HARD AND SOFT EM DERIVATIONS

a) Let $\theta = (\pi, M, g)$

where $\pi = p(z(0))$, $M_{ij} = P(X(t+1) = j \mid X(t) = i)$

and $g_i(j) = P(Y(t) = j \mid X(t) = i)$

Given $z(t) = \arg\max_i P(z(t) = i \mid Y(0), \dots, Y(T))$

we update the parameters θ at iteration $k+1$ using:

$$l(\theta) = \max_{\theta} \log P(z(0), \dots, z(T), Y(0), \dots, Y(T) \mid \theta)$$

$$= \max_{\theta} \log \pi(z(0)) \prod_{t=0}^{T-1} M_{z(t)z(t+1)} \prod_{t=0}^{T} g_{z(t)}(Y(t))$$

$$= \max_{\theta} \log \pi(z(0)) + \sum_{t=0}^{T-1} \log M_{z(t)z(t+1)} + \sum_{t=0}^{T} \log g_{z(t)}(Y(t))$$

We need $\nabla_\theta l(\theta)$ and equate it to respective constraints

$$\frac{\partial l(\theta)}{\partial M_{ij}} = \frac{1}{M_{ij}} \sum_{t=0}^{T-1} \mathbb{1}(z(t) = i, z(t+1) = j) = \frac{1}{M_{ij}} \cdot N_{ij}$$

Note that $\sum_{j=0}^{1} M_{ij} = 1$

$$\Rightarrow \frac{\partial l(\theta)}{\partial M_{ij}} = \lambda$$

$$\Rightarrow \frac{N_{ij}}{M_{ij}} = \lambda$$

$$\Rightarrow \sum_{j=0}^{1} N_{ij} = \sum_{j=0}^{1} M_{ij} \lambda$$

$$\Rightarrow \sum_{j=0}^{1} N_{ij} = \lambda$$

$$\Rightarrow M_{ij} = \frac{N_{ij}}{\sum_{j=0}^{1} N_{ij}}$$

so given the assignments based on smoothing probabilities, at iteration $k+1$, $M^{(k+1)}$ is a 2×2 matrix $\ni$

$$M^{(k+1)} = \begin{bmatrix} \frac{N_{00}}{\sum_{j=0}^{1} N_{0j}} & \frac{N_{01}}{\sum_{j=0}^{1} N_{0j}} \\ \frac{N_{10}}{\sum_{j=0}^{1} N_{1j}} & \frac{N_{11}}{\sum_{j=0}^{1} N_{1j}} \end{bmatrix}$$

Let's go on to the the $g^{(k+1)}$ update given the smoothing probabilities

$$\frac{\partial l(\theta)}{\partial g_j(i)} = \frac{1}{g_j(i)} \sum_{t=0}^{T} \mathbb{1}(Y(t)=i, Z(t)=j)$$

Again, note we are constrained by $\sum_{i=1}^{6} g_j(i) = 1$

$$\Rightarrow \frac{\partial l(\theta)}{\partial g_j(i)} = \lambda$$

$$\Rightarrow \frac{1}{g_j(i)} \sum_{t=0}^{T} \mathbb{1}(Y(t)=i, Z(t)=j) = \lambda$$

$$\Rightarrow \sum_{i=1}^{6} \sum_{t=0}^{T} \mathbb{1}(Y(t)=i, Z(t)=j) = \lambda \sum_{i=1}^{6} g_j(i)$$

$$\Rightarrow \sum_{t=0}^{T} \mathbb{1}(Z(t)=j) = \lambda$$

$$\Rightarrow g_j(i) = \frac{\sum_{t=0}^{T} \mathbb{1}(Y(t)=i, Z(t)=j)}{\sum_{t=0}^{T} \mathbb{1}(Z(t)=j)}$$

So given the assignments, at iteration $k+1$, $g^{(k+1)}$ is a 2×6 matrix $\Rightarrow$

$$g^{(k+1)} = \begin{bmatrix} g_0(1) & g_0(2) & \cdots & g_0(6) \\ g_1(1) & g_1(2) & \cdots & g_1(6) \end{bmatrix}$$

Lastly, we need $\pi(z(0))$

Since we want $\max \log \pi(z(0))$

we let $\pi(z(0) = i) = 1$ if $z(0) = i$ and

$\pi(z(0) = 1-i) = 0$ o/w

This will max $\pi(z(0))$.

b)

Assume $\theta^{(k)}$ given, then we want

$$\max_{\theta^{(k+1)}} Q(\theta^{(k)}, \theta^{(k+1)}) = \max_{\theta^{(k+1)}} E\left[\ell(Y, z \mid \theta^{(k+1)})\right]$$

$$= \max_{\theta^{(k+1)}} \sum_{z} p(z \mid Y, \theta^{(k)}) \, \ell(Y, z \mid \theta^{(k+1)})$$

$$= \max_{\theta^{(k+1)}} \sum_{z} p(z \mid Y, \theta^{(k)}) \log p(Y, z \mid \theta^{(k+1)})$$

$$= \max_{\theta^{(k+1)}} \sum_{z(0)=0}^{1} \cdots \sum_{z(T)=0}^{1} p(z(0), \ldots, z(T) \mid Y, \theta^{(k)}) \cdot$$

$$\cdot \left[\log \pi(z(0)) + \sum_{t=0}^{T-1} \log M_{z(t)z(t+1)} + \sum_{t=0}^{T} \log g_{z(t)}(Y(t))\right]$$

Now, similar to what was done for HardEM:

$$\frac{\partial}{\partial M_{ij}} Q(\theta^{(k)}, \theta^{(k+1)}) = \sum_{z(0)=0}^{1} \cdots \sum_{z(T)=0}^{1} \sum_{t=0}^{T-1} \frac{1}{M_{ij}} \mathbb{1}(z(t)=i, z(t+1)=j)$$
$$\cdot P(z(0), \ldots, z(T) \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

$$= \sum_{t=0}^{T-1} \sum_{z(t)=0}^{1} \sum_{z(t+1)=0}^{1} \frac{1}{M_{ij}} \mathbb{1}(z(t)=i, z(t+1)=j) \, P(z(t)=i, z(t+1)=j \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

$$= \sum_{t=0}^{T-1} \frac{1}{M_{ij}} \mathbb{1}(z(t)=i, z(t+1)=j) \, P(z(t)=i, z(t+1)=j \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

Now, we need to derive $P(z(t)=i, z(t+1)=j \mid Y(0), \ldots, Y(T), \theta^{(k)})$

Let's call it $\xi_t(i,j)$:

$$\xi_t(i,j) = \frac{P(z(t)=i, z(t+1)=j, Y(0), \ldots, Y(T), \theta^{(k)})}{P(Y(0), \ldots, Y(T), \theta^{(k)})}$$

$$= \frac{P(\gamma(0),\dots,\gamma(T) \mid z(t)=i, z(t+1)=j, \theta^{(k)})\, P(z(t)=i, z(t+1)=j, \theta^{(k)})}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})}$$

$$= \frac{P(\gamma(0),\dots,\gamma(t) \mid z(t)=i, z(t+1)=j, \theta^{(k)})\, P(\gamma(t+1),\dots,\gamma(T) \mid z(t)=i, z(t+1)=j, \theta^{(k)})\; P(z(t+1)=j \mid z(t)=i, \theta^{(k)})\, P(z(t)=i \mid \theta^{(k)})}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})}$$

$$= \frac{P(\gamma(0),\dots,\gamma(t) \mid z(t)=i)\, P(z(t)=i \mid \theta^{(k)})\, P(z(t+1)=j \mid z(t)=i)\, P(\gamma(t+2),\dots,\gamma(T) \mid z(t+1)=j, z(t)=i)\; P(\gamma(t+1) \mid z(t+1)=j, z(t)=i)}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})}$$

$$= \frac{P(\gamma(0),\dots,\gamma(t) \mid z(t)=i)\, P(z(t)=i \mid \theta^{(k)})\, P(z(t+1)=j \mid z(t)=i)\, P(\gamma(t+2),\dots,\gamma(T) \mid z(t+1)=j)\; P(\gamma(t+1) \mid z(t+1)=j)}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})}$$

$$= \frac{\alpha_t^{(k)}(i) \cdot M_{ij}^{(k)} \cdot \beta_{t+1}^{(k)}(j) \cdot g_j^{(k)}(\gamma(t+1))}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})}$$

Now note $$\sum_{i,j} \frac{\alpha_t^{(k)}(i) \cdot M_{ij}^{(k)} \cdot \beta_{t+1}^{(k)}(j) \cdot g_j^{(k)}(\gamma(t+1))}{P(\gamma(0),\dots,\gamma(T), \theta^{(k)})} = 1$$

$$\Rightarrow P(\gamma(0),\dots,\gamma(T), \theta^{(k)}) = \sum_{i,j} \alpha_t^{(k)}(i) \cdot M_{ij}^{(k)} \cdot \beta_{t+1}^{(k)}(j) \cdot g_j^{(k)}(\gamma(t+1))$$

So, $$\frac{\partial}{\partial M_{ij}} Q(\theta^{(k)}, \theta^{(k+1)}) = \frac{1}{M_{ij}} \sum_{t=0}^{T-1} \mathbb{1}(z(t)=i, z(t+1)=j)\, \xi_t(i,j) = \lambda$$

$$\Rightarrow \sum_{j=0}^{1} \sum_{t=0}^{T-1} \mathbb{1}(z(t)=i, z(t+1)=j)\, \xi_t(i,j) = \lambda$$

$$\Rightarrow \sum_{j=0}^{1} \sum_{t=0}^{T-1} \xi_t(i,j) = \lambda$$

$$\Rightarrow M_{ij} = \frac{\sum_{t=0}^{T-1} \xi_t(i,j)}{\sum_{t=0}^{T} \sum_{k=0}^{1} \xi_t(i,k)}$$

Finally,

$$M^{(k+1)} = \begin{bmatrix} \frac{\sum_{t=0}^{T-1} \xi_t(0,0)}{\sum_{t=0}^{T-1}\sum_{k=0} \xi_t(0,k)} & \frac{\sum_{t=0}^{T-1} \xi_t(0,1)}{\sum_{t=0}^{T-1}\sum_{k=0} \xi_t(0,k)} \\ \frac{\sum_{t=0}^{T-1} \xi_t(1,0)}{\sum_{t=0}^{T-1}\sum_{k=0} \xi_t(1,k)} & \frac{\sum_{t=0}^{T-1} \xi_t(1,1)}{\sum_{t=0}^{T-1}\sum_{k=0} \xi_t(1,k)} \end{bmatrix}$$

Now, similarly:

$$\frac{\partial}{\partial g_j(i)} Q(\theta^{(k)}, \theta^{(k+1)}) = \sum_{z(0)=0}^{1} \cdots \sum_{z(T)=0}^{1} \sum_{t=0}^{T-1} \frac{1}{g_j(i)} \mathbb{1}(z(t)=j, Y(t)=i) \cdot P(z(0), \ldots, z(T) \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

$$= \sum_{t=0}^{T-1} \frac{1}{g_j(i)} \mathbb{1}(z(t)=j, Y(t)=i) p(z(t)=j \mid Y(0), \ldots Y(T), \theta^{(k)}) = \lambda$$

$$\Rightarrow \sum_{t=0}^{T-1} \sum_{i=1}^{6} \mathbb{1}(z(t)=j, Y(t)=i) \gamma_t(j) = \sum_{i=1}^{6} \lambda g_j(i)$$

$$\Rightarrow \lambda = \sum_{t=0}^{T-1} \sum_{i=1}^{6} \gamma_t(j) \mathbb{1}(z(t)=j, Y(t)=i)$$

$$\Rightarrow g_{ji} = \frac{\sum_{t=0}^{T-1} \gamma_t(j) \mathbb{1}(z(t)=j, Y(t)=i)}{\sum_{t=0}^{T-1} \sum_{i=1}^{6} \gamma_t(j) \mathbb{1}(z(t)=j, Y(t)=i)}$$

$$= \frac{\sum_{t=0}^{T-1} \gamma_t(j)\, \mathbb{1}(Y(t)=i)}{\sum_{t=0}^{T-1} \gamma_t(j)}$$

So:

$$g^{(t)} = \begin{bmatrix} \frac{\sum_{t=0}^{T-1} \gamma_t(0)\, \mathbb{1}(Y(t)=1)}{\sum_{t=0}^{T-1} \gamma_t(0)} & \cdots & \frac{\sum_{t=0}^{T-1} \gamma_t(0)\, \mathbb{1}(Y(t)=6)}{\sum_{t=0}^{T-1} \gamma_t(0)} \\ \frac{\sum_{t=0}^{T-1} \gamma_t(1)\, \mathbb{1}(Y(t)=1)}{\sum_{t=0}^{T-1} \gamma_t(1)} & \cdots & \frac{\sum_{t=1}^{T-1} \gamma_t(1)\, \mathbb{1}(Y(t)=6)}{\sum_{t=0}^{T-1} \gamma_t(1)} \end{bmatrix}$$

Lastly, similar to above

$$\frac{\partial Q(\theta^{(k)}, \theta^{(k+1)})}{\partial \pi_i} = \sum_{z(0)=0}^{1} \cdots \sum_{z(T)=0}^{1} \frac{1}{\pi_i} P(z(0) \cdots z(T) \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

$$= \frac{1}{\pi_i} P(z(0) = i \mid Y(0), \ldots, Y(T), \theta^{(k)}) = \lambda$$

$$\Rightarrow \sum_{i=0}^{1} P(z(0)=i \mid Y(0), \ldots, Y(T), \theta^{(k)}) = \sum_{i=0}^{1} \pi_i \lambda$$

$$\Rightarrow \lambda = \sum_{i=0}^{1} P(z(0)=i \mid Y(0), \ldots, Y(T), \theta^{(k)})$$

$$\Rightarrow \pi_i = \frac{P(z(0)=i \mid Y(0), \ldots, Y(T), \theta^{(k)})}{\sum_{i=0}^{1} P(z(0)=i \mid Y(0), \ldots, Y(T), \theta^{(k)})}$$

HW10

November 6, 2020

1

In this problem we will again revisit the cheating casino model of Problem 1 in homework 8. Please recall the notation from that problem. There are 18 parameters in the cheating casino model, although there are constraints on combinations of these parameters. Let θ represent the vector formed by these parameters. Four parameters form the transition probability matrix of $X(t)$, 6 parameters determine the probabilities of the fair die roll, 6 parameters determine the probabilities of the cheating die roll, and two parameters determine the inital distribution of the hidden state, i.e. $X(0)$. In this problem we'll consider inferring these parameters given the observed states $Y(0), Y(1), ..., Y(200)$.

a

First take a hard EM approach. Assume that we assign the state of $Z(t)$ based on the smoothing probability, $P(Z(t)|Y(0), Y(1), ..., Y(200))$. Derive the formulas for updating the parameters. (In class I derived the formula for the transition probabilities, repeat this derivation, but also provide the derivation for the emission probabilities and the initial state distribution.). Implement the hard EM algorithm and estimate the parameters. Compare to the true parameters.

Let's sample using the true parameters.

[1]:
```
import numpy as np
first = np.repeat(1/6,6)
second = np.concatenate((np.repeat((49/50)/5, 5), np.array([1/50])))

dist_Y = np.concatenate((first, second), axis=0).reshape(2,6)
dist_Y
```

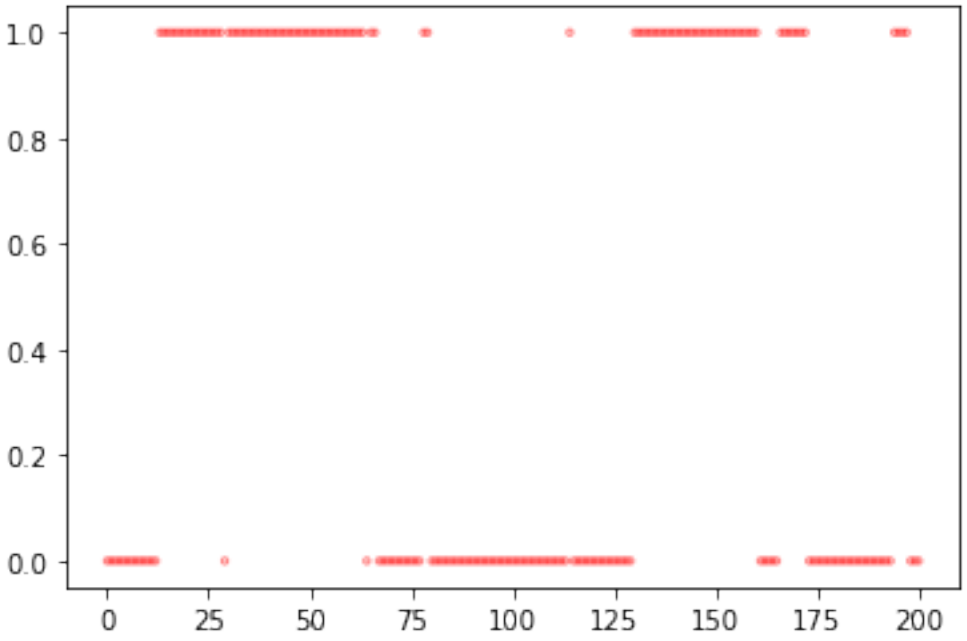
[1]:
```
array([[0.16666667, 0.16666667, 0.16666667, 0.16666667, 0.16666667,
        0.16666667],
       [0.196     , 0.196     , 0.196     , 0.196     , 0.196     ,
        0.02      ]])
```

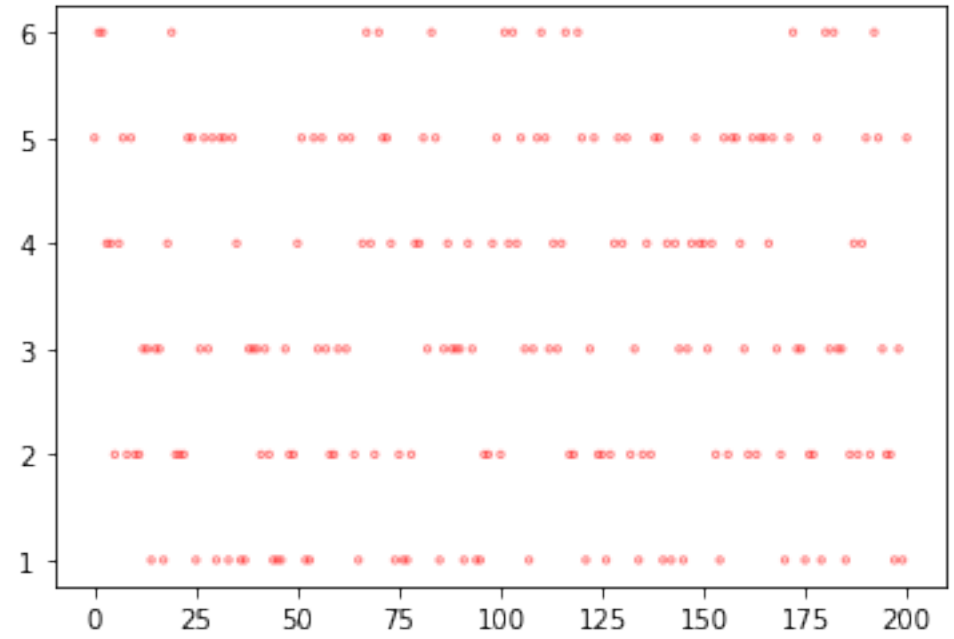
[2]:
```
P = np.array([[0.95,0.05],[0.05,0.95]])
P
```

[2]:
```
array([[0.95, 0.05],
       [0.05, 0.95]])
```

[3]:
```
def SampleCasino(T):
    """
    Args:
        T: scalar
    Returns:
        x, y: matrices of size (T,)
    """
    z_start = 0
    z = [z_start]
    y_start = np.random.choice(range(1,7), p=dist_Y[z[0]])
    y = [y_start]

    for i in range(1, T+1):
        zi = np.random.choice([0, 1], p=P[z[i-1]])
        z.append(zi)
        yi = np.random.choice(range(1,7), p=dist_Y[z[i-1]])
        y.append(yi)

    return np.array(z), np.array(y)
```

[26]:
```
import matplotlib.pyplot as plt
Z, Y = SampleCasino(200)
plt.scatter(range(0, len(Z)), Z, facecolors='none', edgecolors='red', alpha=0.
 ↪5, s=5)
plt.show()

plt.scatter(range(0, len(Y)), Y, facecolors='none', edgecolors='red', alpha=0.
 ↪5, s=5)
plt.show()
```





Now we need the python functions for α_t, β_t written in the previous HW to calculate the smoothing probabilities when needed in the EM algorithm.

[7]:
```
def Forward(y, M, G, pi):
    """Calculates the forward probabilities"""
    alphas = np.zeros([2, 201])
    alphas[0,0] = pi[0]
    alphas[1,0] = pi[1]
    for t in range(1, 201):
        alphas[0, t] = alphas[0, t-1]*G[0, y[t]-1]*M[0,0] + alphas[1, t-1]*G[0,␣
↪y[t]-1]*M[1,0]
        alphas[1, t] = alphas[0, t-1]*G[1, y[t]-1]*M[0,1] + alphas[1, t-1]*G[1,␣
↪y[t]-1]*M[1,1]
        D = alphas[0, t]+alphas[1, t]
        alphas[0, t] = alphas[0, t] / D
        alphas[1, t] = alphas[1, t] / D
    return alphas

def Backward(y, M, G, T=200):
    """Calculates the backward probabilities"""
    betas = np.ones([2, 201])
    betas[0,T-1] = G[0, y[T-1]-1]*M[0, 0]+G[0, y[T-1]-1]*M[0,1]
    betas[1,T-1] = G[1, y[T-1]-1]*M[1, 0]+G[1, y[T-1]-1]*M[1,1]
    for t in np.flip(range(0, T-1)):
        betas[0, t] = betas[0, t+1]*G[0, y[t+1]-1]*M[0,0] + betas[1, t+1]*G[1,␣
↪y[t+1]-1]*M[0,1]
        betas[1, t] = betas[0, t+1]*G[0, y[t+1]-1]*M[1,0] + betas[1, t+1]*G[1,␣
↪y[t+1]-1]*M[1,1]
    return betas

def get_smoothing_probs(betas, alphas):
    """Calculates the smoothing probabilities"""
    return np.multiply(alphas, betas) / np.multiply(alphas, betas).sum(axis=0)
```

Let's define two functions that get our M (transition probability matrix) and G (matrix with the mass of Y given a state of X) matrices when needed in the hard EM algorithm.

[37]:
```
def M_calc(z):
    """ gets the transition probability matrix given the hidden state␣
↪assignments"""
    def counter(i, j):
        """returns the times i appears to the left of j
        for a 1-d array
        Args:
            i: state in
            j: state to
        """
        zt = z[:-1]
        zt1 = z[1:]
        return np.sum(zt1[zt==i]==j)
```

```
    M = [counter(0,0)/np.sum(z[:-1]==0), counter(0,1)/np.sum(z[:-1]==0),
         counter(1,0)/np.sum(z[:-1]==1), counter(1,1)/np.sum(z[:-1]==1)]

    return np.array(M).reshape(2,2)

def G_calc(z, y):
    """ gets the mass of Y given a state of X"""
    def counter(i, j):
        """returns the times i and j appear in the same index position
        for two 1-d arrays
        Args:
            i: state of Z
            j: value of Y
        """
        return np.sum(y[z==i]==j)

    G0 = list(map(counter, np.repeat(0,6), [1,2,3,4,5,6]))/np.sum(z==0)
    G1 = list(map(counter, np.repeat(1,6), [1,2,3,4,5,6]))/np.sum(z==1)

    return np.array([G0, G1]).reshape(2,6)
```

Finally, let's define the functions needed to run the hard EM algorithm itself.

[38]:
```
def hardEM(M, G, pi, y):
    def E_step(M, G, pi, y):
        """updates the assignments based on the state assignments and the␣
 ↪parameters M, G, pi"""
        # in hard EM you are choosing the max
        betas = Backward(y, M, G)
        alphas = Forward(y, M, G, pi)
        smoothing_probs = get_smoothing_probs(betas, alphas)
        z = np.argmax(smoothing_probs, axis=0)
        return z

    def M_step(z, y):
        """updates the parameters M, G, pi based on the assignments"""
        M = M_calc(z)
        G = G_calc(z, y)
        pi = np.array([0,0])
        pi[z[0]] = 1
        return [M, G, pi]

    z = E_step(M, G, pi, y)
    theta = M_step(z, y)
    return z, theta
```

```
def hardEMfunction(theta, y, max_iter=10**3):
    i = 1
    while(i <= max_iter):
        z, theta = hardEM(*theta, y)
        i += 1
    return z, theta
```

[40]:
```
# initializing parameters
import random
M_start = np.random.rand(2,2)
M_start = M_start/M_start.sum(axis=1).reshape(2,-1)
G_start = np.random.rand(2,6)
G_start = G_start/G_start.sum(axis=1).reshape(2,-1)
pi_start = np.random.rand(1,2)
pi_start = pi_start/pi_start.sum(axis=1)
```

[41]:
```
theta = [M_start, G_start, pi_start]
z, theta = hardEMfunction(theta, Y)
M, G, pi = theta
```

[42]:
```
print(M)
```

```
[[0.98333333 0.01666667]
 [0.025      0.975     ]]
```

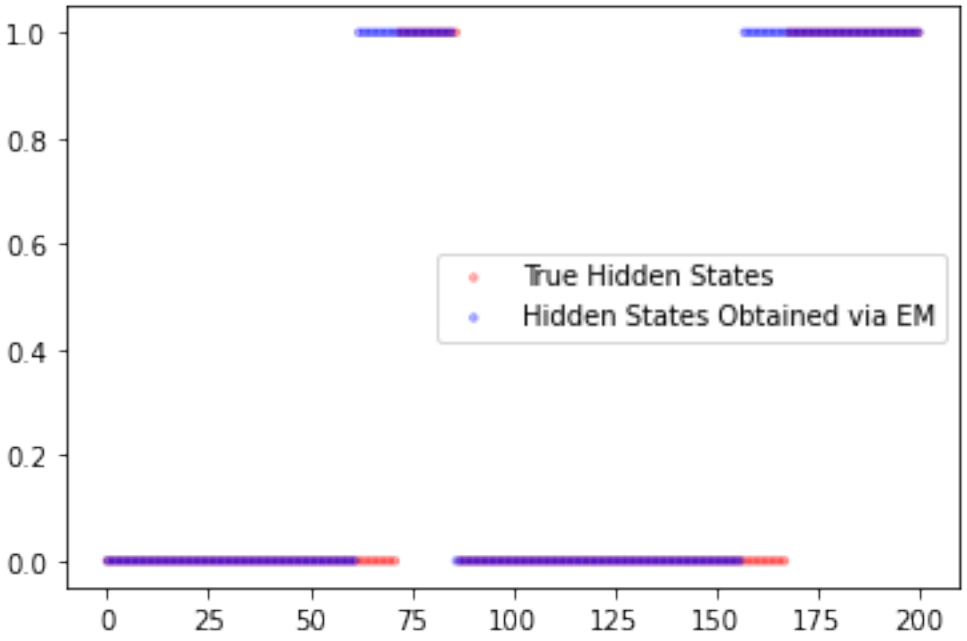
[43]:
```
print(G)
```

```
[[0.15702479 0.2231405  0.19008264 0.14876033 0.15702479 0.12396694]
 [0.1875     0.1625     0.2        0.1625     0.2875     0.        ]]
```

[44]:
```
print(pi)
```

```
[1 0]
```

[15]:
```
plt.scatter(range(0, len(Z)), Z, facecolors='none', edgecolors='red', alpha=0.
 ↪5, s=5, label="True Hidden States")
plt.scatter(range(0, len(z)), z, facecolors='none', edgecolors='blue', alpha=0.
 ↪5, s=5, label="Hidden States Obtained via EM")
plt.legend()
plt.show()
```



We get results that are close enough. You could improve your parameters (M, G, pi) by trying different starting points, cutoffs for the hard assignments, or instead of initializing the assignments try initializing the parameters M, G, and pi.

b

Now repeat using a soft EM approach. Derive the formulas for updating the parameters using the $Q(\theta\,,\theta)$ formalism. Include a derivation for the expression of the two-step smoothing probability, $P(Z(t), Z(t+1)|Y(0), Y(1), ..., Y(200))$, in terms of the forward and backward iterations developed in hw 9. Implement the soft EM algorithm and estimate the parameters. Compare to the true parameters and to your result using the hard EM.

Based on our derivations, let's get functions that calculate our M and G matrices, which make use of ξ_t and γ_t, respectively.

[54]:
```
def M_calc(M, G, y, alphas, betas):
    """gets the transition probability matrix given the smoothing␣
↪probabilities"""
    def xi_calc(i, j):
        return np.sum(np.multiply(alphas[i, :-1], betas[i, 1:], G[j, y[1:
↪]-1])*M[i, j])

    M = [xi_calc(0,0), xi_calc(0,1),
         xi_calc(1,0), xi_calc(1,1)]
    M = np.array(M).reshape(2,2)
```

```
    return M/M.sum(axis=1).reshape(2,-1)

def G_calc(smoothing_probs, y):
    """gets the mass of Y given a state of X"""
    def gamma_calc(i, j):
        return smoothing_probs[j, y==i].sum()

    G0 = list(map(gamma_calc, [1,2,3,4,5,6], np.repeat(0,6)))/np.
 ↪sum(smoothing_probs[0])
    G1 = list(map(gamma_calc, [1,2,3,4,5,6], np.repeat(1,6)))/np.
 ↪sum(smoothing_probs[1])

    return np.array([G0, G1]).reshape(2,6)
```

[55]:
```
def softEM(y, M, G, pi):
    def E_step(y, M, G, pi):
        """updates the smoothing probabilites"""
        alphas = Forward(y, M, G, pi)
        betas = Backward(y, M, G)
        smoothing_probs = get_smoothing_probs(betas, alphas)
        return alphas, betas, smoothing_probs

    def M_step(M, G, y, alphas, betas, smoothing_probs):
        """updates the parameters M, G, pi"""
        M = M_calc(M, G, y, alphas, betas)
        G = G_calc(smoothing_probs, y)
        pi = smoothing_probs[:,0]/np.sum(smoothing_probs[:,0])
        return [M, G, pi]

    alphas, betas, smoothing_probs = E_step(y, M, G, pi)
    theta = M_step(M, G, y, alphas, betas, smoothing_probs)
    return theta, smoothing_probs

def softEMfunction(y, theta, max_iter=10**3):
    i = 1
    while(i <= max_iter):
        theta, smoothing_probs = softEM(y, *theta)
        i += 1
    return theta, smoothing_probs
```

[56]:
```
# using same initializations as for HardEM
import random
M_start = np.random.rand(2,2)
M_start = M_start/M_start.sum(axis=1).reshape(2,-1)
G_start = np.random.rand(2,6)
G_start = G_start/G_start.sum(axis=1).reshape(2,-1)
```

```
pi_start = np.random.rand(1,2)
pi_start = pi_start/pi_start.sum(axis=1)
```

[57]:
```
theta = [M_start, G_start, pi_start]
theta, smoothing_probs = softEMfunction(Y, theta)
M, G, pi = theta
```

[58]:
```
print(M)
```

```
[[0.95 0.05]
 [0.05 0.95]]
```

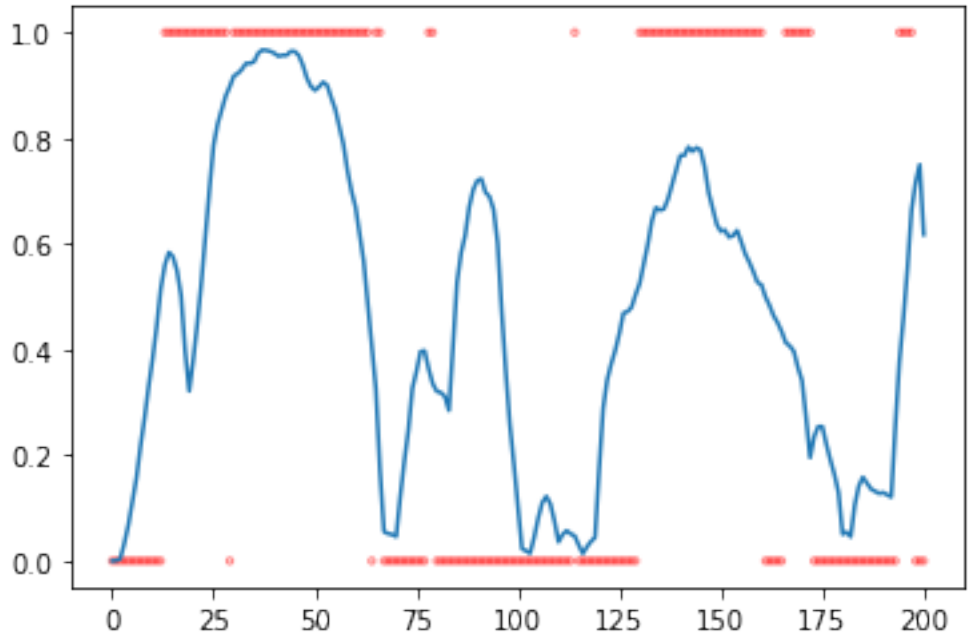
[59]:
```
print(G)
```

```
[[0.12035331 0.21007594 0.16328566 0.17441108 0.2035287  0.12834531]
 [0.22481321 0.1863782  0.22909456 0.13121039 0.21514434 0.0133593 ]]
```

[60]:
```
print(pi)
```

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

[61]:
```
plt.scatter(range(0, len(Z)), Z, facecolors='none', edgecolors='red', alpha=0.
↪5, s=5)
plt.plot(smoothing_probs[1,:])
plt.show()
```



SoftEM seems to do good. SoftEM is able to get us the exact transition probability matrix (unlike HardEM) and the exact initial distribution (HardEM is able to do so too). In terms of the matrix G, consisting of the mass of Y given the Z states, SoftEM does a bit better: for example, SoftEM assigns a probability of 0.013, close to 0.02, to the probability you roll a 6 given the die is not fair – HardEM assigns a probability of 0.

So overall, SoftEM seems to perform better. But this is given the random starting point I chose – the EM algorithm in general is dependent on the starting point and there might be starting points for which HardEM might perform better than SoftEM.

2

Let $B(t)$ be Brownian motion with variance σ^2.

a

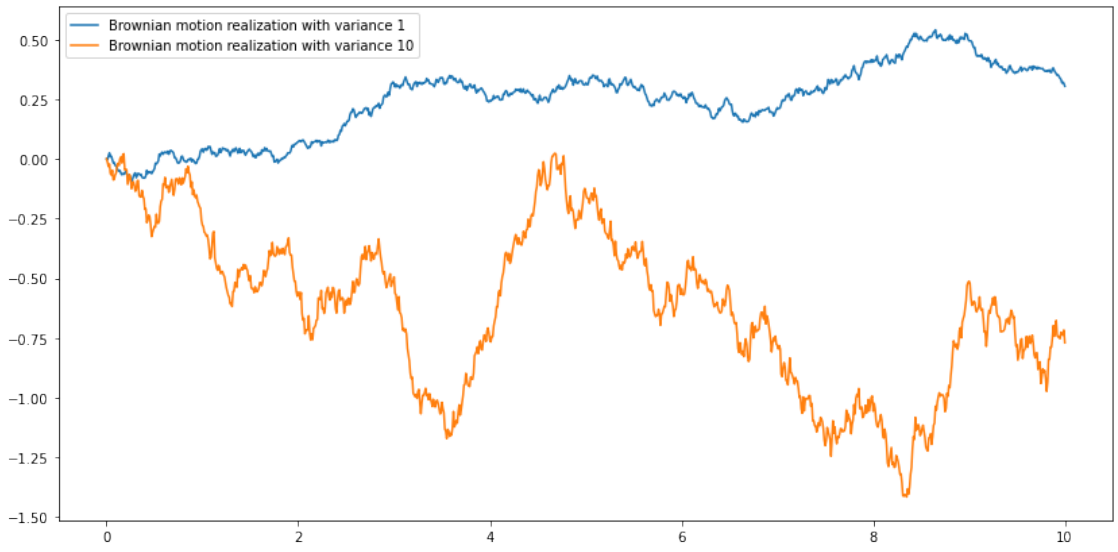
To get a feel for the sample paths of Brownian motion, generate a sample of $B(t)$ for i) $\sigma^2 = 1$ and ii) $\sigma^2 = 10$ up to time $t = 10$ using a grid of width .01.

We know that $B(0) = 0$ and that $B(t_{i+1}) - B(t_i) \sim N(0, \sigma^2 \Delta t)$ where $\Delta t = 0.01$ and $i = 0, \ldots, \frac{t}{\Delta t} - 1 = \frac{10}{0.01} - 1 = 999$.

[46]:
```
def sample_brownian_motion(variance, width, T):
    """Samples paths of brownian motion
    Args:
        variance: variance of the Brownian motion
        width: increment length
        T: time
    Returns:
        sample: array of shape (T/width+1,)

    """
    increments = np.concatenate(([0],
                                 normal(loc=0, scale=width*np.sqrt(variance),␣
↪size=int(T/width))))

    sample = np.cumsum(increments)
    return sample
```

[51]:
```
from numpy.random import normal
import numpy as np
import matplotlib.pyplot as plt

sample = sample_brownian_motion(1, 0.01, 10)
sample2 = sample_brownian_motion(10, 0.01, 10)
```

[52]:
```
plt.figure(figsize=(14,7))
plt.plot(np.arange(0,10.01,0.01), sample, label="Brownian motion realization␣
 ↪with variance 1")
plt.plot(np.arange(0,10.01,0.01), sample2, label="Brownian motion realization␣
 ↪with variance 10")
plt.legend()
plt.show()
```



b

Let $t_1 < t_2 < t_3$**.**

i. Calculate $E[B(t_1)B(t_2)]$**.**

ii. Calculate $E[B(t_1)B(t_2)B(t_3)]$**.**

Attached.

[ ]:

a) We know $B(t_2) = B(t_1) + (B(t_2) - B(t_1))$

and using the fact that $B(t_1)$ and $B(t_2) - B(t_1)$ are independent with mean 0

$$E[B(t_1)B(t_2)] = E[B(t_1)(B(t_1) + (B(t_2) - B(t_1)))]$$

$$= E[B(t_1)^2] + E[B(t_1)(B(t_2) - B(t_1))]$$

$$= E[B(t_1)^2] + E[B(t_1)]E[B(t_2) - B(t_1)]$$

$$= \text{var}(B(t_1))$$

$$= \text{var}(B(t_1) - B(0))$$

$$= \sigma^2 t_1$$ ∎

b) Let's write $B(t_3) = B(t_1) + (B(t_2) - B(t_1)) + (B(t_3) - B(t_2))$ and $B(t_2) = B(t_1) + (B(t_2) - B(t_1))$ and reason as above:

$$E\left[B(t_1)B(t_2)B(t_3)\right]$$

$$= E\left[B(t_1)\left(B(t_1) + (B(t_2) - B(t_1))\right)\left(B(t_1) + (B(t_2) - B(t_1)) + (B(t_3) - B(t_2))\right)\right]$$

$$= E\left[\left(B(t_1)^2 + B(t_1)(B(t_2) - B(t_1))\right)\left(B(t_1) + (B(t_2) - B(t_1)) + (B(t_3) - B(t_2))\right)\right]$$

$$= E\left[B(t_1)^3 + B(t_1)^2(B(t_2) - B(t_1)) + B(t_1)^2(B(t_3) - B(t_2))\right.$$

$$\left. + B(t_1)^2(B(t_2) - B(t_1)) + B(t_1)(B(t_2) - B(t_1))^2 + B(t_1)(B(t_2) - B(t_1))(B(t_3) - B(t_2))\right]$$

$$= E\left[B(t_1)^3\right] + E\left[B(t_1)^2\right]E\left[B(t_2) - B(t_1)\right] + E\left[B(t_1)^2\right]E\left[B(t_3) - B(t_2)\right]$$

$$+ E\left[B(t_1)^2\right]E\left[B(t_2) - B(t_1)\right] + E\left[B(t_1)\right]E\left[(B(t_2) - B(t_1))^2\right]$$

$$+ E\left[B(t_1)\right]E\left[B(t_2) - B(t_1)\right]E\left[B(t_3) - B(t_2)\right]$$

$$= 0$$

- Note that $E\left[B(t_1)^3\right] = 0$ b/c $B(t_1)^3 f_{B(t_1)}(x)$ is an odd function
- $B(t_i)$ and $B(t_{i+1}) - B(t_i)$ are independent $\Rightarrow$ any function of $B(t_i)$ not involving $B(t_{i+1}) - B(t_i)$ and any function of $B(t_{i+1}) - B(t_i)$ not involving $B(t_i)$ are independent too
- Lastly, we know that $B(t_{i+1}) - B(t_i) \sim N(0, \sigma^2(t_{i+1} - t_i))$