An Adaptive Markov Chain Monte Carlo Algorithm-Regime Change Algorithm

Kai Yang*
Department of Statistics, University of Toronto

Supported by University of Toronto Excellence Award (UTEA-NSE)

Supervisors: Jeffrey S. Rosenthal & Radu Craiu

August 14, 2011

*Email: kai.b.yang@utoronto.ca

Contents

1 Introduction **4**

1.1 The Metropolis-Hastings Algorithm 5

1.2 The Proposed Regime Change Algorithm 5

1.3 Background on Parallel Tempering MCMC 6

1.4 Independent Metropolis Sampler 7

2 The Expectation-Maximization (EM) Algorithm **7**

2.1 The General EM Algorithm 7

2.2 The EM Algorithm for the Gaussian Mixture Model 8

2.3 Simulation Studies: Regime Change Algorithm via EM 9

2.3.1 A One Dimensional Target Distribution 10

2.3.2 A Two Dimensional Target Distribution 11

2.3.3 A Ten Dimensional Target Distribution 11

2.4 Draw Backs of the EM Algorithm 13

3 Dirichlet Process Mixture (DPM) Models **14**

3.1 An Introduction to the DPM Models 15

3.2 Proposed Algorithm: Sequential Updating and Greedy Search (SUGS) 17

3.3 Normal Mixture and Conjugate Prior 18

3.3.1 The Univariate Case 19

3.3.2 The Multivariate Case 22

3.4 Simulation Studies: Regime Change Algorithm via DPM 24

3.4.1 A Two Dimensional Target Distribution 24

3.4.2 A Ten Dimensional Target Distribution 26

3.5 Inference at a Higher Level - an Empirical Bayesian Approach 28

3.5.1 Bayesian Inference for Prior DP Precision Parameter α 29

3.5.2 Empirical Bayesian Inference for Other Hyperparameters 30

3.5.3 The Maximum Likelihood Estimator Approach 30

4 Discussion **32**

Appendices **34**

A R Code for Parallel Tempering and IM Sampler **34**

B R Code of EM Algorithm for Gaussian Mixture Model **37**

C R Code of DPM for Gaussian Mixture Model **39**

D R Code of Bayesian Inference for Prior DP Precision Parameter **45**

References **46**

Adaptive MCMC algorithms have been one of the most active areas of research in recent years. In this paper we study one kind of adaptive MCMC algorithm called Regime Change Algorithm (RCA). We will start by an introduction to RCA in chapter 1. In chapter 2, we give an account for the EM algorithm and discuss its performance in conjunction with RCA algorithms. In chapter 3, we explore the possibility of density estimation via Dirichlet Process Mixture Models, which is an alternative to the EM algorithm. Its performance with the RCA is also studied in that chapter. Chapter 4 ends this survey with some discussions and future research directions.

1 Introduction

The class of adaptive MCMC (AMCMC) has gained attraction in recent years. The idea behind AMCMC is to simultaneously run the sampling chain and use the samples obtained to continuously modify the transition kernel used in the algorithm. Since the transition kernel depends on samples from the whole past, the Markovian property of the chain is lost. To ensure ergodicity of the Markov chain $\{X_n\}$ (i.e. $P(X_n \in A) \to \pi(A)$ where π is the invariant distribution), the adaptive MCMC algorithm has to satisfy Diminishing Adaptation and Containment conditions (Roberts and Rosenthal, 2007, Theorem 13). All the AMCMC samplers proposed so far in the literature assume that the same kind of transition kernel is used throughout the simulation. However, advantages can be obtained if we allow for regime changes during the simulation process, which gives rise to the proposed Regime Change Algorithm. For instance, it is known that one is well-advised to consider a random walk Metropolis (RWM) in situations in which little is known about the geography of the target distribution. But independent Metropolis (IM) algorithms, when well-tuned, can have a superior performance compared to RWM algorithms (Mengersen and Tweedie, 1996), giving samples with much smaller dependence structure much more quickly. To account for these possible benefits, we propose a generic MCMC algorithm in which one starts the initialization period using a Random Walk Metropolis sampler that slowly gives way to an independent Metropolis sampler. The rate of regime change must be coordinated to match the closeness between target density π and the proposal density used in the adaptive IM. A good candidate that serves this purpose is the IM sampler acceptance rate. Also since a good approximation of π cannot generally be obtained on the whole state space, we consider adapting only inside a compact subset, $\mathcal{K}$, of the state space, $\mathcal{S}$, while outside $\mathcal{K}$ we use a transition kernel that remains fixed throughout the simulation.

1.1 The Metropolis-Hastings Algorithm

The Metropolis-Hastings algorithm (Metropolis et al., 1953; Hastings, 1970) is a Markov chain Monte Carlo method for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult. Suppose we wish to obtain samples from our target distribution π. Then given X_n, the state of the Markov chain at time n, a proposed value Y_{n+1} is generated from some pre-specified proposal density $q(X_n, y)$, and is then accepted with probability $\alpha(x, y)$ where

$$\alpha(x,y) = \begin{cases} \min\left\{\frac{\pi(y)}{\pi(x)}\frac{q(y,x)}{q(x,y)}, 1\right\} & \text{if } \pi(x)q(x,y) \neq 0, \\ 1 & \text{if } \pi(x)q(x,y) = 0. \end{cases}$$

If the proposed value is accepted, we set $X_{n+1} = Y_{n+1}$; otherwise, we set $X_{n+1} = X_n$. We can prove such choice of acceptance probability $\alpha(x, y)$ ensures that the Markov chain $\{X_n\}$ is reversible with respect to the target density $\pi(x)$, so that the target density is stationary for the chain (Roberts and Rosenthal, 2004, Proposition 1). Under ergodicity condition, the Markov chain will sample approximately from the target distribution after a certain number of iterations.

1.2 The Proposed Regime Change Algorithm

Let $\tilde{P}$ be the transition kernel of the Regime Change Algorithm (RCA), we have

$$\tilde{P}_\gamma(x, A) = 1_{\mathcal{K}}(x)[\lambda_\gamma P_\gamma(x, A) + (1 - \lambda_\gamma)Q_\gamma(x, A)] + 1_{\mathcal{K}^c}(x)R(x, A)$$

γ is the adaptation parameter, P and Q are transition kernels of RWM and IM respectively, and R is the fixed transition kernel used on $\mathcal{K}^c = \mathcal{S} - \mathcal{K}$. Ideally, we would like to have $\lambda_\gamma \to 0$ as the proposal distribution used for Q_γ, q_γ, approaches π on $\mathcal{K}$. A possible choice from our simulation results for λ_γ is the rejection rate for the IM sampler since it is very easy to estimate by proposing using the IM sampler and it is also a very good indicator of the closeness between the target density and the proposal density.

We first go through an initialization process where we run the RWM algorithm only. Then we fit a Gaussian mixture model using the EM algorithm to the obtained samples after discarding the burn-in samples, which will serve as the proposal distribution of the IM sampler later. We make λ_γ the rejection rate of the IM sampler which we compute as we go. We also refine the Gaussian mixture models as we get more and more samples during the simulation process. Since with enough number of components, a mixture of normal distributions can theoretically approximate any target distribution arbitrarily well,

we can expect that when we have more and more samples, the proposal distribution is going to resemble the target distribution better and better where we can have $\lambda_\gamma \to 0$ as our simulation process goes on.

1.3 Background on Parallel Tempering MCMC

It is well-known that the efficiency of the adaptive algorithm depends significantly on the quality of the initialization samples (e.g., Giordani and Kohn, 2010). Usual RWM algorithm does not work well for the multi-dimensional multi-modal distributions, which are natural objects we study in MCMC. A remedy for the quality of initial samples is the parallel tempering MCMC (or the Metropolis-Coupled MCMC). So instead of using standard RMW sampler on the space $\mathcal{K}$ to explore the target density, which could get trapped in one of the local modes, we use parallel tempering MCMC for the transition kernel P.

Suppose we run m parallel tempering MCMC, one RWM chain for each temperature τ. The m chains have m different target density, which is flatter when the temperature τ is higher. When the target density is flatter, it is much easier for the Random Walk Metropolis chain to go between different modes. So the idea of parallel tempering is to have the higher temperature chains to drive the mixing of the lower temperature chains. More specifically, chain τ has target density $\pi_\tau(x) = c_\tau(\pi(x))^{1/\tau}$ for $\tau = 1, 2, \ldots, m$. We couple the m chains together to create a new Markov chain where the state at time n is $X_n = (X_{n,1}, X_{n,2}, \ldots, X_{n,m})$, and $\{X_{n,\tau}\}$ is the chain at temperature τ. The state space of the parallel tempering MCMC is $\mathcal{K}^m$ where the stationary distribution is $\bar{\pi} = \pi_1 \times \pi_2 \times \ldots \times \pi_m$, i.e. $\bar{\pi}(X_1 \in S_1, X_2 \in S_2, \ldots, X_m \in S_m) = \pi_1(S_1)\pi_2(S_2)\ldots\pi_m(S_m)$. We sequentially update the chain at temperature τ (for each $1 \le \tau \le m$), by proposing $Y_{n,\tau} \sim N(X_{n-1,\tau}, \sigma^2)$, where σ^2 is some proposal scaling for the RWM, and accepting with probability $min\Big(1, \frac{\pi_\tau(Y_{n,\tau})}{\pi_\tau(X_{n-1,\tau})}\Big)$. Then we choose temperatures τ and τ' at random, and propose to swap the values $X_{n,\tau}$ and $X_{n,\tau'}$, accepting this with probability $\min\Big(1, \frac{\pi_\tau(X_{n,\tau'})\pi_{\tau'}(X_{n,\tau})}{\pi_{\tau'}(X_{n,\tau'})\pi_\tau(X_{n,\tau})}\Big)$. Some simple algebra shows us:

$$min\left(1, \frac{\pi_\tau(X_{n,\tau'})\pi_{\tau'}(X_{n,\tau})}{\pi_\tau(X_{n,\tau})\pi_{\tau'}(X_{n,\tau'})}\right) = min\left(1, \frac{c_\tau\pi(X_{n,\tau'})^{1/\tau}c_{\tau'}\pi(X_{n,\tau})^{1/\tau'}}{c_\tau\pi(X_{n,\tau})^{1/\tau}c_{\tau'}\pi(X_{n,\tau'})^{1/\tau'}}\right) = min\left(1, \frac{\pi(X_{n,\tau'})^{1/\tau}\pi(X_{n,\tau})^{1/\tau'}}{\pi(X_{n,\tau})^{1/\tau}\pi(X_{n,\tau'})^{1/\tau'}}\right)$$

Hence the normalizing constant is not necessary in the computation for acceptance probability. It is not hard to show such construction makes the chain $\{X_n\}$ ergodic with invariant distribution $\bar{\pi}$. Then if we only look at the first component of the chain, we obtain samples from the desired target distribution π.

1.4 Independent Metropolis Sampler

After the initialization process, by some standard density estimation method, we obtain a Gaussian mixture density q_γ that is close to the target density, which we will continue to refine as the simulation process goes on. q_γ serves as the proposal distribution of the IM sampler for the RCA. In other words, at time n, we propose Y_n i.i.d. from q_γ (normal mixture distributions are easy to simulate in R), with acceptance probability:

$$min\left(1, \frac{\pi(Y_n)q_\gamma(X_{n-1})}{\pi(X_{n-1})q_\gamma(Y_n)}\right)$$

This is essentially a special case of the Metropolis-Hastings algorithm, from which the proof of ergodicity of the IM sampler follows. We estimate the rejection rate by $r = \frac{REJ}{PRO}$. where REJ is the number of rejections and PRO is the total number of proposals. We take the adaptation parameter, λ_γ in section 1.1 to be r for the following reason: Suppose the proposal distribution is the same as the target distribution. Then all proposals will be accepted, with rejection rate 0. With such proposal density, we should completely switch to the IM sampler. However, if the rejection rate is not 0, we still need parallel tempering chain to explore the target distribution, where a new mode might appear. The IM sampler rejection rate is an indicator of closeness between the target density and proposal density. When the rejection rate is high, the RCA consists of more parallel tempering MCMC to discover possible new modes. When the rejection rate is low, the RCA consists of more IM sampler, which gives samples of smaller dependence structure.

R code for parallel tempering and IM sampler is provided in the appendix section.

2 The Expectation-Maximization (EM) Algorithm

A substantial part of RCA is to estimate the target density with a Gaussian mixture model using samples obtained in the past. Such density estimation has to be reliable and reasonably fast for the algorithm to work. The EM algorithm is a common method people use to obtain maximum likelihood estimators for the Gaussian mixture models. In this section, we introduce the EM algorithm and discuss its performance with RCA.

2.1 The General EM Algorithm

Consider a model for the observed data x, which is a matrix of n independent observations of dimension d that is accompanied by a vector of unobserved latent variables z of n components. A model with parameters θ describes the joint distribution of x and z, as $P(x, z|\theta)$.

We want to estimate model parameters θ by maximum likelihood, which means finding the θ that maximizes

$$P(x|\theta) = \int P(x, z|\theta)dz$$

suppose that we can easily find the θ that maximizes $P(x, z|\theta)$, for any known x and z. Then we can use this capability in an iterative algorithm called the EM algorithm for maximizing $P(x|\theta)$. The general EM algorithm alternates these steps:

E Step: Using the current value of the parameter, θ, find the distribution, Q, for the latent variables z, given the observed x:

$$Q(z) = P(z|x, \theta)$$

M Step: Maximize the expected value of log $P(x, z|\theta)$ with respect to θ, where the expectation is with respect to the distribution Q found in the E step:

$$\theta = argmax_\theta E_Q[logP(x, z|\theta)]$$

To see why this iterative algorithm can maximize the log likelihood of the data set at least locally, consider the following function F of the distribution Q over the latent variables z and the parameters θ:

$$\begin{aligned} F(Q, \theta) &= E_Q[logP(x, z|\theta)] - E_Q[logQ(z)] \\ &= logP(x|\theta) + E_Q[logP(z|x, \theta)] - E_Q[logQ(z)] \\ &= logP(x|\theta) - E_Q[log(Q(z)/P(z|x, \theta))] \end{aligned}$$

The final term above is the Kullback-Leibler (KL) divergence between the distribution $Q(z)$ and the distribution $P(z|x, \theta)$. One can show that this divergence is always non-negative, and is zero only when $Q(z) = P(z|x, \theta)$. The E step maximizes $F(Q, \theta)$ with respect to Q, which is a consequence of KL divergence being minimized when $Q(z) = P(z|x, \theta)$. The M step maximizes $F(Q, \theta)$ with respect to θ since $E_Q[logQ(z)]$ does not depend on θ. The maximum of $F(Q, \theta)$ occurs at a θ that maximizes $P(x|\theta)$ because if instead $P(x|\theta^*) > P(x|\theta)$ for some θ^*, then $F(Q^*, \theta^*) > F(Q, \theta)$ with $Q^*(z) = P(z|x, \theta^*)$.

2.2 The EM Algorithm for the Gaussian Mixture Model

In the case where the model is a mixture of Gaussian distributions. The latent variables are discrete, specifying which component of the mixture model the data point is coming from.

Hence distribution Q can be completely specified by what is called responsibilities in machine learning language. The responsibilities are the conditional probability of the observed sample from each of the K components of the normal mixture model, conditioning on the modal parameters θ. In the Gaussian Mixture Model case, $\theta = (\tilde{\pi}, \tilde{\mu}, \tilde{\Sigma})$ is a triple where $\tilde{\pi}$ is the mixing proportions of the mixture, $\tilde{\mu}$ is a list of mean vectors of each of the multidimensional Gaussian component, and $\tilde{\Sigma}$ is a list of covariance matrices of the components. Some simple algebra shows that the general EM algorithm described in section 2.1 can be greatly simplified in the context of Gaussian mixture model, which maximizes the log likelihood of the data set as proven in the previous section. More specifically, the algorithm alternates in between the E steps and the M steps as follows:

E Step: Using the current values of the parameters, compute the responsibilities of components for data items, by applying Bayes' Rule:

$$r_{ik} = P(\text{data point i come from component k } | x_i) = \frac{\pi_k N(x_i|\mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} N(x_i|\mu_{k'}, \Sigma_{k'})}$$

M Step: Using the current responsibilities, re-estimate the parameters, using weighted averages, with weights given by the responsibilities:

$$\pi_k = \frac{1}{n} \sum_i r_{ik}$$

$$\mu_k = \frac{\sum_i x_i r_{ik}}{\sum_i r_{ik}}$$

$$\Sigma_k = \frac{\sum_i r_{ik}(x_i - \mu_k)(x_i - \mu_k)^T}{\sum_i r_{ik}}$$

We start with some initial guess at the parameter values (perhaps random), or perhaps with some initial guess at the responsibilities (in which case we start with an M step). We continue alternating E and M steps until there is little change. See R code in the appendix section of this paper.

2.3 Simulation Studies: Regime Change Algorithm via EM

We first go through an initialization process where we run the parallel tempering chain only. Then we fit a Gaussian mixture model using the EM algorithm to the obtained samples after discarding the burn-in samples, which will serve as the proposal distribution of the IM sampler. We make λ_γ the rejection rate of the IM sampler which we compute as we go. RCA ensures that λ_γ portion of the sample comes from parallel tempering chain and the

rest comes from the IM sampler. We also keep refining the Gaussian mixture models as we get more and more samples during the simulation process. R code for the simulation is in the appendix section.

2.3.1 A One Dimensional Target Distribution

Consider the one dimensional target density, $\pi(x) = \frac{1}{4}N(x| - 3.1, 1.5^2) + \frac{3}{4}N(x|10.2, 1.7^2)$. By the RCA algorithm, we first run 5 parallel tempering chains for 5500 iterations, with burn-in sample of size 500. Then we fit a Gaussian Mixture Model with 2 components to the obtained samples, which will serve as the proposal distribution of the IM sampler. One issue with the EM algorithm is the need to pre-determine the number of components to fit to the data set. In this example we fit two components since we know the number of modes of the target distribution. In general, this is a more difficult problem which motivates the discussion of Dirichlet Process Mixture (DPM) models in the next section.

We estimate the target density using EM algorithm with the initial 5000 samples from the parallel tempering RWM without burn-in samples. See figure 1.

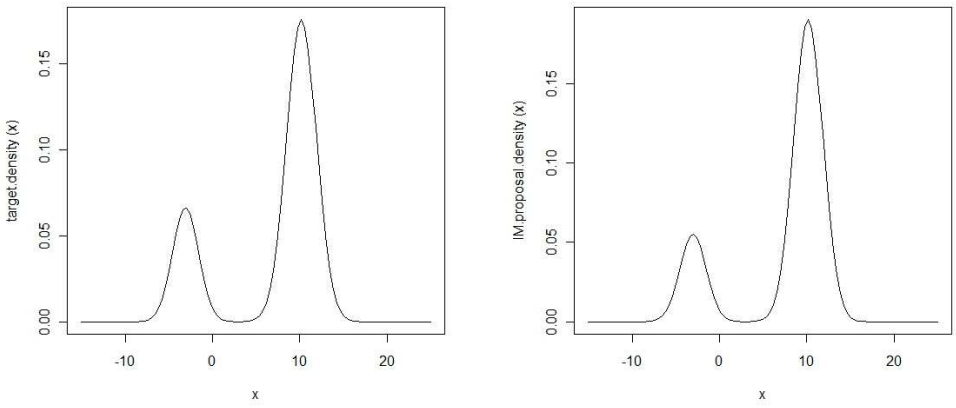

Figure 1: Left is a plot of the target density. Right is a plot of the proposal density used by the IM sampler via EM algorithm.

We use the fitted proposal distribution to obtain another 5000 samples from the IM sampler and compare the samples. See figure 2.

A few advantages of RCA are clear from the simulation results. When well tuned, IM sampler gives samples of much smaller dependence structure, in this case only 2-dependent samples. Also a well tuned IM sampler is fast and reliable. Such well-estimated proposal density gives acceptance rate of 0.9414 for the IM sampler. The quality of the well tuned samples is shown by the acf plots and the trace plots in figure 3.

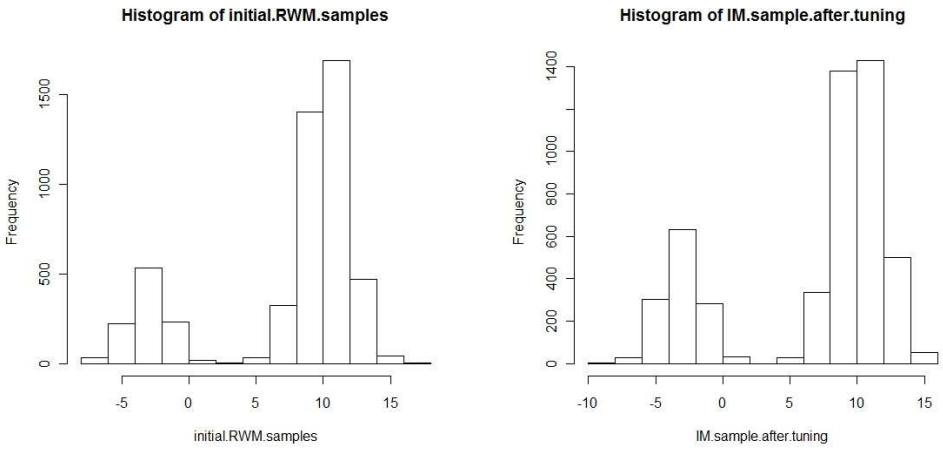

Figure 2: Left is the histogram of the initial samples obtained by the parallel tempering RWM. Right is the histogram of the IM samples.

2.3.2 A Two Dimensional Target Distribution

Let us consider the following target distribution:

$$\pi(x) = \frac{5}{16}N\Big((5,7), \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}\Big) + \frac{5}{16}N\Big((-5,-1), \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}\Big) + \frac{1}{8}N\Big((-1,9), \begin{bmatrix} 0.5 & 1.2 \\ 1.2 & 4.5 \end{bmatrix}\Big) + \frac{1}{4}N\Big((1,2), \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\Big)$$

We go through an initialization process to obtain a sample of size 5000. Then we use EM algorithm to fit a Gaussian Mixture Model (of 4 components) to the obtained sample. The fitted IM proposal is shown in Figure 4.

Then we use the fitted proposal distribution to obtain another 5000 samples from the IM sampler and compare the samples, acf's and trace plots. (See figure $5-7$.) The acceptance rate of IM sampler is 0.913, giving almost i.i.d. samples from the target distribution.

2.3.3 A Ten Dimensional Target Distribution

As an illustration, we consider sampling from a ten-dimensional normal mixture model $\pi(x)$ that has six modes.

$$\pi(x) = \sum_{i=1}^{6} p_i N(x|\mu_i, \Sigma_i)$$

where $(p_1, p_2, \ldots, p_6) = (0.21, 0.20, 0.08, 0.18, 0.08, 0.25)$
$\mu_1 = (4.39, -13.92, -5.52, 19.27, 16.45, -1.71, 8.85, -2.05, 10.51, -7.96)^T$
$\mu_2 = (-14.54, 15.54, 17.70, -17.00, 4.84, 16.03, -11.71, -16.07, 13.03, -15.27)^T$
$\mu_3 = (2.45, 17.01, 10.14, 14.57, 8.47, 6.12, 6.50, 2.47, -8.08, -8.28)^T$

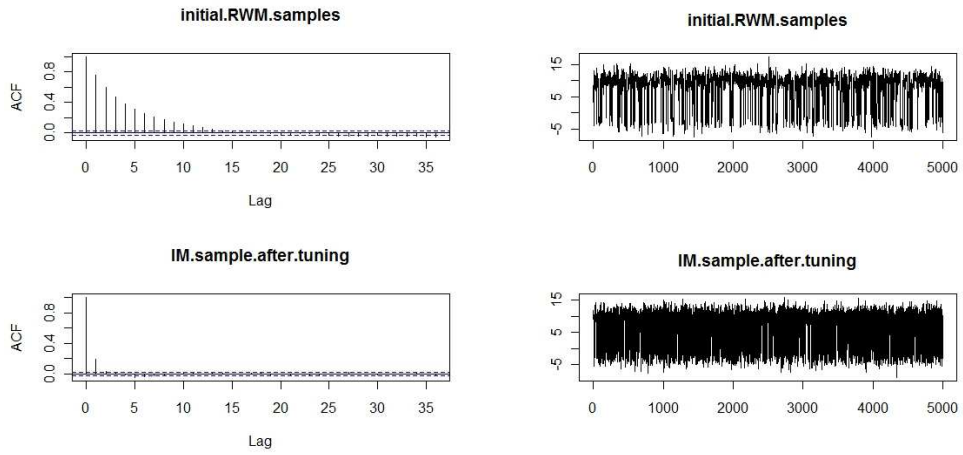

Figure 3: Left is a comparison of the acf's between initial samples and the well-tuned IM samples. Right is a comparison of the trace plots.

$\mu_4 = (13.75, 14.73, -14.87, -12.54, -1.52, -5.16, 8.68, -4.70, 2.19, -0.52)^T$
$\mu_5 = (7.52, -8.98, 16.65, 11.32, 19.41, -7.43, 2.15, 10.52, -2.98, -5.46)^T$
$\mu_6 = (-11.30, -15.25, -14.66, 19.95, -17.58, -1.60, 17.78, 4.22, -14.18)^T$
Covariance matrices are diagonal matrices:
$\Sigma_1 = diag(3.09, 5.28, 7.75, 7.84, 4.34, 7.05, 6.69, 2.67, 8.48, 5.94)$
$\Sigma_2 = diag(5.29, 7.17, 1.35, 3.26, 5.59, 1.25, 8.73, 5.13, 9.37, 3.53)$
$\Sigma_3 = diag(7.97, 1.34, 8.54, 4.10, 6.19, 9.82, 7.18, 6.82, 9.62, 2.75)$
$\Sigma_4 = diag(7.48, 9.08, 1.91, 5.95, 4.03, 2.14, 3.38, 6.64, 4.72, 1.87)$
$\Sigma_5 = diag(3.36, 6.70, 8.04, 3.04, 1.42, 7.98, 1.40, 7.96, 9.29, 6.58)$
$\Sigma_6 = diag(3.48, 5.19, 2.62, 1.81, 1.40, 2.75, 7.82, 5.23, 1.50, 7.42)$

We use the built-in function in R to obtain an i.i.d. sample of size 10000. Compare the i.i.d. samples with the initial 10000 samples obtained from parallel tempering chains in nine 2 dimensional projections. (See figure 8.)

Fitting the 30000 initial parallel tempering samples to a Gaussian mixture model with 6 components. Then we used the fitted Gaussian mixture model as the proposal distribution for the IM sampler to obtain a sample of size 10000 from the IM sampler. The IM sampler acceptance rate is 0.6507. Also as we obtain more and more samples, the fitted Gaussian mixture model will be closer and closer to the target density, giving higher and higher IM acceptance rate. See figure 9 for a comparison between the 10000 samples obtained from the parallel tempering and from the tuned IM sampler. The acf's and trace plots of these two samples are given in figure 10 and 11.

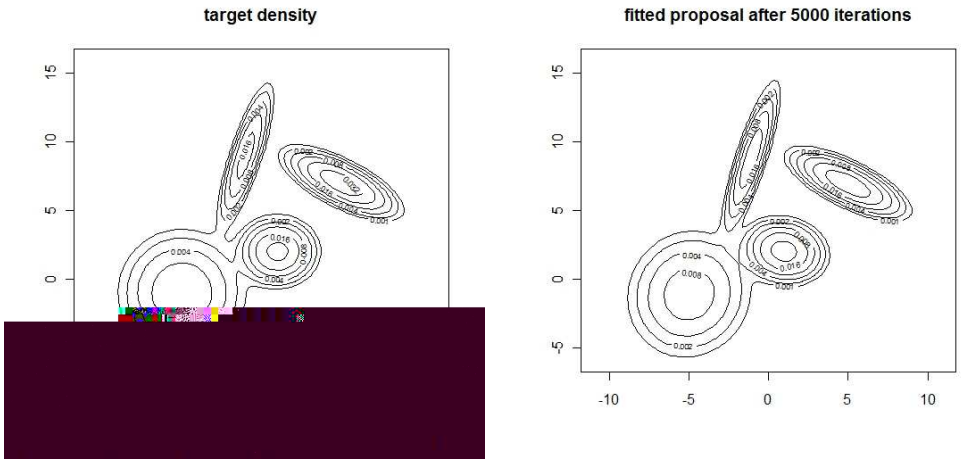

Figure 4: Left is a contour plot of the target density. Right is a contour plot of the IM proposal density.

2.4 Draw Backs of the EM Algorithm

Despite the promising results that are shown above, the EM algorithm has its own intrinsic draw backs, which will lead to the discussion of the Dirichlet Process Mixture Model.

First of all, the maximum likelihood estimator of the Gaussian Mixture Model of any data set is a mixture with one component of covariance matrix $\mathbf{0}$ and mean to be one of the data point. Since such component might not have mixing proportion 0, the global maximum likelihood of the data set is infinity. When we are running the EM algorithm, we hope the log likelihood converges to a local maximum instead of the global maximum. However, this is not always the case, more severe when we use more Gaussian components. Once the log likelihood function start to converge to the global maximum, it will produce a component covariance matrix that is nearly singular. The EM algorithm fails when Gaussian likelihood can no longer be computed because of the singularity of the covariance matrix. When we fit more than 10 Gaussian components to the data set using EM algorithm, it is so easy to converge to the global maximum that it happens almost every time. This is highly undesirable because in the MCMC context we expect to work with multi-modal target distributions.

The second draw back is the computation speed of EM, which is relatively slow among the standard density estimation techniques. As the sample accumulates, such computation becomes even more and more difficult. For example, to fit 70000 data points to a 10 dimensional Gaussian mixture with 6 components will take a few days on a standard laptop. One possible solution to this is the On-line EM algorithm (Craiu, 2011), in which we update the

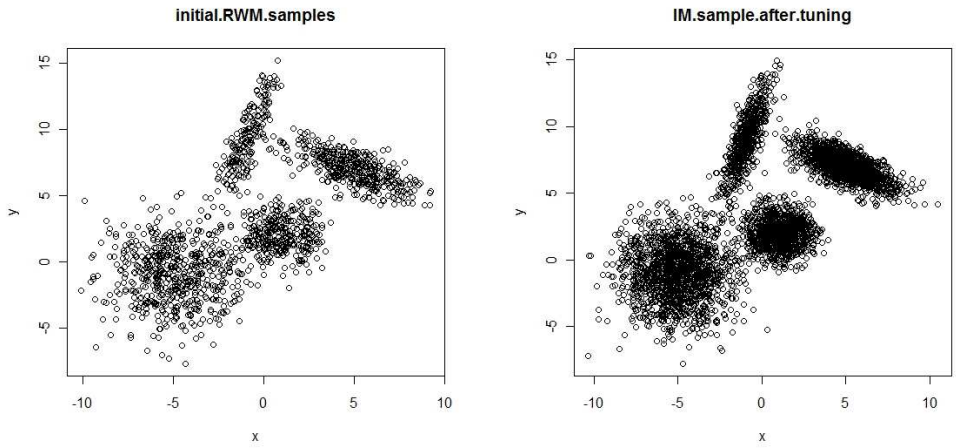

Figure 5: Left is a sample of size 5000 obtained from the initial parallel tempering MCMC. Right is a sample of size 5000 obtained from the tuned IM sampler.

parameters based on each data point in turn. This avoids increasing computational burden as samples accumulate. However, some simulation results suggest that the quality of this density estimation method is far from satisfaction.

The third draw back is that EM requires to pre-determine the number of Gaussian components to use. In the MCMC context, people usually have no or very little idea about the target density (which is why people are doing MCMC to know more about them), it is highly unrealistic to assume the number of modes of the target distribution without knowing the target quite well. This imposes another challenge on the EM algorithm. Thus we look for an alternative for the EM algorithm to solve these problems which we will discuss in the next chapter.

3 Dirichlet Process Mixture (DPM) Models

Dirichlet Process Mixture (DPM) is a mixture model to cluster a data set into different components without specifying the number of components to use. This Bayesian density estimation approach resolves the three major problems associated with the EM algorithm, i.e. infinite global maximum likelihood, heavy computation burden, and necessity to predetermine the number of modes of the target density. With a proper conjugate prior distribution, posterior computations could be pretty fast. In this chapter, we account for the possibility of using DPM instead of EM to approximate the target density in the MCMC context. Its performance in conjunction with the RCA algorithms will also be discussed. R code for simulation studies on DPM is provided in the appendix section.

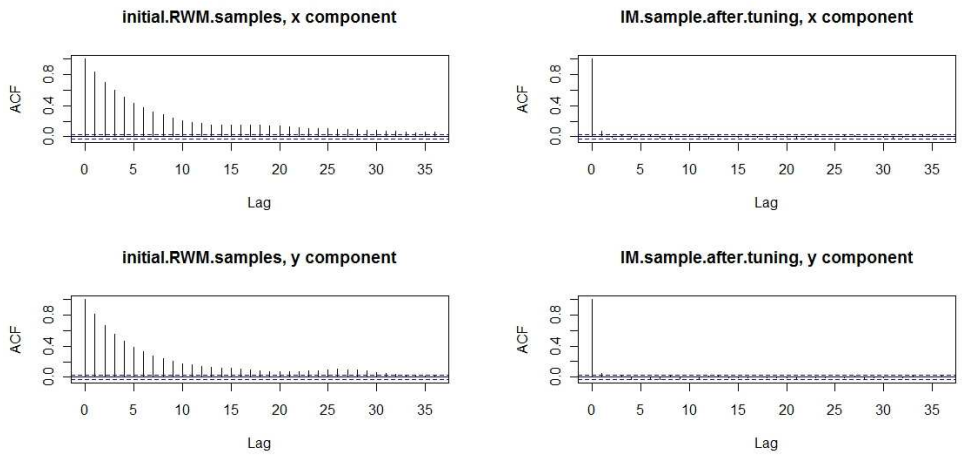

Figure 6: Left is a plot of the acf's of the initial samples. Right is a plot of the acf's of the IM samples.

3.1 An Introduction to the DPM Models

Let $\mathbf{y} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}\}$ be the data set. Our goal is to fit the data set to a Bayesian mixture model. Assume that the density of y has the form $\sum_{h=1}^{K} \rho_h f(y|\theta_h)$ where ρ_h are the mixing proportions and θ_h parameterizes the simple component distributions that are i.i.d. following some distribution p_0. ρ_h follows symmetric Dirichlet distribution with parameter α, with density

$$\frac{\Gamma(\alpha)}{\Gamma(\alpha/K)^K} \prod_{h=1}^{K} \rho_h^{(\alpha/K)-1}$$

where $\rho_h \geq 0$ and $\sum_h \rho_h = 1$. K is the number of components. We can express this model using class indicators γ_i that identifies the mixture component of each observation y_i. Denote $\Theta = \{\theta_1, \theta_2, \ldots, \theta_\mathrm{K}\}$ and F is the c.d.f. of the component distribution.

$$y_i|\gamma_i, \Theta \sim F(y_i|\theta_{\gamma_i})$$

$$\gamma_i|\rho_1, \rho_2, \ldots, \rho_K \sim Discrete(\rho_1, \rho_2, \ldots, \rho_K)$$

$$\theta_h \sim p_0$$

$$\rho_1, \rho_2, \ldots, \rho_K \sim Dirichlet(\alpha/K, \alpha/K, \ldots, \alpha/K)$$

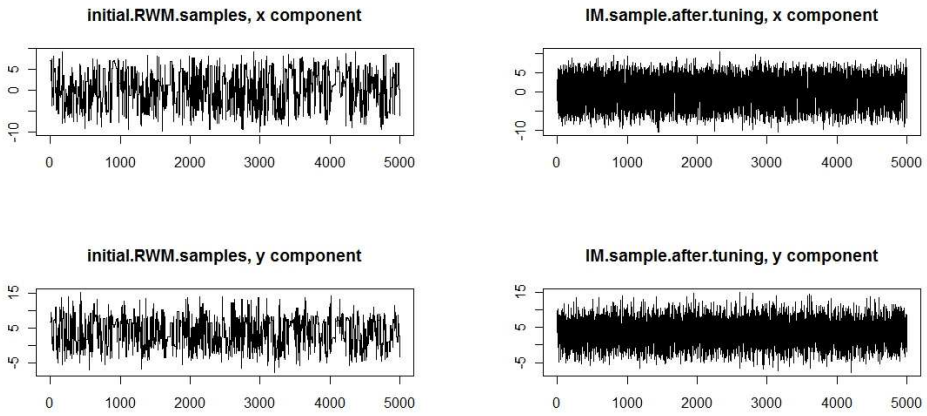

Figure 7: Left are the trace plots of the initial samples. Right are the trace plots of the IM samples.

The class indicators take values in $\{1, 2, \ldots, \mathrm{K}\}$. The mixing proportions ρ_h can be eliminated by integrating with respect to their Dirichlet prior and the resulting law of succession follows:

$$P(\gamma_i = h|\gamma_1, \gamma_2, \ldots, \gamma_{i-1}) = \frac{\sum_{j=1}^{i-1} \mathbb{I}_{(\gamma_j=h)} + \alpha/K}{\alpha + i - 1}$$

For density estimation, there is often reason to believe that the approximation to the real density will get arbitrarily close to the target as K goes to infinity. Actually as K goes to infinity, the behavior of limiting conditional prior distribution of γ_i given $\gamma^{(i-1)} = (\gamma_1, \ldots, \gamma_{i-1})$ is reasonable. It follows multinomial distribution with probability function:

$$\pi_{ih} = P(\gamma_i = h|\gamma^{(i-1)}) = \begin{cases} \frac{\sum_{j=1}^{i-1} \mathbb{I}_{(\gamma_j=h)}}{\alpha+i-1}, & h = 1, 2, \ldots, k_{i-1} \\ \frac{\alpha}{\alpha+i-1}, & h = k_{i-1} + 1. \end{cases} \tag{1}$$

where $\alpha > 0$ is the parameter of the Dirichlet distribution controlling sparseness and $k_{i-1} = max\{\gamma_h\}_{h=1}^{i-1}$ is the number of clusters after $i-1$ observations have been sequentially classified. As α increases, there is an increasing tendency to allocate observations to new clusters instead of clusters occupied by previous subjects. Also this prior favors allocation of observation i to clusters having large numbers of subjects.

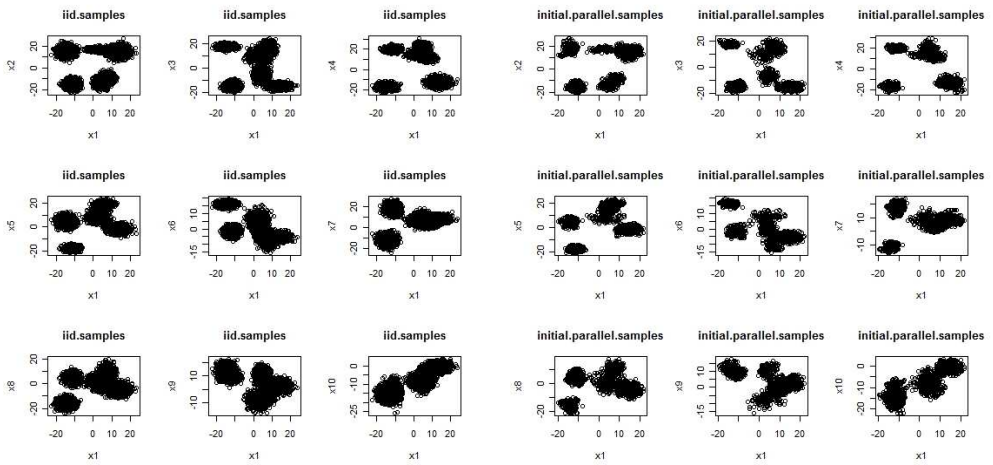

Figure 8: Left is a plot of the 10000 i.i.d. samples in nine 2D projections. Right is a plot of the 10000 parallel tempering chain samples in nine 2D projections.

3.2 Proposed Algorithm: Sequential Updating and Greedy Search (SUGS)

By the result of Dunson and Wang (2010), one can obtain the conditional posterior probability of allocating observation i to cluster h given $y^{(i)} = (y_1, \ldots, y_i)$ and the cluster assignments for $\gamma^{(i-1)} = (\gamma_1, \ldots, \gamma_{i-1})$:

$$P(\gamma_i = h | y^{(i)}, \gamma^{(i-1)}) = \frac{\pi_{ih} L_{ih}(y_i)}{\sum_{l=1}^{k_{i-1}+1} \pi_{il} L_{il}(y_i)}, h = 1, \ldots, k_{i-1} + 1 \tag{2}$$

where $\pi_{ih} = P(\gamma_i = h | \gamma^{(i-1)})$ is the conditional prior probability as in (1) and

$$L_{ih}(y_i) = \int f(y_i|\theta_h) \pi(\theta_h | y^{(i-1)}, \gamma^{(i-1)}) d\theta_h$$

is the conditional likelihood of y_i given allocation to cluster h and the cluster allocation for observations $1, \ldots, i-1$. $f(y_i|\theta_h)$ is the likelihood of y_i given parameters θ_h and $\pi(\theta_h | y^{(i-1)}, \gamma^{(i-1)})$ is the posterior distribution of θ_h given $y^{(i-1)}$ and $\gamma^{(i-1)}$. Notice that $L_{ih}(\cdot)$ is exactly the posterior predictive distribution for new data y_i conditioning on observations in cluster h up to index $i-1$. Under conjugate prior assumption, for each cluster, both posterior distribution of θ_h and posterior predictive distribution $L_{ih}(\cdot)$ will be analytically tractable. This gives rise to the following sequential updating scheme:

(i) As $i = 1$, let $\gamma_1 = 1$. Hence $k_1 = 1$. Compute the posterior distribution $\pi(\theta_1 | y_1, \gamma_1)$ for θ_1, the parameters for the first component distribution.

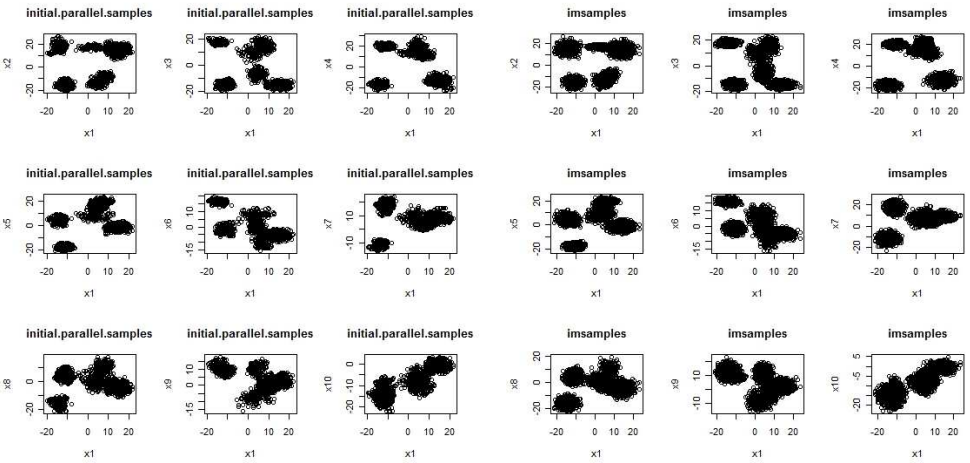

Figure 9: Left is a plot of the 10000 parallel tempering samples in nine 2D projections. Right is a plot of the 10000 tuned IM samples in nine 2D projections.

(ii) Then $i = 2$, compute the probability $P(\gamma_2 = h|y^{(2)}, \gamma^{(1)})$ for $h = 1$ and 2 according to (1). To determine the clustering probability we also need $L_{21}(y_2)$ and $L_{22}(y_2)$. $L_{21}(y_2)$ can be computed using $\pi(\theta_1|y_1, \gamma_1)$. $L_{22}(y_2)$ involves allocation to a new cluster. Hence the posterior predictive distribution should be computed using prior distribution for θ_2. Then we choose γ_2 to maximize the conditional probability of $\gamma_2 = h$ given $y^{(2)}, \gamma^{(1)}$.

(iii) If $\gamma_2 = 1$, then k_2 remains 1. If $\gamma_2 = 2$, then $k_2 = 2$. According to the label of observation 2 we update the posterior distribution of component 1 or 2 correspondingly.

(iv) In general, for $i = 3, \ldots, n$,

(a)Choose γ_i to maximize the conditional probability of $\gamma_i = h$ given $y^{(i)}$ and $\gamma^{(i-1)}$

(b)Update $\pi(\theta_{\gamma_i}|y^{(i-1)}, \gamma^{(i-1)})$ using the observation y_i. If y_i belongs to a new cluster, then we update the prior distribution p_0 with y_i.

3.3 Normal Mixture and Conjugate Prior

We restrict ourselves to normal mixture models. The component distributions are multivariate normal distributions. Simulation results suggest that non-conjugate prior distributions p_0 imposes significant amount of calculation burden so we avoid them here. We set p_0 as the Normal-Scaled-Inverse-χ^2 distribution for the one dimension case. For multivariate cases, we set p_0 as the Normal-Inverse-Wishart distribution.

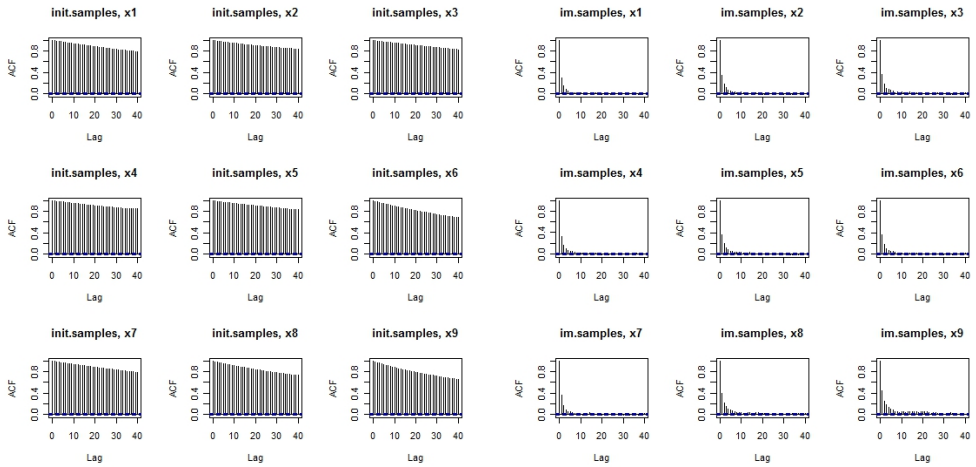

Figure 10: Left is a plot of the acf of 10000 parallel tempering samples in 9 projections. Right is a plot of the acf of 10000 tuned IM samples in 9 projections.

3.3.1 The Univariate Case

When y_i's are one dimensional, we aim to model them with a univariate Gaussian mixture. In this case, the prior distribution p_0 for $\theta_h = (\mu_h, \sigma_h^2)$ is a two-dimensional distribution called Normal-Scaled-Inverse-χ^2 distribution where the marginal distribution of σ_h^2 is scaled-inverse-χ^2 distribution and the condition distribution of μ_h given σ_h^2 is normal. It can be parameterized with the following specification:

$$\mu_h|\sigma_h^2 \sim N(\mu_0, \sigma_h^2/\kappa_0)$$

$$\sigma_h^2 \sim \text{Inv-}\chi^2(\nu_0, \sigma_0^2)$$

which corresponds to the joint prior probability density function:

$$p_0(\mu_h, \sigma_h^2) = \frac{\sqrt{\kappa_0}(\sigma_0^2\nu_0/2)^{\nu_0/2}}{\sqrt{2\pi}\Gamma(\nu_0/2)}\sigma_h^{-1}(\sigma_h^2)^{-(\nu_0/2+1)}exp(-\frac{1}{2\sigma_h^2}[\nu_0\sigma_0^2 + \kappa_0(\mu_0 - \mu_h)^2])$$

This is labeled as the Normal-Scaled-Inverse-χ^2 distribution, denoted as $N - Inv - \chi^2(\mu_0, \sigma_0^2/\kappa_0; \nu_0, \sigma_0^2)$. Its four parameters can be identified as the location and scale of μ_h and the degrees of freedom and scale of σ_h^2, respectively. Let $\mathbf{y} = (y_1, \ldots, y_n)$ be the observations. Multiplying the prior density by the normal likelihood yields the joint posterior density:

$$\begin{aligned}\pi(\mu_h, \sigma_h^2|\mathbf{y}) \propto \sigma_h^{-1}(\sigma_h^2)^{-(\nu_0/2+1)}exp(-\tfrac{1}{2\sigma_h^2}[\nu_0\sigma_0^2 + \kappa_0(\mu_0 - \mu_h)^2]) \\ \times(\sigma_h^2)^{-(n/2)}exp(-\tfrac{1}{2\sigma_h^2}[(n-1)s^2 + n(\bar{y} - \mu_h)^2])\end{aligned}$$

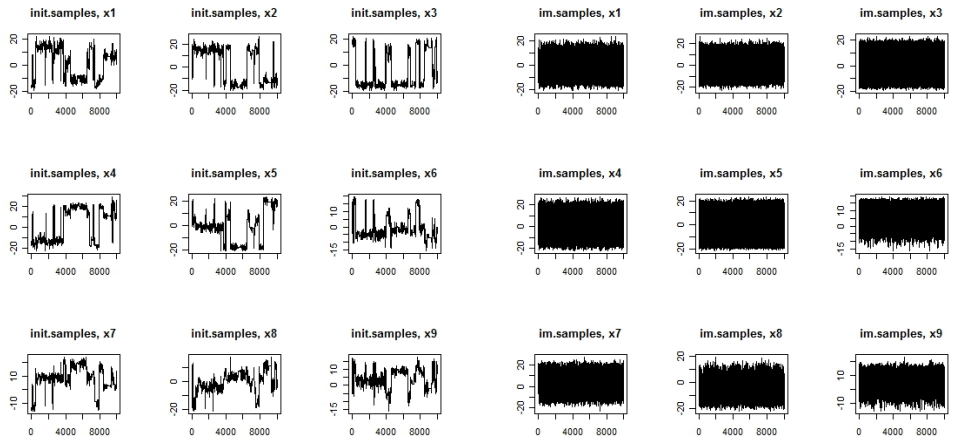

Figure 11: Left is a trace plot of 10000 parallel tempering samples in 9 projections. Right is a trace plot of 10000 tuned IM samples in 9 projections.

$$= N - Inv - \chi^2(\mu_n, \sigma_n^2/\kappa_n; \nu_n, \sigma_n^2)$$

where $\bar{y} = \frac{1}{n}\sum_i y_i$ and $s^2 = \frac{1}{n-1}\sum_i (y_i - \bar{y})^2$. After some algebra we can show that:

$$\mu_n = \frac{\kappa_0}{\kappa_0 + n}\mu_0 + \frac{n}{\kappa_0 + n}\bar{y} \tag{3}$$

$$\kappa_n = \kappa_0 + n \tag{4}$$

$$\nu_n = \nu_0 + n \tag{5}$$

$$\nu_n\sigma_n^2 = \nu_0\sigma_0^2 + (n-1)s^2 + \frac{\kappa_0 n}{\kappa_0 + n}(\bar{y} - \mu_0)^2 \tag{6}$$

The prior mean for (μ_h, σ_h^2) is $(\mu_0, \frac{\nu_0\sigma_0^2}{\nu_0 - 2})$ if $\nu_0 > 2$. The posterior mean for (μ_h, σ_h^2) is $(\mu_n, \frac{\nu_n\sigma_n^2}{\nu_n - 2})$ if $\nu_n > 2$.

Now we give a derivation of the normalizing constant, $\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h$, of the posterior distribution of $\theta_h = (\mu_h, \sigma_h^2)$. By an application of the Baye's rule,

$$\pi(\theta_h|\mathbf{y}) = \frac{f(\mathbf{y}|\theta_h)p_0(\theta_h)}{\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h}$$

Thus,

$$\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h = \frac{f(\mathbf{y}|\theta_h)p_0(\theta_h)}{\pi(\theta_h|\mathbf{y})}$$

$$= \frac{\prod_{l=1}^{n}(\frac{1}{\sqrt{2\pi}\sigma_h}exp(-\frac{1}{2\sigma_h^2}(y_l-\mu_h)^2))\frac{\sqrt{\kappa_0}(\sigma_0^2\nu_0/2)^{\nu_0/2}}{\sqrt{2\pi}\Gamma(\nu_0/2)}\sigma_h^{-1}(\sigma_h^2)^{-(\nu_0/2+1)}exp(-\frac{1}{2\sigma_h^2}[\nu_0\sigma_0^2+\kappa_0(\mu_0-\mu_h)^2])}{\frac{\sqrt{\kappa_n}(\sigma_n^2\nu_n/2)^{\nu_n/2}}{\sqrt{2\pi}\Gamma(\nu_n/2)}\sigma_h^{-1}(\sigma_h^2)^{-(\nu_n/2+1)}exp(-\frac{1}{2\sigma_h^2}[\nu_n\sigma_n^2+\kappa_n(\mu_n-\mu_h)^2])}$$

Simplify the above formula we have,

$$\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h = \frac{\Gamma(\nu_n/2)\sqrt{\kappa_0}(\sigma_0^2\nu_0/2)^{\nu_0/2}}{(\sqrt{2\pi})^n\Gamma(\nu_0/2)\sqrt{\kappa_n}(\sigma_n^2\nu_n/2)^{\nu_n/2}} \tag{7}$$

In order to compute the conditional posterior probability of allocating observation i to cluster h given $y^{(i)}$ and the cluster assignments $\gamma^{(i-1)}$, we need to be able to compute the integral: $L_{ih}(y_i) = \int f(y_i|\theta_h)\pi(\theta_h|y^{(i-1)},\gamma^{(i-1)})d\theta_h$. Under the conjugate prior assumptions, the integral is the ratio between the normalizing constant of the posterior distribution of θ_h up to $i-1$ and the normalizing constant of the posterior distribution of θ_h up to i, which is analytically tractable. To be more specific, we give an explicit formula for $L_{ih}(y_i)$ here under the conjugate prior assumption. Suppose according to the cluster assignments $\gamma^{(i-1)}$, data set $\mathbf{y} = \{\mathrm{y_1}, \mathrm{y_2}, \ldots, \mathrm{y_n}\}$ is assigned to cluster h. $\theta_h|\mathbf{y}, y_i \sim N - Inv - \chi^2(\mu, \sigma^2/\kappa; \nu, \sigma^2)$.

$$\mu = \frac{\kappa_0}{\kappa_0+n+1}\mu_0 + \frac{n+1}{\kappa_0+n+1}\bar{y}$$

$$\kappa = \kappa_0 + n + 1$$

$$\nu = \nu_0 + n + 1$$

$$\nu\sigma^2 = \nu_0\sigma_0^2 + ns^2 + \frac{\kappa_0(n+1)}{\kappa_0+n+1}(\bar{y}-\mu_0)^2$$

where $\bar{y} = \frac{y_1+y_2+\ldots+y_n+y_i}{n+1}$ and $s^2 = \frac{1}{n}(\sum_{l=1}^{n}(y_l-\bar{y})^2 + (y_i-\bar{y})^2)$

By an application of formula (7), we can derive a formula for $L_{ih}(y_i)$:

$$\begin{aligned}
L_{ih}(y_i) &= \int f(y_i|\theta_h)\pi(\theta_h|y^{(i-1)},\gamma^{(i-1)})d\theta_h \\
&= \int f(y_i|\theta_h)\pi(\theta_h|\mathbf{y})d\theta_h \\
&= \int f(y_i|\theta_h)\frac{f(\mathbf{y}|\theta_h)p_0(\theta_h)}{\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h}d\theta_h \\
&= \frac{\int f(\mathbf{y},y_i|\theta_h)p_0(\theta_h)d\theta_h}{\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h} \\
&= \frac{\frac{\Gamma(\nu/2)\sqrt{\kappa_0}(\sigma_0^2\nu_0/2)^{\nu_0/2}}{(\sqrt{2\pi})^{n+1}\Gamma(\nu_0/2)\sqrt{\kappa}(\sigma^2\nu/2)^{\nu/2}}}{\frac{\Gamma(\nu_n/2)\sqrt{\kappa_0}(\sigma_0^2\nu_0/2)^{\nu_0/2}}{(\sqrt{2\pi})^n\Gamma(\nu_0/2)\sqrt{\kappa_n}(\sigma_n^2\nu_n/2)^{\nu_n/2}}}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\sqrt{\kappa_n}(\sigma_n^2\nu_n/2)^{\nu_n/2}\Gamma(\nu/2)}{\sqrt{2\pi}\sqrt{\kappa}(\sigma^2\nu/2)^{\nu/2}\Gamma(\nu_n/2)} \\
&= \frac{\sqrt{\kappa_n}(\sigma_n^2\nu_n/2)^{(\nu_0+n)/2}\Gamma((\nu_0+n+1)/2)}{\sqrt{2\pi}\sqrt{\kappa}(\sigma^2\nu/2)^{(\nu_0+n+1)/2}\Gamma((\nu_0+n)/2)} \\
&= \sqrt{\frac{\kappa_n}{2\pi\kappa}}(\frac{\sigma_n^2\nu_n}{\sigma^2\nu})^{(\frac{\nu_0+n}{2})}\frac{1}{(\sigma^2\nu/2)^{1/2}}\frac{\Gamma(\frac{\nu_0+n+1}{2})}{\Gamma(\frac{\nu_0+n}{2})}
\end{aligned}
$$

where $\Gamma(\cdot)$ is the Gamma function.

3.3.2 The Multivariate Case

When y_i's are d dimensional, we aim to model them with a multivariate Gaussian mixture. In this case, we need to generalize the prior distribution p_0, Normal-Scaled-Inverse-χ^2 distribution in the one dimensional case, to be Normal-Inverse-Wishart distribution. The conjugate prior for $\theta_h = (\mu_h, \Sigma_h)$, the Normal-Inverse-Wishart distribution is parameterized in terms of hyperparameters $(\mu_0, \Lambda_0/\kappa_0; \nu_0, \Lambda_0)$:

$$
\mu_h|\Sigma_h \sim N(\mu_0, \Sigma_h/\kappa_0)
$$

$$
\Sigma_h \sim \text{Inv-Wishart}_{\nu_0}(\Lambda_0^{-1})
$$

We denote the prior distribution as $N - Inv - Wishart(\mu_0, \Lambda_0/\kappa_0; \nu_0, \Lambda_0)$. $\theta_h = (\mu_h, \Sigma_h)$ follows the joint density function:

$$
p_0(\mu_h, \Sigma_h) = \frac{\kappa_0^{d/2}|\Lambda_0|^{\nu_0/2}}{2^{\frac{(\nu_0+1)d}{2}}\pi^{\frac{d(d+1)}{4}}\prod_{i=1}^{d}\Gamma(\frac{\nu_0+1-i}{2})}|\Sigma_h|^{-(\frac{\nu_0+d}{2}+1)}exp(-\frac{1}{2}tr(\Lambda_0\Sigma_h^{-1})-\frac{\kappa_0}{2}(\mu_h-\mu_0)^T\Sigma_h^{-1}(\mu_h-\mu_0))
$$

μ_0 can be identified as the prior mean of μ_h. κ_0 is the number of prior measurements on the Σ_h scale. The parameters ν_0 and Λ_0 describe the degrees of freedom and the scale matrix for the inverse-Wishart distribution on Σ_h. Multiplying the prior density by the normal likelihood results in the posterior density in the same family.

$$
(\mu_h, \Sigma_h|\mathbf{y}) \sim N - Inv - Wishart(\mu_n, \Lambda_n/\kappa_n; \nu_n, \Lambda_n)
$$

We have:

$$
\begin{aligned}
\mu_n &= \frac{\kappa_0}{\kappa_0+n}\mu_0 + \frac{n}{\kappa_0+n}\bar{y} \\
\kappa_n &= \kappa_0 + n
\end{aligned}
$$

$$\nu_n = \nu_0 + n$$

$$\Lambda_n = \Lambda_0 + S + \frac{\kappa_0 n}{\kappa_0 + n}(\bar{y} - \mu_0)(\bar{y} - \mu_0)^T$$

where S is the sum of squares matrix about the sample mean,

$$S = \sum_{i=1}^{n}(y_i - \bar{y})(y_i - \bar{y})^T$$

The prior mean for (μ_h, Σ_h) is $(\mu_0, \frac{\Lambda_0}{\nu_0 - (d+1)})$ if $\nu_0 > d+1$. The posterior mean for (μ_h, Σ_h) is $(\mu_n, \frac{\Lambda_n}{\nu_n - (d+1)})$ if $\nu_n > d+1$.

Now we give a derivation of the normalizing constant, $\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h$, of the posterior distribution, $\pi(\theta_h|\mathbf{y})$, of $\theta_h = (\mu_h, \Sigma_h)$ in the multivariate case. By an application of the Baye's rule,

$$\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h = \frac{f(\mathbf{y}|\theta_h)p_0(\theta_h)}{\pi(\theta_h|\mathbf{y})}$$

$$= \frac{\prod_{i=1}^{n}[(2\pi)^{-\frac{d}{2}}|\Sigma_h|^{-\frac{1}{2}}exp(-\frac{1}{2}(y_i - \mu_h)^T\Sigma_h^{-1}(y_i - \mu_h))]p_0(\theta_h)}{\frac{\kappa_n^{d/2}|\Lambda_n|^{\nu_n/2}}{2^{\frac{(\nu_n+1)d}{2}}\pi^{\frac{d(d+1)}{4}}\prod_{i=1}^{d}\Gamma(\frac{\nu_n+1-i}{2})}|\Sigma_h|^{-(\frac{\nu_n+d}{2}+1)}exp(-\frac{1}{2}tr(\Lambda_n\Sigma_h^{-1}) - \frac{\kappa_n}{2}(\mu_h - \mu_n)^T\Sigma_h^{-1}(\mu_h - \mu_n))}$$

Simplify the above formula we have,

$$\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h = \frac{\kappa_0^{d/2}|\Lambda_0|^{\nu_0/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_n+1-i}{2})}{\pi^{nd/2}\kappa_n^{d/2}|\Lambda_n|^{\nu_n/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_0+1-i}{2})} \tag{8}$$

Now we give an explicit formula for $L_{ih}(y_i)$ here under the conjugate prior assumption in the multivariate case. This is just an generalization of the one dimensional case. Suppose according to the cluster assignments $\gamma^{(i-1)}$, data set $\mathbf{y} = \{\mathrm{y}_1, \mathrm{y}_2, \ldots, \mathrm{y}_\mathrm{n}\}$ is assigned to cluster h. Assume the new data point y_i is clustered into component h, we have $\theta_h|\mathbf{y}, y_i \sim N - Inv - Wishart(\mu, \Lambda/\kappa; \nu, \Lambda)$.

We have:

$$\mu = \frac{\kappa_0}{\kappa_0 + n + 1}\mu_0 + \frac{n+1}{\kappa_0 + n + 1}\bar{y}$$

$$\kappa = \kappa_0 + n + 1$$

$$\nu = \nu_0 + n + 1$$

$$\Lambda = \Lambda_0 + S + \frac{\kappa_0(n+1)}{\kappa_0+n+1}(\bar{y}-\mu_0)(\bar{y}-\mu_0)^T$$

where $\bar{y} = \frac{y_1+y_2+...+y_n+y_i}{n+1}$ and S is the sum of squares matrix about the sample mean,

$$S = \sum_{l=1}^{n}(y_l-\bar{y})(y_l-\bar{y})^T + (y_i-\bar{y})(y_i-\bar{y})^T$$

By an application of formula (8), we can derive a formula for $L_{ih}(y_i)$ in the multivariate case:

$$\begin{aligned}
L_{ih}(y_i) &= \int f(y_i|\theta_h)\pi(\theta_h|y^{(i-1)},\gamma^{(i-1)})d\theta_h \\
&= \frac{\int f(\mathbf{y},y_i|\theta_h)p_0(\theta_h)d\theta_h}{\int f(\mathbf{y}|\theta_h)p_0(\theta_h)d\theta_h} \\
&= \frac{\frac{\kappa_0^{d/2}|\Lambda_0|^{\nu_0/2}\prod_{i=1}^{d}\Gamma(\frac{\nu+1-i}{2})}{\pi^{(n+1)d/2}\kappa^{d/2}|\Lambda|^{\nu/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_0+1-i}{2})}}{\frac{\kappa_0^{d/2}|\Lambda_0|^{\nu_0/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_n+1-i}{2})}{\pi^{nd/2}\kappa_n^{d/2}|\Lambda_n|^{\nu_n/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_0+1-i}{2})}} \\
&= \frac{\kappa_n^{d/2}|\Lambda_n|^{\nu_n/2}\prod_{i=1}^{d}\Gamma(\frac{\nu+1-i}{2})}{\pi^{d/2}\kappa^{d/2}|\Lambda|^{\nu/2}\prod_{i=1}^{d}\Gamma(\frac{\nu_n+1-i}{2})} \\
&= \frac{1}{\pi^{d/2}}(\frac{\kappa_n}{\kappa})^{d/2}\frac{1}{|\Lambda|^{1/2}}(\frac{|\Lambda_n|}{|\Lambda|})^{\frac{\nu_0+n}{2}}\prod_{i=1}^{d}(\frac{\Gamma(\frac{\nu_0+n+2-i}{2})}{\Gamma(\frac{\nu_0+n+1-i}{2})})
\end{aligned}$$

3.4 Simulation Studies: Regime Change Algorithm via DPM

We run SUGS to partition the observations. After we obtain the partition, compute sample mean, variance and proportion for each cluster to obtain the Gaussian mixture. Previous SUGS study shows that the clustering results are sensitive to the selection of prior hyper-parameters. Some more careful studies should be devoted to how the prior distribution should be selected.

3.4.1 A Two Dimensional Target Distribution

In this section, let us consider the following target distribution:

$$\pi(x) = \frac{5}{16}N\Big((5,7),\begin{bmatrix}2 & -1\\ -1 & 1\end{bmatrix}\Big)+\frac{5}{16}N\Big((-5,-1),\begin{bmatrix}3 & 0\\ 0 & 4\end{bmatrix}\Big)+\frac{1}{8}N\Big((-1,9),\begin{bmatrix}0.5 & 1.2\\ 1.2 & 4.5\end{bmatrix}\Big)+\frac{1}{4}N\Big((1,2),\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\Big)$$

We first run the parallel tempering MCMC to obtain a sample of size 5000 from the target distribution. Then we run SUGS to partition the sample into clusters. We note

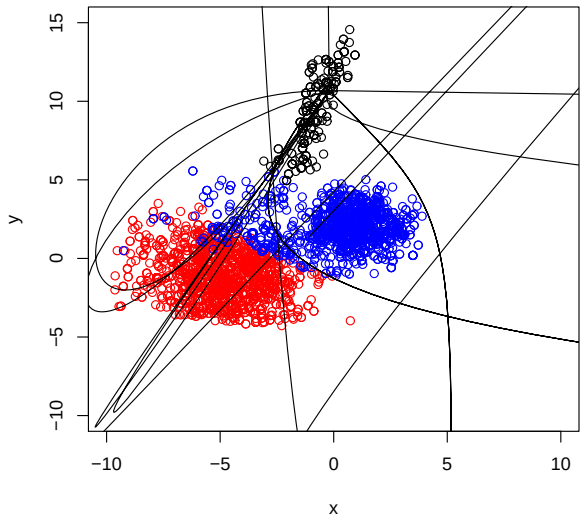

Figure 12: The clustering results for the 5000 samples from the target distribution in section 5.1. Five different colors stand for five different clusters.

the prior hyper-parameters for the clustering algorithm in this case are: $\alpha = 0.1, \mu_0 = (0,0), \Lambda_0 = I, \kappa_0 = 0.01, \nu_0 = 4$. From previous simulation results, as the generalization of the 1-dimensional SUGS, α represents the initial tendency to create a new cluster, the larger the value α is, more clusters will be created. However, creating more clusters does not mean we will end up with more useful clusters. Simulation results show that increasing the value of α only often tends to create a lot of degenerated components with only one data point. μ_0 is the initial belief of where the mean of the component distribution should be. κ_0 shows how certain one is about the initial guess of the component mean, μ_0. The larger the κ_0 value, the less influence there will be for the data set on the posterior mean of the components. Both Λ_0 and ν_0 contribute to the initial guess of the component covariance matrix Σ, whereas ν_0 also represents how certain one is about the initial guess of the component covariance matrix. The larger value ν_0 takes, the harder for the posterior covariance matrix for the component distribution to differ from the prior.

Running SUGS algorithm, we obtain 5 clusters for the samples from the target distribution. See Figure 12 for the results. To compare how close the target distribution and the proposal distribution are, we look at mathematica 3-D plot (Figure 13) and a contour plot produced by R. (See Figure 14.)

Finally we run the IM sampler using the proposal distribution we obtained from SUGS.

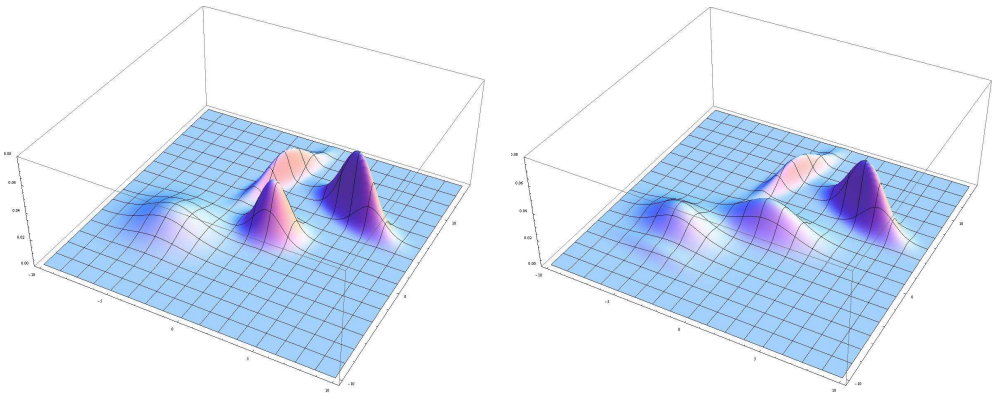

Figure 13: Left is a 3-D Mathematica plot of the target density. Right is a 3-D plot of the proposal density used by the IM sampler via SUGS.

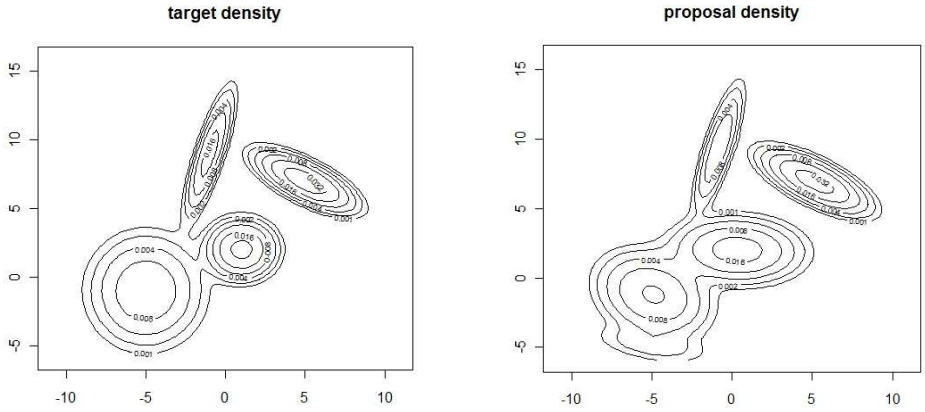

Figure 14: Left is a 2-D contour plot of the target density produced by R. Right is a 2-D contour plot of the proposal density used by the IM sampler via SUGS.

The acceptance probability for the IM sampler is 0.7782. IM sampler gives $2-dependent$ samples demonstrated by the auto-correlation function of the IM samples. Acf's are shown in Figure 15. And the trace plots of the x and y coordinate respectively are given in Figure 16.

3.4.2 A Ten Dimensional Target Distribution

Consider the target distribution $\pi(x)$ in section 2.3.3. Now instead of fitting a Gaussian mixture model to the initial samples, we use DPM model to cluster all the initial samples. We obtain a sample of size 10000 from the target distribution using parallel tempering MCMC. Then we cluster these 10000 data points by DPM. We take prior hyperparameters

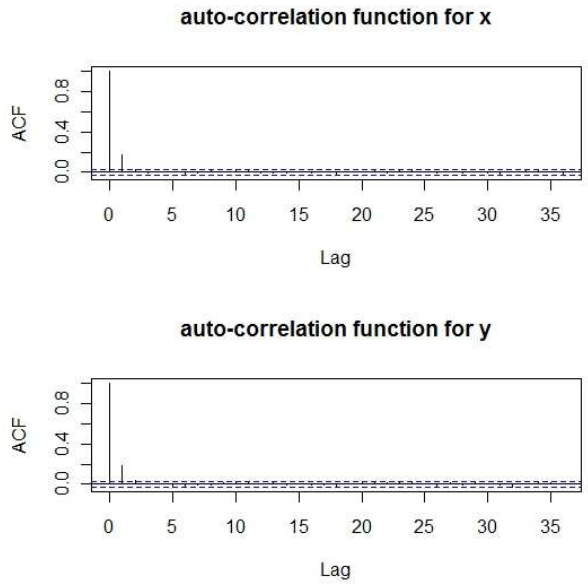

Figure 15: The ACF of the x and y coordinate respectively of the samples obtained from the IM sampler.

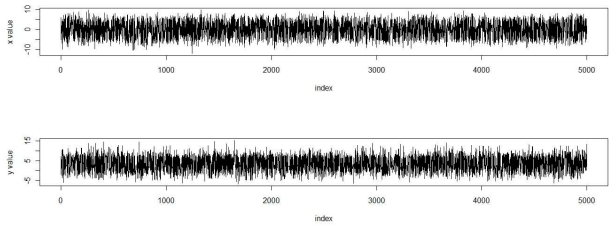

Figure 16: The trace plots of the x and y coordinate respectively of the samples obtained from the IM sampler.

to be the following: $\alpha = 0.1$, $\mu_0 = (0.37, 1.52, 1.57, 5.92, 5.01, 1.04, 5.37, -0.93, 3.91, -8.61)^T$, $\Lambda_0 = 20I$, $\kappa_0 = 0.01$ and $\nu_0 = 12$. DPM produced 32 components for the 10000 data points. Correspondingly, we form a 32-component Gaussian mixture proposal density for the IM sampler. We estimate the mean and covariance matrices by empirical estimates. Some components will have a covariance matrix that is almost singular. In such cases, we replace the empirical estimate with the identity matrix to avoid calculation problems. With estimates from the initial 10000 parallel tempering chain samples, the IM sampler has acceptance rate 0.3151. For a comparison between the initial 10000 parallel tempering chain samples and 10000 tuned IM samples, see figure $17 - 19$.

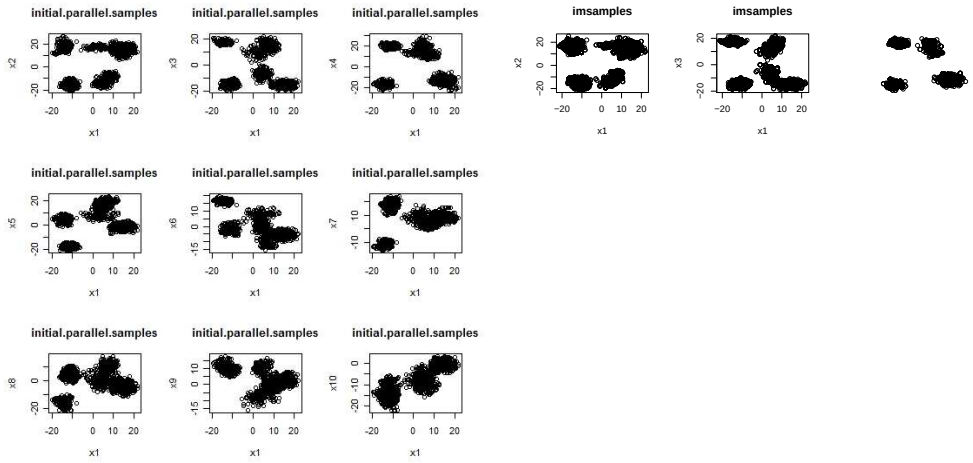

Figure 17: Left is a plot of the 10000 parallel tempering samples in nine 2D projections. Right is a plot of the 10000 tuned IM samples in nine 2D projections using DPM.

3.5 Inference at a Higher Level - an Empirical Bayesian Approach

Simulation results show that the efficiency of the SUGS algorithm will largely depend on the prior hyperparameters. A comprehensive survey of 180 simulations with different prior hyperparameters show that the clustering results of 1000 data points can differ from one component to 1000 components. Hence some more careful analysis of how the prior hyperparameters should be chosen should be done to ensure that the algorithm will work well. We adopt an empirical Bayesian approach, allowing data to affect the choice of the prior hyperparameters.

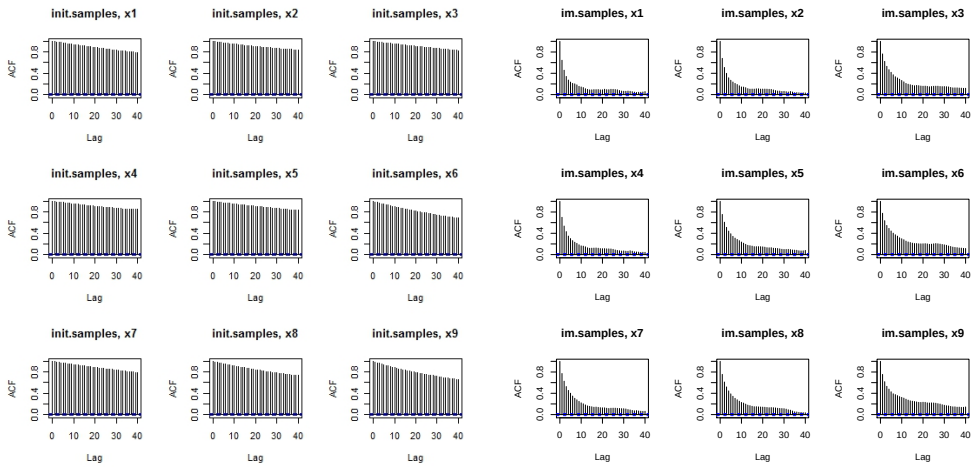

Figure 18: Left is a plot of the acf of 10000 parallel tempering samples in 9 projections. Right is a plot of the acf of 10000 tuned IM samples in 9 projections using DPM.

3.5.1 Bayesian Inference for Prior DP Precision Parameter α

The DP precision parameter α plays an important role in SUGS because it determines the prior tendency to create more or fewer clusters. If we specify a fixed α value, then we are imposing some prior beliefs on how many clusters to create. We use a Bayesian approach to fix this problem. Following the results by Dunson and Wang (2010), we put a discrete prior distribution on α. Let $\alpha^* = (\alpha_1^*, \alpha_2^*, \ldots, \alpha_T^*)'$, where α^* is the vector of all possible values for α and T is the total number of possible values. The prior distribution is specified by $\phi_m^{(0)} = P(\alpha = \alpha_m^*)$ for $m = 1, 2, \ldots, T$. Now define $\pi_{ihm} = P(\gamma_i = h | \alpha = \alpha_m^*, \gamma^{(i-1)})$. Then we obtain the following updating rules:

$$\phi_m^{(i)} = P(\alpha = \alpha_m^* | \gamma^{(i)}) = \frac{\phi_m^{(i-1)} \pi_{i\gamma_i m}}{\sum_{s=1}^{T} \phi_s^{(i-1)} \pi_{i\gamma_i s}}$$

for $m = 1, 2, \ldots, T$. Since for any α value, we will always assign the first data point to the first cluster, we can see that $\phi_m^{(0)} = \phi_m^{(1)}$ for $m = 1, 2, \ldots, T$. Following the notation in (2), we can obtain the new posterior clustering probabilities under the prior distribution for α assumed in this section:

$$P(\gamma_i = h | y^{(i)}, \gamma^{(i-1)}) = \frac{\sum_{m=1}^{T} \phi_m^{(i-1)} \pi_{ihm} L_{ih}(y_i)}{\sum_{m=1}^{T} \phi_m^{(i-1)} \sum_{l=1}^{k_{i-1}+1} \pi_{ilm} L_{il}(y_i)}$$

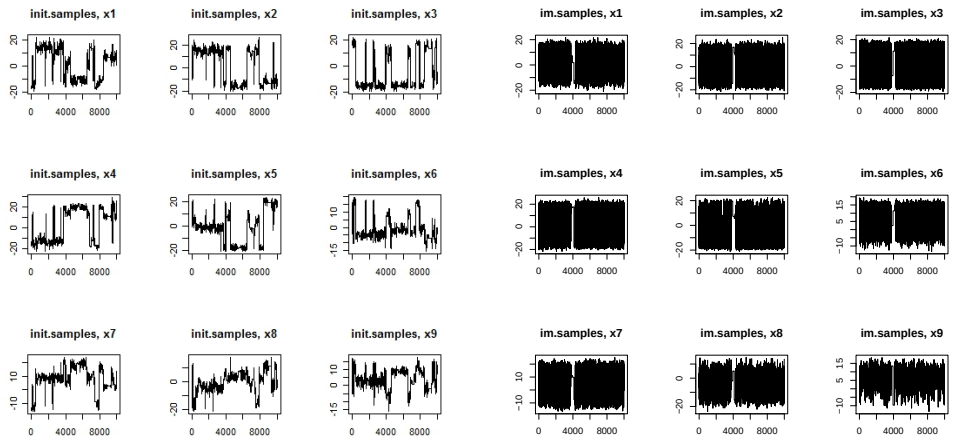

Figure 19: Left is a trace plot of 10000 parallel tempering samples in 9 projections. Right is a trace plot of 10000 tuned IM samples in 9 projections using DPM.

3.5.2 Empirical Bayesian Inference for Other Hyperparameters

In both the univariate and the multivariate case, the clustering result of SUGS is sensitive to the other choice of the hyperparameters used in the prior distribution of the component parameters. In order for the clustering algorithm to work well, we propose an empirical Bayesian method to determine the prior hyper-parameters. The SUGS algorithm should go through this initialization process to obtain maximum performance on clustering data points.

Step I. Run a parallel tempering, non-adaptive chain for a number of steps. The goal is to cover most of the modes of the target distribution.
Step II. Apply a standard non-parametric clustering algorithm to the obtained sample. Use the gap statistic to determine the number of clusters when applying the standard clustering algorithm.
Step III. Compute the sample mean for each of the clusters, find the maximum likelihood estimator of the hyperparameters to maximize the likelihood of the sample means of the clusters.

3.5.3 The Maximum Likelihood Estimator Approach

In the multivariate case, the conjugate prior for $\theta_h = (\mu_h, \Sigma_h)$, the Normal-Inverse-Wishart distribution is parameterized in terms of hyperparameters $(\mu_0, \Lambda_0/\kappa_0; \nu_0, \Lambda_0)$. ν_0 is the initial degrees of freedom, we take it to be $d+2$ for the prior mean of the component covariance matrix to be positive definite, where d is the dimension of the target distribution. The

value of ν_0 should not be big since a big value of ν_0 will strengthen the importance of the prior belief and reduce the variability of component covariance matrix under the posterior distribution. κ_0 is also taken as a constant in our empirical Bayesian method. The value for κ_0 should also be small since a large value in κ_0 will reduce the variability of component mean under the posterior distribution. In practice, we can just take κ_0 to be 1.

We fix the values of ν_0 and κ_0, treating them as constants. Our goal is to find the MLE for μ_0 and Λ_0. First go through an initialization process, obtaining K clusters, denoted as $C_1 = \{x_1^{(1)}, x_2^{(1)}, \ldots, x_{n_1}^{(1)}\}, C_2 = \{x_1^{(2)}, x_2^{(2)}, \ldots, x_{n_2}^{(2)}\}, \ldots, C_K = \{x_1^{(K)}, x_2^{(K)}, \ldots, x_{n_K}^{(K)}\}$. Obtain the K sample means, $\hat{\mu}_h$ from these clusters, for $h = 1, 2, \ldots, K$. Under our model assumptions,

$$\mu_h|\Sigma_h \sim N(\mu_0, \Sigma_h/\kappa_0)$$

$$\Sigma_h \sim \text{Inv-Wishart}_{\nu_0}(\Lambda_0^{-1})$$

Identifying the component sample means as the component means, the maximum likelihood estimator for the parameter μ_0 is $\hat{\mu}_0 = \frac{1}{K}\sum_{h=1}^{K}\hat{\mu}_h$ to maximize the likelihood of the component sample means. Here we derive the MLE for the parameter Λ_0 to maximize the likelihood of the component sample means. Assuming all K components have the same covariance matrix Σ, the likelihood function

$$\begin{aligned}
L(\mu_0, \Lambda_0) &= f(\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_K|\mu_0, \Lambda_0) \\
&= \int f(\hat{\mu}_1, \hat{\mu}_2, \ldots, \hat{\mu}_K|\mu_0, \Lambda_0, \Sigma) f(\Sigma|\mu_0, \Lambda_0) d\Sigma \\
&= \int \prod_{h=1}^{K} \Big\{(2\pi)^{-d/2} \det\big(\Sigma/\kappa_0\big)^{-\frac{1}{2}} exp(-\frac{\kappa_0}{2}(\hat{\mu}_h - \mu_0)^T\Sigma^{-1}(\hat{\mu}_h - \mu_0))\Big\}\times \\
&\Big[2^{\nu_0 d/2}\pi^{d(d-1)/4}\prod_{i=1}^{d}\Gamma(\frac{\nu_0+1-i}{2})\Big]^{-1}|\Lambda_0|^{\nu_0/2}|\Sigma|^{-(\nu_0+d+1)/2}exp(-\frac{1}{2}tr(\Lambda_0\Sigma^{-1}))d\Sigma \\
&\propto |\Lambda_0|^{\nu_0/2}\int |\Sigma|^{-(\nu_0+d+1+K)/2}exp(-\frac{\kappa_0}{2}\sum_{h=1}^{K}(\hat{\mu}_h - \mu_0)^T\Sigma^{-1}(\hat{\mu}_h - \mu_0) - \frac{1}{2}tr(\Lambda_0\Sigma^{-1}))d\Sigma \\
&= |\Lambda_0|^{\nu_0/2}\int |\Sigma|^{-(\nu_0+d+1+K)/2}exp\Big\{-\frac{1}{2}tr\Big(\big[\kappa_0\cdot\sum_{h=1}^{K}(\hat{\mu}_h - \mu_0)(\hat{\mu}_h - \mu_0)^T + \Lambda_0\big]\Sigma^{-1}\Big)\Big\}d\Sigma \\
&\propto \frac{|\Lambda_0|^{\nu_0/2}}{|B+\Lambda_0|^{\frac{\nu_0+K}{2}}}
\end{aligned}$$

where $B = \kappa_0 \cdot \sum_{h=1}^{K}(\hat{\mu}_h - \mu_0)(\hat{\mu}_h - \mu_0)^T$. Substituting the MLE $\hat{\mu}_0$ for μ_0 and maximizing the likelihood function $L(\mu_0, \Lambda_0)$, we obtain the following maximum likelihood estimator for

Λ_0:

$$\hat{\Lambda}_0 = \frac{\nu_0 \kappa_0}{K} \sum_{h=1}^{K} (\hat{\mu}_h - \hat{\mu}_0)(\hat{\mu}_h - \hat{\mu}_0)^T$$

4 Discussion

We have proposed a new generic MCMC algorithm that works well in many cases. In the simulations we considered, we obtained promising results. The introduction of DPM to a large extent solved some problems intrinsic to the EM algorithm. However, such new method also presents some new theoretical challenges. DPM's sensitivity to the prior hyperparameters urges us to find new theoretical justifications for good priors. In this paper, we have proposed some possible approaches such as MLE. More future research could be directed to new approaches in finding good priors for hyperparameters such as ν_0 and κ_0. Another possible approach is parallel computing. The idea is to run several DPM clustering algorithms at the same time and form IM proposal density independently. The subsequent IM transition kernel will be weighted among all the IM proposal densities. The goal is to increase the weights of the IM kernels with high acceptance rate and decrease the weights of the kernels with low acceptance rate as the simulation process goes on. Such ideas will be further explored in future research work.

Acknowledgments I cannot possibly overstate my gratitude to my supervisors Jeffrey Rosenthal and Radu Craiu for their guidance throughout this Summer. This research experience is substantial to my study at University of Toronto. We were always working together as a team. When our ideas worked out, we were all excited about them. When our ideas fail, we encouraged each other. After all, research is an enjoyment from the process of trial and error. Thanks to my supervisors, I appreciate the essence of research, which brings me joy.

A R Code for Parallel Tempering and IM Sampler

```
#Function g is used to define the target density at all temperature
#It is used by the parallel tempered MCMC algorithm. It is valid only
#if target density pi has already been defined.

g = function(x, thetemp, maxtemp) {
  if ( (thetemp<1) || (thetemp>maxtemp) )
    return(0.0)
  else
    return( pi(x)^(1/thetemp) )
}

#ptmcmc is one of the major functions used by RCA. It is used to sample
#from any target density. It has superior performance over IM sampler when
#little is known about the geography of the target density. However it is
#rather slow when maxtemp, the number of parallel chains is large.
#dim is the dimension of the target density, L specifies the range to start
#the chain when initial value of the chain in unknown. size is the size of
#the sample to obtain, burn is the size of the burn-in sample. maxtemp is
#the number of parallel chains to use. It has better performance when
#maxtemp is larger, but it would simultaneously be much slower when maxtemp
#is large. numofswitch is the number of switching proposals each systematic
#scan. When maxtemp is large, numofswitch should accordingly be larger to
#speed up the mixing. X is the initial position of the chain. notice that
#X should be of length dim*maxtemp. It tells us the position of all parallel
#chains. This function will also automatically adjust proposal scalings.

#It will return the samples from the target density, xlist, without burn-in
#samples. It will also return the last position of the coupled chains, X,
#which can be passed on to the next function call.

ptmcmc=function(dim, L,  size, burn, maxtemp, numofswitch, X=runif(dim*maxtemp,-L,L),
                         scalings = rep(1,maxtemp),adjscaling =FALSE, dotempering = TRUE)
{
  #a matrix xlist that keeps track of chain values
  xlist = matrix( rep(0,size*maxtemp*dim), ncol=maxtemp*dim )

  #numxaccept is a vector that keeps track of the number of acceptance
  #for each of the parallel chain.
  numxaccept = rep(0,maxtemp)
  numtempaccept = 0

 for (i in 1:size)
 {
  for (temp in 1:maxtemp)
  {

    #temp is the number of chain that we are keeping track of
    # PROPOSED X[temp] MOVE

    # proposal value
    Y = X[((temp-1)*dim+1):(temp*dim)] + scalings[temp] * rmvnorm(1,rep(0,dim),diag(rep(1,dim)))
```

```
  #the proposal scaling depends on which chain we are moving

  U = runif(1)  # for accept/reject
  A = log(g(Y,temp,maxtemp)) - log( g(X[((temp-1)*dim+1):(temp*dim)],temp,maxtemp))  # for accept/reject
  if (log(U) < A)
  {
  X[((temp-1)*dim+1):(temp*dim)] = Y  # accept proposal
      numxaccept[temp] = numxaccept[temp] + 1;
  }
}
if (dotempering)
{
  for (rounds in (1:numofswitch))
  {

    j = floor(1+runif(1,0,maxtemp))  # uniform on {1,2,...,maxtemp}
    k = floor(1+runif(1,0,maxtemp))  # uniform on {1,2,...,maxtemp}

    U = runif(1)  # for accept/reject
    A = log(g(X[((j-1)*dim+1):(j*dim)],k,maxtemp))+
    log( g(X[((k-1)*dim+1):(k*dim)],j,maxtemp)) -
    log( g(X[((j-1)*dim+1):(j*dim)],j,maxtemp))-
    log( g(X[((k-1)*dim+1):(k*dim)],k,maxtemp));
    if (log(U) < A)
    {
  # accept proposed swap
  tmpval = X[((j-1)*dim+1):(j*dim)];
  X[((j-1)*dim+1):(j*dim)] = X[((k-1)*dim+1):(k*dim)];
  X[((k-1)*dim+1):(k*dim)] = tmpval;
      numtempaccept = numtempaccept + 1;
    }
  }
}
xlist[i,]=X
if (adjscaling)
{
#every 50 iterations as a batch, we try to update the value
#of the proposal scalings
if ((i%%50)==0)
{
  for (temp in 1:maxtemp)
  {
    accrate = (numxaccept[temp]/i)
    #cat("iteration",i,"for chain number",temp,"acceptance rate",accrate,"\n")
    if (accrate > 0.5)
    {
      #acceptance rate too high increase proposal scaling
      scalings[temp] = scalings[temp] + 0.2
    }
    if (accrate < 0.5)
    {
      #acceptance rate too low decrease proposal scaling
      scalings[temp] = max((scalings[temp] - 0.2),0.1)
    }
  }
```

```
  }
  }

 }
 return(list(xlist=((xlist[(burn+1):size,])[,(1:dim)]),X=xlist[size,]))
}

#Function guess is used by the IM sampler to construct the pdf of our guessed proposal
#distribution. We can construct the density of any Gaussian mixture model
#given the mixing proportion p, list of mean vectors, mu, and list of
#covariance matrices, sigma.

guess=function(x,p,mu,sigma)
{
  sum = 0
  K=length(p)
  for (i in 1:K)
  {
    sum = sum + p[i]*dmvnorm(x,mu[[i]],sigma[[i]])
  }
  return(sum)
}

#propose function is used by the IM sampler to propose from our guessed density.
#It can give you samples of any Gaussian mixture model given mu, sigma and
#p as indicated above.

propose=function(MU,SIGMA,P)
{
v=runif(1)
      sum = 0
      K = length(P)

      for (i in (1:K))
      {
        sum = sum + P[i]
        if (sum >= v)
        {
          #we shall sample from ith component
          break
        }
      }

return(rez=rmvnorm(1,MU[[i]],SIGMA[[i]]))
}

#Function imsampler is the IM sampler that we are using after
#we fit a Gaussian mixture model to the observed samples. Its acceptance
#rate tells us how close our proposal distribution is to the target density
#so it will be our adapatation parameter.

#X.now is the position of the main chain.
#p,mu,sigma are our current guess of the target density via Gaussian
#mixture model.
```

```
imsampler <- function (dim, size, X.now, p, mu, sigma)
{

  #We use IM sampler to sample from the target density

  numacc = 0
  xlist = matrix(rep(0,size*dim),size,dim)

  for (i in 1:size)
  {
    Y = propose(mu,sigma,p)
    A = log(pi(Y))-log(pi(X.now))+log(guess(X.now,p,mu,sigma))-log(guess(Y,p,mu,sigma))
    U = runif(1)
    if (log(U)<A)
    {
# accept proposal
X.now = Y
numacc = numacc + 1
    }
    xlist[i,] = X.now
  }

  acceptrate = numacc/size
  cat("the IM sampler has acceptance rate",acceptrate,"\n")
  return(list(imsamples = xlist, accrate = acceptrate))
}
```

B R Code of EM Algorithm for Gaussian Mixture Model

```
# EM ALGORITHM FOR SIMPLE D-DIMENSIONAL GAUSSIAN MIXTURE MODEL.

# The arguments are the data matrix x but now each data point
# is assumed to be D-dimensional. The program will automatically
# figure out the dimension of the input by checking number
# columns of x, the number of components to use, K, the
# number of EM iterations to do,iters,
# and a matrix of initial responsibilities
# of components for data items (default is a random assignment).

# The value returned is a list with elements pi (probabilities of components),
# mu is now a list that keeps track of all the mean vectors of the components
# sigma now is a list that keeps track of all the covariance matrices
# of all the components, and
# r, the matrix of responsibilities (which could be passed to another call
# of mix.em).

#This EM algorithm will automatically stop when we have 4 indentical log probability
#values consecutively.
# We need to assume here we use more than 1 Gaussian component.

mix.em <- function (x, K, iters,
                    r = matrix(runif(K*nrow(x),0.1,1),nrow(x),K))
```

```
{
  N <- nrow(x)
  D <- ncol(x)
  checking = rep(1,iters)

  if (nrow(r)!=N || ncol(r)!=K)
  { stop("Matrix of responsibilities is the wrong size")
  }

  # Make sure initial responsibilities sum to one.

  for (i in 1:N)
  { r[i,] <- r[i,] / sum(r[i,])
  }

  # Do EM iterations, starting with M step, then E step.

  #initialize mu and sigma
  mu <- list()
  sigma <- list()

  for (t in 1:iters)
  {
    # cat("\niteration:",t)
    # cat("\nr:\n")
    # print(round(r,3))
    if ((t>5)&&(checking[t-1]<0) && (checking[t-2]<0) && (checking[t-3]<0) && (checking[t-4]<0) &&
    (round(checking[t-1],6) == round(checking[t-2],6)) && (round(checking[t-2],6) == round(checking[t-3],6)) &&
    (round(checking[t-3],6) == round(checking[t-4],6)))
    {
      break
    }

    rt <- colSums(r)

    # M step.

    pi <- rt / sum(rt)
    for (k in 1:K)
    {
      #initialize the mean vector
      init = rep(0,D)
      #initialize the covariance matrix
      init2 = matrix(0,D,D)
      for (n2 in 1:N)
      {
        init = init + (x[n2,]*r[n2,k])
      }
      sum0 = sum(r[,k])
      init = init/sum0
      for (n3 in 1:N)
      {
        init2 = init2 + ((x[n3,]-init) %*% (t(x[n3,]-init))*r[n3,k])
      }
      init2 = init2/sum0
```

```
      if (abs(det(init2)) < (10^(-10)))
      {
        init2 = init2 + diag(rep(epsilon,D))
      }

      mu[[k]] = init
      sigma[[k]] = init2

      #add epsilon * I to the covariance matrix to make sure it is not singular
    }

    # compute the log likelihood after each iteration
    # *Assume here we use more than 1 Gaussian component*
    init3 = dmvnorm(x,mu[[1]],sigma[[1]])
    for (k2 in 2:K)
    {
      init3 = cbind(init3,dmvnorm(x,mu[[k2]],sigma[[k2]]))
    }
    #print(init3)
    #cat("iteration",t,"ncol of init3 is",ncol(init3),"\n")
    log.like = sum(log(init3 %*% pi))

    cat("\niteration number",t,"log likelihood of the data set",log.like,"\n")
    checking[t] = log.like

    # E step.

    for (num2 in 1:N)
    {
      init.res = rep(0,K)
      for (k3 in 1:K)
      {
        init.res[k3] = pi[k3] * dmvnorm(x[num2,],mu[[k3]],sigma[[k3]])
      }
      r[num2,] = init.res/sum(init.res)
    }
    # cat("\nupdated r:\n")
    # print(round(r,3))
  }
  #cat("\niteration number",t,"log likelihood of the data set",log.like,"\n")
  list (p=pi, mu=mu, sigma=sigma, r=r, checking=checking)
}
```

C R Code of DPM for Gaussian Mixture Model

```
#DPM in the univariate case.

#We use SUGS to cluster the data points.
#Create a vector that classifies the data points.

size = length(sample)
v0 = rep(0,size)
```

```
#The mixing proportions naturally follow the symmetric Dirichlet distribution
#with parameter alpha. Here we take alpha to be 0.1. We put
#Normal-inverse-chi-square distribution on the parameter space of the component
#distribution. The prior is N-Inv-Chi^2(1,1/0.1;3,1).

alpha= 0.1
mu0 =1
sigma0sq = 1
kappa0 = 0.1
nu0 = 3

#The function k.index gives us up to index i, the number of Gaussian components
#that are used.

k.index=function(i)
{
  return (max(v0[1:i]))
}

#First, we use the first Gaussian component.
v0[1]=1

#Now i is starting from 2.

#pi.ih is a function that will return an array of prior cluster probabilities.
#h goes from 1 to k(i-1)+1.
pi.ih=function(i)
{
  max = k.index(i-1)
  #create an array
  result = rep(-1,(max+1))

  for (h in (1:max))
  {
    result[h] = sum(v0[1:(i-1)]==h)/(alpha+i-1)
  }
  result[(max+1)] = alpha/(alpha+i-1)
  return(result)
}

#L.ih is a function that will return an array of conditional likelihood of yi
#given allocation to cluster h. #h goes from 1 to k(i-1)+1.
L.ih=function(i,yi)
{
  max = k.index(i-1)
  #create an array
  result = rep(NA,(max+1))

  #For all the old components. At least one previous data point is
  #classified to belong to the component.

  for (h in (1:max))
  {
    init= c()
```

```
for (g in 1:(i-1))
{
   if (v0[g] == h)
   {
      init = c(init,sample[g])
   }
}
post1 = init
n = length(post1)

kappa.n=kappa0+n
nu.n = nu0+n
ybar.n = mean(post1)
if (n==1)
{
  ssq.n = 0
}
if (n>1)
{
  ssq.n = sd(post1)^2
}
nu.n.sigma.n.sq = nu0*sigma0sq+(n-1)*ssq.n+((kappa0*n)/(kappa0+n))*(ybar.n-mu0)^2

kappa = kappa0 + n +1
nu = nu0 +n+1
combine = c(post1,yi)
ybar = mean(combine)
ssq = sd(combine)^2
nusigmasq = nu0*sigma0sq+n*ssq+((kappa0*(n+1))/(kappa0+n+1))*(ybar-mu0)^2

part1 = sqrt(kappa.n/(2*pi*kappa))
part2 = (nu.n.sigma.n.sq/nusigmasq)^(nu.n/2)
part3 = 1/(sqrt(nusigmasq/2))

if (nu.n%%2==0)
{
  initprod = 1
  for (mm in (1:(nu.n/2-1)))
  {
    initprod = initprod*((mm+0.5)/mm)
  }
  part4 = initprod*0.5*sqrt(pi)
}

if (nu.n%%2==1)
{
  initprod = 1
  for (mm in (1:((nu.n-1)/2)))
  {
    initprod = initprod*(mm/(mm-0.5))
  }
  part4 = initprod/sqrt(pi)
}

result[h] = (part1*part2*part3*part4)
```

```
  }

  #for the new component
  kappanew = kappa0 + 1
  nunew = nu0+1
  nusigmasqnew = nu0*sigma0sq + (kappa0/(kappa0+1))*(yi-mu0)^2
  numernew = gamma(nunew/2)*sqrt(kappa0)*(sigma0sq*nu0/2)^(nu0/2)
  denonew = sqrt(2*pi)*gamma(nu0/2)*sqrt(kappanew)*(nusigmasqnew/2)^(nunew/2)
  result[max+1] = numernew/denonew
  return(result)
}

comp=function(i,yi)
{
  result1 = pi.ih(i)
  result2 = L.ih(i,yi)
  result = result1 * result2
  result = result/sum(result)
  index0 = order(-result)[1]
  #cat("result is,",result,"\n")
  return(index0)
}

#We go through a loop, using comp function to cluster each data point sequentially.
for (i in (2:size))
{
  rez = comp(i,sample[i])[1]
  v0[i] = rez
}

#DPM in the multivariate case

#All functions in the univariate case remain functioning the same except for L.ih function.
#Code for DPM in the multivariate case is simply replacing the L.ih function in the univariate case by the following.
#Still use the comp function provided in the univariate case to cluster data points.

#gamma function returns the following value: gamma((a+1)/2)/gamma(a/2), a must be an integer.
gammaquo = function(a)
{
  #when a is even
    if (a%%2==0)
    {
      initprod = 1
      for (mm in (1:(a/2-1)))
      {
        initprod = initprod*((mm+0.5)/mm)
      }
      res = initprod*0.5*sqrt(pi)
    }

    if (a%%2==1)
    {
      initprod = 1
      for (mm in (1:((a-1)/2)))
```

```
      {
        initprod = initprod*(mm/(mm-0.5))
      }
      res = initprod/sqrt(pi)
    }
    return(res)
}

#L.ih is a function that will return an array of conditional likelihood of yi
#given allocation to cluster h. #h goes from 1 to k(i-1)+1.
L.ih=function(i,yi)
{
  max = k.index(i-1)
  #create an array
  result = rep(NA,(max+1))

  #For all the old components. At least one previous data point is
  #classified to belong to the component.

  for (h in (1:max))
  {
    init= c()
    for (g in 1:(i-1))
    {
       if (v0[g] == h)
       {
          init = rbind(init,sample[g,])
       }
    }
    post1 = init
    #cat("post1 is",post1,"\n")
    #return(post1)

    n = nrow(post1)

    kappa.n=kappa0+n
    nu.n = nu0+n

    ybar.n = colMeans(post1)


    if (n>1)
    {
      temp.n = post1[1,]-ybar.n
      S.n = temp.n %*% t(temp.n)

      for (pp in (2:n))
      {
         temp.n = post1[pp,]-ybar.n
         S.n = S.n + temp.n %*% t(temp.n)
      }

    }

    if (n==1)
```

```
  {
    lambda.n = lambda0+(kappa0/(kappa0+1))*((ybar.n-mu0)%*%t((ybar.n-mu0)))
  }

  if (n>1)
  {
    lambda.n = lambda0+S.n+(kappa0*n/(kappa0+n))*((ybar.n-mu0)%*%t((ybar.n-mu0)))
  }


  kappa = kappa0 + n +1
  nu = nu0 +n+1
  combine = rbind(post1,yi)
  ybar = colMeans(combine)


  temp = combine[1,]-ybar
  S = temp %*% t(temp)

  for (pp in (2:(n+1)))
  {
     temp = combine[pp,]-ybar
     S = S + temp %*% t(temp)
  }

  lambda = lambda0+S+(kappa0*(n+1)/(kappa0+n+1))*((ybar-mu0)%*%t((ybar-mu0)))


  part1 = 1/((pi)^(dim0/2))
  part2 = (kappa.n/kappa)^(dim0/2)
  part3 = 1/(det(lambda)^(1/2))
  part4 = (det(lambda.n)/det(lambda))^(nu.n/2)

  part5 = 1
  for (dd in 1:dim0)
  {
    part5 = part5*gammaquo((nu.n+1-dd))
  }

  result[h] = (part1*part2*part3*part4*part5)

}

#for the new component
kappanew = kappa0 + 1
nunew = nu0+1
lambdanew = lambda0+(kappa0/(kappa0+1))*((yi-mu0)%*%t((yi-mu0)))

part1 = (kappa0/(pi*kappanew))^(dim0/2)*(det(lambda0)^(nu0/2))/(det(lambdanew)^(nunew/2))
part2 = 1
for (dd in 1:dim0)
{
    part2 = part5*gammaquo((nu0+1-dd))
}
```

```
  result[max+1] = part1*part2
  return(result)
}
```

D R Code of Bayesian Inference for Prior DP Precision Parameter

```
#DPM - Bayesian Inference for Prior DP Precision Parameter

#Only the prior belief on alpha is changed. All functions remain the same except for prior clustering
#probabilities pi.ih. In this case, alpha does not take a specific value. Instead we create a vector
#alpha.star that specifies all possible values for alpha.

alpha.star = c(0.01,0.05,0.1,0.2,0.5,1,2,4)
T = length(alpha.star)

#Phi is a matrix that keeps track of the posterior distribution of alpha after each clustering result.
Phi = matrix(NA,sizesample,T)

#initialize phi_m^{(1)}
Phi[1,] = rep(1/T,T)

#The new pi.ih is defined as follows:
#pi.ih is a function that will return an array of prior cluster probabilities.
#h goes from 1 to k(i-1)+1. m is the index of the alpha to use, which goes from
#1 to T.

pi.ihm=function(i)
{
  max = k.index(i-1)
  matrix0 = matrix(NA,T,(max+1))
  for (mm in (1:T))
  {
    alpha = alpha.star[mm]
    #create an array
    result = rep(-1,(max+1))

    for (h in (1:max))
    {
      result[h] = sum(v0[1:(i-1)]==h)/(alpha+i-1)
    }
    result[(max+1)] = alpha/(alpha+i-1)
    matrix0[mm,] = result
  }
  return(matrix0)
}
```

References

[1] Aitkin, M. (2001) Likelihood and Bayesian analysis of mixtures. *Statistical Modelling*, 1: 287-304.

[2] Craiu, R. Research Talk: Recent Advances in Regional Adaptation for MCMC. – Adapskiii, Utah, 2011. *Available on the author's website,* http://www.utstat.toronto.edu/craiu/Talks/index.html

[3] Gelman, A., Carlin, J., Stern, H.,Rubin, D. *Bayesian Data Analysis.* second edition, Boca Raton, Florida: Chapman and Hall/CRC, 2004.

[4] Giordani, P., Robert, K. (2010) Adaptive independent Metropolis-Hastings by fast estimation of mixtures of normals. *J. Comput. Graph. Statist.*, 19, to appear.

[5] Hastings, W. K. (1970) Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57, 97-109.

[6] McAuliffe, J., Blei, D., Jordan, M. (2006) Nonparametric empirical Bayes for the Dirichlet process mixture model. *Stat. Comput.*, 16: 514.

[7] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E. (1953) Equations of state calculations by fast computing machines. *J. Chem. Phys.*, 21, 1087-1091.

[8] Neal, R. Course notes: Bayesian Methods for Machine Learning, Spring 2011. *Available on the author's website,* http://www.cs.utoronto.ca/ radford/csc2541/

[9] Neal, R. Course notes: Statistical Methods for Machine Learning and Data Mining, Spring 2011. *Available on the author's website,* http://www.utstat.utoronto.ca/ radford/sta414/

[10] Roberts, G. O., Rosenthal, J. S. (2007) Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *J. Appl. Probab.*, 44 458-475.

[11] Rosenthal, J. S. (2010) Optimal Proposal Distributions and Adaptive MCMC. *Chapter for MCMC Handbook, S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, eds.*

[12] Rosenthal, J. S. (2004) General state space Markov chains and MCMC algorithms. *Probability Surveys*, 1: 20-71.

[13] Rosenthal, J. S. Course notes: Monte Carlo Methods, Spring 2011. *Available on the author's website,* http://www.probability.ca/jeff/teaching/1011/sta3431/

[14] Tibshirani, R., Walther, G., Hastie, T. (2001) Estimating the number of clusters in a data set via the gap statistic. *J. R. Statist. Soc. B*, 63, Part 2, 411-423.

[15] Wang, L., Dunson, D. (2010) Fast Bayesian Inference in Dirichlet Process Mixture Models. *Journal of Comp. and Graphical Stats.*, 1–21.