A Generic Metropolis Sampler

It's useful to have a **generic Metropolis sampler** in your toolkit, because (a) working out full-conditional distributions can be tedious and error-filled and (b) sometimes you can't use Gibbs sampling.

With a **parameter vector** $\theta$ of length $k$, it's natural to use **Gaussian proposal distributions** (PDs) as the basis of this generic sampler; I can either do this on each component of $\theta$ one at a time with a **univariate Gaussian PD** — this is single-scan Metropolis — or I can block-update some or all of the components of $\theta$ with a **multivariate Gaussian PD**.

(This may sound **silly** if some or all of the components of $\theta$ don't live on the **entire real line**, but we can solve this with **re-parameterization**; see below.)

The **simplest** Gaussian PDs are centered at the **current value** $\theta_t$; such PDs have the **property** that the **probability** of generating a move to $\theta^*$ from $\theta_t$ depends only on the **distance** $|\theta^* - \theta_t|$ between the **target** and **current locations** — in other words, there is a density $h$ such that $\text{PD} (\theta^* | \theta_t) = h(|\theta^* - \theta_t|)$.

People call an **MCMC sampler** based on such a **proposal** a **random-walk** Metropolis (or Hastings) algorithm, because the **output** of the sampler, examined only at the **times** when you actually do **make a move**, forms a **random walk** in $\mathbb{R}^k$.

These samplers are an **important special case** of the **general Metropolis idea**, since it turns out both that they're **easy to program** and they tend to have **decent MCMC mixing properties**.

If the components of $\theta$ are (close to) **uncorrelated** in the posterior, there's no Monte-Carlo-efficiency advantage in **block updating**, but — if some of them are **strongly correlated** — **block updating** can (significantly) improve the **efficiency** of the **sampler**.
Generic Metropolis

Here are two ways to figure out if some components of \( \theta \) are correlated in the posterior:

- Write a single-scan Metropolis sampler, run it, and look at the output for correlations among the \( \theta \) columns in the MCMC data set; if all such correlations are small, you’re done; if not, and efficiency is important, write a Metropolis sampler that either block-updates the components that are (strongly) correlated and single-scans the rest or (possibly better) block-updates on the entire \( \theta \) vector.

- Compute \( \tilde{I}^{-1} \), the approximate posterior covariance matrix with a diffuse prior, and look at its correlation patterns to see how to block-update.

Example. With the heavy-tailed NB10 data examined earlier, the \( t \) model we talked about previously is

\[
(\mu, \sigma, \nu) \sim p(\mu, \sigma, \nu) \\
(y_i|\mu, \sigma, \nu) \overset{\text{IID}}{\sim} t_{\nu}(\mu, \sigma^2), \quad i = 1, \ldots, n = 100. \tag{27}
\]

The idea behind the generic strategy I want to look at here is that, possibly after appropriate re-parameterization, the posterior distribution for \( \theta \) should be close to multivariate normal for moderate to large \( n \), say \( p(\theta|y) \sim N_k(\mu, \Sigma) \).

This suggests a random-walk Metropolis with multivariate normal proposal distribution, centered at where you are now, and with covariance matrix a multiple \( \kappa \) of \( \Sigma \), for suitably chosen \( \kappa \); this will accurately reflect any posterior correlations, thereby improving the efficiency of the sampling.

The steps of the strategy are thus as follows.

1. Transform any components of \( \theta \) that live only on a subset of the real line to all of \( \mathbb{R} \); rewrite the log likelihood in this new parameterization, and recompute the log prior by including the appropriate Jacobian.

\[ * \text{ no Jacobian in likelihood in this section} \]
Generic Metropolis

(2) Use pen and paper, EM or a symbolic computing package to find the posterior mode $\theta_m$; symbolically or numerically obtain the Hessian $H$ (the second partial derivative matrix) of the log posterior, evaluate it numerically at the posterior mode, and compute $\hat{\Sigma} = -H^{-1}\big|_{\theta_m}$; if the prior is diffuse you can replace “posterior mode” by “MLE” and “log posterior” by “log likelihood.”

(3) Code and run a Metropolis sampler that makes $N(\theta, \kappa \hat{\Sigma})$ moves, varying $\kappa$ to achieve a decent acceptance rate; Gelman et al. (1996) have shown that, in a particular class of problems that should give some guidance here, the optimal $\kappa$ behaves roughly like $\frac{5.8}{k}$, and the optimal acceptance rate decreases from about 0.44 for $k=1$ to about 0.27 for $k=10$, roughly along the curve

$$k = \text{length}(\theta) \
0.23 + \frac{0.26}{k} - \frac{0.046}{k^2}.$$ 

If step (2) is too difficult, you’ll need another way to get an approximate $\Sigma$; the simplest idea is probably to use a single-scan sampler to get yourself started, and then switch over to step (3).

One nice thing about MCMC is that, even with an inefficient proposal distribution, the output of the chain — once equilibrium has been reached — is a valid sample from the posterior, so you can try an iterative strategy like the following:

Start with a poorly-tuned proposal; run it awhile; use the sample covariance matrix based on the columns of the resulting MCMC data set as an initial estimate $\hat{\Sigma}_0$ of $\Sigma$; run for awhile with a multivariate normal proposal based on $\hat{\Sigma}_0$; use the sample covariance matrix from the columns of this MCMC data set to produce a better estimate $\hat{\Sigma}_1$; and so on.
Adaptive MCMC

This is called adaptive Metropolis(-Hastings) sampling, and there's only one thing to watch out for: if you keep indefinitely refining the proposal distribution adaptively, based on the previous output of the chain, it's been shown that the sampler will not (necessarily) converge to the right equilibrium distribution; so you need to stop the adaptive process at some point before monitoring to produce the results you'll announce, because if not you may well be monitoring the wrong distribution.

In a bit more detail, the alternative strategy is as follows.

(2')(a) Code a Metropolis sampler that makes a series of $N(\theta_t, \kappa_j \sigma_j^2)$ moves, one for each parameter $\theta_j$, obtaining estimates for the $\sigma_j^2$ by whatever means you can think of (likelihood theory, iterative guesswork, ...) and varying the $\kappa_j$ so that the acceptance probabilities are around 0.44; set $s = 0$.

(2')(b) Run this sampler for awhile from a good starting value and with a bigger-than-usual burn-in, and use the sample covariances of the columns of the resulting MCMC data set to construct an estimate $\Sigma_s$ of $\Sigma$; if $s > 0$ and $\Sigma_s$ and $\Sigma_{s-1}$ don't differ too much, go to (4').

(3') Code and run a Metropolis sampler that makes $N(\theta_t, \kappa \Sigma_s)$ moves, varying $\kappa$ to optimize the acceptance probability as usual; when you have a $\kappa$ you like, increment $s$ and go back to (2')(b).

(4') Now, finally, make your official burn-in and monitoring run for the money using the final $\Sigma_s$.

Note that the correct sequence is adaptation, burn-in, monitoring.
Adaptive MCMC

(In other words, I'm suggesting either strategy \( \{ (1), (2), (3) \} \) or \( \{ (1), (2'), (3'), (4') \} \); and the main use of this is when you need a really efficient sampler, e.g., one that will be inside a loop in which you're evaluating the calibration properties of a Bayesian inferential procedure and you need to run that procedure on many simulated data sets.)

As another example of adaptive MCMC, MLwiN (Browne and Draper 2006, Draper 2008) uses random-walk Metropolis in random-effects and mixed (fixed plus random effects) models, and chooses the PD variances adaptively:

From starting values based on the estimated covariance matrices of the MLEs for the parameters in the given model, the MLwiN method first employs a sampling period of random length (but with an upper bound) during which the proposal distribution variances are adaptively tuned and eventually fixed for the remainder of the run; this is followed by a burn-in period; and then the main monitoring run from which posterior summaries are calculated occurs.

The tuning of the proposal distribution variances is based on achieving an acceptance rate \( r \) for each parameter that lies within a specified tolerance interval \( (r - \delta, r + \delta) \).

The algorithm examines empirical acceptance rates in batches of 100 iterations, comparing them for each parameter with the tolerance interval and modifying the proposal distribution appropriately before going on to the next batch of 100.
Adaptive MCMC

With $r^*$ as the acceptance rate in the most recent batch and $\sigma_p$ as the proposal distribution SD for a given parameter, the modification performed at the end of each batch is as follows:

$$\begin{align*}
\text{If } r^* &\geq r, \quad \sigma_p \rightarrow \sigma_p \left[ 2 - \left( \frac{1 - r^*}{1 - r} \right) \right], \\
\text{else } \sigma_p &\rightarrow \frac{\sigma_p}{2 - \frac{r^*}{r}}.
\end{align*}$$

(28)

This modifies the PD standard deviation by a greater amount the farther the empirical acceptance rate is from the target $r$: if $r^*$ is too low, the proposed moves are too big, so $\sigma_p$ is decreased; if $r^*$ is too high, the parameter space is being explored with moves that are too small, and $\sigma_p$ is increased.

If the $r^*$ values are within the tolerance interval during 3 successive batches of 100 iterations, the parameter is marked as satisfying its tolerance condition, and once all parameters have been marked the overall tolerance condition is satisfied and adapting stops.

After a parameter has been marked it’s still modified as before until all parameters are marked, but each parameter only needs to be marked once for the algorithm to end.

To limit the time spent in the adapting procedure, an upper bound is set (the MLwiN default is 5,000 iterations) and after this time the adapting period ends regardless of whether the tolerance conditions are met (in practice this occurs rarely).

Values of $(r, \delta) = (0.5, 0.1)$ appear to give near-optimal univariate-update Metropolis performance for a wide variety of multilevel models (Browne and Draper (2000, 2002, 2006)).
Here's an example of strategy \{ (1), (2), (3) \} on the NB10 data; this requires creating the new parameters \( \eta = \log(\sigma) \) and \( \lambda = \log(\nu) \), and rewriting the model in terms of \( \theta = (\mu, \eta, \lambda) \).

The log likelihood function, in this parameterization, is

\[
\begin{align*}
\log l(\mu, \eta, \lambda | y) &= c + n \log \left[ \Gamma \left( \frac{e^\lambda + 1}{2} \right) \right] - n \eta \\
&\quad - n \log \left[ \Gamma \left( \frac{e^\lambda}{2} \right) \right] - \frac{n \lambda}{2} \\
&\quad - \frac{e^\lambda + 1}{2} \sum_{i=1}^{n} \log \left[ 1 + e^{-(\lambda + 2\eta)}(y_i - \mu)^2 \right],
\end{align*}
\]

where \( \Gamma(\cdot) \) is Euler's Gamma (generalized factorial) function.

By way of a prior I've tried to bring in a modest amount of information that accords with the science of the problem; with \( n = 100 \) observations it should be OK to use a prior with independent components, because any strong posterior correlations that should be present will be accurately learned from the data, and having transformed to the log scale for \( \sigma \) and \( \nu \) it should be reasonable to work with Gaussians, so I took

\[
p(\mu, \eta, \lambda) = N(\mu | \mu_0, \sigma_\mu^2) N(\eta | \eta_0, \sigma_\eta^2) N(\lambda | \lambda_0, \sigma_\lambda^2). \tag{30}
\]

This reduces the elicitation problem to that of specifying the prior means and SDs for each of \( \mu, \eta, \) and \( \lambda \).

- \( \mu \) represents the true weight of the block of metal NB10, which is supposed to weight around 10g, and the observations are in micrograms below this nominal weight; to give the National Bureau of Standards (NBS) the benefit of the doubt, I should probably take \( \mu_0 = 0 \),
but $\sigma_\mu$ should be big to reflect the possibility of substantial bias on the microgram scale; based on previous results with similar weighing equipment at the NBS (Ku, 1969), I’ve chosen $\sigma_\mu = 500$ in what follows.

- $\nu$ indexes the tail-weight of the underlying distribution of measurement errors; Churchill Eisenhart (1979, personal communication), a leading statistician at NBS for decades, is on record as saying that “Measurement error processes in the physical sciences, when investigators report all their apparently valid data, tend to behave roughly like $t$ on about 7 degrees of freedom”; I’ve interpreted this expert judgment, a bit liberally, as an approximate statement that $P(2 < \nu < 20) = 0.95$; on the log scale this creates a 95% prior interval for $\lambda$ of $(0.69, 3.0)$, which in the Gaussian world implies $\lambda_0 = 1.84$ and $\sigma_\lambda = 0.59$.

- $\sigma$ is related to the underlying SD of the measurement errors made by the NBS weighing process, through $SD(y) = \sqrt{\frac{\nu}{\nu-2}} \sigma$ (as long as $\nu > 2$); a conservative reading of Ku (1969) suggests that errors on the order of 1–200 on the microgram scale are possible, which I’ll translate into the statement $P \left( 1 < \sqrt{\frac{\nu}{\nu-2}} \sigma < 200 \right) = 0.95$ for elicitation purposes; taking $\nu = 7$ for simplicity, for $\eta = \log(\sigma)$ this statement corresponds to the 95% prior interval $(-0.17, 5.13)$, leading to a Gaussian prior mean of $\eta_0 = 2.48$ and SD of $\sigma_\eta = 1.35$.

Before doing any sampling it’s worth looking at the log posterior a bit to see if any pathologies should be anticipated; I can’t plot $p(\mu, \eta, \lambda | y)$ in all its glory, because we’re condemned to three visual dimensions, but — like the blind men and the elephant in the old story — I can try to create a mental image of the whole thing by looking at various views of it one by one.
This figure presents four such views of the log posterior in this problem; I drew the upper left panel by holding constant $\mu$ and $\eta$ at plausible values, not (perhaps) too far from their posterior modes (I took the sample mean for $\mu$ and the log of the sample SD for $\eta$), and tracing out the log posterior as a function of $\lambda$.

If the posterior is multivariate normal, this plot should look locally quadratic around its maximum, and — while it lacks a bit in the symmetry department — it’s at least bowl-shaped down with only a single maximum; the other two similar plots (not shown), obtained by fixing $(\mu, \lambda)$ and $(\eta, \lambda)$, are also reasonably well-behaved — in particular, there are no signs of multimodality.
The other three panels are contour plots of the log posterior, obtained by fixing one component of $\theta$ at a time and letting the other two vary; each of these graphs should look like a set of concentric ellipses if the posterior is close to multivariate normal, and as long as you cast a slightly generous eye on the $(\eta, \lambda)$ plot you can see that things are not terribly far from MVN; it's also interesting to note that $\eta$ and $\lambda$ are fairly strongly positively correlated in the posterior, which on reflection makes sense: if I gave you a moderate-$n$ sample of data with a few points that may or may not be outliers, it would be hard for you to tell if the underlying story was (small $\nu$, small $\sigma$) — in other words, the data really are $t_\nu$ — or (large $\nu$, large $\sigma$), which is like saying that the data are really (close to) Gaussian but just with a big SD; thus scale and shape are confounded in the $t$ family.
OK, now I'm ready to build my Metropolis sampler; I've done step (1) already (I didn't have to compute any Jacobians because it was easy to elicit on the transformed scale); here's some Maple code for step (2):

```maple
# Maple code to find the posterior mode and compute its
# approximate covariance matrix (based on the Hessian at the
# mode), in the NB10 $t$ model.
#
# (written by DD, with some help from Riccardo Gatto)
#
# Input:
# Reads from a file called "nb10.dat" which contains the NB10
# measurements, 1 per line for 100 lines
#
# Outputs:
# Obtains the MAP (maximum a posteriori) equations by
# differentiating the log posterior function symbolically,
# solves them numerically to get the mode, calculates the
# Hessian symbolically, and evaluates it numerically at the mode
#

n := 100;
readlib( readda );
y := readda( 'nb10.dat', float, 1 );

mu.0 := 0.0;
sigma.mu := 500.0;
eta.0 := 3.80;
sigma.eta := 0.77;
lambda.0 := 1.84;
sigma.lambda := 0.59;

log.prior := -0.5 * ( ( mu - mu.0 ) / sigma.mu )^2 - 0.5 * ( ( eta - eta.0 ) / sigma.eta )^2 - 0.5 * ( ( lambda - lambda.0 ) / sigma.lambda )^2;
```
NB10 Example

```plaintext
log.likelihood := n * lnGAMMA( 0.5 * ( exp( lambda ) + 1.0 ) ) - n * eta - n * lnGAMMA( 0.5 * exp( lambda ) ) - 0.5 * n * lambda - 0.5 * ( exp( lambda ) + 1.0 ) * sum( log( 1.0 + exp( -( lambda + 2.0 * eta ) ) ) * ( y[i] - mu )^2 ), i = 1 .. n );

log.posterior := log.prior + log.likelihood;
map.eq1 := diff( log.posterior, mu );
map.eq2 := diff( log.posterior, eta );
map.eq3 := diff( log.posterior, lambda );

fsolve( { map.eq1, map.eq2, map.eq3 }, { mu, eta, lambda }, { mu = 403 .. 406, eta = 1 .. 3, lambda = 0.5 .. 3.0 } );

# At this point Maple takes about 2.5 seconds at 333Mhz to
# iteratively solve the MAP equations, obtaining the values of
# mu, eta, and lambda listed below.

with( linalg );
H := hessian( log.posterior, [mu, eta, lambda] );

mu := 404.2956374;
eta := 1.346258072;
lambda := 1.259790967;

He := matrix( 3, 3, ( i,j ) -> 0 );
for i from 1 to 3 do
  for j from 1 to 3 do
    He[i,j] := evalf( H[i,j] );
  od:
od:

Sigma := inverse( - He );

# At this point Maple returns the covariance matrix:
#
# Sigma := [.2159336246 .002989379323 .008083662383 ]
#           [ .002989379323 .01193790730 .01486075432 ]
#           [ .008083662383 .01486075432 .07490518257 ]
```
From this covariance matrix we get approximate standard errors \((\sqrt{0.216} = 0.46, 0.11, 0.27)\) for \((\hat{\mu}, \hat{\eta}, \hat{\lambda})\) and approximate correlations \((\frac{0.00299}{\sqrt{0.215 \cdot 0.0119}} = 0.059, 0.064, 0.50)\) for \([(\hat{\mu}, \hat{\eta}), (\hat{\mu}, \hat{\lambda}), (\hat{\eta}, \hat{\lambda})]\); thus \(\mu\) is around \(404.3 \pm 0.46\); \(\eta\) is about \(1.346 \pm 0.11\), meaning that \(\sigma\) is likely to be in the range \((\exp(1.346 - 2 \cdot 0.11) = 3.1, \\
\exp(1.346 + 2 \cdot 0.11) = 4.8)\); and \(\lambda\) is around \(1.260 \pm 0.27\), so that \(\nu\) is probably in the interval \((2.1, 6.0)\); all of this is useful information in extracting the full posterior.

Here’s some R code to implement the generic Metropolis strategy in this example:

```r
# # R functions to do Metropolis sampling in the model # # theta = ( mu, eta, lambda ) # ( theta ) ~ N( mu.0, sigma.mu2 ) * N( eta.0, sigma.eta2 ) * # N( lambda.0, sigma.lambda2 ) # ( y_i | theta ) ~ IID t.exp( lambda ) ( mu, exp( 2 * eta ) ), # i = 1, ..., n # # # Inputs: # # y = data vector, of length n = sample size # mu.0 = prior mean for mu # sigma.mu = prior SD for mu # eta.0 = prior mean for eta # sigma.eta = prior SD for eta # lambda.0 = prior mean for lambda # sigma.lambda = prior SD for lambda # kappa = scaling factor for Metropolis proposal distribution # (affects the acceptance rate R; to increase R, decrease # kappa) # Sigma = proposal distribution covariance matrix # theta.0 = initial value for ( mu, eta, lambda ) in # Metropolis iterations # n.burnin = length of burn-in period # n.monitor = length of monitoring period # n.thin = thinning constant (only every n.thin-th iteration # in the monitoring period will be written to disk)
```

49
NB10 Example

# seed = random number seed (for generating repeatable
# sequences of Hastings iterations); must be an integer
# from 0 to 1000
# output.file.prefix = character string naming where you want
# the MCMC data set to go; for example, output.file.prefix
# = "NB10" would write the MCMC data set to the file
# "NB10.d"
#
# Outputs:
#
# Acceptance rate R returned when iterations are finished
# A file called paste( output.file.prefix, ".d", sep = "" ) is
# written (in the same directory where R has been called)
# containing one row for each monitored iteration and six
# columns: the monitored iteration number (from 1 to
# n.monitor/n.thin), the simulated draws from the posterior
# for theta = ( mu, eta, lambda ) for that iteration, and
# the corresponding simulated draws from the posterior for
# sigma = exp( eta ) and nu = exp( lambda ). If the output
# file exists before the function is invoked, it will be
# over-written

metropolis.t <- function( y, mu.0, sigma.mu, eta.0, sigma.eta, 
lambda.0, sigma.lambda, kappa, Sigma, theta.0, n.burnin, 
n.monitor, n.thin, seed, output.file.prefix ) {

# Main routine

theta.old <- theta.0
p <- length( theta.old )
R <- 0
write( c( theta.old, R ), "loop.result", append = F )
set.seed( seed )
L <- t( chol( Sigma ) )
L.kappa <- sqrt( kappa ) * L

for ( i in 1:( n.burnin + n.monitor ) ) {

  null <- loop( p, L.kappa, y, mu.0, sigma.mu, eta.0, sigma.eta, 
    lambda.0, sigma.lambda, output.file.prefix, i )
}

50
loop.result <- scan( "loop.result" )
R <- loop.result[p + 1]
return( R / n.monitor )

}

loop <- function( p, L.kappa, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda, output.file.prefix, i ) {

loop.result <- scan( "loop.result" )
theta.old <- loop.result[1:p]
R <- loop.result[p+1]
N <- length( y )

theta.star <- PD.sim( theta.old, p, L.kappa )
u <- runif( 1 )
b <- ( u <= alpha( theta.old, theta.star, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) )
theta.new <- theta.star * b + theta.old * ( 1 - b )

if ( ( i > n.burnin ) ) R <- R + b
if ( ( i > n.burnin ) & ( ( i - n.burnin ) %% n.thin == 0 ) )
write( c( ( i - n.burnin ) / n.thin, signif( c( theta.new, exp( theta.new[c( 2, 3 )] ) ), digits = 5 ) ), paste( output.file.prefix, ".d", sep = "" ), ncol = p + 3,
append = ( i > n.burnin + n.thin )

theta.old <- theta.new
write( c( theta.old, R ), "loop.result", append = F )
return( NULL )
}

PD.sim <- function( theta, p, L.kappa ) {

# Proposal distribution simulation
Z <- matrix( rnorm( p ), p, 1 )
Mu <- matrix( theta, p, 1 )
theta.star <- c( Mu + ( L.kappa %*% Z ) )

return( theta.star )
}
alpha <- function( theta.old, theta.new, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) {

  # Acceptance probability calculation
  return( min( 1, exp( log.post( theta.new, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) - log.post( theta.old, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) ) ) )
}

log.post <- function( theta, y, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) {

  # log( posterior ) calculation
  return( log.prior( theta, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) + log.lik( theta, y ) )
}

log.prior <- function( theta, mu.0, sigma.mu, eta.0, sigma.eta, lambda.0, sigma.lambda ) {

  # log( prior ) calculation (including Jacobian)
  mu <- theta[1]
  eta <- theta[2]
  lambda <- theta[3]

  return( -0.5 * ((mu - mu.0) / sigma.mu)^2 - 0.5 *
           ((eta - eta.0) / sigma.eta)^2 - 0.5 * ((lambda -
               lambda.0) / sigma.lambda)^2 )
}

log.lik <- function( theta, y ) {

  # log( likelihood ) calculation
  mu <- theta[1]
  eta <- theta[2]

lambda <- theta[3]
n <- length(y)

return( n * lgamma( 0.5 * ( exp( lambda ) + 1.0 ) ) - n * eta - 
n * lgamma( 0.5 * exp( lambda ) ) - 0.5 * n * lambda - 0.5 * 
( exp( lambda ) + 1.0 ) * sum( log( 1.0 + exp( - ( lambda + 
2.0 * eta ) ) * ( y - mu )^2 ) ) )
}

Note how little would have to be changed to use this sampler on a completely different problem.

With $\kappa = 2 \div \frac{5.8}{3}$, a single monitoring run of 45,000 after a burn-in of 2,000 from a starting value of

$\theta_0 = (404.6, 1.699, 1.946) = [\bar{y}, \log(\sqrt{\frac{5}{7}} s), \log(7)]$ produced the output summarized below; all standard MCMC diagnostics were well-behaved; the Metropolis acceptance rate was 0.31, which is near-optimal for $p = 3$ based on Gelman et al. (1996)’s results, so I didn’t try to look for a better $\kappa$.

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>SD</th>
<th>95% Central Interval</th>
<th>MLE (SE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu$</td>
<td>404.3</td>
<td>0.48</td>
<td>(403.4, 405.3)</td>
<td>404.3 (0.46)</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>3.92</td>
<td>0.44</td>
<td>(3.14, 4.87)</td>
<td>3.70 (0.42)</td>
</tr>
<tr>
<td>$\nu$</td>
<td>3.75</td>
<td>1.1</td>
<td>(2.15, 6.44)</td>
<td>3.01 (0.86)</td>
</tr>
</tbody>
</table>

- The posterior mean of the scale parameter $\sigma$ is substantially lower than the sample SD $s = 6.5$, but this is to be expected since $V(y) = \frac{\nu}{\nu-2}\sigma^2$ in this model (as long as $\nu > 2$); indeed, the sample average of the quantity $\frac{\nu\sigma^2}{\nu-2}$ across the rows of the MCMC data set with $\nu* > 2$ is 44.4, not far from the sample variance 41.8.
Bayesian and ML inferences with these data (with my prior, at least) are similar for $\mu$, but the posterior means are about 6% and 25% larger than the MLEs for $\sigma$ and $\nu$, respectively, and the MLE standard errors are smaller than the posterior SDs; part of this difference comes from the prior, part from the difference between means and modes for skewed distributions, and part from the way ML inference (sometimes inaccurately) deals with uncertainty about $\theta_{(j)}$ when summarizing uncertainty about $\theta_j$; note in particular how much smaller the SE for $\hat{\nu}_{\text{MLE}}$ is than the posterior SD for $\nu$. 

54
Most importantly, when you (incorrectly) assume a Gaussian model for these data, the posterior mean and SD of $\mu$ (which has the same meaning in both models, and is thus comparable) are 404.3 and 0.65; with the $t$ model the posterior mean is the same but the posterior SD, 0.47, is substantially (28%) smaller.

With access to the CDF of the $t_\nu$ distribution it’s easy enough to make a $t$ qqplot of a data set, but what should you choose for $\nu$? The left panel of the figure above gives one simple answer: I’ve plotted the correlation between the $t_\nu$ quantiles and the sorted NB10 data values as a function of $\nu$, and you can see that this is maximized for $\nu \approx 2$; so the right panel of the figure gives a $t_2$ qqplot, which does indeed fit the data pretty much like a glove; it’s interesting to consider why the apparently optimal value of $\nu$ in this sense is not particularly well supported by the posterior for $\nu$ summarized above.