Why the Metropolis Algorithm Works

Here's a sketch of the important 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 is 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 the event that the proposed move is **accepted**,

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

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

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

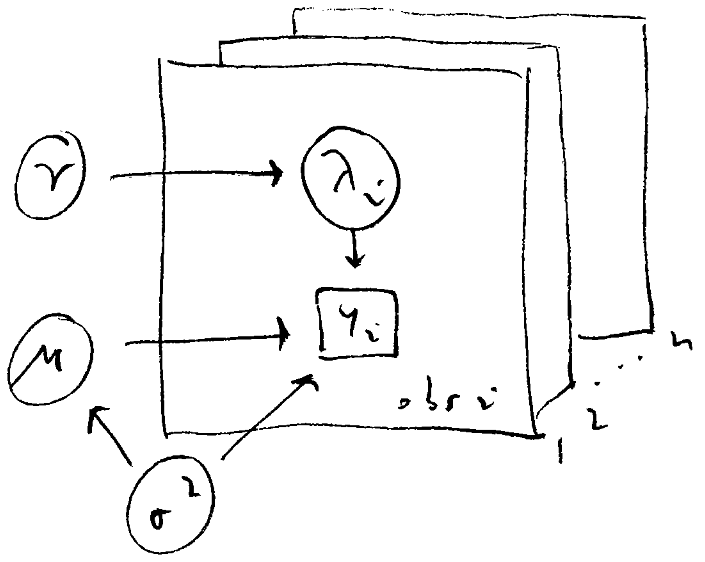
(1) Viewing Bayesian models as **directed (acyclic) graphs** (DAGs).

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 which 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 relatively efficient 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

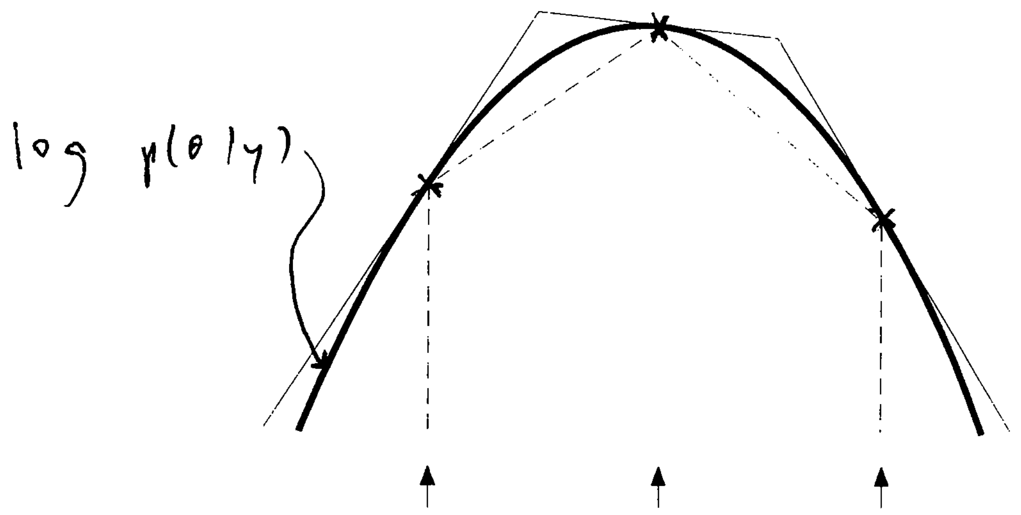
$$\frac{d^2 \log g}{dy\, dy^T} \tag{42}$$

is **non-positive**).

ARS (continued)

For univariate θ the idea (see the figure on p. 14) 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 (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.

BUGS (continued)

BUGS 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 "classic" BUGS **quits** and 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 am 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

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

for A, B **such that**

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

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

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

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
```

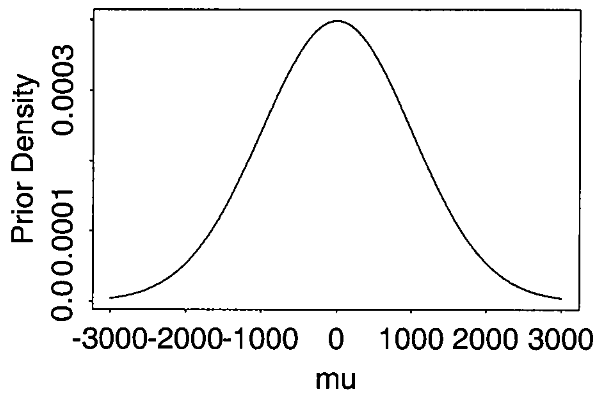
More Details

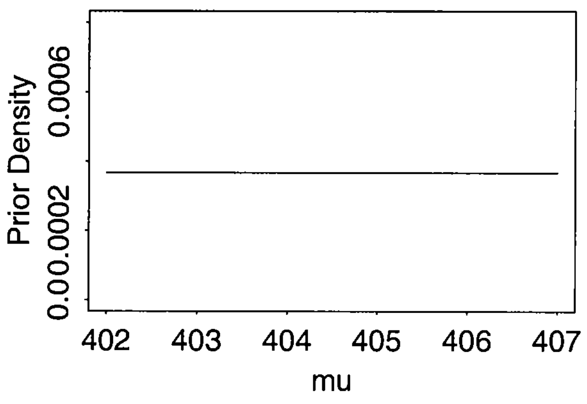
```
> print( n <- length( y ) )

[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
```

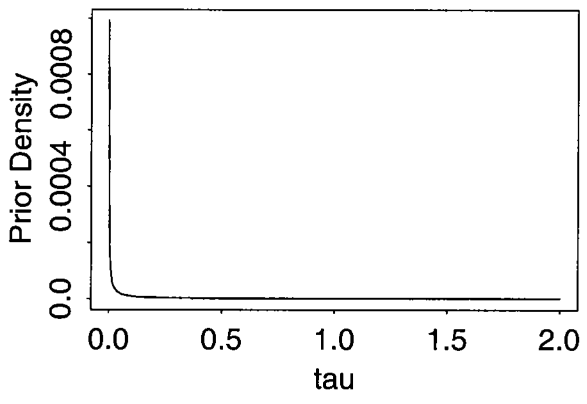
So the conclusion is that the **likelihood** for $\tau = \frac{1}{\sigma^2}$ should be **non-negligible** roughly in the region from about 0.015 to 0.035.

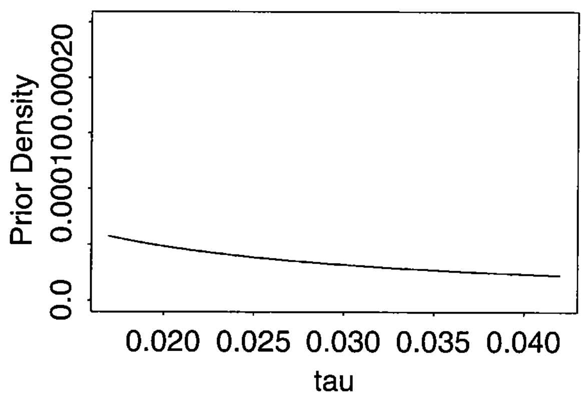
The figure below plots the prior distributions for μ and τ and **verifies their diffuseness** in the relevant regions.

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 which 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 $\tau = \frac{1}{\sigma^2} \doteq 0.04$.

$\boxed{\textbf{A Running Strategy.}}$ 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 54 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. 1995, 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 $\boxed{\texttt{CODA}}$ 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**.

$\boxed{\textbf{Geweke}}$ (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.