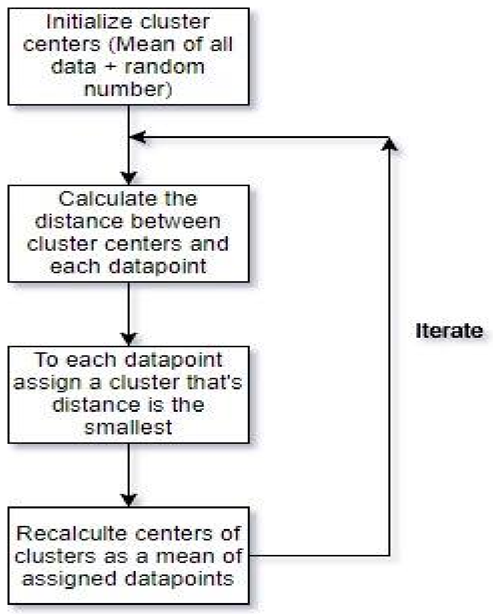
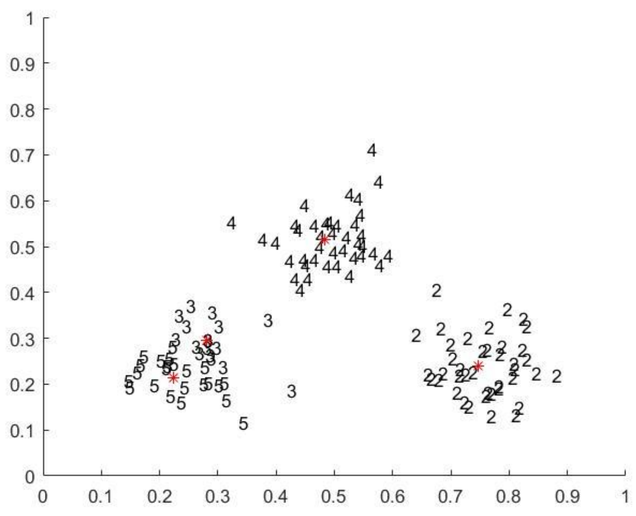
Checkpoint 7.1

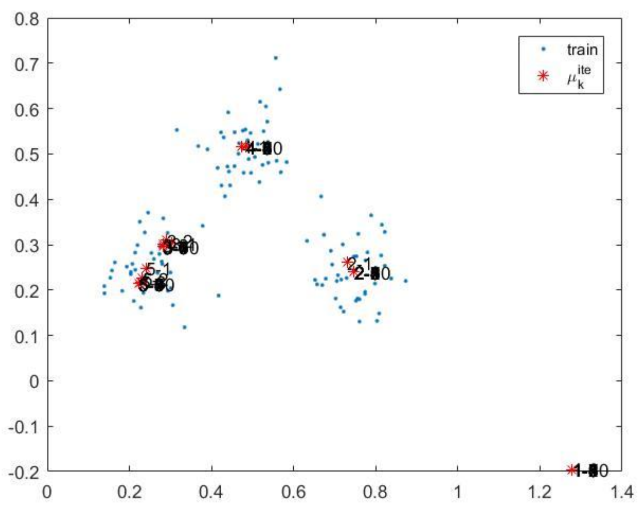
Use the program `main7a.m` to perform K-means analysis on synthetic two-dimensional data with three clusters. The program creates two plots. The first shows the training points and the current position of the cluster centers as time progress. You can zoom to inspect the details of the convergence. The second figure shows the assignment of points to clusters at the end of the procedure. Is the final configuration sensitive to how the initial cluster centers are located? You can change the distribution of the initial clusters by changing the parameter `initial-width` and you can change the number of clusters `K`. Sometimes you will see that a cluster never moves away from its initial position, why?

Firstly, it is interested to understand how the k-means clustering algorithm works, as can be seen in the following flowchart:

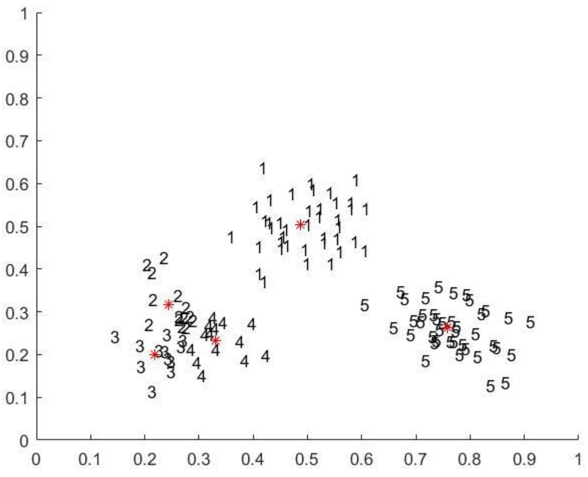

Initialize cluster centers (Mean of all data + random number)
Calculate the distance between cluster centers and each datapoint
To each datapoint assign a cluster that's distance is the smallest
Recalculte centers of clusters as a mean of assigned datapoints
Iterate

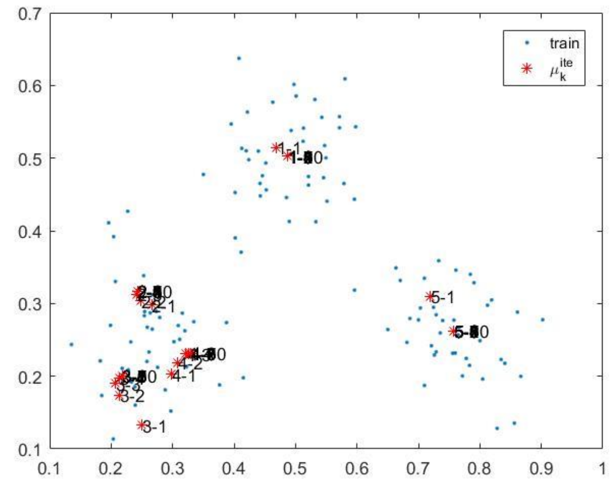
Once the algorithm is understood, let's proceed with the execution of the script, which outputs two different plots:





We can note that for a K=5 and initial_width=0.3 and this random initialization there was one center that does not have any point asigned to it. The reason behind it is this cluster initialization. As the centers are initialized randomly, if one of them is initialized so that there is no point asigned to it this cluster will not be updated, making it idle for the subsequent loops. Sometimes this behaviour is not desired, and it can be corrected by taking a look at how this centers are initialized. In this initialization we add to the main of the training data a random number multiplied by the initial_width parameter. Thus, if we would want to use all the clusters we can reduce the value of that factor, initialazing this way the clusters closer to the mean of the data. We can see this in the next figure:



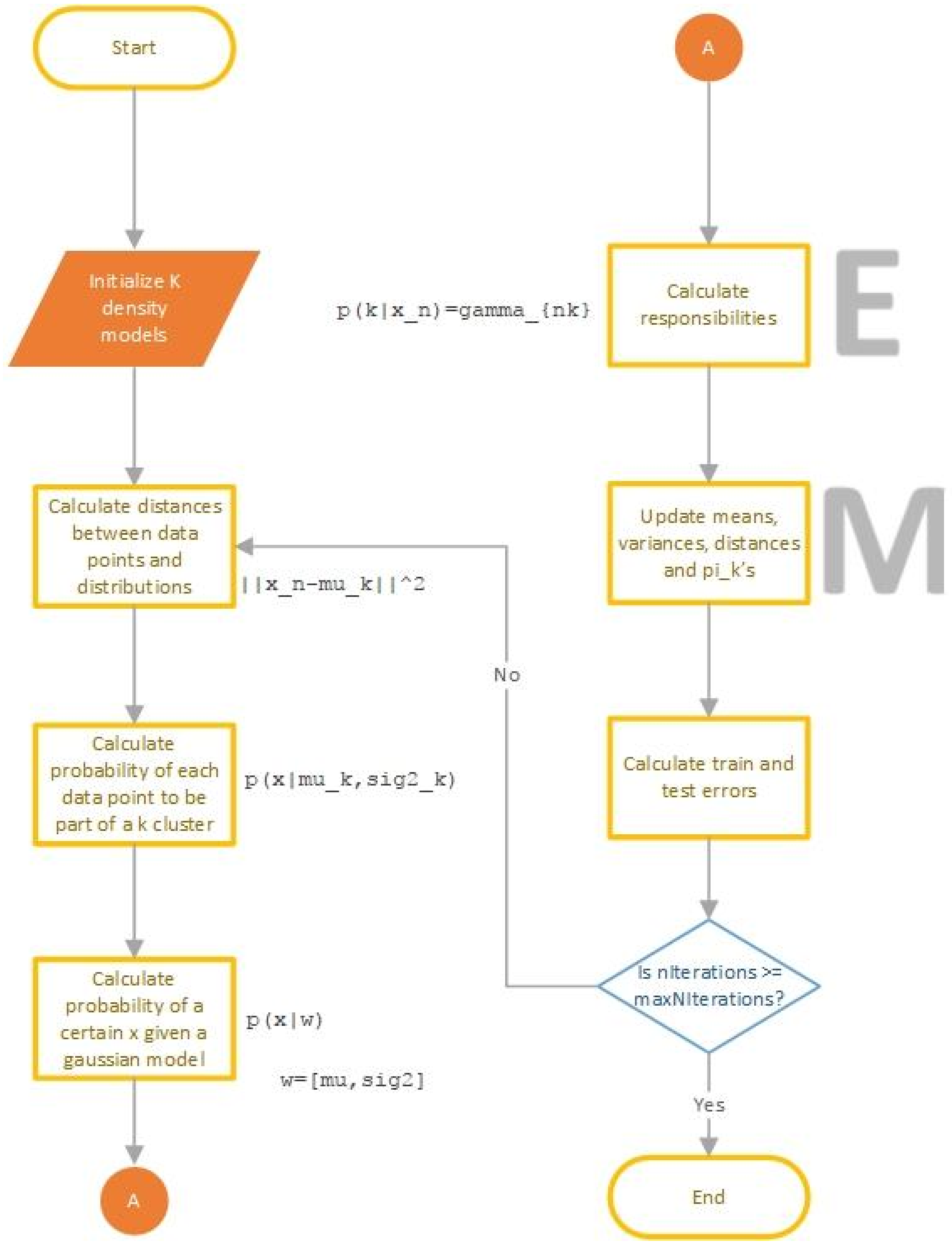


As we can see, the algorithm performs the assignment based on a hard assignment i.e. the point is or it is not part of a cluster. This also allow us to draw the boundaries of each cluster.

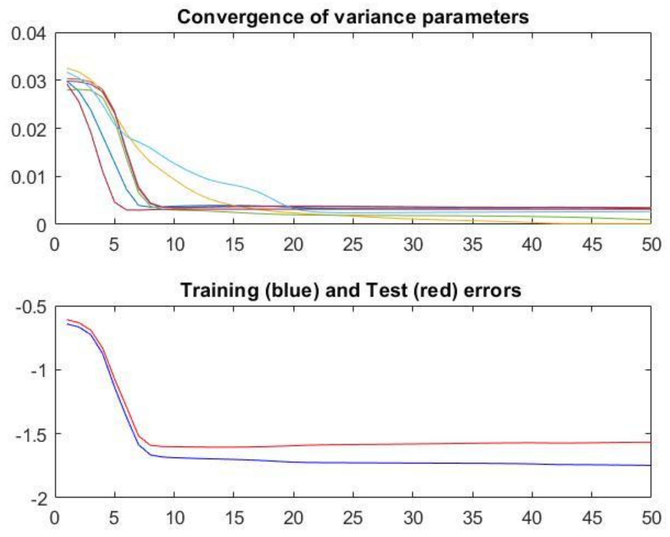
However, we can find also a more complex model, where we perform soft assignment, where each point can be part of different densities functions based on a certain responsibility.

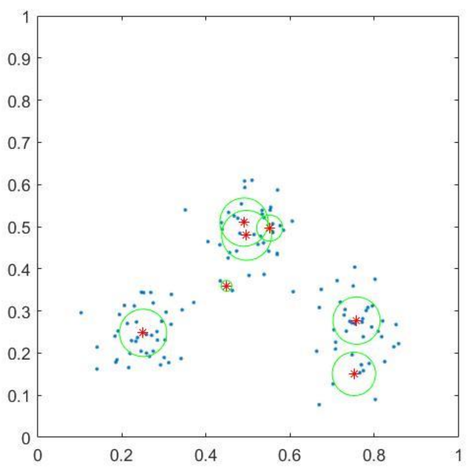
Checkpoint 7.2

The program `main7b.m` uses the Expectation-Maximization algorithm to adapt a Gaussian mixture, as described above. The program shows three plots. The first plot shows the training (blue) and test (yellow) points, the second shows the temporal evolution of the variance parameters and the training and test errors. Create a flow chart of the program `main7b.m` and its functions. The test error is the mean negative log likelihood (the cost function) estimated on the test set. Change the number of clusters $K = 2, 3, 4, 5, 6$ and inspect the temporal evolution and final value of the test error. Explain how the Gaussian mixture can over-fit.

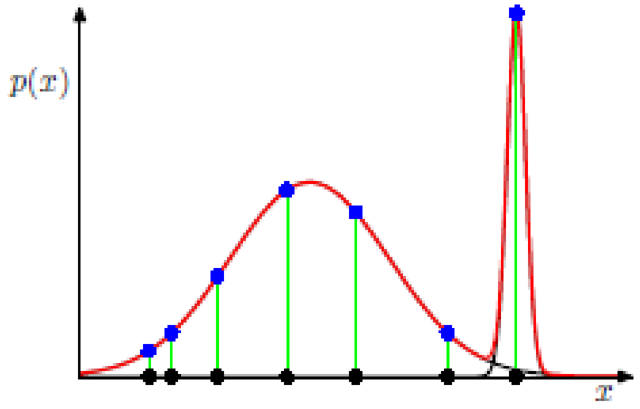

Start
Initialize K density models
Calculate distances between data points and distributions
||x_n-mu_k||^2
Calculate probability of each data point to be part of a k cluster
p(x|mu_k,sig2_k)
Calculate probability of a certain x given a gaussian model
p(x|w)
w=[mu,sig2]
A
A
p(k|x_n)=gamma_{nk}
Calculate responsibilities
E
Update means, variances, distances and pi_k's
M
Calculate train and test errors
Is nIterations >= maxNIterations?
No
Yes
End

Once the algorithm is understood, we can start to run the program with different number of clusters. Thus, for K=7 we obtain the following results:



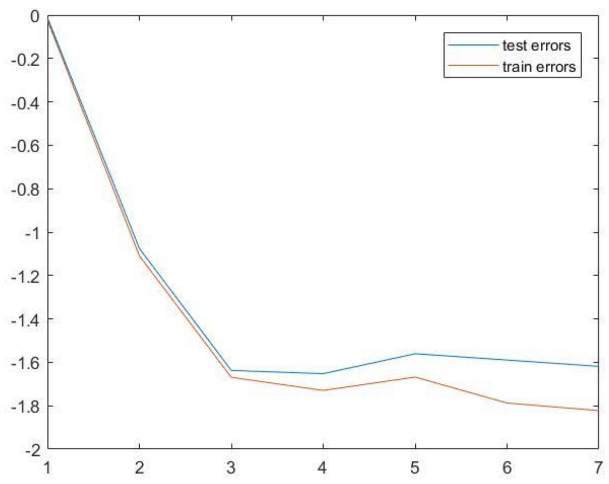


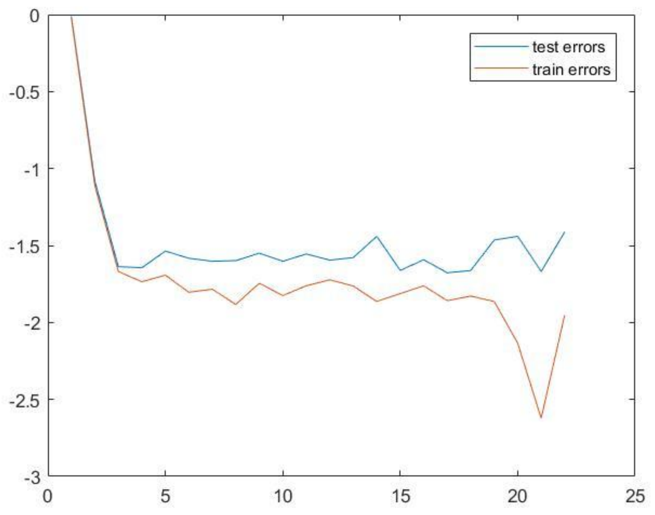
From this graph we can see how the variances converge to a smaller one after being initialize with rather big values. This allows for a reduction in the errors, but also at the risk of overfitting the data with gaussians with really small variance, as it is the case from the training and test error graph. This is actually the case with the estimated gaussian place on the center of the image, which is modelling just two points. This can be better understood if we translate this problem to a one dimensional space:



If we evaluate our error function in the point where the mean is equal to the point of the small variance gaussian we will obtain an small error, because the probabilities are really high based on this model. However, the data is extracted out of a single gaussian distribution, so the lickelihood function should actually give a low value, which means that the error should be higher. This can be seen once the test error is evaluated, because the model will not generalize very well, as a point in the middle of the gaussian distribution will obtain more error than a point equal to the mean of the gaussian on the right.

This overfitting problem, as it is expected it is directly correlated with the number of K selected. In order to probe so we are going to plot the errors as a function of the number of clusters after 30 iterations





As we can see, the error starts to decrease until the optimal number (the true number) of gaussian distributions is selected. From there all the additional clusters will contribute to the overfitting problem that we discuss before. That explains why the test error inmidiatelly goes up while the train error keeps reducing, giving the graph a "u" shape.

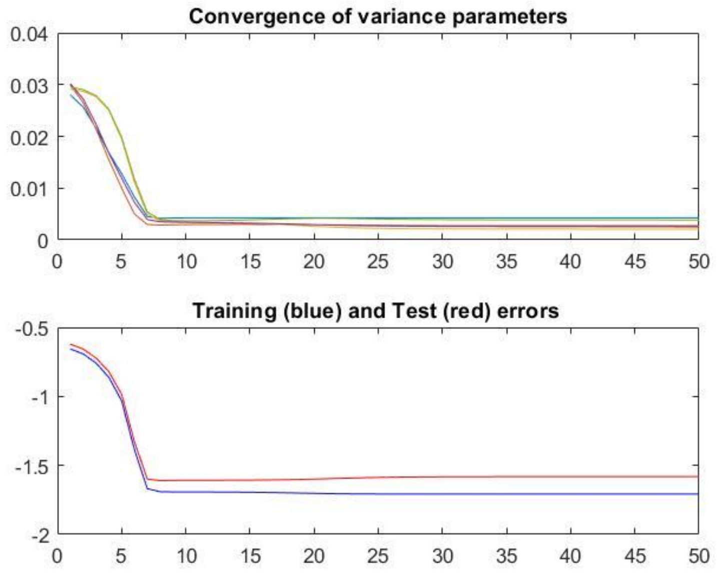
Checkpoint 7.3

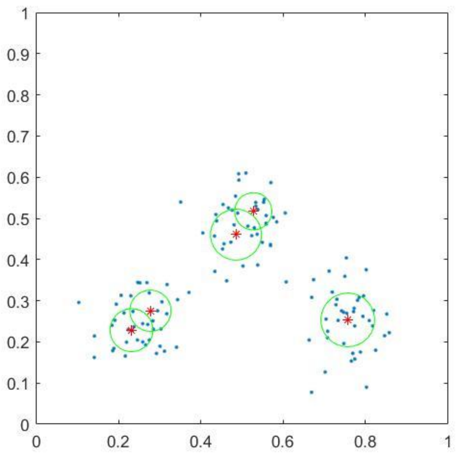
In `main7b.m` you can choose three different strategies for initializing the cluster centers and initial variances. Set $K = 5$ and run the program with the three different schemes.

Describe the strategies and their results. You will see a problem with the Gaussian mixture and the EM algorithm for `method` 3, where one component converge towards a very small variance and is centered on a specific data point. Explain why this is a serious overfit.

Method 1

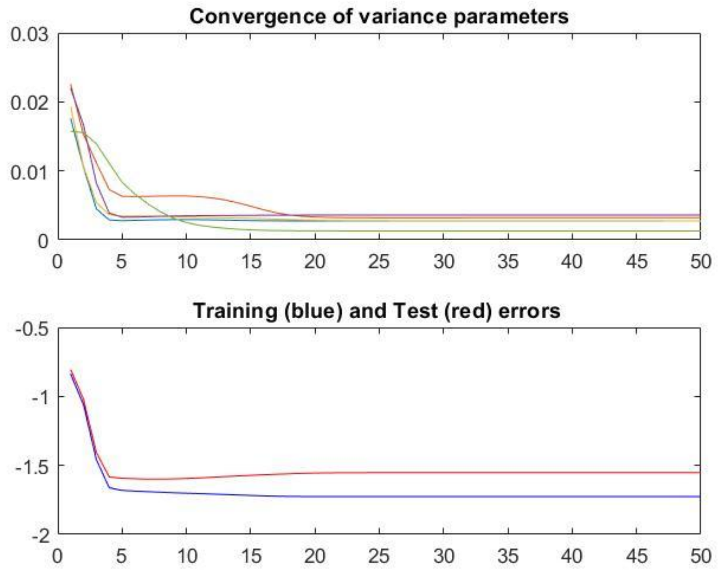
In this case we are going to initialize the centers using K randomly selected points of our dataset. The results, even though sometimes the points selected are far from the optimal values, are pretty good, as can be seen from the images below.

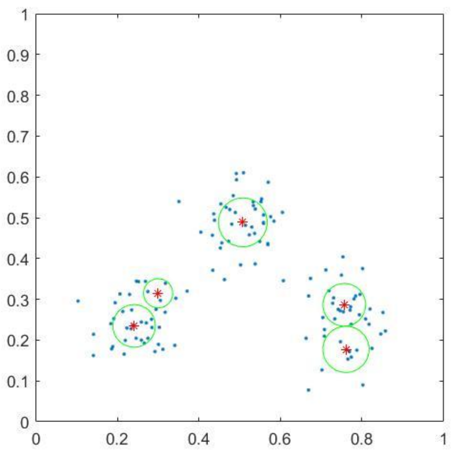




Method 2

In the second method we are selecting the initial points based on a normal distribution calculated over all the training set. Thus, random points of this distribution are selected. The results are similar to the ones in method 1, with a faster convergence in this particular test, but it is not representative of the general performance of the algorithm when different repetitions were performed.





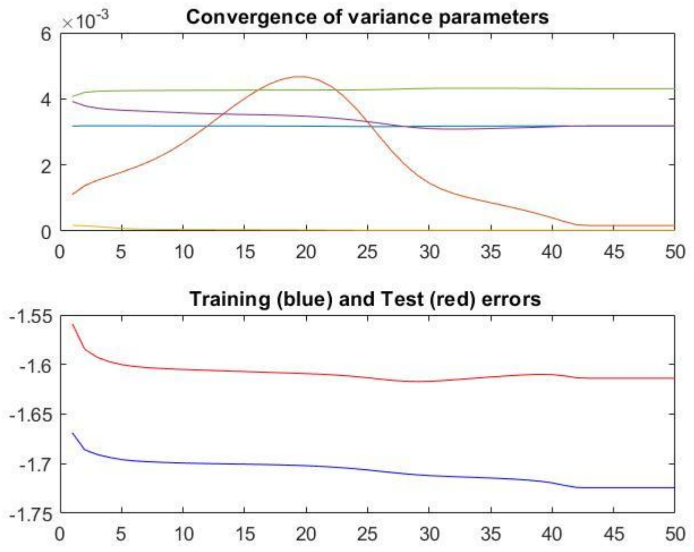
Method 3:

Finally, in the last case we are initializing the points in a similar way of method one, but in this case the variances are set to be really small. In this case, the same issue that we refer to in the previous exercise happens here, causing even bigger issues. This can be explained by taking a look at the equation that gives us the probability of a point of being part of a certain cluster:

$$p(\mathbf{x}|\boldsymbol{\mu}_k, \sigma_k^2) = \frac{1}{(2\pi\sigma_k^2)^{d/2}} \exp\left(-\frac{\|\mathbf{x} - \boldsymbol{\mu}_k\|^2}{2\sigma_k^2}\right) .$$

In the case of the exact points where the means are placed we obtain that exp(0) = 1 and the first term is greately positive, as the small variance is inverted. Thus, the probabilities are really high. However, for any other point of the distribution we obtain a value close to 0 because exp(-big)~0.

Afterwards, in the expectation phase we will obtain all point responsibilities set to [1,0,0,0,0], with the 1 in different positions based on the closest center. This may causes a serious overfitting in the subsequent iterations because a distribution may only take responsibility from a single data point, as we can see in the images below.


Convergence of variance parameters
×10^-3
Training (blue) and Test (red) errors