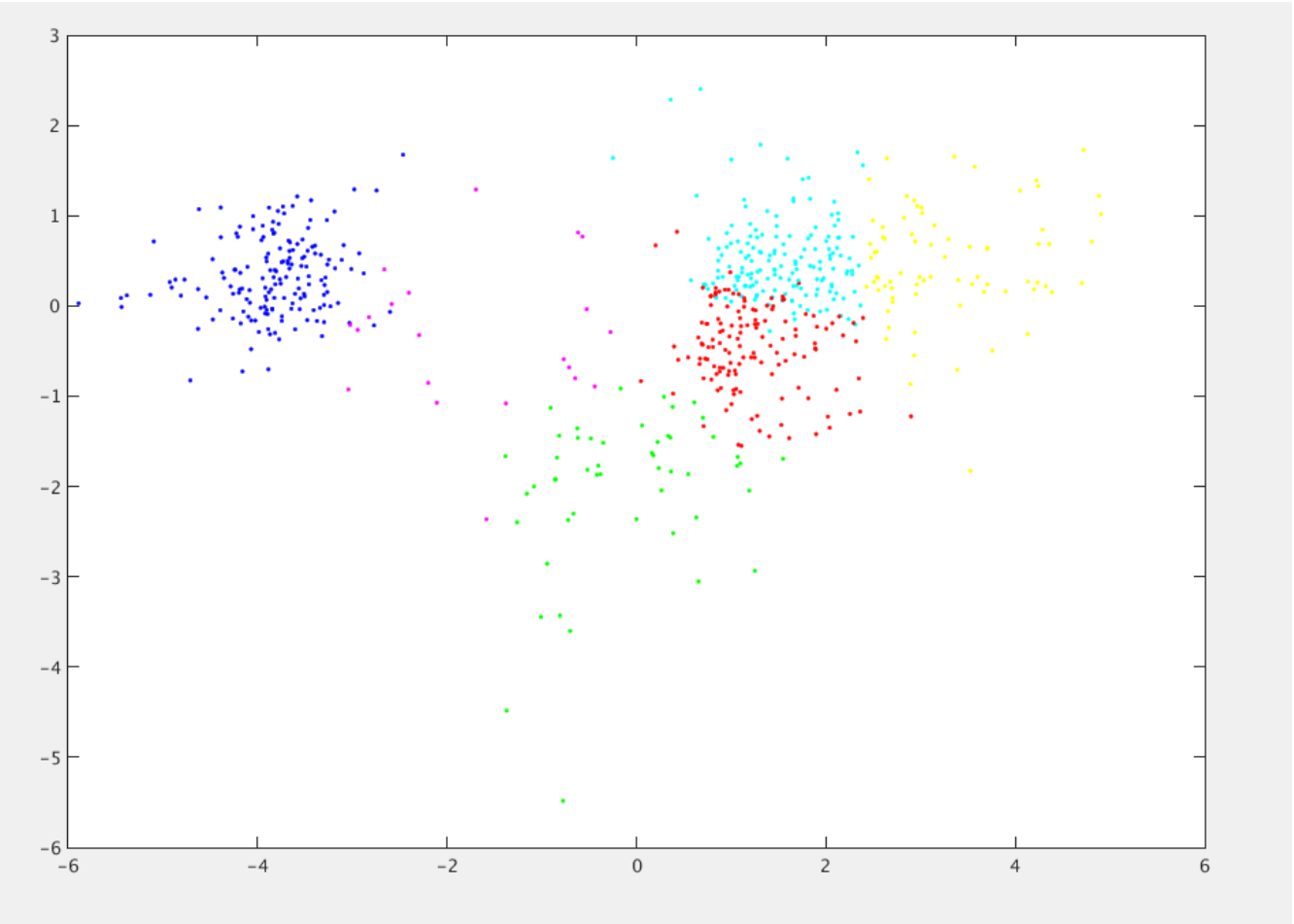
Yeast Clusters:

Cluster 1: -0.2413 -0.1288 0.0622 0.1734 0.2179 1.6517 1.9053
Cluster 2: -0.9535 -1.4716 0.0775 -0.1795 -1.0048 1.1512 0.9688
Cluster 3: 0.1651 0.0917 -0.1039 -0.5526 -0.6301 -1.7232 -1.7548
Cluster 4: 0.0233 0.2508 -0.277 -0.364 -0.7354 -0.8946 0.7001
Cluster 5: -0.0016 0.1565 0.3563 0.7016 1.0097 1.8423 1.6434
Cluster 6: -0.0393 0.1539 0.4361 1.1058 1.4487 3.0163 2.8294

Order for Utilities:

12	21	23
10	13	24
4	24	25
7	23	26
20	25	27
14	19	28
1	18	29
15	26	30
28	29	31
2	27	32
8	16	33
30	32	34
22	34	35
9	31	36
35	36	37
6	37	38
3	38	39
33	39	40
17	40	41
11	41	42
5	42	43



Codes:

```
	private static void run_hierarch(ArrayList<ArrayList<Double>> data) {
//		System.out.println("Running hierarchical algorithm...");
		ArrayList<ArrayList<Double>> distance_matrix = new
ArrayList<ArrayList<Double>>();
		ArrayList<Integer> removed_clusters = new ArrayList<Integer>();

		// First, we create a distance matrix
		for(int i = 0; i < data.size(); i++)
		{
			distance_matrix.add(new ArrayList<Double>());
			for(int j = 0; j < data.size(); j++)
			{

	distance_matrix.get(i).add(euclidean_dist(data.get(i),data.get(j)));
			}
		}

		// Next, the actual algorithm
		while(removed_clusters.size() < distance_matrix.size()-1)
		{
```

```
                  // First, we must find the two clusters with the smallest
distance
                  double min = 90000;
                  int row = 0;
                  int col = 1;
                  for(int i = 0; i < distance_matrix.size(); i++)
                  {
                        if (!removed_clusters.contains(i))
                        {
                              for(int j = i + 1; j < distance_matrix.size(); j++)
                              {
                                    if (!removed_clusters.contains(j))
                                    {
                                          if(distance_matrix.get(i).get(j) < min)
                                          {
                                                min =
distance_matrix.get(i).get(j);
                                                row = i;
                                                col = j;
                                          }
                                    }
                              }
                        }
                  }

                  System.out.println((row+1) + "\t" + (col+1) + "\t" +
(distance_matrix.size()+1));

                  removed_clusters.add(row);
                  removed_clusters.add(col);

                  // Add new cluster to distance matrix
                  distance_matrix.add(new ArrayList<Double>());
                  for(int i = 0; i < distance_matrix.size()-1; i++)
                  {
                        distance_matrix.get(distance_matrix.size()-
1).add(Math.min(distance_matrix.get(row).get(i),distance_matrix.get(col).get(i)));
                  }
                  for(int i = 0; i < distance_matrix.size(); i++)
                  {


distance_matrix.get(i).add(Math.min(distance_matrix.get(row).get(i),distance_matri
x.get(col).get(i)));
                  }

            } // end while loop
      }

      private static int[] run_kmeans(ArrayList<ArrayList<Double>> Data,
ArrayList<ArrayList<Double>> Centroids, int iterations)
      {
            ArrayList<ArrayList<Double>> NewCentroids = Centroids;


            // Stores, for each point, which cluster it is in
            int[] cluster_assignment = new int[Data.size()];
```

```
            // Loop for each iteration: assign points, then reassign cluster
centroids
            for(int iter = 0; iter < iterations; iter++)
            {
                  // Loop for each point
                  for(int point = 0; point < Data.size(); point++)
                  {
                        double closest_dist = euclidean_dist(Data.get(point),
NewCentroids.get(0));
                        // System.out.println(closest_dist);
                        int closest_cen = 0;

                        // Loop for remaining centroids
                        for(int cen = 1; cen < NewCentroids.size(); cen++)
                        {
                              double dist =
euclidean_dist(Data.get(point),NewCentroids.get(cen));
                              if (dist < closest_dist)
                              {
                                    closest_dist = dist;
                                    closest_cen = cen;
                              }
                        }
                        cluster_assignment[point] = closest_cen;
                  }
                  //for(int x = 0; x < cluster_assignment.length; x ++)
{ System.out.println(cluster_assignment[x]);}
                  NewCentroids = reassign_centroids(NewCentroids,
Data,cluster_assignment);
            }

            // write the centroids final location
            DecimalFormat df = new DecimalFormat("#.####");
            for(int i = 0; i < NewCentroids.size(); i++)
            {
                  System.out.print("Cluster " + (i+1) + ": ");
                  for(int j = 0; j < NewCentroids.get(i).size(); j++)
                  {
                        System.out.print(df.format(NewCentroids.get(i).get(j)) +
"\t");
                  }
                  System.out.print("\n");
            }

            return cluster_assignment;
      }

      private static ArrayList<ArrayList<Double>>
reassign_centroids(ArrayList<ArrayList<Double>> Centroids,
ArrayList<ArrayList<Double>> data, int[] cluster_assignment)
      {
            ArrayList<ArrayList<Double>> tempCentroids = new
ArrayList<ArrayList<Double>>();

            // Iterate for each centroid
            for (int i = 0; i < Centroids.size(); i++)
            {
                  tempCentroids.add(new ArrayList<Double>());
```

```
                  // Iterate the mean for each axis
                  for (int axis = 0; axis < Centroids.get(0).size(); axis++)
                  {
                        double dist = 0;
                        int count = 0;

                        // To find mean, we sum up the value at that axis for each
point, then divide by number of points
                        for (int point = 0; point < data.size(); point++)
                        {
                              if(cluster_assignment[point] == i)
                              {
                                    count ++;
                                    dist = dist + data.get(point).get(axis);
                              }
                        }
                        tempCentroids.get(i).add(dist/count);
                  }

            }
            return tempCentroids;
      }

      private static double euclidean_dist(ArrayList<Double> point1,
ArrayList<Double> point2)
      {
            double sum = 0.0;

            for (int i = 0; i < point1.size(); i++)
            {
                  sum = sum + Math.pow(point1.get(i)-point2.get(i),2);
                  //System.out.println(point1.get(i)+" " + point2.get(i)+ " " +
sum);
            }
            sum = Math.sqrt(sum);
            return sum;
      }
```