COMPSCI 527- Homework 3

Due on October 30, 2014

Questions may continue on the back. Type your text. You may write math formulas by hand. What we cannot read, we will not grade.

You may talk about this assignment with others, but do not write down anything while you talk. After that, do the assignment alone. What you hand in must be your work only.

Hand in your solution as a single, stapled paper document at the beginning of class on the due date.

IMPORTANT: Implement all the functions in this assignment from scratch. That is, do not use any MATLAB functions in the Statistics Toolbox, or anyone else's code, unless otherwise instructed.
Keep in mind that the pseudo-code algorithms in this assignment are a way to clarify what a method does, not a recipe for organizing your computation in MATLAB. In particular, when thinking of efficiency, assume that the number K of mixture components is much smaller than the number I of data points. So it is OK to loop explicitly over $1, \ldots, K$ but not over $1, \ldots, I$.

For this assignment you will need the file `data.mat` and a few MATLAB functions, all available in a zip file on the homework page.

1. The K-means algorithm is described in Section 13.4.4 of the textbook. It takes I data points $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_I\}$ in $\mathbb{R}^d$ and K points $\mathbf{m}_1, \ldots, \mathbf{m}_K$ in $\mathbb{R}^d$ and computes a local minimum $\hat{S}$ of the function

$$f(S) = \sum_{k=1}^{K} \min_{\boldsymbol{\mu}_k \in \mathbb{R}^d} \sum_{\mathbf{z} \in S_k} \|\mathbf{z} - \boldsymbol{\mu}_k\|^2$$

where $S = \{S_1, \ldots, S_K\}$ varies over the set $\mathcal{S}$ of all partitions of Z. The partition $\hat{S}$ is a local minimum in the sense that no move of a single point $\mathbf{z}_i$ from one set of $\hat{S}$ to another yields a value of f that is lower than $f(\hat{S})$, nor does any infinitesimal change of the resulting centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$. See algorithm 1 for a pseudo-code listing of the K-means algorithm.

Algorithm 1. The K-means algorithm

```
Input: $\mathbf{z}_1, \ldots, \mathbf{z}_I, \mathbf{m}_1, \ldots, \mathbf{m}_K$
  for $k = 1, \ldots, K$ do
      $\boldsymbol{\mu}_k \leftarrow \mathbf{m}_k$                                   ▷ Initialize the means
  end for
  $f_c \leftarrow \infty$                                      ▷ $f_c$ will become the current value of $f(S)$
  repeat
      for $k = 1, \ldots, K$ do
          $S_k \leftarrow \{\mathbf{z} \mid \mathbf{z} \in Z$ and $\|\mathbf{z} - \boldsymbol{\mu}_k\|^2 \leq \|\mathbf{z} - \boldsymbol{\mu}_\ell\|^2$ for $\ell \neq k\}$   ▷ Partition the points in $Z$ by their closest mean
      end for
      for $k = 1, \ldots, K$ do
          if $S_k \neq \emptyset$ then
              $\boldsymbol{\mu}_k \leftarrow \frac{1}{|S_k|} \sum_{\mathbf{z} \in S_k} \mathbf{z}$                 ▷ $\boldsymbol{\mu}_k$ is the centroid of $S_k$
          end if
      end for
      $f_p \leftarrow f_c$                                   ▷ Remember the previous value of $f(S)$
      $f_c \leftarrow f(\{S_1, \ldots, S_K\})$                    ▷ Compute the current value of $f(S)$
  until $f_p \leq f_c + \tau$                            ▷ Stop if improvement goes below threshold $\tau$
Output: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, S_1, \ldots, S_K$
```

(a) Implement *from scratch* a MATLAB function

```
function [M, R] = kmeans(Z, M)
```

that takes a $d \times I$ matrix `Z` whose columns are the points in Z and a $d \times K$ matrix `M` whose columns are the initial means $\mathbf{m}_k$ and runs the K-means algorithm. The output $d \times K$ matrix `M` has the final means $\boldsymbol{\mu}_k$ as its columns, and the $K \times I$ logical matrix `R` has entries

$$R(k, i) = \begin{cases} \texttt{true} & \text{if } \mathbf{z}_i \in S_k \\ \texttt{false} & \text{otherwise} \end{cases}$$

Use a termination threshold (see the **until** clause in algorithm 1)

$$\tau = \texttt{sqrt}(\texttt{eps})$$

where `eps` is a predefined constant in MATLAB. [Warning: If you compare this matrix with what the textbook calls the *responsibilities* in chapter 7, then your matrix is the transpose of that: $R(k, i) = r_{ik}$.]

Hand in your implementation of `kmeans` and a plot of the result of running

```
load data
[M, R] = kmeans(blobs, M0);
showClusters(blobs, M, R, M0, 1, 'blobsKmeans');
```

The function `showClusters` is provided with this assignment and will produce a PDF file named `blobsKmeans.pdf`. Also turn in the values in `M`.

(b) Run your code on the `cigars` data in `data.mat` using `M0(:, 1:2)` to initialize. Show the output matrix M and your results with `showClusters`.

(c) Run your code on the `cigars` data in `data.mat` again, this time using `M1` (also provided) to initialize. Show the output matrix M and your results with `showClusters`.

(d) Explain in what way the results in your previous two answers show that `kmeans` does not work well with these data points. In particular, are these shortcomings due to the fact that the K-means algorithm computes a local (as opposed to global) minimum? If so, how? If not, what else is responsible?

(e) The K-means algorithm gets trivially in trouble when two or more of the centroids in the initial matrix `M0` are equal to each other. In this way, the coinciding centroids define exactly the same partition, and they never split apart. The resulting behavior is exactly the same as if the duplicate means were removed, and the algorithm effectively returns a result for a lower value of K. However, this is not the only way in which poor initialization leads to bad results.

Construct a set Z of points on the plane and a partition $S = \{S_1, S_2, S_3, S_4\}$ of Z such that the following properties hold:

1. There are four tight, well-separated clusters C_1, C_2, C_3, C_4 in Z, and the cluster centroids are $\boldsymbol{\mu}_1^*$, $\boldsymbol{\mu}_2^*$, $\boldsymbol{\mu}_3^*$, $\boldsymbol{\mu}_4^*$.
2. If $\mathbf{m}_k$ is the centroid of S_k, then the set $\{\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4\}$ is different from the set $\{\boldsymbol{\mu}_1^*, \boldsymbol{\mu}_2^*, \boldsymbol{\mu}_3^*, \boldsymbol{\mu}_4^*\}$.
3. If the K-means algorithm is run on Z with initial centroids $M_0 = [\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \mathbf{m}_4]$, then the algorithm stops after a single iteration of the **repeat** loop in algorithm 1 with

$$\boldsymbol{\mu}_1 = \mathbf{m}_1 \quad , \quad \boldsymbol{\mu}_2 = \mathbf{m}_2 \quad , \quad \boldsymbol{\mu}_3 = \mathbf{m}_3 \quad , \quad \boldsymbol{\mu}_4 = \mathbf{m}_4$$

(in words, the algorithm makes no progress).

Describe your construction clearly and succinctly (a drawing may help), and give the coordinates of the points in S_1, S_2, S_3, S_3, of their centroids $\mathbf{m}_1$, $\mathbf{m}_2$, $\mathbf{m}_3$, $\mathbf{m}_4$, and of the centroids $\boldsymbol{\mu}_1^*$, $\boldsymbol{\mu}_2^*$, $\boldsymbol{\mu}_3^*$, $\boldsymbol{\mu}_4^*$ of the clusters C_1, C_2, C_3, C_4. If you prefer, give formulas for all these coordinates. The clusters need not be large, but each of them must have more than two points. Explain why the algorithm makes no progress with your input. [Hints: It is possible to construct an example with three clusters, but with four it's easier to draw. Symmetry helps. There are many possible answers. You may want to use your `kmeans` function to verify that your construction is correct. However, please do not hand in the result of this check.]

(f) In what way does the construction in the previous question highlight a problem with the K-means algorithm?

2. The EM density estimation algorithm for mixtures of normal distributions is described in section 7.4.2 of the textbook. It takes I data points $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_I\}$ in $\mathbb{R}^d$; K points $\mathbf{m}_1, \ldots, \mathbf{m}_K$ in $\mathbb{R}^d$; and K positive real scalars $\sigma_1^2, \ldots, \sigma_K^2$ and computes a local maximum $\hat{\boldsymbol{\theta}}$ of the likelihood function

$$F(\boldsymbol{\theta}) = \prod_{i=1}^{I} p(\mathbf{z}_i|\boldsymbol{\theta}) \quad \text{where} \quad p(\mathbf{z}_i|\boldsymbol{\theta}) = \sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{z}_i}[\boldsymbol{\mu}_k, \Sigma_k]$$

is a mixture of normals and where $\boldsymbol{\theta} = [\lambda_1, \ldots, \lambda_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K]^T$ is the parameter vector. The vector $\hat{\boldsymbol{\theta}}$ is a local maximum in the sense that no infinitesimal change of $\boldsymbol{\theta}$ away from $\hat{\boldsymbol{\theta}}$ produces an increase of $F(\boldsymbol{\theta})$.

There is a numerical twist to this algorithm. The likelihood $F(\boldsymbol{\theta})$ is the product of positive terms: all the normals are positive, and not all λ_k can be zero (because they add up to one). However, the values of the normals can be close to zero if $\mathbf{z}_i$ is far from $\boldsymbol{\mu}_k$ for all k

where $\lambda_k \neq 0$. In that case, $p(\mathbf{z}_i|\boldsymbol{\theta}) \approx 0$, and if there are enough such small values, their product can cause numerical underflow. That is, the exact value of $F(\boldsymbol{\theta})$ is never zero, but it may bee too small to be stored in a `double` variable, and the EM algorithm will make no progress as a consequence.

To prevent this problem, we instead *minimize* the negative logarithm of $F(\boldsymbol{\theta})$,

$$f(\boldsymbol{\theta}) = -\log F(\boldsymbol{\theta}) = -\sum_{i=1}^{I} \log \sum_{k=1}^{K} \lambda_k \text{Norm}_{\mathbf{z}_i}[\boldsymbol{\mu}_k, \Sigma_k] \ .$$

The minus sign is not important. It is introduced to eliminate the minus signs that arise when taking the logarithm of a Gaussian distribution. In addition, the minus sign makes EM a *minimization* problem, just like K-means.

The important part from a numerical standpoint is the logarithm in the definition of $f(\boldsymbol{\theta})$. The number, say, 10^{-500} is too small to be stored in a `double` variable, but its (natural) logarithm $-500 \log 10$ is about -1,151, and that is not too small. In industrial-quality code, one would have to make sure that EM still works when even $f(\boldsymbol{\theta})$ is too small, but we will not worry about this in this assignment. You may assume that a single value of $p(\mathbf{z}_i|\boldsymbol{\theta})$ can be represented in a `double` variable, even if their product for $i = 1, \ldots, I$ cannot.

See algorithm 2 for a pseudo-code listing of the EM algorithm, cast as a local minimization algorithm.

Algorithm 2. The EM algorithm

```
Input: $\mathbf{z}_1, \ldots, \mathbf{z}_I, \mathbf{m}_1, \ldots, \mathbf{m}_K, \sigma_1^2, \ldots, \sigma_K^2$
  for $k = 1, \ldots, K$ do
    $\lambda_k \leftarrow 1/K$          ▷ Initialize the mixture coefficients to an uninformative prior
    $\boldsymbol{\mu}_k \leftarrow \mathbf{m}_k$          ▷ Initialize the means
    $\Sigma_k \leftarrow \sigma_k^2 I_d$          ▷ Initialize the covariances. $I_d$ is the $d \times d$ identity
  end for
  $f_c \leftarrow \infty$          ▷ $f_c$ will become the current value of $f(\boldsymbol{\theta})$
  repeat
    for $k = 1, \ldots, K$ do
      for $i = 1, \ldots, I$ do
        $r_{ki} \leftarrow \frac{\lambda_k \text{Norm}_{\mathbf{z}_i}[\boldsymbol{\mu}_k, \Sigma_k]}{\sum_{j=1}^{K} \lambda_j \text{Norm}_{\mathbf{z}_i}[\boldsymbol{\mu}_j, \Sigma_j]}$          ▷ Compute the responsibilities. This is the E step
      end for
    end for
    for $k = 1, \ldots, K$ do          ▷ The three assignments in this loop are the M step
      $\lambda_k \leftarrow \frac{1}{I} \sum_{i=0}^{I} r_{ki}$          ▷ $\lambda_k$ is the coefficient of the $k$-th mixture component
      $\boldsymbol{\mu}_k \leftarrow \frac{\sum_{i=1}^{I} r_{ki} \mathbf{z}_i}{\sum_{i=1}^{I} r_{ki}}$          ▷ $\boldsymbol{\mu}_k$ is the sample mean of the $k$-th mixture component
      $\Sigma_k \leftarrow \frac{\sum_{i=1}^{I} r_{ki} (\mathbf{z}_i - \boldsymbol{\mu}_k)(\mathbf{z}_i - \boldsymbol{\mu}_k)^T}{\sum_{i=1}^{I} r_{ki}}$          ▷ $\Sigma_k$ is the sample covariance of the $k$-th mixture component
    end for
    $f_p \leftarrow f_c$          ▷ Remember the previous value of $f(\boldsymbol{\theta})$
    $f_c \leftarrow f(\lambda_1, \ldots, \lambda_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K)$          ▷ Compute the current value of $f(\boldsymbol{\theta})$
  until $f_p \geq f_c + \tau$          ▷ Stop if improvement falls below $\tau$
Output: $\lambda_1, \ldots, \lambda_K, \boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_k, r_{11}, \ldots, r_{KI}$
```

(a) Implement *from scratch* a MATLAB function

```
[lambda, M, Sigma, R] = EM(Z, M, sigma2)
```

that takes the same arguments `Z` and `M` as `kmeans` in addition to a vector `sigma2` of K positive scalars to initialize EM. Your function `EM` uses the variances σ_k^2 in `sigma2` to generate the initial values of the $d \times d$ mixture covariance matrices as

$$\Sigma_k = \mathtt{sigma2(k)} * \mathtt{eye(d)}$$

where `eye(d)` generates the $d \times d$ identity matrix and `d=size(Z,1)`. The output `lambda` of `EM` is a K-dimensional vector with the mixture coefficients λ_k. The output `M` is a $d \times K$ matrix that collects the estimated centroids of the mixture components. The output `Sigma` is a $d \times d \times K$ array such that

$$\mathtt{Sigma}(:,:,\mathtt{k}) = \Sigma_k \ ,$$

the full k-th covariance matrix estimated by the EM algorithm. The $K \times I$ output matrix R contains the *responsibilities* computed by `EM`. Use the same termination threshold as in `kmeans`.

[Warning: If you compare this matrix with the responsibilities the textbook defines in chapter 7, then your matrix is the transpose of that: $R(k, i) = r_{ik}$.]

Hand in your implementation of EM and a plot of the result of running

```
load data
[lambda, M, Sigma, R] = EM(blobs, M0, ones(1, 3));
showMixture(blobs, lambda, M, Sigma, R, M0, 1, 'blobsEM')
```

The function showMixtures is provided with this assignment, and will produce a PDF file named blobsEM.pdf. The resulting plot is in color, so you see more clearly what happens on screen, but it is OK to turn in a black-and-white print. Also give the resulting values of lambda, M, and Sigma.

(b) Run your function EM with first argument cigars, initializing with M0(:, 1:2).

(c) Is this result more satisfactory than using K-means on the same data? Explain briefly.

(d) With either K-means or EM you need to know the number of clusters ahead of time. Show plots (using showMixture) of running your EM function on the blobs data with $K = 2$ and then with $K = 4$. Use M0(:, 1:2)) and M04 (provided) to initialize.

(e) Run your function EM with first argument bananas (provided), initializing with M0(:, 1:2). Turn in M and the plot generated with showMixture.

(f) In what ways is the result in your answer to the previous question unsatisfactory? Is this a limitation of the EM algorithm or of modeling data with a mixture of normal distributions? Explain briefly.

3. The issues with EM (and therefore with K-means) highlighted by some of your answers in the previous problem motivate us to find a clustering method (that is, an algorithm that groups data into tight groups for some definition of "tight") that can detect non-convex clusters and does not require knowing the number of clusters ahead of time. One such method is based on the *mean shift* mode-seeking algorithm introduced by K. Fukunaga and L. D. Hostetler (*IEEE Transactions on Information Theory*, 21(1):32-40, 1975). Mean shift is *not* a clustering algorithm in itself, but can be used to make one. So please bear with me.

Let $K_h(\mathbf{z})$ be a Gaussian *kernel* defined as follows:

$$K_h(\mathbf{z}) = e^{-\left(\frac{\|\mathbf{z}\|}{h}\right)^2}$$

where $h > 0$ is called the *bandwidth* of the kernel ($K_h(\mathbf{z})$ is not a probability density, as it is not normalized to integrate to one). Given a set $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_I\}$ of data points in $\mathbb{R}^d$, the quantity

$$\phi_h(\mathbf{z}) = \sum_{i=1}^{I} K_h(\mathbf{z} - \mathbf{z}_i) ,$$

is a measure of the local density of the data in a neighborhood of $\mathbf{z}$: If there are many data points $\mathbf{z}_i$ near $\mathbf{z}$, then the value of $\phi_h(\mathbf{z})$ is large. The vector

$$\boldsymbol{\mu}_h(\mathbf{z}) = \frac{\sum_{i=1}^{I} \mathbf{z}_i K_h(\mathbf{z} - \mathbf{z}_i)}{\sum_{i=1}^{I} K_h(\mathbf{z} - \mathbf{z}_i)}$$

is an average of the data $\mathbf{z}_i$ weighted by a decreasing function of their distance from $\mathbf{z}$, and can therefore be interpreted as the local centroid (or mean) of the data around $\mathbf{z}$. If the data is locally Gaussian, the density at the centroid $\boldsymbol{\mu}_h(\mathbf{z})$ is no less than the density at $\mathbf{z}$. Figure 1 illustrates this point.

Thus, we have found (i) a way to measure the data density around any point $\mathbf{z} \in \mathbb{R}^d$ and (ii) a new point $\mathbf{z}' = \boldsymbol{\mu}_h(\mathbf{z})$ where the density is equal to or greater than that at $\mathbf{z}$. These observations yield an astonishingly simple algorithm that seeks the mode of the density of the data in Z: Start anywhere (at some initial point $\mathbf{z}_{\text{start}}$) and keep shifting from $\mathbf{z}$ to the local mean $\boldsymbol{\mu}_h(\mathbf{z})$ (which becomes the new $\mathbf{z}$), until $\mathbf{z}$ and $\boldsymbol{\mu}_h(\mathbf{z})$ coincide. Algorithm 3 summarizes this procedure. Of course, this algorithm is local, and which mode it finds depends on $\mathbf{z}_{\text{start}}$. In addition, what "local" means depends on the value of the bandwidth parameter h.

(a) Implement *from scratch* a MATLAB function with header

```
function [z, zh] = meanShift(zstart, Z, h)
```

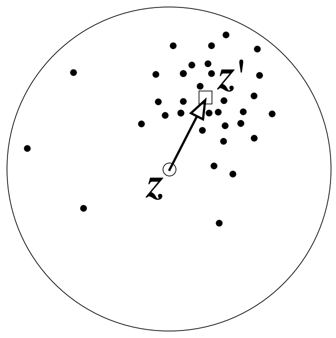

Figure 1. The large circle suggests a Gaussian function centered at $\mathbf{z}$ (small hollow circle). The distribution of points (black dots) under this Gaussian is lopsided, and the weighted mean $\mathbf{z}'$ (small hollow square) of the data does not coincide with $\mathbf{z}$. If $\mathbf{z}$ is moved (arrow) to point $\mathbf{z}'$, then the local density increases.

Algorithm 3. The mean shift algorithm

```
Input: $\mathbf{z}_1, \ldots, \mathbf{z}_I, \mathbf{z}_{\text{start}}, h > 0$
  $\mathbf{z}' \leftarrow \mathbf{z}_{\text{start}}$
  repeat
      $\mathbf{z} \leftarrow \mathbf{z}'$
      $\mathbf{z}' \leftarrow \frac{\sum_{i=1}^{I} \mathbf{z}_i K_h(\mathbf{z}-\mathbf{z}_i)}{\sum_{i=1}^{I} K_h(\mathbf{z}-\mathbf{z}_i)}$
  until $\|\mathbf{z} - \mathbf{z}'\| \leq \tau$                    ▷ Stop if there is not enough improvement
Output: $\mathbf{z}'$
```

that computes a local mode of the data in a set Z starting at some starting point $\mathbf{z}_{\text{start}}$, and with bandwidth h. The input `zstart` is a $d \times 1$ vector with the starting point; the $d \times I$ input matrix `Z` contains I data points in its columns; and the positive input scalar h is the bandwidth. The output `z` is a $d \times 1$ vector with the local mode of the data. If the algorithm takes k steps to reach the mode, then the output `zh` is a $d \times k$ matrix that records the path traversed from `zstart` to `z`.

Use termination threshold

$$\tau = h/1000 .$$

Hand in your code and the two plots resulting from the calls

```
zstart = [0 1.5]';
[z1, zh1] = meanShift(zstart, blobs, 0.2);
[z2, zh2] = meanShift(zstart, blobs, 2);
showPaths({zh1, zh2}, blobs, 'paths');
```

The function `showPaths` is provided with this assignment, and will produce a PDF file named `paths.pdf`. It is OK to hand in a black-and-white print. Also state how many iterations it took for each of the two paths.

(b) Explain the apparently unintuitive result for the greater bandwidth $h = 2$.

4. The mean shift algorithm can be used to construct a *mean shift clustering* algorithm. The recipe is again very simple: run the mean shift algorithm I times, starting at each of the data points in turn, and record the final mode for each run. Points that are in the same cluster (at a scale measured by the bandwidth h of the mean shift algorithm) will converge to the same mode, so we can group all the points in Z by the resulting modes: if two points lead to the same mode, they belong to the same cluster. Algorithm 4 summarizes.

(a) Write a MATLAB function with header

```
function [U, R] = meanShiftCluster(Z, h)
```

that takes a $d \times I$ array `Z` of data and a positive bandwidth `h` and returns a $d \times K$ array `U` with K modes and a logical $K \times I$ array `R` of cluster memberships, in the same style as `kmeans`. K is the number of clusters that `meanShiftCluster` finds.

Algorithm 4. The mean shift algorithm

```
Input: $Z = \{\mathbf{z}_1, \ldots, \mathbf{z}_I\}$ and $h > 0$
  for $i = 1, \ldots, I$ do
      $M(:, i) \leftarrow \text{meanShift}(\mathbf{z}_i)$        ▷ meanShift terminates when there is less than $\tau$ motion
  end for
  $U \leftarrow$ near-unique columns$(M)$        ▷ Two columns in $M$ are the same if their Euclidean distance is less than $\tau/10$
  $K \leftarrow$ number of columns$(U)$        ▷ The algorithm determines the number of clusters on its own
  for $k = 1, \ldots, K$ do
      for $i = 1, \ldots, I$ do
          if $M(:, i) = U(:, k)$ then
              $R(k, i) \leftarrow 1$
          else
              $R(k, i) \leftarrow 0$
          end if
      end for
  end for
Output: Modes $U$ and logical matrix $R$ of cluster memberships
```

Use termination threshold

$$\tau = h/1000$$

for meanShift, and declare two modes to be the same if they are more than 10τ apart.

Hand in your code and a plot that uses `showClusters` to show the modes and the clusters found on the `bananas` data set with a bandwidth $h = 0.8$. [You do not have the argument `M0`, so you can pass the empty matrix instead.] Also report the running time of your code in seconds, and the number of clusters found.

[Hints: While the algorithm is simple in principle, its computational cost is high: For each starting point $\mathbf{z}_i$ in Z and at each iteration of mean shift it is necessary to compute the distance between $\mathbf{z}$ and every point in Z. So with k iterations per point on average mean shift clustering takes $O(kI^2)$ distance computations.

To reduce the computational burden, note that points $\mathbf{z}_i$ that are very far from $\mathbf{z}$ (relative to h) lead to near-zero values of $K_h(\mathbf{z} - \mathbf{z}_i)$, and can be ignored. Ways to take advantage of this fact are beyond the scope of this course, so you may use the function `rangesearch` available in the MATLAB Statistics Toolbox. If you do not have that toolbox, you can download a similar function from the Mathworks web site by searching for "Yi Cao fast range search". Ignore distances that are greater than $2h$.]

(b) Look (carefully) at your result in your previous answer and comment on the extent to which mean shift clustering addresses the problems of EM. Discuss trade-offs on the selection of the bandwidth h.