STAT 5444: Homework #MCMC

Due on 12/5/2016

Professor Scott Leman

Kevin Malhotra

Problem 1

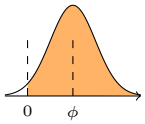
$$
\begin{aligned}
&Assume: \pi(x)g(x'|x) \leq \pi(x')g(x|x') \\
&x \to x' \\
&\alpha_{MH}(x'|x) = Min\left(1, \frac{\pi(x')}{\pi(x)}\frac{g(x|x')}{g(x'|x)}\right) = 1 \\
&p(x'|x) = g(x'|x)\alpha_{MH}(x'|x) = g(x'|x) \\
&x' \to x \\
&\alpha_{MH}(x'|x) = Min\left(1, \frac{\pi(x)}{\pi(x')}\frac{g(x'|x)}{g(x|x')}\right) = \frac{\pi(x)}{\pi(x')}\frac{g(x'|x)}{g(x|x')} \\
&p(x|x') = g(x|x')\alpha_{MH}(x|x') = g(x|x')\frac{\pi(x)}{\pi(x')}\frac{g(x'|x)}{g(x|x')} \\
&p(x|x')p(x') = p(x'|x)p(x) \\
&g(x|x')\frac{\pi(x)}{\pi(x')}\frac{g(x'|x)}{g(x|x')}p(x') = g(x'|x)p(x) \\
&\frac{\pi(x)}{\pi(x')}\frac{g(x'|x)}{g(x|x')} = \frac{p(x)}{p(x')}
\end{aligned}
$$

This solution describes that both steps of the transitions end up being symmetric which concludes that the $\pi(x)$ is the stationary distribution for this Markov chain.

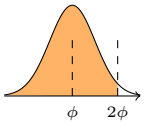
Problem 2

$$
\begin{aligned}
\alpha_{MH} &= \left(1, \frac{\pi(\theta^*)}{\pi(\theta)}\frac{g(\theta|\theta^*)}{g(\theta^*|\theta)}\right) \\
\alpha_{MH} &= Min\left(1, \frac{\pi(\mu^*\phi^*)}{\pi(\mu\phi)}\frac{g(\mu,\phi|\mu^*\phi^*)}{g(\mu^*\phi^*|\mu,\phi)}\right) \\
\alpha_{MH} &= Min\left(1, \frac{\pi(\mu^*\phi^*)}{\pi(\mu\phi)}\frac{g(\mu|\mu^*)g(\phi|\phi^*)}{g(\mu^*|\mu)g(\phi^*|\phi)}\right) \\
\alpha_{MH} &= Min\left(1, \frac{\phi^{*(\frac{n}{2}-1)}exp(-\frac{\phi^*}{2}\sum_{i=1}^{N}(x_i-\mu^*)^2)}{\phi^{(\frac{n}{2}-1)}exp(-\frac{\phi}{2}\sum_{i=1}^{N}(x_i-\mu)^2)}\frac{exp(\frac{-1}{2}(\mu-\mu^*)^2)}{exp(\frac{-1}{2}\mu^*-\mu)^2)}\frac{g(\phi|\phi^*)}{g(\phi^*|\phi)}\right) \\
\alpha_{MH} &= Min\left(1, \frac{\phi^{*(\frac{n}{2}-1)}exp(-\frac{\phi^*}{2}\sum_{i=1}^{N}(x_i-\mu^*)^2)}{\phi^{(\frac{n}{2}-1)}exp(-\frac{\phi}{2}\sum_{i=1}^{N}(x_i-\mu)^2)}\frac{exp(\frac{-1}{2}(\phi-\phi^*)^2)}{\Phi(\phi)}\frac{\Phi(\phi^*)}{exp(\frac{-1}{2}(\phi-\phi^*)^2)}\right) \\
\alpha_{MH} &= Min\left(1, \frac{\phi^{*(\frac{n}{2}-1)}exp(-\frac{\phi^*}{2}\sum_{i=1}^{N}(x_i-\mu^*)^2)}{\phi^{(\frac{n}{2}-1)}exp(-\frac{\phi}{2}\sum_{i=1}^{N}(x_i-\mu)^2)}\frac{\Phi(\phi^*)}{\Phi(\phi)}\right)
\end{aligned}
$$

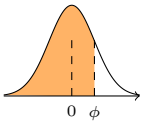
Problem 2 continued on next page...



(a) Figure 1



(b) Figure 2



(c) Figure 3

Figure 1: CDF Plots

$\Phi(\phi)$ is the cdf up to some phi. The reason we need to apply this normalization is that since we are in a truncated space these normalizations will not cancel out and will affect the final target distribution. The CDFs functions in matlab require the CDF calculations to be transformed in a specific manner. The first plot above is the target area we want to cover which is equivalent to the second plot because they cover the same space since we cover half of the area + an area spanning ϕ. The second and third plots are equivalent since normal distributions are shift invariant so we can just scale the center. Thus the final distribution follows a standard normal cdf which is $\int_{-\infty}^{\phi} e^{\frac{-1}{2}x} dx$

$\sigma_{\phi}^2 = 1$

$\sigma_{\mu}^2 = 1000$

$\mu_0 = 0$

$\phi_0 = 5$

$For \rightarrow t = 2 : T$

$1.\mu * -N(\mu_{t-1}, \sigma_{\mu}^2)$

$2.\phi * -N(\phi_{t-1}, \sigma_{\phi}^2)$

$$3.\alpha_{MH} = Min\left(1, \frac{\phi^{*(\frac{n}{2}-1)} exp(-\frac{\phi^*}{2}\sum_{i=1}^{N}(x_i - \mu^*)^2)}{\phi_{t-1}^{(\frac{n}{2}-1)} exp(-\frac{\phi_{t-1}}{2}\sum_{i=1}^{N}(x_i - \mu_{t-1})^2)} \frac{\frac{\Phi(\phi^*)}{\sigma_{\phi}^2}}{\frac{\Phi(\phi_{t-1})}{\sigma_{\phi}^2}}\right)$$

$$4.\pi(\mu, \phi) = \phi^{*(\frac{n}{2}-1)} exp(-\frac{\phi}{2}\sum_{i=1}^{N}(x_i - \mu)^2)\Phi(\phi)$$

$$5.if(log(unif(0,1) \leq \left[log(\pi(\mu^*, \phi^*)) - log(\phi(\mu_{t-1}, \phi_{t-1}))\right])Lemma : (\frac{\pi(\mu^*, \phi^*)}{\pi(\mu_{t-1}, \phi_{t-1})} \leq unif(0,1) < 1)$$

$\mu_t, \phi_t = \mu^*, \phi^*$

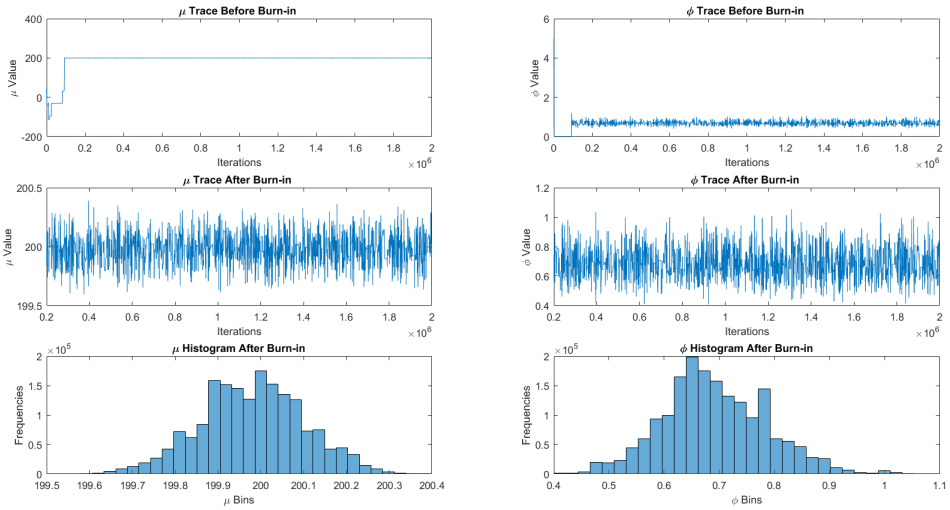
$else$

$\mu_t, \phi_t = \mu_{t-1}, \phi_{t-1}$

end

end

Problem 2 continued on next page...



Burn-in Time = 200 Iterations

```
% Generate the Data
N = 100;
X = normrnd(200, sqrt(2), [N, 1]);
Xbar = mean(X);

T = 2000000;
Burnin = 200;
mu = zeros(T, 1);
mu(1) = 0;
phi = zeros(T, 1);
phi(1) = 5;
proposeVarMu = 1000;
proposeVarPhi = 1;
temp = 0;
for t=2:T
    %disp(t)
    mustar = normrnd(mu(t-1), sqrt(proposeVarMu));
    phistar = -1;
    while phistar < 0
        phistar = normrnd(phi(t-1), sqrt(proposeVarPhi));
    end
    %disp(phistar)
    %pistar = phistar^(N/2 - 1)*exp((-phistar/2)*(sum(X.^2) - 2*mustar*N*Xbar + N*mustar.^2)) * normco
    %pit = phi(t-1)^(N/2 - 1)    *exp((-phi(t-1)/2)*(sum(X.^2) - 2*mu(t-1)*N*Xbar + N*mu(t-1).^2)) * no
    pistar = log(phistar)*(N/2 - 1) + (-phistar/2)*(sum(X.^2) - 2*mustar*N*Xbar + N*mustar.^2) + log(p
    pit = log(phi(t-1))*(N/2 - 1)  + (-phi(t-1)/2)*(sum(X.^2) - 2*mu(t-1)*N*Xbar + N*mu(t-1).^2) + log
    %disp([pistar, pit])
    if (log(rand) <= (pistar - pit))
        mu(t) = mustar;
        phi(t) = phistar;
```

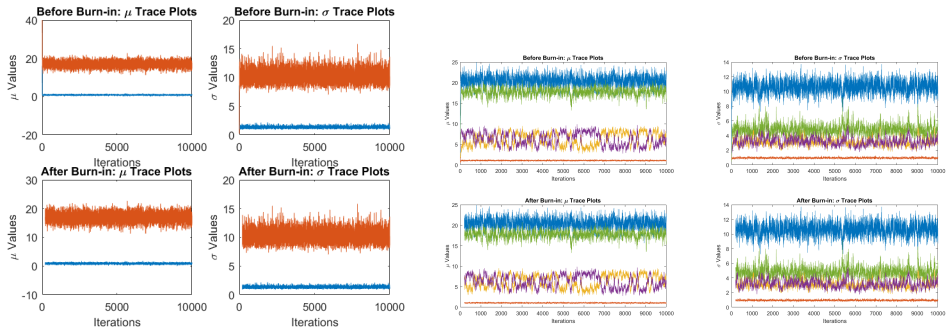
```
    else
        mu(t) = mu(t-1);
        phi(t)= phi(t-1);
    end
    disp(mu(t))
end
subplot(3,2,1)
plot(1:T, mu(1:T));
xlabel('Iterations');ylabel('\mu Value'); title('\mu Trace Before Burn-in ');
subplot(3,2,2)
plot(1:T, phi(1:T))
xlabel('Iterations');ylabel('\phi Value'); title('\phi Trace Before Burn-in ');
subplot(3,2,3)
plot(Burnin:T, mu(Burnin:T));
xlabel('Iterations');ylabel('\mu Value'); title('\mu Trace After Burn-in ');
subplot(3,2,4)
disp(Burnin)
plot(Burnin:T, phi(Burnin:T))
xlabel('Iterations');ylabel('\phi Value'); title('\phi Trace After Burn-in ');
subplot(3,2,5)
histogram(mu(Burnin:T), 30);
xlabel('\mu Bins'); ylabel('Frequencies'); title('\mu Histogram After Burn-in');
subplot(3,2,6)
histogram(phi(Burnin:T), 30);
xlabel('\phi Bins'); ylabel('Frequencies'); title('\phi Histogram After Burn-in');
```

Problem 3

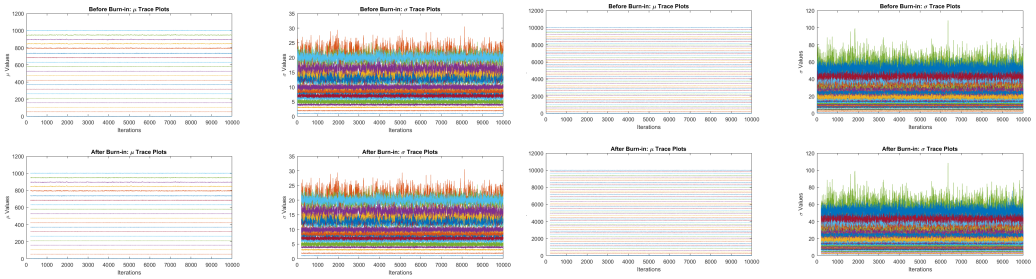
```
K = 2;
N = 100;
p = ones(K,1)*(1/K);
sampleMu = linspace(1, 15, K);
sampleSigma = linspace(1, 10, K); %[1.5, 2.25, 1.75, 2.5, 2.75];
sampSpace = mnrnd(N, p, 1);
mixedModel = cell(K, 1);
for k=1:K
    mixedModel{k} = normrnd(sampleMu(k), sampleSigma(k), sampSpace(k), 1);
end
mixedModel = cell2mat(mixedModel);
mixedModel = mixedModel(randperm(N));

T = 10000;
Burnin = 200;
thetaj = zeros(T, K, 2); % Iterations x Modes x [mu, phi]
thetaj(1, :, 1) = linspace(1, 25, K) + 15; %1:K; %[1, 2, 3, 4, 5]; % mu init
thetaj(1, :, 2) = linspace(1, 10, K) + 5; %1:K; %[1, 2, 3, 4, 5]; % phi init

% Generate Cluster Assignments
clusterI = mnrnd(1, p, N)>0;
clusters = zeros(T, N);
for k=1:K
    clusters(1, clusterI(:, k)) = k;
end

for t=2:T
    % Step 1: Generate New Cluster Assignments with proposed mu's/phi's
    % Generate logPDFs for the entire dataset
    logPDFs = zeros(K, N);
    for k=1:K
       logPDFs(k, :) = log(normpdf(mixedModel, thetaj(t-1, k, 1), sqrt(1/thetaj(t-1, k, 2))));
    end
    maxPDFs = max(logPDFs,[], 1);
    % log pi_max + log(sum(e^(log pi_max - log pi_i))
    marginalModes = maxPDFs' + log(sum(exp(bsxfun(@minus, logPDFs', maxPDFs')), 2));
    % prob(C_i = j) = exp(log piPDFs - log marginal) probability for a
    % given sample to be in any cluster
    clusterProb = exp(bsxfun(@minus, logPDFs', marginalModes));
    % To provide stability and clearing up round off errors for mnrnd
    clusterProb(:, end) = 1 - sum(clusterProb(:, 1:end-1), 2);
    % For a given sample acquire the probabilities and categorically pick
    % them and perform an assignment for that sample.
    clusterI = mnrnd(1, clusterProb)>0;
    for k=1:K
       clusters(t, clusterI(:, k)) = k;
    end

    % Step 2: Update the mu's/phi's with proposed cluster assignments
    for k =1:K
```

```
        % Acquire the clustered data for a given cluster
        currX = mixedModel(clusters(t, :) == k);
        if length(currX) == 0
            disp([t, length(currX)])
        end
        % Updating the Mu/Phi's for the given Cluster
        thetaj(t, k, 1) = normrnd(mean(currX), sqrt(1/(length(currX)*thetaj(t-1, k, 2))));
        thetaj(t, k, 2) = gamrnd(length(currX)/2, 2/(sum((currX - thetaj(t, k, 1)).^2)));
    end
end
subplot(2,2,1)
plot(1:T, thetaj(1:T, :, 1))
xlabel('Iterations'); ylabel('\mu Values'); title('Before Burn-in: \mu Trace Plots')
subplot(2,2,2)
plot(1:T, sqrt(1./thetaj(1:T, :, 2)))
xlabel('Iterations'); ylabel('\sigma Values'); title('Before Burn-in: \sigma Trace Plots')
subplot(2,2,3)
plot(Burnin:T, thetaj(Burnin:T, :, 1))
xlabel('Iterations'); ylabel('\mu Values'); title('After Burn-in: \mu Trace Plots')
subplot(2,2,4)
plot(Burnin:T, sqrt(1./thetaj(Burnin:T, :, 2)))
xlabel('Iterations'); ylabel('\sigma Values'); title('After Burn-in: \sigma Trace Plots')
```

K = 2, K = 5



K = 20, K = 50



Problem 4

Introduction:

Gaussian Mixture Models provide the ability to take an arbitrary set of data and generate a target distribution through Monte Carlo Markov Processes. This process works when we know how many modes or components need to be modeled to approach the target distribution. For a given component there is a mean, standard deviation, and weighting for the Gaussian. Each weighting is defined as the probability or a given data point to be in a specific cluster. Therefore, every single data point has a probability to be in each component. Given these constraints the appropriate assignments to data points must be made to generate the Gaussian Mixture Model to fit the dataset. For the problem, simulations must be performed with different number of components, a random data set that will be generated, and hyper parameters tuning to understand Gaussian Mixture Models.

Procedure:

The following problem was approached by using Gibbs Sampling Procedure, a Monte Carlo Markov Process, to converge towards the solution. The first problem to solve is how to generate the data. Given the number of data points that need to be generated a Multinomial distribution is used with equal weights to determine how many samples are coming from each Gaussian distribution. The data is randomized and stored in an entire vector. Initialization of all the parameters for each component is done by setting each components parameters skewed away from the mean by some arbitrary amount. This tuning is changed throughout the testing and it affects the convergence of the algorithm.

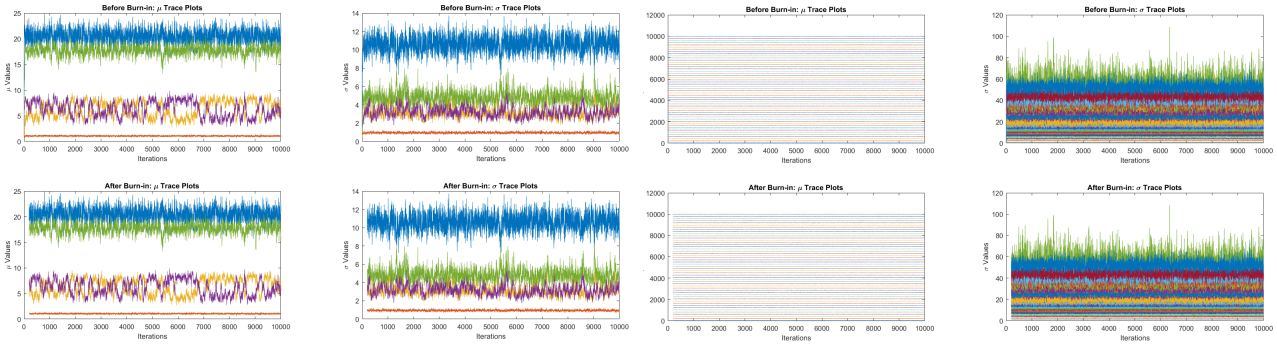
The algorithm is split up into two sections. The first section focusing on keeping the Gaussian component parameters the same and updating the data point assignments. The second section focusing on keeping the data point assignments the same and updating the Gaussian component parameters. The first step is done by acquiring the conditional probability for a data point to belong in a cluster. The equation formulates to be $P(C_i = j|x_i, \theta_1, ...\theta_J) = \frac{P(x_i|\theta_j, C_i=j)P(C_i=j)}{\sum_{i=1}^{J} P(x_i|\theta_j, C_i=j)P(C_i=j)}$ where C_i refers to the current component and θ_j refers to the component's parameters. This is sampled for every possible component and data point. This will generate a NxJ matrix of probabilities. In the algorithm, everything is computed in log form for numerical stability and vectorized for speed. For a given data point and it's J probabilities for the J respective components a cluster assignment must be made. The cluster assignment is done by a Multinomial(Categorical since 1 sample is needed) which generates an assignment based on the probabilities. The assignments are updated for every data point and the second step must be performed.

The second step requires using the posterior distributions for the Gaussian parameters. The Jeffrey's Priors are used for $\mu \propto 1$ and $\phi \propto \frac{1}{\phi}$. The likelihood function chosen are the standard Normal Distribution for μ and Gamma Distribution for ϕ. These generate the posterior distributions $\mu_j - N(\bar{x}^j, \frac{1}{\phi_j N_j})$ and $\phi_j - Gamma(\frac{N_j}{2}, \frac{\sum_{i=1}^{N}(x_i^j - \mu_j)^2}{2})$. The j subscripts represent the subset of the entire datasets component assignments. As a result of the first step, the Gaussian components are updated by using the component assignments to sample from the posterior distribution for a new Gaussian components. These updated parameters will be used in the next step to update the cluster assignments. This process will lead to the convergence and oscillations around the ground truth.

Problem 4 continued on next page...

Simulations Results:

After performing all the simulations across J = 2, 5, 20, 50. I have come across a few results during simulation. When two means of each component are close together they begin to oscillate around the two means, there is a certain threshold to this. I was able to get the true means within 2 units of each other when the components would begin to oscillate around the two means. One of the other issues facing convergence was having the correct number of points for all the components. If there were not enough points for each component then the model would also fail to converge and may miss an component completely. As, I kept increasing the component values, these two major cases had to be in mind when performing initialization. Without the appropriate hyper parameter tuning for each component the model would not converge; therefore, smart initialization is required to tweak the sampler to generate the correct distribution.



The first image contains the issue of oscillating between two different means. The second image is the 50 component model converging towards the solution. As the variance of each component increased so did the oscillations of that variable when it was converging towards the solution.

Numerical Stability:

Numerical Stability was faced throughout the implementation of the entire implementation. The probability equation was converted to log form. The following equation must be put into log form and it required a few derivations to get it completely correct $P(C_i = j|x_i, \theta_1, ...\theta_J) = \frac{P(x_i|\theta_j, C_i=j)P(C_i=j)}{\sum_{i=1}^{J} P(x_i|\theta_j, C_i=j)P(C_i=j)}$. The final derivation for the denominator is $log(\sum_{j=1}^{J} e^{log(\pi_{max}(x))-log(\pi_j(x))})$. This final form was used to to retain numerical stability.

Conclusions:

The sampler is extremely good at performing and it is relatively fast. As I increased the components the data points also had to be increased to allow for a good mass for each component. This caused an increased complexity and it remained near linear time in terms of performance. The sampler converges in relatively small iterations but it is run longer to see the oscillation changes. The biggest issues faced were the numerical stability problems and the figuring out that the number of points was a factor of convergence. Matlab also has different gamma functions which made it difficult to find numerical errors. Overall, the sampler is relatively fast and vectorized and it is clear and works without numerical problems.