In [1]:
```
# Use gammaln for stability
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import loadmat
from scipy.special import digamma, gammaln, multigammaln
from scipy.stats import multivariate_normal, wishart
from sklearn.covariance import empirical_covariance
```

In [2]:
```
# Load data
data = loadmat('hw4_data_mat/data.mat')
X = data['X']
d = X.shape[0]
num = X.shape[1]
np.random.seed(3950)
```

In [3]:
```
# Set prior parameters
c_0 = 0.1
a_0 = d
alpha_0 = 1

# Calculate empirical mean
sum_X = np.sum(X, axis=1)
m_0 = sum_X / float(num)

# Calculate empirical covariance
A = empirical_covariance(X.T)
B_0 = c_0 * d * A
```

In [ ]:
```
def p_x(sample):
    phi_n_t1 = (c_0 / (np.pi * (1 + c_0)))**(0.5 * d)
    x_minus_m = (sample - m_0).reshape((d,1))
    phi_n_t2 = (np.linalg.det(B_0 + c_0/(c_0+1) * np.dot(x_minus_m.T, x_
minus_m)))**(-0.5*(a+1))/np.linalg.det(B_0)**(-0.5*a)
    phi_n_t3 = np.exp(multigammaln(0.5*(a+1), d)- multigammaln(0.5*a,
d))
    return alpha_0/float(alpha_0 + num - 1) * phi_n_t1 * phi_n_t2 * phi_
n_t3
```

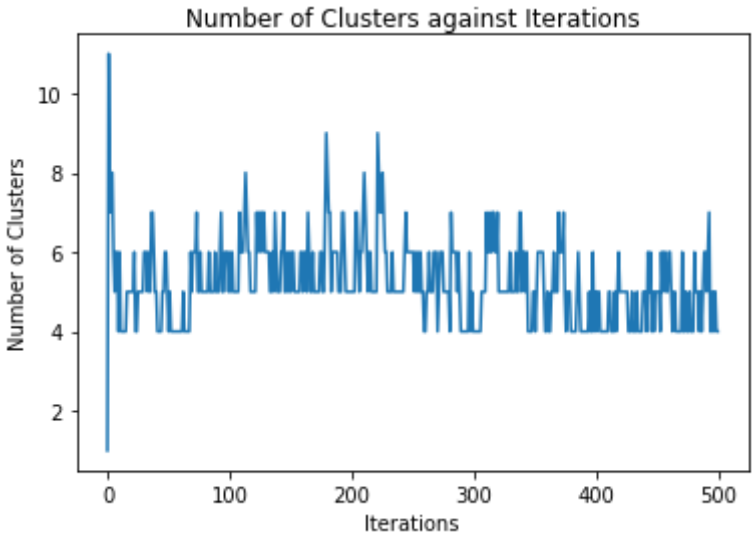
In [ ]:

```
# Initialisation
c = [0] * num
n = {0: range(num)}
theta = {}
lamda = wishart.rvs(a_0, np.linalg.inv(B_0))
covariance = np.linalg.inv(lamda)
theta[0] = [np.random.multivariate_normal(m_0, 1/float(c_0) *
covariance), covariance]

a = a_0
B = B_0
m = m_0

num_clusters = []
largest_six = []

p_x_all = map(lambda i: p_x(X[:, i]), range(num))

for iter in range(500):
    counts_clusters = [len(n[i]) for i in n]
    counts_clusters.sort(reverse=True)
    if len(n.keys()) < 6:
        largest_six.append(counts_clusters)
    else:
        largest_six.append(counts_clusters[:6])
    num_clusters.append(len(n.keys()))

    # 1
    for sample in range(num):
        # a) and b)
        phi = []
        for cluster in n:
            if c[sample] == cluster:
                n[cluster].remove(sample)
            phi_j = multivariate_normal.pdf(X[:, sample], theta[cluster]
[0], theta[cluster][1]) * len(n[cluster]) / float(alpha_0 + num - 1)
            phi.append(phi_j)
        phi.append(p_x_all[sample])

        # c)
        phi = np.array(phi) / sum(phi)
        idx_n = len(phi)
        c[sample] = int(np.random.choice(idx_n, 1, p = phi))
        # Add point to new cluster
        try:
            n[c[sample]].append(sample)
        except KeyError:
            n[c[sample]] = [sample]

        # d)
        if c[sample] == idx_n - 1:
            c_j = 1 + c_0
            m_j = c_0/(c_j) * m_0 + 1/(c_j) * X[:, sample]
            a_j = a_0 + 1
            x_bar_minus_m = np.array(X[:, sample] - m_0).reshape((d,1))
            B_j = B_0 + c_0/(c_j) * np.dot(x_bar_minus_m, x_bar_minus_m.T
```

```
            lamda_j = wishart.rvs(a_j, np.linalg.inv(B_j))
            covariance_j = np.linalg.inv(lamda_j)
            theta[idx_n - 1] = [np.random.multivariate_normal(m_j, 1/flo
at(c_j) * covariance_j), covariance_j]

        # Housekeeping
        # Remove all clusters with 0 entries
        n = { k : v for k,v in n.iteritems() if len(v) > 0}
        # Theta
        exist_c = n.keys()
        theta_n = {}
        for i in range(len(exist_c)):
            theta_n[i] = theta[exist_c[i]]
        theta = theta_n
        # Reindex clusters
        c_n = []
        n = {}
        for i in range(num):
            for j in range(len(exist_c)):
                if c[i] == exist_c[j]:
                    c_n.append(j)
                    try:
                        n[j].append(i)
                    except KeyError:
                        n[j] = [i]
        c = c_n

    # 2
    for cluster in n:
        s_j = len(n[cluster])
        c_j = s_j + c_0
        sum_j = np.sum(X[:, n[cluster]], axis=1)
        m_j = c_0/(c_j) * m_0 + 1/(c_j) * sum_j
        a_j = a_0 + s_j
        x_bar = 1/float(s_j) * sum_j
        x_minus_m_bar = X[:, n[cluster]].T - x_bar.T
        x_bar_minus_m = np.array(x_bar - m_0).reshape((d,1))
        B_j = B_0 + np.dot(x_minus_m_bar.T, x_minus_m_bar) + s_j * c_0/(c
 * np.dot(x_bar_minus_m, x_bar_minus_m.T)
        lamda_j = wishart.rvs(a_j, np.linalg.inv(B_j))
        covariance_j = np.linalg.inv(lamda_j)
        theta[cluster] = [np.random.multivariate_normal(m_j, 1/(c_j) * c
ovariance_j), covariance_j]

    # print 'Iteration ' + str(iter) + ' Done!'
```

In [ ]:
```
iterations = range(500)
plt.plot(iterations, num_clusters)
plt.xlabel('Iterations')
plt.ylabel('Number of Clusters')
plt.title('Number of Clusters against Iterations')
```

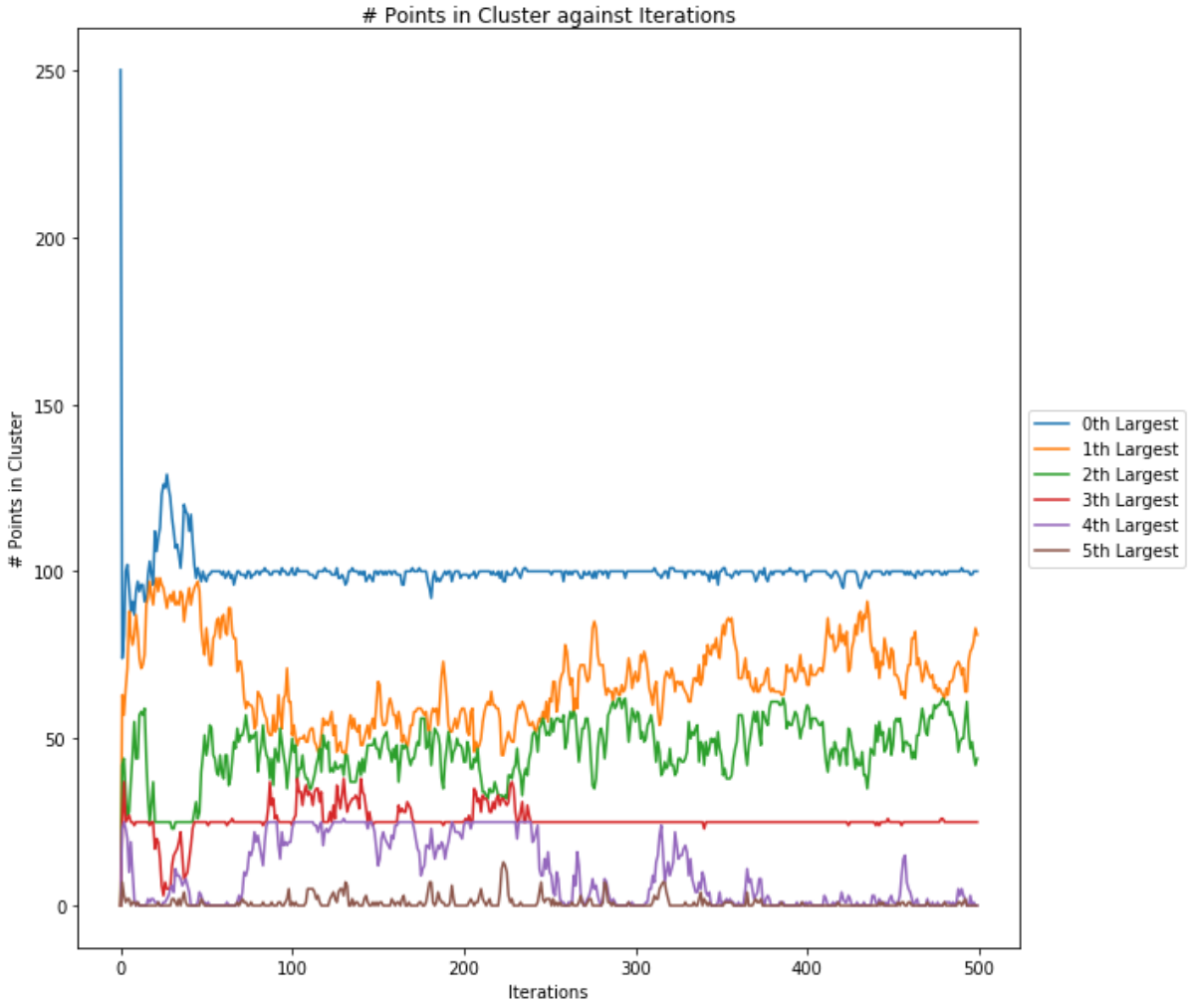
Out[ ]:
```
Text(0.5,1,u'Number of Clusters against Iterations')
```



In [ ]:
```
largest_six_split = {0:[], 1:[], 2:[], 3:[], 4:[], 5:[]}
for i in largest_six:
    for j in range(len(i)):
        largest_six_split[j].append(i[j])
    if len(i) < 6:
        for k in range(len(i), 6):
            largest_six_split[k].append(0)
```

In [ ]:
```
plt.figure(figsize=(10,10))
for i in largest_six_split:
    plt.plot(iterations, largest_six_split[i], label=str(i)+'th
Largest')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.xlabel('Iterations')
plt.ylabel('# Points in Cluster')
plt.title('# Points in Cluster against Iterations')
```

Out[ ]:
```
Text(0.5,1,u'# Points in Cluster against Iterations')
```



In [ ]: