Hyperparameter Estimation with Bayesian Inference: Reversible Jump Markov Chain Monte Carlo for SNP Location Clustering

Yu Chen
CM226: Machine Learning in Bioinformatics Research Project
Professor Sriram Sankararaman
December 11th, 2016

In this research project report, I will provide a high-level overview of the fundamental concepts that form the basis for Reversible Jump Markov Chain Monte Carlo simulations, and ultimately provide two toy implementation examples of utilizing RJMCMC to estimate the posterior distribution of likely SNP clusters across real genetic data. I have also attached R code used to implement the simulation to the appendix of this report. The toy data sets and code I work with can be downloaded from www.github.com/ychennay/RJMCMC.

A fundamental problem encountered in bioinformatics research is the estimation of suitable hyperparameters for statistical modeling. In machine learning, optimization of regularized linear regression, Gaussian mixture models, convolutional neural networks, and other "workhorse" algorithms has often rested upon heuristics or other iterative, numerical methods such as cross-validation. Markov Chain Monte Carlo (MCMC) can be utilized as an iterative, stochastic, and probabilistic approach that converges towards a set of "informed" priors (and inputs) for further learning tasks. Examples of hyperparameters abound in the typical models and algorithms used for learning:

Statistical Model	Parameter	Comments
Mixture Models (EM, Gaussian, admixture)	k	In mixture modelling, k is the designated number of clusters. A variety of other parameters θ_k are generated for each cluster k, depending on the underlying distribution of cluster k.
Deep Neural Networks	α	$N_h = \frac{N_S}{\alpha(N_i+N_o)}$ where N_h is the number of hidden units, N_S is the number of samples in the training set, N_i is the number of input units, and N_o is the number of output units. α is a scaling factor usually between 2 and 10.[i]
Gradient Descent	η	In batch gradient descent, the update equation is $\theta_{new} = \theta_{old} - \eta \nabla_\theta J(\theta)$, where η is the learning rate and $\nabla_\theta J(\theta)$ is the gradient of the function to be optimized.
Regularization Procedures	λ	In the ridge regression equation $\widehat{w} = \arg\min_w (Y - Xw)^T(Y - Xw) + \lambda \, \|w\|_2^2$, λ is a fixed multiplier value where $\lambda = 0$ is unpenalized regression and often chosen through cross-validation.
Decision Trees	λ	The cost complexity criterion for decision trees is defined as $CCC(\hat{y}) = RSS(\hat{y}) + \lambda \, \|nodes(\hat{y})\|$[ii]. λ is a set of constraint parameters defining maximum number of data points per node, maximum depth of true, maximum number of nodes.

Monte Carlo Sampling

The implementation of MCMC begins with the Ordinary Monte Carlo (OMC) method, which attempts to solve the fundamental problem of calculating the expected value of a given function when circumstances make cleaner analytical methods impractical or impossible. For instance, the typical expression for the Bayes rule is

$$\mathbf{P(\theta|x) = \frac{P(x|\theta)P(\theta)}{P(x)}} \qquad \textbf{(Equation 1)}$$

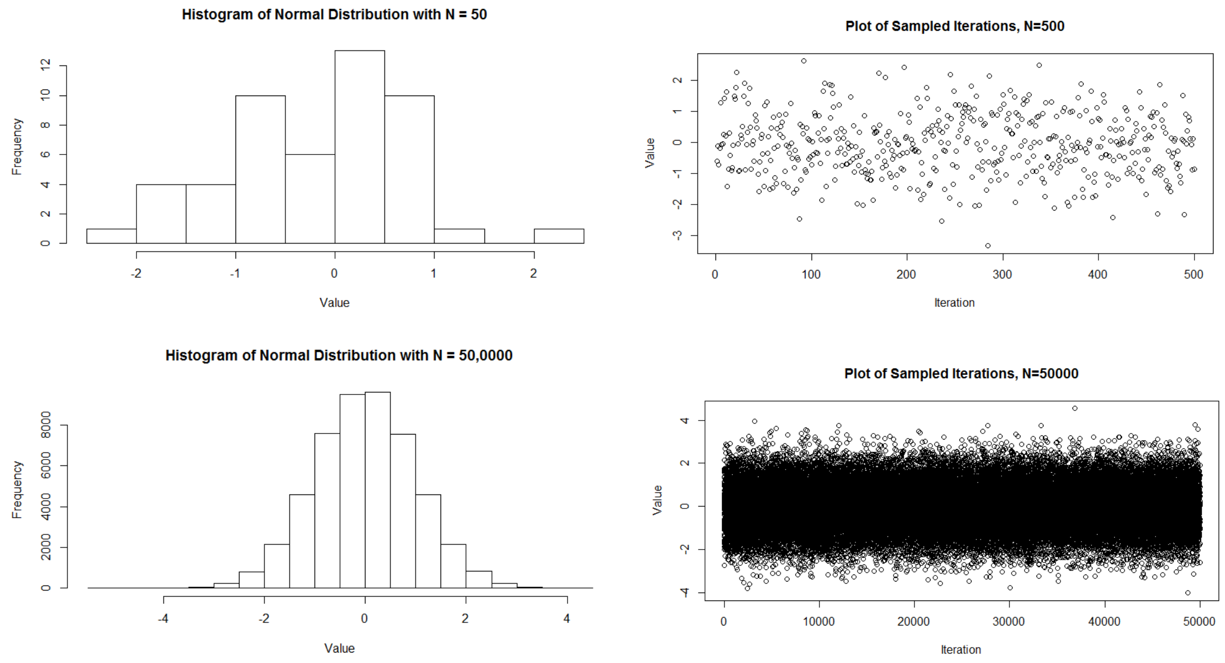
where $P(x)$, or intuitively, the probability of seeing a particular instance x{X} is defined as

$$\mathbf{P(x) = \int_\Theta P(x,\theta)\, d\theta} \qquad \textbf{(Equation 2)}$$

This cannot be calculated closed form for many models or distributions. To work around this issue, Monte Carlo generates random numbers from a distribution to approximate the parameters of this distribution. It has various applications in a variety of fields, including operations management, finance, social science, physics, etc. It utilizes an overarching theorem of statistics: The Strong Law of Large Numbers, which states that as the number of trials of a random process increase to infinity, the difference between the expected value of that process and the average values of the trials will converge to zero. In other words, $\bar{X} = \mu \; as \; n \rightarrow \infty$.

Its random number generation is derived from a proposal distribution that does not vary from one iteration to the next:

$$Proposal\ distribution_{OMC}: \theta_t \sim N(\mu, \sigma^2)$$



Random samples drawn from $N(\mu = 0, \sigma^2 = 1)$ with various values of N. Note that this data is stationary: it assumes constant mean and variance through each iteration 1 to N. Indeed, it makes use of the theorem that if F is a distribution function and ξ is a random number, then $F(\xi)$ is a random variable defined by the distribution function F.[iii] The implication of this is that analytical evaluations of F are not always necessary, especially as the number of available random samples approaches ∞. We can likewise define $E[g(X)] = \int g(x)f(x)dx$, where g(x) is the target function in question, with continuously distributed random variable X and a probability density function f(x). We can derive from $E[g(X)]$ a definition for $\hat{\mu}_n$:

$$\boldsymbol{\hat{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} g(X_i)} \qquad \textbf{(Equation 3)}$$

Where n is defined as the Monte Carlo sample size, and $\hat{\mu}_n$ is the Monte Carlo approximation. The variance of g(X) is defined as σ^2. Central Limit Theorem states that $\hat{\mu}_n$ is normally distributed, with mean μ and standard deviation $\frac{\sigma^2}{n}$:

$$\boldsymbol{\hat{\mu}_n \approx N\left(\mu, \frac{\sigma^2}{n}\right)} \qquad \textbf{(Equation 4)}$$

The estimated variance of this distribution is defined as

$\hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^{n}(g(X_i) - \hat{\mu}_n)^2$ **(Equation 5)**

An observation to highlight here is that Monte Carlo distributions follow the square root law: to increase the accuracy by 1000% (tenfold), there must be a 100x increase in the number of samples.

Stationary Distribution of Markov Chains Theorem

The probability of ending at any state y_T is independent of $y_{1:T-2}$. Intuitively, the only state that influences the current state is the previous state. Consequently, the transition probability distribution is P(X_{n+1}| X_n) and does not depend on n. The transition probabilities are encoded within a matrix A, where $a_{i,j} = p_{(i,j)}$, or the probability of transition from edge *i* to edge *j*:

$$\begin{matrix} a_{1,1} & a_{1,2} & \dots & a_{1,j} \\ a_{2,1} & \dots & \dots & a_{2,j} \\ \dots & \dots & \dots & \dots \\ a_{i,1} & a_{i,2} & \dots & a_{i,j} \end{matrix}$$

Therefore, $a_{i,j}$ is equivalent to P($X_{n+1} = x_j$| $X_n = x_i$). In Markov Chain Monte Carlo (MCMC), variance of the Markov Chain is now defined as

$\sigma^2 = var[g(X_i)] + 2\sum_{k=1}^{\infty} cov\,[g(X_i), g(X_{i+k})] = \Sigma$ **(Equation 6)**

Note that in Equation 6, $var[g(X_i)]$ is not a scalar, but rather is in the form of a covariance matrix. The Markov Chain Central Limit Theorem ultimately will state that given enough iterations,

$\hat{\mu}_n \approx N(\mu, n^{-1}\,\Sigma), where\ N\ is\ a\ normal\ distribution$ **(Equation 7)**

$\gamma_k = cov[g(X_i), g(X_{i+k})]$ **(Equation 8)**

The term γ_k is also the autocovariance of the function g(X). Equation 8 can be rewritten as

$\sigma^2 = \gamma_0 + 2\sum_{k=1}^{\infty}\gamma_k$ **(Equation 9)**

$\pi_t(j) = P(X_t = j) = \sum_i^I \pi_{t\text{-}1}(i)p_{i,j}$ **(Equation 10)**

In other words, the probability of state *j* at time *t* is the sum of the probabilities of moving to j from time *t – 1*.If the following conditions hold, then π is a stationary distribution of P:

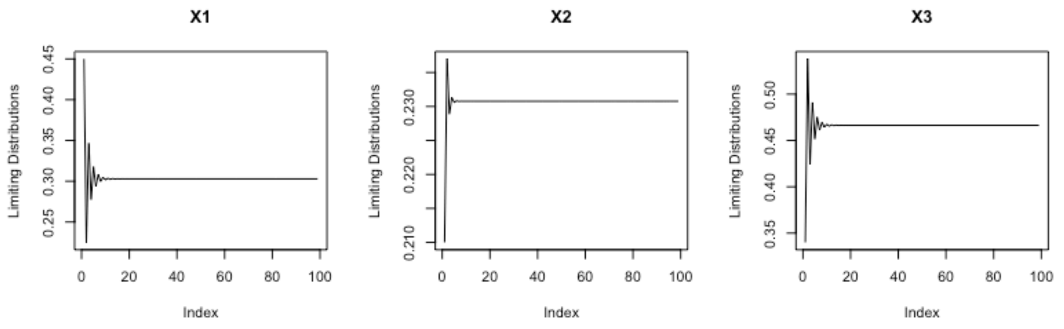
$\pi_i(p_{i,j}) = \pi_j(p_{j,i})\ where$ **(Equation 11)**
$\pi_i > 0$
$\sum_i \pi_i = 1$

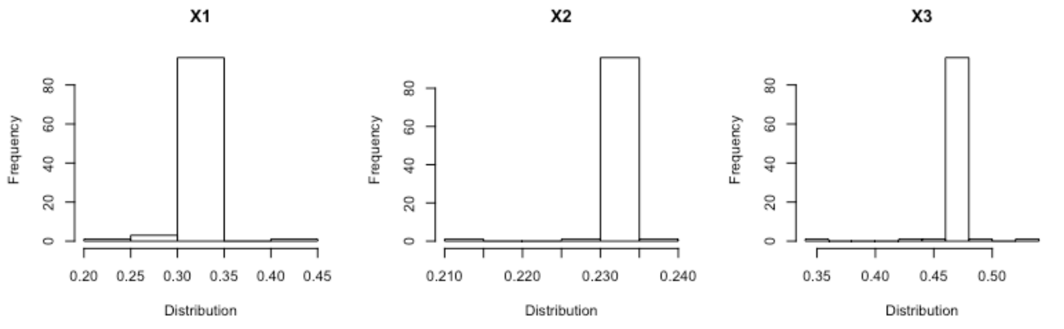
Due to the stationary distribution properties of the Markov Chain, after a given length time, we will arrive at the state j, with probability $\pi_t(j)$. The transition matrix for this particular graphical model on the left is provided on the right.

$$\begin{tikzcd}
X_1 \arrow[rr, bend left=10, "0.3"] \arrow[dr, bend left=10, "0.7"] & & X_2 \arrow[ll, bend left=10, "0.1"] \arrow[dl, bend left=10, "0.9"] \\
& X_3 \arrow[ul, bend left=10, "0.6"] \arrow[ur, bend left=10, "0.3"] \arrow[loop right, "0.1"] &
\end{tikzcd}$$

$$\begin{matrix} 0 & .3 & .7 \\ .1 & 0 & .9 \\ .6 & .3 & .1 \end{matrix}$$

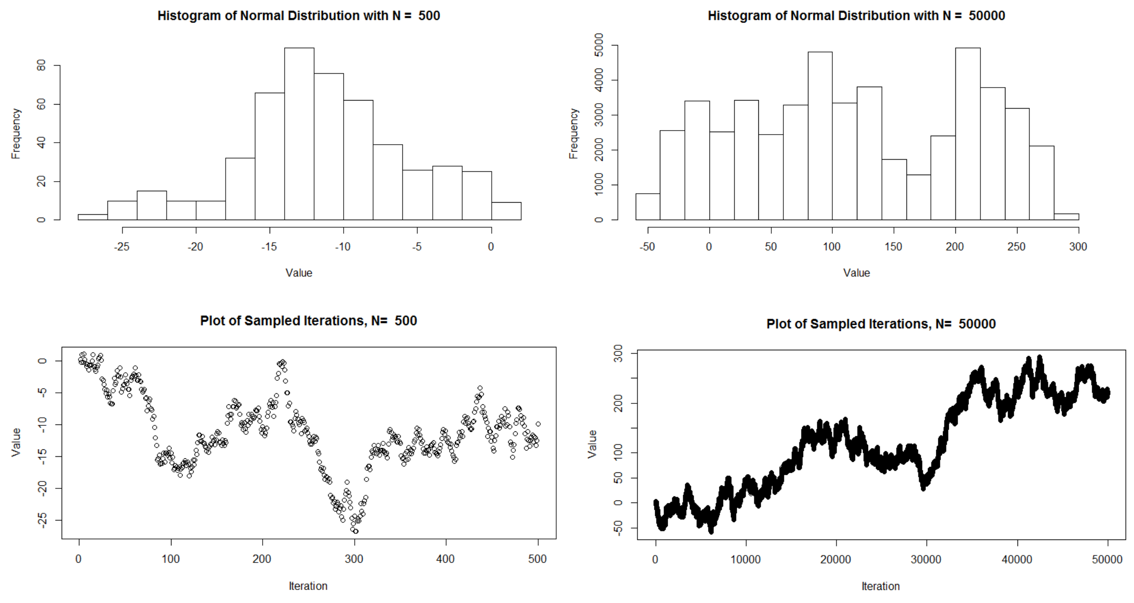
The MCMC chain must be ergodic, or visiting every point in the domain at a frequency proportionate to its true probability. Ergodic chains must have positive transition probabilities relative to other states (all other states are accessible given a particular state), and aperiodic (no cycles). In this particular case, our chain reaches a stationary point, as the various distributions between X_1, X_2, X_3 converge to point values (see below for trace plots and final distributions, and in appendix for implementation code). [iv] It is clear that this particular chain will spend almost half of its time in X_3 (distribution of approximately 0.48).





Next, consider a toy example of a Markov Chain Monte Carlo where the proposal distribution is drawn from the original normal distribution, but with

$$\textbf{Proposal distribution}_{MCMC}: \boldsymbol{\theta_t \sim N(\theta_{t\text{-}1}, \sigma^2)} \qquad \textbf{(Equation 12)}$$



Note here that the data is not stationary. It has constant variance but not constant mean. In fact, each initialization of a random walk may result in unique end points and E[θ]. The density plot (histogram) do not represent the normal distribution in the proposal distribution.

Metropolis-Hastings Algorithm

Markov Chain Monte Carlo ultimately, is a reconciliation of OMC and a random walk by utilization of an algorithm that acts as a switchpoint for acceptance or rejection of random sample sequences from a probability distribution. One prominent and frequently used method is the Metropolis-Hastings algorithm. The procedure for the Metropolis-Hastings algorithm is as follows:

Step 1 – Calculate the acceptance probability of the next iteration of θ:

$$\textit{acceptance probability} = \alpha(\theta_t, \theta_{t\text{-}1}) \qquad \textbf{(Equation 13)}$$

$$= \min[\, r(\theta_t, \theta_{t\text{-}1}), 1]\ \textit{where}$$

$$r(\theta_t, \theta_{t\text{-}1}) = \frac{\textit{Posterior probability}\ (\theta_t)}{\textit{Posterior probability}\ (\theta_{t\text{-}1})} \qquad \textbf{(Equation 14)}$$

$$= \frac{P(Y_t|\theta_t)}{P(Y_t|\theta_{t\text{-}1})} = \frac{P(\theta_t|Y_t)}{P(\theta_{t\text{-}1}|Y_t)}\, x\, \frac{h(\theta_t,\theta_{t\text{-}1})}{h(\theta_{t\text{-}1},\theta_t)}$$

In this case, h($\cdot$) is the random walk function from the Markov chain h(θ_t)= θ_{t+1}, where $\theta_t \sim N(\theta_{t-1}, \sigma^2)$.

Step 2 – Draw a value $u \sim Uniform(0,1)$:

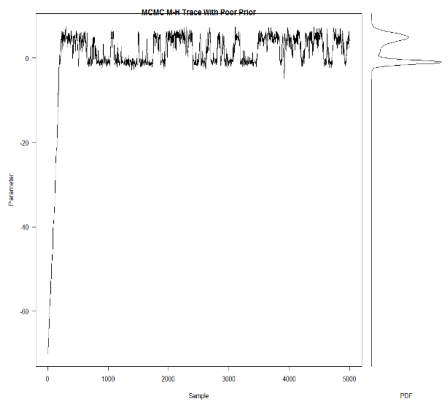
Note here that regardless of the value of u, if the probability of θ_t is greater than the probability of θ_{t-1}, then the acceptance probability will always be greater than 1. Selecting a value $u \sim Uniform(1 > a > 0,1)$ will "exaggerate" the differences between in probability distribution. This will cause the algorithm to achieve stationarity faster, but it may "settle" in to a non-optimum local position for a long, perhaps indefinite period of iterations.

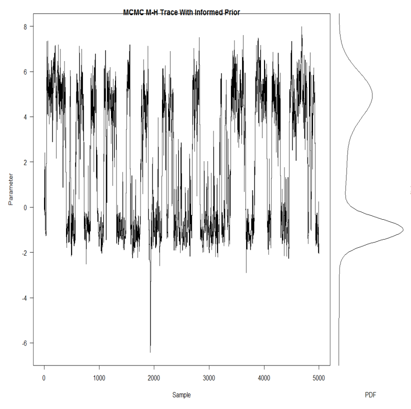
<u>Step 3 – Compare u with the acceptance probability:</u>

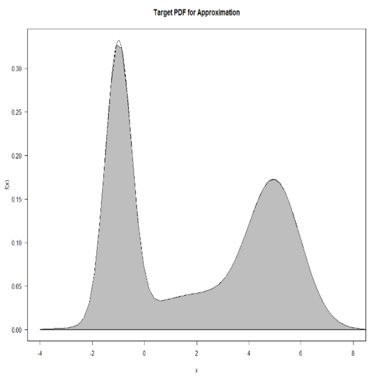
- If $u < \alpha(\theta_t, \theta_{t-1})$, then $\theta_t = \theta_t$ (accept the update)
- Else if $u \geq \alpha(\theta_t, \theta_{t-1})$, then $\theta_t = \theta_{t-1}$(reject the update)

Classical MCMC M-H Toy Data Implementation

For example, we attempt to approximate the true distribution of a certain univariate f(x) function (bottom) that is clearly bimodal when visualized in a two-dimensional space. The distribution is constructed from a mix of Gaussians with $\mu_1 = 1, \mu_2 = 2, \mu_3 = 5$. We initialize the MCMC M-H algorithm with an extremely biased prior (left), and a more informed prior (center), and run both for 5,000 iterations. Both achieve convergence to the approximate "true" distribution (right), albeit at different rates. However, it becomes clear that the required iterations to convergence is smaller with informed priors and larger steps.







We can approximate the amount of necessary iterations for convergence in MCMC M-H by examining autocorrelation, where the autocorrelation r_k is defined as

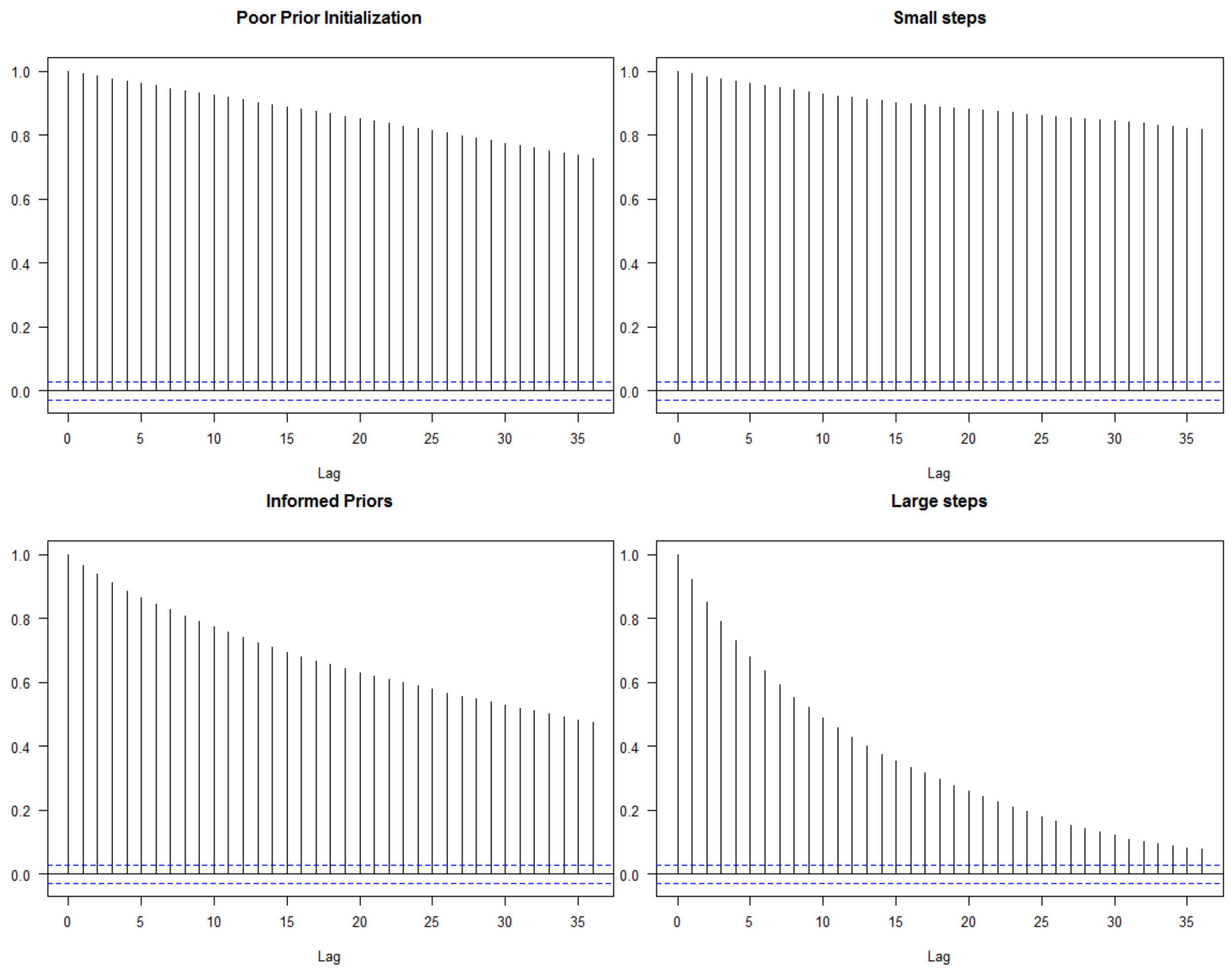
$$\frac{\sum_{i=1}^{N-k}(Y_i - Y)(Y_{i+k} - Y)}{\sum_{i=1}^{N}(Y_i - Y)^2} \quad \textbf{(Equation 15)}$$

The rate at which the MCMC algorithm moves across the parameter space it termed mixing; Jeff Gill has noted in Bayesian Methods: A Social and Behavioral Sciences Approach that slow mixing has two adverse effects:

1) It slows the movement towards the target distribution
2) Once at the target distribution, it impedes the algorithm's ability to explore the distribution space

Mixing tends to originate from highly correlated model parameters: perhaps for instance, a particular underlying Gaussian distribution increases in variance σ^2as its mean μ increases. It is recommended to into build into any implementation of MCMC an ability to assess the acceptance ratios calculated; a tell-tale symbol of poor mixing is low acceptance rates.[v]

When MCMC M-H reaches an acceptably low level of autocorrelation, the algorithm has achieved convergence. Moreover, at convergence MCMC M-H will have the property that the next iteration of will produce an identical distribution of its transition kernel: $\pi f = f$. The effects of step size and prior initialization on convergence are shown below.

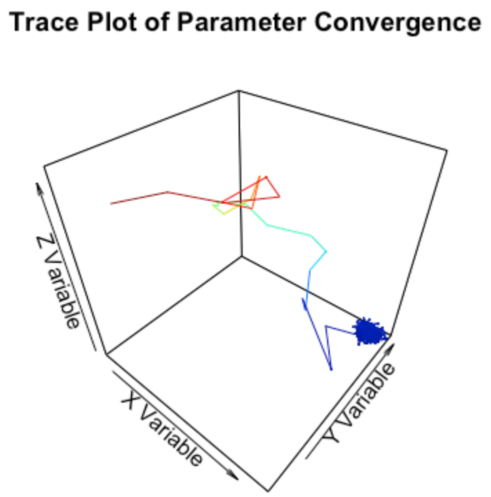


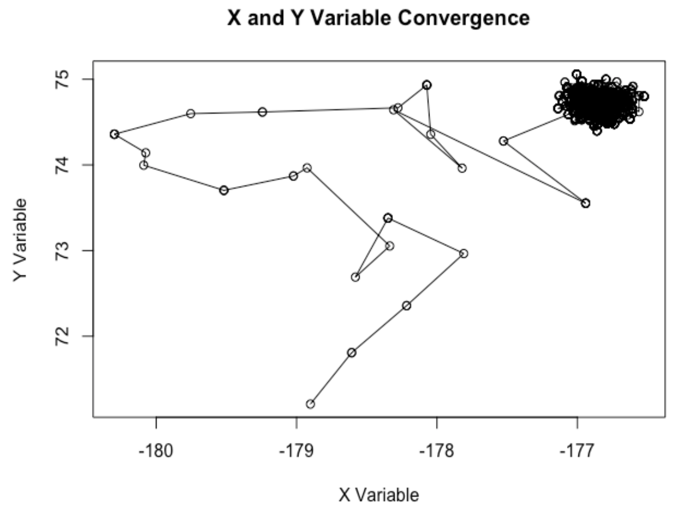
Simulated Annealing:

There are several ways to build into the MCMC M-H algorithm catalysts for a more thorough and efficient exploration of the sample space. Simulated annealing installs a "time schedule" by which a temperature parameter T is defined at time t, with $T_t > 1$. Geman and Geman 1984 proposed a logarithmic scale for decay of temperature: $T_t = \frac{kT_1}{\log(t)}$. In any case, the transition kernel for MCMC is altered such that

$$\pi^*(\theta) = \pi(\theta)^{\frac{1}{T}}$$ **(Equation 16)**

As T approaches +∞, the probability structure of the transition kernel flattens to become a uniform distribution, allowing transition equally likely. As T approaches 1, however, the transition kernel returns to its normal state. The effectiveness simulated annealing in parameter convergence becomes more illustrative through visualization of trace plots in two and three-dimensional spaces. The plot to the left integrates a form of simulated annealing, whereby the distribution "landscape" is flattened in the earlier iterations, allowing for the MCMC algorithm to move more fluidly across the distribution space and avoid becoming stuck in local maximums.





Burn-In Period:

From the autocorrelation plots, it becomes clear that the first iterations of the MCMC-MH algorithm do not achieve stationarity. Thus, they can be safely "discarded" as they are not true approximations of the underlying posterior distribution. For example, the autocorrelation for the large-step MCMC plot reveals that the burn-in period should be at least 35-40 iterations. For small-step MCMC runs, the burn-in period must be significantly longer. Most statistical computing resources for MCMC, including Python's PyMC[vi] and R's mcmc packages include an input parameter to customize burn-in periods.

A major issue, however, with using MCMC –M – H is that the parameter space is fixed; however, when approximating posterior distributions of clusters (k), the dimensionality of the model varies, making each iterative "move" in MCMC non-standard. For instance, a particular Gaussian mixture model with k = 2 may have $\theta_{k=2} = \{\mu_1, \mu_2, \sigma_1, \sigma_2, \Sigma_{1,2}\}$. Suppose, for instance, that an MCMC-M-H algorithm is implemented to model the physical configurations of multiple objects (a three-dimensional space, with x_k, y_k, z_k defining the each object k's state. The number of parameters to approximate each object's state and posterior pdf is 3k, or more generally, mk for an m-dimensional space.

Reversible Jump Markov Chain Monte Carlo

A solution to addressing variable (as opposed to fixed, in the case of classic MCMC) dimensionality is the implementation of reversible jumps, which allows for dimensionality jumps as proposed by Green, 1995.[vii] A core assumption within MCMC is that the chain is reversible- there exists an equilibrium probability $P(\theta_a \rightarrow \theta_b) = P(\theta_b \rightarrow \theta_a)$. In other words, the probability of moving from state θ_a to state θ_b is equivalent to the probability of moving from state θ_b to state θ_a. If the acceptance probability from $\theta_a \rightarrow \theta_b$ is quantified as $\alpha_{forward}(\theta_a, \theta_b)$, then the reverse move's corresponding acceptance probability is $\alpha_{reverse}(\theta_b, \theta_a)$. Thus, the joint distribution integrating candidate models with input data and model parameters is

$$\boldsymbol{p(k, \theta_k, y) = p(k)p(\theta^k|k)p(y|k, \theta^k)} \quad \textbf{(Equation 17)}$$ [viii]

where $p(k)$ is the probability of the given model, $p(\theta^k|k)$ is the prior likelihood (of parameters θ^k given model k), and $p(y|k, \theta^k)$ is the likelihood of the data given current configurations and model space.

Therefore, we can write the equilibrium probability as

$$\int_{(\theta_a,\theta_b)\in A\,x\,B} \boldsymbol{\pi(\theta_a) j_m(\theta_a) g_m(u) \alpha_m(\theta_a,\theta_b) d\theta_a\, du} = \int_{(\theta_b,\theta_a)\in A\,x\,B} \boldsymbol{\pi(\theta_a) j_m(\theta_b) g_m(u) \alpha_m(\theta_b,\theta_a) d\theta_b\, du} \quad \textbf{(Equation 18)}$$

Here, $\theta_{a,}\theta_b$ are the two states that are being transitioned between. $\pi(\theta_a)$ is the prior distribution of states θ where P($\theta = a$). $j_m(\theta_a)$ represents the probability that move m (could be a shift in height, a shift in position, a birth, or a death) is attempted in state θ_a.

The procedure to implement RJMCMC is as follows:

1. Assign the previous configuration of parameters (or initialize parameters) from X_{n-1} as X_n.

   The initialization of parameters can be accomplished using estimation methods such as maximum likelihood estimation. Indeed, even when initialized parameters are poorly chosen, given enough iterations the algorithm will converge on true results.[ix]

2. Sample m from a set of possible moves M using the probability distribution function p_{move}. In other words, choose a move type from a set of possible moves given the point of time (not all moves are available in a given point in time).

3. Apply the chosen move to a target object using a target proposal distribution q_{target} and proposing a new configuration X_n.

The code we implement is fully derived from Ai Jialin's 2012 master's thesis "Reversible-jump MCMC methods in Bayesian statistics" at the University of Leeds, where he implements RJMCMC to reproduce and build upon the results of Green's coal-mining accident analysis for the United Kingdom 1850-1962. Coal mine disasters were prevalent within 19th century UK history, and relatively infrequent further into the 20th century. The question at hand was when exactly did changes in technology, operational safety protocols, and public policy result in statistically significant changes in the rate of accidents per year?

In Green's original implementation of RJMCMC and Jialin's subsequent replication with original code, the relevant dataset is a time-series object consisting of the dates of every occurrence of coal mining disaster incidents. It is therefore a n x m dimensional object, with n = # of disaster incidents, and m = 1. An assumption that both Green and Jialin utilize is that prior probability of each cluster k is defined using the Poisson distribution:

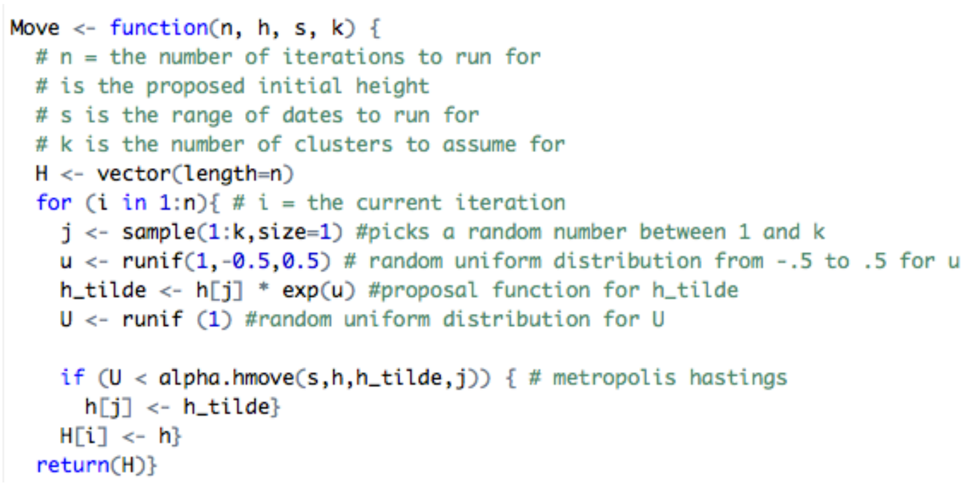
$$\boldsymbol{P(k) = e^{-\lambda} \frac{\lambda^{k-1}}{(k-1)!}} \quad \textbf{(Equation 19)}$$

As a result, the prior distribution builds in a check for model complexity issues stemming from overfitting; the priors are balanced towards low values of k. For instance, $P(k = 2) = e^{-\lambda}\lambda$. However, $P(k = 200) = e^{-\lambda}\frac{\lambda^{199}}{(199)!}$, which approximates 0.

Move Type	Description
Height (η_k)	A vector of heights (intuitively, the frequency of the data) that corresponds to each cluster k. With each new "birth" of addition to k, the new height generated is a geometric mean of the old and new proposed h.
Position (π_k)	A vector of positions that simulate a "piecewise" function of the data. There will always be k + 1 items in the position vector, since when k = 1, the data requires 2 positions to describe the boundaries of the cluster.
Birth (b_{k-1})	The addition of an additional model (k + 1). An additional cluster k implies an addition to the length of both the height and position vectors.
Death (d_{k-1})	The removal of a model from k to k – 1. The acceptance probabilities of birth/death are pseudo-inverses of one another.

Note that $\eta_k + \pi_k + b_{k-1} + d_{k-1} = 1$, and that $d_0 = \pi_1 = 0, b_{k_{max}-1} = 0$. A change in position when k = 1 is impossible, since the positional boundaries must encompass all range of data. For all values k > 1, Jialin sets $\eta_k = \pi_k$ (the probability of a change in height is equal to a probability to a change in position). Moreover, if there exist k + 1 position values for a given k, there must also exist k – 1 change or "switchpoints" between the clusters. As a result, the full parameter set θ^k has a vector length of $n_k = 2k - 1$.

We first initiate the MCMC Metropolis-Hastings algorithm[1]:



```
Move <- function(n, h, s, k) {
  # n = the number of iterations to run for
  # is the proposed initial height
  # s is the range of dates to run for
  # k is the number of clusters to assume for
  H <- vector(length=n)
  for (i in 1:n){ # i = the current iteration
    j <- sample(1:k,size=1) #picks a random number between 1 and k
    u <- runif(1,-0.5,0.5) # random uniform distribution from -.5 to .5 for u
    h_tilde <- h[j] * exp(u) #proposal function for h_tilde
    U <- runif (1) #random uniform distribution for U

    if (U < alpha.hmove(s,h,h_tilde,j)) { # metropolis hastings
      h[j] <- h_tilde}
    H[i] <- h}
  return(H)}
```

Jialin then defines the acceptance probability for a proposed move in height. Note that the proposal distribution is calculated as $\tilde{h} = he^u$ where u is drawn from a random uniform distribution between -0.5 and 0.5. The heights are independently distributed from $\Gamma(\alpha, \beta)$. Green's paper emphasizes that "it is not appropriate to select an improper gamma $\Gamma(0,0)$ for heights, because that causes insurmountable difficulties with normalization across different numbers of steps". Jialin selects $\alpha = 1, \beta = 200/365.24$ since he works with annualized data on coal mine disasters in the United Kingdom.

[1] The code I implement here is completely derived from Jialin's implementation, with minor adaptations and tweaks for the multi-dimensionality of GWAS SNPs datasets.

```r
# value of Gamma(a,b)
a=1
b=200/365.24

alpha.hmove <- function(s,h,h_tilde,j) {
  # s is the range in years
  # h_tilde is the next proposed height
  # j is drawn from a random uniform distribution between 1 and k
  m <- count.disasters(s[j],s[j+1]) #a proposal to move from a model with k = j to k = j + 1

  log_likelihood_ratio <- ((h[j] - h_tilde) * (s[j+1] - s[j])
    + m * (log(h_tilde) - log(h[j])))
  log_prior_ratio <- (a * (log(h_tilde) - log(h[j])) - b * (h_tilde - h[j]))
  log_pi_ratio <- log_likelihood_ratio + log_prior_ratio
  return(exp(log_pi_ratio)) #exp() unpacks the log likelihood into original form}
```

We can integrate switches in position as well, again assuming that k is fixed. Jianlin adopts Green's recommendation to avoid "small steps" whereby s_{j+1}-s_j $\approx$ 0. Because there is the possibility that no data exists in such a small interval, they are not penalized in the likelihood function and thus are often not removed from the posterior distributions for k and s. The acceptance ratio for a proposed move in positions is calculated as follows:

```r
alpha.smove <- function(s,h,s_tilde,j) {
  m1 <- count.disasters(s[j-1],s[j])
  m1_tilde <- count.disasters(s[j-1],s_tilde)
  m2 <- count.disasters(s[j],s[j+1])
  m2_tilde <- count.disasters(s_tilde,s[j+1])
  log_likelihood_ratio <- (
    (h[j] - h[j-1]) * (s_tilde - s[j]) + (m1_tilde - m1) * log(h[j-1])
    + (m2_tilde - m2) * log(h[j])
  )
  log_prior_ratio <- (
    log(s_tilde - s[j-1]) + log(s[j+1] - s_tilde)
    - log(s[j] - s[j-1]) - log(s[j+1] - s[j])
  )
  log_pi_ratio <- log_likelihood_ratio + log_prior_ratio
  return(exp(log_pi_ratio))}
```

We can then integrate position and height moves into a new MCMC-MH algorithm:

```r
Move <- function(n,h_initial,s_initial){
  H <-matrix(NA,nrow=n,ncol=k) #create matrix for height
  S <- matrix(NA,nrow=n,ncol=k+1) # create matrix for position
  h <- h_initial # initialize values for height
  s <- s_initial # initialize values for position
  for (i in 1:n){
    j <- sample(1:(2*k-1),size=1,replace=TRUE) # j is a switch variable that
    # determines whether to sample for height or for position: if j is less than k,
    # then sample for height

    if (j <= k) { #from 1 to k is for Height
      u <- runif(1,-0.5,0.5)
      h_tilde <- h[j] * exp(u) # proposal distribution for height
      U <- runif (1)
      if (U < alpha.hmove(s,h,h_tilde,j)){ #metropolis hastings
        h[j] <- h_tilde}}
    if (j > k) { #from k+1 to 2k-1 is for Position
      j = j - k + 1
      s_tilde <- runif(1,s[j-1],s[j+1])
      U <- runif (1)
      if (U < alpha.smove(s,h,s_tilde,j)){
        s[j] <- s_tilde}}
    H[i,] <- h
    S[i,] <- s}
  return(cbind(H,S))}
```

These examples integrate various dimensions; however, the ultimate number of clusters is fixed at k = K. We can integrate a "birth" of a new cluster with the birth step. The probability of birth and death are defined as

$$\boldsymbol{b_k = c \min\left[1, \frac{p(k+1)}{p(k)}\right]} \qquad \textbf{(Equation 20)}$$

$$\boldsymbol{d_{k+1} = c \min\left[1, \frac{p(k)}{p(k+1)}\right]} \qquad \textbf{(Equation 21)}$$

$p(k+1)$ is the probability of a state existing with an additional cluster, and $p(k)$ is the probability of the current state. Ultimately, $b_k + d_k$ must be < 1, and almost always some number c < 1, since a $b_k + d_k$ sum of 1 would imply that the only moves possible at a given model space would be to add another model or regress to a previous model space.

Jialin and Green draw k from a Poisson distribution with $k < k_{max}$:

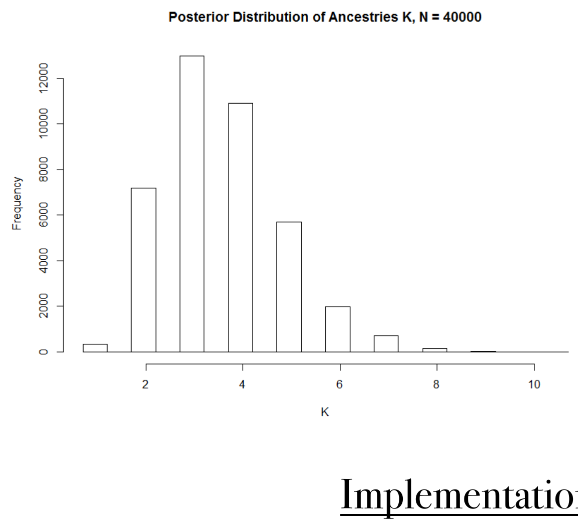
$$\boldsymbol{p(y_1, y_2, \dots y_n|x) = \exp\left(-\sum_{j=1}^{k} h_j(s_j + 1 - s_j) + \sum_{j=1}^{k} m_j \log(h_j)\right)} \qquad \textbf{(Equation 22)}$$

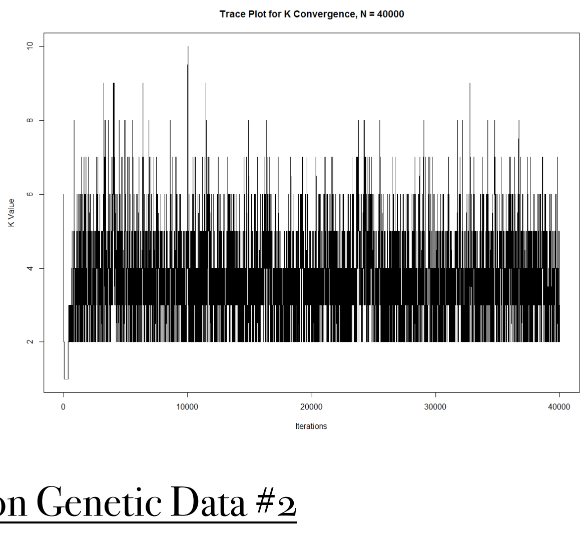
The first expression - $\sum_{j=1}^{k} h_j(s_j + 1 - s_j)$ is a summation of the differences between cluster's boundary positions scaled by the height at cluster j.

Implementation on Genetic Data #1

We load a 5,000 SNP dataset ("mixture1.geno") of 1,500 individuals, and pose the question: are there distinct clusters of SNPS in the dataset? We initialize the RJMCMC implementation with an initial guess of k = 6, and run for 40,000 iterations. The posterior distribution of clusters and trace plot are provided below.

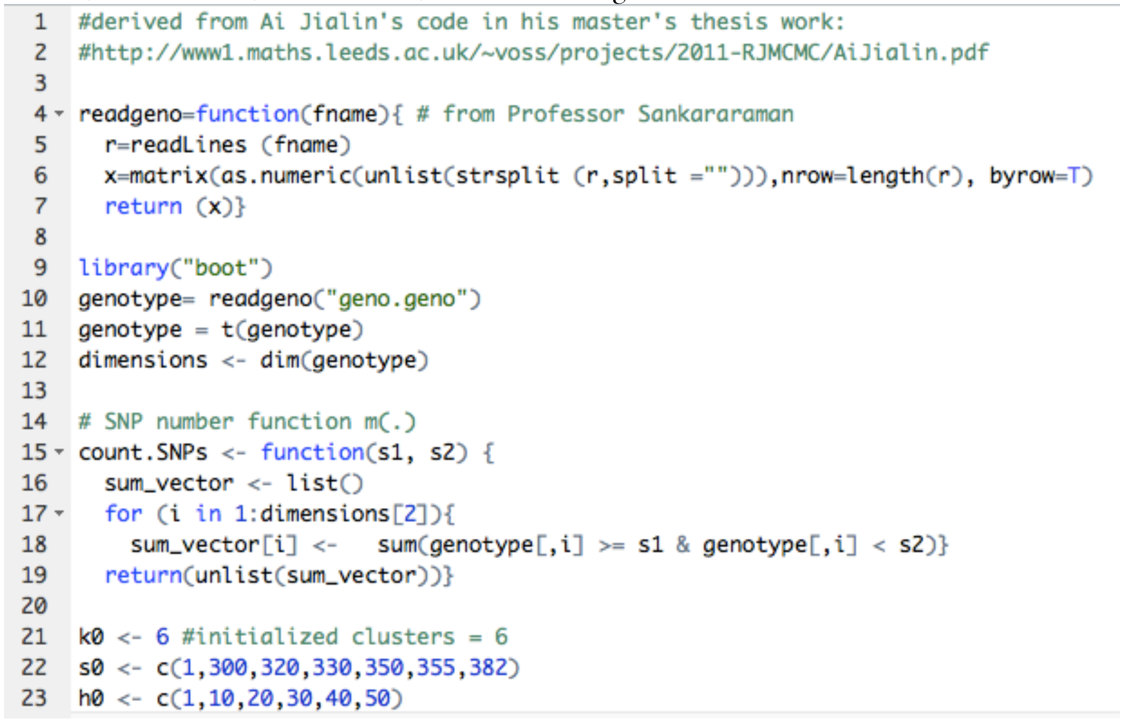
From the data presented, it becomes clear that there are almost certainly distinct clusters of SNPS in the data, and likely between 3-4 groupings. If these 5,000 SNPs all were estimated to come from a single cluster, the posterior distribution mode would fall around 1.





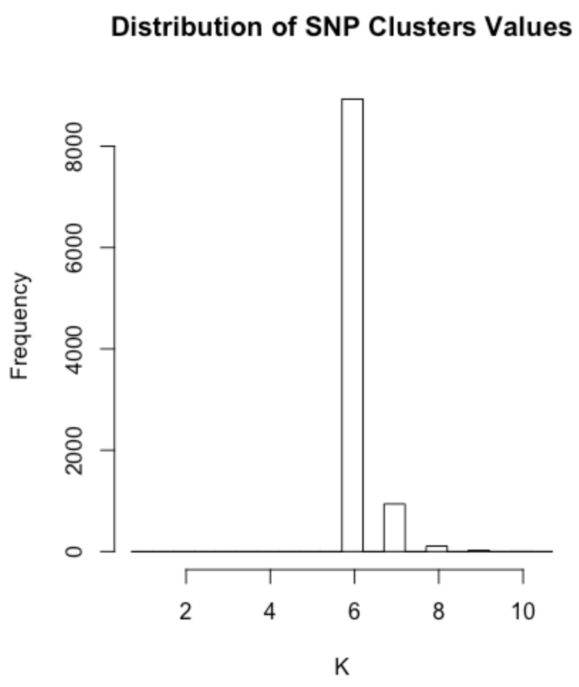
Implementation on Genetic Data #2

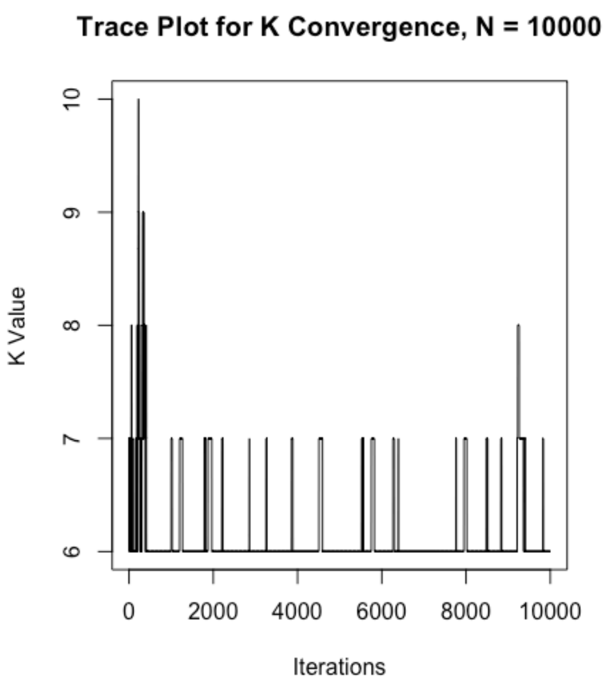
We load a second, smaller dataset ("geno.geno") of 382 SNPs from 500 individuals and once again ask if we can identify any distinct SNP clusters within the data? We initialize the RJMCMC implementation with an initial guess of k = 6, and run for 10,000 iterations, with the following R code:



```r
#derived from Ai Jialin's code in his master's thesis work:
#http://www1.maths.leeds.ac.uk/~voss/projects/2011-RJMCMC/AiJialin.pdf

readgeno=function(fname){ # from Professor Sankararaman
  r=readLines (fname)
  x=matrix(as.numeric(unlist(strsplit (r,split =""))),nrow=length(r), byrow=T)
  return (x)}

library("boot")
genotype= readgeno("geno.geno")
genotype = t(genotype)
dimensions <- dim(genotype)

# SNP number function m(.)
count.SNPs <- function(s1, s2) {
  sum_vector <- list()
  for (i in 1:dimensions[2]){
    sum_vector[i] <-   sum(genotype[,i] >= s1 & genotype[,i] < s2)}
  return(unlist(sum_vector))}

k0 <- 6 #initialized clusters = 6
s0 <- c(1,300,320,330,350,355,382)
h0 <- c(1,10,20,30,40,50)
```

The posterior distribution of clusters and trace plot are provided below.





We can examine further the convergence of positions:

```
[[10000]]
[1]   1.000000   1.000039   1.998772   2.001448 248.593751 310.635161
[7] 382.000000
```

And heights:

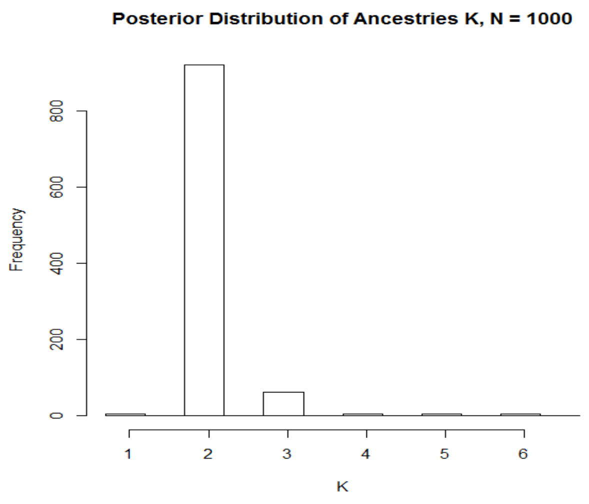
```
[[10000]]
[1] 80041.737622    23.897455 19207.793963     2.010425     2.005461
[6]     2.001661
```

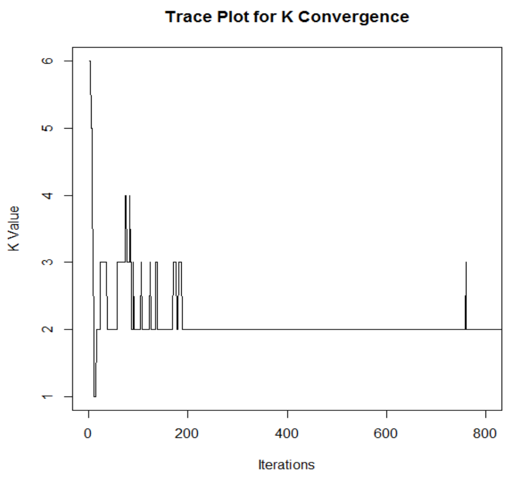
Ultimately, we reach the conclusion that there are *at least* three distinct clusters of SNPs within this dataset, at the following index positions: Cluster 1 (2:249, height 2.010), Cluster 2(250:311, height 2.005), Cluster 3(312:382, height 2.002). The other 3 "clusters" really entirely encapsulated in index position 1, and likely "pseudo-clusters" arising from the inability of the likelihoods to transition to "death steps" when the number of SNPs between an infinitely small position interval ≈ 0.

Final Considerations for GWAS & A Fair Warning on Interpretation of Clustering Results

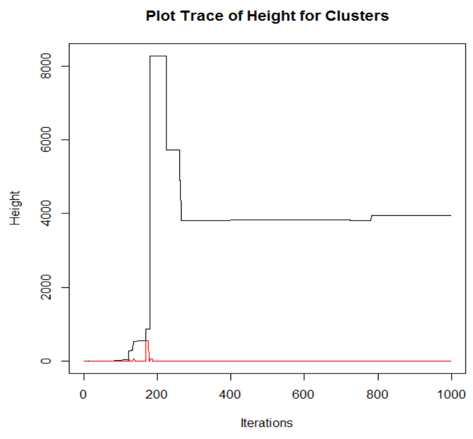
Indeed, it is critical to examine the convergence values for height and position when determining whether or not RJMCMC has produced a valid distribution of potential clusters. In a model that converges to k =2, for instance, an extreme height imbalance (ie. $h_k \approx 0$) suggests that the algorithm has become "stuck" on a minimally small interval with m = 0 (no samples within the position interval). Indeed, it is prudent to check the final distribution of heights and positions; positions extremely close to one another (ie. [0,0.0002, 4201]) represent false clusters.

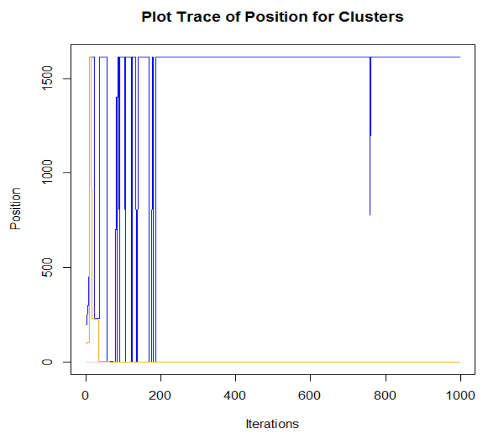
To illustrate this issue, we implemented RJMCMC on a toy dataset provided from Professor Sankararaman for HW #4-5^x, which contains 1614 SNPs rows and 3 columns of distinct population ancestries. We know beforehand that there are three clusters of ancestries, but we do not know if there are any significant cluster of individuals; however, since the RJMCMC algorithm we implement partitions these clusters across a two-dimensional space from 1 to N (with N = 1614), the algorithm should return no significant relationships across the dataset. Intuitively, the manner in which individuals' SNP data are entered into a dataset is assumed to random (in fact, during data import, we add in code to shuffle the rows); thus row 10 should not have any underlying relationship with row 11, row 12, or any other row in question. The code runs to estimate the distribution of likely values for k, the number of populations the SNPs are inherited from. A plot trace and histogram distribution is visualized below, with $k_0 = 6$ (the prior assumption that there are 6 distinct clusters of SNPs here.





The plot trace and posterior distribution appear to strongly k = 2, that there are two distinct cluster of individuals. However, further examination of height and position behavior suggest there is likely one cluster of individuals:





Both the plot trace for height and the plot trace for position show one $p_j \approx 0, h_j \approx 0$ where $j \in k$ (j is one of the final clusters). Since there are no samples that fit into the interval $[p_j, p_{j+1}]$, that particular cluster becomes asymptotically minimized but will never be removed through a death step update. These results imply that across the 1614 individuals, there is most likely any discernible cluster of individuals that better acounts for the data; this is consistent with what we know *a priori*: these individuals' SNP data should not be correlated with the rows before it and rows after it.

When implementing MCMC methods for hyperparameter estimation, it is important to consider the time complexity and scope of the model. MCMC methods can be extremely computational expensive, especially if convergence is slow. Given the size of GWAS datasets, with SNPs sometimes ranging in the hundreds of thousands, it is often well worth the added time to produce informed priors to initialize the MCMC chain.

For instance, the type of distributions by which your P(K = k), or the probability of a particular model *k*, is constructed plays a significant role in final convergence. Indeed, by setting λ values in the Poisson distribution for *k* priors to be high (6 for Implementation 2), we receive different distributions of *k*. In this case, it is important to ask, "What is the underlying nature of the expected clusters from this dataset? Which lambda value best encapsulates prior knowledge?"

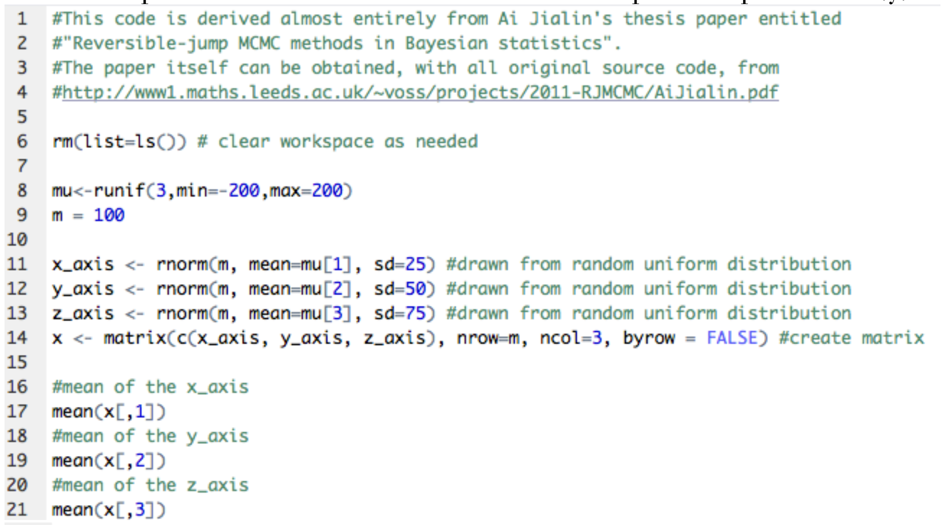
Ultimately, hyperparameter tuning in hierarchical models for GWAS testing has often been a computationally expensive, heuristic process. Hosking et. al[xi] have demonstrated the utility of hidden Markov models (HMMs), logistic regressions, and k-means clustering fitted to a MCMC-M-H algorithm by correctly identifying causal SNPs for phenotypes. However, these various methods for both supervised and unsupervised learning have several core shortcomings:

1) Most of them assume an inherent, underlying distribution behind the data; for instance, both linear models and k-means algorithms incorporate assumptions of normality: both in the signal and in the distribution of errors.
2) Many of the most commonly used models rely upon appropriate selection of hyperparameters. These hyperparameters often are hidden variables that determine other model parameters.

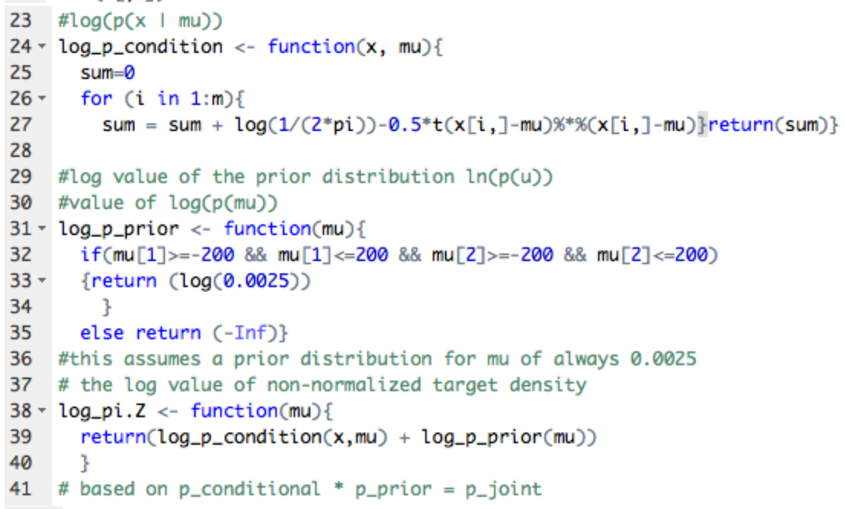
RJMCMC can address several of these core shortcomings by adding or removing models and subsequent model parameters, and is a promising field for active research, as it holds significant promise as a robust, adaptive, stochastic process for hyperparameter estimation.

Appendix

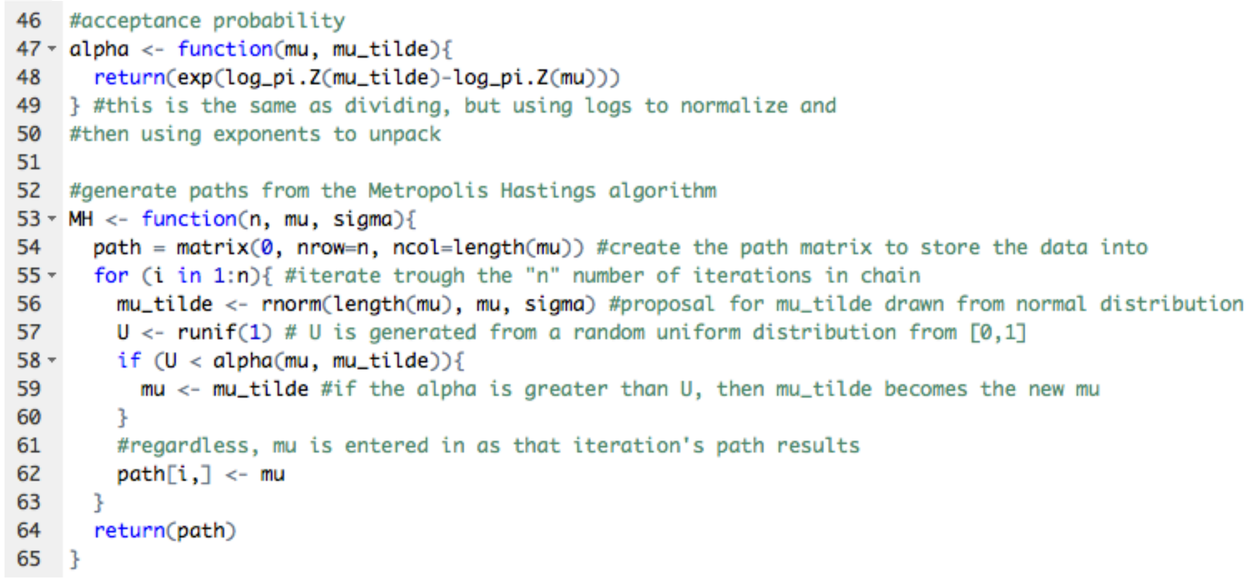
R code to implement MCMC M-H on three-dimensional space with parameters x,y,z:



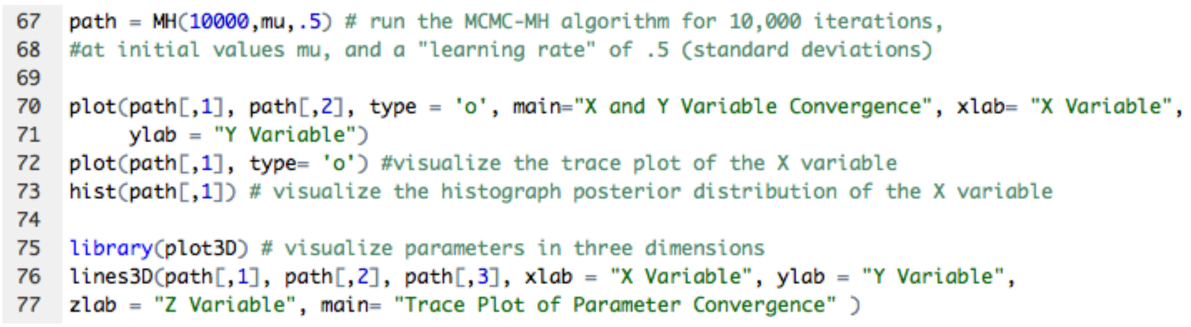
```r
#This code is derived almost entirely from Ai Jialin's thesis paper entitled
#"Reversible-jump MCMC methods in Bayesian statistics".
#The paper itself can be obtained, with all original source code, from
#http://www1.maths.leeds.ac.uk/~voss/projects/2011-RJMCMC/AiJialin.pdf

rm(list=ls()) # clear workspace as needed

mu<-runif(3,min=-200,max=200)
m = 100

x_axis <- rnorm(m, mean=mu[1], sd=25) #drawn from random uniform distribution
y_axis <- rnorm(m, mean=mu[2], sd=50) #drawn from random uniform distribution
z_axis <- rnorm(m, mean=mu[3], sd=75) #drawn from random uniform distribution
x <- matrix(c(x_axis, y_axis, z_axis), nrow=m, ncol=3, byrow = FALSE) #create matrix

#mean of the x_axis
mean(x[,1])
#mean of the y_axis
mean(x[,2])
#mean of the z_axis
mean(x[,3])
```



```r
#log(p(x | mu))
log_p_condition <- function(x, mu){
  sum=0
  for (i in 1:m){
    sum = sum + log(1/(2*pi))-0.5*t(x[i,]-mu)%*%(x[i,]-mu)}return(sum)}

#log value of the prior distribution ln(p(u))
#value of log(p(mu))
log_p_prior <- function(mu){
  if(mu[1]>=-200 && mu[1]<=200 && mu[2]>=-200 && mu[2]<=200)
  {return (log(0.0025))
    }
  else return (-Inf)}
#this assumes a prior distribution for mu of always 0.0025
# the log value of non-normalized target density
log_pi.Z <- function(mu){
  return(log_p_condition(x,mu) + log_p_prior(mu))
  }
# based on p_conditional * p_prior = p_joint
```

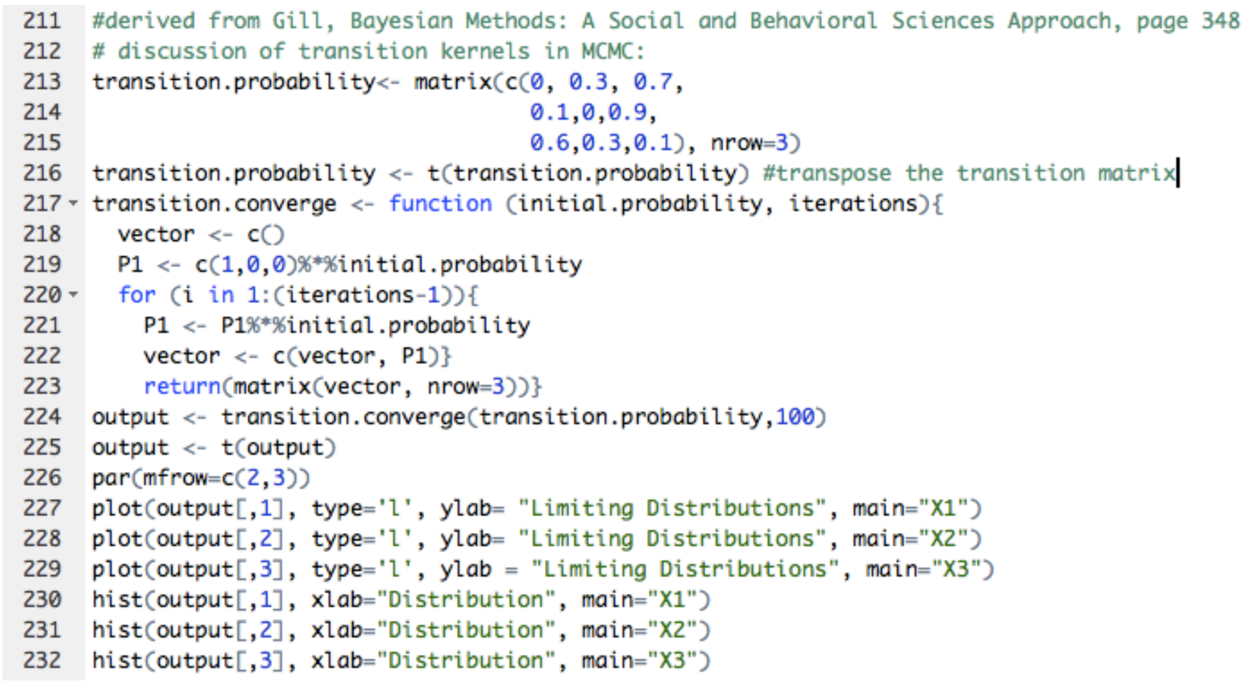


```r
#acceptance probability
alpha <- function(mu, mu_tilde){
  return(exp(log_pi.Z(mu_tilde)-log_pi.Z(mu)))
} #this is the same as dividing, but using logs to normalize and
#then using exponents to unpack

#generate paths from the Metropolis Hastings algorithm
MH <- function(n, mu, sigma){
  path = matrix(0, nrow=n, ncol=length(mu)) #create the path matrix to store the data into
  for (i in 1:n){ #iterate trough the "n" number of iterations in chain
    mu_tilde <- rnorm(length(mu), mu, sigma) #proposal for mu_tilde drawn from normal distribution
    U <- runif(1) # U is generated from a random uniform distribution from [0,1]
    if (U < alpha(mu, mu_tilde)){
      mu <- mu_tilde #if the alpha is greater than U, then mu_tilde becomes the new mu
    }
    #regardless, mu is entered in as that iteration's path results
    path[i,] <- mu
  }
  return(path)
}
```



```r
path = MH(10000,mu,.5) # run the MCMC-MH algorithm for 10,000 iterations,
#at initial values mu, and a "learning rate" of .5 (standard deviations)

plot(path[,1], path[,2], type = 'o', main="X and Y Variable Convergence", xlab= "X Variable",
     ylab = "Y Variable")
plot(path[,1], type= 'o') #visualize the trace plot of the X variable
hist(path[,1]) # visualize the histograph posterior distribution of the X variable

library(plot3D) # visualize parameters in three dimensions
lines3D(path[,1], path[,2], path[,3], xlab = "X Variable", ylab = "Y Variable",
zlab = "Z Variable", main= "Trace Plot of Parameter Convergence" )
```

R code to implement a convergence to a limiting distribution:



```r
#derived from Gill, Bayesian Methods: A Social and Behavioral Sciences Approach, page 348
# discussion of transition kernels in MCMC:
transition.probability<- matrix(c(0, 0.3, 0.7,
                                  0.1,0,0.9,
                                  0.6,0.3,0.1), nrow=3)
transition.probability <- t(transition.probability) #transpose the transition matrix
transition.converge <- function (initial.probability, iterations){
  vector <- c()
  P1 <- c(1,0,0)%*%initial.probability
  for (i in 1:(iterations-1)){
    P1 <- P1%*%initial.probability
    vector <- c(vector, P1)}
    return(matrix(vector, nrow=3))}
output <- transition.converge(transition.probability,100)
output <- t(output)
par(mfrow=c(2,3))
plot(output[,1], type='l', ylab= "Limiting Distributions", main="X1")
plot(output[,2], type='l', ylab= "Limiting Distributions", main="X2")
plot(output[,3], type='l', ylab = "Limiting Distributions", main="X3")
hist(output[,1], xlab="Distribution", main="X1")
hist(output[,2], xlab="Distribution", main="X2")
hist(output[,3], xlab="Distribution", main="X3")
```

Bibliography

[i] StackExchange CrossValidated. "How to choose the number of hidden layers and nodes in a feedforward neural network?". 2010, July 20th. Link.

[ii] Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI & Institute for Computer Science, University of Hildesheim Course on Machine Learning, winter term 2009/2010, Link.

[iii] Vera Lisovskaja, "TMS150, Stochastic data processing and simulation, 2011/12: Chapter 6 Monte Carlo", Link.

[iv] Based upon notes from Mark Holmes and Jesse Goodman, STATS 325 course notes Chapter 9, University of Auckland, Link.

[v] Jeff Gill, *Bayesian Methods: A Social and Behavioral Sciences Approach*, Chapman & Hall: New York, 2008.

[vi] PyMC, Link.

[vii] Peter J. Green, "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination", Department of Mathematics, University of Bristol, Link.

[viii] Ibid.

[ix] Brian Hartman and Jeffrey Hart, "Using Reversible Jump MCMC to Account for Model Uncertainty", Texas A&M University.

[x] Available freely from Sriram Sankararaman, UCLA Fall CM 226 Machine Learning in Bioinformatics class website, Link.

[xi] Fay J. Hosking et. al, "Inference From Genome-Wide Association Studies Using a Novel Markov Model", University of Bristol Department of Mathematics and Department of Social Medicine, Genetic Epidemiology (2008), Link.