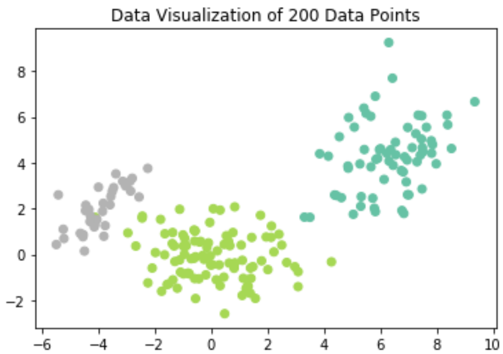
UNIVERSITY OF CALIFORNIA, SANTA BARBARA
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Name: Ashish Vyas
Perm: 6931570
Course: ECE 283 Machine Learning
Homework: 3 Unsupervised Learning

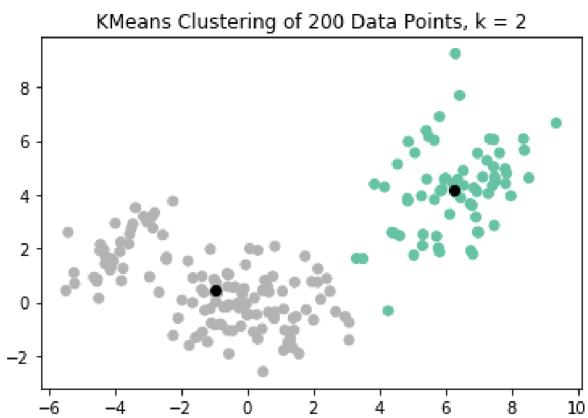
Visualization of Dataset:



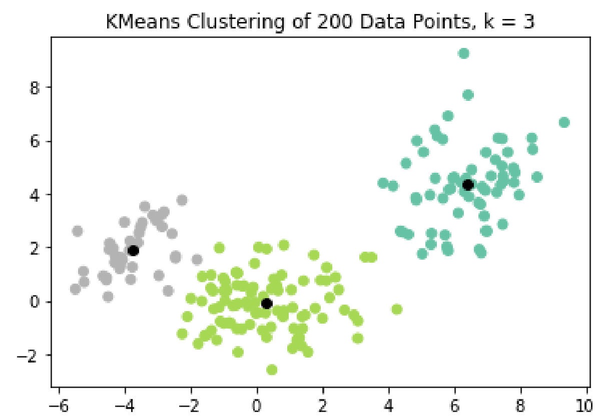
1. K-means Clustering:

Here, Black point denotes cluster centroid. Empirical probabilities are in form:

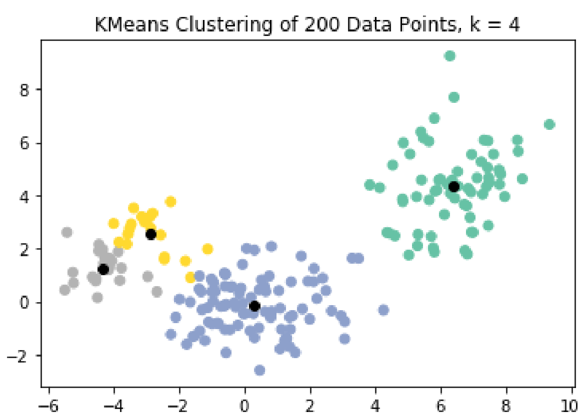
$$P[a_i[k] = 1 \mid z_i[l] = 1], where\ l = 1, 2, 3\ \&\ k = 1, \ldots, K\ in\ form\ of\ 3 \times K\ Table$$



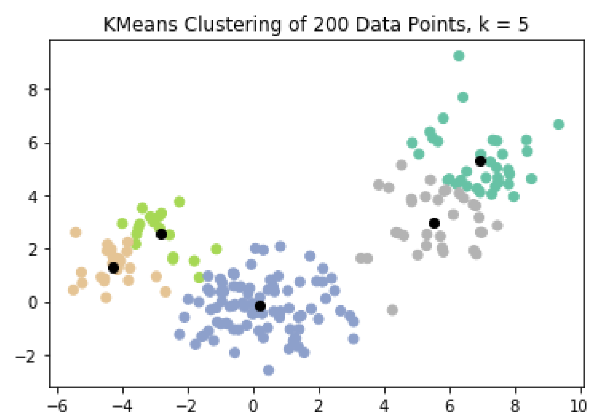

```
Empirical Probabilities :
[[0.335 0.   ]
 [0.005 0.495]
 [0.    0.165]]
```




```
Empirical Probabilities :
[[0.325 0.01  0.   ]
 [0.    0.47  0.03 ]
 [0.    0.    0.165]]
```




```
Empirical Probabilities :
[[0.325 0.01  0.    0.   ]
 [0.    0.46  0.025 0.015]
 [0.    0.    0.07  0.095]]
```

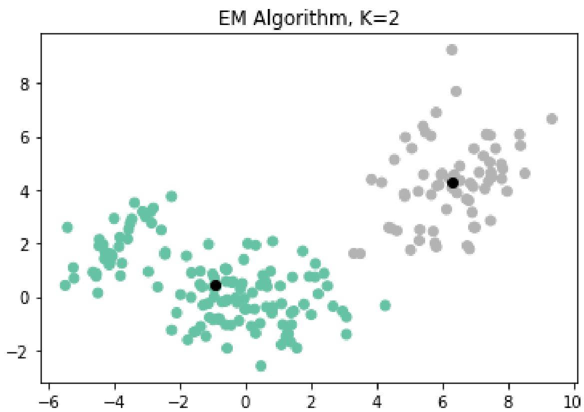



```
Empirical Probabilities :
[[0.175 0.    0.    0.    0.16 ]
 [0.    0.455 0.025 0.015 0.005]
 [0.    0.    0.065 0.1   0.   ]]
```

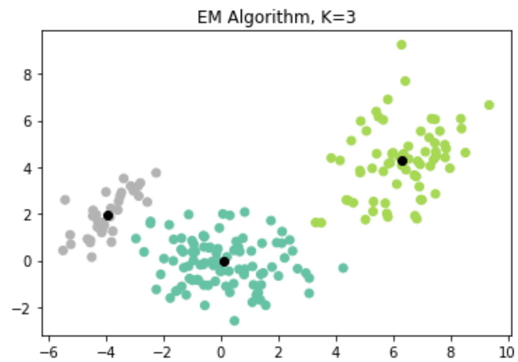
2. EM Algorithm:

Here, Black dot denotes component's mean.
Empirical probabilities are in form:

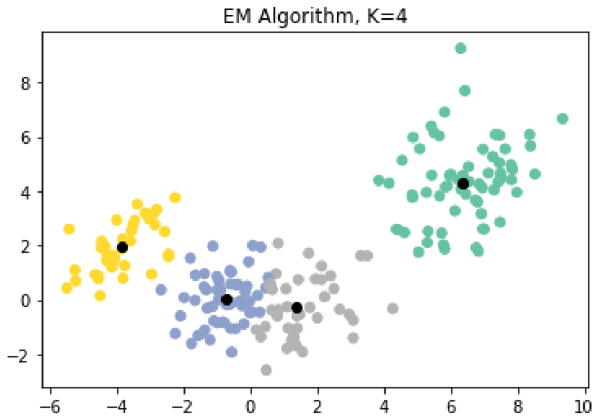
$$P[a_i[k] = 1 \mid z_i[l] = 1], \textit{where } l = 1,2,3 \ \& \ k = 1, \ldots, K \textit{ in form of } 3 \times K \textit{ Table}$$



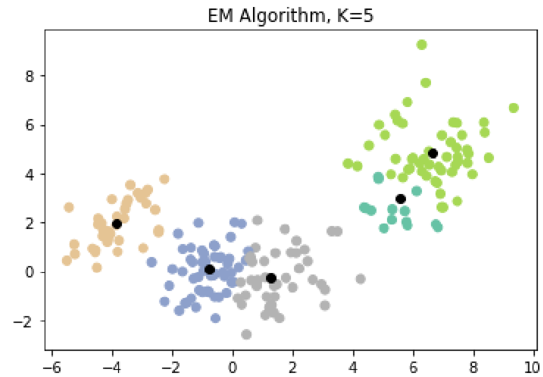

```
Covariances :
[[[ 4.82579724 -1.54350157]
  [-1.54350157  1.77965965]]

 [[ 1.56715373  0.62404714]
  [ 0.62404714  2.38188426]]]

Means :
[[-0.92588033  0.45514894]
 [ 6.31467752  4.27230505]]

Empirical Probabilities :
[[0.    0.335]
 [0.5   0.   ]
 [0.165 0.   ]]
```




```
Covariances :
[[[ 2.15581697 -0.16539493]
  [-0.16539493  1.03173849]]

 [[ 1.53918077  0.60482869]
  [ 0.60482869  2.38605796]]

 [[ 0.64540823  0.54741858]
  [ 0.54741858  0.90562439]]]

Means :
[[ 0.09927901 -0.0570347 ]
 [ 6.32395813  4.27697823]
 [-3.92329695  1.9722266 ]]

Empirical Probabilities :
[[0.    0.335 0.   ]
 [0.495 0.    0.005]
 [0.    0.    0.165]]
```




```
Covariances :
[[[1.48372001 0.54928322]
  [0.54928322 2.35133582]]

 [[0.70572136 0.09402651]
  [0.09402651 0.87583945]]

 [[0.7002795  0.48014602]
  [0.48014602 0.88636819]]

 [[1.19568328 0.16513414]
  [0.16513414 1.12442815]]]

Means :
[[ 6.34637105  4.29695426]
 [-0.74073372  0.05339364]
 [-3.85248034  1.93120201]
 [ 1.36416387 -0.25652136]]

Empirical Probabilities :
[[0.325 0.    0.    0.01 ]
 [0.    0.275 0.02  0.205]
 [0.    0.    0.165 0.   ]]
```




```
Covariances :
[[[1.1495635  0.18224036]
  [0.18224036 1.05414278]]

 [[0.68303906 0.12896576]
  [0.12896576 0.87624201]]

 [[1.29084009 0.0937583 ]
  [0.0937583  1.79537365]]

 [[0.70271842 0.47741525]
  [0.47741525 0.8852116 ]]

 [[1.31088004 0.1794742 ]
  [0.1794742  1.07652871]]]

Means :
[[ 5.60310573  2.98776244]
 [-0.76777876  0.08802133]
 [ 6.66043671  4.85705661]
 [-3.84925192  1.92984783]
 [ 1.28634195 -0.28403762]]

Empirical Probabilities :
[[0.07  0.    0.255 0.    0.01 ]
 [0.    0.27  0.    0.02  0.21 ]
 [0.    0.    0.    0.165 0.   ]]
```

3. Comments

- Centroids from Kmeans & means from EM Algorithm are very similar.
- Empirical probabilities are very similar from both are very similar.
- Covariance from EM algorithm is very similar to original covariance.

E.g. For K=2

```
Centroids :
[[ 6.10880403  4.0451719 ]
 [-0.97444274  0.42973501]]

Empirical Probabilities
[[0.01  0.49 ]
 [0.    0.165]
 [0.33  0.005]]
```

```
Means :
[[ 6.11526374  4.05940127]
 [-0.95617393  0.43347848]]

Empirical Probabilities :
[[0.01  0.49 ]
 [0.    0.165]
 [0.325 0.01 ]]
```

For K = 3

```
Covariances :
[[[ 1.94470452  0.92937185]
  [ 0.92937185  1.93019654]]

 [[ 2.35762202 -0.09127664]
  [-0.09127664  1.08208658]]

 [[ 1.1273601   0.97536036]
  [ 0.97536036  1.11947324]]]
```

```
Cov for comp 1b :
[[2. 1.]
 [1. 2.]]

Cov for comp 0 :
[[2. 0.]
 [0. 1.]]

Cov for comp 1a :
[[1.125 0.875]
 [0.875 1.125]]
```

For k = 2

All points from comp 0 are perfectly mapped onto 2nd gaussian.
All points from comp 1 are mapped onto 1st gaussian.
All points from comp 2 are mapped onto 1st gaussian.

For k = 3

All points from comp 0 are mapped to 2nd gaussian.
Almost all points from comp 1 are mapped to 1st gaussian.
Almost all points from comp 2 are mapped to 3rd gaussian.

For k = 4

Almost all points from comp 0 are mapped to 1st gaussian.
About 55% of comp 1 is mapped to 2nd gaussian & remaining 45% to 4th gaussian.
Almost all points from comp 2 are mapped to 3rd gaussian.

For k = 5

About 70% of comp 0 is mapped to 3rd Gaussian remaining 30% is mapped to 1st gaussian
About 50% of comp 0 has its own gaussian (3)
45% of comp 1 has its own gaussian (4)
55% of comp 1 has its own gaussian (5)
About 40% of comp 2 has its own gaussian (2)

4. Generate a random vector u in d-dimensions. Here, d=30.

```
print(u)
```

```
[[-1.  0.  0.  0.  0.  1.  0.  1.  0.  1.  1.  0.  0. -1. -1.  0.  0.  0.
   1. -1.  0.  0.  0.  0.  1. -1.  0.  0.  0. -1.]
 [ 0.  0.  0.  0.  1.  0. -1.  0. -1.  1.  0.  0.  0.  1.  0.  1.  0.  0.
   0.  0. -1. -1.  1. -1.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  0.  0.  0. -1. -1.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0.
   0.  0.  0. -1.  0.  1.  0.  0.  0.  0.  0.  1.]
 [-1.  0.  0.  1.  0. -1.  0.  0.  0.  1.  0. -1.  0.  0.  1.  0.  0. -1.
   1.  0.  0.  0. -1.  0.  0.  1.  0. -1. -1.  0.]
 [-1.  0. -1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  1.  0.  0.  1. -1.  0.
   0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.
   0.  1.  0.  0.  0.  0.  0.  0.  1.  0. -1.  0.]]
```

```
Correlation Matrix :
[[ 1.       0.       0.      -0.       0.       0.      ]
 [ 0.       1.      -0.      -0.       0.       0.      ]
 [ 0.      -0.       1.      -0.00646 -0.       0.      ]
 [-0.      -0.      -0.00646  1.       0.       0.      ]
 [ 0.       0.      -0.       0.       1.       0.      ]
 [ 0.       0.       0.       0.       0.       1.      ]]
```

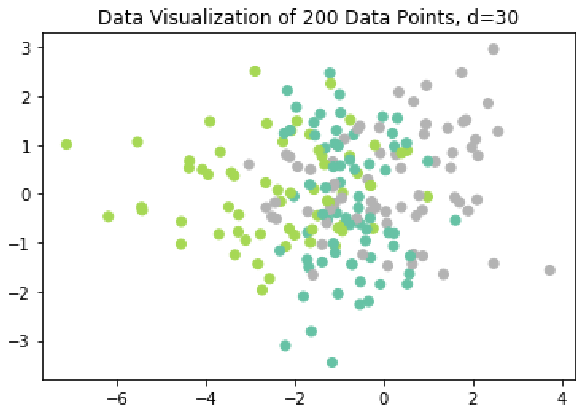
Intuitively, if two vectors are not orthogonal, when we try to use them to generate data, the data points from one component might overlap into another component. Therefore, all u vectors must be perpendicular to each other in d-dimension.

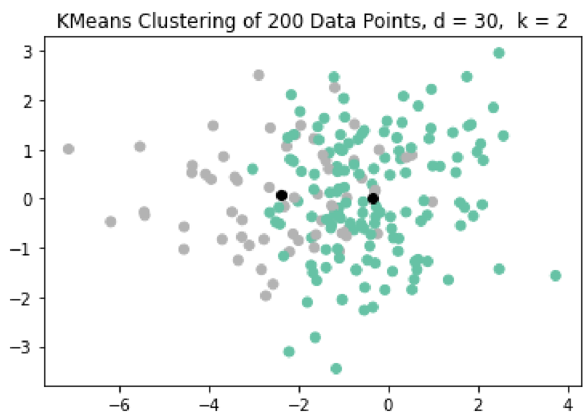
5. Generate d-dimensional data samples from a Gaussian mixture distribution. Shown below is sample from generated data. Here, d=30.

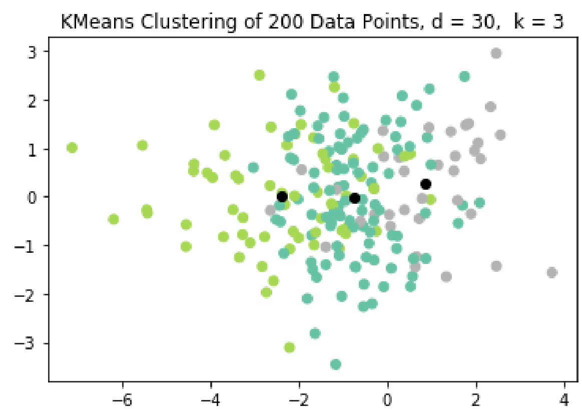
```
print("Data Shape = {}\n".format(data.shape))
print("Data Sample = \n{}\n".format(data[0]))
print("Data Sample's Component = {}".format(labels[0].astype(int)))
```

```
Data Shape = (200, 30)

Data Sample =
[-0.97645341  1.04263334 -1.58472112  1.34251756  1.85453602  3.08497076
 -1.06916342  0.4226387   1.69190744  2.50062806 -2.15223525  3.03171095
 -0.9365996   0.96383224 -2.80381196  1.82664518 -2.49819429 -1.21037025
  3.31729985 -3.63334278  0.78114528 -2.63254319 -1.07553977  0.78963058
  1.97108466 -0.84351933 -1.34037551 -0.38686164  0.71996132 -2.11019838]

Data Sample's Component = 2
```

6. K-means on d-dimensional data, d=30
Generated N = 200 data points. Plotting first 2 dimensions as x & y coordinates.

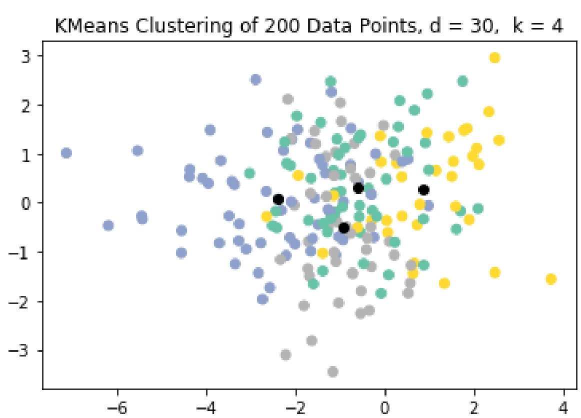


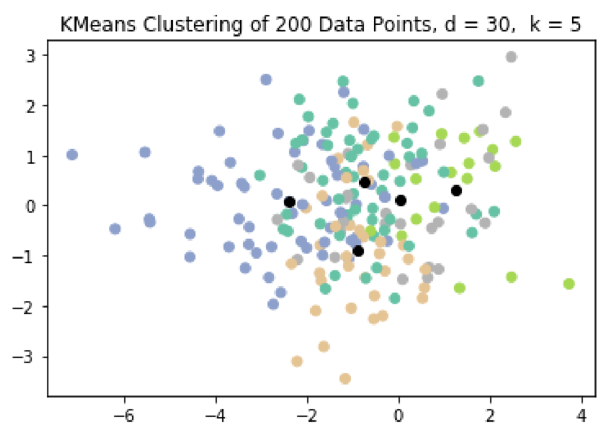





```
Empirical Probabilities :
[[0.34  0.   ]
 [0.    0.315]
 [0.345 0.   ]]
```

```
Empirical Probabilities :
[[0.335 0.005 0.   ]
 [0.    0.315 0.   ]
 [0.17  0.    0.175]]
```






```
Empirical Probabilities :
[[0.105 0.    0.    0.235]
 [0.    0.315 0.    0.   ]
 [0.17  0.    0.175 0.   ]]
```

```
Empirical Probabilities :
[[0.15  0.    0.    0.18  0.01 ]
 [0.    0.31  0.    0.    0.005]
 [0.125 0.    0.095 0.    0.125]]
```

7. Geometric Insight

For k = 2

All points from comp 0 are perfectly mapped onto 1^{st} cluster.
Almost all points from comp 1 are mapped onto 2^{nd} cluster.
All points from comp 2 are mapped onto 1^{st} cluster.

Since comp 2 is a combination of comp 0 & comp 1 & since it has a strong influence of ($u_1 + u_2$), it is mapped to cluster 1 where comp 0 is mapped, which also has a strong influence of ($u_1 + u_2$).

For k = 3

Almost all points from comp 0 are mapped to 1^{st} cluster.
All points from comp 1 are perfectly mapped to 2^{nd} cluster.
50% of points from comp 2 are mapped to 1^{st} cluster & 50% to 3^{rd} cluster.

We can see comp 2 being divided into two clusters where comp 0 & comp 1 are mapped to.

For k = 4

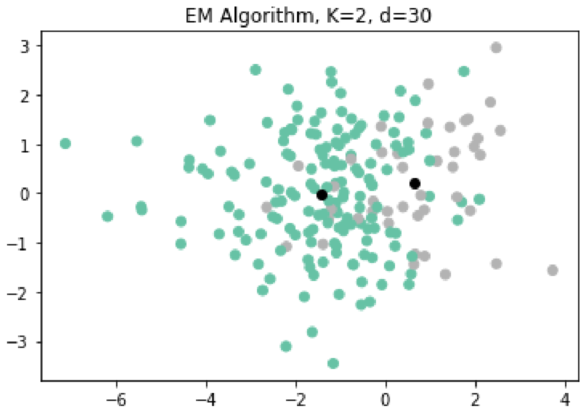
70% of comp 0 has its own cluster 4.
30% of comp 0 is mapped to 1^{st} cluster
All points from comp 1 has its own cluster 2.
About 50% of comp 2 is mapped to 1^{st} cluster & remaining 50% has its own cluster 3.

Similarly, points from comp 0 & comp 2 with strong influence of ($u_1 + u_2$) are mapped to the same cluster 1.

For k = 5

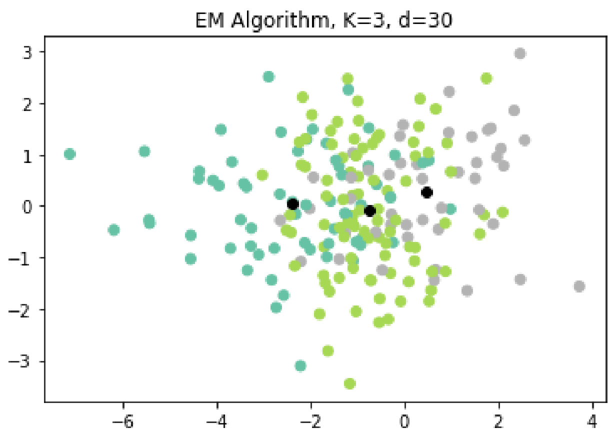
37% of comp 2 is mapped to 1^{st} cluster along with about 50% remaining from comp 0.
Almost entire comp 1 has its own cluster 2.
Remaining 25% of comp 2 has its own cluster 3.
About 50% of comp 0 has its own cluster 4.
37 % of comp 2 has its own cluster 5.

8. EM Algorithm on d-dimensional data, d=30




```
Empirical Probabilities :
[[0.33  0.01 ]
 [0.31  0.005]
 [0.145 0.2  ]]
```

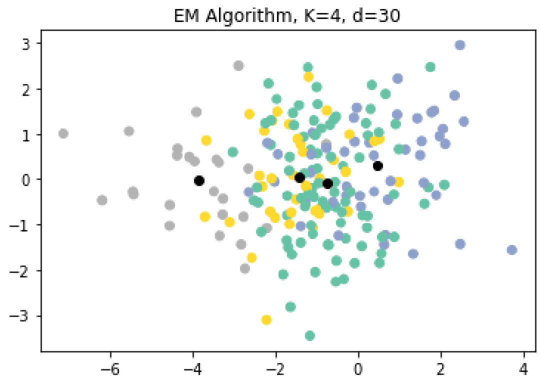
```
Covariances :
[[[ 2.801  0.096  2.001 ... -1.838 -0.073  0.433]
  [ 0.096  0.892  0.098 ... -0.248 -0.096  0.281]
  [ 2.001  0.098  3.452 ... -2.224  0.201 -0.006]
  ...
  [-1.838 -0.248 -2.224 ...  2.921  0.093 -0.396]
  [-0.073 -0.096  0.201 ...  0.093  1.436 -0.603]
  [ 0.433  0.281 -0.006 ... -0.396 -0.603  1.388]]

 [[ 1.751  0.241  0.184 ... -0.28  -0.248  0.674]
  [ 0.241  1.556  0.143 ... -0.185 -0.232 -0.507]
  [ 0.184  0.143  1.158 ... -0.17  -0.042 -0.072]
  ...
  [-0.28  -0.185 -0.17  ...  1.282  0.179 -0.127]
  [-0.248 -0.232 -0.042 ...  0.179  1.591 -0.356]
  [ 0.674 -0.507 -0.072 ... -0.127 -0.356  2.287]]]

Means :
[[-2.395  0.072 -0.336  1.906  0.21  -1.906  0.143  0.006 -0.069  2.223
  -0.453 -1.967  0.164 -0.111  2.021  0.319 -0.257 -1.954  1.899 -0.049
   0.305  0.263 -2.13   0.121 -0.075  1.869  0.066 -1.861 -1.985 -0.103]
 [-0.338  0.023  0.004 -0.714  0.558  0.342  0.064  0.552  0.45   0.269
   0.336  0.072  0.11  -1.356 -0.318 -0.185 -0.094  0.044  0.396  0.207
   0.244  0.044 -0.234  0.264  0.553 -0.387  0.764 -0.05  -0.72  -0.365]]
```




```
Empirical Probabilities :
[[0.005 0.3   0.035]
 [0.31  0.    0.005]
 [0.    0.155 0.19 ]]
```

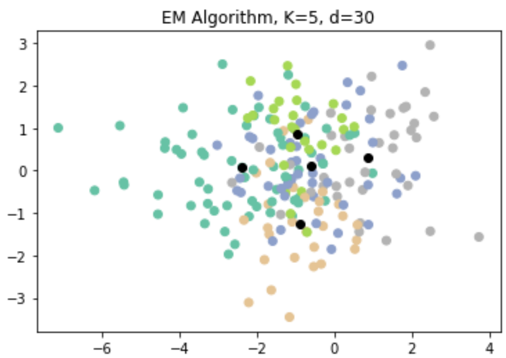
```
Covariances:
[[[ 2.801e+00  9.000e-02  1.996e+00 ... -1.845e+00 -7.000e-02  4.420e-01]
  [ 9.000e-02  1.030e+00  1.250e-01 ... -1.840e-01 -1.370e-01  1.290e-01]
  [ 1.996e+00  1.250e-01  3.388e+00 ... -2.287e+00  2.260e-01  2.200e-02]
  ...
  [-1.845e+00 -1.840e-01 -2.287e+00 ...  2.870e+00  1.100e-01 -4.050e-01]
  [-7.000e-02 -1.370e-01  2.260e-01 ...  1.100e-01  1.432e+00 -5.860e-01]
  [ 4.420e-01  1.290e-01  2.200e-02 ... -4.050e-01 -5.860e-01  1.508e+00]]

 [[ 1.073e+00 -1.000e-02  2.590e-01 ... -2.500e-01 -4.000e-03  3.300e-02]
  [-1.000e-02  1.630e+00  2.200e-01 ... -2.980e-01 -2.850e-01 -7.600e-01]
  [ 2.590e-01  2.200e-01  1.038e+00 ... -1.630e-01 -1.530e-01 -1.800e-02]
  ...
  [-2.500e-01 -2.980e-01 -1.630e-01 ...  1.111e+00  2.980e-01  1.210e-01]
  [-4.000e-03 -2.850e-01 -1.530e-01 ...  2.980e-01  1.575e+00 -6.000e-03]
  [ 3.300e-02 -7.600e-01 -1.800e-02 ...  1.210e-01 -6.000e-03  1.651e+00]]

 [[ 2.086e+00  3.580e-01  2.240e-01 ... -3.350e-01 -2.770e-01  1.156e+00]
  [ 3.580e-01  1.136e+00  1.600e-02 ... -9.700e-02 -1.000e-02 -6.100e-02]
  [ 2.240e-01  1.600e-02  1.386e+00 ... -2.190e-01 -3.000e-03  6.300e-02]
  ...
  [-3.350e-01 -9.700e-02 -2.190e-01 ...  1.482e+00 -1.390e-01 -3.550e-01]
  [-2.770e-01 -1.000e-02 -3.000e-03 ... -1.390e-01  1.536e+00 -6.150e-01]
  [ 1.156e+00 -6.100e-02  6.300e-02 ... -3.550e-01 -6.150e-01  2.484e+00]]]
```

```
Means :
[[-2.396  0.04  -0.365  1.937  0.16  -1.938  0.139  0.018 -0.03   2.204
  -0.376 -1.972  0.169 -0.104  1.951  0.314 -0.222 -1.95   1.881 -0.099
   0.316  0.241 -2.159  0.144 -0.046  1.83   0.026 -1.9   -1.965 -0.049]
 [-0.751 -0.084  0.125 -0.455  0.776  0.801 -0.457  1.097 -0.004  1.174
   0.821  0.02   0.046 -1.16  -0.939  0.244 -0.126  0.033  0.799 -0.614
  -0.166 -0.47   0.256 -0.179  1.023 -0.85   0.499  0.012 -0.564 -0.77 ]
 [ 0.478  0.278 -0.196 -1.269  0.195 -0.522  1.101 -0.543  1.295 -1.494
  -0.729  0.181  0.23  -1.753  1.006 -1.026 -0.079  0.061 -0.376  1.899
   1.042  1.093 -1.165  1.108 -0.416  0.582  1.344 -0.118 -1.058  0.362]]
```




```
Empirical Probabilities :
[[0.3   0.035 0.005 0.    ]
 [0.    0.    0.19  0.125]
 [0.15  0.195 0.    0.    ]]
```

```
Covariances :
[[[ 2.845e+00  1.010e-01  2.027e+00 ... -1.873e+00 -7.200e-02  4.410e-01]
  [ 1.010e-01  8.850e-01  1.380e-01 ... -2.170e-01 -1.110e-01  2.750e-01]
  [ 2.027e+00  1.380e-01  3.442e+00 ... -2.322e+00  2.280e-01  1.300e-02]
  ...
  [-1.873e+00 -2.170e-01 -2.322e+00 ...  2.910e+00  1.160e-01 -3.850e-01]
  [-7.200e-02 -1.110e-01  2.280e-01 ...  1.160e-01  1.450e+00 -6.190e-01]
  [ 4.410e-01  2.750e-01  1.300e-02 ... -3.850e-01 -6.190e-01  1.406e+00]]

  ...

 [[ 5.150e-01  4.300e-02 -3.800e-02 ...  3.000e-03  5.900e-02 -1.610e-01]
  [ 4.300e-02  1.913e+00 -9.000e-03 ... -8.400e-02 -2.970e-01 -1.405e+00]
  [-3.800e-02 -9.000e-03  1.261e+00 ...  5.400e-02 -3.700e-01  4.680e-01]
  ...
  [ 3.000e-03 -8.400e-02  5.400e-02 ...  1.158e+00  2.840e-01 -1.580e-01]
  [ 5.900e-02 -2.970e-01 -3.700e-01 ...  2.840e-01  1.028e+00  1.700e-02]
  [-1.610e-01 -1.405e+00  4.680e-01 ... -1.580e-01  1.700e-02  2.391e+00]]]

Means :
[[-2.398e+00  9.100e-02 -3.680e-01  1.956e+00  1.940e-01 -1.936e+00
   1.460e-01 -2.000e-03 -7.700e-02  2.244e+00 -4.200e-01 -1.965e+00
   1.490e-01 -6.400e-02  2.024e+00  3.250e-01 -2.250e-01 -1.966e+00
   1.907e+00 -9.400e-02  3.010e-01  2.590e-01 -2.132e+00  1.190e-01
  -7.500e-02  1.861e+00  3.500e-02 -1.891e+00 -1.974e+00 -9.400e-02]
 [ 7.670e-01  2.680e-01 -1.290e-01 -1.534e+00  5.710e-01 -9.690e-01
   9.030e-01 -6.330e-01  1.303e+00 -1.639e+00 -9.550e-01  2.180e-01
   1.930e-01 -1.589e+00  1.211e+00 -9.560e-01 -9.400e-02  2.000e-02
  -5.850e-01  2.418e+00  1.059e+00  8.290e-01 -1.001e+00  9.940e-01
  -7.580e-01  9.100e-01  1.480e+00 -1.690e-01 -1.276e+00  9.050e-01]
 [-6.230e-01 -5.500e-02  1.500e-01 -8.210e-01  1.419e+00  8.450e-01
  -1.045e+00  1.109e+00 -4.490e-01  1.524e+00  8.150e-01  1.520e-01
   5.100e-02 -9.580e-01 -8.760e-01  7.110e-01 -2.420e-01  2.200e-02
   8.630e-01 -3.220e-01 -5.850e-01 -8.870e-01  8.290e-01 -6.760e-01
   1.052e+00 -7.890e-01  7.960e-01 -3.300e-02 -8.640e-01 -7.700e-01]
 [-1.040e+00 -1.260e-01 -6.300e-02  2.980e-01 -9.040e-01  8.210e-01
   1.089e+00  8.190e-01  1.105e+00  1.100e-01  7.700e-01 -2.750e-01
   1.510e-01 -1.838e+00 -8.800e-01 -9.150e-01  1.000e-01  7.300e-02
   6.370e-01 -1.094e+00  8.290e-01  8.380e-01 -1.306e+00  1.114e+00
   1.031e+00 -9.600e-01  8.000e-03  4.400e-02  4.200e-02 -9.890e-01]]

Empirical Probabilities :
[[0.    0.    0.155 0.185]
 [0.31  0.005 0.    0.   ]
 [0.    0.185 0.16  0.   ]]
```




```
 Empirical Probabilities :
 [[0.    0.065 0.13  0.145 0.   ]
  [0.315 0.    0.    0.    0.   ]
  [0.    0.16  0.    0.    0.185]]
Covariances :
[[[ 1.883e+00 -2.220e-01  7.360e-01 ... -6.380e-01 -2.000e-01  2.090e-01]
  [-2.220e-01  9.490e-01 -1.660e-01 ... -1.210e-01  1.170e-01  2.080e-01]
  [ 7.360e-01 -1.660e-01  2.539e+00 ... -1.274e+00  3.200e-02 -3.530e-01]
  ...
  [-6.380e-01 -1.210e-01 -1.274e+00 ...  2.173e+00 -1.120e-01  7.900e-02]
  [-2.000e-01  1.170e-01  3.200e-02 ... -1.120e-01  1.605e+00 -8.600e-01]
  [ 2.090e-01  2.080e-01 -3.530e-01 ...  7.900e-02 -8.600e-01  1.273e+00]]

 [[ 1.267e+00 -1.300e-02  3.320e-01 ... -4.300e-01  3.400e-02  7.000e-03]
  [-1.300e-02  1.161e+00  1.760e-01 ... -3.330e-01  1.250e-01 -2.420e-01]
  [ 3.320e-01  1.760e-01  8.970e-01 ... -1.860e-01 -6.700e-02  3.500e-02]
  ...
[[ 6.420e-01 -1.800e-02  1.460e-01 ...  2.140e-01 -5.600e-02  1.240e-01]
  [-1.800e-02  1.035e+00  3.050e-01 ...  2.250e-01  8.100e-02 -5.510e-01]
  [ 1.460e-01  3.050e-01  1.251e+00 ... -1.100e-01 -3.520e-01 -4.000e-02]
  ...
  [ 2.140e-01  2.250e-01 -1.100e-01 ...  1.059e+00  1.980e-01 -5.310e-01]
  [-5.600e-02  8.100e-02 -3.520e-01 ...  1.980e-01  9.150e-01 -2.440e-01]
  [ 1.240e-01 -5.510e-01 -4.000e-02 ... -5.310e-01 -2.440e-01  1.341e+00]]]

Means :
[[-3.773e+00 -1.400e-02 -1.813e+00  1.976e+00  2.120e-01 -2.024e+00
   4.400e-02  1.500e-02  2.000e-03  2.272e+00 -1.730e+00 -1.858e+00
   1.363e+00 -1.930e-01  2.227e+00  1.500e+00 -1.614e+00 -2.198e+00
   2.015e+00  1.940e-01  1.657e+00  1.670e-01 -1.900e+00  2.840e-01
  -3.380e-01  1.698e+00  4.300e-02 -6.240e-01 -1.995e+00 -4.030e-01]
 [-6.790e-01  4.330e-01  1.360e-01 -7.810e-01  1.333e+00  1.283e+00
  -2.500e-01  4.970e-01 -6.520e-01  1.258e+00  7.650e-01  1.300e-02
   8.800e-02 -1.027e+00 -3.640e-01  4.480e-01 -1.550e-01  1.500e-02
   8.590e-01 -4.290e-01 -2.870e-01 -3.720e-01  4.710e-01 -8.190e-01
   1.026e+00 -9.330e-01  8.680e-01 -1.170e-01 -8.940e-01 -1.164e+00]
 [ 6.810e-01  2.950e-01 -2.510e-01 -1.366e+00  2.310e-01 -7.170e-01
   1.092e+00 -5.760e-01  1.337e+00 -1.705e+00 -7.700e-01  2.600e-01
   1.990e-01 -1.704e+00  1.236e+00 -1.040e+00 -5.400e-02  1.060e-01
  -5.840e-01  2.189e+00  1.091e+00  1.005e+00 -1.081e+00  1.127e+00
  -5.860e-01  7.430e-01  1.382e+00 -1.030e-01 -1.091e+00  5.920e-01]
 [-1.362e+00  1.370e-01  7.710e-01  1.853e+00  2.090e-01 -1.818e+00
   2.160e-01 -1.000e-03 -1.210e-01  2.186e+00  5.040e-01 -2.050e+00
  -7.350e-01 -5.000e-02  1.867e+00 -5.660e-01  7.610e-01 -1.771e+00
   1.812e+00 -2.310e-01 -7.080e-01  3.350e-01 -2.302e+00 -1.000e-03
   1.220e-01  1.998e+00  8.400e-02 -2.788e+00 -1.978e+00  1.220e-01]
 [-9.270e-01 -1.200e+00  5.200e-02  2.700e-01 -6.710e-01 -2.960e-01
  -5.930e-01  2.124e+00  1.652e+00  7.130e-01  8.500e-01 -4.700e-02
   4.100e-02 -1.607e+00 -2.224e+00 -4.310e-01 -1.600e-02  2.700e-02
   6.750e-01 -9.960e-01  2.830e-01 -3.090e-01 -6.430e-01  1.455e+00
   1.014e+00 -6.820e-01 -2.540e-01  1.600e-01  1.290e-01  1.000e-01]]
```

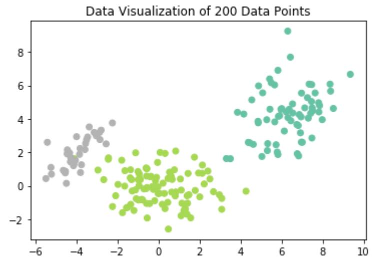
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.colors import ListedColormap
from sklearn.metrics import accuracy_score
```

In [33]:

```python
# Cov matrix
def cov(lam1, lam2, theta):
    d = np.matrix([[lam1, 0], [0, lam2]])
    p = np.matrix([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
    invp = np.linalg.inv(p)
    return np.linalg.multi_dot((p,d,invp))

# Generating Data
def gendata(data_pts, seed_no=30):

    np.random.seed(seed_no)
    data_pts = data_pts/2

    # Class 0
    cov0 = cov(2, 1, 0)
    mean0 = [0,0]

    data0 = np.random.multivariate_normal(mean0, cov0, data_pts)

    # Class 1

    cov1a = cov(2, 0.25, -3*np.pi/4)
    mean1a = [-4,2]
    cov1b = cov(3, 1, np.pi/4)
    mean1b = [6,4]

    data1a = np.random.multivariate_normal(mean1a, cov1a, (data_pts/3))
    data1b = np.random.multivariate_normal(mean1b, cov1b, (data_pts - data_pts/3))
    data1 = np.concatenate((data1a, data1b), axis=0)

    data = np.concatenate((data0,data1), axis = 0)
    labels = np.concatenate((np.ones(data_pts), np.array([2]*(data_pts/3)), np.zeros(data_pts - data_pts/3)))

    return data,labels

def compute_KMeans(k,data):

    def update_labels(data, centroids):
        l = []
        for i in range(data.shape[0]):
            mse = []
            for j in range(centroids.shape[0]):
                mse.append(np.sqrt(np.sum(np.square(data[i] - centroids[j]))))
            l.append(np.argmin(mse))
        return l

    def new_centroids(data, centroids, new_labels):

        indices = []
        for j in range(centroids.shape[0]):
            indices.append([i for i, x in enumerate(new_labels) if x == j])

        new_centroids = []
        for i in range(len(indices)):
            new_centroids.append(np.sum(data[indices[i]], axis=0)/ len(indices[i]))

        return np.array(new_centroids)

    centroids = data[np.random.choice(data.shape[0], k, replace=False), :]

    old_labels = None
    new_labels = []
    while(new_labels != old_labels):
        old_labels = new_labels
        new_labels = update_labels(data, centroids)
        centroids = new_centroids(data, centroids, new_labels)

    return np.array(new_labels), centroids.T

def fit_EM(X, k = 3, eps = 0.0001, max_iters = 1000):

    n, d = X.shape
    mu = X[np.random.choice(n, k, False), :]
    Sigma= [np.eye(d)] * k
    w = [1./k] * k
    R = np.zeros((n, k))


    log_likelihoods = []

    P = lambda mu, s: np.linalg.det(s) ** -.5 ** (2 * np.pi) ** (-X.shape[1]/2.) \
            * np.exp(-.5 * np.einsum('ij, ij -> i',\
                    X - mu, np.dot(np.linalg.inv(s) , (X - mu).T).T ) )

    while len(log_likelihoods) < max_iters:

        # E - Step
        for i in range(k):
            R[:, i] = w[i] * P(mu[i], Sigma[i])
        log_likelihood = np.sum(np.log(np.sum(R, axis = 1)))
        log_likelihoods.append(log_likelihood)
        R = (R.T / np.sum(R, axis = 1)).T
        N_ks = np.sum(R, axis = 0)


        # M Step
        for i in range(k):
            mu[i] = 1. / N_ks[i] * np.sum(R[:, i] * X.T, axis = 1).T
            x_mu = np.matrix(X - mu[i])
            Sigma[i] = np.array(1 / N_ks[i] * np.dot(np.multiply(x_mu.T,  R[:, i]), x_mu))
            w[i] = 1. / n * N_ks[i]

        # check for convergence
        if len(log_likelihoods) < 2 : continue
        if np.abs(log_likelihood - log_likelihoods[-2]) < eps: break

    return mu, np.array(Sigma)

def emp_prob(L, K, k):
    mat = np.zeros((3,k), dtype='float')
    for i in range(3):
        for j in range(k):
            s = 0
            for m in range(len(L)):
```

```
                if (L[m] == i and K[m] == j):
                    s = s+1
            mat[i,j] = s/float(len(L))
    return mat
```

In [3]:
```
N = 200
data, true_labels = gendata(N)
plt.scatter(data.T[0], data.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Data Visualization of {} Data Points".format(N))
plt.show()
```



In [4]:
```
k=2
pred_labels, centroids = compute_KMeans(k,data)
plt.scatter(data.T[0], data.T[1], c=pred_labels, cmap=cm.Set2)
plt.plot(centroids[0], centroids[1], '.', markersize=12, color = 'black')
plt.title("KMeans Clustering of {} Data Points, k = {}".format(N,k))
plt.show()
print("Empirical Probabilities : \n{}".format(emp_prob(true_labels, pred_labels, k)))
```


```
Empirical Probabilities :
[[0.335 0.   ]
 [0.005 0.495]
 [0.    0.165]]
```

In [6]:
```
d = 30
N = 400
m = 40
seed_no = 37

np.random.seed(seed_no)

#Generating Quasi-Orthogonal U
u = np.zeros((6,d))
for i in range(6):
    u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
    j = 0
    while (j<i):
        coeff = np.corrcoef(u[i],u[j])[0,1]
        if (coeff < -0.01 or coeff > 0.0):
            u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
            j = 0
        else:
            j += 1
```

In [7]:
```
print(u)
```

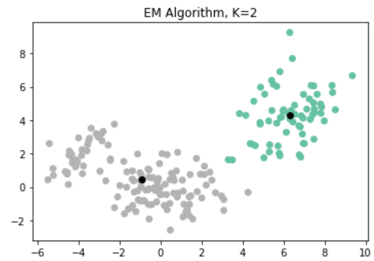
```
[[-1.  0.  0.  0.  0.  1.  0.  1.  0.  1.  1.  0.  0. -1. -1.  0.  0.  0.
   1. -1.  0.  0.  0.  0.  1. -1.  0.  0.  0. -1.]
 [ 0.  0.  0.  0.  1.  0. -1.  0. -1.  1.  0.  0.  0.  1.  0.  1.  0.  0.
   0.  0. -1. -1.  1. -1.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  0.  0.  0. -1. -1.  1.  1.  0.  0.  0.  0.  0. -1.  0.  0.  0.
   0.  0.  0. -1.  0.  1.  0.  0.  0.  0.  0.  1.]
 [-1.  0.  0.  1.  0. -1.  0.  0.  0.  1.  0. -1.  0.  0.  1.  0.  0. -1.
   1.  0.  0.  0. -1.  0.  0.  1.  0. -1. -1.  0.]
 [-1.  0. -1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  1.  0.  0.  1. -1.  0.
   0.  0.  1.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
 [ 0.  0.  0. -1.  1.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  0.
   0.  1.  0.  0.  0.  0.  0.  0.  1.  0. -1.  0.]]
```

In [8]:
```
z = np.zeros((6,6))
for i in range(6):
    for j in range(6):
        z[i,j] = round(np.corrcoef(u[i],u[j])[0,1], 5)
```

In [9]:
```
print("Correlation Matrix : \n{}".format(z))
```

```
Correlation Matrix :
[[ 1.       0.       0.      -0.       0.       0.     ]
 [ 0.       1.      -0.      -0.       0.       0.     ]
 [ 0.      -0.       1.      -0.00646 -0.       0.     ]
 [-0.      -0.      -0.00646  1.       0.       0.     ]
 [ 0.       0.      -0.       0.       1.       0.     ]
 [ 0.       0.       0.       0.       0.       1.     ]]
```

In [37]:
```
k=2
m, s = fit_EM(data, k, eps = 0.0001, max_iters = 1000)
plt.title("EM Algorithm, K={}".format(k))
plt.scatter(data.T[0], data.T[1], c=pred, cmap=cm.Set2)
plt.plot(m[:,0], m[:,1], '.', markersize=12, color = 'black')
plt.show()
print("Covariances : \n{}\n".format(s))
print("Means : \n{}".format(m))
print("\nEmpirical Probabilities :")
print(emp_prob(labels, pred, k))
```




```
Covariances :
[[[ 1.56715373  0.62404714]
  [ 0.62404714  2.38188426]]

 [[ 4.82579724 -1.54350157]
  [-1.54350157  1.77965965]]]

Means :
[[ 6.31467752  4.27230505]
 [-0.92588033  0.45514894]]

Empirical Probabilities :
[[0.335 0.   ]
 [0.    0.5  ]
 [0.    0.165]]
```

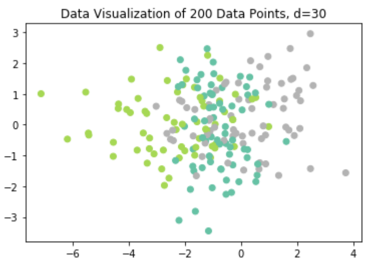
In [ ]:

In [2]:
```python
# Cov matrix
def cov(lam1, lam2, theta):
    d = np.matrix([[lam1, 0], [0, lam2]])
    p = np.matrix([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
    invp = np.linalg.inv(p)
    return np.linalg.multi_dot((p,d,invp))

def gen_u(d):

    np.random.seed(37)

    #Generating Quasi-Orthogonal U
    u = np.zeros((6,d))
    for i in range(6):
        u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
        j = 0
        while (j<i):
            coeff = np.corrcoef(u[i],u[j])[0,1]
            if (coeff < -0.01 or coeff > 0.0):
                u[i] = np.random.choice([0,1,-1], d, p=[0.666, 0.167, 0.167])
                j = 0
            else:
                j += 1
    return u

def gen_d_data(data_pts, d):

    u = gen_u(d)

    data = np.zeros((data_pts,d))
    labels = np.zeros((data_pts))

    for i in range(data_pts):
        c = np.random.randint(3, size=1)[0]
        if(c==0):
            data[i] = u[0] + np.random.normal(loc=0, scale=1)*u[1] +  np.random.normal(loc=0, scale=1)*u[2] +  np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 0
        elif(c==1):
            data[i] = 2*u[3] + np.sqrt(2)*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1)*u[5] + np.random.normal(loc=0, scale=1, size=d)
            labels[i] = 1
        else:
            data[i] = np.sqrt(2)*u[5] + np.random.normal(loc=0, scale=1)*(u[0]+u[1]) + (1/np.sqrt(2))*np.random.normal(loc=0, scale=1)*u[4] + np.random.normal(loc=0, scale=1,
size=d)
            labels[i] = 2

    return data,labels,u

def compute_KMeans(k,data):

    def update_labels(data, centroids):
        l = []
        for i in range(data.shape[0]):
            mse = []
            for j in range(centroids.shape[0]):
                mse.append(np.sqrt(np.sum(np.square(data[i] - centroids[j]))))
            l.append(np.argmin(mse))
        return l

    def new_centroids(data, centroids, new_labels):

        indices = []
        for j in range(centroids.shape[0]):
            indices.append([i for i, x in enumerate(new_labels) if x == j])

        new_centroids = []
        for i in range(len(indices)):
            new_centroids.append(np.sum(data[indices[i]], axis=0)/ len(indices[i]))

        return np.array(new_centroids)

    centroids = data[np.random.choice(data.shape[0], k, replace=False), :]

    old_labels = None
    new_labels = []
    while(new_labels != old_labels):
        old_labels = new_labels
        new_labels = update_labels(data, centroids)
        centroids = new_centroids(data, centroids, new_labels)

    return np.array(new_labels), centroids.T

def fit_EM(X, k = 3, eps = 0.0001, max_iters = 1000):

    n, d = X.shape
    mu = X[np.random.choice(n, k, False), :]
    Sigma= [np.eye(d)] * k
    w = [1./k] * k
    R = np.zeros((n, k))


    log_likelihoods = []

    P = lambda mu, s: np.linalg.det(s) ** -.5 ** (2 * np.pi) ** (-X.shape[1]/2.) \
            * np.exp(-.5 * np.einsum('ij, ij -> i',\
                    X - mu, np.dot(np.linalg.inv(s) , (X - mu).T).T ) )

    while len(log_likelihoods) < max_iters:

        # E - Step
        for i in range(k):
            R[:, i] = w[i] * P(mu[i], Sigma[i])
        log_likelihood = np.sum(np.log(np.sum(R, axis = 1)))
        log_likelihoods.append(log_likelihood)
        R = (R.T / np.sum(R, axis = 1)).T
        N_ks = np.sum(R, axis = 0)


        # M Step
        for i in range(k):
            mu[i] = 1. / N_ks[i] * np.sum(R[:, i] * X.T, axis = 1).T
            x_mu = np.matrix(X - mu[i])
            Sigma[i] = np.array(1 / N_ks[i] * np.dot(np.multiply(x_mu.T,  R[:, i]), x_mu))
            w[i] = 1. / n * N_ks[i]
```

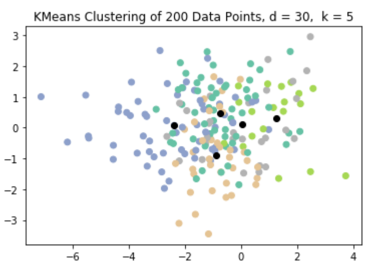
```
        # check for convergence
        if len(log_likelihoods) < 2 : continue
        if np.abs(log_likelihood - log_likelihoods[-2]) < eps: break

    return mu, np.array(Sigma)

def emp_prob(L, K, k):
    mat = np.zeros((3,k), dtype='float')
    for i in range(3):
        for j in range(k):
            s = 0
            for m in range(len(L)):
                if (L[m] == i and K[m] == j):
                    s = s+1
            mat[i,j] = s/float(len(L))
    return mat
```

In [3]:
```
N = 200
d = 30
data, true_labels, u = gen_d_data(N, d)
```

In [4]:
```
plt.scatter(data.T[0], data.T[1], c=true_labels, cmap=cm.Set2)
plt.title("Data Visualization of {} Data Points, d={}".format(N,d))
plt.show()
```



In [5]:
```
k=5
pred_labels, centroids = compute_KMeans(k,data)
plt.scatter(data.T[0], data.T[1], c=pred_labels, cmap=cm.Set2)
plt.plot(centroids[0], centroids[1], '.', markersize=12, color = 'black')
plt.title("KMeans Clustering of {} Data Points, d = {},  k = {}".format(N,d,k))
plt.show()
print("Empirical Probabilities : \n{}".format(emp_prob(true_labels, pred_labels, k)))
```

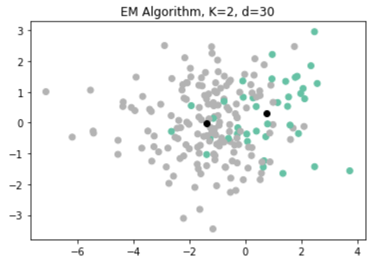



```
Empirical Probabilities :
[[0.15  0.    0.    0.18  0.01 ]
 [0.    0.31  0.    0.    0.005]
 [0.125 0.    0.095 0.    0.125]]
```

In [6]:
```
z = np.zeros((6,6))
for i in range(6):
    for j in range(6):
        z[i,j] = round(np.corrcoef(u[i],u[j])[0,1], 5)

print("Correlation Matrix : \n{}".format(z))
```

```
Correlation Matrix :
[[ 1.       0.       0.      -0.       0.       0.     ]
 [ 0.       1.      -0.      -0.       0.       0.     ]
 [ 0.      -0.       1.      -0.00646 -0.       0.     ]
 [-0.      -0.      -0.00646  1.       0.       0.     ]
 [ 0.       0.      -0.       0.       1.       0.     ]
 [ 0.       0.       0.       0.       0.       1.     ]]
```

In [37]:
```
k=2
m, s = fit_EM(data, k, eps = 0.0001, max_iters = 1000)
plt.title("EM Algorithm, K={}, d={}".format(k, data.shape[1]))
plt.scatter(data.T[0], data.T[1], c=pred, cmap=cm.Set2)
plt.plot(m[:,0], m[:,1], '.', markersize=12, color = 'black')
plt.show()
print("Empirical Probabilities :\n{}\n".format(emp_prob(labels, pred, k)))
print("Covariances : \n{}\n".format(np.around(gmix.covariances_,3)))
print("Means : \n{}".format(np.around(gmix.means_,3)))
print(emp_prob(labels, pred, k))
```




```
Empirical Probabilities :
[[0.01  0.33 ]
 [0.    0.315]
 [0.185 0.16 ]]

Covariances :
[[[ 1.862e+00  3.330e-01  3.050e-01 ... -2.360e-01 -1.200e-01  7.480e-01]
  [ 3.330e-01  1.168e+00  1.120e-01 ... -1.350e-01 -1.670e-01 -2.000e-03]
  [ 3.050e-01  1.120e-01  1.464e+00 ... -2.240e-01  2.270e-01 -1.310e-01]
  ...
  [-2.360e-01 -1.350e-01 -2.240e-01 ...  1.616e+00 -2.740e-01 -7.200e-02]
  [-1.200e-01 -1.670e-01  2.270e-01 ... -2.740e-01  1.282e+00 -2.960e-01]
  [ 7.480e-01 -2.000e-03 -1.310e-01 ... -7.200e-02 -2.960e-01  1.292e+00]]

 [[ 2.360e+00  2.000e-03  1.089e+00 ... -1.200e-01  5.240e-01 -1.440e-01]
  [ 2.000e-03  1.372e+00  1.330e-01 ... -2.780e-01 -2.060e-01 -4.000e-01]
  [ 1.089e+00  1.330e-01  2.011e+00 ... -7.870e-01  1.020e-01 -3.400e-02]
  ...
  [-1.200e-01 -2.780e-01 -7.870e-01 ...  2.669e+00  8.780e-01 -5.080e-01]
  [ 5.240e-01 -2.060e-01  1.020e-01 ...  8.780e-01  2.033e+00 -5.330e-01]
  [-1.440e-01 -4.000e-01 -3.400e-02 ... -5.080e-01 -5.330e-01  1.849e+00]]]

Means :
[[ 0.752  0.293 -0.171 -1.427  0.457 -0.831  1.017 -0.591  1.336 -1.697
  -0.825  0.262  0.166 -1.626  1.177 -1.018 -0.037  0.093 -0.641  2.325
   1.056  0.897 -1.032  1.103 -0.641  0.762  1.41  -0.203 -1.19   0.79 ]
 [-1.407 -0.023 -0.087  0.484  0.446 -0.254 -0.136  0.615  0.033  1.51
   0.309 -0.772  0.118 -0.803  0.235  0.214 -0.172 -0.75   1.236 -0.406
   0.072 -0.077 -0.783  0.005  0.596  0.217  0.335 -0.722 -1.101 -0.542]]
```

In [ ]: