Extra Credit: [Noise in GMM-EM]

Modify your GMM-EM routine by sampling and **injecting Gaussian noise** into the old faithful data at each iteration. **Scale the noise** to a fraction of the standard deviation in each dimension. And let the **noise standard deviation decay** at each iteration (e.g. inversely proportional to the square of the iteration counter). Compare **the average convergence time** of the GMM-EM with and without noise. **Plot the average convergence time** for different initial noise standard deviations.

From the given paper, we get the following procedures.

$$z_i = y_i + n_i$$

$$\alpha_j(t+1) = \frac{1}{N}\sum_{i=1}^{N} p_z(j|y_i,\Theta(t))$$

$$\mu_j(t+1) = \frac{\sum_{i=1}^{N} p_z(j|y_i,\Theta(t))z_i}{\sum_{i=1}^{N} p_z(j|y_i,\Theta(t))}$$

$$\Sigma_j(t+1) = \frac{\sum_{i=1}^{N} p_z(j|y_i,\Theta(t))(z_i-\mu_j(t))(z_i-\mu_j(t))^T}{\sum_{i=1}^{N} p_z(j|y_i,\Theta(t))}$$

The noise should follow the restriction in order to be beneficial to the convergence speed.

$$n_i[n_i - 2(\mu_{j_i} - y_i)] \leq 0$$

The covariance of the σ_N will decay with the iteration number.

$$\sigma_{Ni} = \frac{\sigma_{N0}}{i^2}$$

In [1]:

```
%matplotlib notebook
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv
from scipy.stats import multivariate_normal
from sklearn.cluster import KMeans
```

In [2]:

```
# Import the old faithful data from data.txt.
data = read_csv("data.txt", delim_whitespace=True, skipinitialspace=True)
xy = np.array([data['eruptions'],data['waiting']]).T
```

We define the following noise generation function for getting the noise.

In [3]:

```
# The function to inject the noise.
def noise_gen(var, n, mean1, mean2, xy):
    # Add the Gaussian noise to each independent directions.
    z_value, z_vector = np.linalg.eig(var)
    a_matrix = np.linalg.inv(z_vector.T)
    n_z = np.array([np.random.normal(0,z_value[0],n),np.random.normal(0,z_value[
1],n)])
    n_xy = np.dot(a_matrix, n_z)
    nx = n_xy[0,:]
    ny = n_xy[1,:]
    # Set the noise to zero, if it is not met the beneficial condition.
    is_ok_x_1 = nx*(nx-2*(mean1[0] - xy[:,0])) > 0
    is_ok_y_1 = ny*(ny-2*(mean1[1] - xy[:,1])) > 0
    is_ok_x_2 = nx*(nx-2*(mean2[0] - xy[:,0])) > 0
    is_ok_y_2 = ny*(ny-2*(mean2[1] - xy[:,1])) > 0
    is_ok_1 = np.logical_or(is_ok_x_1,is_ok_y_1)
    is_ok_2 = np.logical_or(is_ok_x_2,is_ok_y_2)
    is_ok = np.logical_or(is_ok_2,is_ok_2)
    nx[is_ok] = 0
    ny[is_ok] = 0
    return nx,ny
```

Next, we define our new noisy EM function.

In [4]:

```
def my_nem_2d2pgmm(xy,noise_level):
    # Initialize the parameters.
    model = KMeans(n_clusters=2)
    model.fit(xy)
    mean1_t = model.cluster_centers_[0]
    mean2_t = model.cluster_centers_[1]
    cov1_t = np.ma.cov(xy.T)
    var_n = noise_level* cov1_t
    cov2_t = cov1_t
    p_t = np.random.rand()
    # EM interation
    MAXITERATION = 10000
    tol = 0.001
    theta = np.r_[p_t, mean1_t, mean2_t, cov1_t.reshape(-1),cov2_t.reshape(-1)]
    iternum = 1
    N = len(xy)
    for i in range(MAXITERATION):
        # N-Step:
        nx,ny = noise_gen(var_n/((i+1)*(i+1)), len(xy), mean1_t , mean2_t, xy)
        n = np.array([nx,ny]).T
        # Inject the noise.
        z = xy + n
        # E-Step:
        w = np.array([p_t*multivariate_normal.pdf(xy, mean = mean1_t, cov = cov1
_t),
                      (1-p_t)*multivariate_normal.pdf(xy, mean = mean2_t, cov = co
v2_t)])
        w = w/sum(w,0)
        # M-Step:
        nml = sum(w.T)
        p_t = nml[0]/N
        mean1_t = np.r_[sum(w[0,:]*z[:,0]), sum(w[0,:]*z[:,1])]
        mean1_t = mean1_t/nml[0]
        mean2_t = np.r_[sum(w[1,:]*z[:,0]), sum(w[1,:]*z[:,1])]
        mean2_t = mean2_t/nml[1]
        c1 = np.array([w[0],w[0]]).T*(z-mean1_t)
        c2 = np.array([w[1],w[1]]).T*(z-mean2_t)
        cov1_t = np.dot(c1.T,z-mean1_t)/nml[0]
        cov2_t = np.dot(c2.T,z-mean2_t)/nml[1]
        # Calculate the difference of the parameters after one iteration.
        theta_t = np.r_[p_t, mean1_t, mean2_t, cov1_t.reshape(-1),cov2_t.reshape
(-1)]
        diff = np.linalg.norm(theta_t - theta,2)
        theta = theta_t
        # If the desired tolerance is met, break!
        if diff < tol:
            iternum = i+1
            break

    return iternum, p_t, mean1_t, mean2_t, cov1_t, cov2_t
```

Also include the original EM function for comparision.

In [5]:

```
def my_em_2d2pgmm(xy):
    # Initialize the parameters.
    model = KMeans(n_clusters=2)
    model.fit(xy)
    mean1_t = model.cluster_centers_[0]
    mean2_t = model.cluster_centers_[1]
    cov1_t = np.ma.cov(xy.T)
    cov2_t = cov1_t
    p_t = np.random.rand()
    # EM interation
    MAXITERATION = 10000
    tol = 0.001
    theta = np.r_[p_t, mean1_t, mean2_t, cov1_t.reshape(-1),cov2_t.reshape(-1)]
    iternum = 1
    N = len(xy)
    for i in range(MAXITERATION):
            # E-Step:
        w = np.array([p_t*multivariate_normal.pdf(xy, mean = mean1_t, cov = cov1
_t),
                      (1-p_t)*multivariate_normal.pdf(xy, mean = mean2_t, cov = co
v2_t)])
        w = w/sum(w,0)
            # M-Step:
        nml = sum(w.T)
        p_t = nml[0]/N
        mean1_t = np.r_[sum(w[0,:]*xy[:,0]), sum(w[0,:]*xy[:,1])]
        mean1_t = mean1_t/nml[0]
        mean2_t = np.r_[sum(w[1,:]*xy[:,0]), sum(w[1,:]*xy[:,1])]
        mean2_t = mean2_t/nml[1]
        c1 = np.array([w[0],w[0]]).T*(xy-mean1_t)
        c2 = np.array([w[1],w[1]]).T*(xy-mean2_t)
        cov1_t = np.dot(c1.T,xy-mean1_t)/nml[0]
        cov2_t = np.dot(c2.T,xy-mean2_t)/nml[1]
        # Calculate the difference.
        theta_t = np.r_[p_t, mean1_t, mean2_t, cov1_t.reshape(-1),cov2_t.reshape
(-1)]
        diff = np.linalg.norm(theta_t - theta,2)
        theta = theta_t
        if diff < tol:
            iternum = i+1
            break

    return iternum, p_t, mean1_t, mean2_t, cov1_t, cov2_t
```

Simulating under different noise levels for getting the relation between average convergence time and noise level.

In [6]:

```
avg_iternum_list = []
Noise_range = 50
for nl_i in np.arange(Noise_range):
    nl = 1.0 / np.power(2,nl_i)
    print nl,
    iter_list = []
    for i in range(30):
        iternum, p_t, mean1_t, mean2_t, cov1_t, cov2_t = my_nem_2d2pgmm(xy,nl)
        iter_list.append(iternum)
        if i%3 == 0:
            print str(round(float(i+1)/30*100)) + '%',
    avg_iternum = np.average(iter_list)
    avg_iternum_list.append(avg_iternum)
    print 'ok'
```

```
1.0 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% ok
0.5 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% ok
0.25 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% ok
0.125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% ok
0.0625 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% ok
0.03125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0% o
k
0.015625 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0%
ok
0.0078125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0%
ok
0.00390625 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.0
% ok
0.001953125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93.
0% ok
0.0009765625 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 93
.0% ok
0.00048828125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0% 9
3.0% ok
0.000244140625 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0%
93.0% ok
0.0001220703125 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0%
93.0% ok
6.103515625e-05 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0%
93.0% ok
3.0517578125e-05 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0
% 93.0% ok
1.52587890625e-05 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
7.62939453125e-06 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
3.81469726562e-06 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.90734863281e-06 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
9.53674316406e-07 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
```

```
0% 93.0% ok
4.76837158203e-07 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.38418579102e-07 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.19209289551e-07 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
5.96046447754e-08 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.98023223877e-08 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.49011611938e-08 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
7.45058059692e-09 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
3.72529029846e-09 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.86264514923e-09 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
9.31322574615e-10 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
4.65661287308e-10 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.32830643654e-10 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.16415321827e-10 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
5.82076609135e-11 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.91038304567e-11 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.45519152284e-11 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
7.27595761418e-12 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
3.63797880709e-12 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.81898940355e-12 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
9.09494701773e-13 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
4.54747350886e-13 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.27373675443e-13 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.13686837722e-13 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
5.68434188608e-14 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
2.84217094304e-14 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
1.42108547152e-14 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.
0% 93.0% ok
```

```
7.1054273576e-15 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0
% 93.0% ok
3.5527136788e-15 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0
% 93.0% ok
1.7763568394e-15 3.0% 13.0% 23.0% 33.0% 43.0% 53.0% 63.0% 73.0% 83.0
% 93.0% ok
```

In [7]:

```
plt.figure()
plt.plot(1.0/np.arange(Noise_range),avg_iternum_list)
for i in range(300):
    iternum, p_t, mean1_t, mean2_t, cov1_t, cov2_t = my_em_2d2pgmm(xy)
plt.plot(1.0/np.arange(Noise_range),np.ones_like(np.arange(Noise_range))*iternum
)
plt.title('Average convergence time')
plt.ylabel('Iteration number')
plt.xlabel('Noise level (fraction of the covariance)')
plt.legend(['Noisy EM','EM'])
plt.grid()
plt.show()
```

From the figure above, the convergence time is not speed up as expected.