Evaluating EM Algorithm Performance Against a Mixture of Normal Distributions

Joseph Free

Department of Mathematics, University of North Florida

April 13, 2020

Abstract

In this report the EM algorithm's performance at estimating the mixture coefficients for a two-component normal mixture model is considered. To fully examine the versatility of the algorithm in this context, multiple samples from four different classes of mixtures are generated and EM used to estimate the mixture coefficients of each. Namely, these mixtures are made of normal components with 1) similar means and variances, 2) similar means and dissimilar variances, 3) dissimilar means and similar variances, and 4) completely dissimilar means and variances.

1 Introduction

The Expectation-Maximization (EM) Algorithm is a popular numerical tool for applied statistical problems. The method was developed by Dempster, Laird, and Rubin [DLR77] as a means for obtaining approximate solutions to difficult likelihood equations. Since its inception, the EM algorithm has undergone many modifications and is used in a number of specific applications including, but not limited to, obtaining approximate maximum likelihood estimators and data imputation (approximating missing values) [EH16]. A particular benefit of the EM algorithm is that the conditions for convergence have been established, and it is known that each step of the algorithm monotonically increases the empirical likelihood function [CB01].

The idea of the EM algorithm is fairly intuitive. The crux of the method is to replace a difficult likelihood maximization problem with a sequence of easier maximizations that tend to a limit that solves the original problem. The sequence in question is developed through the two primary steps of the algorithm: the **E-step** (expectation) and the **M-step** (maximization). During the E-step, the conditional expectation of the log-likelihood (or parameters of interest) is found; and during the M-step, the conditional log-likelihood is maximized. This process it iterated repeatedly, using the parameter values found in the M-step to feed into the following E-step.

2 Problem

In this report, we wish to use the EM algorithm to obtain the maximum likelihood estimate of the mixture coefficient for a normal mixture model with two components. That is, given $Y \sim ZN(\mu_1, \sigma_1^2) + (1 - Z)N(\mu_2, \sigma_2^2)$, where $Z \in \{0, 1\}$ and $P(Z = 1) = \pi$, we wish to find the MLE of π.

The probability density of Y can be expressed as a traditional mixture: $g(y) = \pi f_1(y) + (1 - \pi)f_2(y)$, where $f_1 \sim N(\mu_1, \sigma_1^2)$ and $f_2 \sim N(\mu_2, \sigma_2^2)$. Given n observations from Y, we can find the log-likelihood to be:

$$\ell(\boldsymbol{\theta}|\mathbf{y}) = \sum_{i=1}^{n} log[\pi f_1(y_i) + (1 - \pi)f_2(y_i)]$$

where $\boldsymbol{\theta} = (\pi, \mu_1, \mu_2, \sigma_1^2, \sigma_2^2)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. However this form of the likelihood is analytically intractable because of the sum inside of the logarithm. So another approach must be taken.

A more expedient route involves incorporating the latent variables Z given above. These variables only take on the values 0 and 1 (and hence are Bernoulli distributed). Consider Z_i for observation i. If $Z_i = 1$, then y_i is an observation from a normal distribution with f_1 as its density with probability π. Likewise, if $Z_i = 0$, then y_i comes from the normal with density f_2 with probability $1 - \pi$. Using these latent variables, we can re-express the density of the mixture random variable Y as [HTF01]

$$g(y) = (\pi f_1(y))^Z [(1 - \pi)f_2(y)]^{1-Z}.$$

With this density we can procedure to find the likelihood and corresponding log-likelihood, after which we have:

$$\ell(\boldsymbol{\theta}|\mathbf{y}, Z) = \sum_{i=1}^{n} \left[Z_i log f_1(y_i) + (1 - Z_i) log f_2(y_i) \right] + \sum_{i=1}^{n} \left[Z_i log(\pi) + (1 - Z_i) log(1 - \pi) \right]$$

then, because we wish to find the MLE for π, we can employ the **M-step** of the EM algorithm and differentiate with respect to π and set to zero to obtain our estimator. Doing so yields:

$$\frac{\partial \ell}{\partial \pi} = \sum_i \left(Z_i \frac{1}{\hat{\pi}} - (1 - Z_i) \frac{1}{1 - \hat{\pi}} \right) = 0 \implies \hat{\pi} = \frac{\sum_i Z_i}{\sum_i Z_i + \sum_i (1 - Z_i)} = \frac{\sum_i Z_i}{n}$$

Note that we can do the same for each of the location and scale parameters of the component normals: μ_1, μ_2, σ_1^2, or σ_2^2. If we were to do this, we would have the estimators

$$\hat{\mu_1} = \frac{\sum_i Z_i y_i}{\sum_i Z_i}$$
$$\hat{\mu_1} = \frac{\sum_i (1 - Z_i) y_i}{\sum_i (1 - Z_i)}$$
$$\hat{\sigma_1^2} = \frac{\sum_i Z_i (y_i - \hat{\mu_1})^2}{\sum_i Z_i}$$
$$\hat{\sigma_2^2} = \frac{\sum_i (1 - Z_i)(y_i - \hat{\mu_2})^2}{\sum_i (1 - Z_i)}$$

Now because Z is unobservable, it must be estimated. We do so by employing the **E-step** of the EM algorithm. That is, we estimate Z with its expected value given the observed data. Hence:

$$E[Z_i|\mathbf{y}] = P(Z = 1|\mathbf{y}) = \frac{P(Z = 1, \mathbf{y})}{g(\mathbf{y})} = \frac{\pi f_1(y_i)}{\pi f_1(y_i) + (1 - \pi) f_2(y_i)}.$$

Using this expectation, we can initialize a guess value for the parameter π, and iteratively update $\hat{\pi}$. The procedure is thus:

1. Initialize a guess value $\hat{\pi}_0$ (0.5 is as good as any)
2. Compute $\hat{Z}_i = \frac{\hat{\pi} f_1(y_i)}{\hat{\pi} f_1(y_i) + (1 - \hat{\pi}) f_2(y_i)}$ for each i
3. Update $\hat{pi}$
4. Go back 2 and repeat until the difference between estimates $\hat{\pi}_k$ and $\hat{\pi}_{k+1}$ is within specified tolerance.

For this report, we shall consider the location and scale parameters of the mixture components to be known. So they will not explicitly be updated or even considered in the EM algorithm. However it is a simple matter to incorporate them: in step 1, guesses are also initialized for $\hat{\mu_{10}}$, $\hat{\mu_{20}}$, $\hat{\sigma_{10}^2}$, and $\hat{\sigma_{20}^2}$; step 2 remains unchanged, and step 3 involves updating all the corresponding formulas for the estimates given above. The stopping criteria would then be given by negligible increases in the empirical log-likelihood at each iteration [BS13].

3 Method

The analysis was separated into four cases representing the possible types of mixture distributions one might encounter in practice. Depending on the nature of the scale and location parameters of the two component normals, the resulting density function of the mixture distribution will take on various types of shapes that the algorithm will necessarily have to accommodate. See Table 1 and Figure 1. For instance, if the means and variances of the two component normals are similar to one another, there will be an overlap of the two densities. The algorithm will then have to attempt to

Table 1: EM Algorithm Run Cases

Case	**Location**	**Scale**
1	Similar	Similar
2	Similar	Dissimilar
3	Dissimilar	Similar
4	Dissimilar	Dissimilar

differentiate between which observation resulted from which distribution, and at what proportion. Naturally this would be a much more demanding task than if the means were completely different from one another.

To assess the EM algorithm's performance in estimating the mixture coefficient for a two-component normal mixture, focus was placed on two criteria: speed of convergence and accuracy. First a random uniform (0,1) variate is generated for the mixture coefficient. Then, for each case given in table 1, 30 samples of $n = 1000$ observations from a mixture distribution are generated with normal components whose location and scale parameters are randomly perturbed from one another by either a small or large amount depending on the analysis case. For each sample, the proportion is estimated and number of iterations logged. The change in the iterates is then graphed against the number of iterations, and a histogram of deviations from the true mixture coefficient is given. The algorithm stops according to a Cauchy criteria when estimated mixture coefficient is within a 0.00001 tolerance level. That is, when $|\hat{pi}_{k+1} - \hat{pi}_k| < 0.00001$.

Finally, mean and median iteration times for each case are given. This process was run three times for three randomly generated mixture coefficients.

4 Results

4.1 Run #1

The results for the first run are summarized in Table 2, and Figures 2 and 3. Here the true mixture coefficient was generated to be $\pi = 0.0421$.

Table 2: Mean and Median Iterations for Run 1

	Case 1	Case 2	Case 3	Case 4
Mean	563.6	129.7	2	2.9
Median	276.0	105.0	2	2

From the table, it is apparent that the EM algorithm appears to have poor convergence properties for mixture distributions where the location parameters of the components are near one another. This much is also supported by Figure 2, where the panels for case 1 and case 2 indicate the algorithm's

sluggish evolution. Figure 3 summarizes the accuracy for each of the cases. Ostensibly for this choice of π, the algorithm is less accurate for cases 1 and 2, and quite accurate for cases 3 and 4.

4.2 Run #2

The second run results are given below in Table 3, and Figures 4 and 5. Here the true mixture coefficient was generated to be $\pi = 0.1527$.

Table 3: Mean and Median Iterations for Run 2

	Case 1	Case 2	Case 3	Case 4
Mean	550.3	95.5	2	2.9
Median	200	42	2	2

Here again, we see that as in run #1, the algorithm has an abysmal rate of convergence for mixtures with similar location parameters; but the reverse is the case for location parameters that are spaced far apart. Figure 5 too echos the results seen above: the algorithm yields less accurate estimates for cases 1 and 2, but quite good estimates for cases 3 and 4.

4.3 Run #3

For the third and final run, $\pi = 0.6450$. As before, Table 4 and Figure 6 and 7 contain the summary of results.

Table 4: Mean and Median Iterations for Run 3

	Case 1	Case 2	Case 3	Case 4
Mean	413.9	24.1	2	3.3
Median	134.5	19	2	2

Once more, insofar as the rate of convergence is concerned, the algorithm is quite slow to converge for cases 1 and 2. This information is also conveyed further in Figure 6, from which it is also apparent that the behavior of estimates for case 1 behave quite erratically. Figure 7 additionally bears out the pattern seen from runs #1 and #2. The estimates yields for cases 1 and 2 are fairly inaccurate, whereas those for case 3 and 4 are a significant improvement on this.

5 Conclusion

Based on the previous analysis, it is clear that the EM algorithm performs very poorly for mixture distributions where the components display significant overlap – that is, where the location parameters are close to one another. This much is evidenced not only by the number of iterations taken to

converge, but also the spotty accuracy seen in the top panels of Figures 3, 5, and 7. Intuitively this makes sense, as significant overlap between the distributions becomes a source of considerable noise that the algorithm has to filter through in order differentiate between points that may have come from either one of the two component normals.

Conversely, the algorithm is quite adept at dealing with mixtures where the component distributions are quite far apart from one another. We see this borne out in the tables above, and in the bottom panels of figure 3, 5, and 7. In these instances, the EM algorithm can take as few as two iterations to reach an estimate of the mixture coefficient within five decimal places of accuracy.

Overall this report seems to verify the common advantages and disadvantages of the EM algorithm. Namely, it is a convenient and easily implemented method for handling difficult maximum likelihood estimation problems, and for certain situations performs admirably. However in other situations it can be horrendously slow to converge, and the parameter estimate(s) to which the algorithm does converge may not be ideal, accurate, or realistic. But in the absence of other estimation methods, the EM algorithm is a powerful tool to have in the proverbial Analyst Toolbox, and opens a number of interesting avenues for theory and application. The negative aspects of the algorithm highlighted in this report can easily circumnavigated by following the number one best practice of data analysis: explore the data first before applying any particular approach.

In closing, it is worth observing that because of the fact that the location and scale parameters were treated as known values, the behavior of the EM algorithm here represents a **best case** scenario for estimating the mixture coefficients. In a more general situation, where we apply the maximization step for each parameter, the algorithms output and performance will change drastically and behave in potentially novel ways not seen in this report.

References

[BS13] D.D. Boos and L.A. Stefanski. *Essential Statistical Inference: Theory and Methods*. Springer Texts in Statistics. Springer New York, 2013.

[CB01] George Casella and Roger Berger. *Statistical Inference*. Duxbury Resource Center, June 2001.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[EH16] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Cambridge University Press, USA, 1st edition, 2016.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

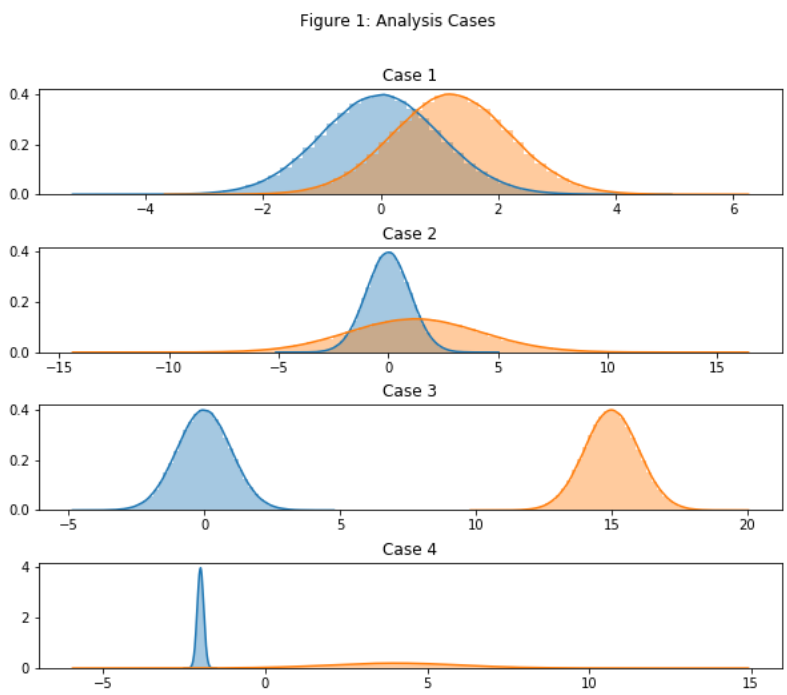


Figure 1: Possible types of two-component normal mixture density behavior one might encounter (not accounting for the actual mixing). Components can be similar shaped and overlap significantly, or possibly differ in location and shape completely.

Figure 2: Mixture coefficient convergence

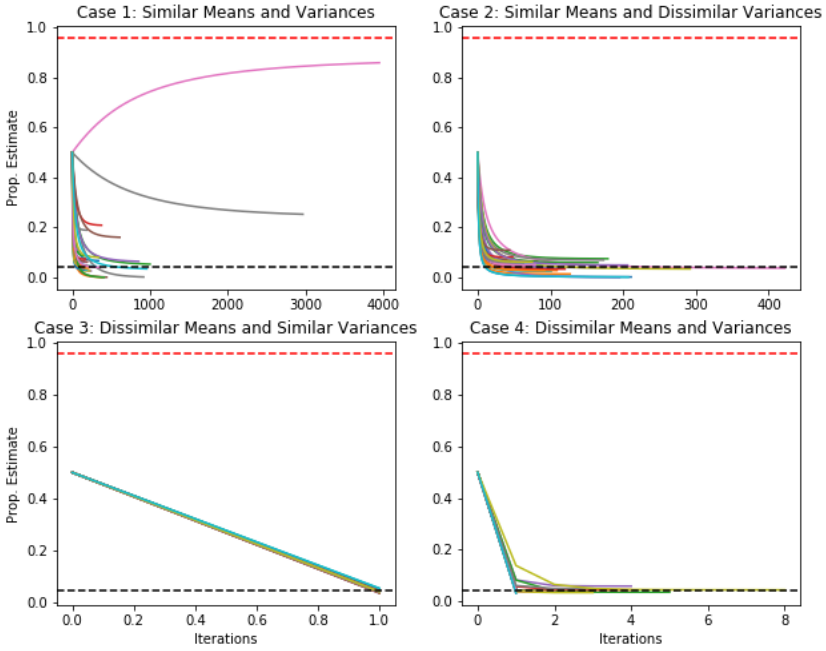


Figure 2: Results for the first run. The black dashed line represents the true mixture proportion $\pi = 0.0421$; the red dashed line $1 - \pi$. Note that case 1 is the most divergent of all cases – taking the longest to converge and having the most erratic estimate behavior.

Figure 3: Deviations from true mixture coefficient

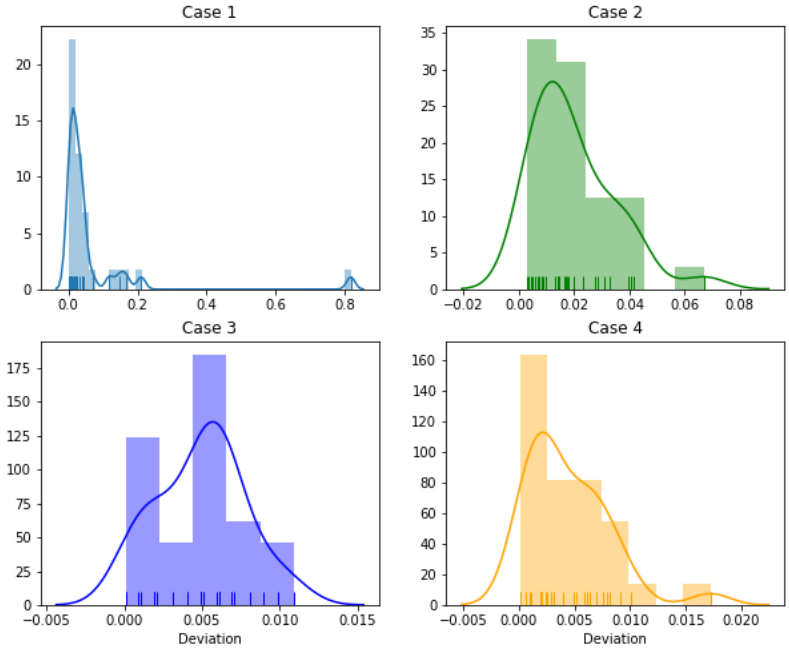


Figure 3: Deviations of cases 1-4 from the true proportion π. Note that estimates are most accurate for cases 3 and 4, where the location parameters of the two components differ significantly from each other. Cases 1 and 2 display fair amount of variability by comparison.

Figure 4: Mixture coefficient convergence (second run)

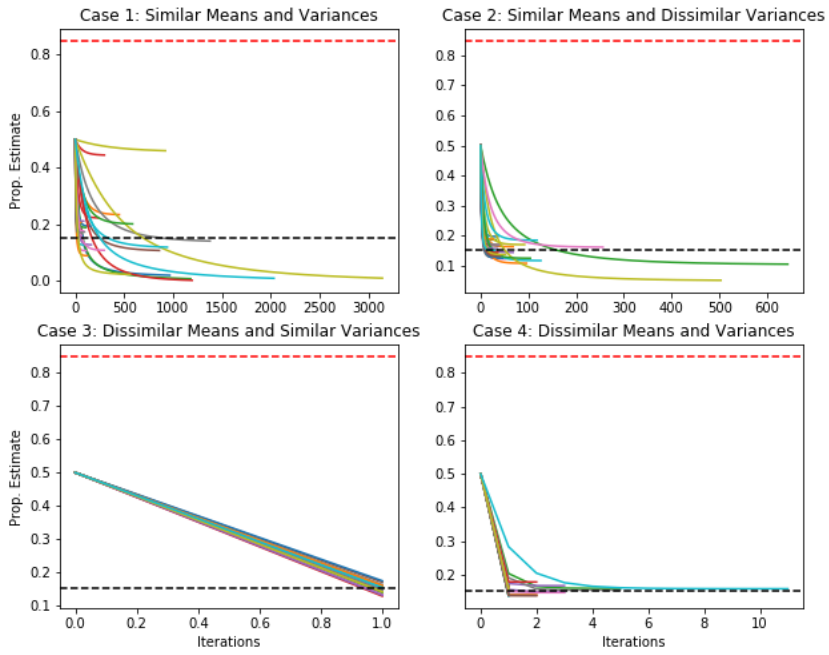


Figure 4: Second run convergence for the four analysis cases. Here $\pi = 0.1527$, and is given by the dashed black line. Again, as in Figure 2, we see that case 1 yields the most divergent results.

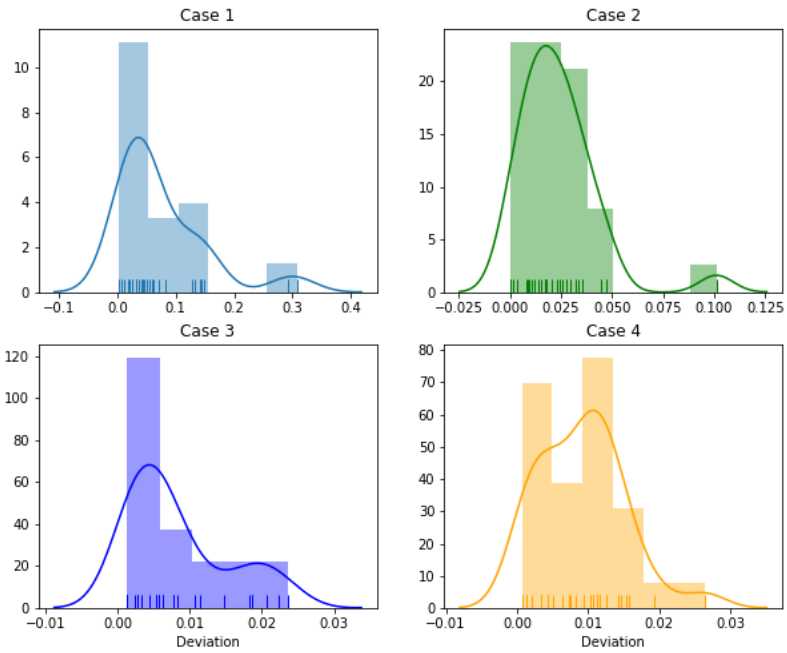


Figure 5: Accuracy of the estimates for run 2. Histograms are of the deviations of the 30 estimates for each case from the true proportion $\pi = 0.1527$

Figure 6: Mixture coefficient convergence (third run)

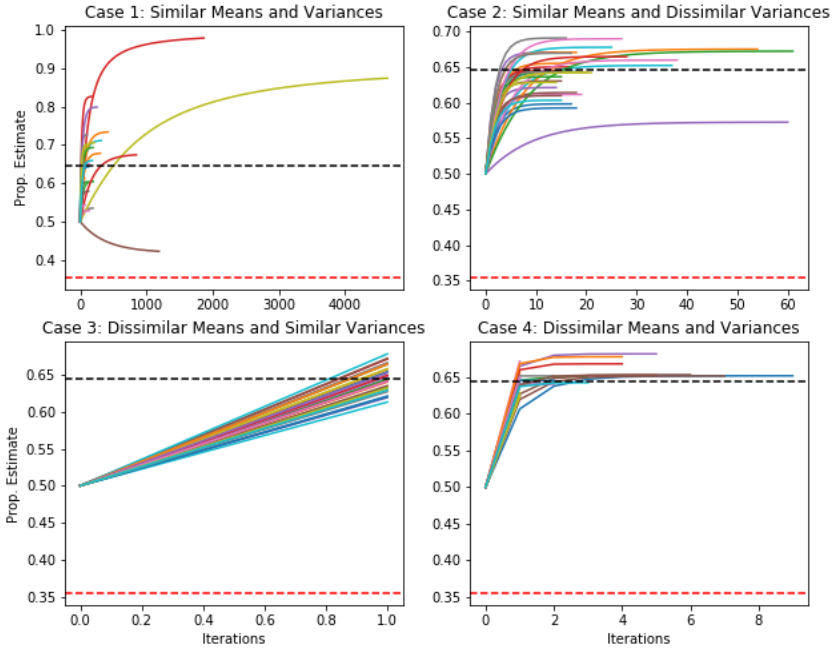


Figure 6: Run 3 convergence results for $\pi = 0.6450$. Here as before the divergent behavior of the estimates for case 1 can be seen. Case 2 here also appears to yield some spurious results. Cases 3 and 4 display well-behaved convergence.

Figure 7: Deviations from true coefficient (third run)

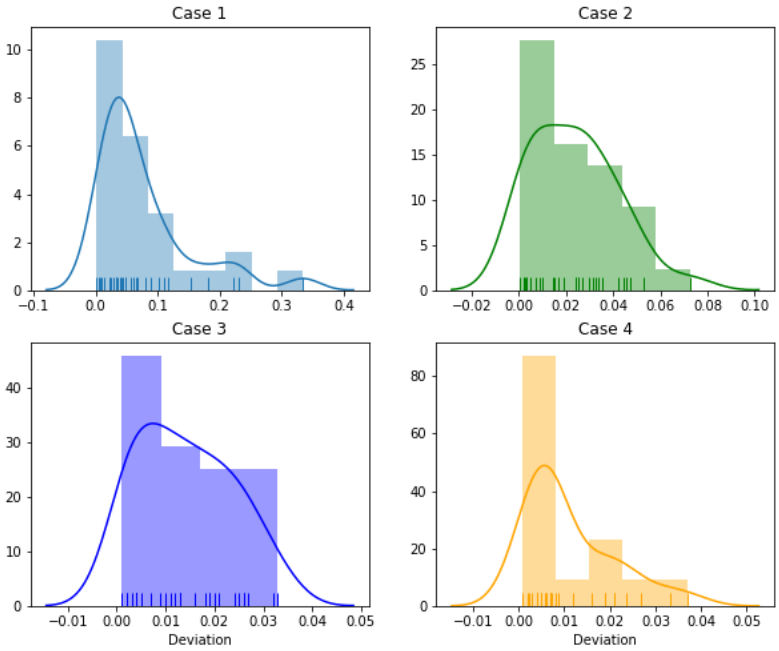


Figure 7: Run 3 accuracy. As seen from previous plots, cases 3 and 4 yield most accurate estimates, while 1 and 2 are most variable.

Appendix

Joseph Free

Project 4

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import norm
```

1. Function Definitions

In [2]:
```
# This function generates variates from a mixture of two normal distributions.
def mixture_dist(mu1, mu2, var1, var2, p1, p2=0, size=1):
    '''
    Generates variates from a mixture of two normal distributions.

    mu1: mean of first distribution.
    mu2: mean of second distribution.
    var1: variance of first distribution
    var2: variance of second distribution
    p1: mixture coefficient for first distribution. This is required.
    p2: mixture coefficient for the second distribution. This is optional. If not specified, this is set to 1-p1.
    size: Optional. Default value is 1. This allows one to generate a single value or a sequence of variates of length _
size_.
    '''
    # Check if weights are legal.
    if (p1>1) or (p2 != 0 and not (p1+p2 == 1)):
        print("Weights do not sum to 1")
        return None
    elif p2 == 0:
        p2 = 1-p1

    # If size is one, generate a single variate.
    if size == 1:
        if np.random.uniform() < p1:
            return np.random.normal(mu1,np.sqrt(var1))
        else:
            return np.random.normal(mu2,np.sqrt(var2))

    # If size is not one, generate
    else:
        sample = np.array([])
        for i in range(0,size):
            if np.random.uniform() < p1:
                sample = np.append(sample, np.random.normal(mu1,np.sqrt(var1)))
            else:
                sample = np.append(sample, np.random.normal(mu2,np.sqrt(var2)))
        return sample
```

In [3]:
```
# This function applies the EM algorithm to a mixture of two normal distributions.
def EM_mix(mu1,mu2, v1,v2, p1, size=1000, tol=0.00001):
    '''
    This function uses the EM algorithm to approximate the mixture coefficients of a mixture of
    two normal distributions.

    Inputs:

    mu1: mean of first distribution.
    mu2: mean of second distribution.
    v1: variance of first distribution
    v2: variance of second distribution
    p1: mixture coefficient for first distribution. This is required.
    size: Optional. Default value is 1. This allows one to generate a single value or a sequence of variates of length _
size_.

    Output:

    The output of this function is a list of the estimated parameters and the estimated values at each iteration of the
algorithm.

    '''

    # Generate sample
    samp = mixture_dist(mu1,mu2,v1,v2, p1, size = size)

    # Initialize guess proportion.
    pi0 = 0.5
    iteration = []
    for i in range(0,10000):
        # Calculate numerator and denominator of iterative procedure.
        numer = pi0*norm.pdf(samp, loc=mu1, scale=np.sqrt(v1))
        denom = numer + (1-pi0)*norm.pdf(samp, loc=mu2, scale=np.sqrt(v2))
        # Calculate ith iterate of latent variable.
        zk = numer/denom

        # Store previous proportion and calculate ith iterate of the
        # mixture coefficient.
        iteration.append(pi0)
        pi0 = sum(zk)/size
        if abs(pi0-iteration[i]) < tol:
            break

    return [(pi0, 1-pi0), iteration, p1 - pi0]
```

2. Testing and Diagnostics

In this section we seek to test the EM algorithms performance at estimating the mixture coefficients. To do so we focus on two main criteria: 1) speed of convergence and 2) accuracy. Furthermore, because the mixture components are univariate normal and on the real line, we additionally consider four possible relationships that exist between the component distributions, which is summarized in the table below:

Location	Scale
Similar	Similar
Similar	Dissimilar
Dissimilar	Similar
Dissimilar	Dissimilar

The crux of the testing approach is to evaluate the EM algorithm's performace using criteria 1) and 2) in each of these four categories. Intuitively we should expect the algorithm to perform better in some categories, and worse in others.

In [4]:
```
def diagnostics(img1_name='Figure 2: Mixture coefficient convergence',
                img2_name='Figure 3: Deviations from true mixture coefficient'):

    # Initialize random coefficient to be used in testing.
    random_proportion = np.random.uniform()
    print("Proportions:")
    print("-"*25)
    print("p1: " + str(random_proportion))
    print("p2: " + str(1-random_proportion))


    fig = plt.figure(figsize=(10,8))
    fig.suptitle(img1_name)

    # Case 1: Mean and variance similar
    ax = fig.add_subplot(2,2,1)
    ax.set_title("Case 1: Similar Means and Variances")

    case1_estimates = []
    case1_iterations = []
    for i in range(0,30):
        out = EM_mix(0, np.random.uniform(0, 1),
                     1, 1 + np.random.uniform(0, .5),
                     random_proportion,
                     size = 1000)
        case1_estimates.append(out[0][0])
        case1_iterations.append(len(out[1]))
        sns.lineplot(range(0,len(out[1])), out[1], ax=ax)
    ax.set_ylabel("Prop. Estimate")
    ax.axhline(y=random_proportion, ls='dashed', color = 'black')
    ax.axhline(y=1-random_proportion, ls='dashed', color = 'red')


    # Case 2: Means similar and variances dissimilar
    ax = fig.add_subplot(2,2,2)
    ax.set_title("Case 2: Similar Means and Dissimilar Variances")

    case2_estimates = []
    case2_iterations = []
    for i in range(0,30):
        out = EM_mix(0, np.random.uniform(0, .5),
                 1, 1 + np.random.uniform(0, 20),
                 random_proportion,
                 size = 1000)
        case2_estimates.append(out[0][0])
        case2_iterations.append(len(out[1]))
        sns.lineplot(range(0,len(out[1])), out[1], ax=ax)
    ax.axhline(y=random_proportion, ls='dashed', color = 'black')
    ax.axhline(y=1-random_proportion, ls='dashed', color = 'red')


    # Case 3: Means dissimilar and variances similar
    ax = fig.add_subplot(2,2,3)
    ax.set_title("Case 3: Dissimilar Means and Similar Variances")

    case3_estimates = []
    case3_iterations = []
    for i in range(0,30):
        out = EM_mix(0, np.random.uniform(10, 100),
                 1, 1 + np.random.uniform(0, .2),
                 random_proportion,
                 size = 1000)
        case3_estimates.append(out[0][0])
        case3_iterations.append(len(out[1]))
        sns.lineplot(range(0,len(out[1])), out[1], ax=ax)
    ax.set_xlabel("Iterations")
    ax.set_ylabel("Prop. Estimate")
    ax.axhline(y=random_proportion, ls='dashed', color = 'black')
    ax.axhline(y=1-random_proportion, ls='dashed', color = 'red')

    # Case 4: Means dissimilar and variances dissimilar
    ax = fig.add_subplot(2,2,4)
    ax.set_title("Case 4: Dissimilar Means and Variances")

    case4_estimates = []
    case4_iterations = []
    for i in range(0,30):
        out = EM_mix(0, np.random.uniform(10, 100),
                 1+np.random.uniform(20,40), 1 + np.random.uniform(0, 20),
                 random_proportion,
                 size = 1000)
        case4_estimates.append(out[0][0])
        case4_iterations.append(len(out[1]))
        sns.lineplot(range(0,len(out[1])), out[1], ax=ax)
    ax.set_xlabel("Iterations")
```

In [5]:
```
np.random.seed(829)
diagnostics()
```

```
Proportions:
-------------------------
p1: 0.04209553382949116
p2: 0.9579044661705088
Case 1 Mean Iteration Length:  563.6666666666666
Case 2 Mean Iteration Length:  129.73333333333332
Case 3 Mean Iteration Length:  2.0
Case 4 Mean Iteration Length:  2.933333333333333
Case 1 Median Iteration Length:  276.0
Case 2 Median Iteration Length:  105.0
Case 3 Median Iteration Length:  2.0
Case 4 Median Iteration Length:  2.0
```

Figure 2: Mixture coefficient convergence

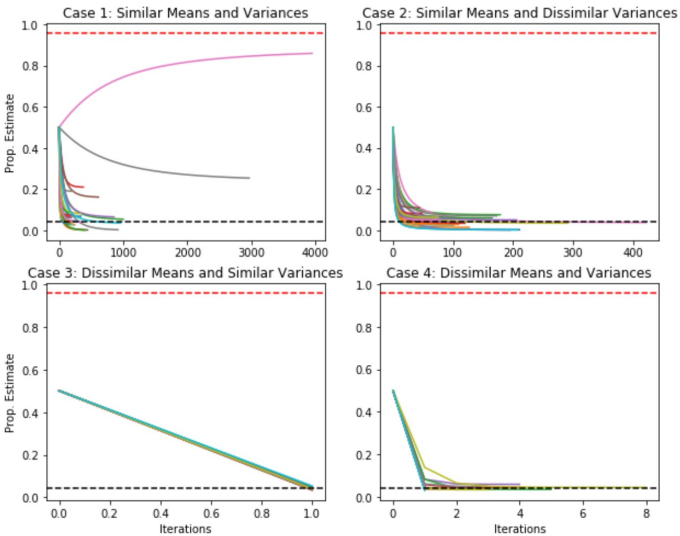
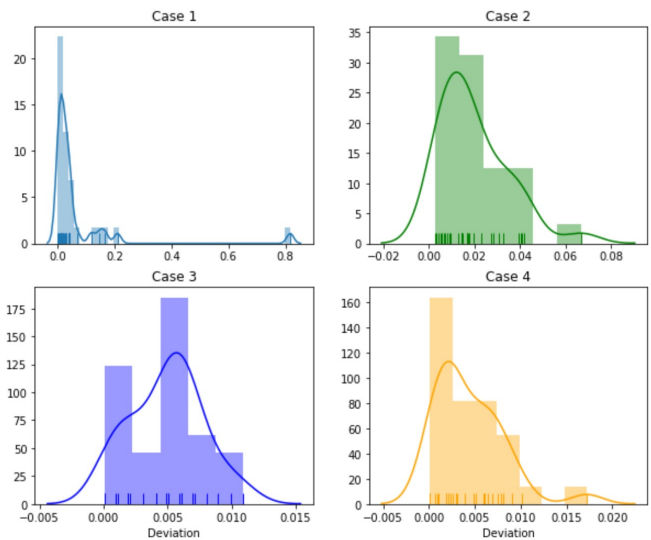


Figure 3: Deviations from true mixture coefficient



In [6]:
```
np.random.seed(777)
diagnostics("Figure 4: Mixture coefficient convergence (second run)",
            "Figure 5: Deviations from true coefficient (second run)")
```

```
Proportions:
-------------------------
p1: 0.152663734901322
p2: 0.847336265098678
Case 1 Mean Iteration Length:  550.3333333333334
Case 2 Mean Iteration Length:  95.56666666666666
Case 3 Mean Iteration Length:  2.0
Case 4 Mean Iteration Length:  2.933333333333333
Case 1 Median Iteration Length:  200.0
Case 2 Median Iteration Length:  42.0
Case 3 Median Iteration Length:  2.0
Case 4 Median Iteration Length:  2.0
```

Figure 4: Mixture coefficient convergence (second run)

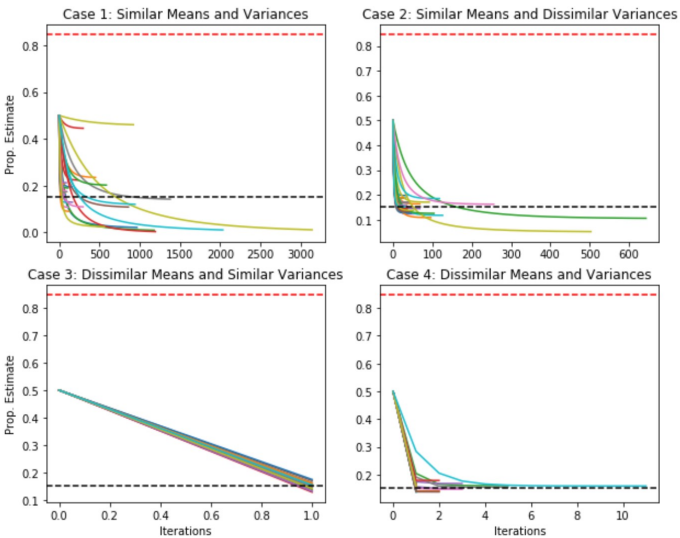
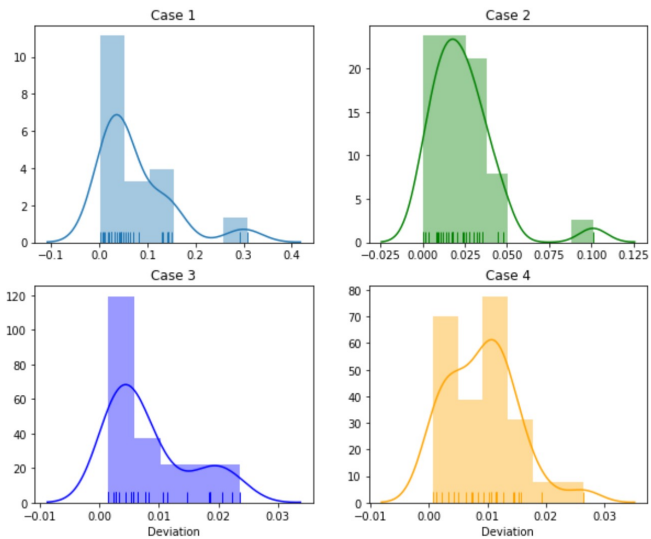


Figure 5: Deviations from true coefficient (second run)



In [7]:
```
np.random.seed(1204)
diagnostics("Figure 6: Mixture coefficient convergence (third run)",
            "Figure 7: Deviations from true coefficient (third run)")
```

```
Proportions:
-------------------------
p1: 0.6450392905877504
p2: 0.35496070941224955
Case 1 Mean Iteration Length:  413.9
Case 2 Mean Iteration Length:  24.133333333333333
Case 3 Mean Iteration Length:  2.0
Case 4 Mean Iteration Length:  3.3333333333333335
Case 1 Median Iteration Length:  134.5
Case 2 Median Iteration Length:  19.0
Case 3 Median Iteration Length:  2.0
Case 4 Median Iteration Length:  2.0
```

Figure 6: Mixture coefficient convergence (third run)

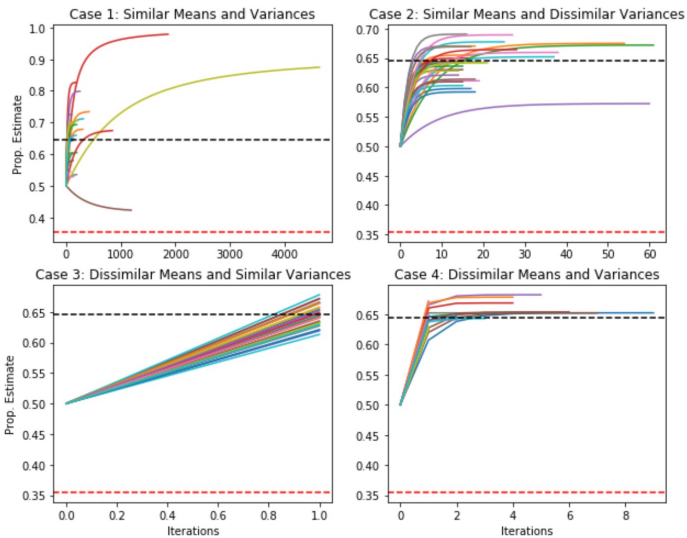
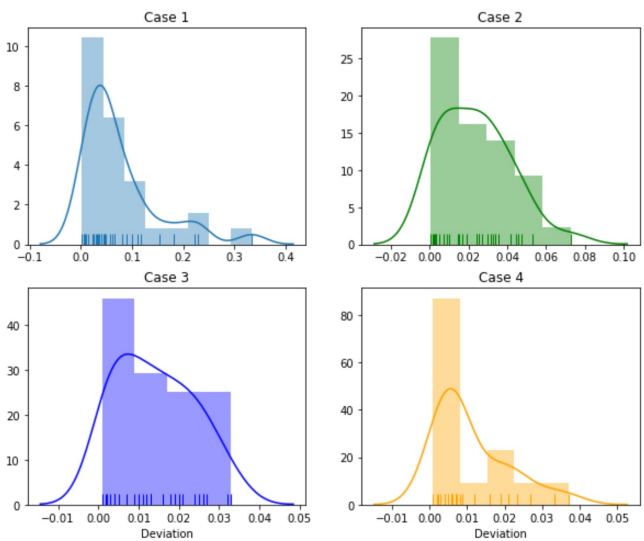


Figure 7: Deviations from true coefficient (third run)



In [ ]: