Question 1

In order to solve for the parameters and derive the EM algorithm, we must first write down the complete log likelihood:

$$\sum_{t=1}^{N}\sum_{i=1}^{k} z_i^t(\log\pi_i + \log m! - \sum_{r=1}^{n}\log x_r^t! + \sum_{r=1}^{n} x_r^t \log p_{ir})$$

Note that r is just another value used to iterate over the different Xs. Since the complete log likelihood contains z terms, and we don't know z, we need to work with the expectation of the complete log likelihood. This is given by:

$$\sum_{t=1}^{N}\sum_{i=1}^{k} \gamma(z_i^t)(\log\pi_i + \log m! - \sum_{r=1}^{n}\log x_r^t! + \sum_{r=1}^{n} x_r^t \log p_{ir})$$

Note that the z terms become gammas which are functions of the z terms; the reason for this is given in the textbook and was explained in class. Before any derivatives are taken, LaGrange constraints must be added:

$$\sum_{t=1}^{N}\sum_{i=1}^{k}[\gamma(z_i^t)\left(\log\pi_i + \log m! - \sum_{r=1}^{n}\log x_r^t! + \sum_{r=1}^{n} x_r^t \log p_{ir}\right) - \alpha_i(\sum_{r=1}^{n} p_{ir} - 1)] - \beta(\sum_{i=1}^{k}\pi_i - 1)$$

Note that the alpha terms are inside the double summation, whereas the beta term is outside both summations. We can now begin solving for the parameters. Let us take the derivative of this expression with respect to a specific p_ij (we'll let i = a and j = b, so that we don't get confused by notation). Then:

$$\frac{dE(L_c)}{dp_{ab}} = \sum_{t=1}^{N}\frac{\gamma(z_a^t)x_b^t}{p_{ab}} - \alpha_i = 0$$

And solving for p gives us:

$$p_{ab} = \frac{\sum_{t=1}^{N}\gamma(z_a^t)x_b^t}{N\alpha_a}$$

So for an abitrary p_ij:

$$p_{ij} = \frac{\sum_{t=1}^{N}\gamma(z_i^t)x_j^t}{N\alpha_i}$$

We know that summing p_i over the js yields 1, since this is one of the constraints. Then summing the LHS and the RHS over the js produces:

$$\frac{m\sum_{t=1}^{N}\gamma(z_i^t)}{N\alpha_i} = 1$$

Solving for alpha yields:

$$\alpha_i = \frac{m}{N}\sum_{t=1}^{N}\gamma\left(z_i^t\right)$$

And finally, plugging this back in to the p_ij equation gives:

$$p_{ij} = \frac{\sum_{t=1}^{N}\gamma\left(z_i^t\right)x_j^t}{m\sum_{t=1}^{N}\gamma\left(z_i^t\right)}$$

Now we aim to solve for the pi terms. To do this, we take the derivative with respect to a particular pi_i (let i = a):

$$\frac{dE(L_c)}{d\pi_a} = \sum_{t=1}^{N}\frac{\gamma(z_a^t)}{\pi_a} - \beta = 0$$

Solving for pi gives:

$$\pi_a = \frac{\sum_{t=1}^{N}\gamma(z_a^t)}{\beta}$$

So for an abitary pi_i:

$$\pi_i = \frac{\sum_{t=1}^{N}\gamma\left(z_i^t\right)}{\beta}$$

But since we know the pi_i add up to 1 since it’s one of our constrains, we can sum the RHS and LHS over the is to get:

$$\sum_{i=1}^{k}\frac{\sum_{t=1}^{N}\gamma\left(z_i^t\right)}{\beta} = 1$$

And solving for beta gives:

$$\sum_{i=1}^{k}\sum_{t=1}^{N}\gamma\left(z_i^t\right) = \beta$$

Finally, plugging that back into the pi_i equation yields:

$$\pi_i = \frac{\sum_{t=1}^{N}\gamma\left(z_i^t\right)}{\sum_{i=1}^{k}\sum_{t=1}^{N}\gamma\left(z_i^t\right)}$$

Lastly, let’s find gamma(z_i^t). We know that:

$$\gamma\left(z_i^t\right) = \frac{\pi_i P(x^t|\Phi_i^l)}{\sum_j^k P(x^t|\Phi_j^t)}$$

Which in our case can be written as:

$$\gamma(z_i^t) = \frac{\pi_\mathrm{i} \frac{m!}{x_1!\,x_2!\dots x_n!} P_{i1}^{x_1} P_{i2}^{x_2} \dots P_{in}^{x_n}}{\sum_j^k \pi_\mathrm{j} \frac{m!}{x_1!\,x_2!\dots x_n!} P_{j1}^{x_1} P_{j2}^{x_2} \dots P_{jn}^{x_n}}$$

But the m! term is a constant, so this can be simplified to:

$$\gamma(z_i^t) = \frac{\pi_\mathrm{i} P_{i1}^{x_1} P_{i2}^{x_2} \dots P_{in}^{x_n}}{\sum_j^k \pi_\mathrm{j} P_{j1}^{x_1} P_{j2}^{x_2} \dots P_{jn}^{x_n}}$$

Thus, we've derived the desired results.

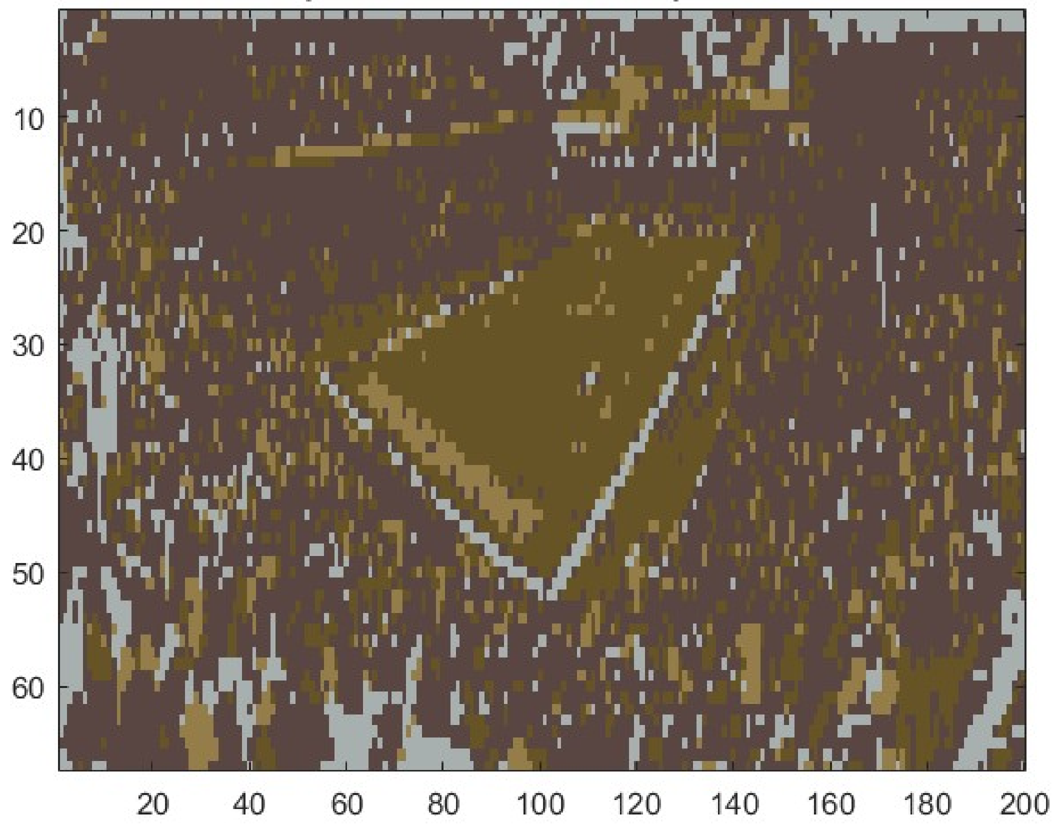
Question 2

NOTE: For all these questions, I did not include the values printed to the MATLAB terminal, because they're too long (and not so meaningful).

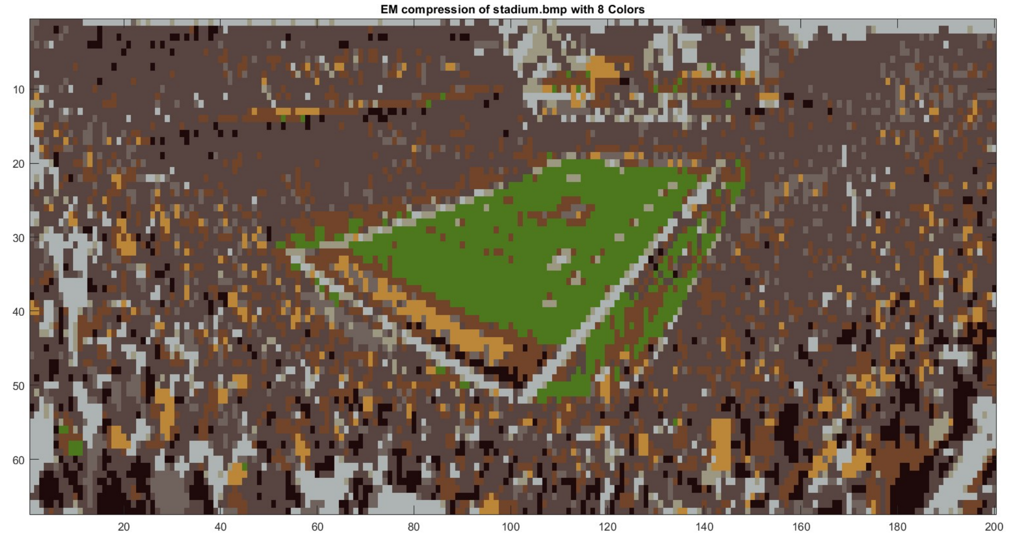
a)

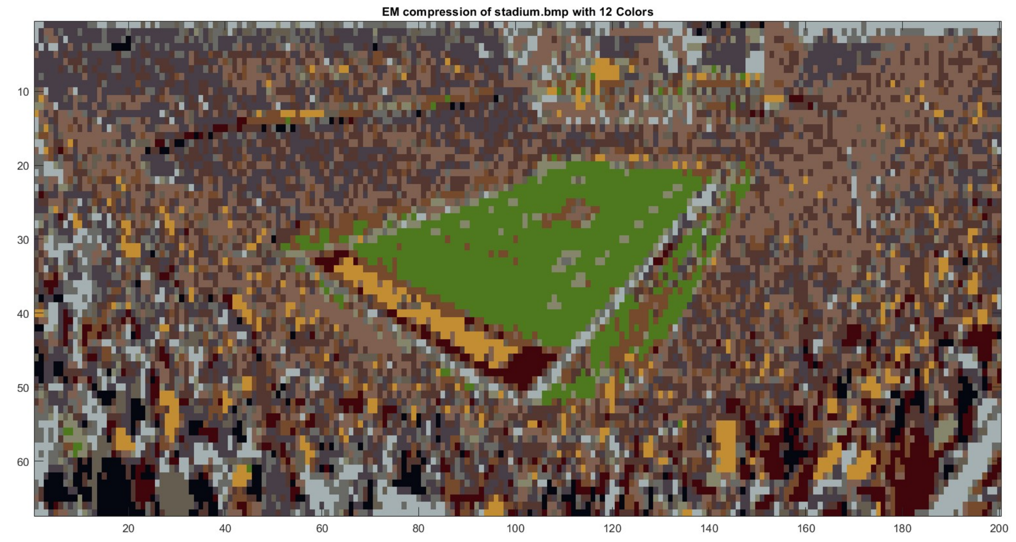
Below are 3 different compressions of stadium.bmp, each one running EM with a different number of colors as specified by the instructions. Note that the resolutions of the image files associated with each plot are slightly different, so they appear to have different aspect ratios. However, each plot has the same number of dimensions.

EM compression of stadium.bmp with 4 Colors



EM compression of stadium.bmp with 8 Colors

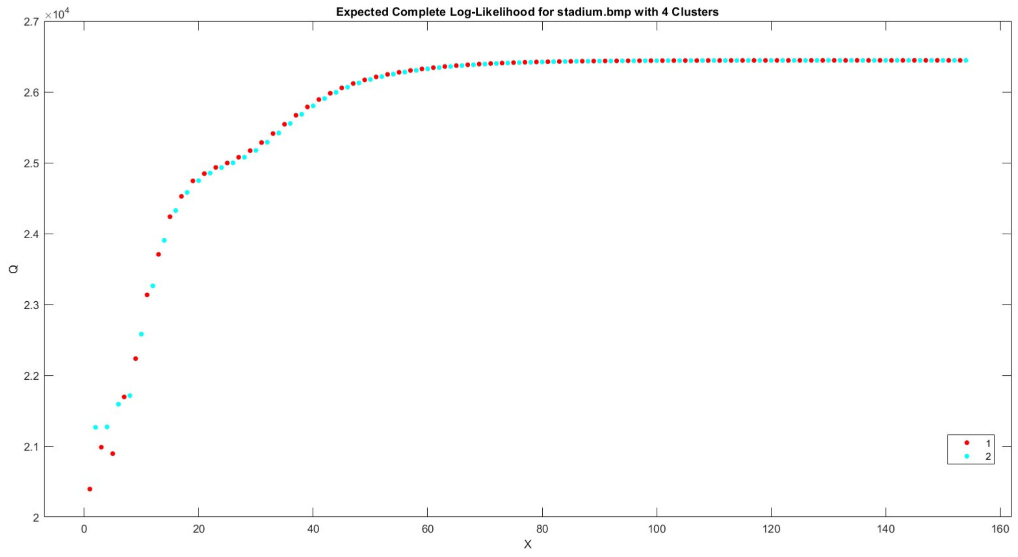


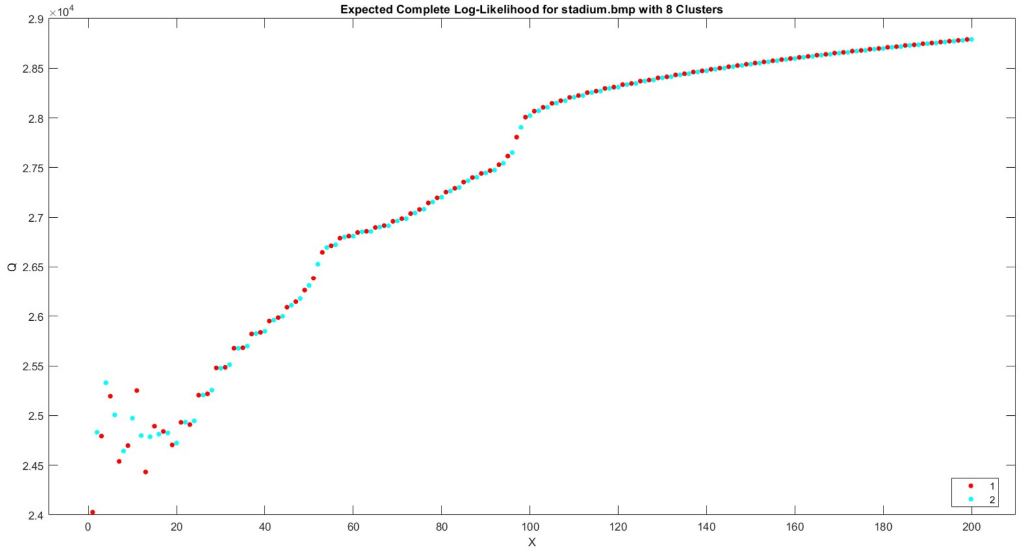


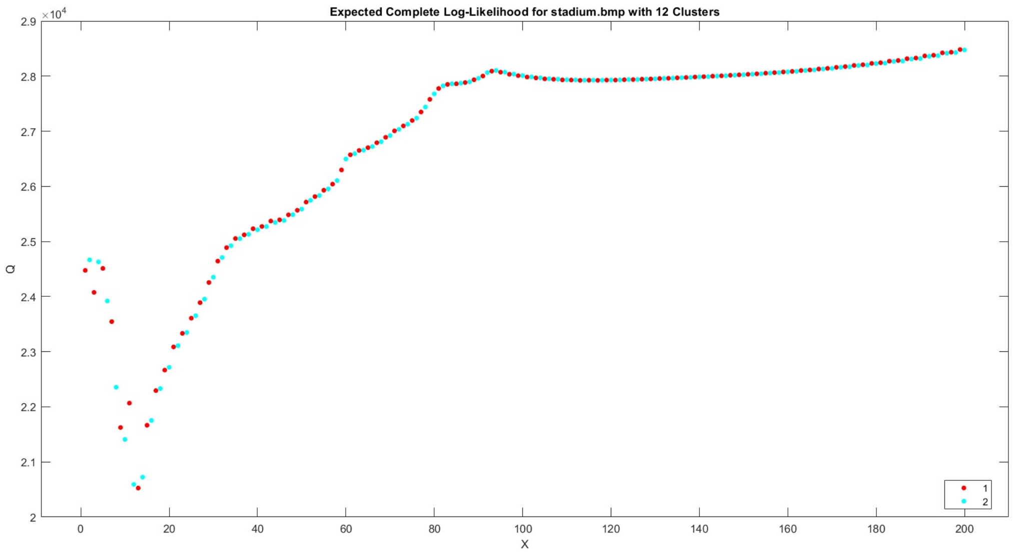
The resulting images appear to be pretty good, however, my algorithm tops out at 100 iterations to avoid super long computation times. It is possible that running the EM algorithm for more iterations would produce better results, particularly in the 12 color image.

b)

Below are images of the plots of the complete log-likelihoods associated with 4, 8, and 12 clusters. The E step is plotted in red and the M step is plotted in blue. It was not clear to me whether there should be two separate lines, or one combined plot (like in my plots). The way I've chosen to do it, the x-axis represents the index in the Q vector, meaning that to find the number of iterations the algorithm ran for, the x axis must be divided by 2. Also, my algorithm is set to stop early if the complete log-likelihood reaches some level of convergence. This is to save computation time; in truth, the convergence condition could probably be relaxed to get even faster calculations, and not much quality would be lost. This is why in the 4 cluster plot, there are not 200 indices in the Q vector.


Expected Complete Log-Likelihood for stadium.bmp with 4 Clusters
×10^4
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2
Q
0
20
40
60
80
100
120
140
160
X
1
2

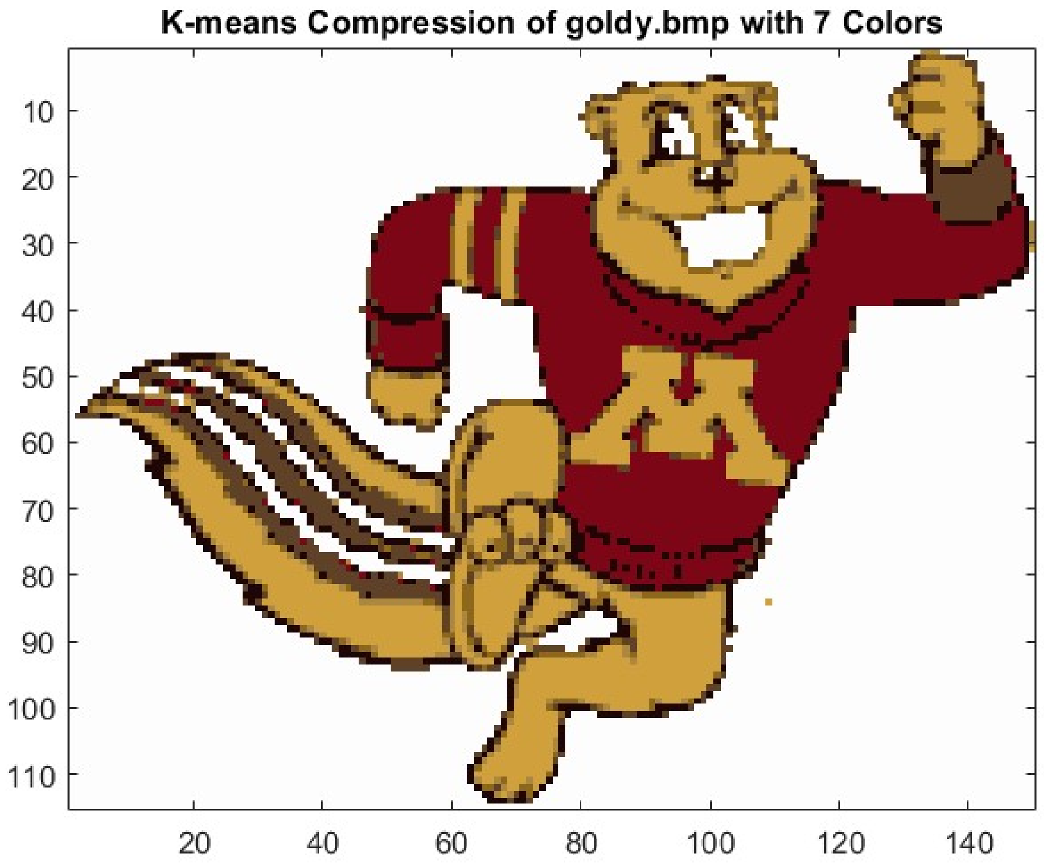

Expected Complete Log-Likelihood for stadium.bmp with 8 Clusters
×10^4
2.9
2.85
2.8
2.75
2.7
2.65
2.6
2.55
2.5
2.45
2.4
Q
0
20
40
60
80
100
120
140
160
180
200
X
1
2



We can see lots of volatility at the beginning, and then a steady convergence to some value. What is somewhat surprising to me is the initial volatility, since my algorithm picks its initial clusters using k-means. I would assume there wouldn't be too much shifting after k-means, but I guess that is not the case.

c)

For this part, my algorithm failed to produce a result even after several attempts. Each time, the covariance matrices for the clusters end up being singular, causing mvnpdf (the normal distribution function) to fail. K-means doesn't have to worry about any matrix singularities (since it assumes all distributions are circular, not Gaussian), so it is able to produce results without fail. Below is the K-means plot of goldy.bmp with 7 colors:



These results actually look really good, with the compressed goldy image being virtually indistinguishable from the original. The performance of k-means on this algorithm is of course somewhat expected, since the image is of a mascot, which by its nature will not have many colors.

d)

We add the regularization term to the likelihood, and since we are told what the derivative of the regularization term with respect to sigma inverse is, we can plug that in to the derivation from the slides:

$$\frac{N}{2}\Sigma - \frac{1}{2}\sum_{t=1}^{N}(x^t - \mu)(x^t - \mu)^T - \frac{\lambda I}{2} = 0$$

Solving for sigma yields:

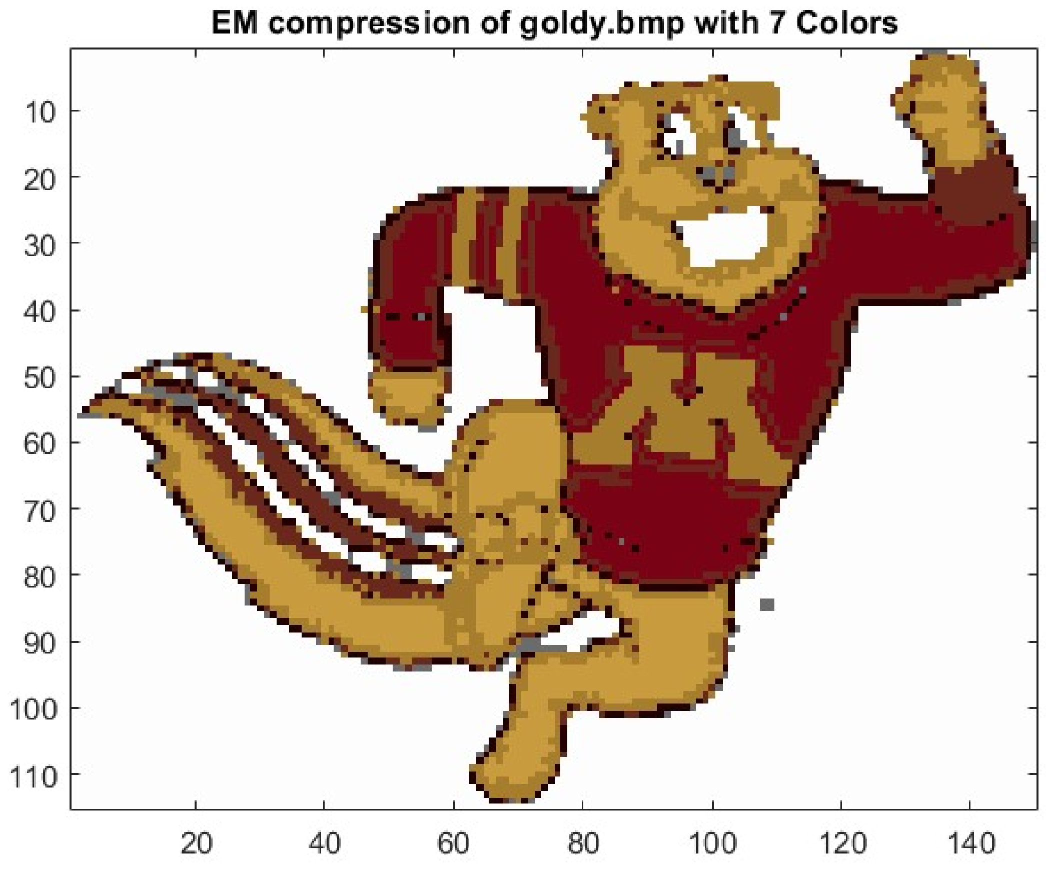
$$\Sigma = \frac{\sum_{t=1}^{N}(x^t - \mu)(x^t - \mu)^T + \lambda I}{N}$$

And so, we see that the covariance should remain unchanged, except that a value should be added to the diagonal. We do not know exactly what lambda to choose, but we know it shouldn't be too large or it will affect the EM algorithm.

In practice, this just means choosing some small value and adding it along the diagonals of the covariance matrices. In my case, I chose 0.00000001 to add to the diagonal, and this seemed to stop singular matrices from popping up, while also retaining decent results.

e)

Below is the compressed goldy.bmp image with 7 colors, except this time done by improved EM. We can see that the results are not as good as k-mean. The reason for this might be two-fold, with the first one being that the algorithm only runs for 100 iterations; perhaps running the algorithm for more iterations would produce better results. However, the main reason this performs worse than k-means is probably because the colors aren't distributed in a way where normal distributions would be an advantage over circular distributions. Since EM is a more complicated algorithm than K-means, it'll perform worse when its ability to capture normal distributions isn't useful. With only a few colors in the entire image, and with them being distributed roughly equally (since they represent the university's colors), this isn't so surprising.


EM compression of goldy.bmp with 7 Colors