SDS386D: Monte Carlo Methods in Statistics **Cesar N. Yahia**
Homework 4 Prof. S.G. Walker

consider the model $g(x|\theta) = wN(X|\mu_1, \sigma^2) + (1-w)N(x|\mu_2, \sigma^2)$
with priors on $\lambda = 1/\sigma^2$ sa Ga(1,1), $f(\mu_1) = N(0, 100)$ and $f(\mu_2) = N(0, 100)$ and $f(w)$ is a uniform prior on the interval (0,1).

1.

Generate data of size $n = 100$ from a standard normal distribution and run the Markov chain Gibbs sampler for the above model using the data you obtained.

First let us start by defining the conditionals needed for running the Markov chain Gibbs sampler.

Since the likelihood for a specific data point involves a summation, to make sampling the conditionals easier we introduce a latent variable that makes the likelihood a bunch of product terms across data points. The latent variable we can introduce is $d = 1, 2$ that specifies if the model from which the data comes from as either $N(X|\mu_1, \sigma^2)$ for $d = 1$ or $N(x|\mu_2, \sigma^2)$ for $d = 2$.

The new model with the latent variable is now
$f(x, d|\theta) = w_d N(x|\mu_d, \sigma^2)$.
By summing out the latent variable we can see that the original model is retrieved.

The resulting likelihood is
$L(\theta; x) = P(data|\theta) = \Pi_{i=1}^{n} w_{d_i} N(x_i|\mu_{d_i}, \sigma^2)$
where $n = 100$ and the i is an index for the data points.

The priors on θ are as given above, so let us start getting the Gibbs conditionals:
$f(\mu_1|d_i, ..., d_n, data, \sigma^2) \propto f(\mu_1) e^{-1/2 \sum_A (x_i - \mu_1)^2/\sigma^2} \propto e^{-\mu_1^2/2(100)} e^{-1/2 \sum_A (x_i - \mu_1)^2/\sigma^2}$
$\propto N(\mu_1 | \frac{\lambda n_A \bar{x}}{n_A \lambda + \frac{1}{100}}, \frac{1}{\frac{1}{100} + n_A \lambda})$ *conjugate prior, or just multiply through to get this*
where A is the set that refers to summing only data points with $d_i = 1$. n_A is the number of data points in A
Here we can see that the latent variable d_i isolates the parameter μ_1 from μ_2 and w since the terms in the likelihood corresponding to data points with $d_i = 2$ do not depend on μ_1 and are constants disappearing in the proportionality sign of the μ_1 posterior. Had we not introduced the latent variable, the μ_1 posterior would depend on μ_2 and w through a nasty product of sums.

Similarly, the posterior for μ_2 can be obtained as
$f(\mu_2|d_i, ..., d_n, data, \sigma^2) \propto f(\mu_2) e^{-1/2 \sum_B (x_i - \mu_2)^2/\sigma^2} \propto e^{-\mu_2^2/2(100)} e^{-1/2 \sum_B (x_i - \mu_2)^2/\sigma^2}$
$\propto N(\mu_2 | \frac{\lambda n_B \bar{x}}{n_B \lambda + \frac{1}{100}}, \frac{1}{\frac{1}{100} + n_B \lambda})$
where B corresponds to data points with $d = 2$ and the average $\bar{x}$ is over such data points, and n_B is the number of data points in B

Given the values of d, the posterior for w is isolated from everything else.
$f(w|d_i, ..., d_n) \propto f(d_i, ..., d_n|w) f(w) \propto w^{n_A}(1-w)^{n_B} f(w) \propto w^{n_A}(1-w)^{n_B}$
Where n_A is the number of data points in set A with $d = 1$, and n_B is the number of data points in set B with $d = 2$. This is a beta distribution with $\alpha = n_A + 1$ and $\beta = n_B + 1$

As for the precision λ, for a normal likelihood, we know that the Gamma distribution is a conjugate prior. The posterior is given by
$f(\lambda|\mu_1, \mu_2, data, d_i, ..., d_n) = Ga(.|n/2 + 1, 1 + 1/2 \sum_A (x_i - \mu_1)^2 + 1/2 \sum_B (x_i - \mu_2)^2)$

Remaining is a posterior for the latent variable d. We note that the latent variable links all the different parameters together, so it will depend on everything.

Since d can take 2 possible values $\{1, 2\}$, it has a Bernoulli distribution.
$P(d_i = 1|x_i, \mu_1, \mu_2, w, \lambda) = \frac{wN(x_i|\mu_1,\sigma^2)}{wN(x_i|\mu_1,\sigma^2)+(1-w)N(x_i|\mu_2,\sigma^2)}$
Following the conditionals above, the Gibbs sampler algorithm is as follows:

Algorithm:
(1) start with arbitrary initial values for λ, μ_1, μ_2, and w from a reasonable domain
(2) sample d_i for all i from a Bernoulli with $P(d_i = 1|x_i, \mu_1, \mu_2, w, \lambda) = \frac{wN(x_i|\mu_1,\sigma^2)}{wN(x_i|\mu_1,\sigma^2)+(1-w)N(x_i|\mu_2,\sigma^2)}$
(3) sample λ from $f(\lambda|\mu_1, \mu_2, data, d_i, ..., d_n)$
(4) sample μ_1 from $f(\mu_1|d_i, ..., d_n, data, \sigma^2)$
(5) sample μ_2 from $f(\mu_2|d_i, ..., d_n, data, \sigma^2)$
(6) sample w from $f(w|d_i, ..., d_n)$
(7) repeat from step 2

The above algorithm was implemented in Python as follows:

```
import numpy as np
import numpy.random as nprand
import matplotlib.pyplot as plt
import scipy.stats as stats


# define a sample from f(d|...)
def sample_d(x_i, mew_1, mew_2, w, lam):
        """
        samples the Bernoulli conditional for di
        :param x_i: data point
        :param mew_1: mean of one mode
        :param mew_2: mean of another mode
        :param w: probability of data point from mode
        :param lam: lambda (precision)
        :return:
        """
        st_dev = 1.0 / np.sqrt(lam)
        term_1 = w * stats.norm(mew_1, st_dev).pdf(x_i)
        term_2 = (1 - w) * stats.norm(mew_2, st_dev).pdf(x_i)
        prob_d_equal_1 = term_1 / (term_1 + term_2)
        di = nprand.binomial(1, prob_d_equal_1)
        return di
```

```
# define a sample from f(lambda|...)
def sample_lambda(a, b, mew_1, mew_2, data, labels):
        """
        sample lambda from the corresponding conditional
        distribution
        :param a: prior parameter
        :param b: prior parameter
        :param mew_1: mean of first mode
        :param mew_2: mean of second mode
        :param data: list of data points sampled
        :param labels: list of generated labels for data points
        :return: new_lambda
        """
        n = len(data)
        new_a = a + float(n) / 2
        data_mean_1 = list()
        data_mean_2 = list()
        for key, label in enumerate(labels):
        if label == 0:
        data_mean_1.append(data[key])
        elif label == 1:
        data_mean_2.append(data[key])
        sum_squares_1 = sum([(data_point - mew_1)**2 for data_point in data_mean
        sum_squares_2 = sum([(data_point - mew_2)**2 for data_point in data_mean
        new_b = b + (1.0 / 2) * sum_squares_1 + (1.0 / 2) * sum_squares_2
        scale = 1.0 / new_b
        new_lambda = nprand.gamma(new_a, scale=scale)
        return new_lambda


# sampling from posterior for mew_1 f(mew_1|..)
def sample_mew_1(labels, data, lam):
        if 0 not in labels:
        sample = nprand.normal(loc=0, scale=10)
        else:
        data_mean_1 = list()
        for key, label in enumerate(labels):
        if label == 0:
        data_mean_1.append(data[key])
        nA = len(data_mean_1)
        mean_data_1 = sum(data_mean_1) / len(data_mean_1)
        new_mean = (lam * nA * mean_data_1) / ((nA * lam) + (1.0 / 100))
        new_var = 1.0 / ((1.0 / 100) + (nA * lam))
        new_std = np.sqrt(new_var)
        sample = nprand.normal(loc=new_mean, scale=new_std)
        return sample
```

```
# sampling from posterior for mew_2 f(mew_2|..)
def sample_mew_2(labels, data, lam):
        if 1 not in labels:
        sample = nprand.normal(loc=0, scale=10)
        else:
        data_mean_2 = list()
        for key, label in enumerate(labels):
        if label == 1:
        data_mean_2.append(data[key])

        nB = len(data_mean_2)
        mean_data_2 = sum(data_mean_2) / len(data_mean_2)
        new_mean = (lam * nB * mean_data_2) / ((nB * lam) + (1.0 / 100))
        new_var = 1.0 / ((1.0 / 100) + (nB * lam))
        new_std = np.sqrt(new_var)
        sample = nprand.normal(loc=new_mean, scale=new_std)
        return sample


# sample from f(w|..)
def sample_w(labels):
        nA = 0
        nB = 0
        for label in labels:
        if label == 0:
        nA += 1
        elif label == 1:
        nB += 1
        alpha = nA + 1
        beta = nB + 1
        sample = nprand.beta(alpha, beta)
        return sample


# evaluate the density
def mix_model(x, w, mew_1, mew_2, lam):
        var = 1.0 / lam
        std_dev = np.sqrt(var)
        g = w * stats.norm(mew_1, std_dev).pdf(x) + (1 - w) * stats.norm(mew_2,
        return g


# implement sampling
if __name__ == '__main__':
        lam = 1  # initial values
        mew_1 = 0
        mew_2 = 0
        w = 0.5
```

```
# prior params for precision
a = 1
b = 1

# generate data points
data = list(nprand.normal(0, 1, 100))

samples_mew1 = list()  # store samples mew_1
samples_mew2 = list()  # store samples mew_2
samples_w = list()  # store samples_w
samples_lam = list()  # store samples lam

numofiter = 1
total_iter = 2000

labels = [0]*100
while numofiter <= total_iter:
        for key, val in enumerate(data):
                labels[key] = sample_d(val, mew_1, mew_2, w, lam)
        lam = sample_lambda(a, b, mew_1, mew_2, data, labels)
        mew_1 = sample_mew_1(labels, data, lam)
        mew_2 = sample_mew_2(labels, data, lam)
        w = sample_w(labels)
        samples_mew1.append(mew_1)
        samples_mew2.append(mew_2)
        samples_w.append(w)
        samples_lam.append(lam)
        numofiter += 1

        print("... plotting samples of mu 1 ...")
        # plot a graph showing distribution
        plt.hist(samples_mew1, bins=50, facecolor='g', normed=True)
        plt.title("generated samples for \mu_1")
        plt.xlabel("samples")
        plt.ylabel("normalized frequency")
        plt.tight_layout()
        plt.savefig('samples_mu1.png')
        plt.show()

print("... plotting samples of mu 2 ...")
# plot a graph showing distribution
plt.hist(samples_mew2, bins=50, facecolor='g', normed=True)
plt.title("generated samples for \mu_2")
plt.xlabel("samples")
plt.ylabel("normalized frequency")
plt.tight_layout()
plt.savefig('samples_mu2.png')
plt.show()
```

```
print("... plotting samples of lambda ...")
# plot a graph showing distribution
plt.hist(samples_lam, bins=50, facecolor='g', normed=True)
plt.title("generated samples for lambda")
plt.xlabel("samples")
plt.ylabel("normalized frequency")
plt.tight_layout()
plt.savefig('samples_lambda.png')
plt.show()

print("... plotting samples from w ...")
plt.hist(samples_w, bins=50, facecolor='g', normed=True)
plt.title("generated samples for w")
plt.xlabel("samples")
plt.ylabel("normalized frequency of w samples")
plt.tight_layout()
plt.savefig('samples_w.png')
plt.show()
```

Note that if there are no data points belonging to a mode, the prior is sampled for the mean conditional.

The resulting samples are plotted as follows:

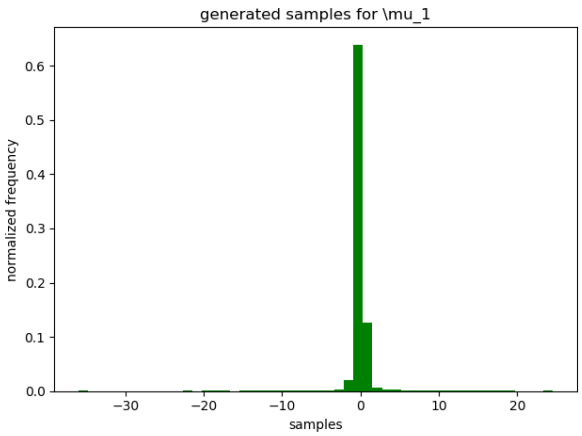

Figure 1: Histogram showing normalized frequency for μ_1 samples

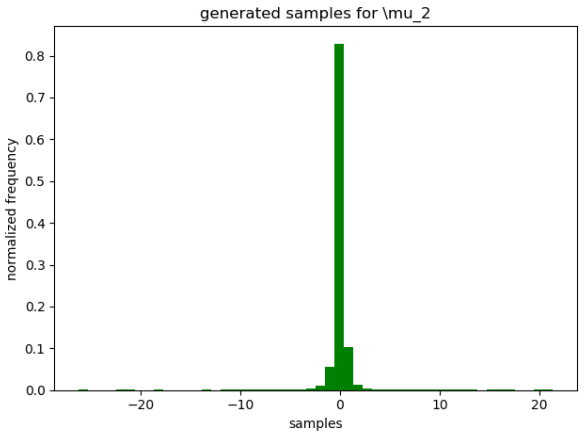

Figure 2: Histogram showing normalized frequency for μ_2 samples

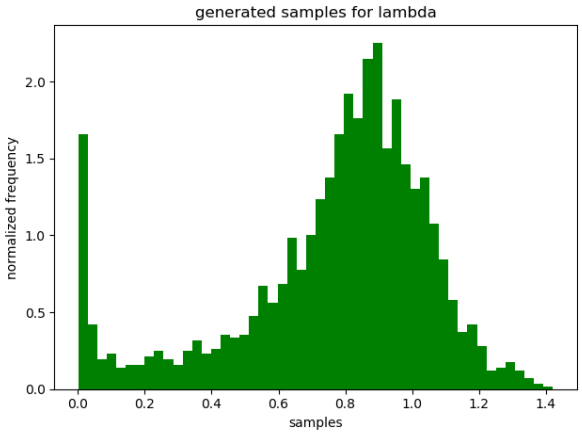

Figure 3: Histogram showing normalized frequency for λ samples

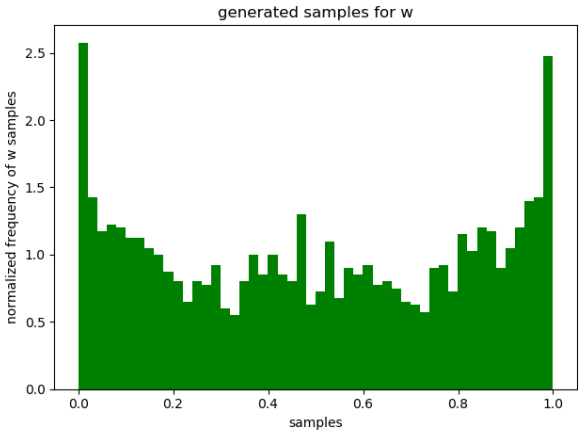

Figure 4: Histogram showing normalized frequency for w samples

2.

Plot the predictive density; tis is done by taking the average of the densities you get at each iteration.
At each iteration of the Markov chain, we can take the parameters θ_i, plug them into $g(x|\theta_i)$ and get the density at that iteration.

The predictive density $g_p(x) = \frac{\sum_i g(x|\theta_i)}{M}$
To plot a density evaluate it's values at many points. Take the points between -5 and 5. Then, evaluate standard normal density and averaged density across iterations for those data points. The averaged density is obtained by calculating the probability of a point using parameters at a current iteration, then averaging that with subsequent probabilities obtained at the next iterations with updated parameters.

The following code was added or modified

```
def mix_model(x, w, mew_1, mew_2, lam):
        var = 1.0 / lam
        std_dev = np.sqrt(var)
        g = w * stats.norm(mew_1, std_dev).pdf(x)
         + (1 - w) * stats.norm(mew_2, std_dev).pdf(x)
        return g

        # implement sampling
        if __name__ == '__main__':
                lam = 1  # initial values
                mew_1 = 0
                mew_2 = 0
                w = 0.5

                # prior params for precision
                a = 1
                b = 1

                # generate data points
                data = list(nprand.normal(0, 1, 100))

                samples_mew1 = list()  # store samples mew_1
                samples_mew2 = list()  # store samples mew_2
                samples_w = list()  # store samples_w
                samples_lam = list()  # store samples lam

                numofiter = 1
                total_iter = 2000

                labels = [0]*100

                density_evaluate = list(np.arange(-5, 5.1, 0.05))
                predictive_density = [0]*len(density_evaluate)
```

```
while numofiter <= total_iter:
        for key, val in enumerate(data):
                labels[key] = sample_d(val, mew_1,
                 mew_2, w, lam)
        lam = sample_lambda(a, b, mew_1, mew_2,
        data, labels)
        mew_1 = sample_mew_1(labels, data, lam)
        mew_2 = sample_mew_2(labels, data, lam)
        w = sample_w(labels)
        samples_mew1.append(mew_1)
        samples_mew2.append(mew_2)
        samples_w.append(w)
        samples_lam.append(lam)
        for key, item in enumerate(density_evaluate):
                predictive_density[key] =
                (predictive_density[key]*(numofiter
                - 1) + mix_model(item, w, mew_1, mew_2,
                 / numofiter

        numofiter += 1

print("... print predictive density ...")
standard_norm = [stats.norm(0, 1).pdf(x) for x
in density_evaluate]
plt.plot(density_evaluate,
 predictive_density)
plt.plot(density_evaluate,
 standard_norm, '--', label='standard_normal')
plt.title("predictive density vs. standard normal")
plt.xlabel("x")
plt.ylabel("density")
plt.legend(loc='upper right')
plt.tight_layout()
plt.savefig('pred_std.png')
plt.show()
```

Plotting the predictive density over a standard normal, the results, are as shown below. We can see that the mixture model gives one mode, and aligns well with the standard normal.

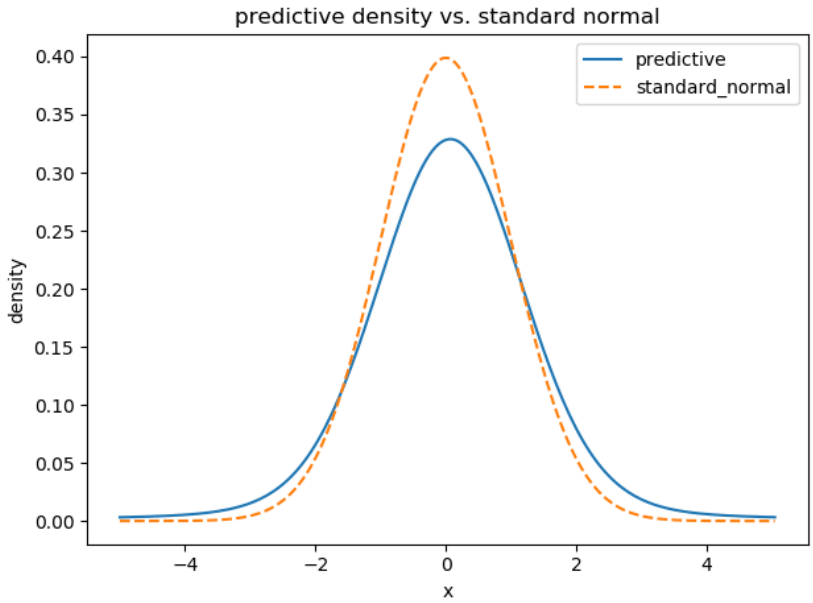

Figure 5: predictive density vs. standard normal

Another way to generate the predictive density $g_p(x)$ without the restriction of calculating probabilities over a specific domain is to use the law of total probability which specifies that $g_p(x) = \int g(x|\theta)f(\theta|x)d\theta$. Therefore, we can sample $g_p(x)$ by for sampling x for every θ at each iteration. Then, by plotting a normalized histogram of the generated x samples, we will have the shape of $g_p(x)$.

This was easily implemented in the code above by sampling the mixture model for the given parameters θ_i at iteration i. To do so, we sample a uniform random variable between (0,1). If the sampled value is less than w then sample x from the first component of the mixture, otherwise sample x from the second component of the mixture.

The generated predictive density is shown below.

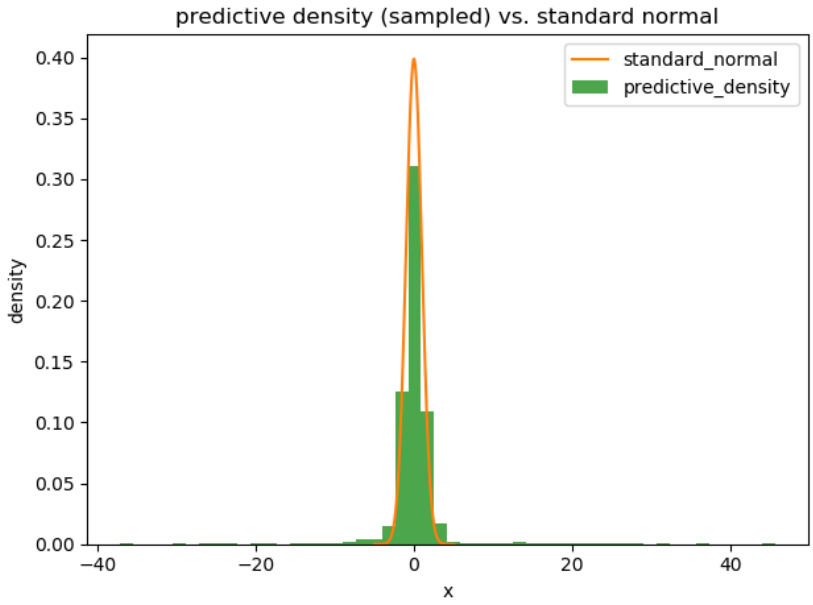

Figure 6: predictive density vs. standard normal

This looks the same as the predictive density from the former approach, except it is clear that the domain of the density in this case is not restricted to (-5,5) (the region we chose to average probabilities over earlier).

3.

The posteriors were plotted in question 1.
An indication that the density is coming from a standard normal is the posterior for the means. Specifically, the posteriors for μ_1 and μ_2 are both centered around zero as shown in Figure 1 and Figure 2.

Moreover, the posterior for λ has a high concentration around 1 as shown in Figure 3. This also indicates the true density may be standard normal.

As for the posterior for w, this is the probability of data coming from $N(.|\mu_1, \sigma^2)$. Since, the posterior for w has a very large variance, this implies that the probability that the data comes from one normal or the other is not influenced strongly by the data. This indicates that the data could be coming from one mode, and hence there is no need for distinction between two distinct modes using w. Mathematically, we note that if we have one mode $N(.|\mu, \sigma^2)$, then a linear combination given by $g(x|\theta) = wN(X|\mu, \sigma^2) + (1-w)N(x|\mu, \sigma^2)$ is the same as $N(.|\mu, \sigma^2)$ by adding the two terms (so w can be anything as shown in the Figure 4)

4.

Yes, in many of the iterations the labels are oscillating between a vectors of all 1's or all 2's indicating that all the data points belong to one mode. This happens for several iterations. Many other iterations have that most labels (if not all) are either 1 or 2.

5.

This integral is just the expected value. To evaluate the integral using Monte Carlo, you can sample the predictive density and sum x.

$\hat{I} = \frac{\sum x_i}{n}$ where x_i are sampled from $g_p(x)$

To sample the predictive density, we note that the CDF of the predictive density is also the average of the CDFs for normals from varying parameters across iterations. We can average the CDFs for data points in the same manner as we did to obtain the predictive density. Therefore, one approach is to get the CDF for all the data points between -5 and 5, where we consider that approximately all our data would lie, and then sample a uniform random variable. If the sampled uniform variable is between the CDFs of a and b then take a as your sample. This method is referred to as inverse transform sampling.

Another approach is to directly use the sampled x_i's when $g_p(x)$ was obtained using the law of total probability. Here all we need to do is sum the x_i's and divide by the total number of sampled values.

Since the latter approach does not involve extra work for sampling the averaged predictive density, we use it to evaluate the integral.

The estimator $\hat{I}$ after 2000 iterations was found to be 0.06 which is close to true value (zero). The figure below shows the convergence of the estimator across iterations.

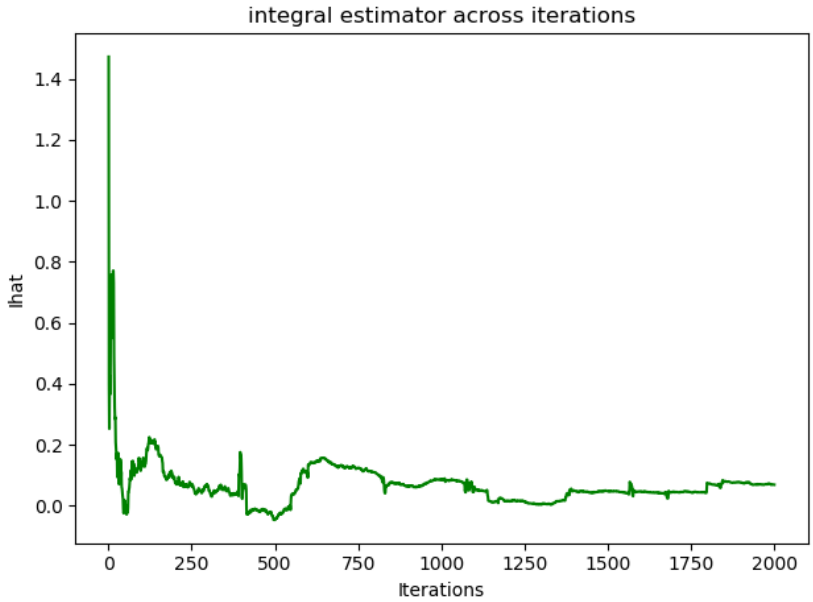

Figure 7: convergence of estimator across iterations

6.

To estimate $w\mu_1 + (1-w)\mu_2$ from the Markov chain, we can take the average of the collected w, μ_1 and μ_2 samples. The plug the averages in $w\mu_1 + (1-w)\mu_2$ to get the estimate.

The equation $w\mu_1 + (1-w)\mu_2$ should also give the expected value of the predictive density (which is what was obtained in question 5). The results indicate that the estimate for $w\mu_1 + (1-w)\mu_2$ was 0.08 after 2000 iterations, which is close to the true expected value zero. However, this value could not be trusted well due to poor convergence of the estimator. The figure below shows the estimator across iterations.

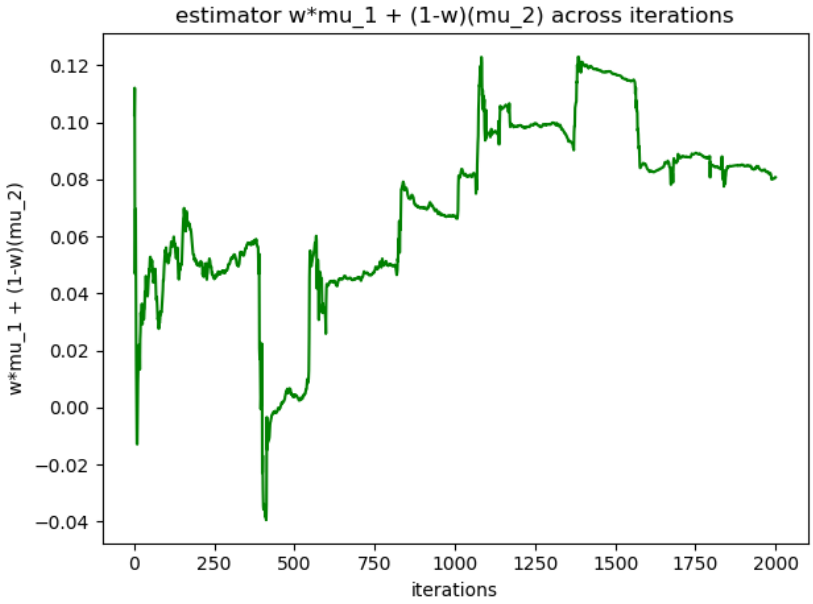

Figure 8: convergence of estimator across iterations

As shown, the estimator is oscillating a lot across iterations. This is happening due to identifiability problems. Specifically, the label for any particular data point keeps switching between one mode and the other. For example, the normalized frequency of the labels for the 10th data point is shown below.

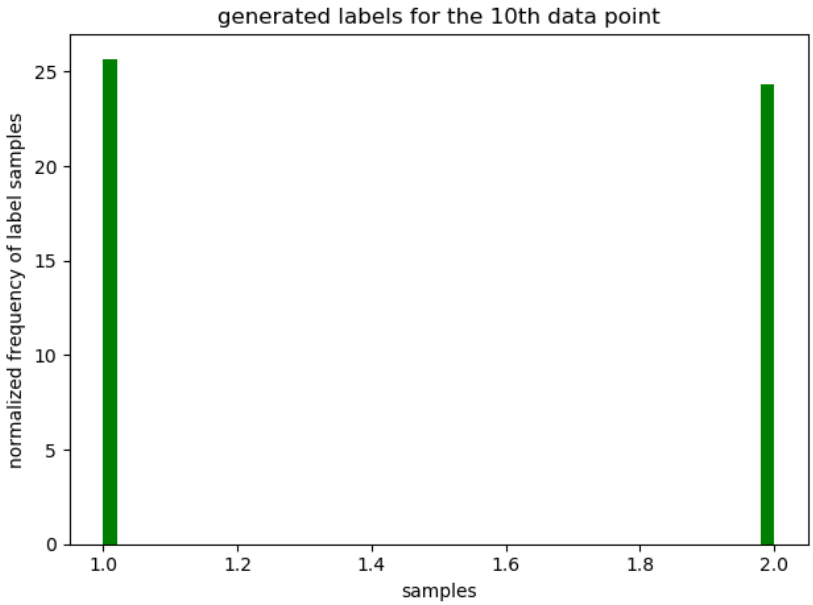

Figure 9: normalized frequency of label for 10th data point

Almost half of the time, the data point belongs to one mode or the other. Therefore, μ for any mode is influence by a different set of data points across iterations.

Also, by observing Figure 4, we can see that w can be 0 or 1, when this happens the mean for one of the modes can take arbitrarily any value without changing the predictive density at that iteration. Therefore, when this arbitrary value is averaged with subsequent samples of μ, this will create oscillations in the estimator.