Homework 1: Expectation Maximization

1 Introduction

Expectation Maximization (EM) is an iterative process for determining the parameters of a statistical model based on data sampled from that model. In this assignment we use a Gaussian Mixture model to generate sample data, and then assume this model in our construction of the EM algorithm. It will be shown that in suitable circumstances the EM algorithm performs efficiently; that is, when the probability density functions $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ have sufficient distance between μ_1 and μ_2 and/or sufficiently small covariance determinants, the EM algorithm will generally converge within a small number of iterations.

2 Implementation

The EM algorithm is implemented using Matlab and is run on sets of randomly generated data. Two Gaussian distributions are used, with randomly-generated ϕ, μ, and Σ values where ϕ represents the probability of choosing that distribution. Thus, $\phi_1 = 1 - \phi_2$. The implementation follows the textbook definitions for each step with a few modifications. The initial "guess" is achieved by simply sorting and partitioning the sample data along the x-axis: points whose x-coordinates appear in the first half of the sort are placed into the first set, and all others are placed in the second. This method clearly makes the algorithm more efficient when a large horizontal discrepancy exists between μ_1 and μ_2, so Section 3 will show that even when μ_1 and μ_2 have identical x-coordinates the algorithm still performs well.

Additionally, the algorithm suggets two alternatives for convergence testing: computing changes in observation assignments (i.e. $z^{(i)}$ values), or changes in weights ($w^{(i)}$ values). This will be explored further in the next section.

3 Experiments

Each experiment in this section makes use of the base parameters in Figure 3.1, with a unique sample set drawn once per trial. One such set, along with a sample of the algorithm output, is visualized in Figure 3.2. The ellipses shown are based on the μ and Σ values. The code for these ellipses was taken from [1].

These parameters were generated as follows:

```
1 phi = rand(1) * .4 + .3;
2 mu = rand(1,2);
3 sigma = rand(2,2);
4 sigma = sigma'*sigma;
```

Line 1 ensures reasonable bounds for ϕ. Line 4 indicates that we multiply sigma with its transpose, guaranteeing it to be a symmetric positive semi-definite matrix [2].

Error in experiments 3.1 through 3.3 is defined as the percentage of observation assignments that are off:

$$100\% \cdot \frac{\sum_{i=1}^{K} |z_e^{(i)} - z_c^{(i)}|}{K}$$

where K is the sample size, z_e is the expected assignment, and z_c is the calculated assignment.

3.1 Sample Size

One of the most important aspects of expectation maximization is optimizing the number of samples provided for the algorithm. Figure 3.3 shows the results of this experiment. The data show that as sample size increases past 100 there is negligible benefit to accuracy.

	Set 1	Set 2
ϕ	.3864	.6136
μ	$(3.3945, 3.1589)$	$(.3105, 0.6563)$
σ	$\begin{pmatrix} 0.7724 & 0.1333 \\ 0.1333 & 0.4290 \end{pmatrix}$	$\begin{pmatrix} 0.3624 & 0.1956 \\ 0.1956 & 0.7413 \end{pmatrix}$

Figure 3.1: The base parameters used for this set of experiments.

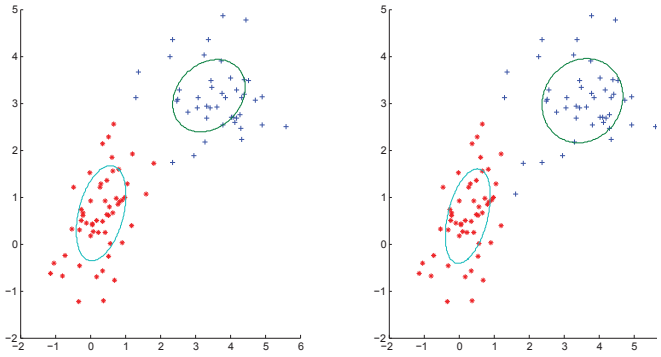

Figure 3.2: A sample set drawn from the parameters given in Figure 3.1. The left side shows the data generated with the given parameters, and the right side shows the assignments generated by the algorithm. Both graphs make use of the ellipse function found in [1].

3.2 Convergence Tests

There are two primary ways of testing the convergence of this algorithm. The first and most natural is testing convergence of the w values calculated in the E-step. But because these values (along with calculated ϕ, μ, and σ) are continuous, we're forced to choose an arbitrary epsilon to avoid iterating indefinitely.

Another possibility is by measuring convergence of the z values, which are necessarily discrete. Figure 3.4 shows another series of trials run on the same parameters as those used in Figure 3.3. In each w convergence test we choose $\epsilon = .001$.

The data show that neither convergence test has a legitimate advantage over the other in terms of accuracy, as both tend toward about 1.5% error given a large enough sample size. For smaller sample sizes there is a significant difference in iteration time, however. Error levels off around 100 samples, and at this point it takes approximately three times as many iterations to complete the algorithm using parameter convergence.

3.3 Horizontal Bias

As described in Section 2, there is a possible bias that would limit this implementation's ability to differentiate between sets if each mean's x-coordinate is the same. This is dispelled when we examine Figure 3.5. The data show that even in the case of identical x-coordinates, the algorithm is able to successfully partition the sets with the same degree of accuracy, although this takes more iterations. It is notable that in the case of small sample sizes, more iterations are necessary using epsilon convergence. This difference is only significant under sample sizes of 50, which is shown to be an inadequate amount in the other experiments as well.

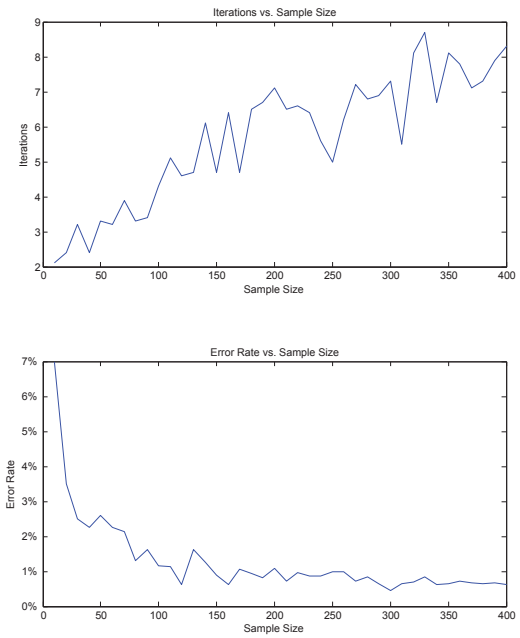

Figure 3.3: Error rates and iterations when compared to sample size.

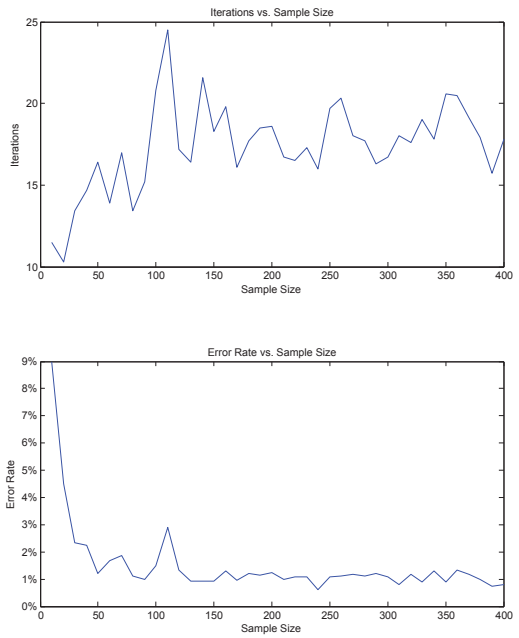

Figure 3.4: Error rates and iterations when compared to sample size, using epsilon convergence.

3.4 Parameter Accuracy and Mean Distance

Parameter accuracy is another method of determining the overall efficacy of the EM algorithm. These trials used the ϕ and Σ values identified in Figure 3.1, with μ_1 set to $(0, 0)$ and μ_2 moved along the positive x-axis to achieve the mean distances denoted in the chart. Error values are taken using $|p_e - p_c|$ where p is the parameter being tested. In the case of μ and Σ, the `norm` function is used. The data show that, as we would expect, error rates decrease significantly for all parameters as the sets are pulled apart.

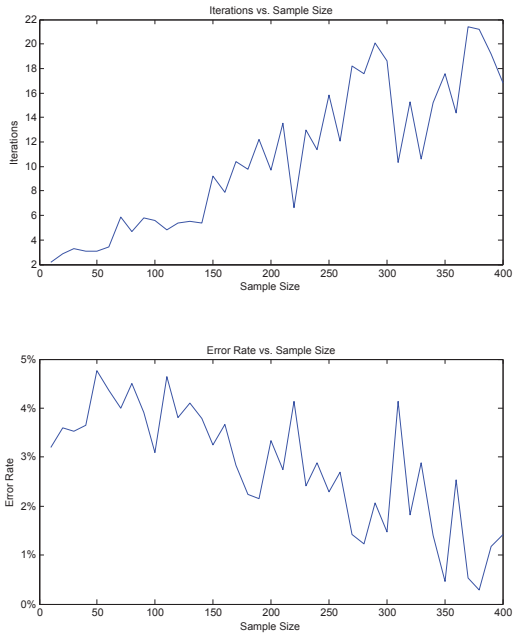

Figure 3.5: Error rates and iterations when compared to sample size. In these experiments, μ values are modified to align on the vertical axis.

4 Implementation Issues

One potential problem with the implementation is the reliance upon loops to accomplish iterative tasks. Although the project guidelines provided some details on vectorization, many calculations required iteration over multiple vectors of varying dimensionality simultaneously. This rendered even more flexible functions such as `arrayfun` inadequate to accomplish the tasks at hand. After researching the issue online, it was unclear that vectorization would necessarily lead to a performance boost, so this aspect of the project was ultimately abandoned.

Another small issue with the implementation is the lack of error checking at the boundaries of valid input. Small sample sizes, for example, can cause the EM function to halt due to invalid covariance matrices being generated.

Overall the implementation performed quickly and robustly enough to carry out a number of experiments successfully.

References

[1] `http://mathworks.com/matlabcentral/fileexchange/4705`.

[2] Positive-definite matrix. `http://en.wikipedia.org/wiki/Positive-definite_matrix`.

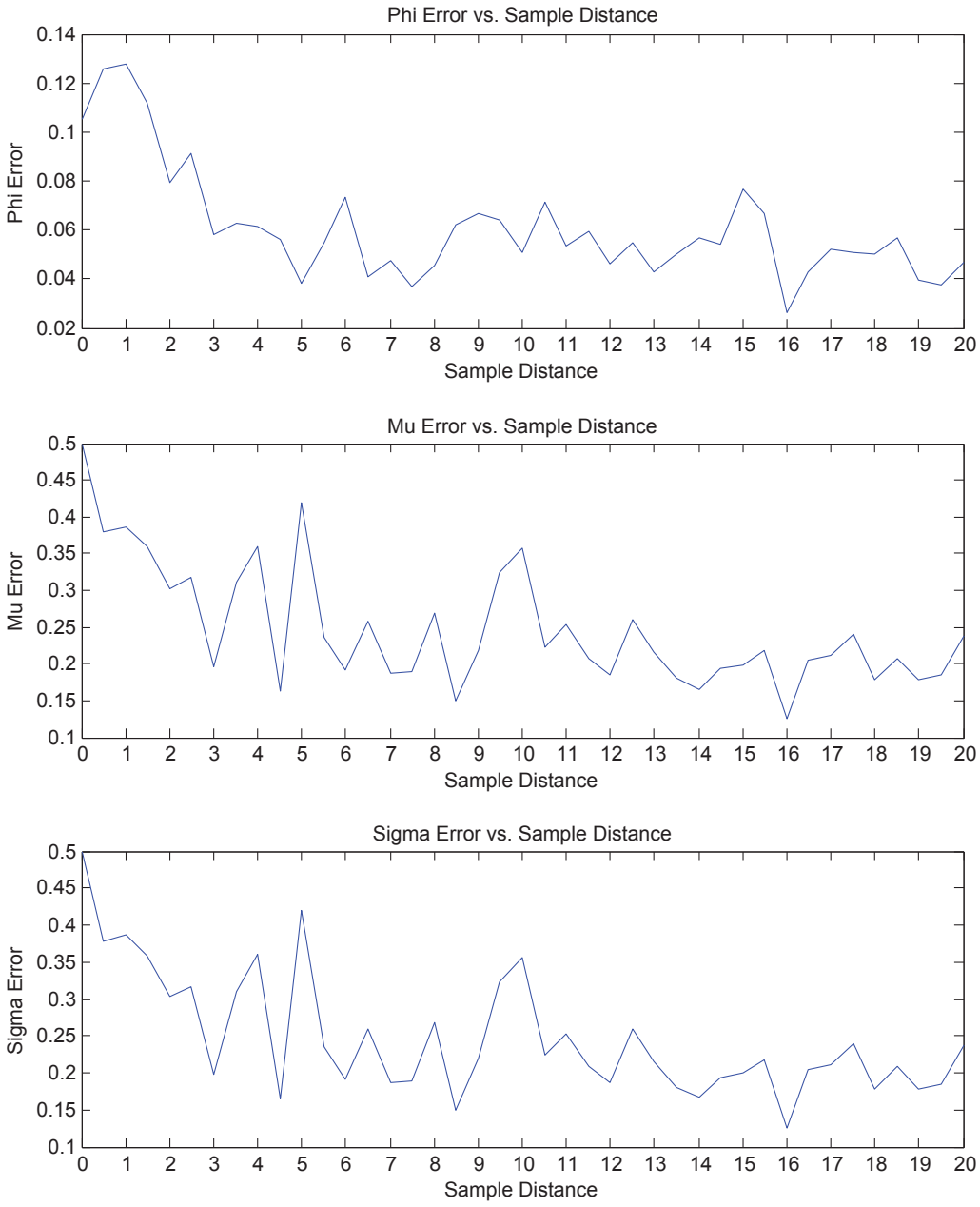

Figure 3.6: Error rates for ϕ, μ, and Σ versus the distance between generating μ_1 and μ_2 values.