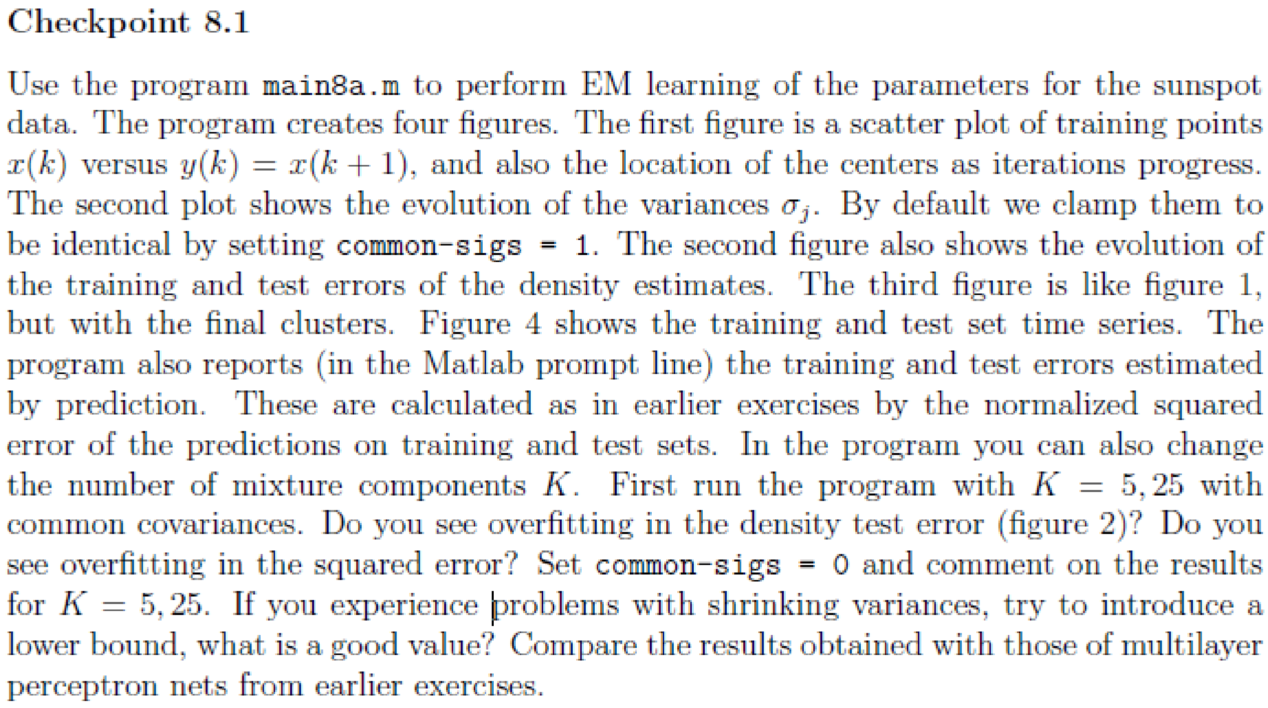


Checkpoint 8.1

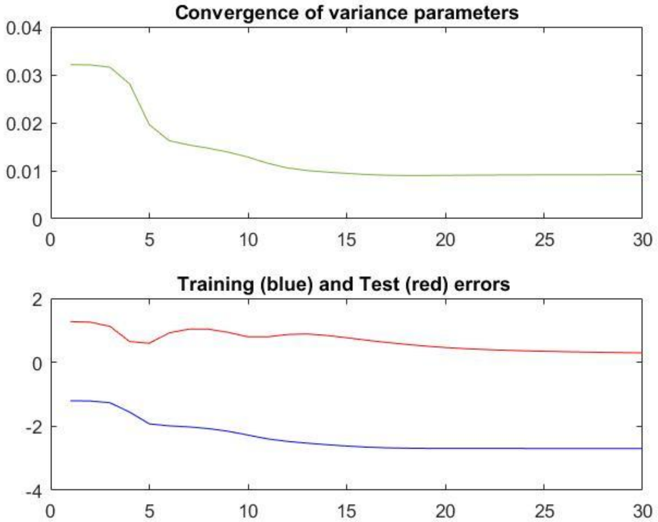
Use the program `main8a.m` to perform EM learning of the parameters for the sunspot data. The program creates four figures. The first figure is a scatter plot of training points $x(k)$ versus $y(k) = x(k+1)$, and also the location of the centers as iterations progress. The second plot shows the evolution of the variances σ_j. By default we clamp them to be identical by setting `common-sigs = 1`. The second figure also shows the evolution of the training and test errors of the density estimates. The third figure is like figure 1, but with the final clusters. Figure 4 shows the training and test set time series. The program also reports (in the Matlab prompt line) the training and test errors estimated by prediction. These are calculated as in earlier exercises by the normalized squared error of the predictions on training and test sets. In the program you can also change the number of mixture components K. First run the program with $K = 5, 25$ with common covariances. Do you see overfitting in the density test error (figure 2)? Do you see overfitting in the squared error? Set `common-sigs = 0` and comment on the results for $K = 5, 25$. If you experience problems with shrinking variances, try to introduce a lower bound, what is a good value? Compare the results obtained with those of multilayer perceptron nets from earlier exercises.

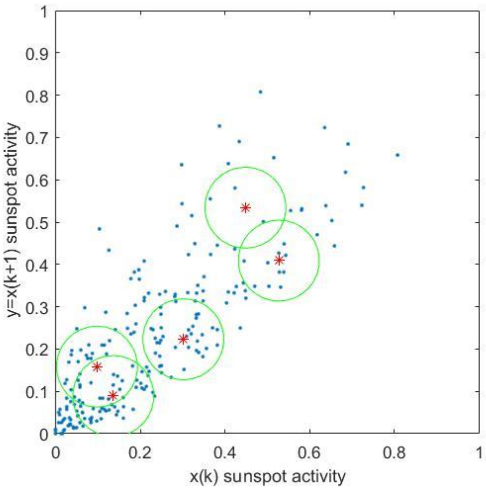
In this exercise we are given the sunspot dataset that was already used in the previous exercises. The problem presented in this exercise is a regression problem so we have measurements from 4 previous years and we try to predict the next year's number of sunspots.

The first thoughts after running the script where that this model does not match very well with this data, and that there are other options which would provide better results. This is due to the fact that the data is not really clustered in gaussians, which makes it difficult to model with them. However, as we will see later, even though the density errors are pretty bad, it can achieve pretty good results in terms of squared error.

Second thing to think about is the dimensionality of the problem. We are working in a 4 dimensional space, one for each past year that we are taking into consideration in out model. That means that even though the results we see seem to be only as a function of the previous year, the results have been casted into that dimension.

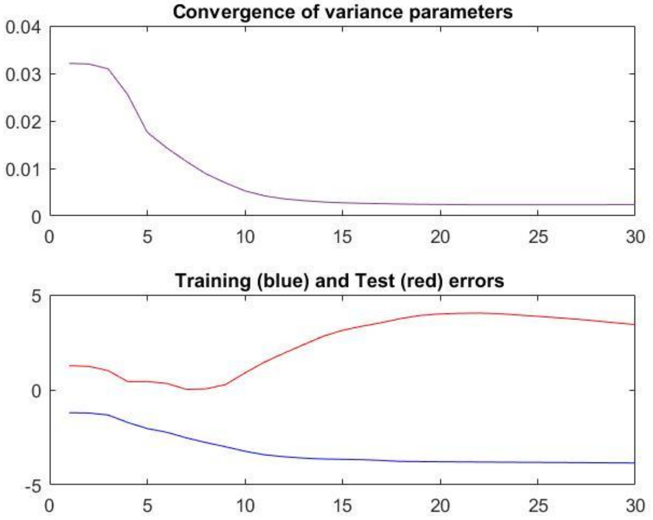
After this considerations, we will proceed with the experiments mention in the exercise. Let's start by plotting the case when K=5 and the variances are fixed.

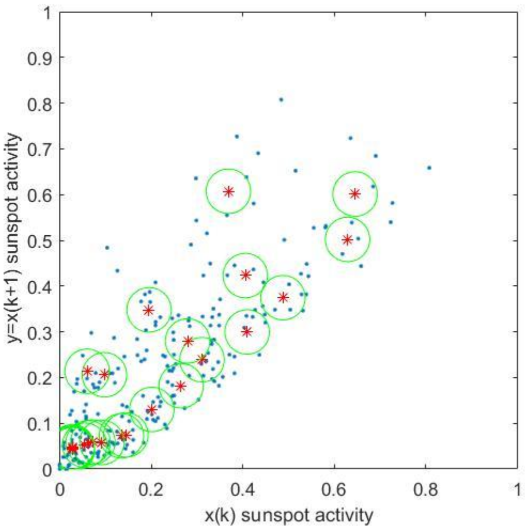




As we can note, there is no overfitting in the density error, partially because the variance converges to a pretty big value, as it is selected by taking the mean of all the variances and the data is pretty sparse on the right side of the plot. .

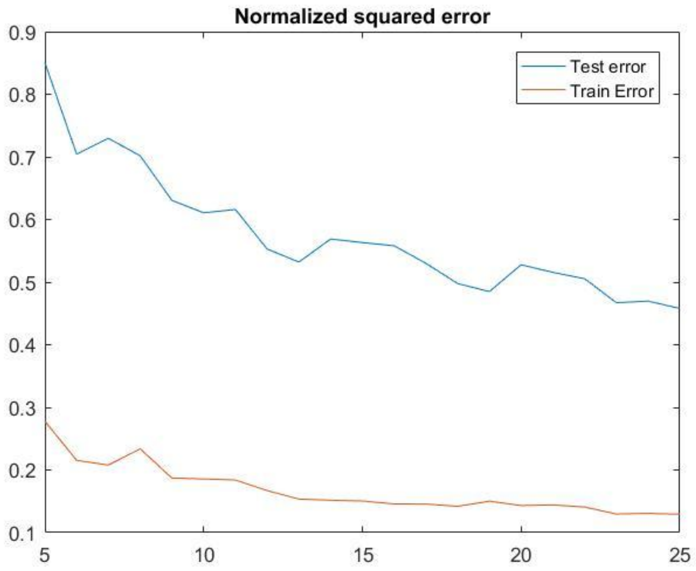
However, if we increase the number of clusters to K=25, we start to see some overfitting, as presented in the following plots:





In this case the variance converges to a smaller value, causing that some of points in the test set are not correctly modeled, increasing the test error. This model becomes really inflexible this way, due to overfitting in the train data.

Nevertheless, when we take a look at the squared error, we cannot see any kind of overfitting.

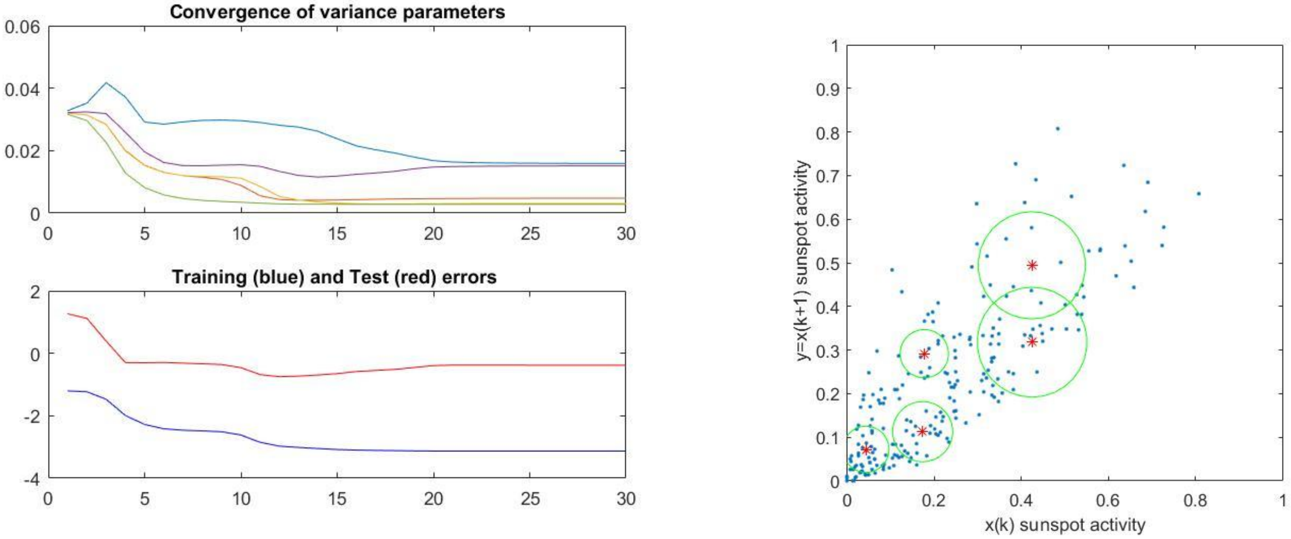


For the range between K=[5,25] the squared error keeps going down when increasing the number of clusters. This is explained by the fact that the cost function that we are trying to minimize:

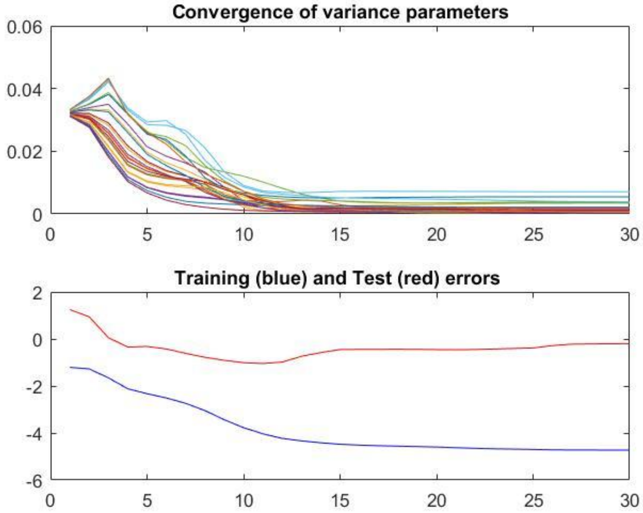
$$E = -\sum_{n=1}^{N} \log p(\mathbf{x}_n, t_n|\mathbf{w}) \ .$$

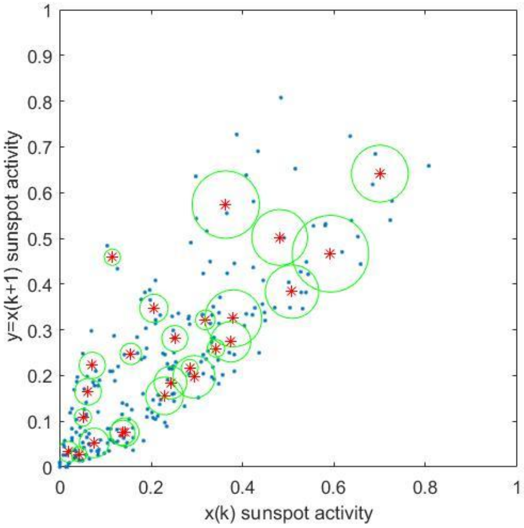
Is not afected by the normalized squared error. Moreover, if we take a look at how does the output of the model is calculated, it makes sense that the model still performs correctly based on this error even when based on the density error the model is overfitted.

Now let's take a look when the variances are not fixed to be the same for all of the clusters.



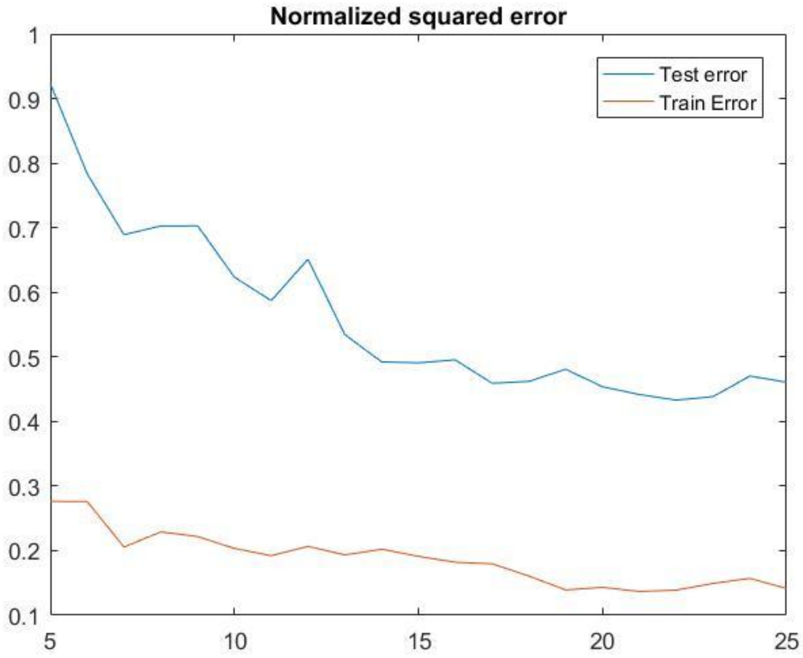
As we expected, the variances in the right side of the plot became large as the data is more spread. This change allows the clusters to fit the data better, causing the density error to decrease.





Now, when we introduce 20 more clusters we can see that the data starts to overfit, but not at the same extent as in the fixed variance. The reason behind it is that by forcing a certain variance, some parts of the data cannot be modeled correctly using that variance, causing that the EM algorithm will optimize other parts of the data where, in this case, the data is more compact.

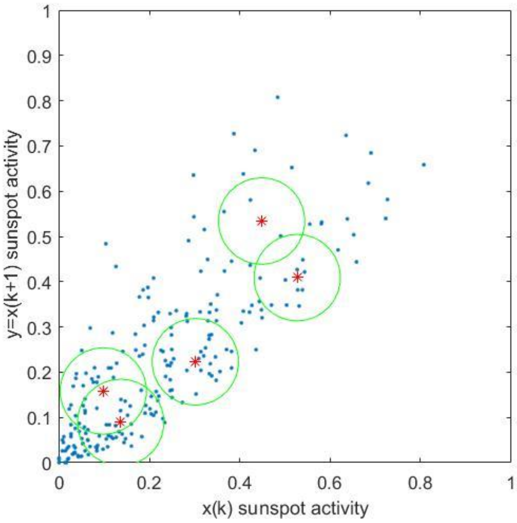
Finally, let's take a look at the normalized squared error:

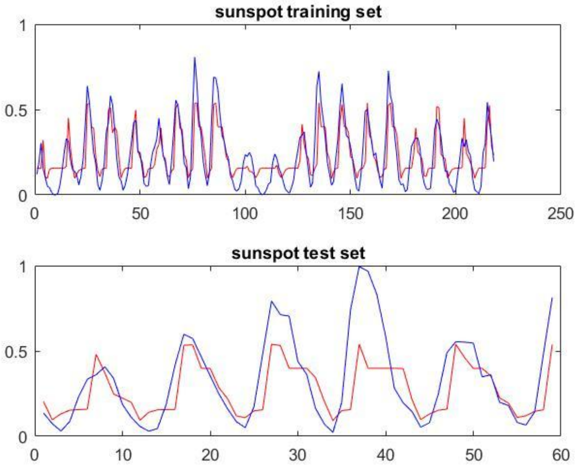


This plot shows a really similar behaviour to the one with the fixed variances. The main difference is that the square error reduces faster with the unfixed variances, even though the difference is really small.

Another important observation to make about this model is that due to how the output is calculated it is not possible to predict a value above the larger mean and below the smaller mean. We can see this by taking a look at the function to calculate the output of our model y(x) and the result from the first part of this checkpoint:

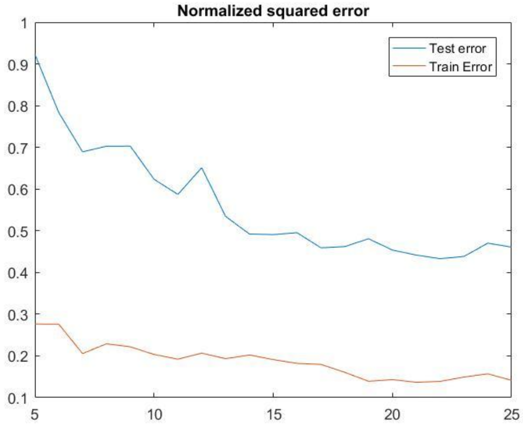
$$y(\mathbf{x}) = \sum_{j=1}^{M} v_j \, P(j|\mathbf{x}) \; .$$

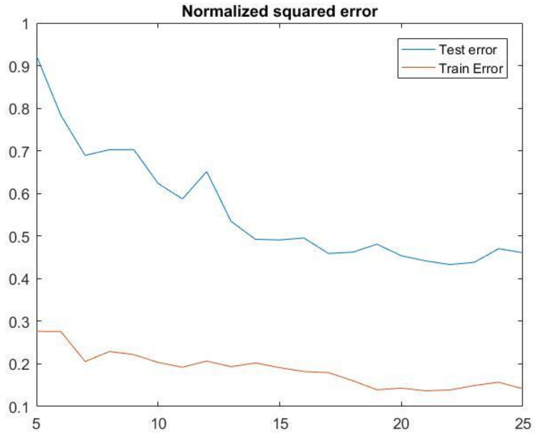


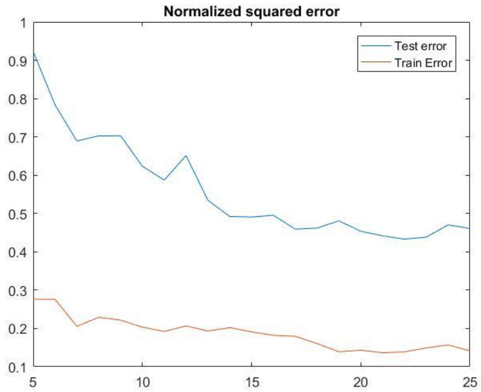


From the equation of the output we can see that it only uses two different variables from each cluster, On the one hand, v_j which is the mean of the cluster in othe output space, and on the other hand P(j|x), which is the responsibility of that cluster. Thus, we can note from the results that for example, for the upper bound, the maximum mean has a value of around 0.55, and if we take a look at the predictions they seem to be cropped in that value as well, which increases the squared error.

Regarding the threshold of the variance, the intuition says that we should choose a value as close to 0 as possible, just in order to avoid division by zeros. We have experimented to probe this hipotesis by repeating this experiment with the threshold value set to e-6, 1e-9 and 1e-15 respectively:

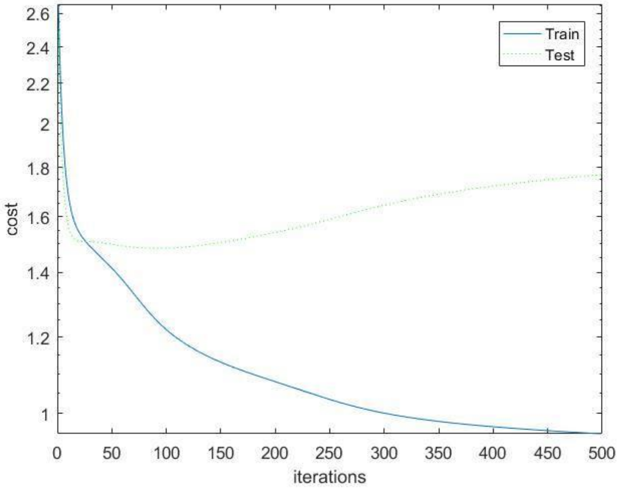


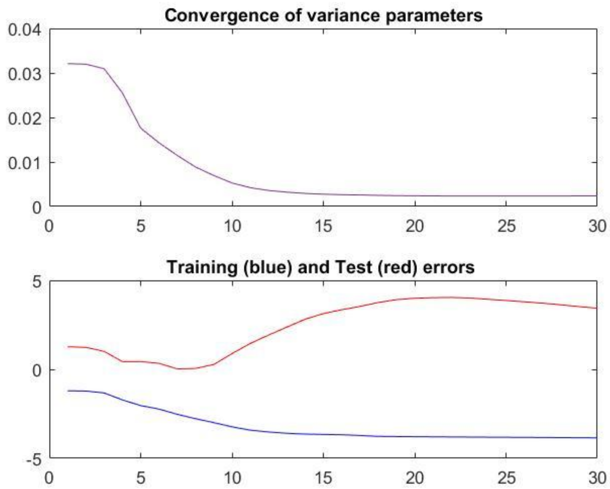


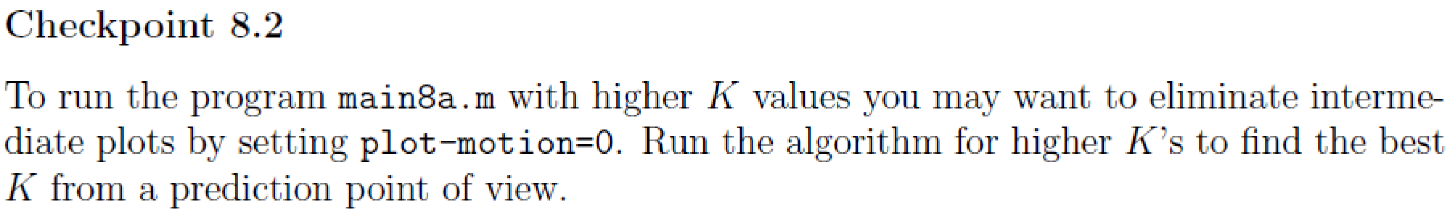


We can see that the differences are almost non-existinging.

Finally, let's compare the results obtained with the ones that we got from the NN exercise. It is important to realize that the tasks on hand are different, as in the NN exercise we are tackling a classification task and in this case we are performing a regression analysis. However, we find some general similarities, when it comes to fitting the data in the model. In the NN when we are using a large number of parameters it is easier to overfit the data. This is the case also in this exercise. It can be visualized by comparing the cost (error) function for a NN and the density function error in this case.



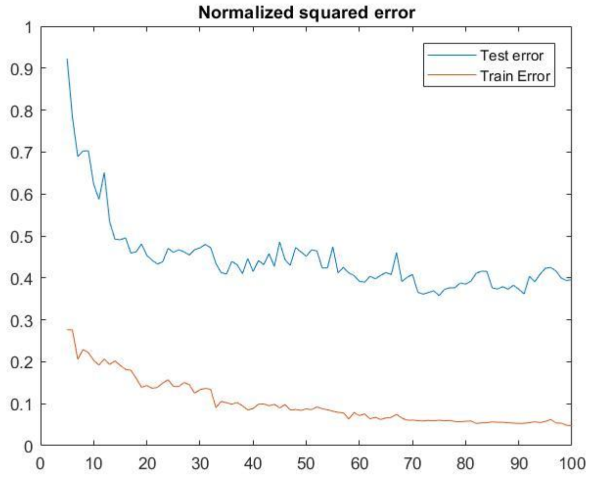


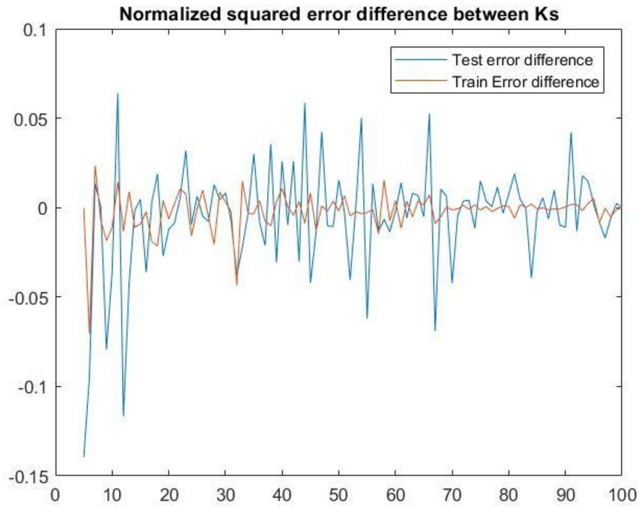


Checkpoint 8.2

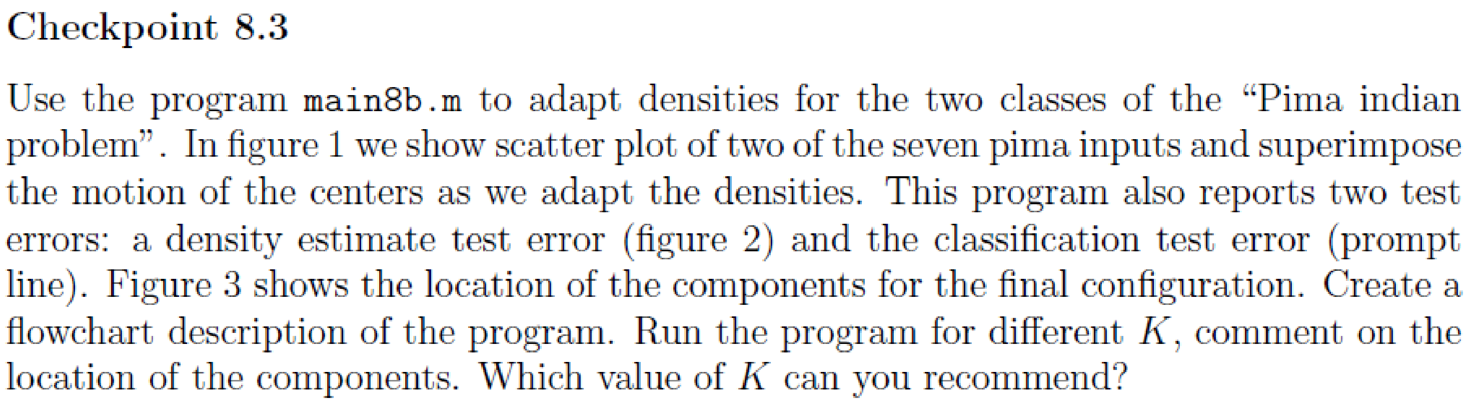
To run the program `main8a.m` with higher K values you may want to eliminate intermediate plots by setting `plot-motion=0`. Run the algorithm for higher K's to find the best K from a prediction point of view.

As we are evaluating our model from a prediction point of view, and we have data available to test it, we will use the squared error, as this is the one that will show the quality of the predictions. This is because, as we saw before, the training and test desinsity estimation error calculated by the EM algorithm does not always relate to the regression squared error.



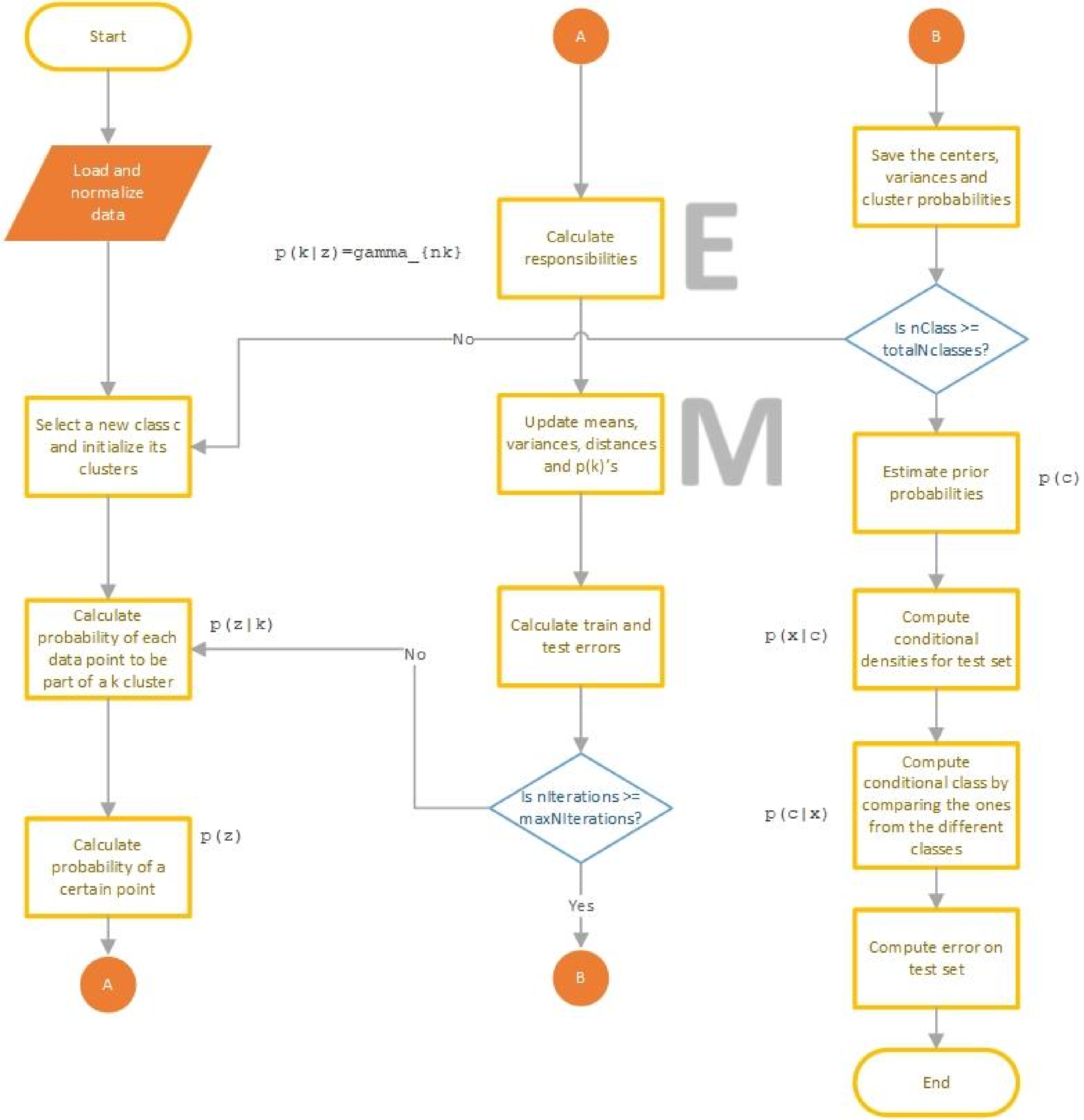


From the experiment, we can see that the value of the error does not change much after the number of K is equal to 15. Thus, it may not be worth it to include more Ks after that point, as the ratio of computations/improvement will be big.



Checkpoint 8.3

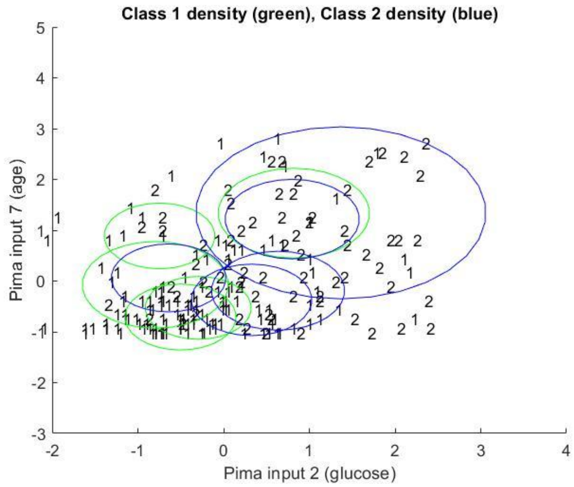
Use the program `main8b.m` to adapt densities for the two classes of the “Pima indian problem”. In figure 1 we show scatter plot of two of the seven pima inputs and superimpose the motion of the centers as we adapt the densities. This program also reports two test errors: a density estimate test error (figure 2) and the classification test error (prompt line). Figure 3 shows the location of the components for the final configuration. Create a flowchart description of the program. Run the program for different K, comment on the location of the components. Which value of K can you recommend?

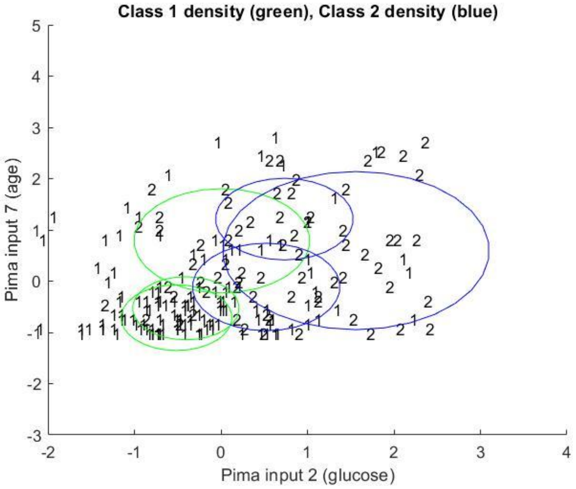


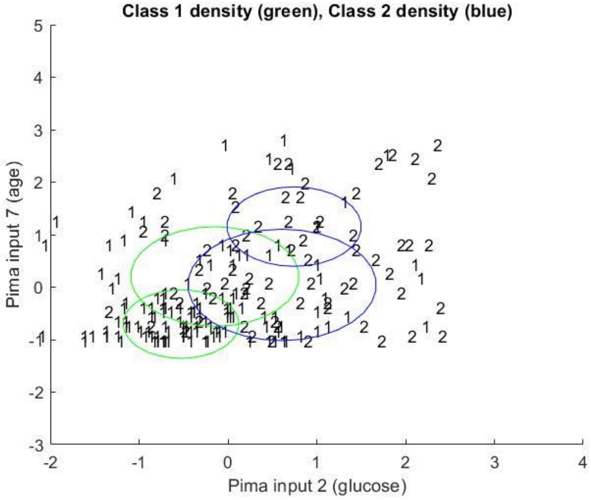
Start
Load and normalize data
Select a new class c and initialize its clusters
Calculate probability of each data point to be part of a k cluster
p(z|k)
Calculate probability of a certain point
p(z)
A
A
p(k|z)=gamma_{nk}
Calculate responsibilities
E
No
Update means, variances, distances and p(k)'s
M
Calculate train and test errors
No
Is nIterations >= maxNIterations?
Yes
B
B
Save the centers, variances and cluster probabilities
Is nClass >= totalNclasses?
Estimate prior probabilities
p(c)
p(x|c)
Compute conditional densities for test set
p(c|x)
Compute conditional class by comparing the ones from the different classes
Compute error on test set
End

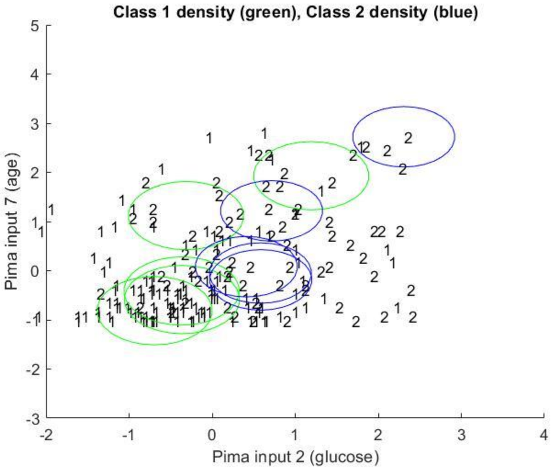
The idea behind this algorithm is that we first estimate the density distributions for each class separetely, and then afterwards we estimate how sure are we that that point is part of our densities. We got the final result by comparing this probability for the different classes.

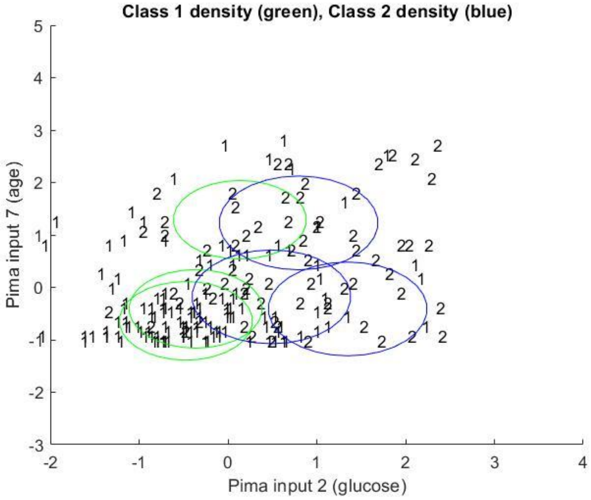
In this case we are plotting just two of the dimensions against each other, which makes it quite difficult to interpret the results, as we are working with 7 dimensional data. Even though we have to deal with this issue, we are still able to get a good idea about the model by representing two important dimensions.

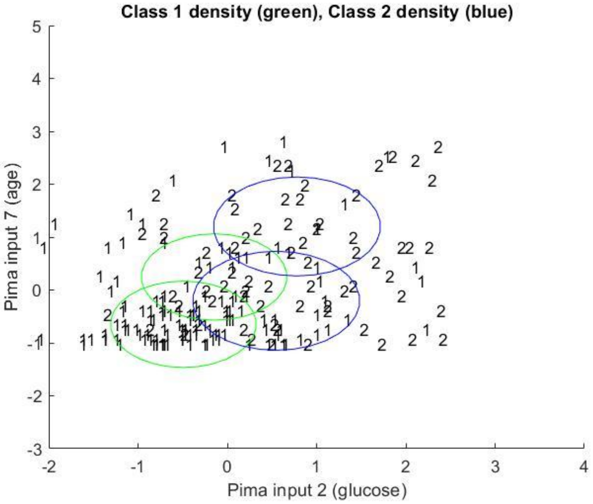






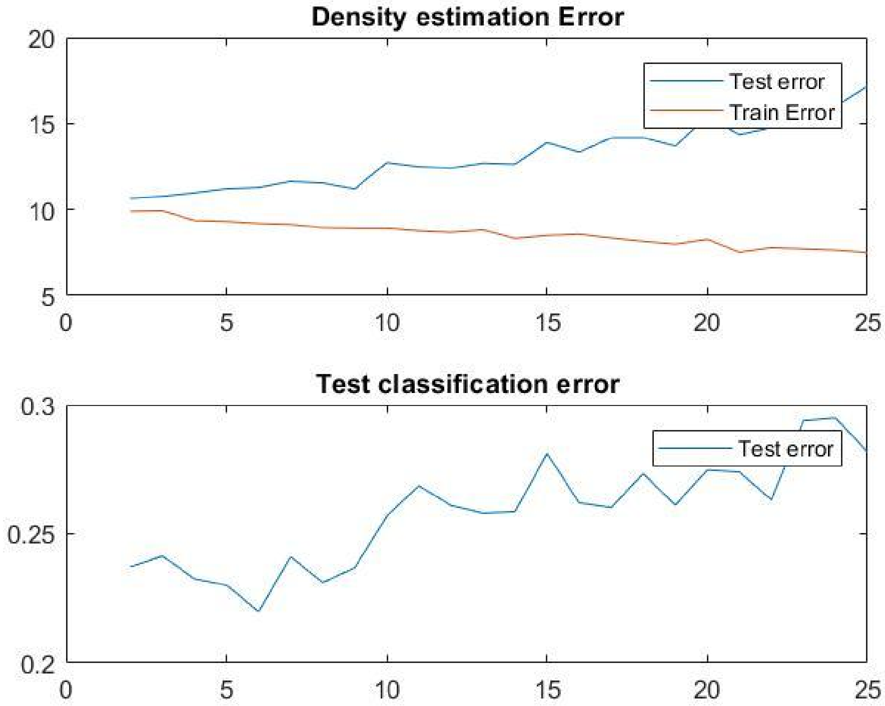






From this results we can see that a higher glucose level correlates with a class 2, and the other way for class 1. However, the threshold between both at around 0 is not that clear, and we have densities from different classes overlapping. Also we note that the class 2 is more spread out on the right side than class 1 in the left side.

The best way to find the most appropiate K is if we take a look now at the error from this experiments as a function of the number of density functions.



We can see how if we only look at the density estimation errors, a K=2 seem to be the best option, as after that the model starts to overfit. However, that error is only representative of how well is our model representing the different classes, but it doesn't provide any information about the classification error.

Thus, it is a better option to look at the second graph, where we can see that the actual lowest classification error is achieved when we use K=6.