Resumen y análisis:

En este caso no tuvimos que preocuparnos por calcular el número de centroides, ni por calcular las coordenadas iniciales de los centroides, ya que nos fueron dados estos datos desde un principio.
Antes de comenzar a trabajar inicializamos algunas variables para que nuestros arrays no tengan la necesidad de crecer sobre la marcha y así ahorrarnos tiempo de cómputo:

```
%Inicializamos variables
num_samples = size(SAMPLES2,1);
K = size(Centroids,1);
Asignacion=zeros(num_samples,1);
Distancias=zeros(num_samples,K);
CentroidesDiferentes=true;
```

Nuestro trabajo con el algoritmo de K-Means comienza desde la parte en que calculamos las distancias entre los centroides con nuestros samples/muestras. A continuación la manera en que atacamos el problema:

```
%Mientras los centroides sean diferentes sigue trabajando
while(CentroidesDiferentes)

    %Sacamos Asignacion y Distancias
    %Recorremos el num de samples (633)
    for i=1 : num_samples
        %Setteamos un valor alto en distMinima para que tome el primer
        %valor
        distanciaMinima=999999999999;

        %Recorremos el num de centroides (8)
        for j=1 : K
            %Calculamos la distancia entre el centroide j y el sample i
            distanciaActual = distancia(SAMPLES2, Centroids, i, j);
            Distancias(i,j) = distanciaActual;
            %Al encontrar la distancia minima del centroide j a nuestro sample
            %(puede ocurrir varias veces)
            % cambiamos la asignacion de nuestro sample i
           if(distanciaMinima > distanciaActual )
             distanciaMinima = distanciaActual;
             Asignacion(i,1) = j;
           end
        end
    end
```

El código de arriba, realiza 3 ciclos for anidados. El primer for recorre el número de muestras en nuestro caso. El segundo recorre el número de centroides con los que

estamos trabajando. El tercero for que se encuentra dentro de los 2 anteriores, se encuentra en nuestra función de distancia, la cual fue elaborada por nosotros y funciona de la siguiente manera:

```
function [dist]=distancia(SAMPLES2, Centroids, numSample, numCentroide)
dist=0;
for i=1 : size(SAMPLES2,2)
      dist = dist + ( SAMPLES2(numSample,i) - Centroids(numCentroide,i) )^2;
end
```

El objetivo de estos 3 for anidados, es comparar la distancia euclidiana que existe entre cada uno de los samples (633) con cada uno de los centroides (8). Al encontrar las 8 distancias que existen desde los centroides hacia un solo sample se determina cual de esas distancias es la más pequeña y sé asigna a un vector (llamado Asignación) que contiene esta información para cada sample.

Al calcular nuestra matriz de distancia, estamos listos para recalcular nuestros centroides, esto se realiza sacando un promedio de las muestras asignadas que tiene cada centroide, para esto utilizaremos nuestro vector de Asignación y nuestra matriz de SAMPLES2 como se muestra a continuación:

```
%Sacamos CentroidsNew:
   CentroidsNew=zeros(K,num_attributes);
   for i=1 : K
      %Conseguimos los indices donde se encuentran asignados los 8 centroides
      indx = find(Asignacion==i);

      %Recorremos todos los indices que encontramos con el K en cuestion
      for j=1 : length(indx)
         %Sumamos en CentroidsNew los samples que le corresponden al
         %centroide (suma de vectores)
         CentroidsNew(i,:)= CentroidsNew(i,:) + SAMPLES2 (indx(j,:),:);

      end

      %Dividir CentroidsNew entre el numero de Samples que se encontraron
      %para ese centroide
      CentroidsNew(i,:) = CentroidsNew(i,:)/length(indx);
   end
```

En nuestro primer for recorremos los centroides (1-8) y en nuestro segundo for recorremos el número de asignaciones que tiene nuestro centroide en cuestión, para esto nos sirve nuestro array indx; el cual se redefine en cada iteración del primer for. Pero además de eso nuestro array indx nos sirve para ir encontrando y acumulando en forma de sumatoria los renglones (samples) que correspondan al centroide con que estemos trabajando.

Para terminar el algoritmo nos queda una parte muy sencilla, un simple if para comprobar si nuestros centroides viejos tienen alguna diferencia con nuestros centroides recién calculados.

```
    %Comprobamos si los Centroides han cambiado
    if (Centroids == CentroidsNew)
        CentroidesDiferentes=false;
    end
    %Nos preparamos para otra iteracion en caso de, ya que CentroidsNew
    %albergara los nuevos centroides
    Centroids=CentroidsNew;
end
```

Además después igualamos nuestra matriz de centroides viejos con la de CentroidsNew, esto con el fin de que en la siguiente iteración (en caso de existir), estemos listos para recibir una nueva matriz con nuevos valores de centroides.

Todo este proceso está ciclado con un while (que se observa al inicio), que se rompe únicamente después de encontrar que las matrices de centroides son iguales.

Para terminar únicamente nos queda comparar nuestros resultados con nuestro “ground truth”. Para esto primeramente inicializamos un vector y una matriz:

```
efectividad=zeros(8,1);
EquivalenteAsignacionLabels=zeros(8,8);
```

El vector efectividad guardará los resultados finales, mientras que la matriz EquivalenteAsignacionLabels, guardara los porcentajes de coincidencia de cada centroide nuestro contra cada label. Por ejemplo en EquivalenteAsignacionLabels(7,4) tenemos un porcentaje de 91.55% de asertividad, que es el más alto en ese renglón. Esto nos está indicando que el label 7 le pertenece a nuestro centroide 4.

La manera en que calculamos estas matrices se basa en 3 fors anidados, los 2 más exteriores recorren de 1 a 8, para poder realizar todas las comparaciones de todos los labels con todos nuestros centroides.
En nuestro segundo for realizamos un mapa de ceros con el valor que tiene “i” y “c” en ese momento (recorrimiento de los 2 primeros fors). En nuestro tercer for, recorremos de 1 a 633 (num de samples) para ir sumando la cantidad de veces que ambos coincidieron en el mismo número.

```
if(mapLabels(j,1) == 1 && mapAsignacion(j,1) == 1)
            PorcentajeCoincidencia = PorcentajeCoincidencia + 1;
        end
```

Hasta este momento solo tenemos el número de veces que coincidieron, no un porcentaje. Para arreglar esto, esperamos a que termine nuestro 3er y más interno for. Al terminar podremos dividir PorcentajeCoincidencia entre el número total de veces que pudieron haber coincidido, basado en labels.

```
EquivalenteAsignacionLabels(i,c) = PorcentajeCoincidencia / length(indxLabels) ;
```

El código completo se ve de esta manera:

```
%Recorrer centroides en Labels para comparacion
for i=1 : K
    %Recorrer centroides en Asignacion para comparacion
    for c=1 : K
        PorcentajeCoincidencia=0;
        mapLabels = SAMPLES_FIXED_Kmeans_Labels==i;

        mapAsignacion = Asignacion==c;
        %Indices del centroide de Labels con el que estamos
        %trabajando
        indxLabels = find(SAMPLES_FIXED_Kmeans_Labels==i);
        for j=1 : num_samples
            if(mapLabels(j,1) == 1 && mapAsignacion(j,1) == 1)
                PorcentajeCoincidencia = PorcentajeCoincidencia + 1;
            end
        end
        EquivalenteAsignacionLabels(i,c) = PorcentajeCoincidencia / length(indxLabels) ;
    end
   efectividad(i,1) = max(EquivalenteAsignacionLabels(i,:));
end
```

Lo único faltante por explicar es que tomamos el valor máximo de EquivalenteAsignacionLables y lo tomamos como nuestro porcentaje de acierto, dentro del vector efectividad.