COMP0086: Probabilistic and Unsupervised Learning

January 24, 2025

Contents

Question 1

Question 1.a

The dataset consists of binary images with each pixel as 0 or 1, meaning the data is discrete. A multivariate Gaussian model assumes continuous values, which would allow non-binary values between 0 and 1, making it unsuitable. A better choice would be a Bernoulli distribution, which models binary outcomes directly.

Question 1.b

The maximum likelihood estimator (MLE) is defined as

$$\hat{\theta}_{\mathrm{ML}} := \arg\max_{\theta \in \mathcal{T}} p\left(x_1, \ldots, x_n \mid \theta\right), \tag{1}$$

which can be found by solving maximum likelihood equation:

$$\nabla_\theta p(\mathcal{D} \mid \theta) = \nabla_\theta \left(\prod_{i=1}^{n} p\left(x_i \mid \theta\right) \right) = 0, \tag{2}$$

where we will estimate parameter p in further discussion. For convenience, we use the log-likelihood to convert the product into a sum, allowing us to rewrite the expression as:

$$\sum_{i=1}^{n} \nabla_p \log p\left(x_i \mid \theta\right) = 0. \tag{3}$$

Substituting with our multivariate Bernoulli distribution, this becomes:

$$\sum_{i=1}^{n} \nabla_p \log \prod_{d=1}^{D} p_d^{x_d} \left(1 - p_d\right)^{(1-x_d)} = 0. \tag{4}$$

We will first solve the expression inside the summation over n and consider the summation afterward. By taking the log of the product, we convert it into a sum, leading to:

$$\nabla_p \sum_{d=1}^{D} \left(x_d \log p_d + (1 - x_d) \log\left(1 - p_d\right)\right) = 0. \tag{5}$$

Taking the derivative yields:

$$\frac{x_d}{\hat{p}_d} - \frac{(1 - x_d)}{1 - \hat{p}_d} = 0, \tag{6}$$

where $\hat{p}_d$ represents the estimated probability for the d-th element in the final MLE solution.

Now, we incorporate the summation across all N observations:

$$\frac{\sum_{n=1}^{N} x_d^{(n)}}{\hat{p}_d} - \frac{\sum_{n=1}^{N} \left(1 - x_d^{(n)}\right)}{1 - \hat{p}_d} = 0, \tag{7}$$

where N is the total number of pixels in our binary images. Since each x value is either 1 or 0, let N_1 be the count of ones and N_0 the count of zeros. In this case, we have:

$$\frac{N_1}{\hat{p}_d} - \frac{N_0}{1 - \hat{p}_d} = 0 \Rightarrow \hat{p}_d = \frac{N_1}{N_1 + N_0}, \tag{8}$$

which gives the d-th element of the MLE solution. Thus, the overall solution can be expressed as:

$$\hat{\mathbf{p}}_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^{(n)}. \tag{9}$$

Question 1.c

The maximum a posteriori (MAP) estimator is defined as:

$$\hat{\theta}_{\text{MAP}} := \arg\max_{\theta \in \mathcal{T}} p\left(\theta \mid x_1, \ldots, x_n\right), \tag{10}$$

which can be rewritten using Bayes' theorem as:

$$\hat{\theta}_{\text{MAP}} := \arg\max_{\theta \in \mathcal{T}} \frac{p\left(x_1, \ldots, x_n \mid \theta\right) p\left(\theta\right)}{p\left(x_1, \ldots, x_n\right)}. \tag{11}$$

Following a similar approach to 1(b), we take the log-likelihood and then differentiate with respect to p. Since the evidence term $p\left(x_1, \ldots, x_n\right)$ is independent of p, we can focus on maximizing $p\left(x_1, \ldots, x_n \mid \theta\right) p\left(\theta\right)$:

$$\nabla_\theta p(\theta \mid \mathcal{D}) = \nabla_\theta \left(\prod_{i=1}^{n} p\left(x_i \mid \theta\right) p\left(\theta\right) \right) = 0. \tag{12}$$

Using the log-likelihood to convert the product into a sum, we can rewrite this as:

$$\sum_{i=1}^{n} \nabla_p \log p\left(x_i \mid \theta\right) p\left(\theta\right) = 0. \tag{13}$$

Substituting the Beta prior and the multivariate Bernoulli distribution, we obtain:

$$\sum_{i=1}^{n} \nabla_p \log \prod_{d=1}^{D} p_d^{x_d} \left(1 - p_d\right)^{(1 - x_d)} \prod_{d=1}^{D} \frac{1}{B(\alpha, \beta)} p_d^{\alpha - 1} \left(1 - p_d\right)^{\beta - 1} = 0. \tag{14}$$

Taking the log of each term, we express this as:

$$-\sum_{d=1}^{D} \log\left(B(\alpha, \beta)\right) + \sum_{d=1}^{D} x_d \log\left(p_d\right) + \left(1 - x_d\right) \log\left(1 - p_d\right) + \sum_{d=1}^{D} (\alpha - 1) \log\left(p_d\right) + (\beta - 1) \log\left(1 - p_d\right). \tag{15}$$

To simplify, we focus on the d-th element and find its derivative as:

$$\left(\frac{x_d}{\hat{p}_d} - \frac{1 - x_d}{1 - \hat{p}_d} \right) + \left(\frac{\alpha - 1}{\hat{p}_d} - \frac{\beta - 1}{1 - \hat{p}_d} \right) = 0. \tag{16}$$

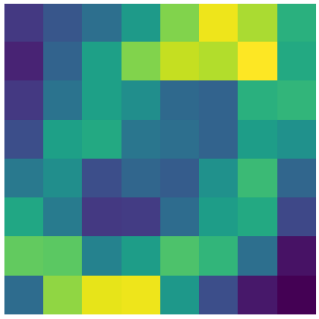
Since $B(\alpha, \beta)$ is independent of p_d, it no longer appears in the derivative.

Applying the summation as in 1(b), we get:

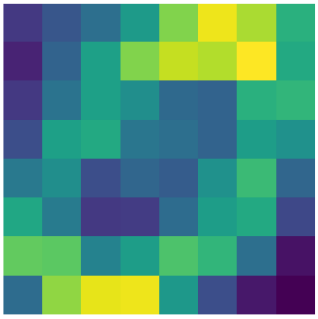
$$\begin{aligned} &\left(\frac{N_1}{\hat{p}_d} - \frac{N_0}{1 - \hat{p}_d} \right) + \left(\frac{\alpha - 1}{\hat{p}_d} - \frac{\beta - 1}{1 - \hat{p}_d} \right) = 0 \\ &\Rightarrow \frac{N_1 + \alpha - 1}{\hat{p}_d} = \frac{N_0 + \beta - 1}{1 - \hat{p}_d} \\ &\Rightarrow \hat{p}_d = \frac{N_1 + \alpha - 1}{N + \alpha + \beta - 2}. \end{aligned} \tag{17}$$

The final MAP estimator can thus be expressed as:

$$\hat{\mathbf{p}}_{\text{MAP}} = \frac{\alpha - 1 + \sum_{n=1}^{N} \mathbf{x}^{(n)}}{N + \alpha + \beta - 2}. \tag{18}$$



(a) MLE



(b) MAP

Figure 1: ML and MAP parameters of a multivariate Bernoulli

Question 1.d and e

Here, we use the code to plot both ML parameters of a multivariate Bernoulli and MAP parameters.

```
def MLE(x):
    return np.mean(x, axis=0)
def MAP(x, alpha, beta):
    N,_ = x.shape
    return  (alpha - 1 + np.sum(x, axis=0)) / (N + alpha + beta - 2)
```

Listing 1: Question 1 code

Maximum Likelihood (ML) and Maximum A Posteriori (MAP) estimation offer different approaches to parameter estimation, particularly in their handling of prior information. ML estimation relies solely on the observed data, maximizing the likelihood to provide the best estimate under the assumption that no prior knowledge about the parameters exists. In contrast, MAP incorporates prior beliefs or information through a prior distribution, allowing for potentially better estimates, especially when the data is limited or noisy. However, MAP's reliance on the prior can be a drawback if the prior is poorly chosen, as it could introduce bias into the estimate. Therefore, when reliable prior knowledge is available, MAP can improve the estimation, particularly if the prior is well-aligned with the likelihood function, such as when a conjugate prior is used. In situations where prior information is unavailable or uncertain, ML is often the more robust choice, avoiding the risk of introducing bias from an inappropriate prior.

Question 2

Let us label these three model as M_1, M_2 and M_3. Then, the equally likely priori for different models can be written as:

$$P(M_1) = P(M_2) = P(M_3) = \frac{1}{3}. \tag{19}$$

Now, We will derive the likelihood $P(x \mid p)$.

For (a), we substitute $p_d = 0.5$ into likelihood equation:

$$P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^1) = \prod_{n=1}^{N} \prod_{d=1}^{D} 0.5^{x_d^{(n)}} \left(1 - 0.5\right)^{\left(1 - x_d^{(n)}\right)} = 0.5^{ND}, \tag{20}$$

where we can apply log:

$$\log P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^1) = -ND \log(2). \tag{21}$$

For (b), we have $\mathbf{p}$ in forms of $(p_d, \cdots, p_d)$ where p_d can be any random number between 0 and 1.

$$\begin{aligned} P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^2) &= \int_0^1 \prod_{n=1}^{N} \prod_{d=1}^{D} (p_d)^{x_d^{(n)}} \left(1 - p_d\right)^{\left(1 - x_d^{(n)}\right)} dp_d, \\ \Rightarrow \quad P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^2) &= \int_0^1 p_d^{\sum_1^N \sum_d^D x_d^{(n)}} \left(1 - p_d\right)^{\sum_1^N \sum_d^D \left(1 - x_d^{(n)}\right)} dp_d, \end{aligned} \tag{22}$$

which reminds us the form of Beta distribution:

$$B(\alpha, \beta) = \int_0^1 u^{\alpha - 1} (1 - u)^{\beta - 1} du. \tag{23}$$

Thus, we can find this likelihood as:

$$P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^2) = B\left(\sum_1^N \sum_d^D x_d^{(n)} + 1, \sum_1^N \sum_d^D \left(1 - x_d^{(n)}\right) + 1\right). \tag{24}$$

For (c), we repeat the similar process as (b) with different entries of p_d. The likelihood function can be expressed as:

$$\begin{aligned} P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^3) &= \int_0^1 \cdots \int_0^1 \prod_{n=1}^{N} (p_d)^{x_d^{(n)}} \left(1 - p_d\right)^{\left(1 - x_d^{(n)}\right)} dp_1 \cdots dp_D, \\ \Rightarrow \quad P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^3) &= \prod_{d=1}^{D} \left(\int_0^1 p_d^{\sum_1^N x_d^{(n)}} \left(1 - p_d\right)^{\sum_1^N \left(1 - x_d^{(n)}\right)} dp_d \right), \end{aligned} \tag{25}$$

where we shall integrate over different p_d. Luckily, we still have the form of Beta distribution and our likelihood can be expressed as:

$$P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \mathbf{p}^3) = \prod_{d=1}^{D} B\left(\sum_1^N x_d^{(n)} + 1, \sum_1^N \left(1 - x_d^{(n)}\right) + 1\right). \tag{26}$$

Following the Bayes rule, we can find the posterior probabilities as:

$$P(M_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid M_i) P(M_i)}{P(x)}, \tag{27}$$

where the evidence term $P(\mathbf{x})$ is constant. For simplicity, we can compare the logarithm of the posterior:

$$\begin{aligned} &\log P(M_1 \mid \mathbf{x}) \propto -ND \log(2) - \log(3), \\ &\log P(M_2 \mid \mathbf{x}) \propto Betaln\left(\sum_1^N \sum_d^D x_d^{(n)} + 1, \sum_1^N \sum_d^D \left(1 - x_d^{(n)}\right) + 1\right) - \log(3), \\ &\log P(M_3 \mid \mathbf{x}) \propto \sum_{d=1}^{D} Betaln\left(\sum_1^N x_d^{(n)} + 1, \sum_1^N \left(1 - x_d^{(n)}\right) + 1\right) - \log(3), \end{aligned} \tag{28}$$

where Betaln is the logarithm of Beta distribution. Here, we can calculate these with the following code.

```
#Load Data
X = np.loadtxt('binarydigits.txt')
N, D = X.shape
# Model 1
log_P_M1 = - N*D*np.log(2) - np.log(3)
# Model 2
log_P_M2 = betaln(np.sum(X)+1,np.sum(1-X)+1) - np.log(3)
# Model 3
log_P_M3 = 0
for d in range(D):
    X_d = X[d,:]
    log_P_M3 += betaln(np.sum(X_d)+1,N-np.sum(X_d)+1)
log_P_M3 -= np.log(3)
print(log_P_M1,log_P_M2,log_P_M3)
```

Listing 2: Question 2 code

Thus, we get the final results as:

$$\begin{aligned} \log P(M_1 \mid \mathbf{x}) &\propto -4437.24, \\ \log P(M_2 \mid \mathbf{x}) &\propto -4284.82, \\ \log P(M_3 \mid \mathbf{x}) &\propto -3727.40. \end{aligned} \tag{29}$$

Question 3

Question 3.a

Now, we want to add a new set of parameters $\pi_1 \cdots \pi_K$ as a proportions to our original multivariate Bernoulli distributions:

$$P(\mathbf{x}^{(n)} \mid \mathrm{P}) = \prod_{d=1}^{D} p_d^{x_d} \left(1 - p_d\right)^{(1-x_d)}, \tag{30}$$

which can be expressed as:

$$P(\mathbf{x}^{(n)} \mid \boldsymbol{\pi}, \mathrm{P}) = \sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} p_{kd}^{x_d} \left(1 - p_{kd}\right)^{(1-x_d)}. \tag{31}$$

For N images, the likelihood can be written as:

$$P(\mathbf{x}^1 \cdots \mathbf{x}^N \mid \boldsymbol{\pi}, \mathrm{P}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} p_{kd}^{x_d} \left(1 - p_{kd}\right)^{(1-x_d)}. \tag{32}$$

Question 3.b

Using the expression of responsibility, we can get:

$$\begin{aligned}
& r_{nk} = P\left(s^{(n)} = k \mid \mathbf{x}^{(n)}, \boldsymbol{\pi}, \mathrm{P}\right), \\
\Rightarrow \quad & r_{nk} = \frac{P\left(\mathbf{x}^{(n)} \mid s^{(n)} = k, \boldsymbol{\pi}, \mathrm{P}\right) P\left(s^{(n)} = k \mid \boldsymbol{\pi}, \mathrm{P}\right)}{\sum_{k'=1}^{K} P\left(\mathbf{x}^{(n)} \mid s^{(n)} = k', \boldsymbol{\pi}, \mathrm{P}\right) P\left(s^{(n)} = k' \mid \boldsymbol{\pi}, \mathrm{P}\right)}, \\
\Rightarrow \quad & r_{nk} = \frac{\pi_k \prod_{d=1}^{D} p_{kd}^{x_d} \left(1 - p_{kd}\right)^{(1-x_d)}}{\sum_{k'=1}^{K} \pi_{k'} \prod_{d=1}^{D} p_{k'd}^{x_d} \left(1 - p_{k'd}\right)^{(1-x_d)}}.
\end{aligned} \tag{33}$$

Question 3.c

Let us calculate the expected log-joint first.

$$\begin{aligned}
& \left\langle \sum_n \log P\left(\mathbf{x}^{(n)}, s^{(n)} \mid \boldsymbol{\pi}, \mathrm{P}\right) \right\rangle_{q\left(\left\{s^{(n)}\right\}\right)}, \\
= & \sum_{k=1}^{K} \sum_{n=1}^{N} q\left(s^{(n)} = k\right) \log P\left(\mathbf{x}^{(n)}, s^{(n)} \mid \boldsymbol{\pi}, \mathrm{P}\right), \\
= & \sum_{k,n} r_{nk} \log P\left(s^{(n)} \mid \mathbf{x}^{(n)}, \boldsymbol{\pi}, \mathrm{P}\right) P\left(\mathbf{x}^{(n)} \mid s^{(n)}, \boldsymbol{\pi}, \mathrm{P}\right), \\
= & \sum_{k,n} r_{nk} \log \left(\pi_k \prod_{d=1}^{D} p_{kd}^{x_d} \left(1 - p_{kd}\right)^{(1-x_d)} \right), \\
= & \sum_{k,n} r_{nk} \left(\log \pi_k + \sum_{d=1}^{D} x_d^{(n)} \log\left(p_{kd}\right) + \left(1 - x_d^{(n)}\right) \log\left(1 - p_{kd}\right) \right).
\end{aligned} \tag{34}$$

We denote the expectation above as E. To maximize E, we first take the partial derivative with respect to p_{kd}:

$$\begin{aligned}
& \frac{\partial E}{\partial p_{kd}} = 0, \\
\Rightarrow \quad & \sum_{n=1}^{D} r_{nk} \left(\sum_{d=1}^{D} x_d^{(n)} \frac{1}{p_{kd}} - \left(1 - x_d^{(n)}\right) \frac{1}{1 - p_{kd}} \right) = 0, \\
\Rightarrow \quad & \sum_{n=1}^{D} r_{nk} \left(x_d^{(n)} \frac{1}{p_{kd}} - \left(1 - x_d^{(n)}\right) \frac{1}{1 - p_{kd}} \right) = 0, \\
\Rightarrow \quad & \left(1 - p_{kd}\right) \sum_{n=1}^{D} \left(r_{nk} x_d^{(n)} \right) = p_{kd} \sum_{n=1}^{D} r_{nk} \left(1 - x_d^{(n)}\right),
\end{aligned} \tag{35}$$

which leads to:

$$\hat{p}_{kd} = \frac{\sum_{n=1}^{D} \left(r_{nk} x_d^{(n)} \right)}{\sum_{n=1}^{D} r_{nk}}. \tag{36}$$

Now, we move to the derivative with respect to π_k. We need to apply a Lagrangian multiplier as a constraint $\sum_{k=1}^{K} \pi_k = 1$ is given. Thus, we have:

$$\begin{aligned} & \frac{\partial E}{\partial \pi_k} + \lambda = 0, \\ \Rightarrow \quad & \sum_{k,n} \frac{r_{nk}}{\pi_k} + \lambda = 0, \\ \Rightarrow \quad & \sum_{k,n} r_{nk} + \sum_{k} \pi_k \lambda = 0, \\ \Rightarrow \quad & N + \lambda = 0, \end{aligned} \tag{37}$$

where we notice that r_{nk} is normalized and the constraint of π_k is used. In that case, we can find $\hat{\pi_k}$:

$$\hat{\pi}_k = \frac{\sum_n r_{nk}}{N}. \tag{38}$$

Question 3.d

(d) Implement the EM algorithm for a mixture of K multivariate Bernoulli. Let us first define the likelihood function.

```
def likelihood_K_Mixing_Bernoulli(X,K,Pi,P):
    N, D = X.shape
    likelihood = 0
    for n in range(N): # sum over N
        likelihood_N = []
        for k in range(K): # sum over K
            likelihood_NK = Pi[k]*np.prod((P[k,:]**X[n,:])*((1-P[k,:])**(1-X[n,:])))
            likelihood_N.append(likelihood_NK)
        likelihood += np.log(np.sum(likelihood_N))
    return likelihood
```

Question (3) Log-likelihood

Then, the E-step and M-step are coded as:

```
def E_step(X,K,Pi,P):
    N, D = X.shape
    # create an empty matrix for responsibility
    Responsibility = np.empty((N,K))
    # sum over N
    for n in range(N):
        Numerator = np.empty(K)
        # sum over K
        for k in range(K):
            Numerator[k] = Pi[k]*np.prod((P[k,:]**X[n,:])*((1-P[k,:])**(1-X[n,:])))
        Denominator = np.sum(Numerator)
        # find responsibility
        Responsibility[n,:] = Numerator/Denominator
    return Responsibility

def M_step(X,K,Responsibility):
    N, D = X.shape
    # create hat_p_kd and hat_pi_k
    Hat_P = np.empty((K,D))
    Hat_Pi = np.empty(K)
    for k in range(K):
        # Follow the expressions in Q3(c)
        Hat_P[k,:] = np.sum(Responsibility[:,k]*X.T,1)/np.sum(Responsibility[:,k])
        Hat_Pi[k] = np.sum(Responsibility[:,k])/N
    return Hat_P, Hat_Pi
```

Question (3) E-step and M-step

Finally, we combine above code together to iterate our EM steps:

K values	π_k values
k=2	0.59014409, 0.40985591
k=3	0.42000426, 0.17999998, 0.39999575
k=4	0.3199043 , 0.16981744, 0.19026642, 0.32001185
k=7	0.09, 0.09969762, 0.08037638, 0.18030827, 0.06, 0.08962362, 0.39999412
k=10	0.14, 0.04, 0.15000267, 0.10999683, 0.12999862, 0.06000787, 0.04, 0.03999998, 0.13999403, 0.15

Table 1: π_k values

```
def EM_Start(X,K):
    N, D = X.shape
    max_iterations = 100 # set iterations as 100
    convergence = 0.0001 # less than convergence, the loop breaks
    # initials Pi and P
    Pi_initial = np.random.rand(K)
    Pi__initial_normal = Pi_initial/np.sum(Pi_initial)
    P_initial = np.random.rand(K,D)
    Pi = Pi__initial_normal
    P = P_initial
    log_likelihoods = []
    log_likelihoods.append(likelihood_K_Mixing_Bernoulli(X,K,Pi,P))
    # start our EM
    for i in range(max_iterations):
        # E-step
        Responsibility = E_step(X,K,Pi,P)
        # M-step
        P, Pi = M_step(X,K,Responsibility)
        log_likelihoods.append(likelihood_K_Mixing_Bernoulli(X,K,Pi,P))
        difference = np.abs(log_likelihoods[i]-log_likelihoods[i+1])
        if difference < convergence:
            break
    return Responsibility, P, Pi, log_likelihoods
```

Question (3) EM Alogrithm

Following the questions, we run EM algorithm for different K values.

```
# load data
X = np.loadtxt('binarydigits.txt')
# K list:
K_list = [2,3,4,7,10]
# create the list for collection
Responsibility_list = []
P_list = []
Pi_list = []

plt.figure(figsize=(10,8))
# EM algorithm for different K
for K in K_list:
    Responsibility, P, Pi, log_likelihoods= EM_Start(X, K)
    # collect parameters
    Responsibility_list.append(Responsibility)
    P_list.append(P)
    Pi_list.append(Pi)

    # plot
    num_iterations = len(log_likelihoods)
    iterations = np.linspace(0, num_iterations, num_iterations)
    plt.plot(iterations, log_likelihoods, label = "K = " + str(K))

print(Responsibility_list, P_list, Pi_list)
plt.legend(loc='lower right')
plt.xlabel('Iterations', fontsize = 15)
plt.ylabel('Log-likelihood', fontsize = 15)
plt.grid()
plt.title("Log-likelihood VS Iterations",fontsize = 15)
plt.savefig('Q3(d).png')
```

Question (3) Vary K values

Here we have the plot and parameters:

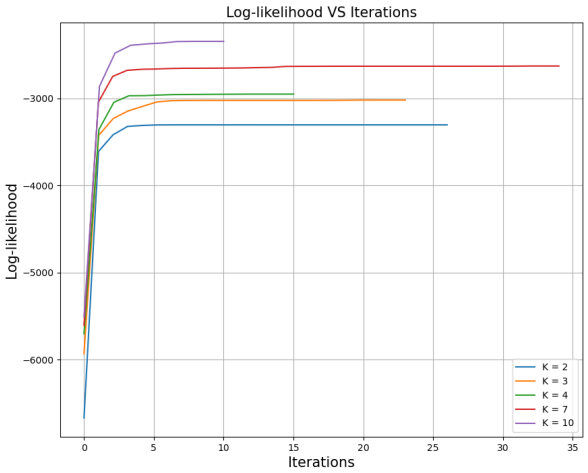


Figure 2: Log-likelihood vs Iterations

Question 3.e

(e) Run the algorithm a few times starting from randomly chosen initial conditions. We get different log-likelihoods and parameters each time and the log-likelihoods will converge to a fix value after several iterations. We can deduce that the value of log-likelihood depends on value of K from Figure 2.

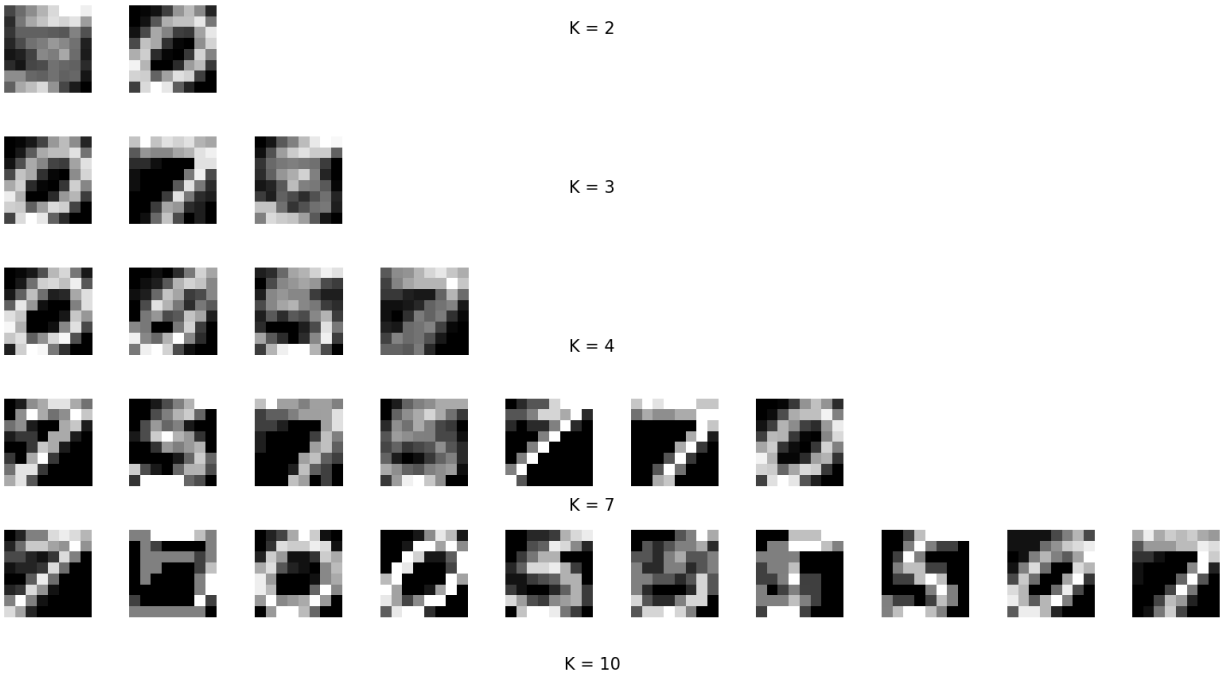


Figure 3: Learned probability vectors

The plot shows the number 0, 5 and 7. We get different mixtures of the numbers that depends on the choices of K-value. In $K = 2$, we have roughly the same value of π_k which gives a mixture of 0, 5 and 7 into two different group. For higher K values, we see more groups. It seems that we can get a clear version for each number at $K = 3$.

Here is the comment on the algorithm. Overall, it works well as we can find when we check the responsibilities. At the end of the iterations, we get an array containing only one element non-zero. This means our algorithm leaves only one cluster at the end. If we add more K values i.e in Figure 4, we can find that the algorithm works not well at large K. It is not a good choice to have large value K since we only three types of number. A desired K value should be larger or equal to 3. The assumption is made as we know the data set compositions. In this case, one can

improve our model by using prior information of the data. For example, we have three numbers in the data set which means $K = 3$ could be a good value. At the same time, we notice that handwritings are influenced by personal style which gives different types of writing number 0, 5 and 7. Once we have a preferred K-value, our algorithm could more efficient.

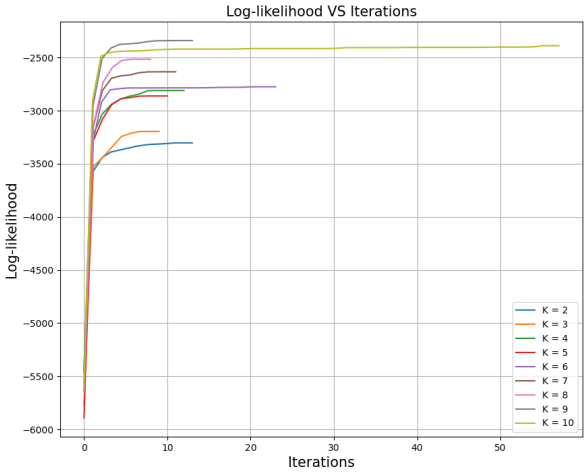


Figure 4: Log-likelihood vs Iterations

Question 5

Question 5.a

Let $N(\alpha, \beta)$ be the counts of numbers of occurrences of symbols and pairs of symbols. We have the transition function in form of $\psi(\alpha, \beta)$ which represents a change from β to α. In that case, we can assume that

$$\sum_{\beta} \psi(\alpha, \beta) = 1, \tag{39}$$

as sum of the probability from β to α must be 1. This can be expressed as a constraint with Lagrangian Multiplier λ. In that case, we can find ML by:

$$\frac{\partial \left(\log p\left(s_1 s_2 \cdots s_n\right) + \sum \lambda \left(1 - \sum_{\beta} \psi(\alpha, \beta)\right)\right)}{\partial \psi(\alpha, \beta)} = 0, \tag{40}$$

where the log-likelihood is :

$$\log p\left(s_1 s_2 \cdots s_n\right) = \log p(s_1) + \sum_{i=2}^{n} \log(\psi(\alpha, \beta)). \tag{41}$$

Then, we can write equation 40 as:

$$\begin{aligned} & \frac{N(\alpha, \beta)}{\psi(\alpha, \beta)} - \lambda = 0, \\ \Rightarrow \quad & N(\alpha, \beta) = \psi(\alpha, \beta)\lambda. \end{aligned} \tag{42}$$

Recall that $\sum_{\beta} \psi(\alpha, \beta) = 1$. Thus, we get:

$$\begin{aligned} & \sum_{\beta} N(\alpha, \beta) = \sum_{\beta} \psi(\alpha, \beta)\lambda, \\ \Rightarrow \quad & \sum_{\beta} N(\alpha, \beta) = \lambda, \\ \Rightarrow \quad & \psi(\alpha, \beta) = \frac{N(\alpha, \beta)}{\sum_{\beta} N(\alpha, \beta)}. \end{aligned} \tag{43}$$

Here is the code for estimated probabilities.

```
# load data and preprocess them
message  = open('message.txt','r').read().replace('\n', '')
symbols = open('symbols.txt','r').read().replace('\n', '')
war_and_peace = open('2600-0.txt','r').read().replace('\n', '').lower()
```

Question 5(a) Load data

```
def Psi(text, symbols):
    num_text = len(text)
    # numbers of symbols
    num_sym = len(symbols)
    # initial the counts
    transition_counts = np.zeros((num_sym,num_sym))
    # go through text
    for i in range(num_text):
        # pick the character
        current_sym = text[i-1] # beta
        next_sym = text[i] # alpha
        # check whether these character in the symbols.
        # if character within symbols, we get id. Otherwise, return -1
        current_sym_id = symbols.find(current_sym)
        next_sym_id = symbols.find(next_sym)
        if (current_sym_id != -1) and (next_sym_id != -1):
            transition_counts[current_sym_id, next_sym_id] += 1
        else:
            continue
    # normalize
    row_sums = np.sum(transition_counts, axis=1).reshape(-1, 1)
    Psi = np.divide(transition_counts, row_sums, where=row_sums != 0)
    return Psi
```

Question 5(a) Transition Matrix

```
transition_matrix = Transition_Matrix(war_and_peace, symbols)
plt.figure(figsize=(25,25))
sns.heatmap(np.round(transition_matrix,2),annot=True, xticklabels=True,yticklabels=True, square=
    True).get_figure
plt.savefig('Q5(a).png')
```

Question 5(a) Estimated probabilities in heatmap

Here is the output plot.

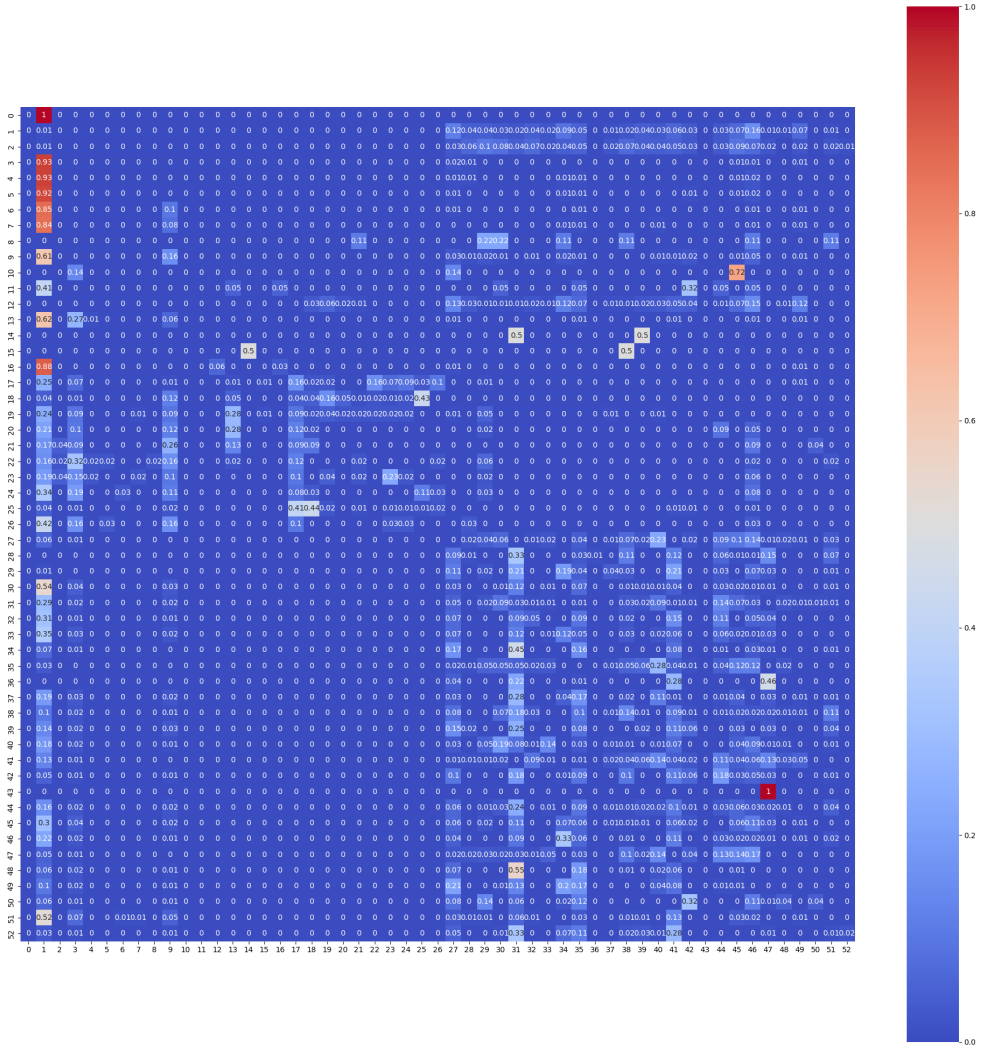


Figure 5: Heatmap for estimated probability

Question 5.b

In the question, we assume that the mapping between symbols is one-to-one. This means one cannot choose a selected symbol. For example, if the previous $\sigma(a) = !$, the following $\sigma \neq !$. Thus, the latent variables are not independent.

The joint probability can be expressed as:

$$p\left(e_1 e_2 \cdots s_n \mid \sigma\right) = p\left(e_1 \mid \sigma\right) \prod_{i=2}^{n} p\left(e_i \mid e_{i-1}, \sigma\right). \tag{44}$$

Question 5.c

Choosing 2 random symbols from n, there is $\binom{n}{2}$ possible choices. In this case, the proposal probability can be written as:

$$S\left(\sigma \to \sigma'\right) = \frac{1}{\binom{n}{2}} = \frac{2}{53 \times 52}, \tag{45}$$

where we let $n = 53$. At the same time, we find the proposal probability is symmetric. Then, we can express MH acceptance probability as:

$$A\left(x_{i+1} \mid x_i\right) = \min\left\{1, \frac{q\left(x_i \mid x_{i+1}\right) p\left(x_{i+1}\right)}{q\left(x_{i+1} \mid x_i\right) p\left(x_i\right)}\right\}, \tag{46}$$

where $q\left(x_{i+1} \mid x_i\right) = q\left(x_i \mid x_{i+1}\right)$ for symmetric proposal probability. Substituting the probability in our question, we get:

$$\begin{aligned}
& A\left(\sigma \mid \sigma'\right) = \min\left\{1, \frac{p\left(\sigma'\right)}{p\left(\sigma\right)}\right\}, \\
\Rightarrow \quad & A\left(\sigma \mid \sigma'\right) = \min\left\{1, \frac{p\left(e_1 \mid \sigma'\right) \prod_{i=2}^{n} p\left(e_i \mid e_{i-1}, \sigma'\right)}{p\left(e_1 \mid \sigma\right) \prod_{i=2}^{n} p\left(e_i \mid e_{i-1}, \sigma\right)}\right\}, \\
\Rightarrow \quad & A\left(\sigma \mid \sigma'\right) = \min\left\{1, \frac{p\left((\sigma')^{-1}(e_1)\right) \prod_{i=2}^{n} \psi((\sigma')^{-1}(e_i), \sigma'(e_{i-1}))}{\prod_{i=2}^{n} \psi(\sigma^{-1}(e_i), \sigma^{-1}(e_{i-1}))}\right\},
\end{aligned} \tag{47}$$

where we write σ^{-1} as a function of e_i that decrypt the symbols and apply the transition probability defined in (a).

Question 5.d

Here is the Metropolis-Hastings (MH) chain in code. We first define a function to decrypt the message which helps us to use the key from message. Then, we follow the instruction in 5(c) and define the proposal for the key. After that, we compute the log-likelihood function.

```
def decrypt_message(text, symbols, key):
    # sigma inverse (e)
    # decode the key to symbols
    decoded_message=str()
    for i in text:
        index = list(key).index(i)
        letter = symbols[index]
        decoded_message+=letter
    return decoded_message

def Proposal(key):
    # Select two distinct indices to swap
    key_new = key.copy()
    idx1, idx2 = random.sample(range(len(key)), 2)

    # Swap the characters at these indices
    key_new[idx1], key_new[idx2] = key[idx2], key[idx1]

    return key_new

def log_likelihood_function(text, symbols, key, matrix):
    decoded_message =  decrypt_message(text, symbols, key)
    log_likelihood = 0
    for i in range(1,len(text)):
        beta = decoded_message[i-1]
        alpha = decoded_message[i]
        beta_id = symbols.index(beta)
        alpha_id = symbols.index(alpha)
        log_likelihood += np.log(matrix[beta_id, alpha_id])
    return log_likelihood


def MCMC_MH(symbols, text, matrix, intelligent_guess, iterations=10000):
    key =  intelligent_guess.copy()
    log_likelihoods = []
    for i in range(iterations):
        proposal_key = Proposal(key).copy()
        current_prob = log_likelihood_function(text, symbols, key, matrix)
        proposal_prob = log_likelihood_function(text, symbols, proposal_key, matrix)
        log_accept_ratio = proposal_prob - current_prob
        u = np.log(np.random.rand())
        if u < log_accept_ratio:
            key = proposal_key.copy()
        # print every 100 iteration
        if i % 100 == 0:
            print('Iteration', i, ': \n', decrypt_message(text, symbols, key)[:60])
            print(current_prob)
            log_likelihoods.append(current_prob)
    return key, log_likelihoods
```

Question 5(d) Metropolis-Hastings (MH) chain

It is also important to have good initial guess. As we investigate 'War and Peace' in previous question, it is a good idea to find some hints from this book. We reorder the characters based on occurrence in both 'War and Peace' and 'message'. We guess that the occurrence are related. For example, we find 'space' is the most character in 'War and Peace' and 'p' is the most character in 'message'. Then, 'space' might be encoded as 'p'. We follow this guess and repeat for all characters. For those are not shown in the 'War and Peace', we randomly map them to rest character. Then, we reorder again to follow the symbols. This gives us an initial guess for key and here is the code.

```
# Order text by counts
def count_and_rank_characters(message, symbols):
    # count each character in the message
    char_count = Counter(message)
    # rank characters by frequency, with the most common characters first
    ranked_chars = [char for char, _ in char_count.most_common()]
    # create an ordered list of symbols based on the ranked characters
    ordered_symbols = []
    seen_symbols = set()
```

```
    # First, add symbols that match the ranked characters
    for char in ranked_chars:
        if char in symbols:
            ordered_symbols.append(char)
            seen_symbols.add(char)
    # get the remaining symbols that were not in the ranked characters
    remaining_symbols = [char for char in symbols if char not in seen_symbols]
    # shuffle the remaining symbols randomly and add to the ordered list
    random.shuffle(remaining_symbols)
    ordered_symbols.extend(remaining_symbols)
    return ordered_symbols
# mapping text to find the key
def combined_mapping_functions(list1, list2, original):
    # map list1 elements to unique numbers (their index)
    number_mapping = {element: index for index, element in enumerate(list1)}
    # map list1 to list2 if they have the same length
    if len(list1) != len(list2):
        raise ValueError("list1 and list2 must have the same length.")
    shuffled_map = {list1[i]: list2[i] for i in range(len(list1))}
    # reorder shuffled_map to match the order of original
    reordered_map = {key: shuffled_map[key] for key in original if key in shuffled_map}
    # return the reorder values
    return reordered_map.values()
```

Question 5(d) Initial guess

To start the chain, we use

```
r1 = count_and_rank_characters(war_and_peace, symbols)
r2 = count_and_rank_characters(message, symbols)
transition_matrix = Psi(war_and_peace, symbols) + 0.001
intelligent_guess = combined_mapping_functions(r1, r2, list(symbols))
decrypt_key, log_values = MCMC_MH(symbols, message, transition_matrix, list(intelligent_guess),
    iterations=10000)
```

Question 5(d) Start to decrypt messages

Here is the list of iterations. After 6000 iteration, the text is fixed.

```
Iteration 0 :
 on lw whunges tnm lhse furnestpre wetsi lw ytades gtfe le ih
-4913.005879300815
Iteration 100 :
 on cd dhungem tns chme lurnemtire detmp cd ytawem gtle ce ph
-4816.650894335458
Iteration 200 :
 on cd dtungel hns ctle murnelhire dehlf cd yhawel ghme ce ft
-4648.682392184368
Iteration 300 :
 on cd dtungel hns ctle murnelhire dehlf cd yhawel ghme ce ft
-4624.31119562341
Iteration 400 :
 on cd dtungel hns ctle murnelhire dehlf cd yhawel ghme ce ft
-4624.521313081218
Iteration 500 :
 on cd diungel tns cile murnelthre detlf cd ytawel gtme ce fi
-4567.078251264491
Iteration 600 :
 on cd diungel tns cile murnelthre detlf cd ytawel gtme ce fi
-4563.6137610479645
Iteration 700 :
 on cy yiungel tnf cile murnelthre yetls cy dtawel gtme ce si
-4504.451488550505
Iteration 800 :
 on cy yiungel tn, cile zurnelthre yetls cy dtawel gtze ce si
-4490.852170848223
Iteration 900 :
 on cy yiungel tn, cile zurnelthre yetls cy dtwael gtze ce si
-4458.92164593478
Iteration 1000 :
 on cy yiungel tnd cile zurnelthre yetls cy ,twael gtze ce si
-4448.969449694636
Iteration 1100 :
 on cy yiungel tnd cile zurnelthre yetls cy ,twael gtze ce si
-4444.745708121956
Iteration 1200 :
```

```
 in cy youngel tnd cole vurnelthre yetls cy ,twael gtve ce so
-4320.917470787748
Iteration 1300 :
 in cy youngel tnd cole vurnelthre yetls cy ,twael gtve ce so
-4320.917470787748
Iteration 1400 :
 an cy ytungel ond ctle vurnelohre yeols cy mowiel gove ce st
-4194.391689769387
Iteration 1500 :
 an cy ytunger ond ctre vulnerohle yeors cy mowier gove ce st
-4144.736226056704
Iteration 1600 :
 an hy ytunger ond htre vulnerocle yeors hy mowier gove he st
-4128.0836534097825
Iteration 1700 :
 an hy ytunger ond htre vulnerocle yeors hy mowier gove he st
-4129.434031119794
Iteration 1800 :
 an hy ytunger ond htre vulnerocle yeors hy mowier gove he st
-4125.538626283407
Iteration 1900 :
 an hy ytunger ond htre vulnerofle yeors hy mowier gove he st
-4125.500471863725
Iteration 2000 :
 an hy ytunger ond htre vulnerofle yeors hy mowier gove he st
-4111.046671905882
Iteration 2100 :
 an hy ytunger ond htre vulnerofle yeors hy mowier gove he st
-4110.360219082787
Iteration 2200 :
 an hy ytunger ond htre vulnerofle yeors hy mowier gove he st
-4104.101692632136
Iteration 2300 :
 an iy ytunger ond itre vulnerofle yeors iy mowher gove ie st
-4079.3138075944585
Iteration 2400 :
 an ty yiunger ond tire vulnerofle yeors ty mowher gove te si
-3936.715608710116
Iteration 2500 :
 an ty yiunger ond tire vulnerofle yeors ty mowher gove te si
-3934.365399182161
Iteration 2600 :
 an ty yiunger ond tire vulnerofle yeors ty mowher gove te si
-3934.365399182161
Iteration 2700 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3931.989558103351
Iteration 2800 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3921.829691740311
Iteration 2900 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3921.0980737400137
Iteration 3000 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3921.0980737400137
Iteration 3100 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3921.0980737400137
Iteration 3200 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3923.457207966006
Iteration 3300 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3913.6758807342153
Iteration 3400 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3904.838654218881
Iteration 3500 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3904.838654218881
Iteration 3600 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
-3906.044784164552
Iteration 3700 :
 an ty yiunger ond tire vulneromle yeors ty fowher gove te si
```

```
-3903.915903707606
Iteration 3800 :
 an wy yiunger ond wire vulneromle yeors wy fother gove we si
-3789.8702741883667
Iteration 3900 :
 an wy yiunger ond wire vulneromle yeors wy fother gove we si
-3789.8702741883667
Iteration 4000 :
 on wy yiunger and wire vulneramle years wy father gave we si
-3747.334993839508
Iteration 4100 :
 on wy yiunger and wire vulneramle years wy father gave we si
-3748.0666118398053
Iteration 4200 :
 on my yiunger and mire vulnerawle years my father gave me si
-3737.330370823384
Iteration 4300 :
 on my yiunger and mire vulnerawle years my father gave me si
-3737.330370823384
Iteration 4400 :
 on my yiunger and mire vulnerawle years my father gave me si
-3737.330370823384
Iteration 4500 :
 on my yiunger and mire vulnerawle years my father gave me si
-3737.330370823384
Iteration 4600 :
 on my yiunger and mire vulnerawle years my father gave me si
-3739.8215556333244
Iteration 4700 :
 on my yiunger and mire vulnerawle years my father gave me si
-3739.8215556333244
Iteration 4800 :
 on my yiunger and mire vulnerawle years my father gave me si
-3738.6947738045683
Iteration 4900 :
 in my younger and more vulnerawle years my father gave me so
-3583.4950298706535
Iteration 5000 :
 in my younger and more vulnerawle years my father gave me so
-3583.4950298706535
Iteration 5100 :
 in my younger and more vulnerawle years my father gave me so
-3583.4950298706535
Iteration 5200 :
 in my younger and more vulnerawle years my father gave me so
-3583.4950298706535
Iteration 5300 :
 in my younger and more vulnerawle years my father gave me so
-3583.4950298706535
Iteration 5400 :
 in my younger and more vulnerawle years my father gave me so
-3582.7634118703563
Iteration 5500 :
 in my younger and more vulnerawle years my father gave me so
-3580.7622846745494
Iteration 5600 :
 in my younger and more vulnerawle years my father gave me so
-3580.7622846745494
Iteration 5700 :
 in my younger and more vulnerawle years my father gave me so
-3580.7622846745494
Iteration 5800 :
 in my younger and more vulnerable years my father gave me so
-3526.706700483252
Iteration 5900 :
 in my younger and more vulnerable years my father gave me so
-3526.706700483252
Iteration 6000 :
 in my younger and more vulnerable years my father gave me so
-3526.706700483252
```

We also track the change of log-likelihood in Figure 6

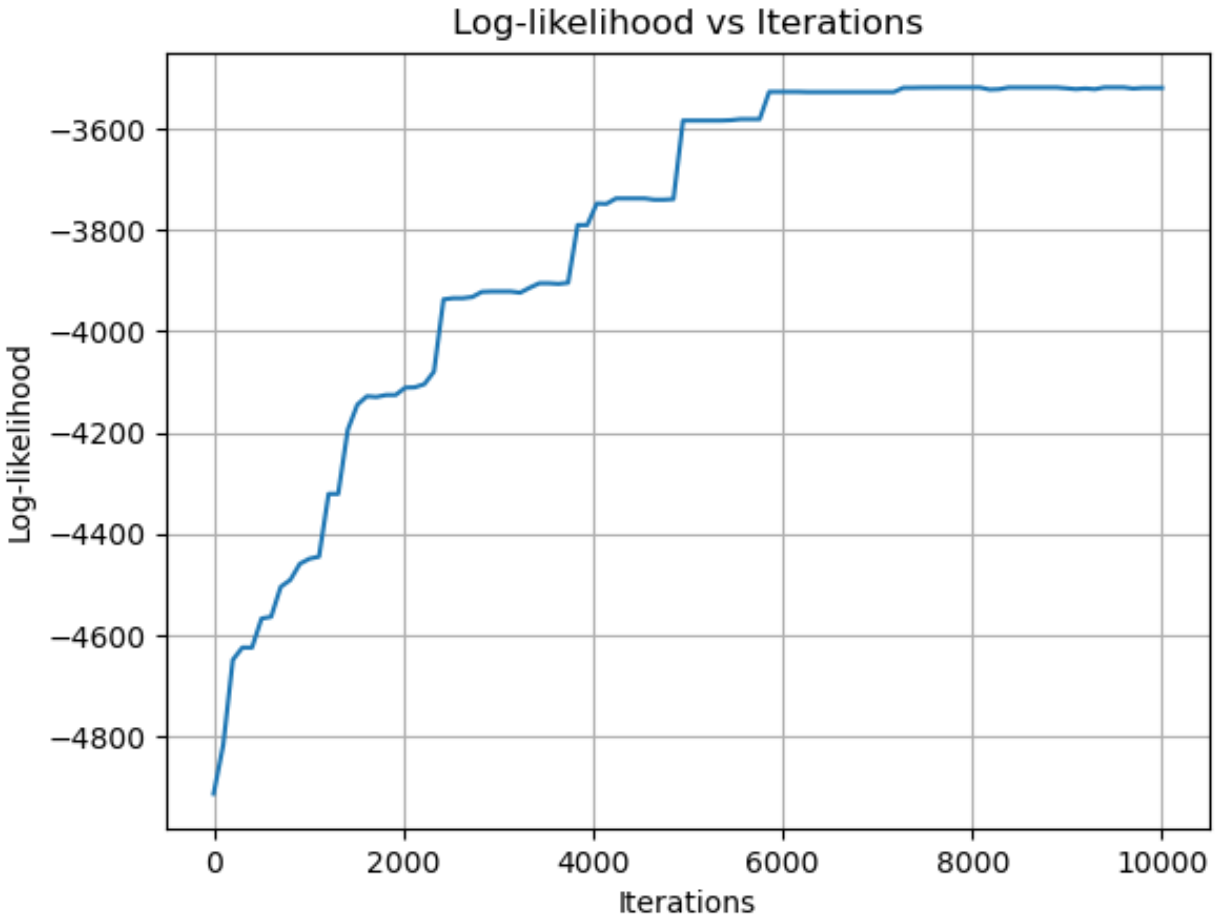


Figure 6: log-likelihood value for 10000 iterations

Question 5.e

A Markov chain is called an ergodic chain if some power of the transition matrix has only positive elements. In other words, it can transit from any state to any state. In our transition matrix, we find zero values which means that we no longer hold the ergodic property. To solve this, we add a small value like 0.001 to our transition matrix before running MCMC. Thus, we restore the ergodicity. The code has already shown in 5(d) and it works good as we get some good starting points. Instead of negative infinite log-likelihood, the none-zero transition matrix avoid a lot of issues.

Question 5.f

If we consider symbol probabilities alone, it would be insufficient for the English language. The structure of the language is influenced by sequences of letters, not just independent symbols. The interpretative ability is reduced when we rely on single-symbol probabilities. This approach would likely produce an output with little semantic coherence, as it disregards how symbols combine within words and phrases.

Using a second-order Markov chain can improve accuracy in text decoding. However, this increases the complexity, as it requires a three-dimensional transition matrix or tensor. Additionally, since we have successfully decoded messages with a first-order Markov chain, the benefit of using a second-order model may only result in a minor improvement in output.

If the encryption scheme permits multiple symbols to map to the same encrypted value, it introduces ambiguity in decoding. Our probability function is defined under the assumption of bijectivity, as are the algorithms that follow. Allowing non-bijective mappings leads to coding errors, causing the entire algorithm to fail.

For Chinese, with over 10,000 characters, the transition matrix would be vast and highly sparse, with most entries near zero. This extreme sparsity and high dimensionality would make many states nearly unreachable within finite

MCMC iterations, significantly limiting the model's efficiency and making decoding impractical.

Question 7

Question 7.a

Find the local extrema of the function $f(x, y) := x + 2y$ subject to the constraint $y^2 + xy = 1$. To apply Lagrangian multiplier, one can write:

$$\nabla f = \lambda \nabla g. \tag{48}$$

Differentiating over x and y separately, we get:

$$\begin{aligned} 1 &= \lambda y, \\ 2 &= \lambda(2y + x). \end{aligned} \tag{49}$$

Together with the constraint, we have three equation for three unknown value. Starting with $y = \frac{1}{\lambda}$ with $\lambda \neq 0$, we find:

$$\begin{aligned} 2 = \lambda(\frac{2}{\lambda} + x) \quad &\Rightarrow \quad x = 0, \\ y^2 = 1 \quad &\Rightarrow \quad y = \pm 1, \end{aligned} \tag{50}$$

where the local extrema are points $(0, 1)$ and $(0, -1)$. Thus, the Lagrangian multiplier is ± 1 and the function value at local extrema is ± 2.

Question 7.b

Suppose we have a numerical routine to evaluate the exponential function $\exp(x)$. Using Newton's method, we can compute $ln(a)$. Let $x = ln(a)$, we the define

$$f(x, a) = e^x - a, \tag{51}$$

where the root of function is $ln(a)$. Applying Newton's method, we have:

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)}, \\ \Rightarrow \quad x_{n+1} &= x_n - \frac{e^{x_n} - a}{e^{x_n}}. \end{aligned} \tag{52}$$

The algorithm always converges as $f'' > 0$.

Bonus Question 4

Question 4.a

We follow the instruction of the question. Here are the plots for filter mode and smooth mode:

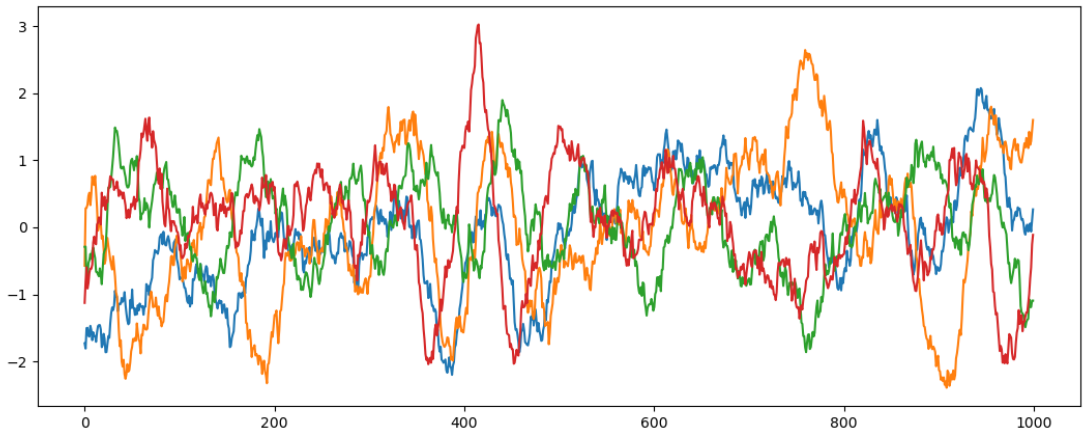


Figure 7: Kalman Filter

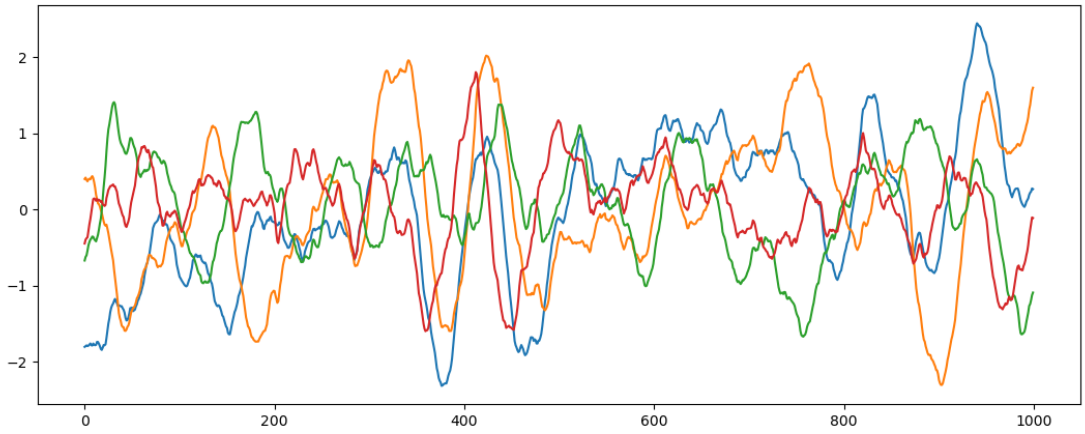


Figure 8: Kalman Smooth

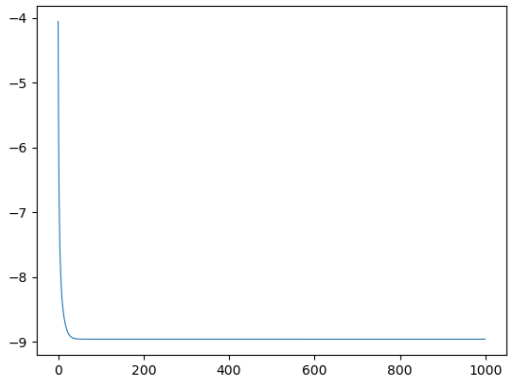
Figure 7 and Figure 8 have a similar trends. The filer mode has more details or zigzag in the plot and the smooth mode looks smooth.

For log determinant, we can find that they all converge quickly. We have the filter log-determinant of covariance at about -9 and smooth on at about -11.

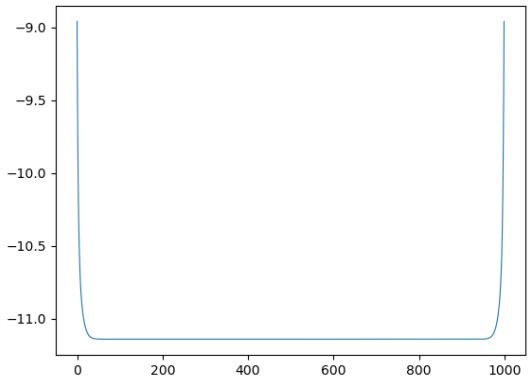
The difference can be explained if we look at the code. For filter mode, we stop at t. However, we have the whole dataset for smooth mode which yields a relatively smaller covariance and a smooth estimate.

The code is here

```
def logdet(A):
    return 2 * np.sum(np.log(np.diag(np.linalg.cholesky(A))))
# %%
# load data
train =  np.loadtxt('ssm_spins.txt')
# %%
# def parameter
theta1 = 2 * np.pi / 180
```



(a) Log-det(V) for Kalman Filter



(b) Log-det(V) for Kalman Smooth

Figure 9: Plot of log determinant for Covariances

```
theta2 = 2 * np.pi / 90
A = 0.99 * np.array([
    [np.cos(theta1), -np.sin(theta1), 0, 0],
    [np.sin(theta1), np.cos(theta1), 0, 0],
    [0, 0, np.cos(theta2), -np.sin(theta2)],
    [0, 0, np.sin(theta2), np.cos(theta2)]
])
C = np.array([
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [1, 0, 0, 1],
    [0, 1, 1, 1],
    [0.5, 0.5, 0.5, 0.5]
])
R = np.eye(5)
Q = np.eye(4) - A @ A.T
# %%
x = train.T
Y0 = np.random.multivariate_normal(np.zeros(4), np.eye(4))
Q0 = np.eye(4)
Y, V, _, L = run_ssm_kalman(x,Y0,Q0,A,Q,C,R, "filt")
Ys, Vs, Vj, Ls = run_ssm_kalman(x,Y0,Q0,A,Q,C,R, mode='smooth')
```

Question 4(a)

Bonus Question 6

Bonus Question 8

Question 8.a

Starting with extreme value theorem of calculus, a continuous function on a compact domain attains its maximum and minimum. In that case, we need to transform the original not-compact space to a compact space. From the hints, we know that unit sphere is good choice. Let us first focus on R_A:

$$\begin{aligned} R_A(x) &= \frac{x^\top A x}{x^\top x} = \frac{q_A(x)}{\|x\|^2}, \\ \Rightarrow \quad R_A(x) &= \frac{x^\top}{\|x\|} A \frac{x}{\|x\|}. \end{aligned} \tag{53}$$

One can define $y = \frac{x}{\|x\|}$, and its normal can be write as:

$$\|y\| = \sqrt{y^\top y} = \sqrt{\frac{x^\top}{\|x\|}\frac{x}{\|x\|}} = 1, \tag{54}$$

which satisfies a unit sphere. Thus, we can rewrite R_A in terms of y:

$$R_A(y) = y^\top A y \quad \text{for } y \in \mathbb{S}, \tag{55}$$

where its domain is compact. Finally, we attain $\sup_{x\in\mathbb{R}^n} R_A(x)$.

Question 8.b

Now, we substitute $x = \sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i$ into R_A:

$$\begin{aligned} R_A(x) &= \frac{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top A \left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)}{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top \left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)}, \\ \Rightarrow \quad R_A(x) &= \frac{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top \left(\sum_{i=1}^n \left(\xi_i^\top x\right)A\xi_i\right)}{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top \left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)}. \end{aligned} \tag{56}$$

Using $A\xi_i = \lambda_i\xi_i$, we can get:

$$R_A(x) = \frac{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top \left(\sum_{i=1}^n \left(\xi_i^\top x\right)\lambda_i\xi_i\right)}{\left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)^\top \left(\sum_{i=1}^n \left(\xi_i^\top x\right)\xi_i\right)}, \tag{57}$$

where we can use ONB property to reduce the summations. Since λ_1 is the biggest eigenvalues, we can deduce:

$$R_A(x) = \frac{\sum_{i=1}^n \lambda_i \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)}{\sum_{i=1}^n \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)} \leq \lambda_1 \times \frac{\sum_{i=1}^n \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)}{\sum_{i=1}^n \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)}, \tag{58}$$

and this is

$$R_A(x) \leq \lambda_1. \tag{59}$$

(c) We can use the similar process in (b). Instead of summing from i to n, we will remove the span $\{\xi_1, \ldots, \xi_k\}$.

$$R_A(x) = \frac{\sum_{i=k+1}^n \lambda_i \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)}{\sum_{i=k+1}^n \left(\left(\xi_i^\top x\right)\xi_i\right)^\top \left(\left(\xi_i^\top x\right)\xi_i\right)}. \tag{60}$$

This equation must be equal or less than λ_{k+1}. From the question, we know that λ_1 is strictly larger than λ_i for $i = k+1, \cdots, n$. Thus, we have

$$R_A(x) \leq \lambda_{k+1} < \lambda_1. \tag{61}$$