Data generation

There are basically three priors that I used to generate my training data: π, μ, and Σ. π is a vector with as many items are there are clusters, and I am using only two clusters for most of this report. μ is two vectors, two to match the number of clusters, and the vectors are each as long as the number of dimensions of the data, which, for most of this report, is also two. Σ is two matrices, symmetric and positive definite.

While I could have used a simple draw from the uniform U(0, 1) for π, I wanted to handle additional clusters easily, so I drew from a standard Dirichlet distribution (parameters of all ones). Matlab has no Dirichlet function that I could find, so I simply used Gammas draws and normalized.

I drew each μ from the standard normal distribution ($\mathcal{N}(0, 1)$). The Σ was also generated by the standard normal, and then multiplied by its transpose to ensure that it was symmetric and positive definite.

The number of samples I drew varied. Below fifty, my results started to become much less predictable. For most of the report, I am using 1000 samples. I visually evaluated the variety of samples I was getting by using a simple scatter plot, and the degree to which the clusters overlapped or not, on different runs with different draws of priors and data, seemed to be suitable.

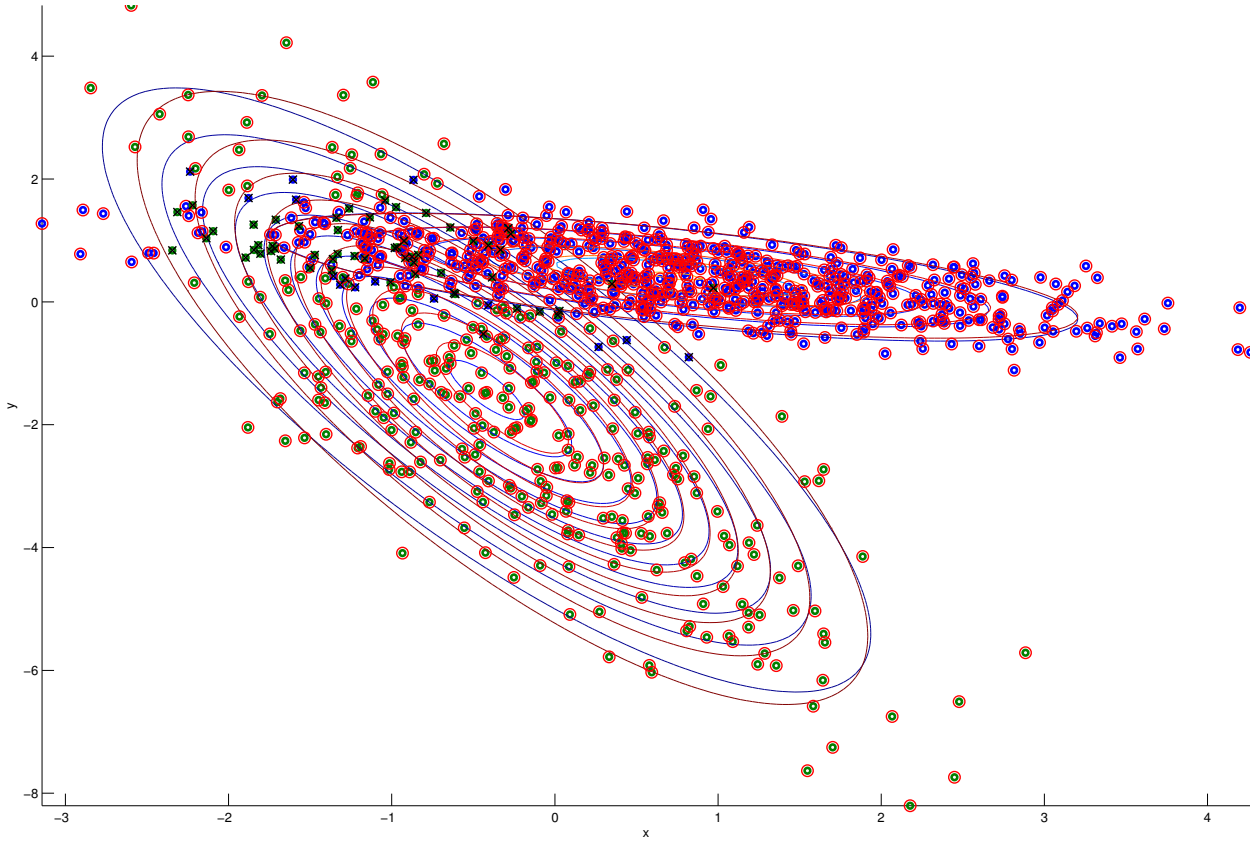

Figure 1: Two clusters, two dimensions, 1000 samples. Circles of red indicate correct assignment after last iteration, while a black 'x' indicates incorrect assignment. Data from the cluster 1 is colored blue, and points from cluster 2 are colored green. The contours reflect the parameters; blue contours are the parameters used to generate the data, red contours are the result of the EM algorithm.

Expectation Maximization

I used the Matlab `kmeans` function to initialize cluster parameters (I evaluated the mean and covariance of each cluster k-means picked out). However, I also found that purely random starts (using the same generation parameters used to generate the data) did not differ much from k-means, except that the log-likelihood would not increase quite so steeply in the first few iterations.

The expectation step was straightforward with `mvnpdf`. However, my Σ parameter would sometimes drop to all zeros, which would throw an error in `mvnpdf`, in which case I restarted Σ for that cluster.

I was able to optimize the estimation of the new μ parameter for each cluster in the Maximization step, using linear algebra instead of a for-loop, but I couldn't figure out how to optimize the Σ estimation step using linear algebra.

I defined my convergence criterion as an increase in log likelihood of less than 0.001, which seems kind of arbitrary, but works well enough (I tried several values). I still had a hard limit of a 100 iterations, for cases of extreme overlap between clusters.

Because of label switching, I chose the assignment of labels to clusters that maximized the number of correct assignments. This seems almost like cheating, since I'm looking at the original labels before evaluation, but since the clustering is unsupervised, I believe it's a valid step.

I am not sure exactly what was meant by "plot the accuracy of means and other parameters found by your algorithm compared over several trials," but the chart below (Figure 2) plots the fraction correctly clustered, and then the sum of the pairwise euclidean difference between guessed parameters and the priors (ground truths), for various initializations of the σ prior.

$\sigma = 0.01$ $\sigma = 100$

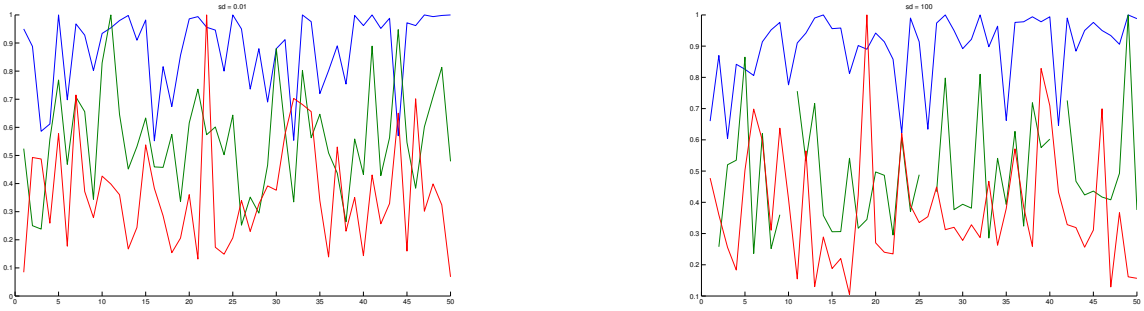

Figure 2: The fraction correctly clustered is blue. μ is green, Σ is red, and are both reweighted to fit within the plot (since their units are arbitary already).

It's pretty easy to scale up the coded algorithm to more than two clusters, but it doesn't support more than 2 dimensions at the moment. The plots would be particularly tricky, but I don't think it would be too much more work to extend it past two dimensions. With more than two clusters, it simply requires more iterations, but it still gives pretty good results (see Figure 3 on the next page). The computational-intensive part is evaluating which cluster goes to which label with more than a few clusters.

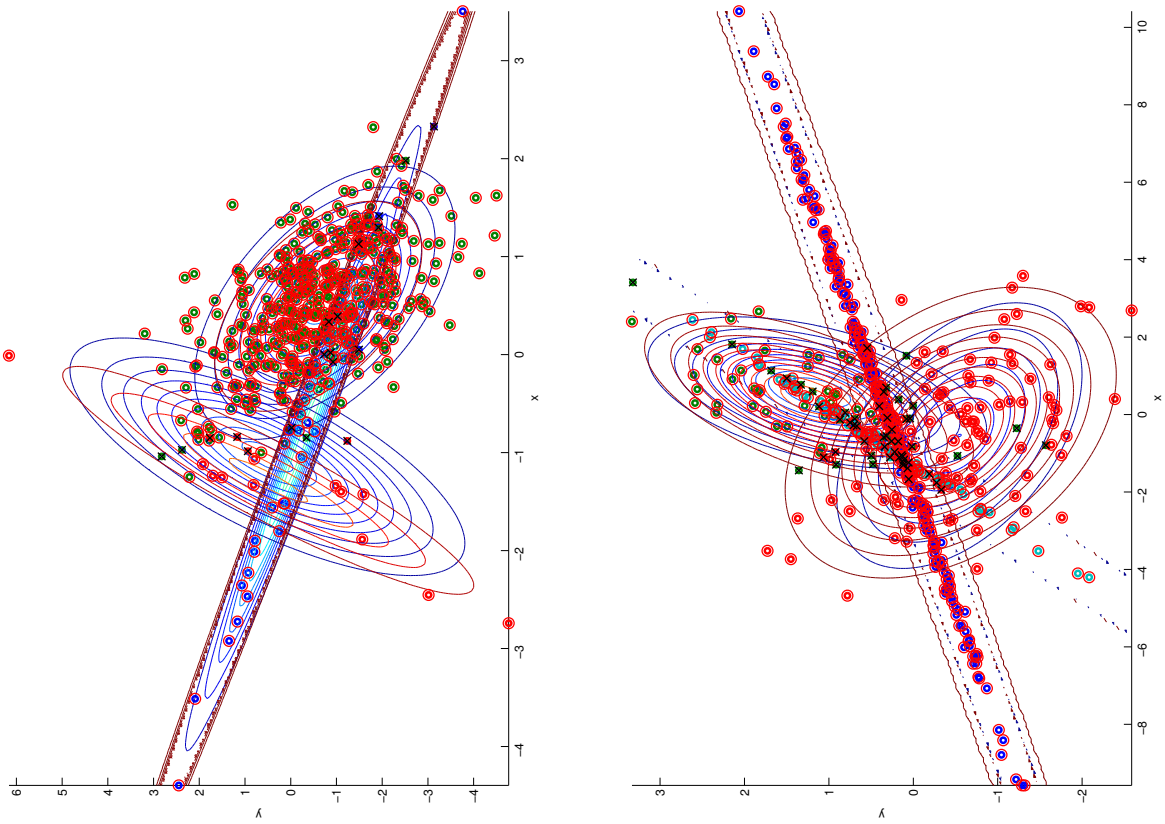

Figure 3: Three, four, and seven clusters.