The EM Algorithm in Multivariate Gaussian Mixture Models using Anderson Acceleration

by
Joshua H. Plasse

A Project Report

Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Master of Science
in
Applied Mathematics

by

May 2013

APPROVED:

Dr. Homer F. Walker, Thesis Advisor

Dr. Bogdan Vernescu, Department Head

Abstract

Over the years analysts have used the EM algorithm to obtain maximum likelihood estimates from incomplete data for various models. The general algorithm admits several appealing properties such as strong global convergence; however, the rate of convergence is linear which in some cases may be unacceptably slow. This work is primarily concerned with applying Anderson acceleration to the EM algorithm for Gaussian mixture models (GMM) in hopes of alleviating slow convergence.

As preamble we provide a review of maximum likelihood estimation and derive the EM algorithm in detail. The iterates that correspond to the GMM are then formulated and examples are provided. These examples show how faster convergence is experienced when the data are well separated, whereas much slower convergence is seen whenever the sample is poorly separated. The Anderson acceleration method is then presented, and its connection to the EM algorithm is discussed. The work is then concluded by applying Anderson acceleration to the EM algorithm which results in reducing the number of iterations required to obtain convergence.

Acknowledgments

I would first like to thank my advisor and project mentor Dr. Homer Walker. His guidance throughout the duration of this exposition proved invaluable. I also would like to express my gratitude to Professor Mayer Humi and Professor Marcus Sarkis for instilling a strong background in various areas of numerical analysis. I would also like to thank Dr. Joseph Fehribach for taking his time to conduct an independent study.

Secondly, I would like to express my thanks to Kyle Dunn, Marisa Zemsky, William Sanguinet and Dan Brady, who helped vectorize MATLAB code, produce aesthetically appealing graphics and help alleviate problems in LaTeX. I would also like to acknowledge Mike Malone and Sia Najafi who provided me with a powerful computer that was used extensively in the computation of large scale data sets.

Lastly, I would like to thank my mother, father, Briannah and Alannah for all the love and support that they have given more over the span of the last twenty four years. To Caitlin, I express my utmost gratitude for the patience and encouragement you have shown me which has helped me get where I am today. It is to them I dedicate this project.

This work was supported in part through U.S. Department of Energy Office of Science Grant DE-SC0004880 to Worcester Polytechnic Institute.

Contents

List of Figures

List of Tables

Introduction

Of interest is the estimation of parameters in a mixture model where all underlying components are multivariate Gaussian distributions of dimension at least two. To be precise, throughout this exposition the model used will be a Gaussian mixture model (GMM) that represents a population composed of $m \in \mathbb{Z}^+$ subpopulations. To accompany our model we also assume that we have an unlabeled random sample $\mathcal{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$ which was obtained in an independent and identically distributed (iid) fashion. An unlabeled sample in this sense means that for any $\mathbf{x}_k \in \mathcal{X}, \quad k = 1, ..., N$, the true subpopulation to which $\mathbf{x}_k$ belongs is not known. The goal is to make accurate statistical inferences on properties of the subpopulations using only the unlabeled sample $\mathcal{X}$. As will be seen in the sequel, maximum likelihood estimation via the EM algorithm is a powerful tool in obtaining accurate parameter estimates for the GMM as well as various other models.

Mathematically we define the GMM as

$$p(\mathbf{x}|\Phi) = \sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}|\phi_i), \tag{1}$$

where $\mathbf{x} = (x_1, ..., x_d)^T \in \mathbb{R}^d$, $\phi_i = (\mu_i, \mathbf{\Sigma}_i)$, $\Phi = (\alpha_1, ..., \alpha_m, \phi_1, ..., \phi_m) \in \Omega$ and each p_i is a d-dimensional multivariate Gaussian distribution given by,

$$p_i(\mathbf{x}|\phi_i) = \frac{1}{(2\pi)^{d/2}(\det \mathbf{\Sigma}_i)^{1/2}} e^{-1/2(\mathbf{x}-\mu_i)^T \mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mu_i)}. \tag{2}$$

Similar to the univariate Gaussian distribution, $\mu_i \in \mathbb{R}^d$ represents the mean vector for the i^{th} subpopulation; whereas, $\mathbf{\Sigma}_i$ is the $d \times d$ symmetric positive definite covariance matrix that corresponds to the i^{th} subpopulation. The collection of α_i's are known as the model's *mixture proportions*, i.e. each α_i represents the probability that a randomly selected $\mathbf{x}_k \in \mathcal{X}$ was generated from the i^{th} subpopulation. Since each α_i is a probability, it follows that $\alpha_i \in [0, 1]$ for all i and the α_i's are constrained to sum to one. Due to this constraint, it follows that $p(\mathbf{x}|\Phi)$ is a probability density function (pdf) since

$$\begin{aligned} 1 &= \sum_{i=1}^{m} \alpha_i \\ &= \sum_{i=1}^{m} \alpha_i \int_{\mathbb{R}^d} p_i(\mathbf{x}|\phi_i) d\mathbf{x} \\ &= \int_{\mathbb{R}^d} \sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}|\phi_i) d\mathbf{x} \\ &= \int_{\mathbb{R}^d} p(\mathbf{x}|\Phi) d\mathbf{x}. \end{aligned}$$

where pdf stands for probability density function. Now that we have defined the model we seek numerical methods that will allow one to compute accurate estimates for Φ. Since ultimately our goal is to introduce, derive and accelerate the EM algorithm for (1), which is used for maximum likelihood estimation, we provide the reader with a review of the method.

Maximum Likelihood Estimation

The notations that will be developed reflects those that are used in Casella and Berger [2]. First we suppose that we have an n-tuple of random vectors $\mathbf{X} = (X_1, X_2, ..., X_n)$ that was generated in an iid fashion. We also assume that the distribution of $\mathbf{X}$ depends on a *fixed* unknown parameter $\theta = (\theta_1, ..., \theta_k)$ that takes its value in the parameter space Θ. Thus for any $X_i \in \mathbf{X}$ the individual probability density (mass) function (pdf (pmf)) for X_i will be denoted by $X_i \overset{iid}{\sim} f(x_i|\theta)$ for $i = 1, ..., n$. We may now take advantage of our sample being iid by defining our *joint* pdf (pmf) for $\mathbf{X}$ as

$$f(\mathbf{x}|\theta) = \prod_{i=1}^{n} f(x_i|\theta_1, ..., \theta_k). \tag{3}$$

Here $\mathbf{x} = (x_1, ..., x_n)$ denotes the observed values of $X_1.,,,.X_n$. To understand how to compute a maximum likelihood estimate (MLE) for a given distribution, one must be familiar with the concept of *likelihood functions*, which we define below.

Definition 1. Let $f(\mathbf{x}|\theta)$ denote the joint pdf (pmf) of our random sample $\mathbf{X} = (X_1, ..., X_n)$ and be defined as in (3). Then given the observed values $\mathbf{X} = \mathbf{x}$, the *likelihood function* is defined as

$$\mathcal{L}(\theta|\mathbf{x}) = \mathcal{L}(\theta_1, ...\theta_k|x_1, ..., x_n) = f(\mathbf{x}|\theta) = \prod_{i=1}^{n} f(x_i|\theta_1, ..., \theta_k). \tag{4}$$

The reader should be aware of the subtle distinction between $\mathcal{L}(\theta|\mathbf{x})$ and $f(\mathbf{x}|\theta)$. When one talks about $f(\mathbf{x}|\theta)$, it is assumed that θ is some unknown fixed quantity, whereas $\mathbf{x}$ is allowed to vary over all possible values in our sample space. When the likelihood function $\mathcal{L}(\theta|\mathbf{x})$ is in consideration we treat $\mathbf{x}$ as known, whereas now θ is allowed to vary over the parameter space Θ. The only distinction between the two functions is which variable is considered fixed and which is allowed to vary. The idea of the likelihood function is that given $\theta', \theta'' \in \Theta$ we may use the likelihood function to determine which parameter values are more likely for the observed sample $\mathbf{x}$. Without loss of generality we shall assume that $\mathcal{L}(\theta'|\mathbf{x}) > \mathcal{L}(\theta''|\mathbf{x})$. It then follows that for the observed sample $\mathbf{x}$, θ' is a more likely parameter value for θ than θ'' since it resulted in a larger value of the likelihood function. Therefore, we should restrict our attention to the parameter values that yield the largest possible values of our likelihood function. Using these ideas we can now give a rigorous definition for an MLE.

Definition 2. For an observed sample $\mathbf{x}$ let

$$\hat{\theta}(\mathbf{x}) \in \underset{\theta \in \Theta}{\arg\max}\ \mathcal{L}(\theta|\mathbf{x}).$$

Then $\hat{\theta}(\mathbf{x})$ is a maximum likelihood estimate for θ based on $\mathbf{x}$.

Here $\arg\max\ \mathcal{L}(\theta|\mathbf{x})$ denotes the set of all values $\theta \in \Theta$ which maximize $\mathcal{L}(\theta|\mathbf{x})$ over our parameter space Θ. In some cases we may appeal to differential calculus to obtain the MLE's for distributions; however, in general we cannot maximize $\mathcal{L}(\theta|\mathbf{x})$ analytically and must rely on numerical techniques to obtain reasonable approximations for $\hat{\theta}(\mathbf{x})$. We conclude the introduction to MLE's by providing a simple example that allows one to obtain $\hat{\theta}$ analytically.

Calculating an MLE Analytically

Let $X_1, ..., X_n$ be a random sample with $X_i \overset{iid}{\sim} \text{gamma}(\alpha, \beta)\ \ \forall\, i = 1, ..., n$. We are interested in calculating the MLE for β under the assumption that $\alpha > 0$ is known.

Solution: Each X_i has the following pdf

$$f(x_i|\beta) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x_i^{\alpha-1} e^{-x_i/\beta}.$$

Since we have iid random variables, the joint pdf is

$$\begin{aligned} f(\mathbf{x}|\beta) &= \prod_{i=1}^n \frac{1}{\Gamma(\alpha)\beta^\alpha} x_i^{\alpha-1} e^{-x_i/\beta} \\ &= \frac{1}{(\Gamma(\alpha)\beta^\alpha)^n} \prod_{i=1}^n x_i^{\alpha-1} e^{-x_i/\beta} \\ &= \mathcal{L}(\beta|\mathbf{x}). \end{aligned}$$

We now implement a frequently used technique. We shall consider $\log(\mathcal{L}(\beta|\mathbf{x}))$ and note that because the natural logarithm is a monotonic strictly increasing function, the maximization of $\log(\mathcal{L}(\beta|\mathbf{x}))$ is equivalent to the maximization of $\mathcal{L}(\beta|\mathbf{x})$. We see that

$$\begin{aligned} \log(\mathcal{L}(\beta|\mathbf{x})) &= \log\left[\frac{1}{(\Gamma(\alpha)\beta^\alpha)^n} \prod_{i=1}^n x_i^{\alpha-1} e^{-x_i/\beta}\right] \\ &= -n\log(\Gamma(\alpha)) - n\log(\beta^\alpha) + \sum_{i=1}^n \log(x_i^{\alpha-1} e^{-x_i/\beta}) \\ &= -n\log(\Gamma(\alpha)) - n\alpha\log(\beta) + \sum_{i=1}^n \log(x_i^{\alpha-1}) - \sum_{i=1}^n x_i/\beta. \end{aligned}$$

To maximize $\log(\mathcal{L}(\beta|\mathbf{x}))$ we use methods from differential calculus to obtain,

$$\frac{\partial(\log(\mathcal{L}(\beta|\mathbf{x})))}{\partial\beta} = -\frac{n\alpha}{\beta} + \frac{\sum x_i}{\beta^2} = 0.$$

Thus the only possible candidate for the MLE is $\hat{\beta} = \bar{x}/\alpha$ where $\bar{x} = \sum x_i/n$. We verify that $\hat{\beta}$ is a maximizer by showing the second partial derivative evaluated at $\hat{\beta}$ is indeed always negative:

$$\begin{aligned} \frac{\partial^2(\log(\mathcal{L}(\beta|\mathbf{x})))}{\partial\beta^2}\Big|_{\beta=\hat{\beta}} &= \left(\frac{n\alpha}{\beta^2} - \frac{2\sum x_i}{\beta^3}\right)\Big|_{\beta=\hat{\beta}} \\ &= \frac{(n\alpha)^3}{(\sum x_i)^2} - \frac{2(n\alpha)^3}{(\sum x_i)^2} \\ &= -\frac{(n\alpha)^3}{(\sum x_i)^2} < 0. \end{aligned}$$

Thus we have shown that $\hat{\beta}$ is a local maximum of $\log(\mathcal{L}(\beta|\mathbf{x}))$ and since this is the *only* value where the derivative is equal to zero it follows that $\hat{\beta} = \bar{x}/\alpha$ is the global maximizer and the required MLE.

As seen in the previous example, the calculation of MLE's can be rather involved even in simple cases. In general, obtaining MLE's analytically is either impossible or impractical due to exhaustive algebra or the presence of highly non-linear systems. Therefore, there is a great need for a numerical technique to provide one with accurate approximations for maximum likelihood estimates.

In the next section of the project, we introduce the EM algorithm as a technique to approximate MLE's from incomplete data and relate the algorithm to general mixture models. We then turn our attention to the normal mixture model defined by (1) and (2) and derive the algorithm. We then discuss convergence properties and how convergence depends on the separation of the generated data, and we provide several examples.

The EM Algorithm

The **E**xpectation **M**aximization algorithm, or EM algorithm for short, was given its name in a 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin [3]. The algorithm is an iterative method for finding MLE's of a statistical model where there is a dependency upon unobserved latent variables. Throughout the remainder of this section we seek to apply the EM algorithm to obtain accurate approximations to the MLE's of a particular model which has *incomplete* data associated with it. Here the data are considered incomplete in the sense that the observations do not contain complete information.

The general form of the algorithm is twofold. The first step is an **E**-Step, which creates a function Q that represents the expectation of the log-likelihood and evaluates Q at the current parameter estimates. The second step is the **M**-Step, the purpose of which is to maximize the expectation previously found in the **E**-Step. In the next section we present the algorithm in its most general form.

General EM

We now formulate the EM algorithm in general for an incomplete data problem. The notations to be developed are a mixture of those presented in [1] and [4]. We assume that we have a random incomplete data set $\mathcal{X}$ that follows some probability distribution and that there exist unobserved observations $\mathcal{Y}$ such that the data set defined by $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$ may be deemed complete. What it means to have a complete data set depends on the context of the problem. For example, one could have missing data due to errors in data collection or due to some time constraint.

To establish some notation, we let $f(\mathbf{z}|\Phi) = f(\mathbf{x}, \mathbf{y}|\Phi)$ represent the joint pdf of our random variables $\mathcal{X}$ and $\mathcal{Y}$ and denote the marginal pdf of $\mathcal{X}$ by $g(\mathbf{x}|\Phi)$. We also let $k(\mathbf{y}|\mathbf{x}, \Phi)$ denote the conditional probability distribution of $\mathcal{Y}$ given $\mathcal{X} = \mathbf{x}$. The goal of the EM algorithm is to maximize the incomplete data log-likelihood, defined as $\log[\mathcal{L}(\Phi|\mathcal{X})] = \log[g(\mathbf{x}|\Phi)]$ over $\Phi \in \Omega$, by using the relationship between $f(\mathbf{x}, \mathbf{y}|\Phi)$ and $g(\mathbf{x}|\Phi)$. From an introductory course on probability, we know that $f(\mathbf{z}|\Phi)$ may be represented as

$$f(\mathbf{z}|\Phi) = f(\mathbf{x}, \mathbf{y}|\Phi) = k(\mathbf{y}|\mathbf{x}, \Phi)g(\mathbf{x}|\Phi) \qquad \mathbf{x} \in \mathcal{X},\ \mathbf{y} \in \mathcal{Y}, \tag{5}$$

where (5) will be used to show certain useful properties of the EM iterates. In the case of missing data, the **E**-Step refers to finding the expected value of the complete data log-likelihood, defined as $\log[\mathcal{L}(\Phi|\mathcal{X}, \mathcal{Y})] = \log[f(\mathbf{x}, \mathbf{y}|\Phi)]$, where the observed sample $\mathcal{X}$ and some current parameter estimate $\Phi^c \in \Omega$ are given. Thus we define

$$Q(\Phi|\Phi^c) = \mathbb{E}[\log f(\mathbf{x}, \mathbf{y}|\Phi)|\mathbf{x}, \Phi^c] \qquad \mathbf{x} \in \mathcal{X},\ \mathbf{y} \in \mathcal{Y},\ \Phi^c \in \Omega \tag{6}$$

where $\mathbb{E}[\cdot]$ denotes the expectation operator. The **M**-Step then seeks to maximize (6), i.e, we choose a $\Phi^+ \in \Omega$ such that

$$\Phi^+ \in \underset{\Phi \in \Omega}{\arg\max}\ Q(\Phi|\Phi^c). \tag{7}$$

Thus in its most abstract form the EM algorithm given by Dempster, Laird and Rubin in [3] is given by the following algorithm.

General EM Iteration

(1) **E**-Step: Calculate $Q(\Phi|\Phi^c)$.
(2) **M**-Step: Choose $\Phi^+ \in \arg\max_{\Phi\in\Omega} \; Q(\Phi|\Phi^c)$.
(3) Let $\Phi^c = \Phi^+$.
(4) Repeat (1)-(3) as necessary.

From this general description, the reader may question the usefulness of the algorithm since the topic of convergence has not yet been addressed. Therefore, we provide some theory on convergence.

Convergence Properties

As stated previously, the purpose of the EM algorithm is to maximize the incomplete data log-likelihood which we now denote by $L(\Phi) = \log[g(\mathbf{x}|\Phi)]$[1]. Using (5) we see that

$$\begin{aligned} f(\mathbf{x},\mathbf{y}|\Phi) &= k(\mathbf{y}|\mathbf{x},\Phi)g(\mathbf{x}|\Phi) \\ \implies \log[f(\mathbf{x},\mathbf{y}|\Phi)] &= \log[k(\mathbf{y}|\mathbf{x},\Phi)] + \log[g(\mathbf{x}|\Phi)] \\ \implies \log[g(\mathbf{x}|\Phi)] &= \log[f(\mathbf{x},\mathbf{y}|\Phi)] - \log[k(\mathbf{y}|\mathbf{x},\Phi)], \end{aligned}$$

where by the linearity of the expectation operator we arrive at

$$\mathbb{E}[\log(g(\mathbf{x}|\Phi))] = \mathbb{E}[\log(f(\mathbf{x},\mathbf{y}|\Phi))] - \mathbb{E}[\log(k(\mathbf{y}|\mathbf{x},\Phi))]. \tag{8}$$

We seek to express (8) in terms of $L(\Phi)$ and the function $Q(\Phi|\Phi')$ for some arbitrary parameter $\Phi' \in \Omega$. Therefore, in (8) we shall treat $\mathbf{x}$ and Φ' as given, which leads to

$$\mathbb{E}[\log(g(\mathbf{x}|\Phi))|\mathbf{x},\Phi'] = \mathbb{E}[\log(f(\mathbf{x},\mathbf{y}|\Phi))|\mathbf{x},\Phi'] - \mathbb{E}[\log(k(\mathbf{y}|\mathbf{x},\Phi))|\mathbf{x},\Phi']. \tag{9}$$

Since we are treating $\mathbf{x}$ and $\Phi = \Phi'$ as known, the left hand side of (9) is just some constant. We also know that $\mathbb{E}[c] = c$ for any constant $c \in \mathbb{R}$. Therefore (9) reduces to

$$\begin{aligned} \log(g(\mathbf{x}|\Phi)) &= \mathbb{E}[\log(f(\mathbf{x},\mathbf{y}|\Phi))|\mathbf{x},\Phi'] - \mathbb{E}[\log(k(\mathbf{y}|\mathbf{x},\Phi))|\mathbf{x},\Phi'] \\ \implies L(\Phi) &= Q(\Phi|\Phi') - H(\Phi|\Phi'), \end{aligned}$$

where $H(\Phi|\Phi') = \mathbb{E}[\log(k(\mathbf{y}|\mathbf{x}))|\mathbf{x},\Phi']$ and $Q(\Phi|\Phi')$ is defined as in (6). Hence, we have now successfully expressed $L(\Phi)$ (which we ultimately want to maximize) in terms of $Q(\Phi|\Phi')$, which appears in our EM iterates.

[1]The change in notation is so that we may be consistent with notations in [1]

Upon inspection of the above, we see that if the following conditions hold then the EM algorithm would possess convergence properties:

$$\begin{aligned} &\text{(i)} \quad L(\Phi^+) \geq L(\Phi^c) \\ &\text{(ii)} \quad Q(\Phi^+|\Phi^c) \geq Q(\Phi^c|\Phi^c) \\ &\text{(iii)} \quad H(\Phi^+|\Phi^c) \leq H(\Phi^c|\Phi^c) \end{aligned}$$

where $\Phi^c \in \Omega$ is the current parameter estimate for our MLE and Φ^+ is defined in (7). Simply by the definition of Φ^+, we see that (ii) must always be satisfied. To verify (iii), we paraphrase a lemma which is given in [3, pp. 6].

Lemma 1. For any pair $(\Phi^+, \Phi^c) \in \Omega \times \Omega$, we have that

$$H(\Phi^+|\Phi^c) \leq H(\Phi^c|\Phi^c). \tag{10}$$

The proof of Lemma 1 uses the concavity of the natural logarithm to appeal to Jensen's Inequality, from which (10) follows. Part (i) follows trivially from (ii) and (iii). Thus (i)-(iii) all hold, and (i) implies that L is monotone increasing on any iteration sequence produced by the EM algorithm. Therefore each iteration of the algorithm will increase the log-likelihood and it is likely that the iterates will converge to some local maximum of the likelihood function. It is well known that the rate at which the EM algorithm converges is linear. For one such proof we refer the reader to [1, Thm 5.2]. As will be seen in the sequel, for us the rate at which the algorithm converges will depend on how well our data are "separated". For more properties on convergence of the EM algorithm we refer the reader to [1],[3] and [5].

Thus we have thoroughly explained the EM algorithm in its most general form. In the next section we shall discuss how to approximate MLE's for mixture models as well as derive the necessary EM iterates for the d-dimensional multivariate Gaussian distribution.

EM Algorithm for Mixture Models

To begin discussing the EM algorithm for mixture models, we shall assume as in the introduction that we have a parametric family of mixture densities of the form (1) with the understanding that at this moment each p_i need not be Gaussian. The model in consideration is of the form

$$p(\mathbf{x}|\Phi) = \sum_{i=1}^{m} \alpha_i p_i(\mathbf{x}|\phi_i) \qquad i = 1, ..., m, \;\; \mathbf{x} \in \mathcal{X}.$$

As before, $\mathcal{X} = (\mathbf{x}_1, ..., \mathbf{x}_N)$ represents an unlabeled random sample, $\Phi = (\alpha_1, ..., \alpha_m, \phi_1, ..., \phi_m)$, our mixing proportions α_i are constrained to be non-negative and sum to one, and each p_i represents some density function parameterized by ϕ_i. Hence, our mixture model is composed of m component densities mixed together with our mixture proportions α_i, where $m \in \mathbb{Z}^+$ denotes the number of subpopulations present.

We now treat the unlabeled sample $\mathcal{X}$ as an incomplete data set by regarding each unlabeled observation $\mathbf{x}_k \in \mathcal{X}$ as "missing" a label which lets us know which subpopulation $\mathbf{x}_k$ originated in. Thus we let $\mathcal{Y} = (y_1, ..., y_N)$ where $y_k \in \{1, ..., m\}$ denote the unobserved data which indicates what component density generated $\mathbf{x}_k$. Thus $\mathcal{Z} = (\mathcal{X}, \mathcal{Y})$ constitutes our complete data set. A valid question the reader should be asking at this time is whether or not this reformulation of our problem is worth it. The answer in short is yes because treating $\mathcal{X}$ as incomplete and introducing the existence of $\mathcal{Y}$ simplifies the maximization process. For example, if $\mathcal{Y}$ is known it is shown in [4] that the log-likelihood for the complete data is given as

$$\log(\mathcal{L}(\Phi|\mathcal{X}, \mathcal{Y})) = \sum_{k=1}^{N} \log(\alpha_{y_k} p_{y_k}(\mathbf{x}_k|\phi_{y_k}))$$

which is a much easier expression to maximize than the unlabeled data log-likelihood. However, it is wishful thinking to think that $\mathcal{Y}$ is known and we must proceed by treating $\mathcal{Y}$ as a random vector.

Our goal now is to derive expressions for the distribution of the unobserved data and the function $Q(\Phi|\Phi')$. As preamble we let $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_N)$ and $\mathbf{y} = (y_1, ..., y_N)$ be the sample variables whose associated probability density functions are given as follows:

$$g(\mathbf{x}|\Phi) = \prod_{k=1}^{N} p(\mathbf{x}_k|\Phi)$$

$$f(\mathbf{x}, \mathbf{y}|\Phi) = \prod_{k=1}^{N} \alpha_{y_k} p_{y_k}(\mathbf{x}_k|\phi_{y_k}).$$

Then for any $\Phi' = (\alpha'_1, ..., \alpha'_m, \phi'_1, ..., \phi'_m) \in \Omega$ we may appeal to the identity given by (5) to obtain a closed expression for the conditional density $k(\mathbf{y}|\mathbf{x}, \Phi')$. This density may be expressed as

$$k(\mathbf{y}|\mathbf{x}, \Phi') = \prod_{k=1}^{N} \frac{\alpha'_{y_k} p_{y_k}(\mathbf{x}_k|\phi'_{y_k})}{p(\mathbf{x}_k|\Phi')}.$$

We now will derive the relevant expression for $Q(\Phi|\Phi')$ for our mixture model, which is rather involved. Throughout the derivation, we shall appeal to a few clever tricks that were presented by the author of [4]. First, we shall compute the expectation that appears in (6). To do so, the first thing to realize is that $\mathcal{Y}$ can only take on the *discrete* values $\{1, ..., m\}$. Therefore, it follows that $\mathcal{Y}$ is a discrete random variable, and the definition of conditional expectation tells us that

$$\mathbb{E}[h(\mathcal{Y})|\mathbf{x}] = \sum_{y \in \mathsf{Y}} h(y) l(y|\mathbf{x})$$

where $h(\mathcal{Y})$ is any function of $\mathcal{Y}$, $l(y|\mathbf{x})$ is the joint pdf and Y is the set of all allowable y values. Since we are given $\mathcal{X} = \mathbf{x}$ in the definition (6), we know that $\log(f(\mathbf{x}, y|\Phi))$ is purely a function of y, and by the definition previously stated we see that

$$\begin{aligned}
Q(\Phi|\Phi') &= \mathbb{E}[\log f(\mathbf{x}, y|\Phi)|\mathbf{x}, \Phi'] \\
&= \sum_{y \in \mathsf{Y}} \log(f(\mathbf{x}, y|\Phi)) k(y|\mathbf{x}, \Phi') \\
&= \sum_{y \in \mathsf{Y}} \log \left[\prod_{k=1}^{N} \alpha_{y_k} p_{y_k}(\mathbf{x}_k|\phi_{y_k}) \right] k(y|\mathbf{x}, \Phi') \\
&= \sum_{y \in \mathsf{Y}} \sum_{k=1}^{N} \log(\alpha_{y_k} p_{y_k}(\mathbf{x}_k|\phi_{y_k})) \left[\prod_{j=1}^{N} \frac{\alpha'_{y_j} p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')} \right] \\
&= \sum_{y_1=1}^{m} \sum_{y_2=1}^{m} \cdots \sum_{y_N=1}^{m} \sum_{k=1}^{N} \log(\alpha_{y_k} p_{y_k}(\mathbf{x}_k|\phi_{y_k})) \left[\prod_{j=1}^{N} \frac{\alpha'_{y_j} p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')} \right] \qquad (11) \\
&= \sum_{y_1=1}^{m} \sum_{y_2=1}^{m} \cdots \sum_{y_N=1}^{m} \sum_{k=1}^{N} \sum_{i=1}^{m} \delta_{i,y_k} \log(\alpha_i p_i(\mathbf{x}_k|\phi_i)) \left[\prod_{j=1}^{N} \frac{\alpha'_{y_j} p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')} \right] \qquad (12) \\
&= \sum_{i=1}^{m} \sum_{k=1}^{N} \log(\alpha_i p_i(\mathbf{x}_k|\phi_i)) \sum_{y_1=1}^{m} \sum_{y_2=1}^{m} \cdots \sum_{y_N=1}^{m} \delta_{i,y_k} \left[\prod_{j=1}^{N} \frac{\alpha'_{y_j} p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')} \right]. \qquad (13)
\end{aligned}$$

Before we continue we believe the steps labeled by (11) and (12) warrant some explanation. The equality labeled by (11) simply uses the fact that $\mathcal{Y} = (y_1, ..., y_N)$ and each $y_k \in \{1, ..., m\}$ for all k; therefore, the summation over $y \in \mathsf{Y}$ is identically equal to the N summations which appear on the RHS of the equality since these summations represent all possible y values. In (12) we introduce δ_{i,y_k} which represents the Kronecker delta. Equality holds on this line simply because the only value of the summation indexed by i that is non-zero is when $i = y_k$, which preserves the equality. At this juncture one may think that we are making our already convoluted expression worse; however, as will be seen below this trick greatly simplifies the expression labeled by (13). Focusing on the second half of (13) we see that

$$\begin{aligned}
\sum_{y_1=1}^{m}\sum_{y_2=1}^{m}\cdots\sum_{y_N=1}^{m}\delta_{i,y_k}\left[\prod_{j=1}^{N}\frac{\alpha'_{y_j}p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')}\right] &= \left[\sum_{y_1=1}^{m}\cdots\sum_{y_{k-1}=1}^{m}\sum_{y_{k+1}=1}^{m}\cdots\sum_{y_N=1}^{m}\prod_{\substack{j=1\\ j\neq k}}\left(\frac{\alpha'_{y_j}p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')}\right)\right]\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')} \\
&= \left[\prod_{\substack{j=1\\ j\neq k}}\left(\sum_{y_j=1}^{m}\frac{\alpha'_{y_j}p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')}\right)\right]\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')} \\
&= \frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')}.
\end{aligned}$$

The last equality was obtained by realizing that

$$\begin{aligned}
\sum_{y_j=1}^{m}\frac{\alpha'_{y_j}p_{y_j}(\mathbf{x}_j|\phi'_{y_j})}{p(\mathbf{x}_j|\Phi')} &= \frac{\alpha'_1 p_1(\mathbf{x}_j|\phi'_1) + ...\alpha'_m p_m(\mathbf{x}_j|\phi'_m)}{p(\mathbf{x}_j|\Phi')} \\
&= \frac{p(\mathbf{x}_j|\Phi')}{p(\mathbf{x}_j|\Phi')} \\
&= 1.
\end{aligned}$$

Therefore substituting this equality back into (13) allows us to obtain a closed expression for Q. Upon making this substitution we see that

$$\begin{aligned}
Q(\Phi|\Phi') &= \sum_{i=1}^{m}\sum_{k=1}^{N}\log(\alpha_i p_i(\mathbf{x}_k|\phi_i))\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')} \\
&= \sum_{i=1}^{m}\sum_{k=1}^{N}\left(\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')}\log(\alpha_i) + \frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')}\log(p_i(\mathbf{x}_k|\phi_i))\right) \\
&= \sum_{i=1}^{m}\sum_{k=1}^{N}\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')}\log(\alpha_i) + \sum_{i=1}^{m}\sum_{k=1}^{N}\frac{\alpha'_i p_i(\mathbf{x}_k|\phi'_i)}{p(\mathbf{x}_k|\Phi')}\log(p_i(\mathbf{x}_k|\phi_i)).
\end{aligned}$$

This is the closed form expression for Q that appears in the **E**-Step of the EM algorithm and may be maximized if given a particular probability distribution p_i. In the next section we shall consider the case where each p_i is a d-dimensional multivariate Gaussian distribution and present the necessary updated parameter estimates for α_i and ϕ_i.

EM on Normal Mixtures

We now turn our attention to the case when each p_i may be represented as,

$$p_i(\mathbf{x}|\phi_i) = \frac{1}{(2\pi)^{d/2}(\det \mathbf{\Sigma}_i)^{1/2}} e^{-1/2(\mathbf{x}-\mu_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mu_i)} \qquad \mathbf{x} \in \mathcal{X},\ \phi_i = (\mu_i, \mathbf{\Sigma}_i),$$

where as stated in the introduction $\mathbf{x} \in \mathbb{R}^d$, $\mu_i \in \mathbb{R}^d$ and $\mathbf{\Sigma}_i$ is a $d \times d$ symmetric positive definite matrix. Throughout this section we assume that we have a current approximate maximizer of our function Q which we denote by $\Phi^c = (\alpha_1^c ..., \alpha_m^c, \phi_i^c, ..., \phi_m^c)$. Our goal is to now implement the maximization that occurs during the **M**-Step of the EM algorithm to obtain updated maximizers denoted by $\Phi^+ = (\alpha_1^+, ..., \alpha_m^+, \phi_1^+, ..., \phi_m^+)$.

The updated maximizers for our mixture proportions are derived first. Since these proportions are constrained to sum to one we must maximize Q with respect to α_i by introducing the Lagrange multiplier λ and solving the following system of equations:

$$\begin{aligned} \frac{\partial Q}{\alpha_i} &= \lambda \frac{\partial}{\alpha_i}\left(\sum_{i=1}^m \alpha_i\right) \\ \sum_{i=1}^m \alpha_i &= 1. \end{aligned}$$

Or equivalently we may maximize the Lagrangian given by

$$\Lambda(\alpha_i, \lambda) = Q + \lambda\left(\sum_{i=1}^m \alpha_i - 1\right). \tag{14}$$

For simplicity, when we perform the maximization we let $j \in \{1, ..., m\}$ be a dummy index and $\beta_{ik} = \alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)/p(\mathbf{x}_k|\Phi^c)$. Upon maximization we see that

$$\begin{aligned} \frac{\partial \Lambda}{\partial \alpha_j} &= \frac{\partial}{\partial \alpha_j}\left(\sum_{k=1}^N\sum_{i=1}^m \beta_{ik}\log(\alpha_i) + \sum_{k=1}^N\sum_{i=1}^m \beta_{ik}\log(p_i(\mathbf{x}_k|\phi_i))\right) + \lambda\frac{\partial}{\partial \alpha_j}\left(\sum_{i=1}^m \alpha_i - 1\right) \\ &= \frac{\partial}{\partial \alpha_j}\left(\sum_{k=1}^N \beta_{1k}\log(\alpha_1) + \cdots \sum_{k=1}^N \beta_{jk}\log(\alpha_j) + \cdots \sum_{k=1}^N \beta_{mk}\log(\alpha_m) + \sum_{k=1}^N\sum_{i=1}^m \beta_{ik}\log(p_i(\mathbf{x}_k|\phi_i))\right) + \lambda\frac{\partial}{\partial \alpha_j}\left(\sum_{i=1}^m \alpha_i - 1\right) \\ &= \sum_{k=1}^N \frac{1}{\alpha_j}\beta_{jk} + \lambda\frac{\partial}{\partial \alpha_j}(\alpha_1 + \cdots \alpha_j + \cdots \alpha_m - 1) \\ &= \sum_{k=1}^N \frac{1}{\alpha_j}\beta_{jk} + \lambda \\ &= 0. \end{aligned}$$

Replacing the dummy index j with i and solving for λ yields

$$\sum_{k=1}^N \frac{1}{\alpha_i}\beta_{ik} + \lambda = 0 \implies -\lambda\alpha_i = \sum_{k=1}^N \beta_{ik} \implies -\lambda\sum_{i=1}^m \alpha_i = \sum_{k=1}^N\sum_{i=1}^m \frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)} \implies \lambda = -N,$$

from which it follows that

$$\alpha_i^+ = \frac{1}{N}\sum_{k=1}^N \frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}.$$

The derivations of μ_i^+ and $\mathbf{\Sigma}_i^+$ are similar; however, they require techniques which are beyond the scope of this project. Hence, we omit their derivations and refer the reader to [4] for a detailed derivation. The updated parameter estimates for the multivariate Gaussian distribution are then given by the following.

Updated Parameter Estimates

$$\alpha_i^+ = \frac{1}{N}\sum_{k=1}^{N}\frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}$$

$$\mu_i^+ = \frac{\displaystyle\sum_{k=1}^{N}\mathbf{x}_k\frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}}{\displaystyle\sum_{k=1}^{N}\frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}}$$

$$\mathbf{\Sigma}_i^+ = \frac{\displaystyle\sum_{k=1}^{N}(\mathbf{x}_k-\mu_i^+)(\mathbf{x}_k-\mu_i^+)^T\ \frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}}{\displaystyle\sum_{k=1}^{N}\frac{\alpha_i^c p_i(\mathbf{x}_k|\phi_i^c)}{p(\mathbf{x}_k|\Phi^c)}}$$

Examples

We now turn our attention to implementing the EM algorithm for normal mixtures on computer generated data. The programming language that has been used explicitly for all analysis has been MATLAB versions 2011a and 2011b with minor use of its statistical toolbox. All MATLAB scripts used may be found in Appendix A.

It has been stated previously that the convergence of the algorithm will depend heavily on how our data are separated; therefore, to be complete we present two examples which range from having well separated to poorly separated samples.

The following examples all possess the following structure. We assume in each case that we are given a d-dimensional random sample of size $N = 50,000$ and that the number of subpopulations m is known. We then run the EM algorithm twice. The first time we naively make initial guesses for each α_i, μ_i and $\mathbf{\Sigma}_i$ by assigning randomly generated values for each μ_i, letting $\alpha_i = 1/m$ for all i and by assuming that each subpopulation has covariance matrix $\mathbf{\Sigma}_i = \mathbf{I}_{d\times d}$, where $\mathbf{I}_{d\times d}$ represents the $d \times d$ identity matrix. The second time we appeal to the K-means algorithm to cluster our data into m clusters. The centroids of each cluster are then calculated and used as our initial guesses for the means. K-means also allows us to calculate the proportion of points that were assigned to each cluster and the covariance matrix associated with each proposed cluster. These approximations will serve as our initial guesses for α_i and $\mathbf{\Sigma}_i$ respectively.

Well-Separated Case (WSC)

We consider $m = 2$ where the true parameter values to be estimated are given in Table 1.

Sub-Population 1	Sub-Population 2
$\alpha_1 = 0.3$	$\alpha_2 = 0.7$
$\mu_1 = \begin{pmatrix} 1 \\ 2.25 \end{pmatrix}$	$\mu_2 = \begin{pmatrix} 12.9 \\ 10.3 \end{pmatrix}$
$\mathbf{\Sigma}_1 = \begin{pmatrix} 1.75 & 1 \\ 1 & 1.75 \end{pmatrix}$	$\mathbf{\Sigma}_2 = \begin{pmatrix} 3.2 & 1.25 \\ 1.25 & 3.2 \end{pmatrix}$

Table 1: True Parameter Values (WSC)

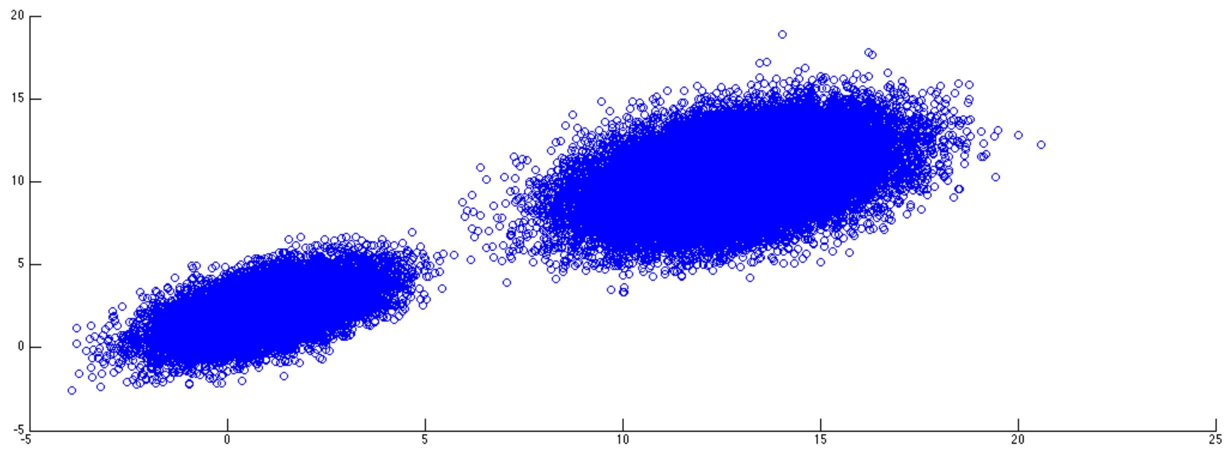

Figure 1: Scatter Plot of our Random Sample (WSC)

As seen in Figure 1 the data suggest the existence of two distinct subpopulations which are very well separated from one another. Hence, one can conjecture that the EM algorithm should converge rather quickly for this case. For the naive choice of initial guesses we obtain the following plots.

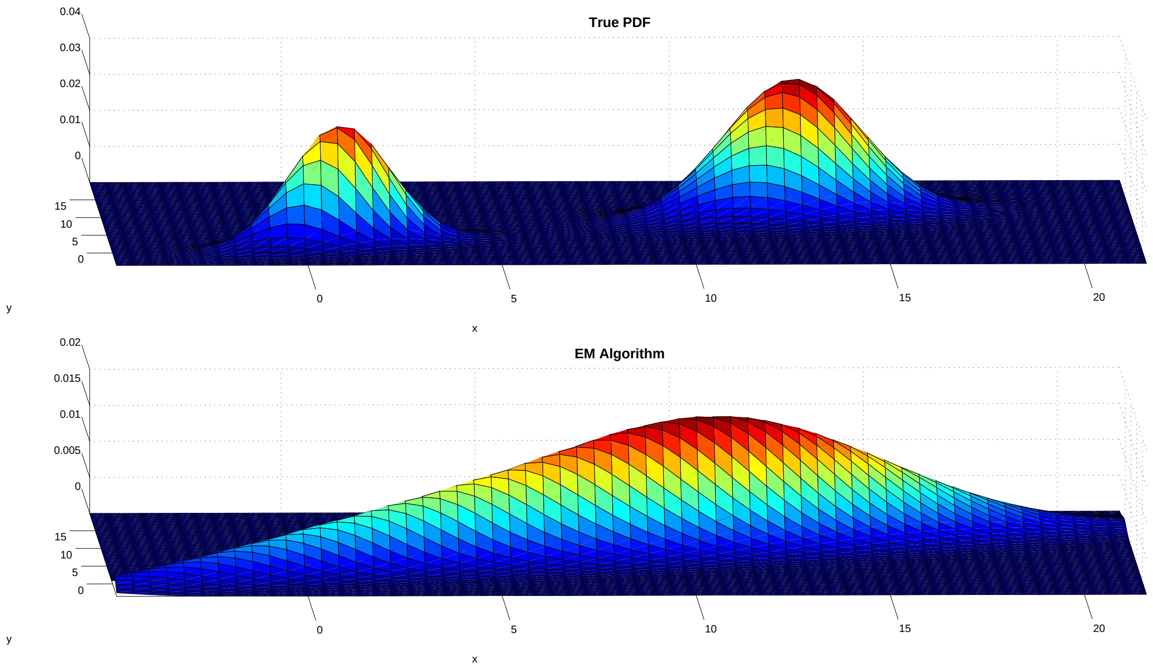

Figure 2: After 3 Iterations (WSC)

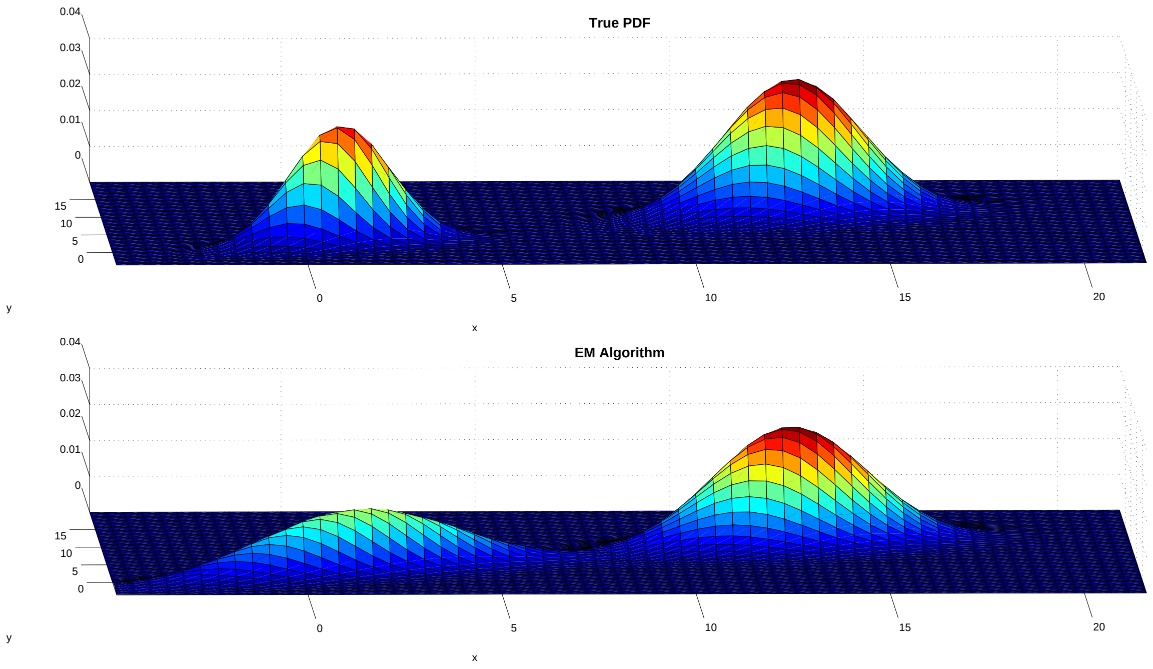

Figure 3: After 5 Iterations (WSC)

Convergence was achieved after the fifteenth iteration, and the error plots are given by Figures 5-7.

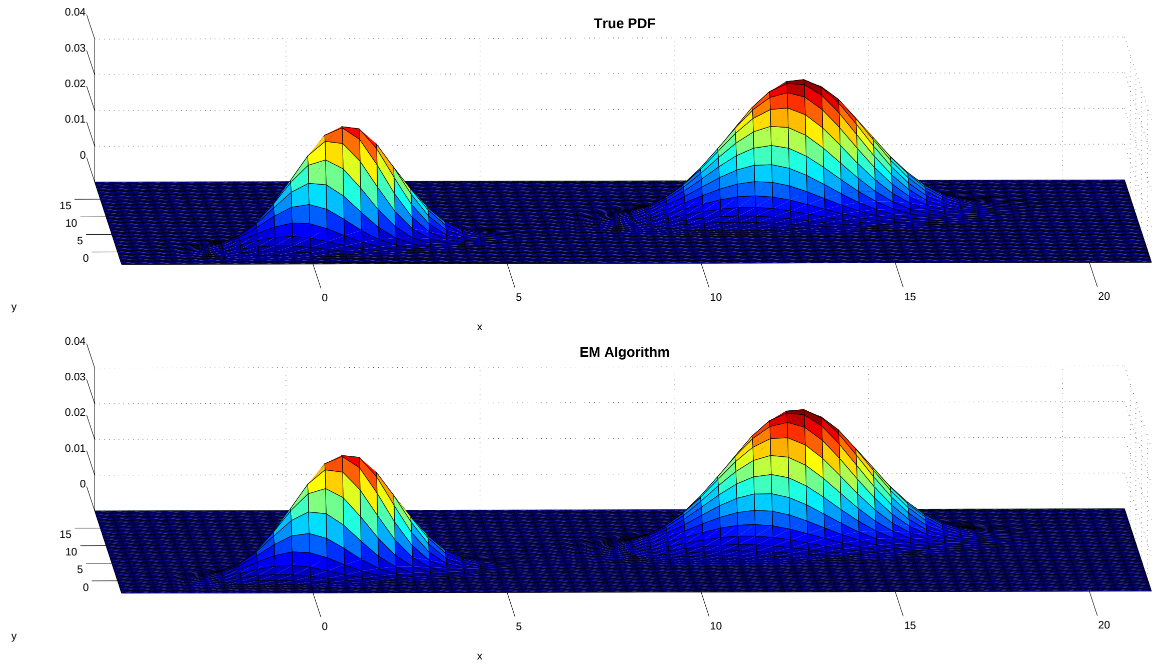

Figure 4: After 15 Iterations (WSC)

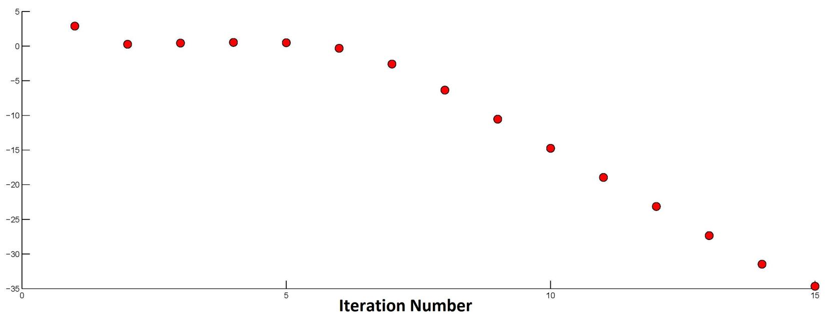

Figure 5: Mean Errors - $\log ||\mu_{i+1} - \mu_i||_\infty$ (WSC)

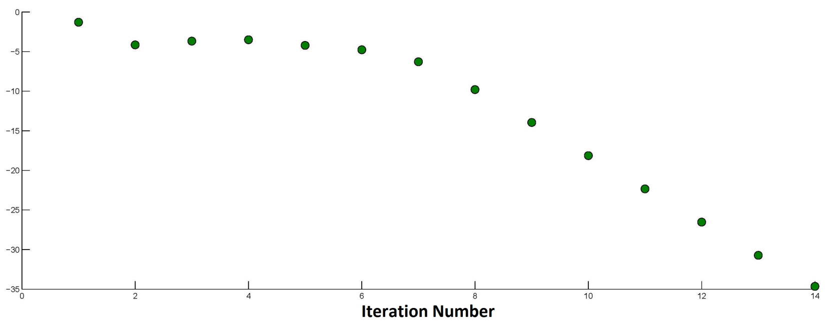

Figure 6: Proportion Errors - $\log ||\alpha_{i+1} - \alpha_i||_\infty$ (WSC)

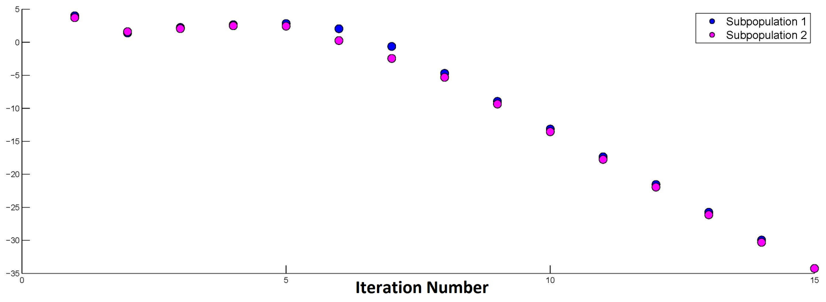

Figure 7: Covariance Errors - $\log ||\mathbf{\Sigma}_{i+1} - \mathbf{\Sigma}_i||_\infty$ (WSC)

Sub-Population 1	Sub-Population 2
$\alpha_1 = 0.3014799$	$\alpha_2 = 0.6985201$
$\mu_1 = \begin{pmatrix} 1.0016354 \\ 2.2455534 \end{pmatrix}$	$\mu_2 = \begin{pmatrix} 12.8742387 \\ 10.3009386 \end{pmatrix}$
$\mathbf{\Sigma}_1 = \begin{pmatrix} 1.7809924 & 1.0112942 \\ 1.0112942 & 1.7388996 \end{pmatrix}$	$\mathbf{\Sigma}_2 = \begin{pmatrix} 3.2138611 & 1.2759833 \\ 1.2759833 & 3.2333514 \end{pmatrix}$

Table 2: Final Approximated Parameter Values (WSC)

As seen in Table 2, in the case of well-separated data the EM algorithm yields reasonable approximations for Φ in a low number of iterations. For a sample size of $N = 50,000$, the values found in Table 2 probably approximate the true parameters about as well as the MLE. We now shall apply the algorithm for the same problem; however, we shall appeal to K-means to obtain a reasonably good initial guess for Φ. Implementing K-means on our sample provides the following clusters and initial parameter estimates.

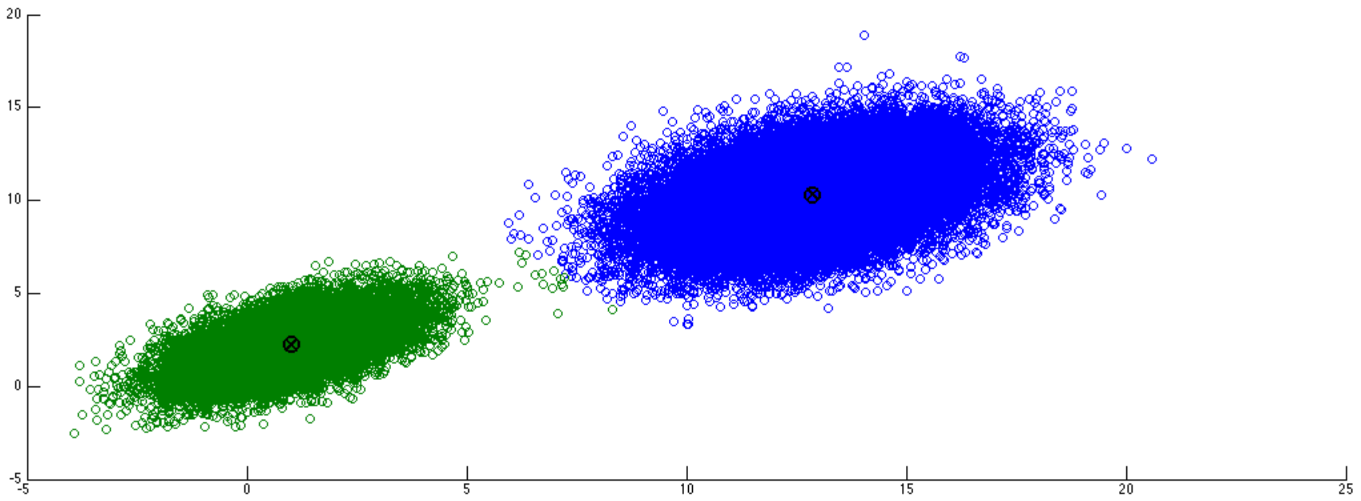

Figure 8: Applying K-Means to our Sample (WSC)

Sub-Population 1	Sub-Population 2
$\alpha_1^0 = 0.30176$	$\alpha_2^0 = 0.69824$
$\mu_1^0 = \begin{pmatrix} 1.0070872 \\ 2.2487461 \end{pmatrix}$	$\mu_2^0 = \begin{pmatrix} 12.8766451 \\ 10.3027901 \end{pmatrix}$
$\mathbf{\Sigma}_1^0 = \begin{pmatrix} 1.8117701 & 1.0288619 \\ 1.0288618 & 1.7490788 \end{pmatrix}$	$\mathbf{\Sigma}_2^0 = \begin{pmatrix} 3.200662 & 1.2655483 \\ 1.2655483 & 3.2258868 \end{pmatrix}$

Table 3: Initial Parameter Guesses Using K-Means (WSC)

Using the initial guesses presented in Table 3, the EM algorithm converges in only nine iterations to the graph presented in Figure 4 and the same values shown in Table 2. In this case the difference in iterations seems insignificant. However, as will be seen in the presence of poorly separated data a good choice for our initial guesses can drastically reduce the number of iterations required to obtain convergence. The newly acquired error plots are presented below followed by a brief comparison of the errors obtained in both cases.

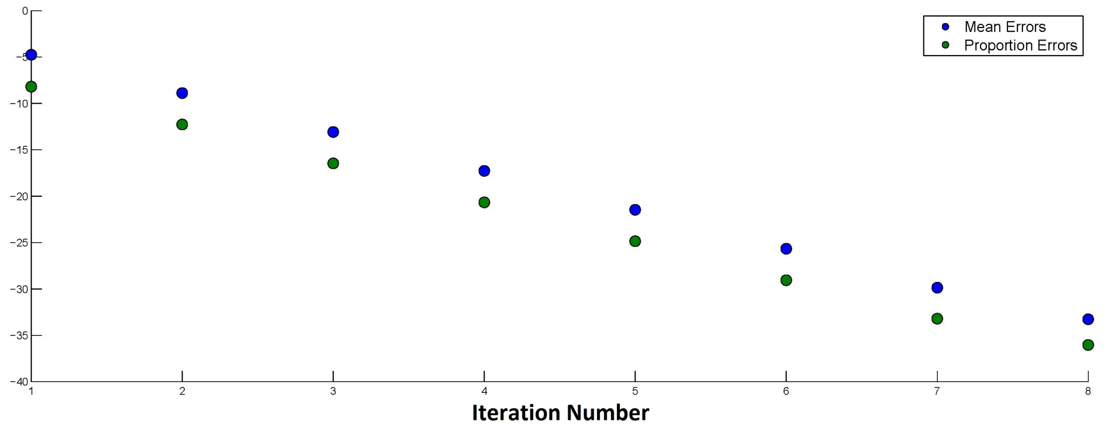

Figure 9: Mean and Proportion Errors using K-Means (WSC)

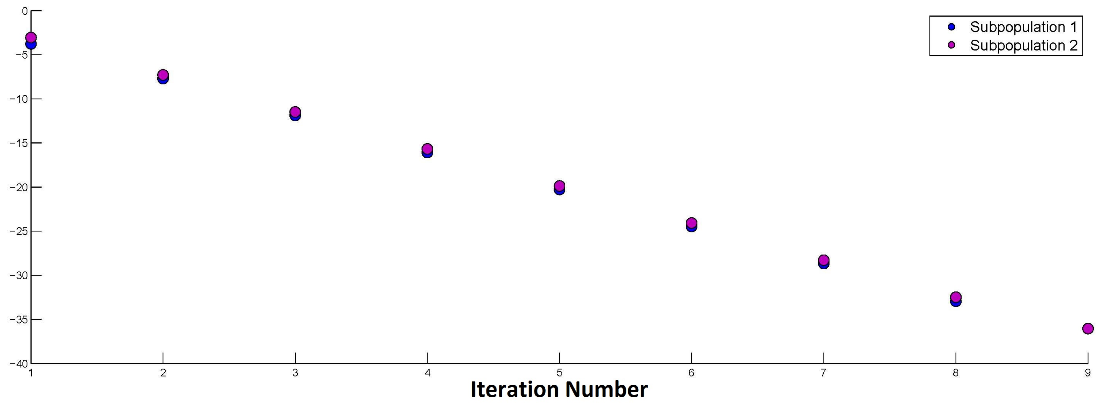

Figure 10: Covariance Errors using K-Means (WSC)

It was stated previously in [3] that the EM algorithm has a linear rate of convergence which is evident in both of our cases. Upon inspection of Figures 5-7 we see slight oscillation in our errors before showing a linear trend, which can be attributed to the naive choices that were made for our initial parameter estimates. Conversely, Figures 9-10 show a strong linear rate of convergence throughout all of the iterations and exhibits no such oscillations. This is to be expected since we began our iteration process with much better initial estimates.

In conclusion, we see that when we are given a well separated sample the EM iterates converges quickly in both cases to accurate parameter approximations. It is also clear that the difference in successive iterates approaches zero linearly which in this case was not problematically slow.

Poorly-Separated Case (PSC)

Throughout this example we shall consider the more difficult task of accurately approximating parameters from a poorly separated random sample. We now consider the existence of three subpopulations for which the true parameters to be estimated are given in Table 4.

Sub-Population 1	Sub-Population 2	Sub-Population 3
$\alpha_1 = 0.3$	$\alpha_2 = 0.5$	$\alpha_3 = 0.2$
$\mu_1 = \begin{pmatrix} 4.5 \\ 6.25 \end{pmatrix}$	$\mu_2 = \begin{pmatrix} 7 \\ 8.95 \end{pmatrix}$	$\mu_3 = \begin{pmatrix} 5.12 \\ 9.5 \end{pmatrix}$
$\mathbf{\Sigma}_1 = \begin{pmatrix} 0.75 & -0.25 \\ -0.25 & 0.75 \end{pmatrix}$	$\mathbf{\Sigma}_2 = \begin{pmatrix} 1.1 & 0.5 \\ 0.5 & 1.1 \end{pmatrix}$	$\mathbf{\Sigma}_3 = \begin{pmatrix} 0.45 & 0.3 \\ 0.3 & 0.45 \end{pmatrix}$

Table 4: True Parameter Values (PSC)

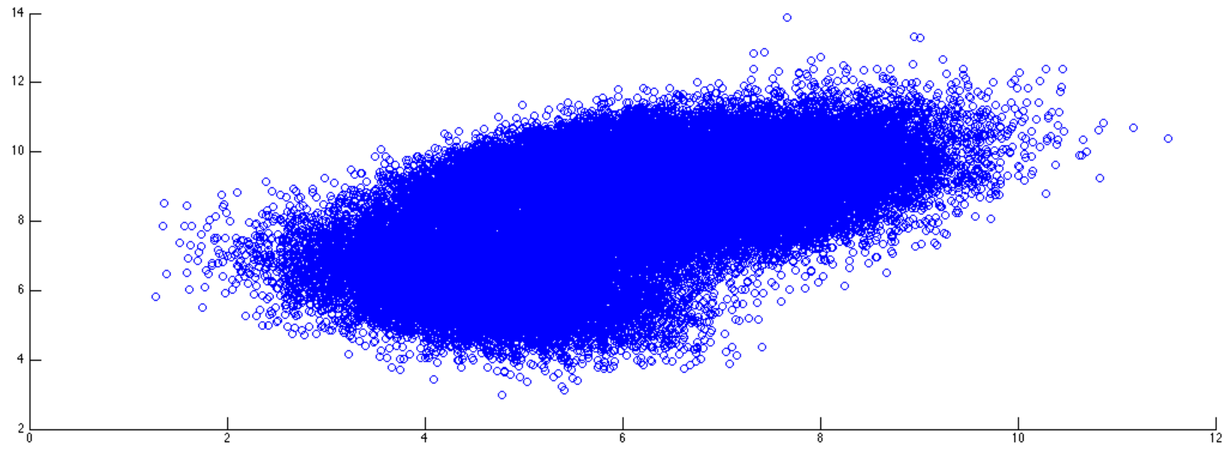

Figure 11: Scatter Plot of our Random Sample (PSC)

Even though we are treating m as known, one surely cannot differentiate between subpopulations by looking at the scatter plot presented in Figure 11. Therefore unlike the well separated case, the EM algorithm will take more iterations to converge since the subpopulations are severely intertwined with one another. Also, in the first example the use of the K-means algorithm to obtain a good initial guess seemed unnecessary; however, when the data are this poorly separated the ability to choose a good initial guess will prove invaluable. As before, we present the case where we use naive initial estimates for our parameters first, followed by analysis.

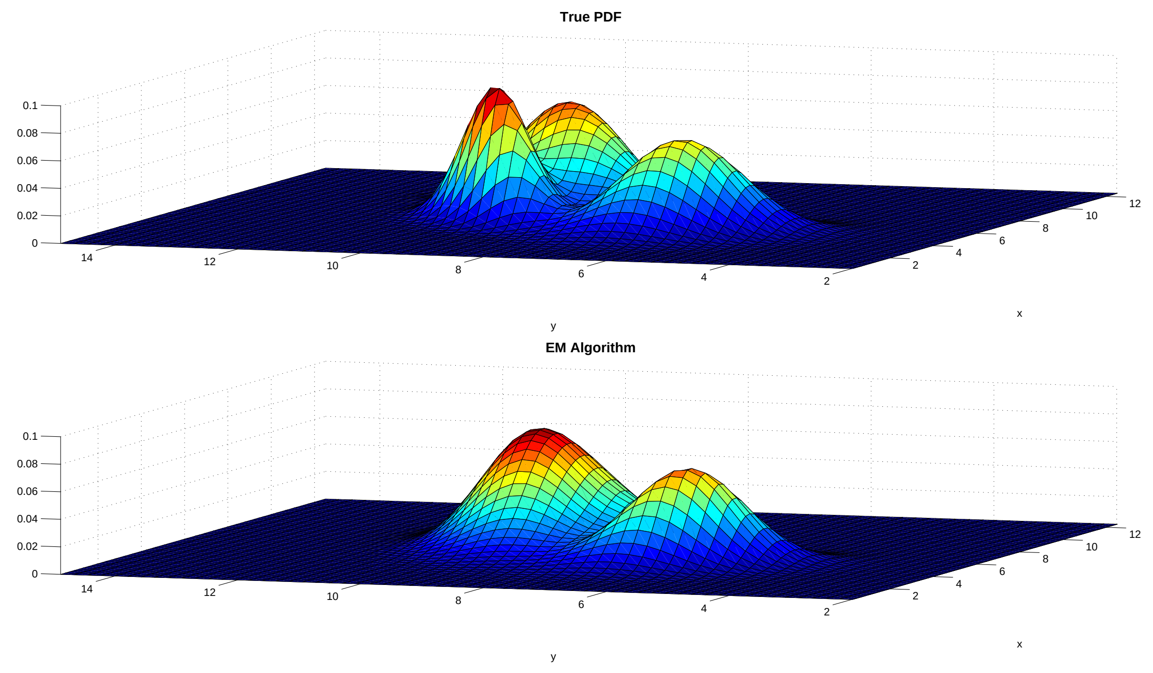

Figure 12: After 50 Iterations (PSC)

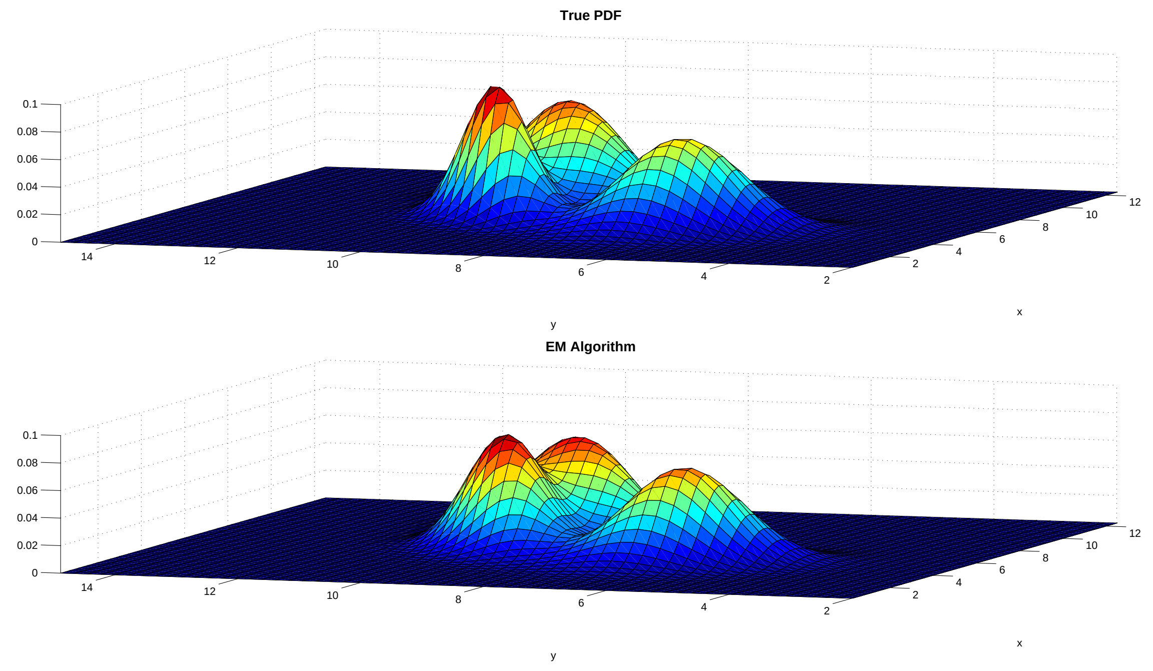

Figure 13: After 100 Iterations (PSC)

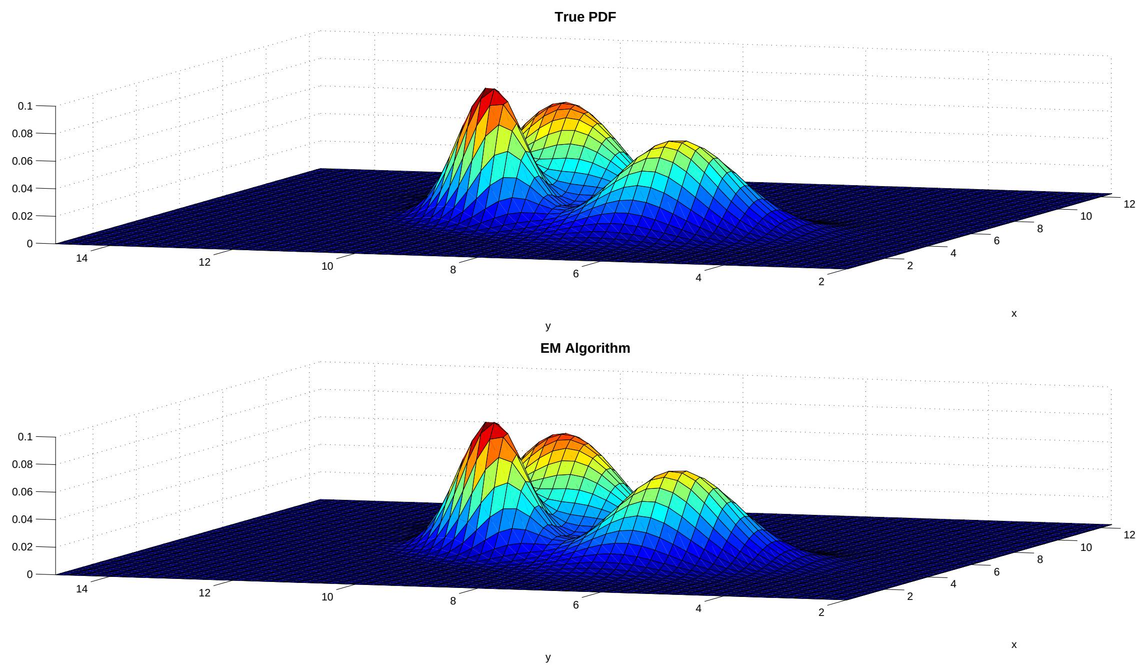

Figure 14: After 250 Iterations (PSC)

Figures 12-14 reinforce our intuition that the EM iterates would take longer to converge. As shown in the figures, it takes roughly 250 iterations for the pdf generated by the algorithm to resemble that of the true pdf. Nonetheless, even in the presence of poorly separated data the EM iterates do converge, and the final estimated values for Φ are presented in Table 5.

Sub-Population 1	Sub-Population 2	Sub-Population 3
$\alpha_1 = 0.3022322$	$\alpha_2 = 0.4991065$	$\alpha_3 = 0.1986613$
$\mu_1 = \begin{pmatrix} 4.5045099 \\ 6.2429225 \end{pmatrix}$	$\mu_2 = \begin{pmatrix} 6.9886764 \\ 8.9585970 \end{pmatrix}$	$\mu_3 = \begin{pmatrix} 5.1154634 \\ 9.4856487 \end{pmatrix}$
$\mathbf{\Sigma}_1 = \begin{pmatrix} 0.7638369 & -0.2518461 \\ -0.2518461 & 0.7360869 \end{pmatrix}$	$\mathbf{\Sigma}_2 = \begin{pmatrix} 1.0851758 & 0.4933538 \\ 0.4933538 & 1.1070892 \end{pmatrix}$	$\mathbf{\Sigma}_3 = \begin{pmatrix} 0.4567573 & 0.3054033 \\ 0.3054033 & 0.4596297 \end{pmatrix}$

Table 5: Approximated Parameter Values (PSC)

Before we run the EM algorithm using K-means, we display the error plots generated on the next page. Note that as in the previous example we see oscillations at the beginning of the iterations for our estimated parameters; however, as the iteration number reaches around 100 a linear trend develops.

The reader should note that in the code that appears in Appendix A, we stop iterating when the error is less than a specified tolerance or when we have reached a maximum allowable iteration number. In this example, we reached our maximum iteration number before our errors were less than our specified tolerance. Therefore, if we chose to allow more than 250 iterations we would obtain slightly more accurate parameter estimates than those presented in Table 5; however, for demonstration purposes we believe 250 iterations is sufficient.

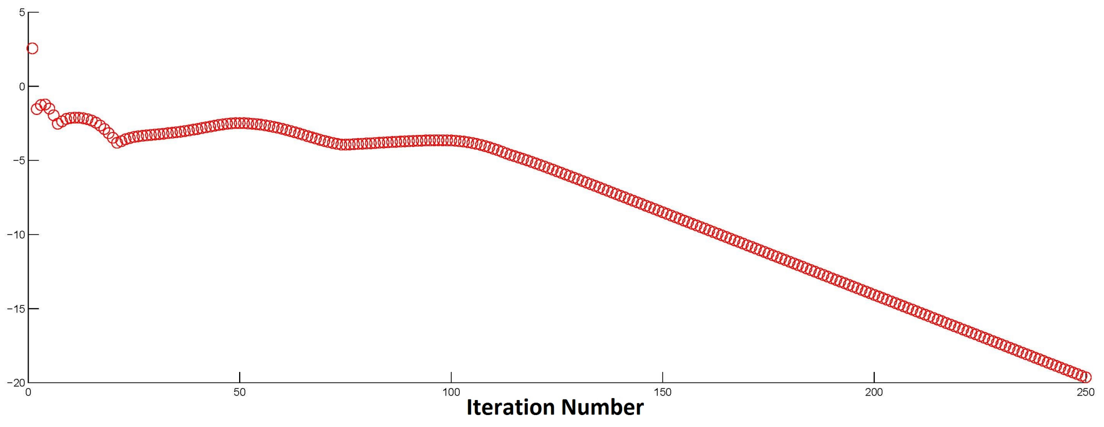

Figure 15: Mean Errors - $\log ||\mu_{i+1} - \mu_i||_\infty$ (PSC)

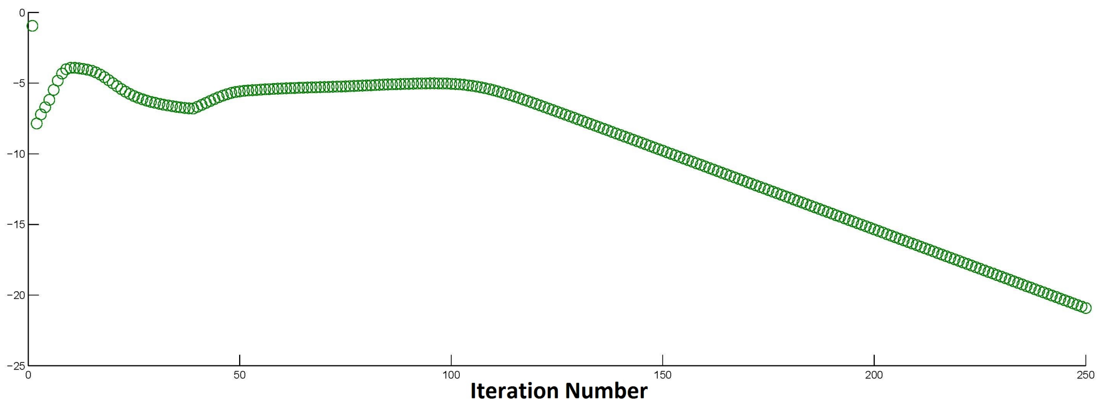

Figure 16: Proportion Errors - $\log ||\alpha_{i+1} - \alpha_i||_\infty$ (PSC)

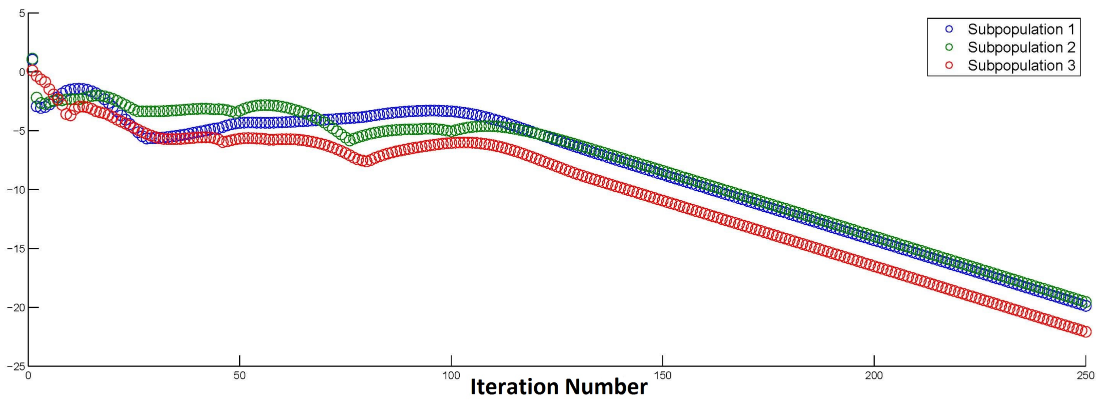

Figure 17: Covariance Errors - $\log ||\mathbf{\Sigma}_{i+1} - \mathbf{\Sigma}_i||_\infty$ (PSC)

We now implement the K-means algorithm. The suggested clustering scheme proposed is presented in Figure 18, whereas the initial parameter estimates are presented in Table 6.

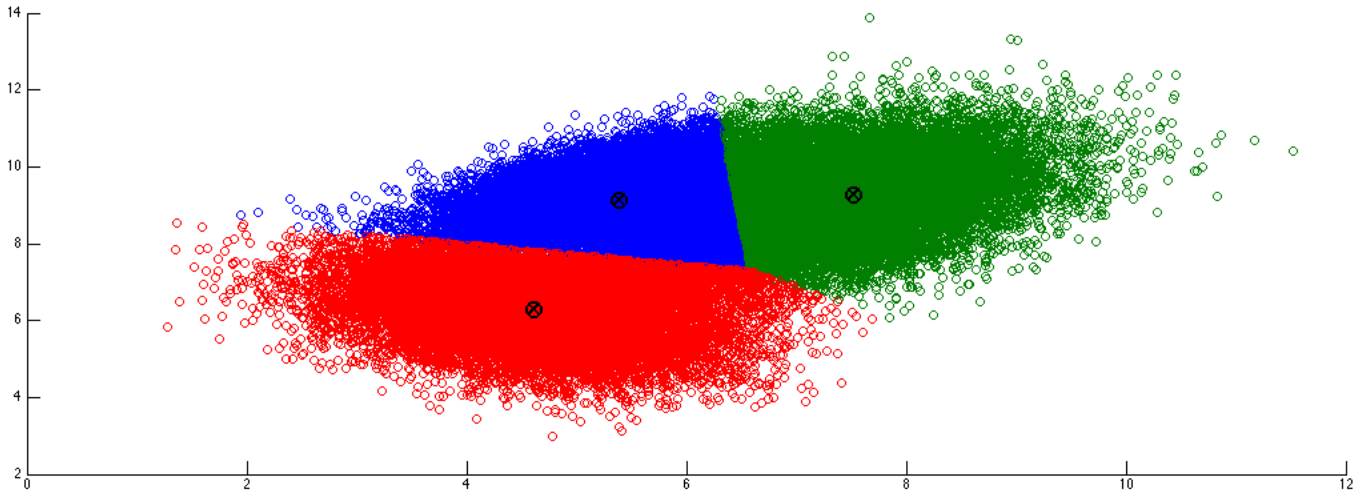

Figure 18: Applying K-Means to our Sample (PSC)

Sub-Population 1	Sub-Population 2	Sub-Population 3
$\alpha_1^0 = 0.32962$	$\alpha_2^0 = 0.34444$	$\alpha_3^0 = 0.32594$
$\mu_1^0 = \begin{pmatrix} 4.6090467 \\ 6.2835143 \end{pmatrix}$	$\mu_2^0 = \begin{pmatrix} 7.5210443 \\ 9.2780852 \end{pmatrix}$	$\mu_3^0 = \begin{pmatrix} 5.3873837 \\ 9.1293548 \end{pmatrix}$
$\mathbf{\Sigma}_1^0 = \begin{pmatrix} 0.8547355 & -0.1634364 \\ -0.1634364 & 0.6802715 \end{pmatrix}$	$\mathbf{\Sigma}_2^0 = \begin{pmatrix} 0.5272842 & 0.1720240 \\ 0.1720240 & 0.9360783 \end{pmatrix}$	$\mathbf{\Sigma}_3^0 = \begin{pmatrix} 0.4976098 & 0.0014338 \\ 0.0014338 & 0.6148206 \end{pmatrix}$

Table 6: Initial Parameter Guesses using K-Means (PSC)

As expected, our initial guesses for the poorly separated case are not as accurate as those found in the first example; however, as seen in Figures 19-20 we do experience faster convergence while using K-means. Figure 19 shows that the EM algorithm generates a reasonable approximation to the true pdf in just 25 iterations. The reader should note that after around 70 iterations there was no substantial difference between the true and approximated pdfs. We chose 150 iterations as a stopping criterion arbitrarily and allowed the algorithm to reach this maximum iteration number for convenience.

The error plots for this case mimic those of Figures 9 and 10 in the sense that there do not exist any oscillations in the beginning iterations and a strong linear pattern is present throughout the entire process. Also, as in the case of well separated data, we converge to the same values presented in Table 5; therefore, the error plots and approximated values for this case are omitted.

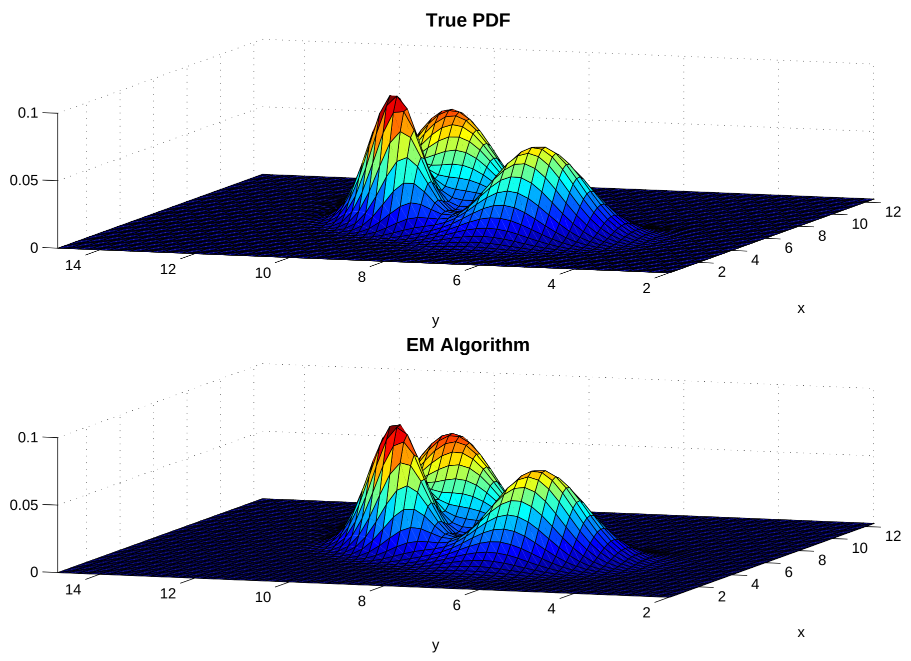

Figure 19: After 25 Iterations using K-Means

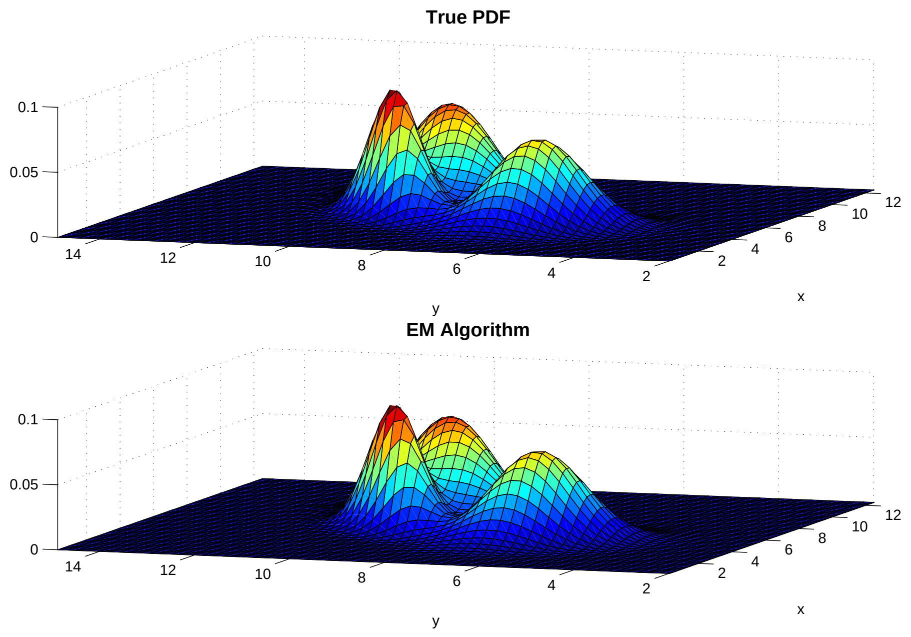

Figure 20: After 150 Iterations using K-Means

Observations

To conclude the section we believe it is necessary to discuss some of the motivations behind the error analysis as well as some drawbacks of the EM algorithm which are not readily apparent in the examples.

We first explain why throughout our error analysis we chose to use the infinity *matrix* norm. In our code presented as a supplement to this project, we see that the way we initialized all of our parameters was in a matrix format; i.e, we constructed a mean matrix which was $m \times d$. Therefore, the j^{th} row of this matrix corresponded to the means associated with the j^{th} subpopulation. We initialized our mixing probabilities and covariance matrices similarly, and we refer the reader to Appendix A to see the commented code. Thus in all of our error plots, the subscript i represents the iteration number, and it should be understood that $||\cdot||_\infty$ is a matrix norm. Since each row corresponds to a different subpopulation, it is evident that the maximum row difference would be of interest, which is why the infinity norm is used in the analysis.

One problem that may arise during the EM iterations is something referred to as "label switching" in [1]. The idea of label switching is best explained through a simple example. For the the moment we consider just the means where $m = 3$ and $d = 2$. Constructing the matrix of true means results in

$$\mu = \begin{pmatrix} \mu_{11} & \mu_{12} \\ \mu_{21} & \mu_{22} \\ \mu_{31} & \mu_{32} \end{pmatrix},$$

where if label switching occurs, we would obtain a matrix of approximate parameter values that may look something like

$$\tilde{\mu} = \begin{pmatrix} \tilde{\mu}_{31} & \tilde{\mu}_{32} \\ \tilde{\mu}_{11} & \tilde{\mu}_{12} \\ \tilde{\mu}_{21} & \tilde{\mu}_{22} \end{pmatrix}.$$

Hence, in our example, we began by labeling our subpopulations $(1, 2, 3)$, which was permuted by the EM algorithm to $(3, 1, 2)$. Or in other words, subpopulation 1 is relabeled to subpopulation 2 and so forth. Whether or not label switching is of major concern depends on whether the estimation of a particular component of a density is of interest. If the estimation of all the parameters is the main concern without emphasis on individual components, then label switching is not a problem. Throughout the course of this exposition label switching will not be a concern for us. For a more detailed discussion on this topic we refer to reader to [1] and [6].

It has been stated that the EM algorithm converges linearly and, depending on the case, may be disappointingly slow. It is clear in the case of well separated data that linear convergence is not problematic; however, as the data become increasingly more poorly separated the EM iterates may converge in an unacceptably large number of iterations. This in turn would require much more computing time. In our PSC, to iterate 250 times without using K-means for initialization took MATLAB a few minutes to finish computing, which may be costly.

Therefore, there is a need for an acceleration method to speed up the convergence of the EM iterates and reduce the computation time in the case of poorly separated data. Throughout the course of the next section, we introduce such a method and provide constructive examples that highlight the acceleration of the algorithm.

Anderson Acceleration

An acceleration method for fixed-point iterations that was developed in [7] by Donald Anderson for nonlinear integral equations is now introduced and eventually applied to the EM algorithm to expedite convergence. The framework for this section is as follows. The general algorithm for a fixed-point iteration is presented and its connection to the EM algorithm is discussed. The Anderson acceleration method is then formulated and the basic ideas of the algorithm are revealed. The section is then concluded with an example involving a dimension and sample size that are much larger than the examples mentioned in previous sections. As will be seen, the choice to use such a vast example highlights how faster convergence can be achieved by implementing Anderson acceleration within our EM iterates.

Fixed-Point Iteration

The notations to be developed in this section reflect those used in [8]. Before we introduce the acceleration method we review briefly the idea of fixed-point iteration.

Definition 3. Let $g : \mathbb{R}^d \to \mathbb{R}^d$ be given. Then a point $x \in \mathbb{R}^d$ is called a *fixed-point* for the function g if and only if $g(x) = x$.

The goal of fixed-point iteration is to solve $g(x) = x$ in an iterative manner. For any fixed-point problem we have the following general algorithm.

General Fixed-Point Iteration

```
(1) Choose an initial guess  $x_0 \in \mathbb{R}^d$.
(2) Specify an error tolerance  $\epsilon > 0$ and a maximum iteration number $I$.
(3) Iterate.
        For $k = 1, 2, ..., I$
            Set $x_k = g(x_{k-1})$.
                If $||x_k - x_{k-1}|| \leq \epsilon$
                    Return $x_k$.
                    Break.
                End If
        End For
```

Note that a fixed-point problem may also be posed as a nonlinear-equation problem of the form $f(x) = 0$ through the relationship $f(x) = g(x) - x = 0$. In many cases a fixed-point problem is formulated in this way to take advantage of existing algorithms with fast rates of convergence. Nevertheless, it is often advantageous to use fixed-point iteration in particular applications.

Before continuing, we discuss the connection between fixed-point problems and the EM algorithm. It is shown in [3, pp.8] that a maximum-likelihood estimate is a fixed-point of the EM iteration map and the often-slow convergence of the EM iterates justifies considering an acceleration method.

Acceleration Algorithm

We now present the acceleration method of interest. In its most general form, the Anderson acceleration algorithm as given in [9] and [8] is as follows.

```
Anderson Acceleration Algorithm (constrained)

(1) Given $x_0$, $m \geq 1$ and a maximum iteration number $I$.
(2) Set $x_1 = g(x_0)$.
(3) Iterate.
        For $k = 1, 2, ..., I$
            Set $m_k = \min\{m, k\}$.
            Set $F_k = (f_{k-m_k}, ..., f_k)$    where $f_j = g(x_j) - x_j$.
            Find $\alpha^{(k)} = (\alpha_0^{(k)}, ..., \alpha_{m_k}^{(k)})^T$    that solves
                $\min_{\alpha=(\alpha_0,...,\alpha_{m_k})^T} ||F_k\alpha||_2$    subject to $\sum_{j=0}^{m_k} \alpha_j = 1.$    $(\star)$
            Set $x_{k+1} = \sum_{j=0}^{m_k} \alpha_j^{(k)} g(x_{k-m_k+j})$.
        End For
```

The basic idea behind the Anderson acceleration algorithm is to make use of information gained from previous iterations. Let $x_{k-m}, ..., x_k \in \mathbb{R}^d$ denote the most current $m + 1$ iterates and, $f_{k-m}, ..., f_k \in \mathbb{R}^d$ the corresponding function evaluations. It should be noted that one should not necessarily store as many iterates as possible but try to preserve a "balance" in some sense. For example, one should keep a sufficient number of stored iterates to be able to accurately predict the next; however, we should not necessarily keep all the iterates since earlier iterations may possess less accurate information about future iterations.

We see in the general algorithm that $(\star)$ represents a constrained linear least squares problem, which may be solved in numerous ways. We now briefly explain how this problem is solved in our code, which may be found in Appendix A. As suggested in [8] and [9] we reformulate the constrained optimization problem as an equivalent *unconstrained* problem. To establish notation, we let $\Delta f_j = f_{j+1} - f_j$ and $\mathcal{F}_k = (\Delta f_{k-m_k}, ..., \Delta f_{k-1})$. It is then shown in [10] that our constrained problem is equivalent to

$$\min_{\gamma=(\gamma_0,...,\gamma_{m_k-1})^T} ||f_k - \mathcal{F}_k\gamma||_2, \tag{15}$$

where the constrained and unconstrained parameters α and γ are connected through the relationships

$$\begin{aligned} \alpha_0 &= \gamma_0, \\ \alpha_j &= \gamma_j - \gamma_{j-1}, \\ \alpha_{m_k} &= 1 - \gamma_{m_k-1}. \end{aligned}$$

for $j = 1, ..., m_k$. We now assume that we have a solution to (15), which will be denoted by $\gamma^{(k)} = (\gamma_0^{(k)}, ..., \gamma_{m_k-1}^{(k)})$. Using the relationships between our parameters, we now derive a closed form expression for the next iteration of the algorithm for our unconstrained problem. Upon direct substitution, we see that

$$
\begin{aligned}
x_{k+1} &= \sum_{j=0}^{m_k} \alpha_j^{(k)} g(x_{k-m_k+j}) \\
&= \alpha_0^{(k)} g(x_{k-m_k}) + \alpha_1^{(k)} g(x_{k-m_k+1}) + \alpha_2^{(k)} g(x_{k-m_k+2}) + \ldots + \alpha_{m_k}^{(k)} g(x_k) \\
&= \gamma_0^{(k)} g(x_{k-m_k}) + (\gamma_1^{(k)} - \gamma_0^{(k)}) g(x_{k-m_k+1}) + (\gamma_2^{(k)} - \gamma_1^{(k)}) g(x_{k-m_k+2}) + \ldots + (1 - \gamma_{m_k-1}^{(k)}) g(x_k) \\
&= g(x_k) - \gamma_0^{(k)} (g(x_{k-m_k+1}) - g(x_{k-m_k})) - \gamma_1^{(k)} (g(x_{k-m_k+2}) - g(x_{k-m_k+1})) - \ldots - \gamma_{m_k-1}^{(k)} (g(x_k) - g(x_{k-1})) \\
&= g(x_k) - \sum_{j=0}^{m_k-1} \gamma_j^{(k)} [g(x_{k-m_k+j+1}) - g(x_{k-m_k+j})] \\
&= g(x_k) - \mathcal{G}_k \gamma^{(k)},
\end{aligned}
$$

where $\Delta g_j = g(x_{j+1}) - g(x_j)$ and $\mathcal{G}_k = (\Delta g_{k-m_k}, ..., \Delta g_{k-1})$. Thus in its unconstrained form, the Anderson acceleration algorithm may be expressed in the following way.

Anderson Acceleration Algorithm (unconstrained)

```
(1) Given $x_0$, $m \geq 1$ and a maximum iteration number $I$.
(2) Set $x_1 = g(x_0)$.
(3) Iterate.
        For $k = 1, 2, ..., I$
            Set $m_k = \min\{m, k\}$.
            Find $\gamma^{(k)} = (\gamma_0^{(k)}, ...\gamma_{m_k-1}^{(k)})^T$   that solves
                $\min_{\gamma=(\gamma_0,...,\gamma_{m_k-1})^T} ||f_k - \mathcal{F}_k \gamma||_2$.
            Set $x_{k+1} = g(x_k) - \mathcal{G}_k \gamma^{(k)}$.
        End For
```

As we implement our algorithm, the unconstrained linear least squares problem is solved efficiently by appealing to QR decomposition. For each k we decompose $\mathcal{F}_k = Q_k R_k$ where each Q_k is an orthogonal matrix and each R_k is upper triangular. The code presented in Appendix A is a bit more involved than the algorithm presented above; i.e, we update the QR factors according to certain criterion as well as monitor the condition number of R_k so that $\mathcal{F}_k$ does not become ill-conditioned. For a thorough discussion as to how the rest of the code is designed, we refer the reader to [9].

Throughout the next section, we seek to show the advantages of using Anderson acceleration within the EM iterations by considering a much larger sample size and dimension than used in previous examples. As will be seen in the sequel, when the populations are sufficiently poorly separated it may be unacceptable to use the EM algorithm without acceleration due to the very slow rate of convergence.

Example

This section is the culmination of our work. We now present an example that shows how favorable it is to implement Anderson acceleration within the EM iterates. This example differs from those previously presented in several ways. We first note that in this example, we consider a much larger sample size and dimension by letting $N = 1,000,000$, $d = 10$ whereas m is kept small at $m = 2$. An initial set of well separated true parameter values will be defined; however, estimation of these parameters is not the main concern of the section. The purpose of choosing a well separated starting point is so that we may "contract" the means of the distinct subpopulations so that they approach one another and become very poorly separated. In doing this, we will be able to see how the performance of Anderson acceleration varies with the separation. The initial parameter values are given below.

Initial Parameter Values

$$\mu = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \end{pmatrix}$$

$$\alpha = (0.5, 0.5)$$

$$\mathbf{\Sigma}_i = \mathbf{I}_{d\times d} \qquad i = 1,2$$

We introduce a contraction parameter $t \in [0,1]$ and contract according to the relationship,

$$\mu^+_{ij}(t) = \bar{\mu} + t(\mu_{ij} - \bar{\mu}) \tag{16}$$

where in (16) $\bar{\mu}$ represents the overall average of the components of μ; i.e.,

$$\bar{\mu} = \frac{1}{md}\sum_{j=1}^{d}\sum_{i=1}^{m}\mu_{ij}.$$

Upon inspection of (16) we see that for $t \approx 1$, the populations remain fairly well separated; however, as $t \to 0$ the populations converge to one another. Hence, as seen by previous examples we expect the EM iterates (without acceleration) to converge slowly for small t. Before presenting the example, it is worth mentioning that we are *not* contracting the mixing proportions or covariance matrices. It has been observed that the rate of convergence depends heavily on the degree of separation in our data, so it is intuitive to contract only the location parameters.

The framework for our example is as follows. We let $t = 0.03, 0.04, ..., 1$, and comment that for $t < 0.03$ the population means are so close together that one could question the existence of two distinct subpopulations. Hence, smaller values of t are uninformative and disregarded. The EM algorithm is then run with and without acceleration with a maximum iteration number of 250. Error plots are presented and tables are displayed that keep track of the t values, the number of iterations and computation times required to reach convergence. To measure the timings, we have used MATLAB's built in tic and toc commands. Depending on the computer, these timings are likely to vary; therefore, the reader should use these timings as rough estimates of how long the algorithm runs and not precise measurements.

Before presenting the results it is necessary to make a few more annotations. In this example, our initial parameter estimates have been generated using K-means clustering only. As seen previously, when using K-means the algorithm converges fast when the populations are well separated ($t \approx 1$); therefore, we present plots and tables only for interesting values of t that require more iterations. The parameter estimates generated by the EM algorithm for these values of t are not presented in this section to avoid clutter; however, they may be found in Appendix B. The results are displayed on the following pages.

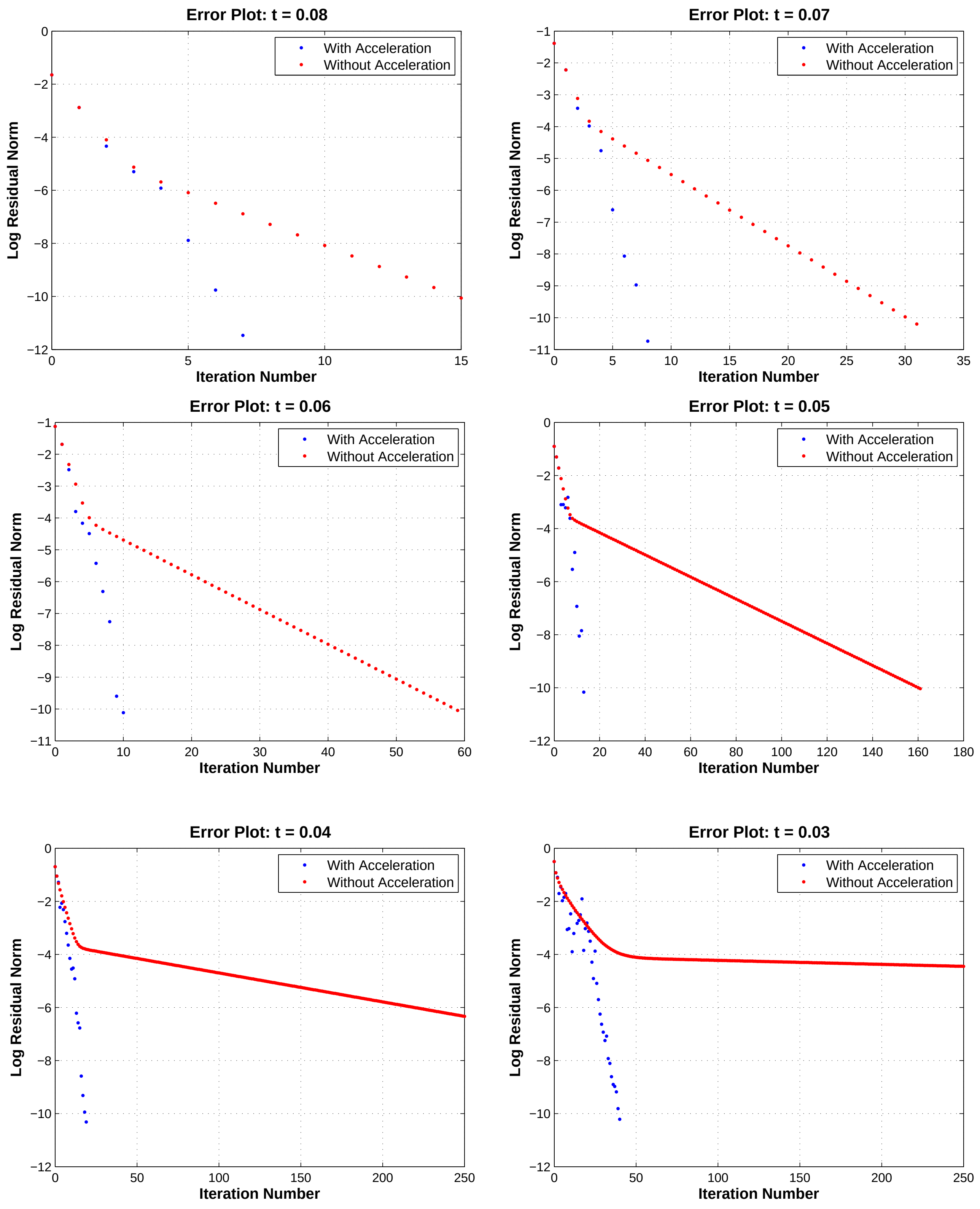

Figure 21: Plots of Performance With and Without Acceleration

t	Iteration Number	Timing (seconds)
0.08	15	240.5
0.07	31	468.5
0.06	59	869.1
0.05	161	2418.3
0.04	250	3592.2
0.03	250	3865.2

Table 7: Without Acceleration

t	Iteration Number	Timing (seconds)
0.08	7	114.3
0.07	8	139.9
0.06	10	157.9
0.05	13	218.8
0.04	19	326.9
0.03	40	585.6

Table 8: With Acceleration

The plots and tables generated provide us with significant results. It is evident that when the subpopulations are sufficiently poorly separated using EM without acceleration would be unwise. From Table 7 we see that when $t = 0.03$ the algorithm ran for over a hour while yielding errors on the order of 10^{-4}, whereas, while using acceleration, errors on the order of 10^{-10} were achieved in under ten minutes. In conclusion, we have shown that the slow convergence of the EM algorithm can be mitigated by appealing to Anderson acceleration. This may be of practical significance since the acceleration method is easily implementable and costs very little to apply.

Conclusions

The aim of this work was to acquaint one with the EM algorithm and show that applying Anderson acceleration to the EM algorithm improves rates of convergence. As preamble, the method of maximum likelihood estimation was reviewed and an example was presented. The EM algorithm was then formulated in detail, and convergence properties were discussed. The linear convergence of the EM algorithm was demonstrated through an example, and it was shown in this exposition to be unacceptably slow when the populations are poorly separated.

The Anderson acceleration method was then presented in its most general form. This method was then applied to the EM algorithm, yielding beneficial results. From Tables 7 and 8, we see that iteration numbers and run times do not increase dramatically when using acceleration, compared to the results without acceleration.

In closing, we reiterate that the EM algorithm is very useful for obtaining MLE's for various models across a broad area of applications. However, EM iterates converge slowly with poorly separated data. Through this exposition, we have introduced a way of alleviating such slow convergence in EM iterations for mixture models by introducing Anderson acceleration. This algorithm can be efficiently added to existing EM code to expedite convergence.

Future Work

There are several directions in which to continue this project. We first explain some possible problems that may arise when using acceleration. The EM iterates preserve symmetric-positive-definiteness of each $\mathbf{\Sigma}_i$ and maintain that each α_i is non-negative and constrained to sum to one. However, the Anderson acceleration algorithm need not preserve these special attributes. Throughout this project, no problems occurred with the mixture proportions; however, when poor initial parameter estimates were chosen, symmetric-positive-definiteness was lost in some experiments. This is the reason why in our concluding example we chose to use K-means. Having a better initial guess seems to ensure that each $\mathbf{\Sigma}_i$ remains symmetric-positive definite. In further research a remedy will be sought that will allow one to choose almost any initial starting guess for our parameters while maintaining this attribute.

Throughout this project, all random samples were computer generated. In further works, we will look for an opportunity to use acceleration with data that were collected in some real world application.

Parallel computing could be investigated in hopes of allowing significantly larger values of N, m and d. It would be interesting and useful to investigate how acceleration behaves on much larger sets of much higher-dimensional data.

Bibliography

[1] H. Walker and R. Redner, “Mixture Densities, Maximum Likelihood and the EM algorithm”, *SIAM Review*, vol. 26, April 1984, pp.195-239.

[2] G. Casella and R. Berger, “Statistical Inference”, *Duxbury Press*, 2nd ed. (2002).

[3] A. Dempster, N. Laird and D. Rubin, “Maximum-likelihood from incomplete data via the EM algorithm” *Journal of the Royal Statistical Society* , Series B (methodological), 39 (1977), pp. 1-38.

[4] J. Blimes, “A Gentle Tutorial of the EM algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models” University of California – Berkeley International Computer Science Institute Technical Report TR-97-021, 1997.

[5] C. F. J. Wu “On the Convergence Properties of the EM algorithm” *The Annals of Statistics*, vol. 11, March 1983, pp. 95-103.

[6] M. Stephens “Dealing with label switching in mixture models” *Journal of the Royal Statistical Society*, 62 B, 2000, pp. 795-809.

[7] D. Anderson “Iterative procedures for nonlinear integral equations” *Journal Association for Computer Machinery*, 12, 1965, pp. 547-560.

[8] H. Walker and P. Ni “Anderson acceleration for fixed-point iterations” *SIAM Journal of Numerical Analysis*, 49 (2011), pp. 1715-1735.

[9] H. Walker “Anderson Acceleration: Algorithms and Implementations” WPI Math. Sciences Dept. Report MS-6-15-50, June 2011.

[10] H. Fang and Y.Saad, “Two classes of multisecant methods for nonlinear acceleration” *Numerical Linear Algebra with Applications*, 16, 197-221, 2009.

Appendix A

```matlab

% The EM algorithm for Gaussian Mixture Models
% Worcester Polytechnic Institute
% Joshua Plasse
% jhplasse@wpi.edu
% Below is the MATLAB script that was used
% for the examples found throughout the project %%%%%
%
%------------------------------------------------------------------------------------------------%

%% Well Separated Case (WSC)
clc
format longG
N = 50000;
mu = [1 2.25; 12.9 10.3];
alpha = [.3 .7];
sigma(:,:,1) = [1.75 1 ;1 1.75];
sigma(:,:,2) = [3.2 1.25;1.25 3.2];
[muFinal,sigmaFinal,alphaFinal,meanError,sigmaError,alphaError] = em_driver_MultiVariate(N,
    alpha,mu,sigma,250);

% Plot Mean Errors
figure
scatter(1:length(meanError),log(meanError),100)
xlabel('Iteration Number','FontSize',15,'FontWeight','bold')
% Plot Alpha Errors
figure
scatter(1:length(meanError),log(alphaError),100)
xlabel('Iteration Number','FontSize',15,'FontWeight','bold')
% Plot Covariance Errors
figure
for(i = 1:length(sigmaError(1,:)))
    scatter(1:length(sigmaError(:,1)),log(sigmaError(:,i)),100)
    hold on
end
xlabel('Iteration Number','FontSize',15,'FontWeight','bold')

%% Poorly Separated Case (PSC)
clc
format longG
N = 50000;
mu = [4.5 6.25; 7 8.95; 5.12 9.5];
alpha = [.3 .5 .2];
sigma(:,:,1) = [.75 -0.25;-0.25 .75];
sigma(:,:,2) = [1.1 .5;.5 1.1];
sigma(:,:,3) = [.45 .3;.3 .45];
[muFinal,sigmaFinal,alphaFinal,meanError,sigmaError,alphaError] = em_driver_MultiVariate(N,
    alpha,mu,sigma,250);

% Plot Mean Errors
figure
scatter(1:length(meanError),log(meanError),50)
xlabel('Iteration Number','FontSize',15,'FontWeight','bold'
% Plot Alpha Errors
figure
```

```
scatter(1:length(meanError),log(alphaError),100)
xlabel('Iteration Number','FontSize',15,'FontWeight','bold')
% Plot Covariance Errors
figure
for(i = 1:length(sigmaError(1,:)))
    scatter(1:length(sigmaError(:,1)),log(sigmaError(:,i)),100)
    hold on
end
xlabel('Iteration Number','FontSize',15,'FontWeight','bold')

%% Anderson Acceleration Example
clc
format longG
N = 1000000;
mu = [1:10;21:30];
alpha = [.5 .5];
sigma(:,:,1) = eye(10);
sigma(:,:,2) = eye(10);
[muEstimates,sigmaEstimates,alphaEstimates,timeIter,timeIter_noAcc,muContract] =
    EM_Acceleration_Driver(N,alpha,mu,sigma,250);
```

```
function y = mvgmmrnd(mu,sigma,p,n)
% This script generates the random samples used throughout the project
% MVGMMRND - Multivariate Gaussian Mixture Model Random Sample
% MVGMMRND Random vectors from a mixture of multivariate normals.
% MU is an M-by-D matrix of means for the M component normals
% SIGMA is a D-by-D-by-M array of covariance matrices for the
% M component normals.
% P is an M-by-1 vector of component mixing probabilities.
% N is the desired number of random vectors.

[M,d] = size(mu);
% randomly pick from the components
[dum,compon] = histc(rand(n,1), [0; cumsum(p(:))./sum(p)]);
% generate random vectors from the selected components with a
% "stacked" matrix multiply
for i = 1:M
     Rt(i,:,:) = chol(sigma(:,:,i)); % holds the transposed cholesky factors
end
Z = repmat(randn(n,d), [1,1,d]);
y = squeeze(sum(Z.*Rt(compon,:,:),2)) + mu(compon,:);

% Reference:
%%%%          This code was found on the MathWorks site
%%%% http://www.mathworks.com/matlabcentral/newsreader/view_thread/36174
%%%%            From the thread author Peter Perkins
```

```
function  [phiplus,covplus,alphaplus,meanError,covErrorMatrix,alphaError] =
    em_driver_MultiVariate(N,alpha,mu,sigma,itmax)
%
% This is the driver to demonstrate the EM
% algorithm for approximating MLEs of the parameters in a mixture
% of normal (multivariate) distributions
%
%
% Inputs:
%   N          = number of observations
%   alpha      = 1xm-vector of proportions
%   mu         = mxd matrix where m is the number of subpopulations and d is
%                the dimension we are working in. The ith row of mu should
%                correspond to the true means for the ith subpopulation
%   sigma      = is a m page multidimensional array where the ith page
%                stores the covariance matrix for the ith subpopulation
%   itmax      = maximum allowable number of iterations.
%
% Outputs:
% phiplus    = next iterate for the means
% covplus    = next iterate for the covariances
% alphaplus  = next iterate for the mixing proportions
% meanError  = vector containing errors within the iterates
% covErrorMatrix = matrix containing errors within the iterates
% alphaError = vector containing errors within the iterates

% Declare global variables:
global alpha_trueGL;
global mu_trueGL;
global sigma_trueGL;
global mGL;
global XGL;
global dGL;

% These allow the user to enter in as little as 4 arguments into
% em_driver and defines the other arguments below
if nargin<4, error('em_mean_driver requires at least four arguments.');end
if nargin<5, itmax = 100; end

m = length(mu(:,1)); % Number of sub-populations
d = length(mu(1,:)); %Dimension we are working in

% Set the seed of random
rng(31415,'v4');
% Generate the random sample using the function mvgmmrnd
X = mvgmmrnd(mu,sigma,alpha,N);
% Present Scatter Plot of the Sample
figure
scatter(X(:,1),X(:,2));
title('Scatter Plot without Clustering')

% Set more global variables to pass to g_em_MultiVariate
alpha_trueGL = alpha;
mu_trueGL = mu;
sigma_trueGL = sigma;
mGL = m;
XGL = X;
dGL = d;

% Calculates the centroids of the clusters using kmeans algorithm
[idx,phi0] = kmeans(X,m);

% Uncomment the following to randomly guess at our means
% s = rng;
% phi0 = rand(size(phi0));

% Initialize initial cov matrices/sample proportion vector
intCov = zeros(size(sigma));
```

```
intAlpha = size(alpha);

% Calculuate the initial covariance and sample proportions
for(i = 1:m)
    temp = find(idx==i);
    intCov(:,:,i) = cov(X(temp,:));
    intAlpha(i) = length(temp)/length(X(:,1));
end

% Uncomment to randomly guess at alpha's and sigma's
% intAlpha = ones(1,length(intAlpha))/m;
% for(i=1:m)
%     intCov(:,:,i) = zeros(d,d)+eye(d);
% end

% Specify an error tolerance
% Essentially we iterate until the error between iterates drops below
% machine epsilon
etol = 10^(-30);

% If d>2 we cannot produce plots
     if(d == 2)
      figure
        for(i = 1:length(alpha))
        scatter(X(idx==i,1),X(idx==i,2))
        hold on
        end
    plot(phi0(:,1),phi0(:,2),'kx',...
     'MarkerSize',12,'LineWidth',2)
    plot(phi0(:,1),phi0(:,2),'ko',...
     'MarkerSize',12,'LineWidth',2)
    title('Calculating Centroids for our Initial Guesses')
    hold off

 % Plots the true pdf
 figure
 subplot(2,1,1)
 xMin = min(XGL(:,1)); xMax = max(XGL(:,1));
 yMin = min(XGL(:,2)); yMax = max(XGL(:,2));
 obj = gmdistribution(mu,sigma,alpha);
 ezsurf(@(x,y)pdf(obj,[x y]),[xMin-1,xMax+1],[yMin-1 yMax+1])
 title('True PDF','FontSize',12.5,'FontWeight','bold');

% Run EM iterates
  for (i = 1:itmax)

% Iterate on the cholesky factors of our covariance matrices
          for(j = 1:m)
                cholCov(:,:,j) = chol(intCov(:,:,j),'lower');
          end
    [phiplus,cholCovPlus,alphaplus] = g_em_MultiVariate(phi0,cholCov,intAlpha);
    meanError(i) = norm(phi0 - phiplus,inf);
    alphaError(i) = norm(intAlpha - alphaplus,inf);

        % Reconstruct the covariance matrices
          for(j = 1:m)
            covplus(:,:,j) = cholCovPlus(:,:,j)*cholCovPlus(:,:,j)';
          end

         % Compute Covariance Errors
          for(j = 1:m)
            covError(j) = norm(intCov(:,:,j)-covplus(:,:,j),inf);
          end
     covErrorMatrix(i,:) = covError;

    if(meanError(i)<=etol && alphaError(i)<=etol)
        phi0 = phiplus;
        intCov = covplus;
        intAlpha = alphaplus;
    break
    end
    phi0 = phiplus;
    intCov = covplus;
    intAlpha = alphaplus;
    end

else
% For d>2 we simply iterate the EM algorithm without plotting
% Run EM iterates

  for (i = 1:itmax)
% Iterate on the cholesky factors of our covariance matrices
        for(j = 1:m)
              cholCov(:,:,j) = chol(intCov(:,:,j),'lower');
        end
    [phiplus,cholCovPlus,alphaplus] = g_em_MultiVariate(phi0,cholCov,intAlpha);
    meanError(i) = norm(phi0 - phiplus,inf);
    alphaError(i) = norm(intAlpha - alphaplus,inf);

       % Reconstruct the covariance matrices
        for(j = 1:m)
          covplus(:,:,j) = cholCovPlus(:,:,j)*cholCovPlus(:,:,j)';
        end
        % Compute Covariance Errors
        for(j = 1:m)
          covError(j) = norm(intCov(:,:,j)-covplus(:,:,j),inf);
        end
     covErrorMatrix(i,:) = covError;

    if(meanError(i)<=etol && alphaError(i)<=etol)
        phi0 = phiplus;
        intCov = covplus;
        intAlpha = alphaplus;
    break
    end
    phi0 = phiplus;
    intCov = covplus;
    intAlpha = alphaplus;
    end
    end
end
```

```
function [phiplus,cholCovplus,alphaplus] = g_em_MultiVariate(phi,cholCov,intAlpha)

%%%%%%%%%%% Performs the EM Iterates for multivariate Guassians %%%%%%%%%%%
% Inputs: phi = mean estimate at the ith iteration
%         cholCov = Cholesky factor of the covariance matrices at ith
%         iteration
%         intAlpha = mixing proportion estimates at the ith iteration
% Outputs: Updated parameter estimates
format longG

% Declare variables
global alpha_trueGL;
global mu_trueGL;
global sigma_trueGL;
global mGL;
global XGL;
global dGL;
persistent N;
X = XGL;
d = dGL;
N = length(X(:,1));      % Set N = sample size

% Top of the EM iteration.
C = (2*pi)^(d/2);
m = length(intAlpha);
em_posteriors = zeros(m,N);

% Compute the EM posterior probabilities
for(i = 1:m)
  duplicateMu = repmat(phi(i,:),length(X(:,1)),1);
  L = cholCov(:,:,i);
  frac = intAlpha(i)/(C*prod(diag(L)));
  Y = L\(X-duplicateMu)'; % Solves LY = (X-duplicateMu)' columwise, so Y = inv(L)*((X-
      duplicateMu)').
 em_posteriors(i,:) = frac*exp(-2\sum(Y.^2)); %Forms a 1xN vector with components alpha_i*
     p_i(x_k).
end

% Construct alpha_i*p_i(x_k|phi_i)/p(x_k|PHI)
em_posteriors = (ones(m,1)*sum(em_posteriors)).\em_posteriors;

denom = sum(em_posteriors');
num = em_posteriors*X;
% Update the means.
phiplus = diag(denom)\num;
% Update the alphas
alphaplus = (1/N).*denom;

% Update the covariance matrices
for (i =1:m)
    temp = zeros(d,d);
    for(k = 1:length(X(:,1)))
        temp = temp + ((X(k,:)-phiplus(i,:))'*(X(k,:)-phiplus(i,:)))*em_posteriors(i,k);
    end
    % Constructs the 'full' covariance matrix
    covplus(:,:,i) = temp/denom(i);
    % Return the cholesky factor
    cholCovplus(:,:,i) = chol(covplus(:,:,i),'lower');
end
% Cannot plot if d>2
if(d ==2)
% Plot the current mixture PDF.
subplot(2,1,2)
xMin = min(XGL(:,1)); xMax = max(XGL(:,1));
yMin = min(XGL(:,2)); yMax = max(XGL(:,2));
obj = gmdistribution(phiplus,covplus,alphaplus);
em_plot = ezsurf(@(x,y)pdf(obj,[x y]),[xMin-1,xMax+1],[yMin-1 yMax+1]);
title('EM algorithm','FontSize',12.5,'FontWeight','bold')
pause(.0001);
end
end
```

```
function [muEstimates,sigmaEstimates,alphaEstimates,timeIter,timeIter_noAcc,muContract] =
    EM_Acceleration_Driver(N,alpha,mu,sigma,itmax,t)

%%% The purpose of this driver is to show how the Anderson Acceleration %%%
%%%%%% method speeds up the convergence rate of the EM algorithm %%%%%%

% Inputs:
%   N          = number of observations
%   alpha      = 1xm-vector of proportions
%   mu         = mxd matrix where m is the number of subpopulations and d is
%                the dimension we are working in. The ith row of mu should
%                correspond to the true means for the ith subpopulation
%   sigma      = is a m page multidimensional array where the ith page
%                stores the covariance matrix for the ith subpopulation
%   itmax     = maximum allowable number of iterations.
%   t         = a range of values between 0 and 1 that are used to
%             "contract" the means of our sample, where lower values of t
%              will correspond to the sample becomming more poorly separated

% Initialize some global variables
global alpha_trueGL;
global mu_trueGL;
global sigma_trueGL;
global mGL;
global XGL;
global dGL;
% Number of argument errors
if nargin<4, error('EM_Acceleration_Driver requires at least four arguments.');end
if nargin<5, itmax = 100; end
if nargin<6, t = [0.03:0.01:1]; end

m = length(mu(:,1)); % Number of sub-populations
d = length(mu(1,:)); % Dimension we are working in
% Initialize time table arrays
timeCount = zeros(1,length(t));
itNum = zeros(1,length(t));

% Store a string for our error plots
str = 'Error Plot: ';
% Set the seed of random
rng(312415,'v4');

% Take the average of our mean matrix row wise
muBar = mean2(mu);
% Flip the t vector so we start off at the best case
t = fliplr(t);

% Performs the contraction of our means
for(k = 1:length(t))

    for(i = 1:m)
        for(j = 1:d)
            muPlus(i,j) = muBar + t(k)*(mu(i,j)-muBar);
        end
    end

    % Generate the random sample using the function mvgmmrnd
    X = mvgmmrnd(muPlus,sigma,alpha,N);
    % Store all of the contracted means
    muContract(:,:,k) = muPlus;

    alpha_trueGL = alpha;
    mu_trueGL = mu;
    sigma_trueGL = sigma;
    mGL = m;
    XGL = X;
    dGL = d;
 % Use K-means on the sample drawn
    [idx,phi0] = kmeans(X,m);
% Initialize cov matrices/sample proportion vector
    intCov = zeros(size(sigma));
    intAlpha = size(alpha);
    % Calculuate the initial covariance and sample proportions
    for(i = 1:m)
        temp = find(idx==i);
        intCov(:,:,i) = cov(X(temp,:));
        intAlpha(i) = length(temp)/length(X(:,1));
    end
    % Make a copy of the initial parameters so we do not write over them
    % when we use no accelration
    intCov_noAcc = intCov;
    intAlpha_noAcc = intAlpha;
    phi0_noAcc = phi0;

    %%% Implement Anderson Acceleration %%%
    % In hopes to preserve PD of the covariance matrix we iterate on its
    % Cholesky factors
    for(i = 1:m)
       cholCov(:,:,i) = chol(intCov(:,:,i),'lower');
    end

    tic
    [phiplus,cholCovPlus,alphaplus,iter,res_hist] =  AndAcc(@g_em_mean_MultiVariate,phi0,
        cholCov,intAlpha,10,itmax);
    muEstimates(:,:,k) = phiplus;
    alphaEstimates(:,:,k) = alphaplus;
    timeCount(k) = toc;
    itNum(k) = iter;
    % Reconstruct our covariance matrices from the Cholesky factors
    for(i = 1:m)
        covplus(:,:,i) = cholCovPlus(:,:,i)*cholCovPlus(:,:,i)';
    end
    sigmaEstimates{k} = covplus;

    % Iterate without acceleration
    for(i = 1:m)
       cholCov_noAcc(:,:,i) = chol(intCov_noAcc(:,:,i),'lower');
    end
    tic
    [phiplus_noAcc,cholCovPlus_noAcc,alphaplus_noAcc,iter_noAcc,res_hist_noAcc] =  AndAcc(
        @g_em_mean_MultiVariate,phi0_noAcc,cholCov_noAcc,intAlpha_noAcc,0,itmax);
    timeCount_noAcc(k) = toc;
    itNum_noAcc(k) = iter_noAcc;

    % Reconstruct our covariance matrices from the Cholesky factors
    for(i = 1:m)
        covplus_noAcc(:,:,i) = cholCovPlus_noAcc(:,:,i)*cholCovPlus_noAcc(:,:,i)';
    end

    % Error Plots
    if(iter>=5 && iter_noAcc >= 10)
        figure
        plot(res_hist(:,1),log10(res_hist(:,2)),'b.','LineWidth',0.4,'MarkerSize',8.0);
        hold on
        plot(res_hist_noAcc(:,1),log10(res_hist_noAcc(:,2)),'r.','LineWidth',0.4,'MarkerSize
            ',8.0);
        grid
        ylabel('Log Residual Norm','FontSize',12,'FontWeight','bold')
        xlabel('Iteration Number','FontSize',12,'FontWeight','bold')
        title([str '' 't = ' num2str(t(k))],'FontSize',13,'FontWeight','bold')
        leg = legend('With Acceleration','Without Acceleration');
        set(leg,'FontSize',11)
    end
end
% Construct the t/iteration/time tables
timeIter = [t',itNum',timeCount'];
timeIter_noAcc = [t',itNum_noAcc',timeCount_noAcc'];
end
```

```
function [phi0,cholCov,intAlpha,iter,res_hist] = AndAcc(g,phi0,cholCov,intAlpha,mMax,itmax,
    atol,rtol,droptol,beta,AAstart)
% This performs fixed-point iteration with or without Anderson
% acceleration for a given fixed-point map g and initial
% approximate solution x.
%
% Required inputs:
%   g  = fixed-point map (function handle); form gval = g(x).
%   x  = initial approximate solution (column vector).
%
% Optional inputs:
%   mMax    = maximum number of stored residuals (non-negative integer).
%                     NOTE: mMax = 0 => no acceleration.
%   itmax    = maximum allowable number of iterations.
%   atol       = absolute error tolerance.
%   rtol        = relative error tolerance.
%   droptol  = tolerance for dropping stored residual vectors to improve
%                     conditioning: If droptol > 0, drop residuals if the
%                     condition number exceeds droptol;  if droptol <= 0,
%                     do not drop residuals.
%   beta      = damping factor: If beta > 0 (and beta ~= 1), then the step is
%                     damped by beta; otherwise, the step is not damped.
%                     NOTE: beta can be a function handle; form beta(iter), where iter is
%                     the iteration number and 0 < beta(iter) < 1.
%   AAstart = acceleration delay factor: If AAstart > 0, start acceleration
%                     when iter = AAstart.
%
% Output:
%   x            = final approximate solution.
%   iter         = final iteration number.
%   res_hist  = residual history matrix (iteration numbers and residual norms).
%
% Homer Walker (walker@wpi.edu), 10/14/2011.
% Updated to work for Multivariate Gaussian Distributions,
% Joshua Plasse (jhplasse@wpi.edu), 4/1/2013
% For EM iterations.
global iterGL;
m = length(phi0(:,1));
d = length(phi0(1,:));

% Set the method parameters.
if nargin < 4, error('AndAcc requires at least two arguments.'); end
if nargin < 5, mMax = 10; end
if nargin < 6, itmax = 100; end
if nargin < 7, atol = 1.e-10; end
if nargin < 8, rtol = 1.e-10; end
if nargin < 9, droptol = 1.e10; end
if nargin < 10, beta = 1; end
if nargin < 11, AAstart = 0; end

% Initialize the storage arrays.
res_hist = [];   % Storage of residual history.
DG = [];          % Storage of g-value differences.
% Initialize printing.
if mMax == 0
  fprintf('\n No acceleration.');
elseif mMax > 0
  fprintf('\n Anderson acceleration, mMax = %d \n',mMax);
else
  error('AndAcc.m: mMax must be non-negative.');
end
fprintf('\n iter    res_norm  \n');
% Initialize the number of stored residuals.
mAA = 0;
% Top of the iteration loop.
for iter = 0:itmax
% Apply g and compute the current residual norm.
iterGL = iter;

[muAccPlus,cholCovAccPlus,alphaAccPlus] = g(phi0,cholCov,intAlpha);

% This code is specified for a column vector x; therefore, we must reshape
% our mu matrix, alpha vector, and sigma 3d array to be represented as a
% vector

% Reshape the initial means
a = (reshape(phi0',1,m*d))';
% Reshape the initial proportions
b = intAlpha';
% Reshape the initial covariances
for(i =1:m)
    temp{i} = (reshape(cholCov(:,:,i)',1,d*d))';
end
c = (cell2mat(temp.'));
x = [a;b;c];

% Reshape the updated means
aa = (reshape(muAccPlus',1,m*d))';
% Reshape the updated proportions
bb = alphaAccPlus';
% Reshape the updated covariances
for(i =1:m)
    foo{i} = (reshape(cholCovAccPlus(:,:,i)',1,d*d))';
end
cc = (cell2mat(foo.'));
gval = [aa;bb;cc];
fval = gval - x;
res_norm = norm(fval);
fprintf(' %d    %e  \n', iter, res_norm);
res_hist = [res_hist;[iter,res_norm]];
 % Set the residual tolerance on the initial iteration.
  if iter == 0, tol = max(atol,rtol*res_norm); end

  % Test for stopping.
  if res_norm <= tol,
    fprintf('Terminate with residual norm = %e \n\n', res_norm);
    break;
  end

  if mMax == 0 || iter < AAstart,
    % Without acceleration, update x <- g(x) to obtain the next
    % approximate solution.
    x = gval;
  else
    % Apply Anderson acceleration.

    % Update the df vector and the DG array.
    if iter > AAstart,
      df = fval-f_old;
      if mAA < mMax,
        DG = [DG gval-g_old];
      else
        DG = [DG(:,2:mAA) gval-g_old];
      end
     mAA = mAA + 1;
    end
    f_old = fval;
    g_old = gval;

    if mAA == 0
      % If mAA == 0, update x <- g(x) to obtain the next approximate solution.
      x = gval;
    else
      % If mAA > 0, solve the least-squares problem and update the
      % solution.
      if mAA == 1
        % If mAA == 1, form the initial QR decomposition.
        R(1,1) = norm(df);
        Q = R(1,1)\df;
      else

 % If mAA > 1, update the QR decomposition.
        if mAA > mMax
          % If the column dimension of Q is mMax, delete the first column and
          % update the decomposition.
          [Q,R] = qrdelete(Q,R,1);
          mAA = mAA - 1;
          % The following treats the qrdelete quirk described below.
          if size(R,1) ~= size(R,2),
            Q = Q(:,1:mAA-1); R = R(1:mAA-1,:);
          end
          % Explanation: If Q is not square, then qrdelete(Q,R,1) reduces the
          % column dimension of Q by 1 and the column and row
          % dimensions of R by 1. But if Q *is* square, then the
          % column dimension of Q is not reduced and only the column
          % dimension of R is reduced by one. This is to allow for
          % MATLAB's default "thick" QR decomposition, which always
          % produces a square Q.
        end
        % Now update the QR decomposition to incorporate the new
        % column.
        for j = 1:mAA - 1
          R(j,mAA) = Q(:,j)'*df;
          df = df - R(j,mAA)*Q(:,j);
        end
        R(mAA,mAA) = norm(df);
        Q = [Q,R(mAA,mAA)\df];
      end
      if droptol > 0
        % Drop residuals to improve conditioning if necessary.
        condDF = cond(R);
        while condDF > droptol && mAA > 1
          fprintf('    cond(D) = %e, reducing mAA to %d \n', condDF, mAA-1);
          [Q,R] = qrdelete(Q,R,1);
          DG = DG(:,2:mAA);
          mAA = mAA - 1;
          % The following treats the qrdelete quirk described above.
          if size(R,1) ~= size(R,2),
            Q = Q(:,1:mAA); R = R(1:mAA,:);
          end
          condDF = cond(R);
        end
      end
      % Solve the least-squares problem.
      gamma = R\(Q'*fval);
      % Update the approximate solution.
      x = gval - DG*gamma;
      % Apply damping if beta is a function handle or if beta > 0
      % (and beta ~= 1).
      if isa(beta,'function_handle'),
        x = x - (1-beta(iter))*(fval - Q*R*gamma);
      else
        if beta > 0 && beta ~= 1,
          x = x - (1-beta)*(fval - Q*R*gamma);
        end
      end
    end
  end
% Update the parameters
% 'Inverse' reshape
a = x(1:m*d);
phi0 = reshape(a',d,m)';
intAlpha = x(m*d+1:m*d+m)';
% Delete the indices that don't correspond to sigma entries
ind = 1:length(a)+length(intAlpha);
x(ind) = [];

for(i = 1:m)
    temp1 = x((i-1)*d^2+1:i*d^2);
    cholCov(:,:,i) = reshape(temp1',d,d)';
end
end

% Bottom of the iteration loop.
if res_norm > tol && iter == itmax,
  fprintf('\n Terminate after itmax = %d iterations. \n', itmax);
  fprintf(' Residual norm = %e \n\n', res_norm);
end
end
```

Appendix B

For $t = 0.08$:

Contracted Means

$$\mu = \begin{pmatrix} 14.34 & 14.42 & 14.5 & 14.58 & 14.66 & 14.74 & 14.82 & 14.9 & 14.98 & 15.06 \\ 15.94 & 16.02 & 16.1 & 16.18 & 16.26 & 16.34 & 16.42 & 16.5 & 16.58 & 16.66 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 14.339 & 14.420 & 14.499 & 14.581 & 14.659 & 14.740 & 14.819 & 14.898 & 14.980 & 15.060 \\ 15.942 & 16.023 & 16.098 & 16.181 & 16.259 & 16.338 & 16.421 & 16.503 & 16.579 & 16.661 \end{pmatrix}$$
$$\alpha = (0.50077, 0.49923)$$

For $t = 0.07$:

Contracted Means

$$\mu = \begin{pmatrix} 14.485 & 14.555 & 14.625 & 14.695 & 14.765 & 14.835 & 14.905 & 14.975 & 15.045 & 15.115 \\ 15.885 & 15.955 & 16.025 & 16.095 & 16.165 & 16.235 & 16.305 & 16.375 & 16.445 & 16.515 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 15.884 & 15.950 & 16.027 & 16.094 & 16.165 & 16.233 & 16.304 & 16.373 & 16.441 & 16.515 \\ 14.486 & 14.554 & 14.622 & 14.693 & 14.767 & 14.833 & 14.903 & 14.975 & 15.045 & 15.116 \end{pmatrix}$$
$$\alpha = (0.50037, 0.49963)$$

For $t = 0.06$:

Contracted Means

$$\mu = \begin{pmatrix} 14.63 & 14.69 & 14.75 & 14.81 & 14.87 & 14.93 & 14.99 & 15.05 & 15.11 & 15.17 \\ 15.83 & 15.89 & 15.95 & 16.01 & 16.07 & 16.13 & 16.19 & 16.25 & 16.31 & 16.37 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 14.628 & 14.689 & 14.749 & 14.810 & 14.869 & 14.931 & 14.989 & 15.049 & 15.111 & 15.171 \\ 15.831 & 15.892 & 15.952 & 16.009 & 16.071 & 16.128 & 16.191 & 16.251 & 16.309 & 16.367 \end{pmatrix}$$
$$\alpha = (0.50039, 0.49960)$$

For $t = 0.05$:

Contracted Means

$$\mu = \begin{pmatrix} 14.775 & 14.825 & 14.875 & 14.925 & 14.975 & 15.025 & 15.075 & 15.125 & 15.175 & 15.225 \\ 15.775 & 15.825 & 15.875 & 15.925 & 15.975 & 16.025 & 16.075 & 16.125 & 16.175 & 16.225 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 15.777 & 15.824 & 15.875 & 15.925 & 15.975 & 16.023 & 16.078 & 16.124 & 16.176 & 16.227 \\ 14.774 & 14.822 & 14.875 & 14.923 & 14.973 & 15.024 & 15.077 & 15.124 & 15.174 & 15.225 \end{pmatrix}$$
$$\alpha = (0.50086, 0.49914)$$

For $t = 0.04$:

Contracted Means

$$\mu = \begin{pmatrix} 14.92 & 14.96 & 15 & 15.04 & 15.08 & 15.12 & 15.16 & 15.2 & 15.24 & 15.28 \\ 15.72 & 15.76 & 15.8 & 15.84 & 15.88 & 15.92 & 15.96 & 16 & 16.04 & 16.08 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 15.720 & 15.758 & 15.799 & 15.839 & 15.878 & 15.917 & 15.960 & 16.004 & 16.039 & 16.078 \\ 14.919 & 14.959 & 14.998 & 15.037 & 15.081 & 15.117 & 15.158 & 15.199 & 15.237 & 15.282 \end{pmatrix}$$
$$\alpha = (0.50191, 0.49809)$$

For $t = 0.03$:

Contracted Means

$$\mu = \begin{pmatrix} 15.065 & 15.095 & 15.125 & 15.155 & 15.185 & 15.215 & 15.245 & 15.275 & 15.305 & 15.335 \\ 15.665 & 15.695 & 15.725 & 15.755 & 15.785 & 15.815 & 15.845 & 15.875 & 15.905 & 15.935 \end{pmatrix}$$

EM Estimates

$$\mu = \begin{pmatrix} 15.059 & 15.094 & 15.122 & 15.147 & 15.184 & 15.208 & 15.240 & 15.271 & 15.299 & 15.331 \\ 15.658 & 15.691 & 15.721 & 15.751 & 15.779 & 15.813 & 15.839 & 15.870 & 15.899 & 15.930 \end{pmatrix}$$
$$\alpha = (0.49301, 0.50699)$$