1. (a) The joint probability over the observed data and latent variables, expressed in terms of the counts N_d, M_k, A_{dk} and B_{kw}

• W possible words, D documents, K topics

• $A_{dk} = \sum_{i=1}^{N_d} \delta(z_{id} = k)$ is the number of z_{id} taking on value k in document d

• $B_{kw} = \sum_{d=1}^{D} \sum_{i=1}^{N_d} \delta(x_{id} = w)\delta(z_{id} = k)$ is the number of times word w is assigned to topic k across all documents.

• N_d is the total nuber of words in document d

• $M_k = \sum_{w=1}^{W} B_{kw}$ be the total number of words assigned to topic k.

• $\boldsymbol{\phi}$ is a matrix of $K \times \boldsymbol{\phi}_k$ vectors of $W \times 1$ with probabilities of all W words of topic k following a distribution of $Dir(\beta)$

• $\boldsymbol{\theta}$ is a matrix of $D \times \boldsymbol{\theta}_d$ vectors of $K \times 1$ with probabilities of all K topics of document d following a distribution of $Dir(\alpha)$

Starting from $\boldsymbol{\theta}$ variables, we have the below

$$\begin{aligned} p(\boldsymbol{\Theta}) &= \prod_{d=1}^{D} p(\boldsymbol{\theta}_d|\alpha) \\ &= \prod_{d=1}^{D} \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_{dk}^{(\alpha-1)} \end{aligned} \tag{Eq 1.0}$$

Next we examine the latent variable Z that has its topic K distribution driven by $\boldsymbol{\theta}$

$$\begin{aligned} p(\boldsymbol{Z}|\boldsymbol{\Theta}) &= \prod_{d=1}^{D} \left(\prod_{i=1}^{N_d} p(z_{id}|\boldsymbol{\theta}_d) \right) \\ p(z_{id}|\boldsymbol{\theta}_d) &= \prod_{k=1}^{K} \theta_{dk}^{\delta(z_{id}=k)} \\ \prod_{i=1}^{N_d} p(z_{id}|\boldsymbol{\theta}_d) &= \prod_{i=1}^{N_d} \prod_{k=1}^{K} \theta_{dk}^{\delta(z_{id}=k)} \\ &= \prod_{k=1}^{K} \theta_{dk}^{\sum_{i=1}^{N_d} \delta(z_{id}=k)} && (x^a * x^b = x^{a+b}) \\ &= \prod_{k=1}^{K} \theta_{dk}^{A_{dk}} \\ p(\boldsymbol{Z}|\boldsymbol{\Theta}) &= \prod_{d=1}^{D} \left(\prod_{k=1}^{K} \theta_{dk}^{A_{dk}} \right) \end{aligned} \tag{Eq 1.1}$$

Similarly for $\boldsymbol{\Phi}$,

$$\begin{aligned} p(\boldsymbol{\Phi}) &= \prod_{k=1}^{K} p(\boldsymbol{\phi}_k|\beta) \\ &= \prod_{k=1}^{K} \left(\frac{1}{B(\beta)} \prod_{w=1}^{W} (\phi_{kw}^{\beta-1}) \right) \end{aligned} \tag{Eq 1.2}$$

Next we examine the variable X that the word distribution driven by latent variable Z and Φ

$$p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi}) = \prod_{d=1}^{D} \left(\prod_{i=1}^{N_d} \left(\prod_{k=1}^{K} p(x_{id}|z_{id}, \boldsymbol{\phi}_k) \right) \right)$$

$$p(x_{id}|z_{id}, \boldsymbol{\phi}_k) = \prod_{w=1}^{W} \phi_{kw}^{\delta(x_{id}=w)\delta(z_{id}=k)}$$

(where x_{id} is the word i in document d and w is the word in W possible words)

$$\begin{aligned} p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi}) &= \prod_{d=1}^{D}\left(\prod_{i=1}^{N_d}\left(\prod_{k=1}^{K}\left(\prod_{w=1}^{W}\phi_{kw}^{\delta(x_{id}=w)\delta(z_{id}=k)}\right)\right)\right) \\ &= \prod_{k=1}^{K}\left(\prod_{w=1}^{W}\phi_{kw}^{\sum_{d=1}^{D}\sum_{i=1}^{N_d}\delta(x_{id}=w)\delta(z_{id}=k)}\right) \\ &= \prod_{k=1}^{K}\left(\prod_{w=1}^{W}\phi_{kw}^{B_{kw}}\right) \end{aligned} \tag{Eq 1.3}$$

The joint distribution is given by

$$\begin{aligned} p(\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{\theta}, \boldsymbol{\phi}) &= \prod_{k=1}^{K} p(\boldsymbol{\phi}_k) \times \prod_{d=1}^{D} p(\boldsymbol{\theta}_d) \times \prod_{d=1}^{D}\prod_{i=1}^{N_d} p(z_{id}|\boldsymbol{\theta}_d) \times \prod_{d=1}^{D}\prod_{i=1}^{N_d} p(x_{id}|\boldsymbol{\phi}_d, z_{id}) \\ &= \prod_{k=1}^{K}\left(\frac{1}{B(\beta)}\prod_{w=1}^{W}(\phi_{kw}^{\beta-1})\right) \times \prod_{d=1}^{D}\frac{1}{B(\alpha)}\prod_{k=1}^{K}\theta_{dk}^{(\alpha-1)} \\ &\times \prod_{d=1}^{D}\prod_{k=1}^{K}\theta_{dk}^{A_{dk}} \times \prod_{k=1}^{K}\left(\prod_{w=1}^{W}\phi_{kw}^{B_{kw}}\right) \\ &= \prod_{k=1}^{K}\left(\frac{1}{B(\beta)}\prod_{w=1}^{W}(\phi_{kw}^{\beta-1})\right) \times \prod_{d=1}^{D}\frac{1}{B(\alpha)}\prod_{k=1}^{K}\theta_{dk}^{(\alpha-1)} \\ &\times \prod_{d=1}^{D}\prod_{k=1}^{K}\theta_{dk}^{A_{dk}} \times \prod_{k=1}^{K}\left(\boldsymbol{\phi}_w^{M_k}\right) \end{aligned}$$

(b) To derive the Gibbs sampling updates for all the latent variables z_{id} and parameters $\boldsymbol{\theta}_d$ and $\boldsymbol{\phi}_k$,

$$\begin{aligned} \boldsymbol{\Theta}_i &\sim p(\boldsymbol{\Theta}_i|\boldsymbol{Z}, \boldsymbol{\Phi}, \boldsymbol{\Theta}_{-i}, \boldsymbol{X}) \\ \boldsymbol{\Phi}_i &\sim p(\boldsymbol{\Phi}_i|\boldsymbol{Z}, \boldsymbol{\Phi}_{-i}, \boldsymbol{\Theta}, \boldsymbol{X}) \\ \boldsymbol{Z}_i &\sim p(\boldsymbol{Z}_i|\boldsymbol{Z}_{-i}, \boldsymbol{\Phi}, \boldsymbol{\Theta}, \boldsymbol{X}) \end{aligned}$$

Where subscript $_{-i}$ is to mean all bar the i^{th} variable

To get the update for $\boldsymbol{\Theta}$, we can obtain it using the posterior having "observed" the latents $\boldsymbol{Z}$ and given that $\boldsymbol{\Theta}$ are iid and no dependence on $\boldsymbol{\Phi}$ and $\boldsymbol{X}$, we can reduce to

$$\begin{aligned} p(\boldsymbol{\Theta}|\boldsymbol{Z}) &= \frac{p(\boldsymbol{Z}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})}{p(\boldsymbol{Z})} \\ &= \frac{\prod_{d=1}^{D}\left(\prod_{k=1}^{K}\theta_{dk}^{A_{dk}}\right) * \prod_{d=1}^{D}\frac{1}{B(\alpha)}\prod_{k=1}^{K}\theta_{dk}^{(\alpha-1)}}{\int p(\boldsymbol{Z}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})d(\boldsymbol{\Theta})} \\ &\propto \prod_{d=1}^{D}\frac{1}{B(\alpha)}\prod_{k=1}^{K}\theta_{dk}^{(A_{dk}+\alpha-1)} \\ &\propto \prod_{d=1}^{D} Dir(\boldsymbol{A}_d + \alpha) \end{aligned}$$

Differentiating it wrt to θ_{dk} and including the constraint of $\boldsymbol{\theta}_d = \sum_{k=1}^{K}\theta_{dk} = 1$,

$$\frac{\partial lg(p(\boldsymbol{\Theta}|\boldsymbol{Z}))}{\partial\theta_{dk}} = \frac{(A_{dk}+\alpha-1)}{\theta_{dk}} - \lambda = 0$$

$$\frac{\partial lg(p(\boldsymbol{\Theta}|\boldsymbol{Z}))}{\partial \lambda} = -\sum_{k=1}^{K} \theta_{dk} + 1 = 0$$

$$\implies \sum_{k=1}^{K} \theta_{dk} = 1, \theta_{dk} = \frac{A_{dk} + \alpha - 1}{\lambda}$$

$$\theta_{dk}^{MAP} = \frac{A_{dk} + \alpha - 1}{\sum_{k=1}^{K}(A_{dk} + \alpha - 1)}$$

We note that MAP estimates we obtained by maximising wrt $\boldsymbol{\theta}_d$ is the mode of the Dirichlet Distribution $Dir(\boldsymbol{A}_d + \alpha)$

Based on the fact it follows a Dirichlet Distribution, we can obtain expected value of it easily as below

$$\hat{\theta}_{dk} = \frac{A_{dk} + \alpha}{K\alpha + \sum_{k=1}^{K} A_{dk}}$$

Similarly for $\boldsymbol{\Phi}$, which is driven by $\boldsymbol{X}$ and $\boldsymbol{Z}$

$$\begin{aligned} p(\boldsymbol{\Phi}|\boldsymbol{X}, \boldsymbol{Z}) &= \frac{p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi})p(\boldsymbol{\Phi}|\boldsymbol{Z})}{p(\boldsymbol{X}|\boldsymbol{Z})} \\ &= \frac{\prod_{k=1}^{K}\left(\prod_{w=1}^{W} \phi_{kw}^{B_{kw}}\right) \times \prod_{k=1}^{K}\left(\frac{1}{B(\beta)}\prod_{w=1}^{W}(\phi_{kw}^{\beta-1})\right)}{p(\boldsymbol{X}|\boldsymbol{Z})} \\ &\propto \prod_{k=1}^{K}\frac{1}{B(\beta)}\prod_{w=1}^{W}\phi_{kw}^{B_{kw}+\beta-1} \\ &\propto \prod_{k=1}^{K} Dir(\boldsymbol{B}_k + \beta) \end{aligned}$$

Differentiating it wrt to ϕ_{kw} and including the constraint of $\boldsymbol{\phi}_k = \sum_{w=1}^{W}\phi_{kw} = 1$,

$$\frac{\partial lg(p(\boldsymbol{\Phi}|\boldsymbol{X}, \boldsymbol{Z}))}{\partial \phi_{kw}} = \frac{(B_{kw} + \beta - 1)}{\phi_{kw}} - \lambda = 0$$

$$\frac{\partial lg(p(\boldsymbol{\Phi}|\boldsymbol{X}, \boldsymbol{Z}))}{\partial \lambda} = -\sum_{w=1}^{W}\phi_{kw} + 1 = 0$$

$$\implies \sum_{w=1}^{W}\phi_{kw} = 1, \phi_{kw} = \frac{B_{kw} + \beta - 1}{\lambda}$$

$$\phi_{kw}^{MAP} = \frac{B_{kw} + \beta - 1}{\sum_{w=1}^{W}(B_{kw} + \beta - 1)}$$

Based on the fact it follows a Dirichlet Distribution as well, we can obtain expected value of it easily as below

$$\hat{\phi}_{kw} = \frac{B_{kw} + \beta}{W\beta + \sum_{w=1}^{W} B_{kw}}$$

The conditional posterior of z

$$\begin{aligned} p(\mathbf{z}_i|\boldsymbol{Z}_{-i}, \boldsymbol{X}) &= \frac{p(\boldsymbol{Z}, \boldsymbol{X})}{p(\boldsymbol{Z}_{-i}, \boldsymbol{X}|)} \\ &= \frac{p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})}{p(\boldsymbol{X}_{-i}|\boldsymbol{Z}_{-i})p(\boldsymbol{Z}_{-i})p(\mathbf{x}_i)} \\ &= \frac{p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})}{p(\boldsymbol{X}_{-i}|\boldsymbol{Z}_{-i})p(\boldsymbol{Z}_{-i})p(\mathbf{x}_i)} \\ &\propto \frac{p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})}{p(\boldsymbol{X}_{-i}|\boldsymbol{Z}_{-i})p(\boldsymbol{Z}_{-i})} \end{aligned} \tag{Eq 1.4}$$

Where $\boldsymbol{Z}_{-i}$ means all $\boldsymbol{Z}$ bar the i^{th} variable $\mathbf{z}_i$ and $\boldsymbol{X}_{-i}$ means all $\boldsymbol{X}$ bar the i^{th} variable $\mathbf{x}_i$.

This gives us the probability of z_{id} being allocated to a particular topic k using all the current information bar the observed variable $\mathbf{x}_i$ itself.

There are 2 things to note here.

First and foremost, this step of updating is invariant to the order such that i can be any observations and we will require to use the latest information set for the update.

Secondly, we note that this is very similar to the joint distribution of $\boldsymbol{X}$ and $\boldsymbol{Z}$ where $p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z}) = p(\boldsymbol{X}, \boldsymbol{Z})$. We can obtain this by integrating $\boldsymbol{\Phi}$ out of $p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi})$ and multiplying by the integral of $p(\boldsymbol{Z}|\boldsymbol{\Theta})$ wrt $\boldsymbol{\Theta}$. In a way, it is like the Metropolis Hastings algorithm where we look at $\frac{p(x_i+1)}{p(x_i)}$ assuming that he proposal density of q is symmetrical. We look at the ratios of the density and thus simplifying the calculations. In Gibbs sampling case, we look at the ratio of the joint distribution with and without the i^{th} variable.

Thus, for the Gibbs Sampling updates, we will obtain z_{id} at each step of the update repeating this for all variables until convergence.

$$
\begin{aligned}
p(\boldsymbol{X}, \boldsymbol{Z}) &= p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z}) \\
&= \int \left(p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi})p(\boldsymbol{\Phi})\right) d(\boldsymbol{\Phi}) \times \int \left(p(\boldsymbol{Z}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})\right) d(\boldsymbol{\Theta}) \\
p(\boldsymbol{Z}) = \int \left(p(\boldsymbol{Z}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})\right) d(\boldsymbol{\Theta}) &= \int \left(\prod_{d=1}^{D} \left(\prod_{k=1}^{K} \theta_{dk}^{A_{dk}}\right)\right) \left(\prod_{d=1}^{D} \frac{1}{B(\alpha)} \prod_{k=1}^{K} \theta_{dk}^{(\alpha-1)}\right) d(\boldsymbol{\Theta}) \\
&\qquad \text{(From Eq 1.0 and 1.1)} \\
&= \prod_{d=1}^{D} \frac{1}{B(\alpha)} \int \left(\prod_{k=1}^{K} \theta_{dk}^{A_{dk}+\alpha-1}\right) d(\boldsymbol{\theta}_d) \\
&\qquad \text{(As } \boldsymbol{\theta}_d \text{ is independent across } d\text{)} \\
&= \prod_{d=1}^{D} \frac{1}{B(\alpha)} B(\boldsymbol{A}_d + \alpha) \int \frac{1}{B(\boldsymbol{A}_d + \alpha)} \prod_{k=1}^{K} \theta_{dk}^{A_{dk}+\alpha-1} d(\boldsymbol{\theta}_d) \\
&= \prod_{d=1}^{D} \frac{B(\boldsymbol{A}_d + \alpha)}{B(\alpha)} \qquad \text{(Eq 1.5)} \\
&= \prod_{d=1}^{D} \frac{\Gamma\left(\sum_{k=1}^{K} \alpha\right)}{\prod_{k=1}^{K} \Gamma(\alpha)} \frac{\prod_{k=1}^{K} \Gamma(A_{dk} + \alpha)}{\Gamma\left(\sum_{k=1}^{K} A_{dk} + \alpha\right)} \qquad \text{(Eq 1.6)} \\
p(\boldsymbol{X}|\boldsymbol{Z}) = \int \left(p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi})p(\boldsymbol{\Phi})\right) d(\boldsymbol{\Phi}) &= \int \prod_{k=1}^{K} \left(\prod_{w=1}^{W} \phi_{kw}^{B_{kw}}\right) \prod_{k=1}^{K} \left(\frac{1}{B(\beta)} \prod_{w=1}^{W} (\phi_{kw}^{\beta-1})\right) d(\boldsymbol{\Phi}) \\
&= \prod_{k=1}^{K} \frac{1}{B(\beta)} \int \left(\prod_{w=1}^{W} \phi_{kw}^{B_{kw}+\beta-1}\right) d(\boldsymbol{\phi}_k) \\
&= \prod_{k=1}^{K} \frac{B(\boldsymbol{B}_k + \beta)}{B(\beta)} \qquad \text{(Eq 1.7)} \\
&= \prod_{k=1}^{K} \frac{\Gamma\left(\sum_{w=1}^{W} \beta\right)}{\prod_{w=1}^{W} \Gamma(\beta)} \frac{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)} \qquad \text{(Eq 1.8)} \\
p(\mathbf{z}_i|\boldsymbol{Z}i, X) &\propto p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z}) \\
&= \left(\prod_{d=1}^{D} \frac{B(\boldsymbol{A}_d + \alpha)}{B(\alpha)}\right) \left(\prod_{k=1}^{K} \frac{B(\boldsymbol{B}_k + \beta)}{B(\beta)}\right)
\end{aligned}
$$

(c)

To integrate out the $\boldsymbol{\Theta}$ and the $\boldsymbol{\Phi}$ terms from the joint distribution of X and Z,

$$\begin{aligned} p(\boldsymbol{X}, \boldsymbol{Z}) &= p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z}) \\ &= \int \left(p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\Phi})p(\boldsymbol{\Phi})\right) d(\boldsymbol{\Phi}) \times \int \left(p(\boldsymbol{Z}|\boldsymbol{\Theta})p(\boldsymbol{\Theta})\right) d(\boldsymbol{\Theta}) \end{aligned}$$

Thus we can see that the joint distribution is

$$p(\boldsymbol{X}, \boldsymbol{Z}) = \left(\prod_{d=1}^{D} \frac{B(\boldsymbol{A}_d + \alpha)}{B(\alpha)}\right)\left(\prod_{k=1}^{K} \frac{B(\boldsymbol{B}_k + \beta)}{B(\beta)}\right) \tag{Eq 1.9}$$

$$\begin{aligned} &= \left(\prod_{d=1}^{D} \frac{\Gamma\left(\sum_{k=1}^{K} \alpha\right)}{\prod_{k=1}^{K} \Gamma(\alpha)} \frac{\prod_{k=1}^{K} \Gamma(A_{dk} + \alpha)}{\Gamma\left(\sum_{k=1}^{K} A_{dk} + \alpha\right)}\right) \\ &\times \left(\prod_{k=1}^{K} \frac{\Gamma\left(\sum_{w=1}^{W} \beta\right)}{\prod_{w=1}^{W} \Gamma(\beta)} \frac{\prod_{w=1}^{W} \Gamma(B_{kw} + \beta)}{\Gamma\left(\sum_{w=1}^{W} B_{kw} + \beta\right)}\right) \end{aligned} \tag{Eq 1.10}$$

(d) Using the Gibbs sampling equation from part b and the joint distribution with $\boldsymbol{\Theta}$ and $\boldsymbol{\Phi}$ integrated out, we can easily construct the collapsed Gibbs sampler from Eq 1.4

$$\begin{aligned} p(\mathbf{z}_i|\boldsymbol{Z}_{-i}, \boldsymbol{X}) &= \frac{p(\boldsymbol{Z}, \boldsymbol{X})}{p(\boldsymbol{Z}_{-i}, \boldsymbol{X})} \\ &\propto \frac{p(\boldsymbol{X}|\boldsymbol{Z})p(\boldsymbol{Z})}{p(\boldsymbol{X}_{-i}|\boldsymbol{Z}_{-i})p(\boldsymbol{Z}_{-i})} \end{aligned}$$

We note that this is in the same form as Eq 1.9 and thus,

$$\begin{aligned} &= \frac{\left(\prod_{k=1}^{K} \frac{B(\boldsymbol{B}_k+\beta)}{B(\beta)}\right)\left(\prod_{d=1}^{D} \frac{B(\boldsymbol{A}_d+\alpha)}{B(\alpha)}\right)}{\left(\prod_{k=1}^{K} \frac{B(\boldsymbol{B}_{k,-i}+\beta)}{B(\beta)}\right)\left(\prod_{d=1}^{D} \frac{B(\boldsymbol{A}_{d,-i}+\alpha)}{B(\alpha)}\right)} \\ &\propto \prod_{k=1}^{K} \frac{B(\boldsymbol{B}_k + \beta)}{B(\boldsymbol{B}_{k,-i} + \beta)} \prod_{d=1}^{D} \frac{B(\boldsymbol{A}_d + \alpha)}{B(\boldsymbol{A}_{d,-i} + \alpha)} \end{aligned}$$

Where the subscript of $_{-i}$ signifies all bar the i_{th} one

For a single element of the first term,

$$\begin{aligned} \frac{B(B_{kw} + \beta)}{B(B_{kw,-i} + \beta)} &= \frac{\prod_{w=1}^{W} \Gamma(\beta + B_{kw})}{\Gamma(\sum_{w=1}^{W} \beta + B_{kw})} \frac{\Gamma(\sum_{w=1}^{W} \beta + B_{kw,-i})}{\prod_{w=1}^{W} \Gamma(\beta + B_{kw,-i})} \\ &= \frac{\prod_{w=1}^{W} \Gamma(\beta + B_{kw,-i} + 1)}{\Gamma(\sum_{w=1}^{W} \beta + B_{kw,-i} + 1)} \frac{\Gamma(\sum_{w=1}^{W} \beta + B_{kw,-i})}{\prod_{w=1}^{W} \Gamma(\beta + B_{kw,-i})} \\ &= \frac{\beta + B_{kw,-i}}{\sum_{w=1}^{W} \beta + B_{kw,-i}} \frac{\prod_{w=1}^{W} \Gamma(\beta + B_{kw,-i})}{\Gamma(\sum_{w=1}^{W} \beta + B_{kw,-i})} \frac{\Gamma(\sum_{w=1}^{W} \beta + B_{kw,-i})}{\prod_{w=1}^{W} \Gamma(\beta + B_{kw,-i})} \\ &= \frac{\beta + B_{kw,-i}}{\sum_{w=1}^{W} \beta + B_{kw,-i}} \\ &= \frac{\beta + B_{kw,-i}}{W\beta + M_{k,-i}} \end{aligned}$$

Similarly for the second term,

$$\frac{B(A_{dk} + \alpha)}{B(A_{dk,-i} + \alpha)} = \frac{\alpha + A_{dk,-i}}{\sum_{k=1}^{K} \alpha + A_{dk,-i}}$$

$$p(\mathbf{z}_i|\boldsymbol{Z}_{-i}, \boldsymbol{X}) \propto \prod_{k=1}^{K} \left(\frac{\beta + B_{kw,-i}}{W\beta + M_{k,-i}}\right) \prod_{d=1}^{D} \left(\frac{\alpha + A_{dk,-i}}{\sum_{k=1}^{K} \alpha + A_{dk,-i}}\right)$$

We note that at each step of Gibbs Sampling update, we fix $z_{id} = k$ and thus, for a fixed topic denominator can be dropped leaving us with

$$p(z_{id} = k|\boldsymbol{Z}_{-i}, \boldsymbol{X}) \propto \prod_{k=1}^{K} \left(\frac{\beta + B_{kw,-i}}{W\beta + M_{k,-i}}\right) \prod_{d=1}^{D} \left(\frac{\alpha + A_{dk,-i}}{\sum_{k=1}^{K} \alpha + A_{dk,-i}}\right)$$

(e) I will generate the hyperpriors based on my understanding of the dataset of text. However, without sufficient prior knowledge, I will initiate the hyperpriors as 1. This avoids skewing/anchoring the Dirichlet distribution due to my assumptions of topic and words distrbutions.

I will generate samples for the hyperpriors by going through similar documents and compare how differentiated the documents are (a load of common topics/words in a document/topic?).

2. (a) The maximum likelihood estimates of these probabilities as a function of counts of numbers of occurrences of symbols and pairs of symbols will be given by:

$$lim_{i->\infty} p(s_1 = \alpha) \equiv \phi(\alpha)$$
$$p(s1 = \alpha | s_{i-1} = \beta) \equiv \psi(\alpha, \beta)$$

with these 2 definitions in mind, we will represent the ML estimates of them as counts

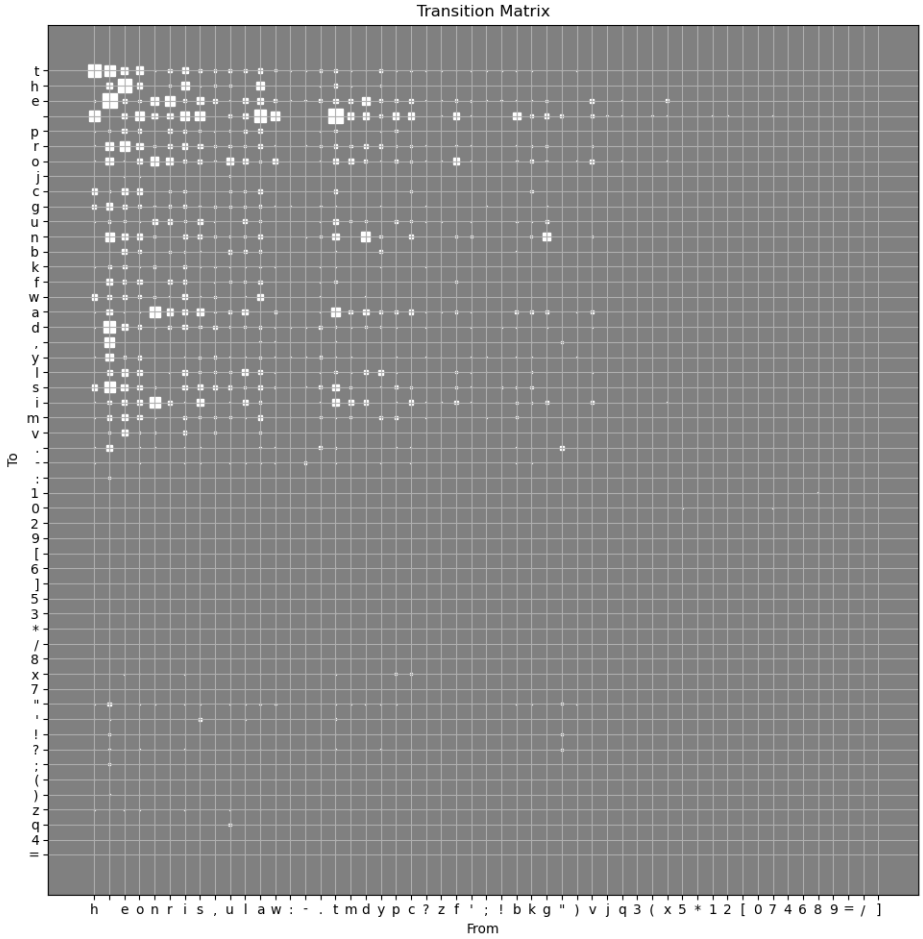
$$\phi(\alpha) = \frac{\sum \delta(s_i = \alpha)}{N}$$
$$\psi(\alpha, \beta) = \frac{\sum \delta(s_i = \alpha, s_{i-1} = \beta)}{N}$$

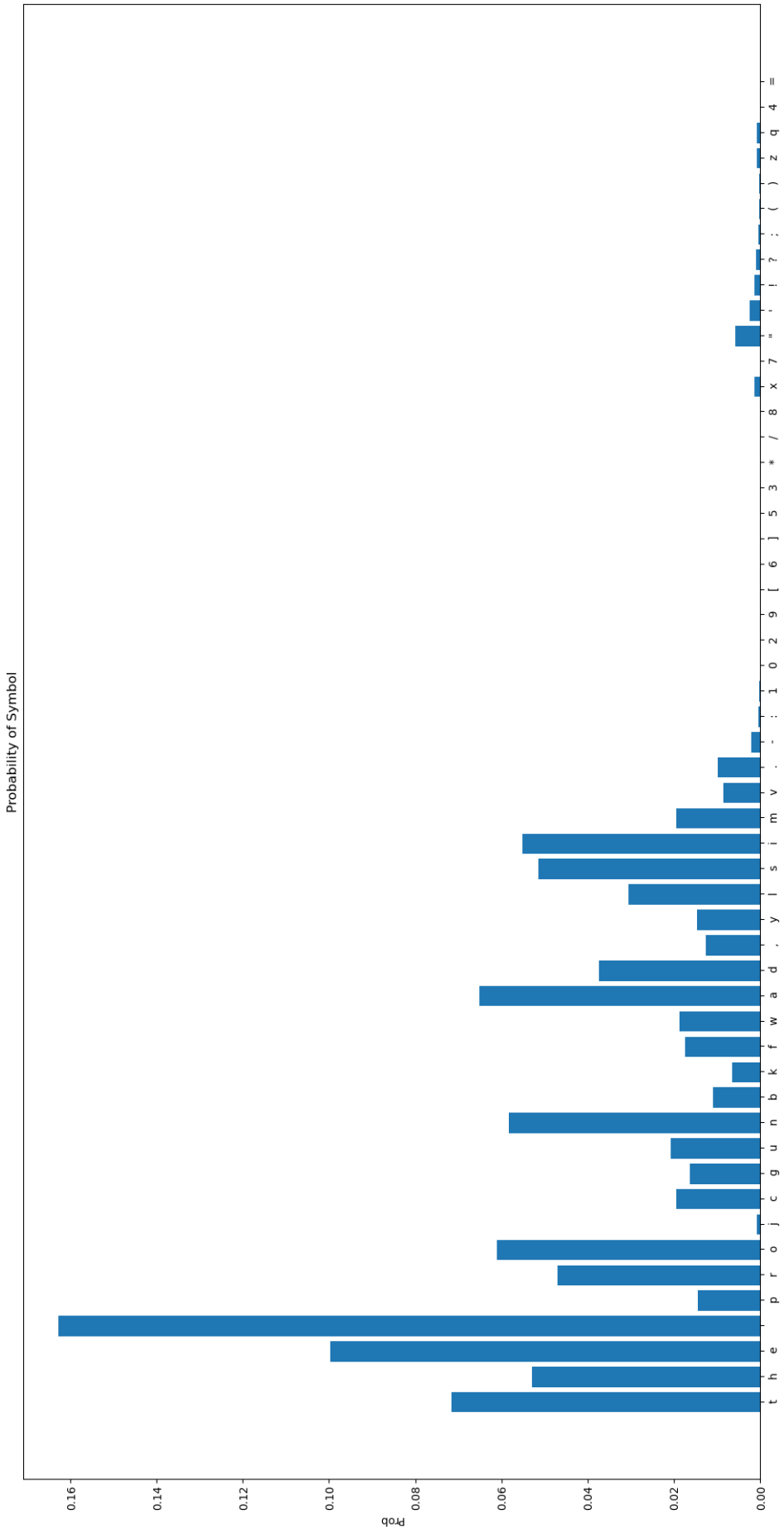
where $\delta(s_i = \alpha)$ is a indicator that shows 1 when symbol s_i equals to α in the list of possible symbols and N is the total number of symbols in the text. Thereafter, I used the Power Method to make the transition matrix irreducible and aperiodic.

$$P_{old} = P_{init}$$
$$P_{new} = P^T P_{old}$$

This is done in a loop until $||P_{new} - P_{old}|| < threshold$ where I have set my threshold as exp(-10). I noticed that the choice of P_{init} does not impact the solution a great deal however for the purpose of this exercise, I initiated it as uniformly distributed.
Using the information from War and Peace.txt, I computed and plotted the Hinton Plots for the 2 functions below. I note that there are several obvious yet counter-intuitive transitions in the matrices below which are quite interesting for example, the alphabet "t" appears most frequently after a blank space whereas "e" appears most frequently before a blank space.

Hinton Plot of Transition Matrix $\psi(\alpha, \beta)$


Transition Matrix
To
From
t h e _ p r o j c g u n b k f w a d , _ y l s i m v . - ; 1 0 2 9 [6] 5 3 * / 8 x 7 " ' ! ? : ( ) z q 4 =
h e o n r i s , u l a w : - . t m d y p c ? z f ' ; ! b k g ") v j q 3 (x 5 * 1 2 [0 7 4 6 8 9 = /]


Probability of Symbol
Prob
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
t h e p r o j c g u n b k f w a d , y l s i m v . - : 1 0 2 9 [6] 5 3 * / 8 x 7 " ' ! ? ; ( ) z q 4 =

Top 10	Ln(Prob)	Bot 10	Ln(Prob)
" "	-1.8148	[	-14.9666
e	-2.3055	]	-14.9666
t	-2.6365	=	-14.2734
a	-2.7319	4	-11.8311
o	-2.7968	/	-11.5993
n	-2.8430	9	-11.4112
i	-2.8982	7	-11.2777
h	-2.9384	5	-10.9593
s	-2.9657	6	-10.9235
r	-3.0587	3	-10.8557

```
import numpy as np
import random
from copy import deepcopy
import matplotlib.pyplot as plt

def counter(str, pair=False):
    '''
    :param str: String of words
    :param pair: True if we are looking at symbol transitions in pairs, False to
         count number of symbols in str
    :return: Dictionary of counts be it pair or singleton
    '''
    output = {}
    if pair:
        for i in range(len(str) - 1):
            if str[i] not in output.keys():
                output[str[i]] = {str[i + 1]: 1}
            else:
                if str[i + 1] not in output[str[i]].keys():
                    output[str[i]][str[i + 1]] = 1
                else:
                    output[str[i]][str[i + 1]] += 1
    else:
        for key in str:
            if key not in output.keys():
                output[key] = 1
            else:
                output[key] += 1
    return output


def random_sigmainv(sym, sigmainv=None):
    '''
    :param sym: List of symbols that are used for encoding the msg
    :return: Dictionary of Encoding (Key) to Decode (Value)
    '''
    sym = deepcopy(sym)
    if sigmainv is not None:
        sigmainv = deepcopy(sigmainv)
        for key in sigmainv.keys():
            sym.remove(key)
    else:
        sigmainv = {}
    r = random.sample(range(len(sym)), len(sym))
    output = [[symbols, sym[i]] for symbols, i in zip(sym, r)]
    for i, j in output:
        sigmainv[i] = j
    return sigmainv


def simple_sigmainv(init_dict, msg):
    '''
    Creating a simple sigmainv based on init_dict
```

```
    :param init_dict: Dictionary of init dict where we have characters that are more
        frequent
    :param msg: String of msg to be decoded
    :return:
    '''
    msg_dict = counter(msg)
    output = {}
    for key, ref in zip(msg_dict, init_dict):
        output[key] = ref
    return output


def swap(sigmainv):
    '''
    Swap a random sigmainv with another random sigmainv
    :param sigmainv: Dictionary of Encrypt (Key) to Decrypt (Value)
    :return: Dictionary of Encoding (Key) to Decode (Value)
    '''
    s_1 = random.choice([key for key in sigmainv.keys()])
    s_2 = random.choice([key for key in sigmainv.keys()])
    # Ensures that s_1 and s_2 are different
    while s_1 == s_2:
        s_2 = random.choice([key for key in sigmainv.keys()])
    # Swap index
    deepcopy_sigmainv = deepcopy(sigmainv)
    deepcopy_sigmainv[s_1], deepcopy_sigmainv[s_2] = deepcopy_sigmainv[s_2],
        deepcopy_sigmainv[s_1]
    return deepcopy_sigmainv


def decode(msg, sigmainv):
    '''
    :param msg: String of encoded msg
    :param sigmainv: Dictionary of Encoding (Key) to Decode (Value)
    :return: String of decoded msg
    '''
    output = [sigmainv[encrypt] for encrypt in msg]
    return ''.join(output)


def loglikelihood(msg, transit_dict):
    '''
    Calculates the probability of getting such a msg based on probabilities in
        transition dictionary
    :param msg: String of decoded msg
    :param transit_dict: Dictionary of Dictionary of transition probs of symbols e.g
        . {'a':{'b':0.1,'e':0.2},'e':{'t':0.5}}
    :return: Float64 of loglikelihood of msg
    '''
    lglikelihood = 0
    for i in range(1, len(msg)):
        if msg[i - 1] in transit_dict.keys():
            if msg[i] in transit_dict[msg[i - 1]].keys():
                lglikelihood += np.log(transit_dict[msg[i - 1]][msg[i]])
    return lglikelihood

def accept_proposal(proposed_lg,curr_lg):
    '''
    :param proposed_lg: float64
    :param curr_lg: float64
    :return: Check if we should accept or reject the proposal swap given the 2
        loglikelihoods
    True if proposal is accepted and False if not
    '''
    if proposed_lg > curr_lg:
        return True
    else:
        if np.random.rand() < np.exp(proposed_lg - curr_lg):
            return True
    return False
```

```
118
119 def hinton(matrix, title, ax=None):
120     """
121     Draw Hinton diagram for visualizing a weight matrix.
122     Reference from
123     https://matplotlib.org/3.1.1/gallery/specialty_plots/hinton_demo.html
124     """
125     index = matrix.index
126     columns = matrix.columns
127     matrix = np.array(matrix)
128     ax = ax if ax is not None else plt.gca()
129     max_weight = 2 ** np.ceil(np.log(np.abs(matrix).max()) / np.log(2))
130
131     ax.patch.set_facecolor('gray')
132     ax.set_aspect('equal', 'box')
133     for (x, y), w in np.ndenumerate(matrix):
134         color = 'white' if w > 0 else 'black'
135         size = np.sqrt(np.abs(w) / max_weight)
136         rect = plt.Rectangle([x - size / 2, y - size / 2], size, size,
137                              facecolor=color, edgecolor=color)
138         ax.add_patch(rect)
139
140     ax.set_title(title)
141     ax.autoscale_view()
142     ax.invert_yaxis()
143     ax.set_xlabel("From")
144     ax.set_ylabel("To")
145     ax.set_xticks(np.arange(len(index)))
146     ax.set_yticks(np.arange(len(columns)))
147     ax.set_xticklabels(list(index))
148     ax.set_yticklabels(list(columns))
149     ax.grid()
150
151 import numpy as np
152 import pandas as pd
153 import matplotlib.pyplot as plt
154 from Assignment3.MH import counter, swap, decode, random_sigmainv, loglikelihood,
      hinton, accept_proposal
155
156 with open('Assignment3\symbols.txt') as f:
157     sym = [line.rstrip() for line in f]
158     sym = [' ' if c == '' else c for c in sym]
159 with open('Assignment3\message.txt') as f:
160     msg = f.readlines()[0]
161 with open('Assignment3\war-and-peace.txt') as f:
162     wnp = f.read()
163     wnp = wnp.replace("\n", "").lower()
164 unique = ''.join(set(wnp))
165 for char in unique:
166     if char not in sym and char != " ":
167         wnp = wnp.replace(char, "")
168
169 '''
170 Create a dictionary that counts the transition of each characters to the next
171 as well as the number of times the character appears in the text
172 '''
173 transit_dict = counter(wnp, True)
174 init_dict = counter(wnp, False)
175
176 '''
177 Place the transition dictionary into a DataFrame and normalise it.
178 '''
179 df_transit_matrix = pd.DataFrame(transit_dict).fillna(0)
180 normalizer = np.einsum('ij->', df_transit_matrix)
181 df_transit_matrix = df_transit_matrix / normalizer
182 threshold = np.exp(-10)
183 x = 0
184
185 '''
```

```
Create P_inf by using P_new = P_init^T @ P_old where P_init is a uniform random
    distribution
'''
p_init = np.random.rand(df_transit_matrix.shape[0], df_transit_matrix.shape[1])
p = df_transit_matrix.copy(deep=True)
p_new = np.einsum('ij,ik->jk', p_init, p)
# p_new = np.einsum('ij,ik->jk', p_init, p)
while x > threshold:
    p_new = np.einsum('ij,ik->jk', p_new, p)
    x = np.linalg.norm(p - p_new)
    p = p_new
df_transit_matrix = p.copy(deep=True)

'''
Create normalized data chart for how character distribution in War and Peace
'''
df_init = pd.DataFrame(init_dict, index=init_dict.keys()).iloc[:1:].T
normalizer = np.einsum('ij->', df_init)
df_init = df_init / normalizer
df_init.columns = ['Prob']

'''
Plot Hinton of Transition Prob and BarChart of character distribution
'''
fig, axs_transit = plt.subplots(1, 1, constrained_layout=True)
hinton(df_transit_matrix, "Transition Matrix", axs_transit)
fig, axs_init = plt.subplots(1, 1, constrained_layout=True)
axs_init.bar(df_init.index, df_init['Prob'])
axs_init.set_title('Probability of Symbol')
axs_init.set_ylabel('Prob')
top10 = df_init.sort_values(by="Prob", ascending=False).iloc[:10, :]
bot10 = df_init.sort_values(by="Prob", ascending=True).iloc[:10, :]
print(pd.DataFrame.join(top10.reset_index(),bot10.reset_index(),lsuffix='_Top10',
    rsuffix='_Bot10'))

'''
Simple analysis of the data structure and frequency of the encoded msg
'''
msg_dict = counter(msg)
m = pd.DataFrame(msg_dict, index=range(len(msg_dict))).iloc[0].sort_values(ascending
    =False)

'''
Decode message and print our decoded message every 500 iterations
'''
sigmainv = random_sigmainv(sym)
for iter in range(int(10001)):
    curr_msg = decode(msg, sigmainv)
    curr_lg = loglikelihood(curr_msg, transit_dict)
    proposed_sigmainv = swap(sigmainv)
    proposed_msg = decode(msg, proposed_sigmainv)
    proposed_lg = loglikelihood(proposed_msg, transit_dict)
    if accept_proposal(proposed_lg, curr_lg):
        sigmainv = proposed_sigmainv
    if iter % 500 == 0:
        print("\item Run {} has loglikelihood of {:.2f}. \\\\Message is \\emph
            {{\"{}\"}}".format(iter, curr_lg,decode(msg,sigmainv)[:60]))

# pickle.dump(sigmainv, open( "sigmainv.p", "wb" ))
```

(b) i. The latent variables $\sigma(s)$ for different symbols s are not independent. This is because, from the previous table, we can observe that there is a higher probability of an "e" after a "h" and this means that there is dependency in the symbol sequence. Thus, the probability of the latent variable $\sigma(e)$ is much higher than $\sigma(z)$ if the preceding $\sigma(h)$. Also, no two encoder σ share the same decoded character i.e.$\sigma(a) =!$ then $\sigma(b) \neq!$. In a way, we can see it as

drawing balls from a pot without replacement.

$$p(e_1, e_2, ..., e_n|\sigma) = \prod_{i=2}^{n} p(e_i|e_{i-1}, \sigma)$$
$$= \prod_{i=2}^{n} \psi(e_i, e_{i-1})$$

(c) Metropolis Hastings Chain where the proposal given by choosing two symbols at random and swapping the corresponding encrypted symbols. If the proposal density increases, we will accept the proposal and set them as the new values.

i. First and foremost, we note that this proposal is symmetric as probability of choosing s given s' is exactly the same as the probability of choosing s' given s since they are drawn randomly from the same pool of symbols. This means that we can easily remove the proposal densities in the rejection kernel A.

ii. The proposal probability $S(\sigma -> \sigma')$ depends on σ and σ' in that it is the product of them. I.e. The swap in them will result in

iii. The acceptance probability is given by $A(\sigma'|\sigma) = min(1, \frac{q(\sigma'|\sigma)}{q(\sigma|\sigma')} \frac{p(\sigma')}{p(\sigma)})$. In words, it will mean that if there is an improvement in likelihood from the character swap, it will always be executed. If there is a decrease in likelihood, it will depend on the ratio of the the decrease (A) as compared to a standard uniform random draw between 0 and 1. If A is larger than the standard uniform random draw, we will accept the character swap and if it is less, we will reject the proposed swap.

(d) I initialised my sigmainv function by allocating the most frequent symbol in the message to the most frequent character in $\phi(\alpha)$. That allowed me to start the algorithm at a log-likelihood of about 10k.

However, there were some issues on repeated runs where I started drifting into local maximums and failing to get out of it.

i. Run 0 has loglikelihood of 9640.56.
Message is "*the peprohjcgeuhne rgcebokhcgufkcepcugwe peaud,cgejubce cewr*"

ii. Run 500 has loglikelihood of 12940.52.
Message is "*en ft troneth ond frht hoanthogat ttohs ft dolith eoht ft sr*"

iii. Run 1000 has loglikelihood of 13314.04.
Message is "*an od doandth ent ooht haenthemet dtehr od fesith deht ot ro*"

iv. Run 1500 has loglikelihood of 13639.09.
Message is "*an td doingor end toro hienoremeo doert td feshor geho to to*"

v. Run 2000 has loglikelihood of 13969.57.
Message is "*en so oainger and sare lidneramde oeart so father gale se ta*"

vi. Run 2500 has loglikelihood of 14023.42.
Message is "*en to oainger and tare milneradle oears to father game te sa*"

vii. Run 3000 has loglikelihood of 14043.80.
Message is "*en to oainger and tare dilneramle oears to father gade te sa*"

viii. Run 3500 has loglikelihood of 14044.34.
Message is "*en to oainger and tare cilneramle oears to father gace te sa*"

ix. Run 4000 has loglikelihood of 14054.96.
Message is "*en to oainger and tare cilneramle oears to iather gace te sa*"

x. Run 4500 has loglikelihood of 14069.11.
Message is "*en ty yainger and tare milneracle years ty iather game te sa*"

xi. Run 5000 has loglikelihood of 14070.18.
Message is "*en to oainger and tare milneracle oears to iather game te sa*"

xii. Run 5500 has loglikelihood of 14068.15.
Message is "*en to oainger and tare milneracle oears to iather game te sa*"

xiii. Run 6000 has loglikelihood of 14068.15.
Message is "*en to oainger and tare milneracle oears to iather game te sa*"

xiv. Run 6500 has loglikelihood of 14069.23.
Message is "*en to oainger and tare milneraple oears to iather game te sa*"

xv. Run 7000 has loglikelihood of 14068.15.
Message is "*en to oainger and tare milneracle oears to iather game te sa*"

xvi. Run 7500 has loglikelihood of 14073.21.
Message is "*en ty yainger and tare milnerable years ty iather game te sa*"

xvii. Run 8000 has loglikelihood of 14071.02.
Message is "*en ty yainger and tare milnerable years ty iather game te sa*"

xviii. Run 8500 has loglikelihood of 14077.03.
Message is "*en ty yainger and tare milneraple years ty iather game te sa*"

xix. Run 9000 has loglikelihood of 14080.76.
Message is "*en ty yainger and tare milnerahle years ty iather game te sa*"

xx. Run 9500 has loglikelihood of 14077.24.
Message is "*en ty yainger and tare milnerahle years ty iather game te sa*"

xxi. Run 10000 has loglikelihood of 14077.24.
Message is "*en ty yainger and tare milnerahle years ty iather game te sa*"

Only by repeating the sampler with a random starting point multiple times and keeping all the best records did I manage to get a sensible decoded message but I noticed that, often, the log likelihood of the sensible decoded message (below) is actually lower than the non-sensible ones (above).

i. Run 0 has loglikelihood of 3960.84.
Message is "*aduoxux-cd,*/ued"uo-/*usc=d*/eh=*ux*e/8uoxu(e;w*/u,es*uo*u8-*"

ii. Run 500 has loglikelihood of 11732.64.
Message is "*urelfef,orgi earnel, iebodri a.diefia selfe*"*athi egabielies,*"

iii. Run 1000 has loglikelihood of 12471.04.
Message is "*ur nb beorgil arm neli ,odrilacdi bials nb fathil ga,i ni se*"

iv. Run 1500 has loglikelihood of 12825.06.
Message is "*en ly yuanfor ing luro mapnoricpo yoirs ly dithor fimo lo su*"

v. Run 2000 has loglikelihood of 13294.95.
Message is "*on py yuinwer and pure milneracle years py gather wame pe su*"

vi. Run 2500 has loglikelihood of 13329.38.
Message is "*on cy yuinger and cure vilneraple years cy wather gave ce su*"

vii. Run 3000 has loglikelihood of 13416.81.
Message is "*on cy yiunger and cire vulnerafle years cy mather gave ce si*"

viii. Run 3500 has loglikelihood of 13416.81.
Message is "*on cy yiunger and cire vulnerafle years cy mather gave ce si*"

ix. Run 4000 has loglikelihood of 13440.38.
Message is "*on cy yiunger and cire vulnerafle years cy mather gave ce si*"

x. Run 4500 has loglikelihood of 13666.99.
Message is "*in my younker and more vulneracle years my father kave me so*"

xi. Run 5000 has loglikelihood of 13666.99.
Message is "*in my younker and more vulneracle years my father kave me so*"

xii. Run 5500 has loglikelihood of 13667.74.
Message is "*in my younker and more vulneracle years my father kave me so*"

xiii. Run 6000 has loglikelihood of 13667.74.
Message is "*in my younker and more vulneracle years my father kave me so*"

xiv. Run 6500 has loglikelihood of 13670.34.
Message is "*in my younker and more vulnerable years my father kave me so*"

xv. Run 7000 has loglikelihood of 13668.02.
Message is "*in my younker and more vulnerable years my father kave me so*"

xvi. Run 7500 has loglikelihood of 13743.44.
Message is "*in my younger and more vulnerable years my father gave me so*"

xvii. Run 8000 has loglikelihood of 13781.77.
Message is "*in my younger and more vulnerable years my father gave me so*"

xviii. Run 8500 has loglikelihood of 13781.77.
Message is "*in my younger and more vulnerable years my father gave me so*"

xix. Run 9000 has loglikelihood of 13777.94.
Message is "*in my younger and more vulnerable years my father gave me so*"

xx. Run 9500 has loglikelihood of 13777.94.
Message is "*in my younger and more vulnerable years my father gave me so*"

xxi. Run 10000 has loglikelihood of 13781.77.
Message is "*in my younger and more vulnerable years my father gave me so*"

(e) There are some $\psi(\alpha, \beta) = 0$ meaning to say that there are some state α we can never get to from β directly. This does not affect the ergodicity of the chain. The condition for aperiodicity states that $P(X_n = s|X_{n-1} = s) = p_{s->s} \forall s \in \chi$. In other words, the Markov Chain has to satisfy 2 conditions namely irreducibility and aperiodicity.

i. Irreducibility means that there is a path from each state to every other state thus, even if some $\psi(\alpha, \beta) = 0$, as long as there is no sub-path that points back to itself with probability of 1 we can still satisfy the condition.

ii. Aperiodicity means that a limit exists and the chain does not oscillates.

This can easily by observation of the the transition matrix used for the Hinton Plot. All elements in the transition matrix has at least one path in and one path out(that is not pointing to itself). Thus the transition matrix is irreducible.
There are only 2 symbols namely "]" and "=" that has a single path out from them that goes to "l" and " " respectively. "l" and " " have multiple paths leading into them and out of them. Thus, this transition matrix is aperiodic.
Hence I believe that this transition matrix is ergodic.
In any case, if it is not, we can easily rectify that by introducing a small weight instead of zeros into all the elements in the transition matrix and normalize it. This will create "artificial" paths that will help in satisfying the condition of ergodicity.

(f) i. Symbol probabilities alone will not be sufficient as the maximisation of the log likelihood will effectively be putting all the most frequent characters in the English language into one with the most counts in the encrypted text.

ii. Using a second order Markov chain will result in a huge increase in computation cost as we will need to hold the transition matrix in triplets rather than pairs. However, it should not result in the result being compromised. It could help tremendously in avoiding artificial maximum that our first order transition matrix has created where we see messages like ""en ty yainger and tare milnerahle" yielding a higher log-likelihood than ""in my younger and more vulnerable".

iii. This encryption scheme could, in theory, work if there were 2 symbols mapped to the same encrypted value. The loglikelihood will be calculated based on the weighted probability of those 2 symbols. This will require an introduction of another set of "emission" probabilities as well as complexities. It will be extremely challenging to figure out what are those emission probabilities without any evidence. Furthermore, it will be near impossible to properly decipher the message into a human-readable message in the same way that we have deciphered this message.

iv. In theory, this would work for Chinese with >10000 symbols. However, the transition matrix will be extremely sparse and thus, we will constantly run into situations where it will be near impossible to improve the log-likelihood. Thus, in practice, it will not be a feasible solution.

3. (a) Below is the Python code updated in the various section requested as well as the plots for the Gibbs Sampler

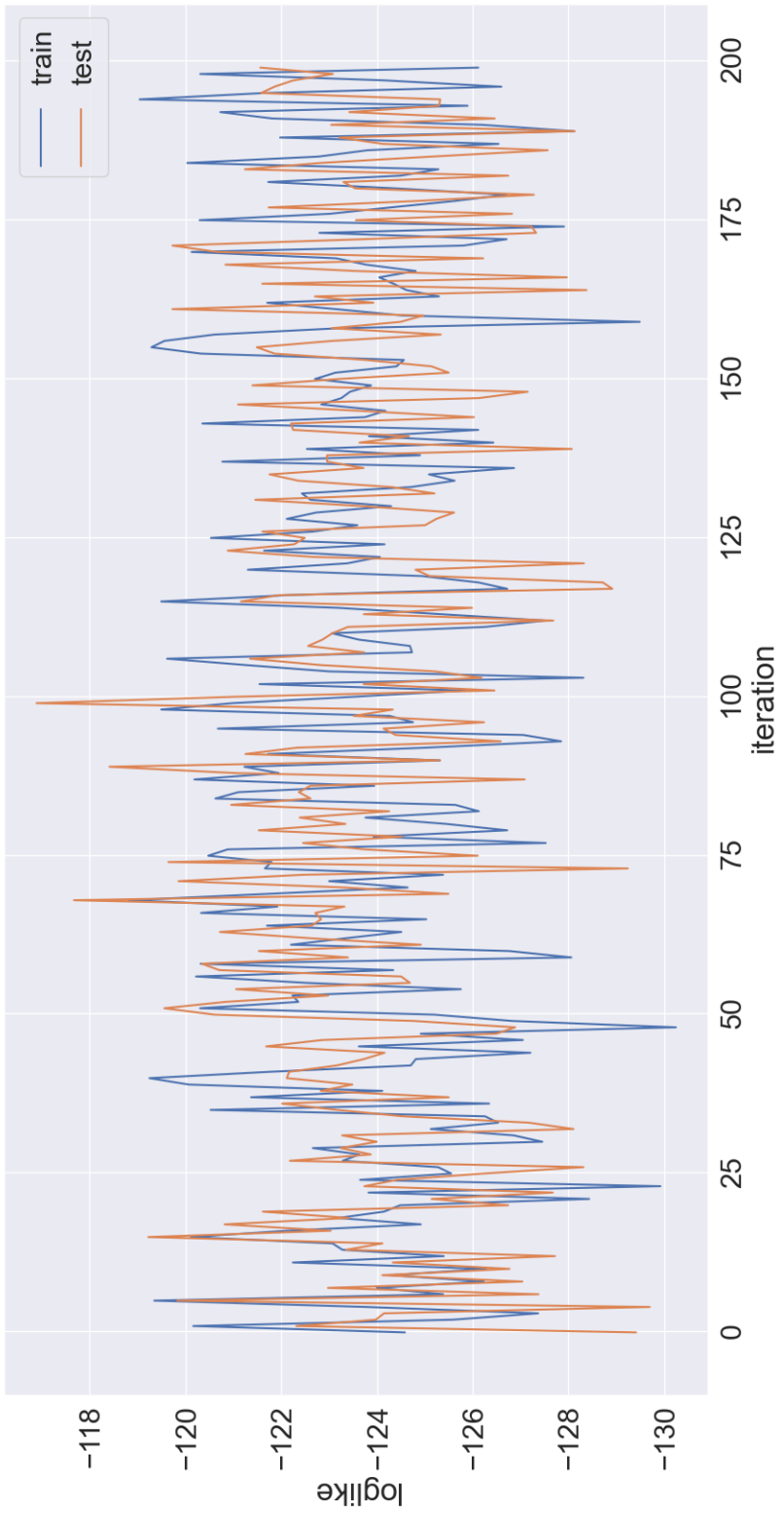


Figure 1: Gibbs Sampler

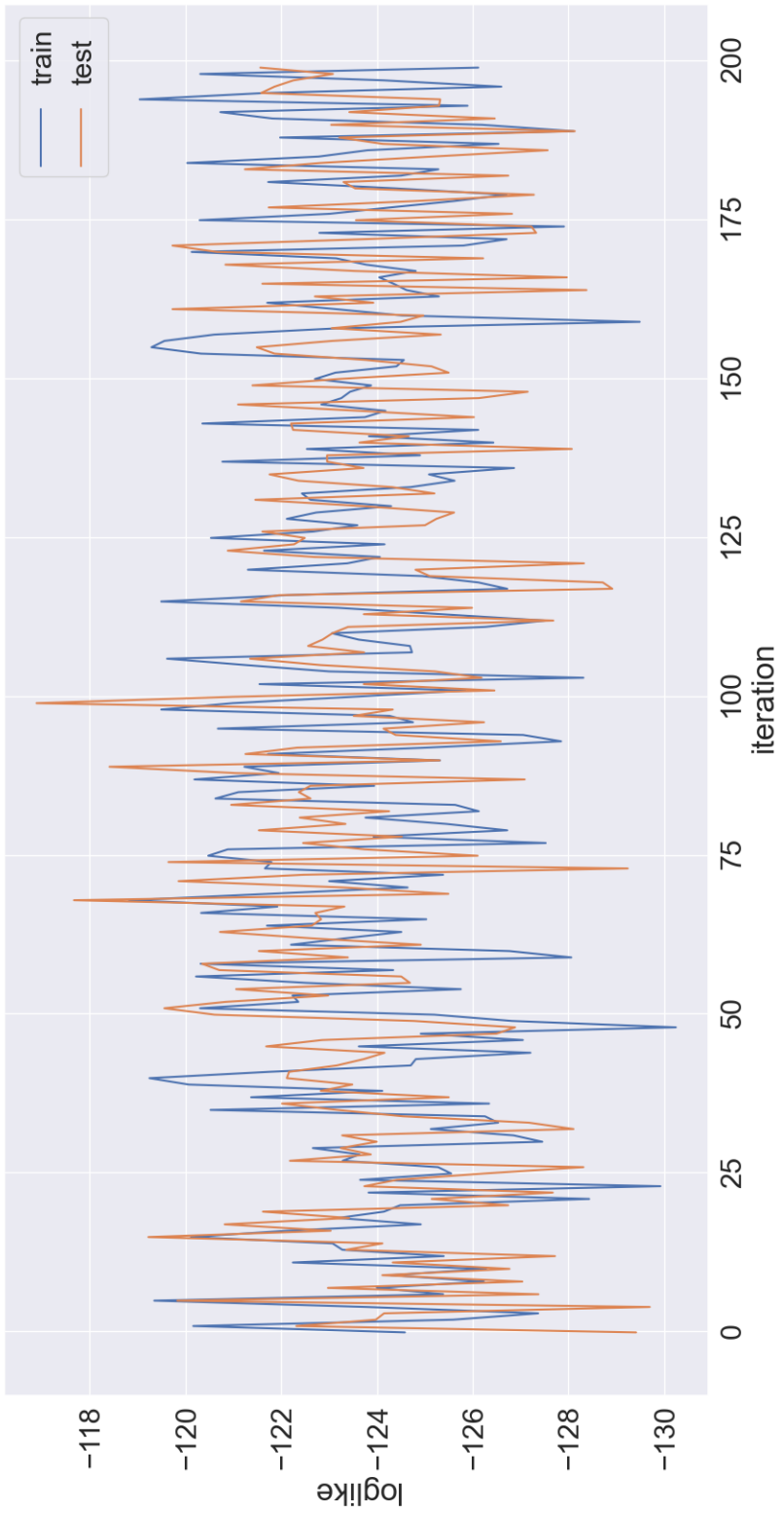


Figure 2: Collapsed Gibbs Sampler

```
def update_params(self):
    """
    Samples theta and phi, then computes the distribution of
    z_id and samples counts A_dk, B_kw from it
    """
    # todo: sample theta and phi
    # Using MAP
    # self.phi = np.einsum('kw,k->kw', (self.B_kw + self.beta - 1),
    #                      1 / (self.B_kw.sum(axis=1) + self.n_words * self.beta -
        self.n_words * 1))
    # self.theta = np.einsum('dk,d->dk', (self.A_dk + self.alpha - 1),
    #                        1 / (self.A_dk.sum(axis=1) + self.n_topics * self.alpha
      - self.n_topics * 1))
    for topic_ix, topic in enumerate(self.phi):
        self.phi[topic_ix] = np.random.dirichlet(self.B_kw[topic_ix, :] + self.beta)
    for doc_ix, doc in enumerate(self.theta):
        self.theta[doc_ix] = np.random.dirichlet(self.A_dk[doc_ix, :] + self.alpha)
    self.update_topic_doc_words()
    self.sample_counts()

def sample_counts(self):
    """
    For each document and each word, samples from z_id|x_id, theta, phi
    and adds the results to the counts A_dk and B_kw
    """
    self.A_dk.fill(0)
    self.B_kw.fill(0)

    if self.do_test:
        self.A_dk_test.fill(0)
        self.B_kw_test.fill(0)

    # todo: sample a topic for each (doc, word) and update A_dk, B_kw
        correspondingly
    for doc_ix, doc in enumerate(self.docs_words):
        for word_ix, word in enumerate(self.docs_words[doc_ix]):
            # select most likely topic for each word in each doc based on the
                maximum probability from the n_topics
            topic_ix = np.random.multinomial(1, self.topic_doc_words_distr[:, doc_ix
                , word_ix]).argmax()
            # print("Document {}'s {}th word is allocated to topic {}".format(doc_ix
                , word_ix, topic_ix))
            # Generate A_dk and B_kw based on self.topic_doc_words_distr
            self.A_dk[doc_ix][topic_ix] += 1
            self.B_kw[topic_ix][word_ix] += 1
    if self.do_test:
        for doc_ix, doc in enumerate(self.docs_words_test):
            for word_ix, word in enumerate(self.docs_words_test[doc_ix]):
                # select most likely topic for each word in each doc based on the
                    maximum probability from the n_topics
                topic_ix = np.random.multinomial(1, self.topic_doc_words_distr[:,
                    doc_ix, word_ix]).argmax()
                # print("Document {}'s {}th word is allocated to topic {}".format(
                    doc_ix, word_ix, topic_ix))
                # Generate A_dk and B_kw based on self.topic_doc_words_distr
                self.A_dk_test[doc_ix][topic_ix] += 1
                self.B_kw_test[topic_ix][word_ix] += 1

def update_loglike(self, iteration):
    """
    Updates loglike of the data, omitting the constant additive term
    with Gamma functions of hyperparameters
    """
    # todo: implement log-like
    # Hint: use scipy.special.gammaln (imported as gammaln) for log(gamma)
    lg_beta_func = lambda param: np.sum(gammaln(param)) - gammaln(np.sum(param))

    for topic in range(self.n_topics):
        self.loglike[iteration] += lg_beta_func(self.B_kw[topic] + self.beta)
        # self.loglike[iteration] -= lg_beta_func(self.beta)
```

```
62
63      for doc in range(self.n_docs):
64          self.loglike[iteration] += lg_beta_func(self.A_dk[doc] + self.alpha)
65          # self.loglike[iteration] -= lg_beta_func(self.alpha)
66
67      if self.do_test:
68          for topic in range(self.n_topics):
69              self.loglike_test[iteration] += lg_beta_func(self.B_kw_test[topic] +
                    self.beta)
70              self.loglike_test[iteration] -= lg_beta_func(self.beta)
71
72          for doc in range(self.n_docs):
73              self.loglike_test[iteration] += lg_beta_func(self.A_dk_test[doc] + self.
                    alpha)
74              self.loglike_test[iteration] -= lg_beta_func(self.alpha)
75      pass
```

(b) Based on the plot of the joint probabiliies produced by the toy example, I believe that we will need a few more thousand runs required for burn-in as we can clear see that the loglikeligood has yet to converge

(c) Based on the computed autocorrelations, I believe that the 2 Gibbs samplers will converge at similar rate. The normal Gibbs sampler picks up θ and ϕ based on the distribution draw whereas it is integrated out in the collapsed form (or can been seen as incorporated into the sampling of Z). This means they should be similar in nature in long run, however, due to the collapsed form taking a "analytic" approach to θ and ϕ, it could have a different rate.

(d) Varying α,β and K has the effect of changing the Dirichlet distribution. For example, if α is lower than 1, we will see maximums occuring at the corners of the Dirichlet distribution whereas if α is all the same and greater than 1, the maximum will be found in the centre of the Dirichlet distribution. Similarly, K will change the number of "vertices" in the Dirichlet distribution.
Effectively, this encodes the prior assumptions we have on the model and the sum of α and sum of β encodes how "big" the prior "datasets" are as seen by the distribution of $Dir(A_d + \alpha)$ and $Dir(B_k + \beta)$. The bigger the prior "datasets" (and thus more confident we are about our prior assumptions), will result in α and β further from 1 and can be interpreted as higher "weight" allocation compared to our current observations.
Another way to think about it is that low α value places more weight on having each document composed of only a few dominant topics (whereas a high value will return many more relatively dominant topics) and similarly for β on words distribution
Thus, changing them will have significant impact on the posterior and predictive performance of the model if the dataset is small but if we are provided with overwhelming data, they will eventually be negated. If our prior is accurate, it will help the model to converge faster.

(e) There are various items that are applicable in this aspect where we look to remove words from the vocabulary to make the problem tractable.

i. The one recommended by the question is tf-idf which is a scoring system that is commonly used such that we look at the frequency of a word occuring in a document vs the inverse frequency which I implemented as $log(\frac{k}{N}$ where k is the number of documents containing the words and N is the number of documents in total.

ii. However, prior to implementing this, I did a few more steps of pre-processing that helps facilitate the process.

A. Removal of stop words - This helps to remove the largest chunk of connecting words and this is useful in cleaning of documents.

B. Stemming of words - This helps to reduce words to its stem words. For example, "vectorized" and "vector" or "regression" and "regress". They imply keep ideas that are consistent in the topic space and thus by stemming them will help to make the objective of understand topics across words and documents more intepretable. If our aim is to understand how "active" or "passive" the language used is, "doing" vs "done" this step has to be skipped. For this step, I utilised the NLTK package and used the below:

```
from nltk.stem.porter import PorterStemmer
porter = PorterStemmer()
```

This has the additional benefit of making everything lowercase which avoids any inconsistency of treatment in Python

C. I note that I am able to successfully implement the tfidf vectorizer even without this 2 steps but from a document and word distribution on topic point of view, I believe that the pre-processing step helps to consolidate word counts without losing interpretability.