td>( \mu_0 )</td>
<td>( \sigma^2_0 )</td>
<td>( \nu_0 )</td>
</tr>
<tr>
<td>1</td>
<td>Burn-In</td>
<td>( \mu_1(y, \sigma^2_1, \nu_0) )</td>
<td>( \sigma^2_1(y, \mu_1, \nu_0) )</td>
<td>( \nu_1(y, \mu_1, \sigma^2_1) )</td>
</tr>
<tr>
<td>2</td>
<td>Burn-In</td>
<td>( \mu_2(y, \sigma^2_1, \nu_1) )</td>
<td>( \sigma^2_2(y, \mu_2, \nu_1) )</td>
<td>( \nu_1(y, \mu_2, \sigma^2_2) )</td>
</tr>
<tr>
<td>( b )</td>
<td>Burn-In</td>
<td>( \mu_b )</td>
<td>( \sigma^2_b )</td>
<td>( \nu_b )</td>
</tr>
<tr>
<td>( b + 1 )</td>
<td>Monitoring</td>
<td>( \mu_{b+1} )</td>
<td>( \sigma^2_{b+1} )</td>
<td>( \nu_{b+1} )</td>
</tr>
<tr>
<td>( b + 2 )</td>
<td>Monitoring</td>
<td>( \mu_{b+2} )</td>
<td>( \sigma^2_{b+2} )</td>
<td>( \nu_{b+2} )</td>
</tr>
<tr>
<td>( b + m )</td>
<td>Monitoring</td>
<td>( \mu_{b+m} )</td>
<td>( \sigma^2_{b+m} )</td>
<td>( \nu_{b+m} )</td>
</tr>
</tbody>
</table>

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 \( \mu \), \( p(\mu | y) \), 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 \( \mu \), 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:

\[ \sigma^2 \sim \text{SI-}\chi^2(\nu_0, \sigma_0^2) \]

\[ (\mu|\sigma^2) \sim N\left(\mu_0, \frac{\sigma^2}{\kappa_0}\right) \]

\[ (Y_i|\mu, \sigma^2) \overset{\text{IID}}{\sim} N(\mu, \sigma^2) . \]

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

\[ p(\mu|\sigma^2, y) = \frac{p(\mu, \sigma^2, y)}{p(\sigma^2, y)} \]

\[ = \frac{c \ p(\mu, \sigma^2, y)}{p(\sigma^2, y)} \]

\[ = \frac{c \ p(\sigma^2) \ p(\mu|\sigma^2) \ p(y|\mu, \sigma^2)}{p(\sigma^2)} \]

\[ = \frac{c \ \exp\left[-\frac{\kappa_0}{2\sigma^2} (\mu - \mu_0)^2\right] \prod_{i=1}^{n} \exp\left[-\frac{1}{2\sigma^2} (y_i - \mu)^2\right]}{p(\sigma^2)}. \]
Full Conditionals

From this

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

Expanding out the squares, collecting powers of \( \mu \), and completing the square in \( \mu \) gives

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

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

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

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

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

\[ (\sigma^2)^{-\frac{n}{2}} \exp \left[ -\frac{\kappa_0}{2\sigma^2} (\mu - \mu_0)^2 \right] \cdot \]

\[ (\sigma^2)^{-\frac{n}{2}} \exp \left[ -\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mu)^2 \right]. \]

When this is simplified you get
Full Conditionals (continued)

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

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

From the form of this distribution it becomes clear that

\[(\sigma^2 | \mu, y) \sim \text{SI-}\chi^2 \left( \nu_0 + 1 + n, \frac{\nu_0 \sigma^2 + \kappa_0 (\mu - \mu_0)^2 + n s^2_\mu}{\nu_0 + 1 + n} \right). \]

(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 two such programs now exist:

- **[BUGS]**, a fairly general-purpose Gibbs sampling program produced by David Spiegelhalter and others at the MRC Biostatistics Unit in Cambridge, UK (Spiegelhalter et al., 1997), and

- **[MLwiN]**, a program that does both maximum-likelihood and Bayesian calculations in hierarchical (multilevel) models (Rasbash et al. 2000).

**BUGS** runs under Unix or DOS in a wide variety of hardware configurations, and a Windows version called WinBUGS is also available; we will look here at both Unix BUGS and WinBUGS (together with MLwiN if there's time).

**BUGS** and WinBUGS are available for free downloading at

[www.mrc-bsu.cam.ac.uk/bugs](http://www.mrc-bsu.cam.ac.uk/bugs)

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

[mlwi.net](http://mlwi.net/)

43