In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import log, exp, sqrt, pi, pow, e
from scipy.stats import multivariate_normal as mn
import time
import scipy
```

In [3]:
```
traindata=pd.read_table('/Users/adilrhiulam/Downloads/classification_data_HWK2/EMGaussiandata.t
xt',names=['x','y'],sep=' ')
testdata=pd.read_table('/Users/adilrhiulam/Downloads/classification_data_HWK2/EMGaussiantest.tx
t',names=['x','y'],sep=' ')
```

In [436]:
```
means_opt = np.array([[ 3.78877284, -3.9680643 ],\
       [-1.92961689,  4.16234703],\
       [ 3.99443411,  3.64129121],\
       [-2.99518081, -3.46966615]])

covariances_opt = np.array([[[  0.94516471,   0.05891181],\
        [  0.05891181,   1.57294922]],\
       [[  3.31567711,   0.21109042],\
        [  0.21109042,   2.93788675]],\
       [[  0.19737178,   0.25893186],\
        [  0.25893186,  12.31797841]],\
       [[  6.74686364,   6.53198784],\
        [  6.53198784,   6.64296548]]])

train_data=traindata.values.tolist()
test_data=testdata.values.tolist()
data1 = [np.random.multivariate_normal(means_opt[0],covariances_opt[0]) for i in range(2000)]
data2 = [np.random.multivariate_normal(means_opt[1],covariances_opt[1]) for i in range(2000)]
data3 = [np.random.multivariate_normal(means_opt[2],covariances_opt[2]) for i in range(2000)]
data4 = [np.random.multivariate_normal(means_opt[3],covariances_opt[3]) for i in range(2000)]
data= train_data+test_data+data1+data2+data3+data4
```

In [246]:
```
means_ini = [[ 0.71618713,  0.59347407],\
             [ 0.63137568,  0.82226145],\
             [ 0.14756696,  0.41482631],\
             [ 0.71879351,  0.08157871]]
```

In [248]:
```
def log_alpha_recursion(data,t,log_alpha_t_minus_1,A,log_PI,means,covariances):
    log_alpha_t = np.zeros(4)
    if (t==0):
        for k in range(4):
            log_alpha_t[k] = log_PI[k]+\
            log(mn.pdf(data[t],means[k],covariances[k]))
        return log_alpha_t
    maximum = np.max(log_alpha_t_minus_1)
    log_alpha_t_minus_1_ = np.array(log_alpha_t_minus_1) - maximum
    for k in range(4):
        somme = 0
        for i in range(4):
            somme+=exp(log_alpha_t_minus_1_[i])*A[k,i]
        log_alpha_t[k] = log(somme)+ maximum +log(mn.pdf(data[t],\
                                 means[k],covariances[k]))
    return log_alpha_t

def log_beta_recursion(data,t,log_beta_t_plus_1,A,means,covariances):
    log_beta_t = np.zeros(4)
    T=len(data)
    if (t==T-1):
        log_beta_t = [0]*4
        return log_beta_t
    maximum = np.max(log_beta_t_plus_1)
    log_beta_t_plus_1_ = np.array(log_beta_t_plus_1) - maximum
    for k in range(4):
        somme = 0
        for i in range(4):
            somme+=exp(log_beta_t_plus_1_[i])*A[i,k]*mn.pdf(data[t+1],\
                                                         means[i],covariances[i])
        log_beta_t[k] = log(somme) + maximum
    return log_beta_t

def log_alpha(data,A,log_PI,means,covariances):
    T = len(data)
    log_alpha = np.zeros((T,4))
    log_alpha_t = []
    for t in range(T):
        log_alpha_t = log_alpha_recursion(data,t,log_alpha_t,A,log_PI,means,covariances)
        log_alpha[t] = log_alpha_t
    return log_alpha

def log_beta(data,A,means,covariances):
    T = len(data)
    log_beta = np.zeros((T,4))
    log_beta_t = []
    for t in range(T):
        log_beta_t = log_beta_recursion(data,T-1-t,log_beta_t,A,means,covariances)
        log_beta[T-1-t] = log_beta_t
    return log_beta

def log_smoothing(data,A,means,covariances,log_alpha_,log_beta_):
    T = len(data)
    log_product = log_alpha_ + log_beta_
    log_likelihood = log(np.sum(np.exp(log_alpha_[-1]-\
                         np.max(log_alpha_[-1]))))+np.max(log_alpha_[-1])
    log_smoothing = np.zeros((T,4))
    log_smoothing = log_product-log_likelihood
    return [log_smoothing,log_likelihood]

def log_joint_proba(data,A,means,covariances,log_alpha_,log_beta_,log_likelihood):
    T = len(data)
    log_joint_proba = np.zeros((T-1,4,4))
    for t in range(T-1):
        for i in range(4):
            for j in range(4):
                log_joint_proba[t,i,j] = log_alpha_[t,j] + log_beta_[t+1,i] + log(A[i,j])+\
                +log(mn.pdf(data[t+1],means[i],covariances[i])) - log_likelihood
    return log_joint_proba

def intializationStepHMM():
    means = means_ini #np.array([[ 0, -0.79729724],\
       #[.03436695,  4.17258597],\
       #[ 3.97793028,  3.77333115],\
       #[-3.0619607 , -3.53454046]])

    covariances = np.array([[[  0.92127919,   0.05738079],\
        [  0.05738079,   1.86586056]],\
        [[  2.90442382,   0.20655764],\
        [  0.20655764,   2.75617077]],\
        [[  0.21035665,   0.29045072],\
        [  0.29045072,  12.23996355]],\
        [[  6.2414092 ,   6.05017473],\
        [  6.05017473,   6.18245537]]])
    A = np.array([[0.5,1/6,1/6,1/6],\
           [1/6,0.5,1/6,1/6],\
           [1/6,1/6,0.5,1/6],\
           [1/6,1/6,1/6,0.5]])
    log_PI = np.log([1/4,1/4,1/4,1/4])
    return [means,covariances,A,log_PI]

def stepE_HMM(data,means,covariances,A,log_PI):
    log_alpha_ = log_alpha(data,A,log_PI,means,covariances)
    log_beta_ = log_beta(data,A,means,covariances)
    log_smoothing_,log_likelihood = log_smoothing(data,A,means\
                                              ,covariances,log_alpha_,log_beta_)
    log_joint_proba_ = log_joint_proba(data,A,means\
                                   ,covariances,log_alpha_,log_beta_,log_likelihood)
    return [log_smoothing_,log_joint_proba_,log_likelihood]

def stepM_HMM(data,log_smoothing_,log_joint_proba_):
    T = len(data)
    log_PI = log_smoothing_[0]
    A = (np.sum(np.exp(log_joint_proba_),axis=0).T/np.sum(np.exp(log_smoothing_[:T-1]),axis=0
)).T
    x = np.zeros((T,4,2))
    y = np.sum(np.exp(log_smoothing_),axis=0)
    for t in range(T):
        for i in range(4):
            x[t,i] = np.exp(log_smoothing_)[t,i]*np.array(data[t])
    means = (np.sum(x,axis=0).T/y).T
    covariances = np.array([np.sum([np.exp(log_smoothing_)[t,i]*\
                                np.dot(np.array([data[t]-means[i]]).T,[data[t]-means[i]])\
                                for t in range(T)],axis=0)/y[i] for i in range(4)])
    return [log_PI,A,means,covariances]

def algorithmEM_HMM(data,r):
    [means,covariances,A,log_PI] = intializationStepHMM()
    means_history = [means]
    [log_smoothing_,log_joint_proba_,log_likelihood] = stepE_HMM(data,means,covariances,A,log_
PI)
    log_likelihood_history = [log_likelihood]
    log_llh2 = log_likelihood+2
    for i in range(r):
        log_llh2 = log_likelihood
        [log_PI,A,means,covariances] = stepM_HMM(data,log_smoothing_,log_joint_proba_)
        means_history.append(means)
        [log_smoothing_,log_joint_proba_,log_likelihood] = stepE_HMM(data,means,covariances,A,
log_PI)
        log_likelihood_history.append(log_likelihood)
    return [log_PI,A,means,covariances,log_likelihood_history,means_history]
```

In the usual EM we have to do a forward-backward messages transmission to calculate the filtering and smoothing probabilities in the E step to evaluate all the parameters in the M step.

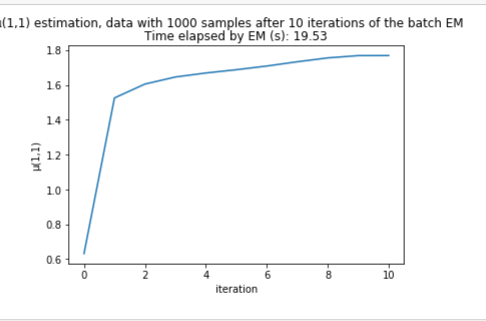
In [397]:
```
def phi(data,n,phi_n_minus_1,A,PI,means,covariances):
    phi_n = np.zeros(4)
    if (n==0):
        resp = np.array([PI[k]*mn.pdf(data[0],means[k],covariances[k]) for k in range(4)])
        for k in range(4):
            phi_n[k] = resp[k]/np.sum(resp)
        return phi_n
    tmp = np.array([[phi_n_minus_1[i]*A[k,i]*mn.pdf(data[n],means[k],covariances[k]) for i in
range(4)]\
           for k in range(4)])
    for k in range(4):
        tmp_1 = np.sum(tmp,axis=1)
        phi_n[k] = tmp_1[k]/np.sum(tmp_1)
    return phi_n

def r_seq(n,phi_n_minus_1,A):
    assert(n>0)
    r_n = np.zeros((4,4))
    for j in range(4):
        tmp = [phi_n_minus_1[i]*A[j,i] for i in range(4)]
        for i in range(4):
            r_n[i,j] = tmp[i]/np.sum(tmp)
    return r_n

def rho_q(data,n,rho_q_n_1,r_n):
    rho_q_n = np.zeros((4,4,4))
    if(n==0):
        return rho_q_n
    for j,k in zip(range(4),range(4)):
        for i in range(4):
            rho_q_n[i,j,k]+=(1/n)*r_n[i,j]
    for i in range(4):
        for j in range(4):
            for k in range(4):
                tmp = [rho_q_n_1[i,j,k1]*r_n[k1,k] for k1 in range(4)]
                rho_q_n[i,j,k]+=(1-(1/n))*np.sum(tmp)
    return rho_q_n

def rho_g1(data,n,rho_g1_n_1,r_n):
    rho_g1_n = np.zeros((4,4))
    if(n==0):
        for i,k in zip(range(4),range(4)):
            rho_g1_n[i,k] = 1
        return rho_g1_n
    for i,k in zip(range(4),range(4)):
            rho_g1_n[i,k]+=1/n
    for i in range(4):
        for k in range(4):
            tmp = [rho_g1_n_1[i,k1]*r_n[k1,k] for k1 in range(4)]
            rho_g1_n[i,k]+=(1-(1/n))*np.sum(tmp)
    return rho_g1_n

def rho_g2(data,n,rho_g2_n_1,r_n):
    rho_g2_n = np.zeros((4,4,2))
    if(n==0):
        for i,k in zip(range(4),range(4)):
            rho_g2_n[i,k] = data[n]
        return rho_g2_n
    for i,k in zip(range(4),range(4)):
            rho_g2_n[i,k]+=(1/n)*np.array(data[n])
    for i in range(4):
        for k in range(4):
            tmp = [rho_g2_n_1[i,k1]*r_n[k1,k] for k1 in range(4)]
            rho_g2_n[i,k]+=(1-(1/n))*np.sum(tmp,axis=0)
    return rho_g2_n

def rho_g3(data,n,rho_g3_n_1,r_n):
    rho_g3_n = np.zeros((4,4,2,2))
    if(n==0):
        for i,k in zip(range(4),range(4)):
            rho_g3_n[i,k] = np.dot(np.array([data[n]]).T,[data[n]])
        return rho_g3_n
    for i,k in zip(range(4),range(4)):
            rho_g3_n[i,k]+=(1/n)*np.dot(np.array([data[n]]).T,[data[n]])
    for i in range(4):
        for k in range(4):
            tmp = [rho_g3_n_1[i,k1]*r_n[k1,k] for k1 in range(4)]
            rho_g3_n[i,k]+=(1-(1/n))*np.sum(tmp,axis=0)
    return rho_g3_n

def trans_matrix_upd(rho_q_n,phi_n):
    update = np.zeros((4,4))
    S = np.zeros((4,4))
    for i in range(4):
        for j in range(4):
            tmp = [rho_q_n[i,j,k]*phi_n[k] for k in range(4)]
            S[i,j] = np.sum(tmp)
    for i in range(4):
        update[i,:] = S[i,:]/np.sum(S[i,:])
    return update.T

def mean_update(rho_g1_n,rho_g2_n,phi_n):
    update = np.zeros((4,2))
    for i in range(4):
        tmp1 = [rho_g1_n[i,k]*phi_n[k] for k in range(4)]
        tmp2 = [rho_g2_n[i,k]*phi_n[k] for k in range(4)]
        update[i] = np.sum(tmp2,axis=0)/np.sum(tmp1)
    return update

def covariance_update(rho_g1_n,rho_g3_n,phi_n,mean_updated):
    update = np.zeros((4,2,2))
    for i in range(4):
        tmp1 = [rho_g1_n[i,k]*phi_n[k] for k in range(4)]
        tmp3 = [rho_g3_n[i,k]*phi_n[k] for k in range(4)]
        tmp = np.dot(np.array([mean_updated[i]]).T,[mean_updated[i]])
        update[i] = (np.sum(tmp3,axis=0)/np.sum(tmp1)) - tmp
    return update

def _ini_():
    #means = np.array([[ 0, -0.79729724],\
     #  [-2.03436695,  4.17258597],\
      # [ 3.97793028,  3.77333115],\
      # [-3.0619607 , -3.53454046]])
    means = means_ini
    #means = [0.1,2,-3,3]
    #covariances = [1,10,3,2]
    covariances = np.array([[[  0.92127919,   0.05738079],\
        [  0.05738079,   1.86586056]],\
        [[  2.90442382,   0.20655764],\
        [  0.20655764,   2.75617077]],\
        [[  0.21035665,   0.29045072],\
        [  0.29045072,  12.23996355]],\
        [[  6.2414092 ,   6.05017473],\
        [  6.05017473,   6.18245537]]])
    A = np.array([[0.5,1/6,1/6,1/6],\
           [1/6,0.5,1/6,1/6],\
           [1/6,1/6,0.5,1/6],\
           [1/6,1/6,1/6,0.5]])
    PI = np.array([1/4,1/4,1/4,1/4])
    return [means,covariances,A,PI]

def online_EM_Gaussian_HMM(data,n_min):
    means,covariances,A,PI = _ini_()
    means_history = [means]
    #for t in range(T):
        #np.random.shuffle(data)
        #data_online = data[0:np.random.randint(50,len(data))]
    data_online = data
    phi_ = phi(data_online,0,[],A,PI,means,covariances)
    rho_q_ = rho_q(data_online,0,[],[])
    rho_g1_ = rho_g1(data_online,0,[],[])
    rho_g2_ = rho_g2(data_online,0,[],[])
    rho_g3_ = rho_g3(data_online,0,[],[])
    #n = 0
        #if (len(data_online)>n_min):
    for n in range(len(data)-1):
            r_seq_ = r_seq(n+1,phi_,A)
            rho_q_ = rho_q(data_online,n+1,rho_q_,r_seq_)
            rho_g1_ = rho_g1(data_online,n+1,rho_g1_,r_seq_)
            rho_g2_ = rho_g2(data_online,n+1,rho_g2_,r_seq_)
            rho_g3_ = rho_g3(data_online,n+1,rho_g3_,r_seq_)
            phi_ = phi(data_online,n+1,phi_,A,PI,means,covariances)

        #cov = covariance_update(rho_g1_,rho_g3_,phi_,means)
            if(n>=n_min):# and not((False in list(map(lambda x: (scipy.linalg.eigh(x)[0]>10**-
13).all(),cov))))):
                A = trans_matrix_upd(rho_q_,phi_)
                means = mean_update(rho_g1_,rho_g2_,phi_)
                covariances = covariance_update(rho_g1_,rho_g3_,phi_,means)
            means_history.append(means)
    return [A,means,covariances,means_history]
```

In [350]:
```
t = time.time()
[log_PI_1,A_1,means_1,covariances_1,log_likelihood_history_1,means_history_1] = algorithmEM_HM
M(data[:1000],10)
elapsed_EM_1 = time.time() - t
```

In [351]:
```
print('The elapsed time by the batch EM algorithm after 10 iterations with 500 samples:', elap
sed_EM_1)
```

The elapsed time by the batch EM algorithm after 10 iterations with 500 samples: 19.52837014
1983032

In [442]:
```
plt.plot(np.array(means_history_1)[:,:,0][:,1],label='EM')
#plt.plot(np.array(means_history_)[:,:,0][:,1],label='EM')
plt.suptitle('μ(1,1) estimation, data with 1000 samples after 10 iterations of the batch EM ')
plt.title('Time elapsed by EM (s): '+str(round(elapsed_EM_1,2)))
plt.xlabel('iteration')
plt.ylabel('μ(1,1)')
plt.show()
```

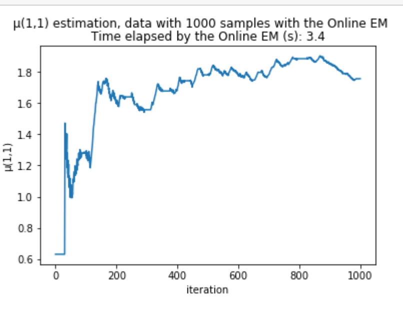


In [439]:
```
t = time.time()
[A_2,means_2,covariances_2,means_history_2] = online_EM_Gaussian_HMM(data[:1000],30)
elapsed_1 = time.time() - t
```

In [440]:
```
print('elapsed time by the online EM algorithm with 500 samples is: ', elapsed_1 )
```

elapsed time by the online EM algorithm with 500 samples is: 3.4038500785827637

In [443]:
```
plt.plot(np.array(means_history_2)[:,:,0][:,1],label='Online EM')
#plt.plot(np.array(means_history_)[:,:,0][:,1],label='EM')
plt.suptitle('μ(1,1) estimation, data with 1000 samples with the Online EM ')
plt.title('Time elapsed by the Online EM (s): '+str(round(elapsed_1,2)))
plt.xlabel('iteration')
plt.ylabel('μ(1,1)')
plt.show()
```

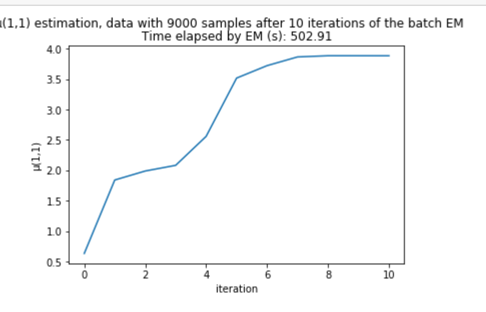


In [249]:
```
t = time.time()
[log_PI,A,means,covariances,log_likelihood_history,means_history] = algorithmEM_HMM(data,10)
elapsed_EM = time.time() - t
```

In [352]:
```
print('The elapsed time by the batch EM algorithm after 10 iterations with 9000 samples:', ela
psed_EM)
```

The elapsed time by the batch EM algorithm after 10 iterations with 9000 samples: 502.905247
92671204

In [270]:
```
plt.plot(np.array(means_history)[:,:,0][:,1],label='EM')
#plt.plot(np.array(means_history_)[:,:,0][:,1],label='EM')
plt.suptitle('μ(1,1) estimation, data with 9000 samples after 10 iterations of the batch EM ')
plt.title('Time elapsed by EM (s): '+str(round(elapsed_EM,2)))
plt.xlabel('iteration')
plt.ylabel('μ(1,1)')
plt.show()
```

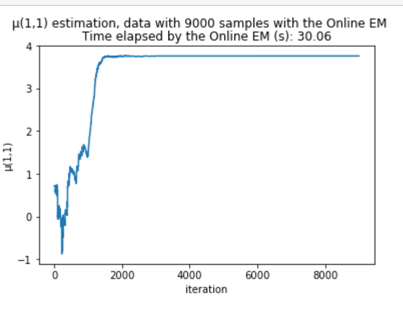


In [256]:
```
t = time.time()
[A_,means_,covariances_,means_history_] = online_EM_Gaussian_HMM(data,30)
elapsed = time.time() - t
```

In [257]:
```
print('elapsed time by the online EM algorithm is: ', elapsed )
```

elapsed time by the online EM algorithm is: 30.061241149902344

In [268]:
```
plt.plot(np.array(means_history_)[:,:,0][:,0],label='Online EM')
#plt.plot(np.array(means_history_)[:,:,0][:,1],label='EM')
plt.suptitle('μ(1,1) estimation, data with 9000 samples with the Online EM ')
plt.title('Time elapsed by the Online EM (s): '+str(round(elapsed,2)))
plt.xlabel('iteration')
plt.ylabel('μ(1,1)')
plt.show()
```



In [368]:
```
means=[[],[],[],[]]
for k in range(4):
    for i in range(50):
```