(a):

Using the following code, we can obtain the p(x0) with maximum likelihood:

```
from sklearn.preprocessing import normalize
import numpy as np
p0 = np.bincount([row[0] for row in x], minlength = 8)
p0 = normalize(p0, axis=1, norm='l1')
```

The results are listed in Table 1.

Table 1: Initial statement distribution

x_0	0	1	2	3	4	5	6	7
$p(x_0)$	0.8765	0	0	0	0.0359	0	0.0876	0

(b) & (c):

The following codes estimated the transition matrix T and emission matrix O with maximum likelihood:

```
from sklearn.preprocessing import normalize
import numpy as np

T = np.zeros((dx,dx), dtype=np.double)
O = np.zeros((dx,do), dtype=np.double)
p0 = np.bincount([row[0] for row in x], minlength = 8)

for i in range(len(x)):
   # Counting the frequency of each state change
   for j in range(x[i].size-1):
      T[x[i][j]][x[i][j+1]] = T[x[i][j]][x[i][j+1]] + 1
   # Counting the frequency of each pair of
   # observation and hidden state
   for j in range(x[i].size):
      O[x[i][j]][o[i][j]] = O[x[i][j]][o[i][j]] + 1

# Normalize T and O
T = normalize(T, axis=1, norm='l1')
O = normalize(O, axis=1, norm='l1')
```

Output is as below:

```
T[0][0] = 0.602215434439
T[0][1] = 0.034055864277
T[0][2] = 0.120877341479
T[0][3] = 0.0368453980054
T[0][4] = 0.102646008906
T[1][0] = 0.460651516848
T[1][1] = 0.0219411172059
T[1][2] = 0.0417552893765
T[1][3] = 0.0252994514721
T[1][4] = 0.0452255681182
T[2][0] = 0.0853601578646
T[2][1] = 0.00356129514379
T[2][2] = 0.814666697804
T[2][3] = 0.00489824036991
T[2][4] = 0.00541783916958
T[3][0] = 0.161422754652
T[3][1] = 0.00777332839789
T[3][2] = 0.0213345896288
T[3][3] = 0.702897331494
T[3][4] = 0.0312952182253
T[4][0] = 0.0562093935588
T[4][1] = 0.000589229504892
T[4][2] = 0.000755919430618
T[4][3] = 0.00338806965313
T[4][4] = 0.918310306865
```

Notation: T[i][j] = P[xt=j|xt-1=i]

```
O[0][0] = 0.0594699697537
O[0][1] = 0.0145254212876
O[0][2] = 0.0670891545441
O[0][3] = 0.0498703730376
O[0][4] = 0.0319530462336
O[1][0] = 0.0529497369305
O[1][1] = 0.0241800067167
O[1][2] = 0.0470166797269
O[1][3] = 0.039180566439
```

```
O[1][4] = 0.0509347363708
O[2][0] = 0.0618264189718
O[2][1] = 0.0192543465315
O[2][2] = 0.0335929381269
O[2][3] = 0.04737690179
O[2][4] = 0.0559356865126
O[3][0] = 0.0920012114278
O[3][1] = 0.0122488811118
O[3][2] = 0.0758824915032
O[3][3] = 0.0923040683784
O[3][4] = 0.0397079112966
O[4][0] = 0.116337938627
O[4][1] = 0.0116062706424
O[4][2] = 0.0478477617032
O[4][3] = 0.0895745142733
O[4][4] = 0.0400521002931
```

Notation: Ob[i][k] = P[ot=k|xt=i]

The stationary distribution:

```
p[0] = 0.178591936571
p[1] = 0.0117113618502
p[2] = 0.224954493376
p[3] = 0.038969821529
p[4] = 0.334418947825
p[5] = 9.81586644726e-05
p[6] = 0.210937144252
p[7] = 0.000318135933071
```

(d):

To determine the length of sequence, we can try to find out the distribution of the last several observations. The length of the segment can be decided by EM. Then we can have the probability of whether a sequence will end or not by the permutation of the last several states.

A more simple approach is to give each state a Bernoulli variable, which tells whether a state is in the last of the sequence or not. By estimating the probability of a hidden state to be the end a sequence, we can predict the sequence's length.

(e):

```
def markovMarginals(x,o,p0,Tr,Ob):
   '''Compute p(o) and the marginal probabilities p(x_t|o)
   for a Markov model defined by P[xt=j|xt-1=i] = Tr(i,j)
   and P[ot=k|xt=i] = Ob(i,k) as numpy matrices'''
   dx,do = Ob.shape   # if a numpy matrix
   L = len(o)
   f = np.zeros((L,dx))
   r = np.zeros((L,dx))
   p = np.zeros((L,dx))
   f[0,:] = p0 * Ob[:,o[0]] # compute initial forward message
   from math import log
   log_pO = log(f[0,:].sum()) # update probability of sequence so far
   f[0,:] /= f[0,:].sum() # normalize (to match definition of f)
   for t in range(1,L):    # compute forward messages
      f[t,:] = Ob[:,o[t]]*np.dot(f[t-1,:], Tr)
      log_pO += log(f[t,:].sum())
      f[t,:] /= f[t,:].sum()
   r[L-1,:] = 1.0  # initialize reverse messages
   p[L-1,:] = f[L-1,:]*r[L-1,:]  # and marginals
   for t in range(L-2,-1,-1):
      r[t,:] = np.dot((r[t+1,:]*Ob[:,o[t+1]]),np.transpose(Tr))
      r[t,:] /= r[t,:].sum()
      p[t,:] = f[t,:]*r[t,:]
      p[t,:] /= p[t,:].sum()
   return log_pO, p, f, r
```

It can be tested with a simple observation $(0, 1, 2)$ with transmission matrix, emission matrix and initial prob as follows:

$$Tr = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$Ob = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$p_0 = (1, 0, 0)$$

Filename: “12as.txt”

$$p\left(x_6^{(0)}\middle|o^{(0)}\right) =$$

(1.02101704e-01,
5.60632171e-03,
1.54825441e-01,
2.52220214e-02,
6.45436063e-01,
3.70925988e-06,
6.67175945e-02,
8.71440954e-05)

Filename: “16pk.txt”

$$p\left(x_9^{(2)}\middle|o^{(2)}\right) =$$

(1.92279325e-01,
2.16531983e-02,
2.04049763e-01,
5.81202218e-02,
2.37895589e-01,
2.72872172e-04,
2.85012935e-01,
7.16096328e-04)

Filename: “1914.txt”
$log\ p(o^{(4)}) = -493.7366670340388$

(f):

Here’s my code of predicting most likely statement by given observation:

```
def most_likely_statement(o, p0, Tr, Ob):
    dx, do = Ob.shape
    L = len(o)
    f = np.zeros((L,dx), dtype=np.double)
    r = np.zeros((L,dx), dtype=np.int)
    f[0] = p0*Ob[:, o[0]]
    for t in range(1,L):   # compute forward messages
        for k in range(dx):
            f[t][k] = max(f[t-1]*np.transpose(Tr)[k]*Ob[:, o[t]])
            r[t][k] = np.argmax(f[t-1]*np.transpose(Tr)[k]*Ob[:, o[t]])
    ord = list()
    ord.append(np.argmax(f[L-1]))
    for t in range(L-2, -1, -1):
        ord.insert(0, r[t+1][ord[0]])
    return f,r,ord
```