COMP0080 Graphical Models
Assignment 2

Daniel May
ucabdd3@ucl.ac.uk

Olivier Kraft
ucabpk0@ucl.ac.uk

Daniel O'Hara
daniel.o'hara.20@ucl.ac.uk

Stephen O'Brien
stephen.o'brien.20@ucl.ac.uk

Umais Zahid
umais.zahid.16@ucl.ac.uk

December 15, 2020

Contents

1 Question 1

(a)

The parameters of this model are the elements of the 3x3 transition matrix A where $A_{ij} = P(v_{t+1} = i | v_t = j)$ and the initial distribution between states $\alpha \in \mathbb{R}^3$ where $\alpha_i = P(v_1 = i)$.

To get the maximum likelihood estimates for these parameters, first define a count matrix C, where

$$C \in \mathbb{R}^{3\times} \tag{1}$$

$$C_{ij} = \sum_{n=1}^{N} \sum_{t=1}^{T-1} \mathbb{I}[X_n(t+1) = i, X_n(t) = j] \tag{2}$$

The maximum likelihood estimates are then

$$\hat{A}_{i,j} = \frac{C_{ij}}{\sum_{j=1}^{3} C_{ij}} \tag{3}$$

$$\hat{\alpha}_i = \frac{\sum_{n=1}^{N} \mathbb{I}[X_n(1) = i]}{N} \tag{4}$$

Applying this to the weather station data gives the maximum likelihood parameters for the model as

$$\hat{A} = \begin{pmatrix} 0.6998 & 0.1008 & 0.1007 \\ 0.3012 & 0.6995 & 0.1987 \\ 0.000 & 0.1997 & 0.7006 \end{pmatrix} \tag{5}$$

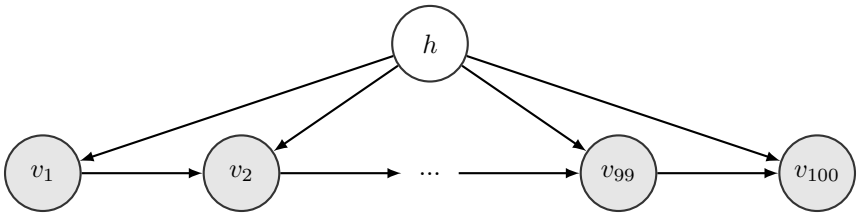
$$\hat{\alpha} = \begin{pmatrix} 0.308 \\ 0.512 \\ 0.18 \end{pmatrix} \tag{6}$$

Python Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

V_csv = pd.read_csv("meteo0.csv", header = None, delimiter = " ")
V = np.array(V_csv)

A = np.zeros((3, 3))
N = np.zeros((3, 3))
a = np.zeros((3, 1))
for i in range(500):
  a[V[i,0]] += 1
  for j in range(99):
    N[V[i,j],V[i,j+1]] += 1

A = N/N.sum(axis = 1).reshape(3, 1)
a = a/a.sum()
print(A.round(4));print(a.round(4))
```

2 Question 2

(a)



Notation
we denote $H = \{0, 1, 2\}$ as the set of weather stations from which the N sequences are generated. We say that the sequence v^n is generated by $h \in H$ if $x^n = h$. These will later act as hidden variables. We also set

$$v = \{v^n\}_{n=1}^N \quad v^n = \{v_t^n\}_{t=1}^T$$

We define the parameters of the model as follows. A^h is a 3×3 matrix made up of entries

$$A_{i,j}^h = \mathrm{P}\left(v_{t+1}^n = i | v_t^n = j, x^n = h\right) \quad \text{for } h \in H$$

We also have the parameter associated to the initial point in each sequence

$$a_i^h = \mathrm{P}(v_1^n = i | x^n = h) \quad \text{for } h \in H$$

Finally, we have the parameter associated to the probability of any row being generated by the weather station 'h'.

$$q(h) = \mathrm{P}(x^n = h)$$

For ease of notation, we set $\theta = \{\theta^h\}_{h \in H}$ and $\theta^h = \{A^h, a^h\}$

Expectation

The first step in the E-M algorithm is to maximise $\mathrm{P}(v|\theta)$ w.r.t θ. From the notes, we know that by including the auxillary function $q(h)$ we can instead maximise a lower bound of the following expression w.r.t θ

$$\underset{\theta}{\operatorname{argmax}} \log(\mathrm{P}(v|\theta)) = \underset{\theta}{\operatorname{argmax}} \log\left(\sum_h \frac{q(h)}{q(h)} \mathrm{P}(v, h|\theta)\right)$$

From the notes, we know that this is achieved by setting $q(h) = P(h|v, \theta)$

Maximisation

From the notes again, we know that

$$\theta_{\text{new}} = \underset{\theta}{\operatorname{argmax}} < \log(P(v, h|\theta) >_{q(h)}$$

We notice that

$$\begin{aligned}\log(P(v,h|\theta)) &= \log(P(v|h,\theta)) + \log(P(h|\theta)) \\ &= \log\left(\prod_{n=1}^{N} P(v^n|x^n,\theta)\right) + \log\left(\prod_{n=1}^{N} P(x^n|\theta)\right) \\ &= \sum_{n=1}^{N} \log\left(P(v^n|x^n,\theta)\right) + \sum_{n=1}^{N} \log\left(P(x^n|\theta)\right) \\ &= \sum_{h\in H}\left[\sum_{n=1}^{N}[x^n=h]\log\left(P(v^n|x^n,\theta)\right) + \sum_{n=1}^{N}[x^n=h]\log\left(P(x^n|\theta)\right)\right]\end{aligned}$$

Thus, the expression that we are maximising w.r.t θ is

$$\sum_{h\in H} P(h|v,\theta)\left(\sum_{h\in H}\left[\sum_{n=1}^{N}[x^n=h]\log\left(P(v^n|x^n,\theta)\right) + \sum_{n=1}^{N}[x^n=h]\log\left(P(x^n|\theta)\right)\right]\right)$$

We take a look at $P(v^n|x^n,\theta)$. Intuitively, we know that this will look as follows

$$P(v^n|x^n,\theta) = \prod_{i=0}^{2} a_i^{[v_1^n=i]} \prod_{i,j\in 0,1,2} A_{i,j}^{h\,\sum_{t=1}^{T}[v_t^n=i][v_{t-1}^n=j]}$$

Plugging this in, we get

$$\sum_{h\in H} P(h|v,\theta)\left(\sum_{h\in H}\left[\sum_{n=1}^{N}[x^n=h]\log\left(\prod_{i=0}^{2} a_i^{[v_1^n=i]} \prod_{i,j\in 0,1,2} A_{i,j}^{h\,\sum_{t=1}^{T}[v_t^n=i][v_{t-1}^n=j]}\right) + \sum_{n=1}^{N}[x^n=h]\log\left(P(x^n|\theta)\right)\right]\right)$$

We must now maximise this with respect to each element of θ. As seen in David Barber's book, the following parameter updating functions do exactly that

$$\begin{aligned}a_i^{h,new} &= \frac{\sum_{n=1}^{N} P(x_n=h|x_n,\theta^{old})[x_1^n=i]}{\sum_{k=0}^{2}\sum_{n=1}^{N} P(x_n=h|x_n,\theta^{old})[x_1^n=k]} \\ A_{i,j}^{h,new} &= \frac{\sum_{n=1}^{N} P(x_n=h|x_n,\theta^{old})\sum_{t=1}^{T}[v_t^n=i][v_{t-1}^n=j]}{\sum_{k=0}^{2}\sum_{n=1}^{N} P(x_n=h|x_n,\theta^{old})\sum_{t=1}^{T}[v_t^n=k][v_{t-1}^n=j]} \\ q(h)^{new} &= \frac{\sum_{n=1}^{N} P(h|v^n,\theta^{old})}{\sum_{h\in H}\sum_{n=1}^{N} P(h|v^n,\theta^{old})}\end{aligned}$$

Where we normalise the following matrix for each of it's n rows

$$P(h|v^n,\theta^{old}) \propto q^{old}(h)P(v^n|h)$$

This procedure is repeated until convergence.

(b)

The probability distribution of the hidden variables, z is

$$\alpha = \begin{pmatrix} 0.192 \\ 0.296 \\ 0.512 \end{pmatrix} \tag{7}$$

$$\tag{8}$$

The three transition matrices are

$$A_1 = \begin{pmatrix} 0.389 & 0.329 & 0.511 \\ 0.118 & 0.523 & 0.055 \\ 0.493 & 0.147 & 0.435 \end{pmatrix} \tag{9}$$

$$A_2 = \begin{pmatrix} 0.06 & 0.116 & 0.426 \\ 0.159 & 0.324 & 0.419 \\ 0.781 & 0.56 & 0.155 \end{pmatrix} \tag{10}$$

$$A_3 = \begin{pmatrix} 0.07 & 0.24 & 0.409 \\ 0.346 & 0.226 & 0.539 \\ 0.584 & 0.534 & 0.052 \end{pmatrix} \tag{11}$$

$$\tag{12}$$

The three initial distributions are

$$\pi_1 = \begin{pmatrix} 0.427 \\ 0.573 \\ 0.0 \end{pmatrix} \tag{13}$$

$$\pi_2 = \begin{pmatrix} 0.184 \\ 0.413 \\ 0.403 \end{pmatrix} \tag{14}$$

$$\pi_3 = \begin{pmatrix} 0.464 \\ 0.336 \\ 0.201 \end{pmatrix} \tag{15}$$

$$\tag{16}$$

The log-likelihood for these parameters is -93.65858757.

The first ten rows of the posterior distribution of hidden variable are

$$P(h|V,\theta) = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.981 & 0.019 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.005 & 0.995 \\ 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.002 & 0.998 \\ 0.0 & 0.015 & 0.985 \end{pmatrix} \tag{17}$$

$$\tag{18}$$

(c)

For the EM algorithm, the initial guesses are important. As explained in (d), a uniform initial distribution doesn't work at all. Additionally, a problem can arise with the EM algorithm where an entire estimated

distribution explains only a few outliers very well, leaving another distribution explaining the data from two true distributions. This occurs as the outlier overfit has very high likelihood, so no gradual adjustment towards a more accurate estimate would increase the likelihood in the short term, which the EM algorithm must do. However, we did not experience this issue in this assignment.

Our initialisation strategy was to sample from a standard uniform distribution for all parameters and normalise the distributions. This worked, giving us an equal likelihood on each iteration.

Numerical issues with underflow were a problem when calculating the unnormalised likelihoods, $P(h_i|v)$, as underflow issues would occur, causing all values to be zero. This was corrected for by instead calculating the log-likelihood of each individual $P(h_i|v_j)$ and calculating the likelihood as

$$P(h_i|v) = \prod_{j=1}^{500} P(h_i|v_j) = exp(\sum_{j=1}^{500} log(P(h_i|v_j))) \tag{19}$$

This was then normalised to be a probability and prevented underflow issues.

(d)

While initialising parameters uniformly may seem intuitive, it does not work with the EM algorithm. The problem with this can be seen by examining the methods for updating each of the parameter estimates.

$\log(P(h|v,\theta))$ would also be uniform distribution, with $\log(P(h_i|v,\theta)) = \frac{1}{3} \forall i \in 1,2,3$ for the first iteration, since the parameters for each of the 3 distributions are identical.

This posterior probability is then used to update the relevant transition probabilities and initial probabilities. However, since the posterior values are equal for each hidden variable value.

The update functions become

$$\begin{aligned}
a_i^{h,new} &= \frac{\sum_{n=1}^{N} \frac{1}{3}[x_1^n = i]}{\sum_{k=0}^{2} \sum_{n=1}^{N} \frac{1}{3}[x_1^n = k]} \\
A_{i,j}^{h,new} &= \frac{\sum_{n=1}^{N} \frac{1}{3} \sum_{t=1}^{T} [v_t^n = i][v_{t-1}^n = j]}{\sum_{k=0}^{2} \sum_{n=1}^{N} \frac{1}{3} \sum_{t=1}^{T} [v_t^n = k][v_{t-1}^n = j]} \\
q(h)^{new} &= \frac{\sum_{n=1}^{N} \frac{1}{3}}{\sum_{h \in H} \sum_{n=1}^{N} \frac{1}{3}} = \frac{1}{3}
\end{aligned}$$

All of these updates are identical for each of the transition matrix and for each initial distribution. Therefore, the same update is applied to each for the first iteration, resulting in an identical distributions for each state of the hidden variable.

$q(h)$ also remains uniformly distributed, so no further update step beyond the first actually affects any parameter estimate, leaving us with identical estimates for each set of parameters.

For this case, the following estimates would have been made:

$$A_h = \begin{pmatrix} 0.16 & 0.188 & 0.452 \\ 0.258 & 0.284 & 0.385 \\ 0.582 & 0.528 & 0.162 \end{pmatrix} \tag{20}$$

$$\alpha_h = \begin{pmatrix} 0.374 \\ 0.404 \\ 0.222 \end{pmatrix} \tag{21}$$

Python Code

```
V = pd.read_csv("meteo1.csv", header = None, delimiter = " ")
X = np.array(V)
```

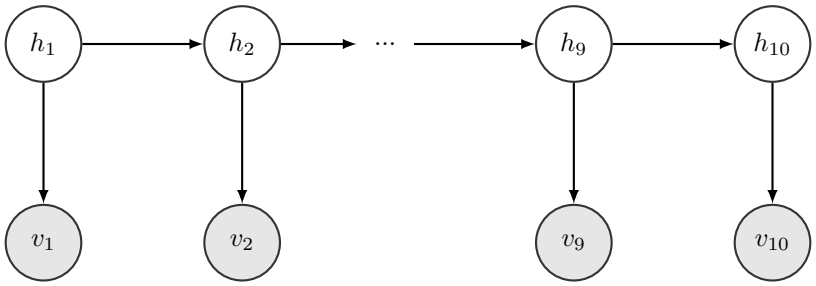
```
# calculate N(S)
# number of transitions between each states in each sequence
C = [np.zeros((3, 3)) for i in range(500)]
for i in range(500):
  for j in range(99):
    C[i][X[i,j+1],X[i,j]] += 1

N = 500 # sequences
K = 3 # 3 values for h
D = 3 # 3 values for each day's weather
L = 100 # length of sequence


B = [np.random.normal(0, 1, (3, 3))**2 for i in range(3)]
A = [i/i.sum(axis = 1).reshape(3, 1) for i in B]
alpha = np.random.normal(0, 1, (3, 1))**2
alpha /= alpha.sum()
pi = [np.random.normal(0, 1, (3, 1))**2 for i in range(3)]
pi = [i/i.sum() for i in pi]

# Uniform Parameters test
# A = [np.ones((3, 3))/3 for i in range(3)]
# alpha = np.ones((3, 1))/3
# pi = [np.ones((3, 1)) for i in range(3)]
likelihoods = []
for i in range(500):
  for j in range(99):
    C[i][X[i,j],X[i,j+1]] += 1
#EM Algorithm
'''
Note this calculates the transpose of the matrices given in the report
so A[i,j] = P[x_{n+1+ = j|x_{n} = i]
'''
for iter in range(10): # enough to converge
  P = np.zeros((N, K), dtype = np.longdouble)
  for n in range(N):
    for k in range(K):
      P[n,k] = np.log(alpha[k] * pi[k][X[n,0]])
      for l in range(L - 1):
        P[n,k] += np.log(A[k][X[n,l], X[n,l+1]])
  P = np.exp(P)/np.exp(P).sum(axis = 1).reshape(500, 1)

  # update alpha
  new_alpha = np.zeros_like(alpha)
  for k in range(K):
    new_alpha[k] = P[:,k].sum()
  new_alpha /= new_alpha.sum()

  #update A
  new_A = [np.zeros((3, 3)) for i in range(3)]
  for k in range(K):
    for i in range(D):
      for j in range(D):
        for n in range(N):
```

```
        new_A[k][i,j] += P[n,k]*C[n][i,j]
  new_A[k] /= new_A[k].sum(axis = 1).reshape(3, 1)

#update pi
new_pi = [np.zeros_like(pi[0]) for i in range(3)]
for k in range(K):
  for i in range(D):
    new_pi[k][i] = np.sum(P[:,k]*(X[:,0]==i))
  new_pi[k] /= new_pi[k].sum()


A, alpha, pi = new_A.copy(), new_alpha.copy(), new_pi.copy()
  # calculate log-likelihood\
LL = 0
for i in range(N):
  probi = 0
  for k in range(K):
    p = alpha[k] * pi[k][X[i,0]]
    for l in range(L-1):
      p *= A[k][X[i,l], X[i,l+1]]
    probi += p
LL += np.log(probi)
likelihoods.append(LL)
```

3 Question 3

(a)

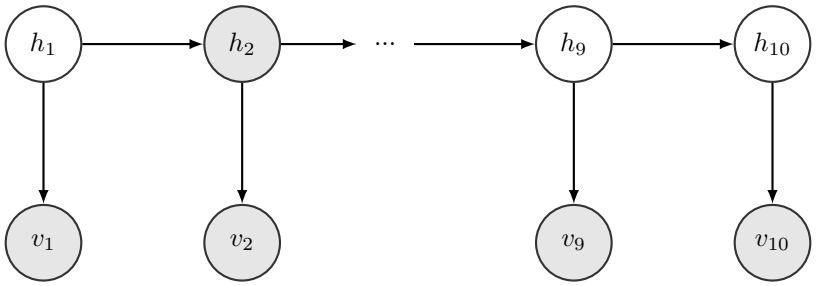
Draw the graphical model corresponding to this setup:



h_1 and h_{10} are d-connected, so, based on the graph, we cannot say that they are independent.
Note: we ignore here the fact that h_1 is given. Otherwise, h_1 would necessarily be independent of h_{10}.

(b)

The observed variable can only take value 1 when the hidden variable is in state 0. Therefore, we can conclude from the observed sequence that h_2 and h_7 must be equal to 0. As a result, h_2 and h_7 are now in the conditioning set, and therefore h_1 and h_{10} are d-separated and therefore independent.



(c)

To compute $p(h_t|v_{1:10})$, we observe first that:

$$p(h_t|v_{1:10}) \propto p(h_t, v_{1:10})$$

.

We therefore focus on computing $p(h_t, v_{1:10})$, which we can then normalise to obtain $p(h_t|v_{1:10})$.

$$\begin{aligned} p(h_t, v_{1:10}) &= p(v_{t+1:10}|h_t, v_{1:t})p(h_t, v_{1:t}) \\ &= p(v_{t+1:10}|h_t)p(h_t, v_{1:t}) \\ &= \alpha(h_t) \cdot \beta(h_t) \end{aligned} \tag{22}$$

where we define:

$$\alpha(h_t) := p(h_t, v_{1:t})$$
$$\beta(h_t) := p(v_{t+1:10}|h_t) \quad \text{for } 1 \leq t \leq 9, \text{ and } \beta(h_{10}) = 1$$

- Computing $\alpha(h_1)$ to $\alpha(h_{10})$

– $\alpha(h_1)$

$$\begin{aligned}\alpha(h_1) &= p(v_1 = 0, h_1) \\ &= p(v_1 = 0|h_1)p(h_1) \\ &= \begin{pmatrix}0.3\\0.5\end{pmatrix} \times \begin{pmatrix}1\\0\end{pmatrix} \\ &= \begin{pmatrix}0.3\\0\end{pmatrix}\end{aligned}$$

– Computing $\alpha(h_2)$

$$\begin{aligned}\alpha(h_2) &= p(v_2 = 1|h_2) \sum_{h_1} p(h_2|h_1) \times \alpha(h_1) \\ &= \begin{pmatrix}0.6\\0\end{pmatrix} \cdot \left[\begin{pmatrix}0.5 & 0.8\\0.5 & 0.2\end{pmatrix} \times \begin{pmatrix}0.3\\0\end{pmatrix} \right] \\ &= \begin{pmatrix}0.6\\0\end{pmatrix} \times \begin{pmatrix}0.15\\0.15\end{pmatrix} \\ &= \begin{pmatrix}0.09\\0\end{pmatrix}\end{aligned}$$

– We compute $\alpha(h_3)$ to $\alpha(h_{10})$ recursively (see code in appendix), using the same approach as for $\alpha(h_2)$ and the relevant value of v_t as given:

$$\alpha(h_t) = p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}) \times \alpha(h_{t-1})$$

- Computing $\beta(h_9)$ to $\beta(h_1)$
  - As indicated above $\beta(h_{10}) = 1$
  - Computing $\beta(h_9)$:

$$\begin{aligned}\beta(h_9) &= \sum_{h_{10}} \beta(h_{10}) \cdot p(v_{10} = 0|h_{10}) \cdot p(h_{10}|h_9) \\ &= \begin{pmatrix}0.3 & 0.5\end{pmatrix} \times \begin{pmatrix}0.5 & 0.8\\0.5 & 0.2\end{pmatrix}\end{aligned}$$

  – Computing $\beta(h_8)$ to $\beta(h_1)$: We compute $\beta(h_8)$ to $\beta(h_1)$ recursively (see code in appendix), using the same approach as for $\beta(h_9)$:

$$\begin{aligned}\beta(h_{t-1}) &= \sum_{h_t} \beta(h_t) \cdot p(v_t|h_t) \cdot p(h_t|h_{t_1}) \\ &= \beta(h_t) \left[p(v_t|h_t) \times \begin{pmatrix}0.5 & 0.8\\0.5 & 0.2\end{pmatrix} \right]\end{aligned}$$

- Computing $p(h_t, v_{1:10})$:

  Using the values of α and β, we can compute $p(h_t, v_{1:10})$ ($t \in [1, 10]$) as described above. We then normalise the probabilities to obtain the conditional probabilities $p(h_t|v_{1:10})$:

t	$p(h_t\|v_{1:10}) = 0$	$p(h_t\|v_{1:10}) = 1$
1	1	0
2	1	0
3	0.502	0.498
4	0.298	0.702
5	0.732	0.268
6	0.171	0.829
7	1	0
8	0.488	0.512
9	0.341	0.659
10	0.593	0.407

(d)

To compute the pairwise marginals, we note that:

$$p(h_t, h_{t+1}|v_{1:T}) \propto p(h_t, h_{t+1}, v_{1:T})$$

And:

$$\begin{aligned} p(h_t, h_{t+1}, v_{1:T}) &= p(h_t, h_{t+1}, v_{1:t}, v_{t+1}, v_{t+2:T}) \\ &= p(v_{t+2:T}|h_t, h_{t+1}, v_{1:t+1}) \cdot (v_{t+1}|h_{t+1}, h_t, v_{1:t}) \cdot p(h_{t+1}|h_t, v_{1:t}) \cdot p(h_t, v_{1:t}) \\ &= \beta(h_{t+1}) \cdot p(v_{t+1}|h_{t+1}) \cdot p(h_{t+1}|h_t) \cdot \alpha(h_t) \end{aligned}$$

We can then determine the value for $p(h_1, h_2|v_{1:T})$ and $p(h_4, h_5|v_{1:T})$:

- Probability distribution of $p(h_1, h_2|v_{1:T})$:

	$h_1 = 0$	$h_1 = 1$
$h_2 = 0$	1	0
$h_2 = 1$	0	0

- Probability distribution of $p(h_4, h_5|v_{1:T})$:

	$h_4 = 0$	$h_4 = 1$
$h_5 = 0$	0.158	0.574
$h_5 = 1$	0.14	0.128

(e)

Baum-Welch algorithm: We introduce the variational distribution $q(h|v)$. The free energy as a function of q and the parameter θ (which in this case corresponds to the transition matrix) is then given by:

$$F(q, \theta) = \left\langle \log \frac{p(v, h|\theta)}{q(h)} \right\rangle_{q(h)}$$

- E-step:

  We can rewrite $F(q, \theta)$ as:

  $$F(q, \theta) = \log p(v|\theta) - \mathrm{KL}(q(h)||p(h|v, \theta))$$

  To maximise the free energy, we set $q(h)$ to $p(h|v, \theta)$. In practice, for the M-step, we do not need the full distribution $q(h)$, but only the pair-wise marginals $q(h_t, h_{t+1})$, which we set to $p(h_t, h_{t+1}|v_{1:T}, T)$, using the answer obtained under (d).

  To avoid numerical problems, we can use normalised α and β messages. To the extent that the we need to normalise the eventual result to obtain $p(h_t, h_{t+1}|v_{1:T}, T)$, normalising any of the factors during the process does not change the result.

- M-step:

  We can rewrite $F(q, \theta)$ as:

$$F(q, \theta) =< \log p(h_{1:T}, v_{1:T}|\theta) >_{q(h)} + H(q)$$

  We need to maximise the free energy w.r.t. θ. The second term does not depend on θ. For the first term, we can replace $q(h)$ with $p(h_{1:T}|v_{1:T}, \theta^{old})$ based on the results from the E-step.

  We therefore need to maximise:

$$\begin{aligned}
< \log p(h_{1:T}, v_{1:T}|\theta) >_{q(h)} &=< \log p(h_{1:T}, v_{1:T}|\theta) >_{p(h_{1:T}|v_{1:T}, \theta^{old})} \\
&=< \log p(h_1) + \sum_{t=1}^{T} \log p(v_t|h_t) + \sum_{t=2}^{T} \log p(h_t|h_{t-1}) >_{p(h_{1:T}|v_{1:T}, \theta^{old})} \\
&=< \log p(h_1) >_{p(h_1|v_{1:T}, \theta^{old})} + \sum_{t=1}^{T} < \log p(v_t|h_t) >_{p(h_t|v_{1:T}, \theta^{old})} + \\
&\sum_{t=2}^{T} < \log p(h_t|h_{t-1}) >_{p(h_t, h_{t-1}|v_{1:T}, \theta^{old})}
\end{aligned}$$

  The only term that depends on θ is the last one, which we differentiate with respect to T_{ij}:

$$\frac{\partial}{\partial T_{ij}} F(q, \theta) = \frac{\partial}{\partial T_{ij}} \Big[\sum_{t=2}^{T} \sum_{i,j} p(h_t = i, h_{t-1} = j|v_{1:T}, \theta^{old}) \log T_{ij} \Big] \quad (i, j \in \{0, 1\})$$

  To enforce T_{ij} to be a probability distribution over i, we can use the fact that: $T_{1j} = 1 - T_{0j}$. The equation above then becomes:

$$\begin{aligned}
\frac{\partial}{\partial T_{ij}} F(q, \theta) &= \frac{\partial}{\partial T_{ij}} \Big[\sum_{t=2}^{T} \sum_{j} p(h_t = 0, h_{t-1} = j|v_{1:T}, \theta^{old}) \log T_{0j} + p(h_t = 1, h_{t-1} = j|v_{1:T}, \theta^{old}) \log(1 - T_{0j}) \Big] \\
&= \frac{\partial}{\partial T_{ij}} \Big[\sum_{t=2}^{T} \sum_{j} p(h_t = 0, h_{t-1} = j|v_{1:T}, \theta^{old}) \log T_{0j} + p(h_t = 1, h_{t-1} = j|v_{1:T}, \theta^{old}) \log(1 - T_{0j}) \Big]
\end{aligned}$$

  Differentiating w.r.t. to T_{0j} (for a given value of j):

$$\frac{\partial}{\partial T_{0j}} F(q, \theta) = \frac{1}{T_{0j}} \Big[\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j|v_{1:T}, \theta^{old}) \Big] + \frac{1}{1 - T_{0j}} \Big[\sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j|v_{1:T}, \theta^{old}) \Big]$$

Setting the expression to 0:

$$\frac{1}{T_{0j}}\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old}) - \frac{1}{1-T_{0j}}\sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j | v_{1:T}, \theta^{old}) = 0$$

$$\frac{1}{T_{0j}}\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old}) = \frac{1}{1-T_{0j}}\sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j | v_{1:T}, \theta^{old})$$

$$\frac{1-T_{0j}}{T_{0j}}\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old}) = \sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j | v_{1:T}, \theta^{old})$$

$$\frac{1}{T_{0j}}\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old}) = \sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j | v_{1:T}, \theta^{old}) + \sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old})$$

$$T_{0j} = \frac{\sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old})}{\sum_{t=2}^{T} p(h_t = 1, h_{t-1} = j | v_{1:T}, \theta^{old}) + \sum_{t=2}^{T} p(h_t = 0, h_{t-1} = j | v_{1:T}, \theta^{old})}$$

- We then iterate the EM algorithm until convergence and obtain the following transition matrix:

$$\begin{pmatrix} 0.83 & 0.07 \\ 0.17 & 0.93 \end{pmatrix}$$

The algorithm converges after 17 iterations, at which point the variation from one estimate to the next is less than 0.001.

(f)

Compute $p(h_{1000}|v_{1:1000}$ by normalising $\alpha(h_{1000})$, which gives:

$$p(h_{1000}|v_{1:1000}) \approx \begin{pmatrix} 0.054 \\ 0.946 \end{pmatrix}$$

From there, use the transition matrix and the emission matrix to compute $p(v_{1001}|h_{1001})$.

$$\begin{aligned} p(v_{1001}|v_{1:1000}) &= \sum_{h_{1000}, h_{1001}} p(v_{1001}, h_{1000}, h_{1001}|v_{1:1000}) \\ &= \sum_{h_{1000}, h_{1001}} \Big[p(v_{1001}|h_{1001}, v_{1:1000}) \cdot p(h_{1001}|h_{1000}) \cdot p(h_{1000}|v_{1:1000}) \Big] \\ &= \sum_{h_{1001}} \Big[p(v_{1001}|h_{1001}) \sum_{h_{1000}} p(h_{1001}|h_{1000}) \cdot p(h_{1000}|v_{1:1000}) \Big] \end{aligned}$$

We obtain the following probability distribution for v_{1001}:

$$p(v_{1001}|v_{1:1000}) = \begin{pmatrix} 0.477 \\ 0.068 \\ 0.455 \end{pmatrix}$$

Python Code

```
import numpy as np

v = [0,1,0,2,0,2,1,0,2,0]
emission_matrix = np.array(([0.3,0.5],[0.6,0],[0.1,0.5]))
transition_matrix = np.array(([0.5,0.8],[0.5,0.2]))

# (c)

## forward recursion

### Standard method for £\alpha£

alpha_h2 = emission_matrix[v[2-1]] * np.matmul(transition_matrix, alpha_h1)

p_h1 = np.array(([1,0]))
alpha_h1 = np.multiply(emission_matrix[v[0]], p_h1)
alphas = [alpha_h1]
for i in range(1, 10):
    alpha_h_t = emission_matrix[v[i]] * np.matmul(transition_matrix, alphas[i-1])
    alphas.append(alpha_h_t)

### Normalised alphas

p_h1 = np.array(([1,0]))
alpha_h1 = np.multiply(emission_matrix[v[0]], p_h1)
sum_h1 = sum(alpha_h1)
norm_alpha_h1 = alpha_h1/sum_h1
norm_alphas = [norm_alpha_h1]
for i in range(1, 10):
    alpha_h_t = emission_matrix[v[i]] * np.matmul(transition_matrix, norm_alphas[i-1])
    norm_alpha_h_t = alpha_h_t/sum(alpha_h_t)
    norm_alphas.append(norm_alpha_h_t)

## Backward recursion

### Standard method for £\beta£

beta_h10 = 1
betas = [beta_h10]
for i in range(1, 10):
    beta_h_t = np.matmul(np.multiply(betas[i-1],emission_matrix[v[len(v)-i]]),
    ↪  transition_matrix)
    betas.append(beta_h_t)
betas.reverse()
betas

### Normalised betas

beta_h10 = 1
norm_betas = [beta_h10]
for i in range(1, 10):
```

```
    beta_h_t = np.matmul(np.multiply(norm_betas[i-1],emission_matrix[v[len(v)-i]]),
    ↪  transition_matrix)
    norm_beta_h_t = beta_h_t/sum(beta_h_t)
    norm_betas.append(norm_beta_h_t)
norm_betas.reverse()

## Forward-backward

for i in range(10):
    joint_prob = np.multiply(alphas[i], betas[i])
    normalising_constant = sum(joint_prob)
    conditional = joint_prob
    for j in range(2):
        conditional[j] = conditional[j]/normalising_constant
    print(f'{i+1} & {round(conditional[0],3)} & {round(conditional[1],3)}')

### Using normalised messages

for i in range(10):
    joint_prob = np.multiply(norm_alphas[i], norm_betas[i])
    joint_prob = joint_prob/sum(joint_prob)
    print(f'At t = {i+1}: P(h=0) = {round(joint_prob[0],3)} and P(h=1) =
    ↪  {round(joint_prob[1],3)}')

# (d)

## using normalised messages

for t in [1,4]:
    joint_prob_distribution = np.zeros((2,2))
    for i in range(2):
        for j in range(2):
            prob =
            ↪  norm_betas[t][i]*emission_matrix[v[t]][i]*transition_matrix[i][j]*norm_alphas[t-1][j]
            joint_prob_distribution[i,j] = prob
    normalising_constant = sum(sum(joint_prob_distribution))
    normalised_prob = joint_prob_distribution/normalising_constant
    print(f'P(h{t} = 0, h{t+1} = 0) = {round(normalised_prob[0][0],3)}')
    print(f'P(h{t} = 0, h{t+1} = 1) = {round(normalised_prob[1][0],3)}')
    print(f'P(h{t} = 1, h{t+1} = 0) = {round(normalised_prob[0][1],3)}')
    print(f'P(h{t} = 1, h{t+1} = 1) = {round(normalised_prob[1][1],3)} \n')

# (e)

chess_results = np.loadtxt('chess.txt')
chess_results = np.array(chess_results, dtype = int)

## E step

p_h1 = np.array(([1,0]))
alpha_h1 = np.multiply(chess_results[v[0]], p_h1)
sum_h1 = sum(alpha_h1)
norm_alpha_h1 = alpha_h1/sum_h1
norm_alphas = [norm_alpha_h1]
```

```
for i in range(1, 1000):
    alpha_h_t = emission_matrix[chess_results[i]] * np.matmul(transition_matrix,
    ↪  norm_alphas[i-1])
    norm_alpha_h_t = alpha_h_t/sum(alpha_h_t)
    norm_alphas.append(norm_alpha_h_t)

beta_h10 = 1
norm_betas = [beta_h10]
for i in range(1, 1000):
    beta_h_t =
    ↪  np.matmul(np.multiply(norm_betas[i-1],emission_matrix[chess_results[len(chess_results)-i]]),
    ↪  transition_matrix)
    norm_beta_h_t = beta_h_t/sum(beta_h_t)
    norm_betas.append(norm_beta_h_t)
norm_betas.reverse()

consolidated_prob = []
for t in range(1, len(chess_results)):
    joint_prob_distribution = np.zeros((2,2))
    for i in range(2):
        for j in range(2):
            # special case to acknowledge that beta_1000 is 1 rather than a vector
            if t == 999:
                prob = emission_matrix[chess_results[t]][i] * transition_matrix[i][j] *
                ↪  norm_alphas[t-1][j]
                joint_prob_distribution[i,j] = prob
            else:
                prob = norm_betas[t][i] * emission_matrix[chess_results[t]][i] *
                ↪  transition_matrix[i][j] * norm_alphas[t-1][j]
                joint_prob_distribution[i,j] = prob
    normalising_constant = sum(sum(joint_prob_distribution))
    normalised_prob = joint_prob_distribution/normalising_constant
    consolidated_prob.append(normalised_prob)

# joint probability h_t, h_t+1 is the entry t-1. For example, consolidated_prob[0] is the
↪  joint probability
# distribution of h_1 and h_2. Rows = values of h_t+1, columns, values of h_t
consolidated_prob[998]

## M step

### Distribution for £h_{t-1}=0£

sum_of_prob_00 = []
for i in range(999):
    sum_of_prob_00.append(consolidated_prob[i][0][0])
sum_of_prob_00 = sum(sum_of_prob_00)
sum_of_prob_00

sum_of_prob_10 = []
for i in range(999):
    sum_of_prob_10.append(consolidated_prob[i][1][0])
sum_of_prob_10 = sum(sum_of_prob_10)
sum_of_prob_10
```

```
new_T_00 = sum_of_prob_00/(sum_of_prob_00+sum_of_prob_10)
new_T_10 = 1 - new_T_00

### Distribution for £h_{t-1}=1£

sum_of_prob_01 = []
for i in range(999):
    sum_of_prob_01.append(consolidated_prob[i][0][1])
sum_of_prob_01 = sum(sum_of_prob_01)
sum_of_prob_11 = []
for i in range(999):
    sum_of_prob_11.append(consolidated_prob[i][1][1])
sum_of_prob_11 = sum(sum_of_prob_11)
new_T_01 = sum_of_prob_01/(sum_of_prob_01+sum_of_prob_11)
new_T_11 = 1 - new_T_01

new_T = np.array(([new_T_00, new_T_01],[new_T_10, new_T_11]))

### Multiple iterations

#define initial transition matrix
transition_matrix = (([0.5,0.8],[0.5,0.2]))

for r in range(25):
    p_h1 = np.array(([1,0]))
    alpha_h1 = np.multiply(chess_results[v[0]], p_h1)
    sum_h1 = sum(alpha_h1)
    norm_alpha_h1 = alpha_h1/sum_h1
    norm_alphas = [norm_alpha_h1]
    for i in range(1, 1000):
        alpha_h_t = emission_matrix[chess_results[i]] * np.matmul(transition_matrix,
        ↪  norm_alphas[i-1])
        norm_alpha_h_t = alpha_h_t/sum(alpha_h_t)
        norm_alphas.append(norm_alpha_h_t)

    beta_h10 = 1
    norm_betas = [beta_h10]
    for i in range(1, 1000):
        beta_h_t =
        ↪  np.matmul(np.multiply(norm_betas[i-1],emission_matrix[chess_results[len(chess_results)-i]])
                            transition_matrix)
        norm_beta_h_t = beta_h_t/sum(beta_h_t)
        norm_betas.append(norm_beta_h_t)
    norm_betas.reverse()

    consolidated_prob = []
    for t in range(1, len(chess_results)):
        joint_prob_distribution = np.zeros((2,2))
        for i in range(2):
            for j in range(2):
                # special case to acknowledge that beta_1000 is 1 rather than a vector
                if t == 999:
```

```
                    prob = emission_matrix[chess_results[t]][i] * transition_matrix[i][j]
                    ↪  * norm_alphas[t-1][j]
                    joint_prob_distribution[i,j] = prob
                else:
                    prob = norm_betas[t][i] * emission_matrix[chess_results[t]][i] *
                    ↪  transition_matrix[i][j] * norm_alphas[t-1][j]
                    joint_prob_distribution[i,j] = prob
        normalising_constant = sum(sum(joint_prob_distribution))
        normalised_prob = joint_prob_distribution/normalising_constant
        consolidated_prob.append(normalised_prob)

    sum_of_prob_00 = []
    for i in range(1000-1):
        sum_of_prob_00.append(consolidated_prob[i][0][0])
    sum_of_prob_00 = sum(sum_of_prob_00)

    sum_of_prob_10 = []
    for i in range(1000-1):
        sum_of_prob_10.append(consolidated_prob[i][1][0])
    sum_of_prob_10 = sum(sum_of_prob_10)

    new_T_00 = sum_of_prob_00/(sum_of_prob_00+sum_of_prob_10)
    new_T_10 = 1 - new_T_00

    sum_of_prob_01 = []
    for i in range(1000-1):
        sum_of_prob_01.append(consolidated_prob[i][0][1])
    sum_of_prob_01 = sum(sum_of_prob_01)
    sum_of_prob_11 = []
    for i in range(1000-1):
        sum_of_prob_11.append(consolidated_prob[i][1][1])
    sum_of_prob_11 = sum(sum_of_prob_11)
    new_T_01 = sum_of_prob_01/(sum_of_prob_01+sum_of_prob_11)
    new_T_11 = 1 - new_T_01

    # update transition matrix
    transition_matrix = np.array(([new_T_00, new_T_01],[new_T_10, new_T_11]))
    print(transition_matrix)


# (f)

# Compute the normalised alpha for £h_{1000}£, which gives £p(h_{1000}|v_{1:1000})£
norm_alphas[-1]

# From there, use the transition matrix to compute £p(h_{1001}|v_{1:1000})£
p_h_1001 = np.matmul(transition_matrix, norm_alphas[-1])
p_v_1001 = np.matmul(emission_matrix, p_h_1001)
p_v_1001
```