DD2434 - Machine Learning, Advanced Course
Assignment 1A

Tristan Perrot
tristanp@kth.se

Étienne Riguet
riguet@kth.se

November 2023

Contents

1 Exponential Family

1.1 Question 1.1

$$\begin{aligned}
p(x|\theta) &= h(x)\exp(\eta(\theta)\cdot T(x) - A(\eta)) \\
&= h(x)\exp(\eta(\lambda)\cdot T(x) - A(\eta(\lambda))) \\
&= h(x)\exp(\log\lambda\cdot x - A(\log\lambda)) \\
&= h(x)\exp(\log\lambda\cdot x - \lambda) \\
&= h(x)\exp(\log\lambda\cdot x)\exp(-\lambda) \\
&= e^{-\lambda}\frac{\lambda^x}{x!}
\end{aligned} \tag{1}$$

We can see that the distribution correspond to a <u>Poisson distribution of parameter λ</u>.

1.2 Question 1.2

$$\begin{aligned}
p(x|\theta) &= h(x)\exp(\eta(\theta)\cdot T(x) - A(\eta)) \\
&= \exp(\eta([\alpha,\beta])\cdot[\log x, x] - A(\alpha-1,-\beta)) \\
&= \exp([\alpha-1,-\beta]\cdot[\log x, x] - \log\Gamma(\alpha) + \alpha\log(\beta)) \\
&= \exp((\alpha-1)\log x - \beta x - \log\Gamma(\alpha) + \alpha\log(\beta)) \\
&= \frac{\beta^\alpha}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x}
\end{aligned} \tag{2}$$

We can see that the distribution correspond to a <u>Gamma distribution of parameters α and β</u>.

1.3 Question 1.3

$$\begin{aligned}
p(x|\theta) &= h(x)\exp(\eta(\theta)\cdot T(x) - A(\eta)) \\
&= \frac{\exp(\eta([\mu,\sigma^2])\cdot[x,x^2] - A(\eta([\mu,\sigma^2])))}{\sqrt{2\pi}} \\
&= \frac{\exp([\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}]\cdot[x,x^2] - A([\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}]))}{\sqrt{2\pi}} \\
&= \frac{\exp(\frac{\mu x}{\sigma^2} - \frac{x^2}{2\sigma^2} - \frac{\mu^2}{2\sigma^2} - \log\sigma)}{\sqrt{2\pi}} \\
&= \frac{\exp(-\frac{(x-\mu)^2}{2\sigma^2})}{\sigma\sqrt{2\pi}}
\end{aligned} \tag{3}$$

We can see that the distribution correspond to a <u>Normal distribution of parameters μ and σ^2</u>.

1.4 Question 1.4

$$\begin{aligned}
p(x|\theta) &= h(x)\exp(\eta(\theta)\cdot T(x) - A(\eta)) \\
&= 2\exp(\eta(\lambda)\cdot x - A(\eta(\lambda))) \\
&= 2\exp(-\lambda x - A(-\lambda)) \\
&= 2\exp\left(-\lambda x + \log\left(\frac{\lambda}{2}\right)\right) \\
&= \lambda e^{-\lambda x}
\end{aligned} \tag{4}$$

We can see that the distribution correspond to a Exponential distribution of parameter λ.

1.5 Question 1.5

$$\begin{aligned}
p(x|\theta) &= h(x)\exp(\eta(\theta)\cdot T(x) - A(\eta)) \\
&= \exp(\eta([\psi_1,\psi_2])\cdot[\log x, \log(1-x)] - A(\eta([\psi_1,\psi_2]))) \\
&= \exp([\psi_1 - 1, \psi_2 - 1]\cdot[\log x, \log(1-x)] - A([\psi_1 - 1, \psi_2 - 1])) \\
&= \exp((\psi_1 - 1)\log x + (\psi_2 - 1)\log(1-x) - \log\Gamma(\psi_1) - \log\Gamma(\psi_2) + \log\Gamma(\psi_1+\psi_2)) \\
&= \frac{\Gamma(\psi_1+\psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)} x^{\psi_1 - 1}(1-x)^{\psi_2 - 1}
\end{aligned} \tag{5}$$

We can see that the distribution correspond to a Beta distribution of parameters ψ_1 and ψ_2.

2 Dependencies in a Directed Graphical Model

2.1 Question 2.6

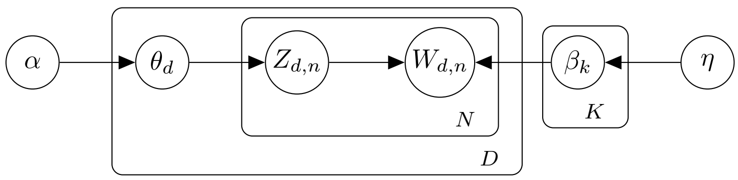
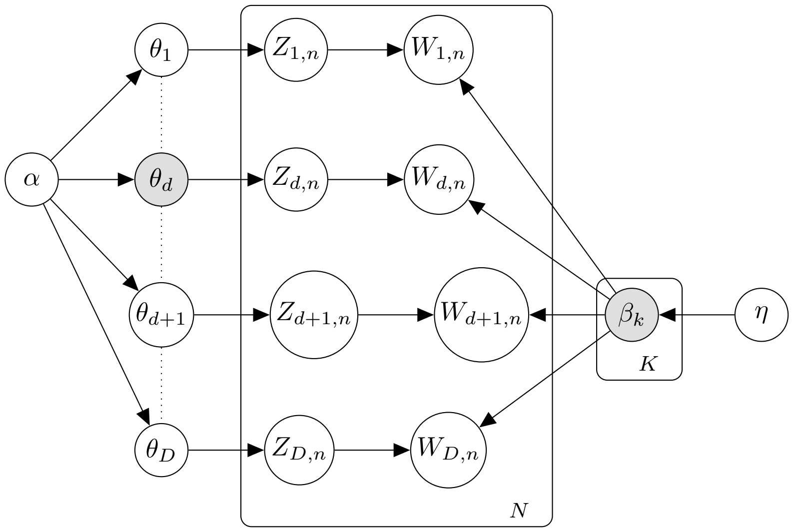


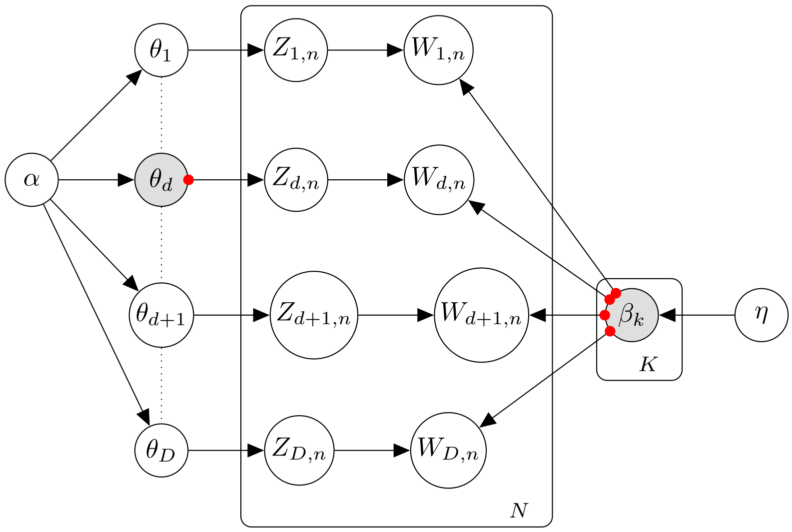
Figure 1: Graphical model of smooth LDA.

The Bayes net take this form :

Tristan Perrot
Étienne Riguet



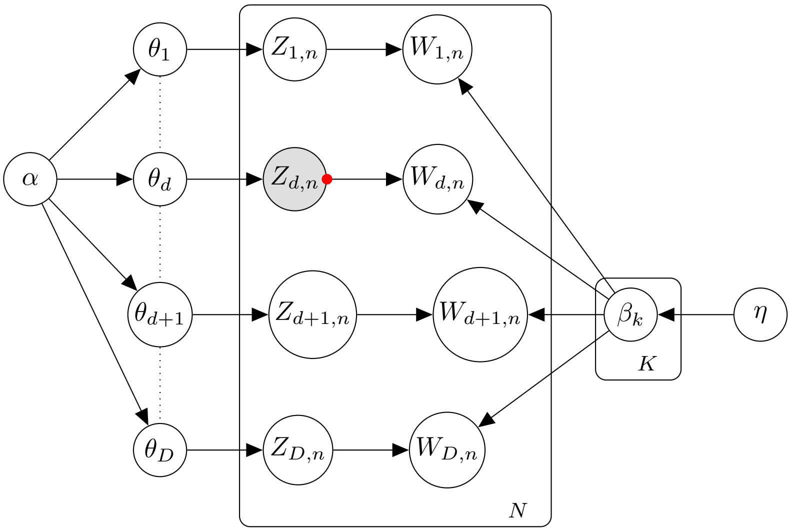
Then, if we use the method using the d-separation, we obtain this :



Therefore, we can see that $W_{d,n} \perp W_{d,n+1} | \theta_d, \beta_{1:K}$ is <u>true</u>.

2.2 Question 2.7

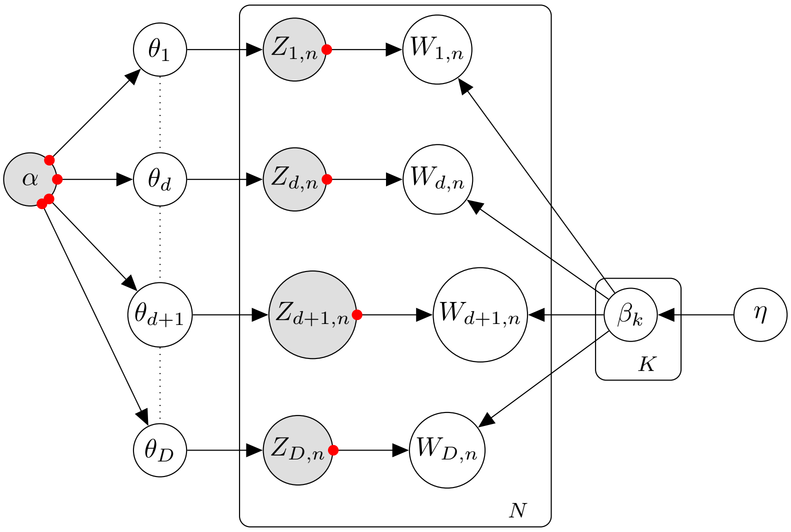
The Bayes net take this form (with d-separation marks) :



Therefore, we can see that $\theta_d \perp \theta_{d+1} | Z_{d,1:N}$ is <u>false</u>.

2.3 Question 2.8

The Bayes net take this form (with d-separation marks) :



Therefore, we can see that $\theta_d \perp \theta_{d+1} | \alpha, Z_{1:D,1:N}$ is <u>true</u>.

2.4 Question 2.9

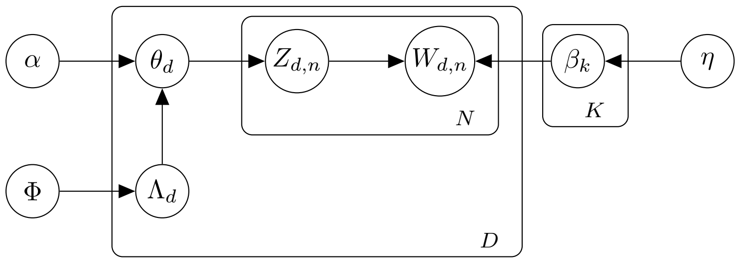
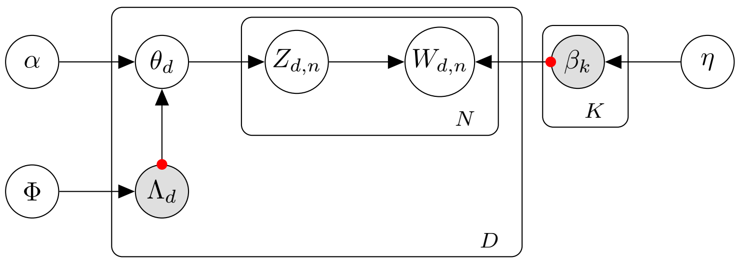


Figure 2: Graphical model of Labeled LDA.

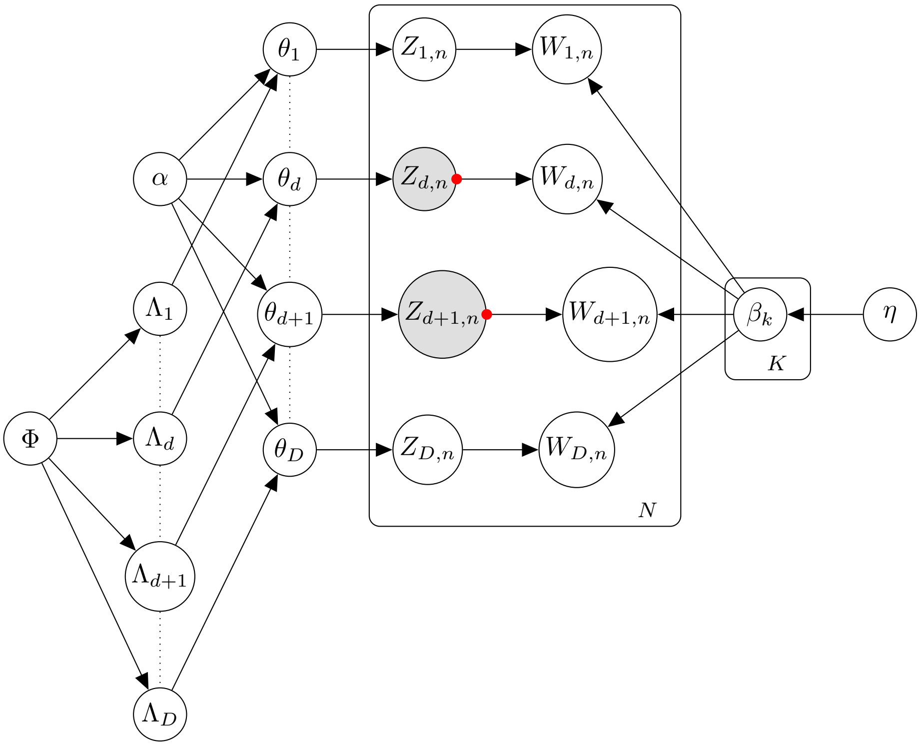
The Bayes net take this form (with d-separation marks) :



Therefore, we can see that $W_{d,n} \perp W_{d,n+1} | \Lambda_d, \beta_{1:K}$ is <u>false</u>.

2.5 Question 2.10

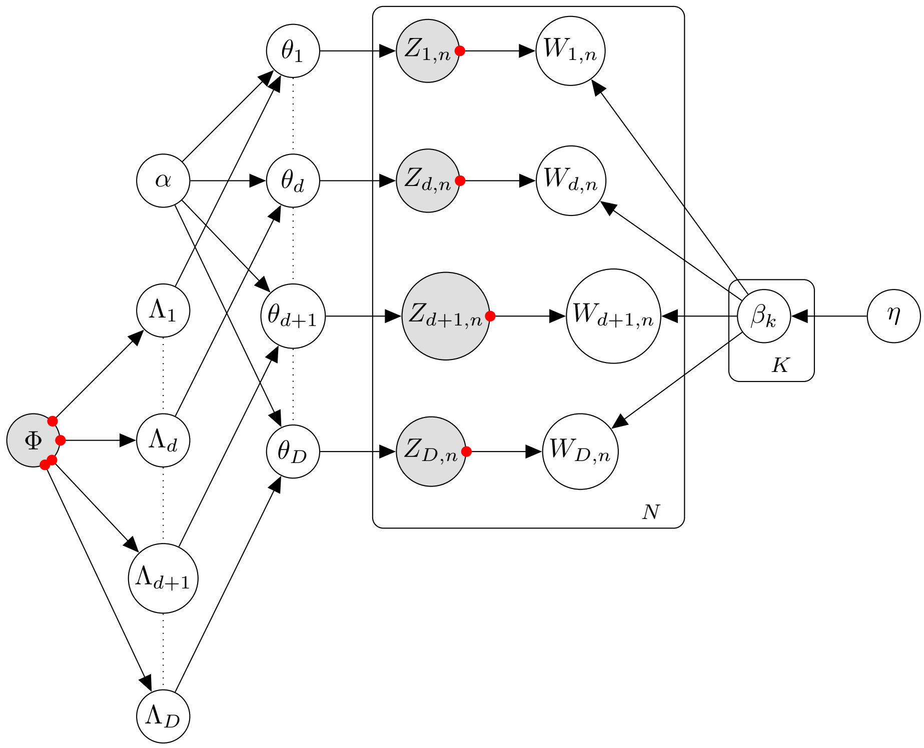
The Bayes net take this form (with d-separation marks) :



Therefore, we can see that $\theta_d \perp \theta_{d+1} | Z_{d,1:N}, Z_{d+1,1:N}$ is false.

2.6 Question 2.11

The Bayes net take this form (with d-separation marks) :



Therefore, we can see that $\Lambda_d \perp \Lambda_{d+1} | \Phi, Z_{1:D,1:N}$ is <u>false</u>.

3 CAVI

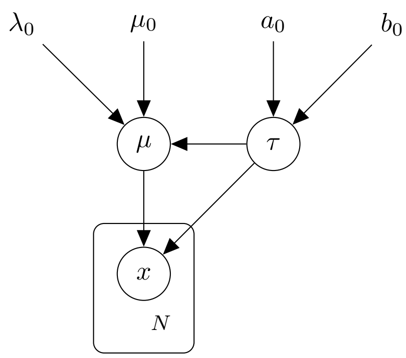


Figure 3: DGM

3.1 Question 3.12

In the bishop book, we can see that :

$$p(X|\mu,\tau) = \left(\frac{\tau}{2\pi}\right)^{N/2} \exp\left\{-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2\right\} \tag{6}$$

$$p(\mu|\tau) = \mathcal{N}(\mu|\mu_0,(\lambda_0\tau)^{-1}) \tag{7}$$

$$p(\tau) = \mathrm{Gam}(\tau|a_0,b_0) \tag{8}$$

Then, by using the code in appendix A.1, we obtain :

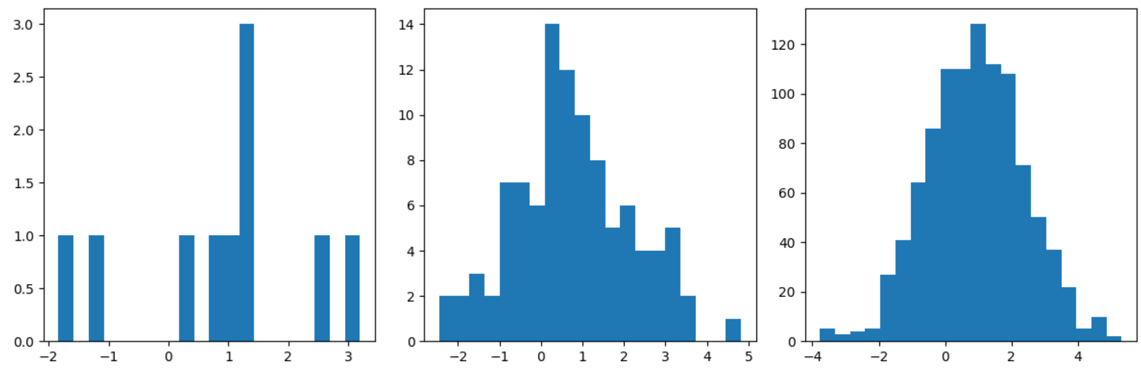


Figure 4: Generated Data

3.2 Question 3.13

Let's find the ML estimates of μ and τ. We know that $\log(q^*(\mu)) = \mathbb{E}_{-\mu}[\log p(X,\mu,\tau)]$. Then, we can write :

$$\begin{aligned}
\log(q^*(\mu)) &= \mathbb{E}_{-\mu}[\log p(X,\mu,\tau)] \\
&\stackrel{+}{=} \mathbb{E}_{\tau}[\log p(X|\mu,\tau) + \log p(\mu|\tau)] \\
&= \mathbb{E}_{\tau}\left[\frac{N}{2}\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2 + \frac{1}{2}\log\left(\frac{\lambda_0\tau}{2\pi}\right) - \frac{\lambda_0\tau}{2}(\mu-\mu_0)^2\right] \\
&\stackrel{+}{=} -\frac{\mathbb{E}_{\tau}[\tau]}{2}\left(\lambda_0(\mu-\mu_0)^2 + \sum_{n=1}^{N}(x_n-\mu)^2\right) \\
&= -\frac{\mathbb{E}_{\tau}[\tau]}{2}\left(\lambda_0\mu^2 - 2\lambda_0\mu\mu_0 + \lambda_0\mu_0^2 + \sum_{n=1}^{N}x_n^2 - 2\mu\sum_{n=1}^{N}x_n + N\mu^2\right) \\
&= -\frac{\mathbb{E}_{\tau}[\tau]}{2}\left((\lambda_0+N)\mu^2 - 2(\lambda_0\mu_0 + \sum_{n=1}^{N}x_n)\mu + \lambda_0\mu_0^2 + \sum_{n=1}^{N}x_n^2\right) \\
&\stackrel{+}{=} -\frac{\mathbb{E}_{\tau}[\tau](\lambda_0+N)}{2}\left(\mu^2 - 2\mu\frac{\lambda_0\mu_0 + \sum_{n=1}^{N}x_n}{\lambda_0+N}\right)
\end{aligned} \tag{9}$$

Therefore we can conclude that $q^*(\mu) = \mathcal{N}(\mu|\mu_N, \lambda_N^{-1})$ with :

$$\mu_N = \frac{\lambda_0\mu_0 + \sum_{n=1}^N x_n}{\lambda_0 + N} \tag{10}$$

$$\lambda_N = (\lambda_0 + N)\mathbb{E}[\tau] \tag{11}$$

And for τ we have :

$$\begin{aligned}
\log(q^*(\tau)) &= \mathbb{E}_{-\tau}[\log p(X, \mu, \tau)] \\
&\stackrel{\pm}{=} \mathbb{E}_\mu[\log p(X|\mu, \tau) + \log p(\mu|\tau)] + \log p(\tau) \\
&\stackrel{\pm}{=} (a_0 - 1)\log\tau - b_0\tau + \frac{N+1}{2}\log\tau - \frac{\tau}{2}\mathbb{E}_\mu\left[\sum_{n=1}^N (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right] \\
&= (a_0 + \frac{N+1}{2} - 1)\log\tau - \left(b_0 + \frac{1}{2}\mathbb{E}_\mu\left[\sum_{n=1}^N (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right]\right)\tau
\end{aligned} \tag{12}$$

Therefore we can conclude that $q^*(\tau) = \text{Gam}(\tau|a_N, b_N)$ with :

$$a_N = a_0 + \frac{N+1}{2} \tag{13}$$

$$\begin{aligned}
b_N &= b_0 + \frac{1}{2}\mathbb{E}_\mu\left[\sum_{n=1}^N (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right] \\
b_N &= b_0 + \frac{1}{2}\left(\sum_{n=1}^N x_n^2 + N\mathbb{E}_\mu[\mu^2] - 2\mathbb{E}_\mu[\mu]\sum_{n=1}^N x_n + \lambda_0\left(\mathbb{E}_\mu[\mu^2] + \mu_0^2 - 2\mu_0\mathbb{E}_\mu[\mu]\right)\right)
\end{aligned} \tag{14}$$

With :

$$\begin{aligned}
\mathbb{E}_{q(\mu)}[\mu] &= \mu_N \\
\mathbb{E}_{q(\mu)}[\mu^2] &= \frac{1}{\lambda_N} + \mu_N^2 \\
\mathbb{E}_{q(\tau)}[\tau] &= \frac{a_N}{b_N}
\end{aligned} \tag{15}$$

If we take non-informative priors then $a_0 = b_0 = \mu_0 = \lambda_0 = 0$, then we have :

$$\begin{aligned}
\mu_N &= \overline{x} \\
\lambda_N &= N\mathbb{E}[\tau] \\
a_N &= \frac{N+1}{2} \\
b_N &= \frac{1}{2}\mathbb{E}_\mu\left[\sum_{n=1}^N (x_n - \mu)^2\right]
\end{aligned} \tag{16}$$

And by using $\mathbb{E}[\tau] = \frac{a_N}{b_N}$ we obtain :

$$
\begin{aligned}
\frac{1}{\mathbb{E}[\tau]} &= \frac{b_N}{a_N} \\
\frac{1}{\mathbb{E}[\tau]} &= \frac{2}{2(N+1)} \mathbb{E}_\mu \left[\sum_{n=1}^{N} (x_n - \mu)^2 \right] \\
\frac{1}{\mathbb{E}[\tau]} &= \frac{N}{N+1} \left(\overline{x^2} - 2\overline{x}\mathbb{E}[\mu] + \mathbb{E}[\mu^2] \right)
\end{aligned}
\tag{17}
$$

And, with the fact that $\mathbb{E}[\mu] = \mu_N$ and $\mathbb{E}[\mu^2] = \frac{1}{\lambda_N} + \mu_N^2$, we obtain :

$$
\begin{aligned}
\mathbb{E}[\mu] &= \overline{x} \\
\mathbb{E}[\mu^2] &= \frac{1}{N\mathbb{E}[\tau]} + \overline{x}^2
\end{aligned}
\tag{18}
$$

And therefore:

$$
\begin{aligned}
\frac{1}{\mathbb{E}[\tau]} = \frac{N}{N+1} \left(\overline{x^2} - 2\overline{x}^2 + \frac{1}{N\mathbb{E}[\tau]} + \overline{x}^2 \right) &\Leftrightarrow \frac{1}{\mathbb{E}[\tau]} - \frac{1}{(N+1)\mathbb{E}[\tau]} = \frac{N}{N+1} \left(\overline{x^2} - \overline{x}^2 \right) \\
&\Leftrightarrow \frac{N+1-1}{(N+1)\mathbb{E}[\tau]} = \frac{N}{N+1} \left(\overline{x^2} - \overline{x}^2 \right) \\
&\Leftrightarrow \frac{1}{\mathbb{E}[\tau]} = \left(\overline{x^2} - \overline{x}^2 \right) \\
&\Leftrightarrow \frac{1}{\mathbb{E}[\tau]} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \overline{x})^2
\end{aligned}
\tag{19}
$$

Which define the ML estimates. The implementation is in the code in appendix A.1.

3.3 Question 3.14

The posterior is defined as $p(\mu, \tau | x)$. Then, we can write :

$$
\begin{aligned}
p(\mu, \tau | x) &= \frac{p(x|\mu, \tau) p(\mu, \tau)}{p(x)} \\
&\propto p(x|\mu, \tau) p(\mu, \tau)
\end{aligned}
\tag{20}
$$

Where $x|\mu, \tau \sim \mathcal{N}(\mu|\mu, \tau^{-1})$ and $\mu, \tau \sim NormalGamma(\mu_0, \lambda_0, a_0, b_0)$. Therefore, as we saw in the question 1.3 in the Module 1 exercise, we have $\mu, \tau | x \sim NormalGamma(\mu', \lambda', a', b')$, where :

$$
\begin{aligned}
\mu' &= \frac{N\overline{x} + \mu_0 \lambda_0}{N + \lambda_0} \\
\lambda' &= N + \lambda_0 \\
a' &= a_0 + \frac{N-1}{2} \\
b' &= b_0 + \frac{1}{2} \left(\sum_{n=1}^{N} x_n^2 + \lambda_0 \mu_0^2 - \frac{(N\overline{x} + \mu_0 \lambda_0)^2}{N + \lambda_0} \right)
\end{aligned}
\tag{21}
$$

Therefore, if we plot the contour for each datasets we obtain :

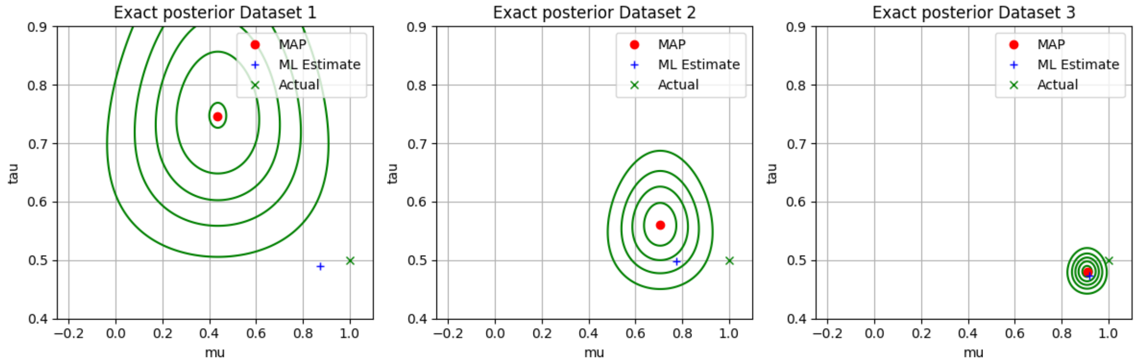


Figure 5: Contours of exact posteriors by datasets

The rest of the answer is in the code in appendix A.1.

3.4 Question 3.15

The equation (10.24) in the Bishop is the mean-field approximation which is :

$$q(\mu, \tau) = q(\mu)q(\tau) \tag{22}$$

This time, we take the result of the question 3.13 without setting the priors to 0. Then, we have :

$$\begin{aligned} q(\mu) &= \mathcal{N}(\mu|\mu_N, \lambda_N^{-1}) \\ q(\tau) &= \mathrm{Gam}(\tau|a_N, b_N) \end{aligned} \tag{23}$$

with updates equations in the cavi algorithm described by :

$$\begin{aligned} \mu_N &= \frac{\lambda_0\mu_0 + N\overline{x}}{\lambda_0 + N} \\ \lambda_N &= (\lambda_0 + N)\mathbb{E}[\tau] \\ a_N &= a_0 + \frac{N+1}{2} \\ b_N &= b_0 + \frac{1}{2}\left(\sum_{n=1}^{N} x_n^2 + N\mathbb{E}_\mu[\mu^2] - 2\mathbb{E}_\mu[\mu]\sum_{n=1}^{N} x_n + \lambda_0\left(\mathbb{E}_\mu[\mu^2] + \mu_0^2 - 2\mu_0\mathbb{E}_\mu[\mu]\right)\right) \end{aligned} \tag{24}$$

and the expectations are the ones described in the equation (15).
Now, we need to find the ELBO formula :

$$\begin{aligned} \mathcal{L}(q) &= \mathbb{E}_{q(\mu),q(\tau)}[\log p(X, \mu, \tau)] - \mathbb{E}_{q(\mu),q(\tau)}[\log q(\mu, \tau)] \\ &= \mathbb{E}_{q(\mu),q(\tau)}[\log p(X|\mu, \tau) + \log p(\mu, \tau)] - \mathbb{E}_{q(\mu),q(\tau)}[\log q(\mu) + \log q(\tau)] \\ &= \mathbb{E}_{q(\mu),q(\tau)}[\log p(X|\mu, \tau)] + \mathbb{E}_{q(\mu),q(\tau)}[\log p(\mu, \tau)] - \mathbb{E}_{q(\mu)}[\log q(\mu)] - \mathbb{E}_{q(\tau)}[\log q(\tau)] \\ &= \mathbb{E}_{q(\mu),q(\tau)}[\log p(X|\mu, \tau)] + \mathbb{E}_{q(\mu),q(\tau)}[\log p(\mu, \tau)] + \mathbb{H}_q[\mu] + \mathbb{H}_q[\tau] \end{aligned} \tag{25}$$

If we compute the first term we have:

$$\begin{aligned}
\mathbb{E}_{q(\mu),q(\tau)}[\log p(X|\mu,\tau)] &= \mathbb{E}_{q(\mu),q(\tau)}\left[\frac{N}{2}\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2\right] \\
&= \frac{N}{2}\left(\mathbb{E}_{q(\tau)}[\log\tau] - \log(2\pi)\right) \\
&\quad - \frac{\mathbb{E}_{q(\tau)}[\tau]}{2}\left(\sum_{n=1}^{N}x_n^2 - 2\mathbb{E}_{q(\mu)}[\mu]N\overline{x_n} + N\mathbb{E}_{q(\mu)}[\mu^2]\right) \\
&\overset{+}{=} \frac{N}{2}\mathbb{E}_{q(\tau)}[\log\tau] - \frac{\mathbb{E}_{q(\tau)}[\tau]}{2}\left(\sum_{n=1}^{N}x_n^2 - 2\mathbb{E}_{q(\mu)}[\mu]N\overline{x_n} + N\mathbb{E}_{q(\mu)}[\mu^2]\right)
\end{aligned} \tag{26}$$

And the second one is:

$$\begin{aligned}
\mathbb{E}_{q(\mu),q(\tau)}[\log p(\mu,\tau)] &= \mathbb{E}_{q(\mu),q(\tau)}\left[\log\left(\frac{b_0^{a_0}\sqrt{\lambda_0}}{\Gamma(a_0)\sqrt{2\pi}}\right) + (a_0-\frac{1}{2})\log\tau - b_0\tau - \frac{\lambda_0\tau(\mu-\mu_0)^2}{2}\right] \\
&= \log\left(\frac{b_0^{a_0}\sqrt{\lambda_0}}{\Gamma(a_0)\sqrt{2\pi}}\right) + (a_0-\frac{1}{2})\mathbb{E}_{q(\tau)}[\log\tau] - b_0\mathbb{E}_{q(\tau)}[\tau] \\
&\quad - \frac{\lambda_0\mathbb{E}_{q(\tau)}[\tau]}{2}\left(\mathbb{E}_{q(\mu)}[\mu^2] - 2\mu_0\mathbb{E}_{q(\mu)}[\mu] + \mu_0^2\right) \\
&\overset{+}{=} (a_0-\frac{1}{2})\mathbb{E}_{q(\tau)}[\log\tau] - b_0\mathbb{E}_{q(\tau)}[\tau] - \frac{\lambda_0\mathbb{E}_{q(\tau)}[\tau]}{2}\left(\mathbb{E}_{q(\mu)}[\mu^2] - 2\mu_0\mathbb{E}_{q(\mu)}[\mu] + \mu_0^2\right)
\end{aligned} \tag{27}$$

And we can compute all of that because entropies are known and we have the following expectations:

$$\begin{aligned}
\mathbb{E}_{q(\mu)}[\mu] &= \mu_N \\
\mathbb{E}_{q(\mu)}[\mu^2] &= \frac{1}{\lambda_N} + \mu_N^2 \\
\mathbb{E}_{q(\tau)}[\tau] &= \frac{a_N}{b_N} \\
\mathbb{E}_{q(\tau)}[\log\tau] &= \psi(a_N) - \log b_N
\end{aligned} \tag{28}$$

Then we obtain this result :

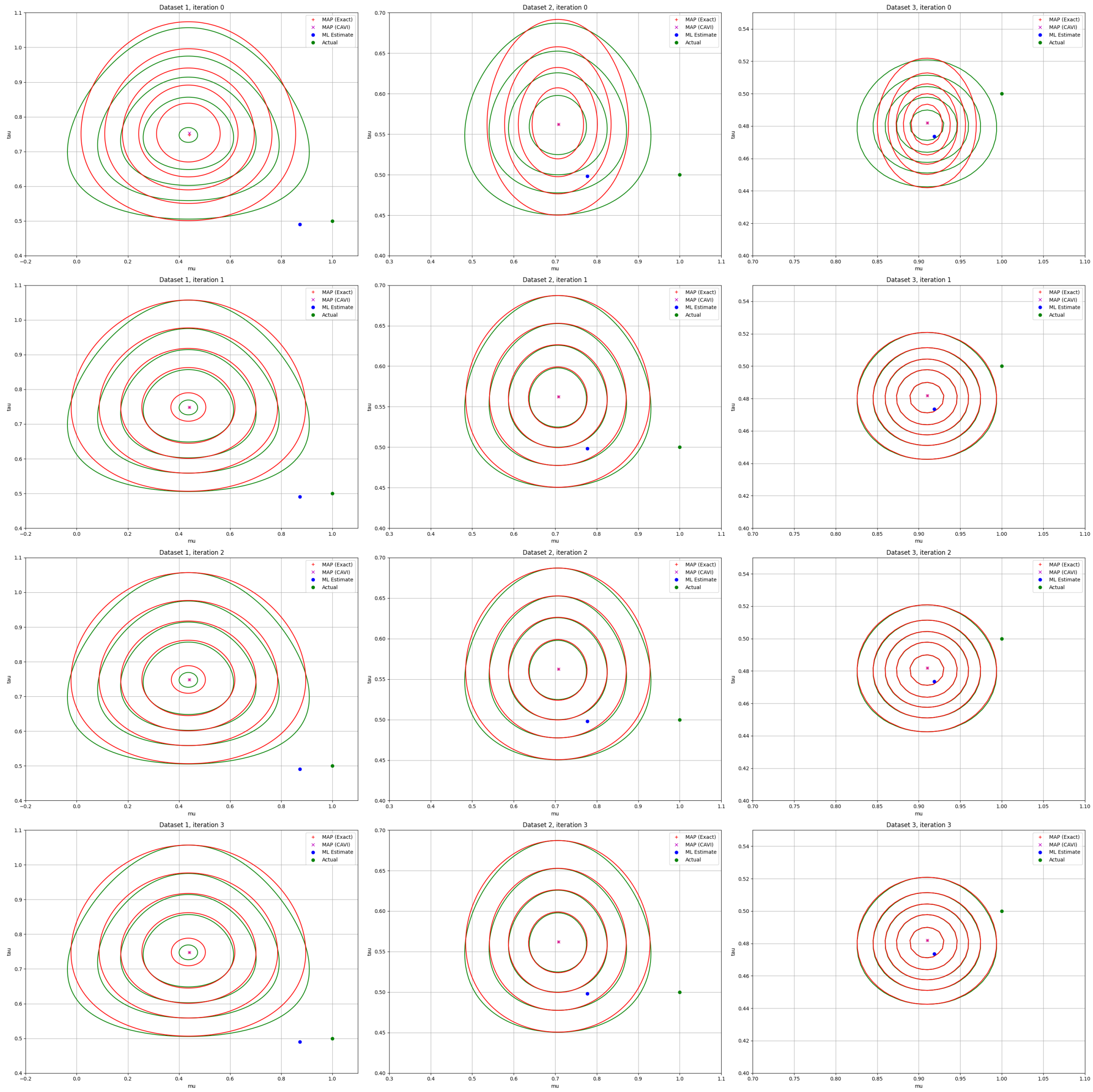


Figure 6: Contours of the approximations by VI and the exact posterior by datasets, by iterations

And we obtain an elbo plot :

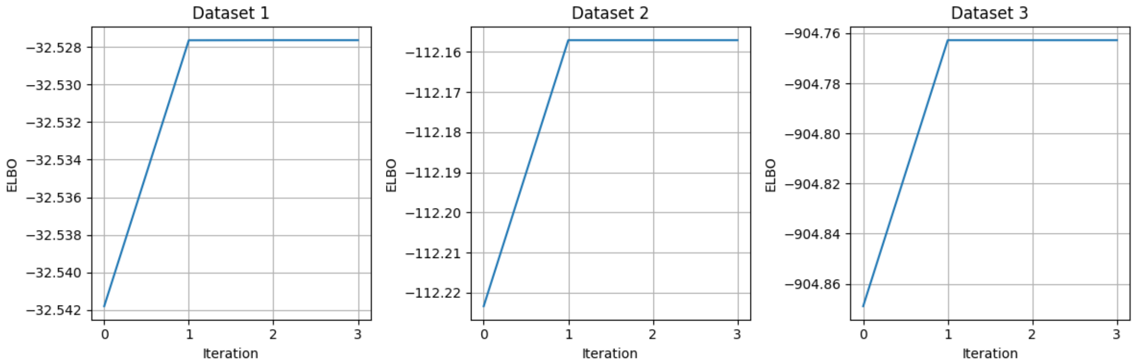


Figure 7: ELBO plot by datasets

The code is in appendix A.1.

4 SVI - LDA

4.1 Question 4.16

According to the Hoffman paper, the local hidden variables are defined by the model where the distribution of each observation x_n only depends on its corresponding local variable z_n and the global variables $\beta_{1:K}$. Therefore, we can write:

$$p(x, z, \beta | alpha) = p(\beta|\alpha) \prod_{n=1}^{N} p(x_n, z_n|\beta) \tag{29}$$

Because:

$$p(x_n, z_n | x_{-n}, z_{-n}, \beta, \alpha) = p(x_n, z_n|\beta, \alpha) \tag{30}$$

4.2 Question 4.17

In this figure the global hidden variables are the topics $\beta_{1:K}$ and the local hidden variables are the topic proportions θ_d and the topic assignments $z_{d,1:N}$.

4.3 Question 4.18

The ELBO formula is:

$$\mathcal{L}(q) = \mathbb{E}_{q(\theta,z,\beta)}[\log p(w, \theta, z, \beta)] - \mathbb{E}_{q(\theta,z,\beta)}[\log q(\theta, z, \beta)] \tag{31}$$

And here we recall that we have:

$$p(w, z, \theta, \beta) = \prod_{k=1}^{K} p(\beta_k) \prod_{d=1}^{D} p(\theta_d) \prod_{d=1}^{D} \prod_{n=1}^{N} p(z_{dn}|\theta_d)p(w_{dn}|\beta_{z_{dn}}) \tag{32}$$

Therefore we have:

$$\begin{aligned}
&\mathbb{E}_{q(\theta,z,\beta)}[\log p(w,\theta,z,\beta)] \\
&= \mathbb{E}_{q(\theta,z,\beta)}\left[\sum_{k=1}^{K}\log p(\beta_k) + \sum_{d=1}^{D}\log p(\theta_d) + \sum_{d=1}^{D}\sum_{n=1}^{N}\log p(z_{dn}|\theta_d)p(w_{dn}|\beta_{z_{dn}})\right] \\
&\qquad - \mathbb{E}_{q(\theta,z,\beta)}\left[\sum_{k=1}^{K}\log q(\beta_k) + \sum_{d=1}^{D}\log q(\theta_d) + \sum_{d=1}^{D}\sum_{n=1}^{N}\log q(z_{dn}|\theta_d) + \log q(w_{dn}|\beta_{z_{dn}})\right] \\
&= \sum_{k=1}^{K}\mathbb{E}_{q(\beta_k)}[\log p(\beta_k)] + \sum_{d=1}^{D}\mathbb{E}_{q(\theta_d)}[\log p(\theta_d)] + \sum_{d=1}^{D}\sum_{n=1}^{N}\mathbb{E}_{q(z_{dn}),q(\theta_d)}[\log p(z_{dn}|\theta_d)] \\
&\qquad + \sum_{d=1}^{D}\sum_{n=1}^{N}\mathbb{E}_{q(z_{dn}),q(\beta_{z_{dn}})}[\log p(w_{dn}|\beta_{z_{dn}})] \\
&\qquad + \sum_{k=1}^{K}\mathbb{H}_q[\beta_k] + \sum_{d=1}^{D}\mathbb{H}_q[\theta_d] + \sum_{d=1}^{D}\sum_{n=1}^{N}\mathbb{H}_q[z_{dn}|\theta_d] + \mathbb{H}_q[w_{dn}|\beta_{z_{dn}}]
\end{aligned} \tag{33}$$

Using Hoffman updates equations, we can write:

$$\begin{aligned}
\lambda_k &= \eta + \sum_{d=1}^{D}\sum_{n=1}^{N} z_{dn}^{k} w_{dn} \\
\gamma_d &= \alpha + \sum_{n=1}^{N} z_{dn} \\
\phi_{dn} &= \log\beta_{k,w_{dn}} + \log\theta_{dk}
\end{aligned} \tag{34}$$

And by using the expectations given in the Hoffman paper, we can write:

$$\begin{aligned}
\sum_{k=1}^{K}\mathbb{E}_{q(\beta_k)}[\log p(\beta_k)] &= \sum_{k=1}^{K}\sum_{\nu=1}^{W}(\eta-1)\mathbb{E}_{q(\beta_{k\nu})}[\log\beta_{k\nu}] \\
&= \sum_{k=1}^{K}\sum_{\nu=1}^{W}(\eta-1)\left(\Psi(\lambda_{k\nu}) - \Psi\left(\sum_{y=1}^{W}\lambda_{ky}\right)\right)
\end{aligned} \tag{35}$$

$$\begin{aligned}
\sum_{d=1}^{D}\mathbb{E}_{q(\theta_d)}[\log p(\theta_d)] &= \sum_{d=1}^{D}\sum_{k=1}^{K}(\alpha-1)\mathbb{E}_{q(\theta_{dk})}[\log\theta_{dk}] \\
&= \sum_{d=1}^{D}\sum_{k=1}^{K}(\alpha-1)\left(\Psi(\gamma_{dk}) - \Psi\left(\sum_{v=1}^{K}\gamma_{dv}\right)\right)
\end{aligned} \tag{36}$$

$$\begin{aligned}
\sum_{d=1}^{D}\sum_{n=1}^{N}\mathbb{E}_{q(z_{dn}),q(\theta_d)}[\log p(z_{dn}|\theta_d)] &= \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K}\mathbb{E}_{q(z_{dn})}[z_{dn}^{k}]\mathbb{E}_{q(\theta_{dk})}[\log\theta_{dk}] \\
&= \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K}\phi_{dn}^{k}\left(\Psi(\gamma_{dk}) - \Psi\left(\sum_{v=1}^{K}\gamma_{dv}\right)\right)
\end{aligned} \tag{37}$$

$$\begin{aligned}\sum_{d=1}^{D}\sum_{n=1}^{N}\mathbb{E}_{q(z_{dn}),q(\beta_{z_{dn}})}[\log p(w_{dn}|\beta_{z_{dn}})] &= \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{\nu=1}^{W} w_{dn}^{k}\mathbb{E}_{q(z_{dn})}[z_{dn}^{k}]\mathbb{E}_{q(\beta_{k\nu})}[\log\beta_{k\nu}] \\ &= \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{\nu=1}^{W} w_{dn}^{k}\phi_{dn}^{k}\left(\Psi(\lambda_{k\nu}) - \Psi\left(\sum_{y=1}^{W}\lambda_{ky}\right)\right)\end{aligned} \tag{38}$$

And the entropies can be found on Wikipedia.

4.4 Question 4.19

The code is in appendix A.2.

5 BBVI

5.1 Question 5.20

We have the simple model:

$$\begin{aligned} X|\theta &\sim \mathcal{N}(\theta, \sigma^2) \\ \theta &\sim Gamma(\alpha, \beta) \end{aligned} \tag{39}$$

With α, β and σ^2 known. Now, we will derive the gradient estimate w.r.t. ν without Rao-Blackwellization using one sample $z \sim q_\nu(\theta)$, $q_\nu(\theta) = LogNormal(\nu, \epsilon^2)$. We recall the formula:

$$\nabla_\lambda \mathcal{L} \approx \frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda \log q(z_s|\lambda)\left(\log p(x, z_s) - \log q(z_s|\lambda)\right) \tag{40}$$

Where $z_s \sim q(z|\lambda)$. Therefore, here we have:

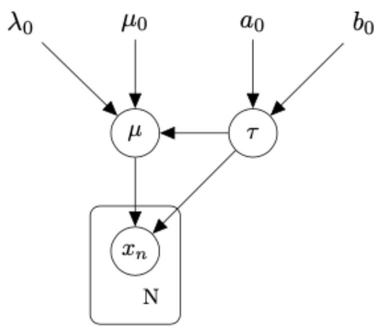
$$\begin{aligned}\nabla_\nu \mathcal{L} &\approx \nabla_\nu \log q(z|\nu)\left(\log p(x,z) - \log q(z|\nu)\right) \\ &\approx \nabla_\nu \log\left(\frac{\exp\left(-\frac{(\log\theta-\nu)^2}{2\epsilon^2}\right)}{\theta\epsilon\sqrt{2\pi}}\right)\left(\log\left(\frac{\exp\left(-\frac{(x-\theta)^2}{2\sigma^2}\right)}{\sigma\sqrt{2\pi}}\right)\right. \\ &\qquad \left. + \log\left(\frac{\beta^\alpha\theta^{\alpha-1}e^{-\beta\theta}}{\Gamma(\alpha)}\right) - \log\left(\frac{\exp\left(-\frac{(\log\theta-\nu)^2}{2\epsilon^2}\right)}{\theta\epsilon\sqrt{2\pi}}\right)\right) \\ &\approx \nabla_\nu\left(-\frac{(\log\theta-\nu)^2}{2\epsilon^2}\right)\left(-\frac{(x-\theta)^2}{2\sigma^2} - \log\left(\sigma\sqrt{2\pi}\right)\right. \\ &\qquad \left. -\beta\theta + (\alpha-1)\log\theta + \log\left(\frac{\beta^\alpha}{\Gamma(\alpha)}\right) + \frac{(\log\theta-\nu)^2}{2\epsilon^2} + \log\left(\theta\epsilon\sqrt{2\pi}\right)\right) \\ &\approx \frac{\log\theta-\nu}{\epsilon^2}\left(\frac{(\sigma(\log\theta-\nu))^2 - (\epsilon(x-\theta))^2}{2\sigma^2\epsilon^2} + \log\left(\frac{\epsilon}{\sigma}\right)\right. \\ &\qquad \left. -\beta\theta + \alpha\log\theta + \log\left(\frac{\beta^\alpha}{\Gamma(\alpha)}\right)\right)\end{aligned} \tag{41}$$

A Appendix

A.1 CAVI

Assignment 1.3 - CAVI

Consider the model defined by Equation (10.21)-(10-23) in Bishop, for which DGM is presented below:



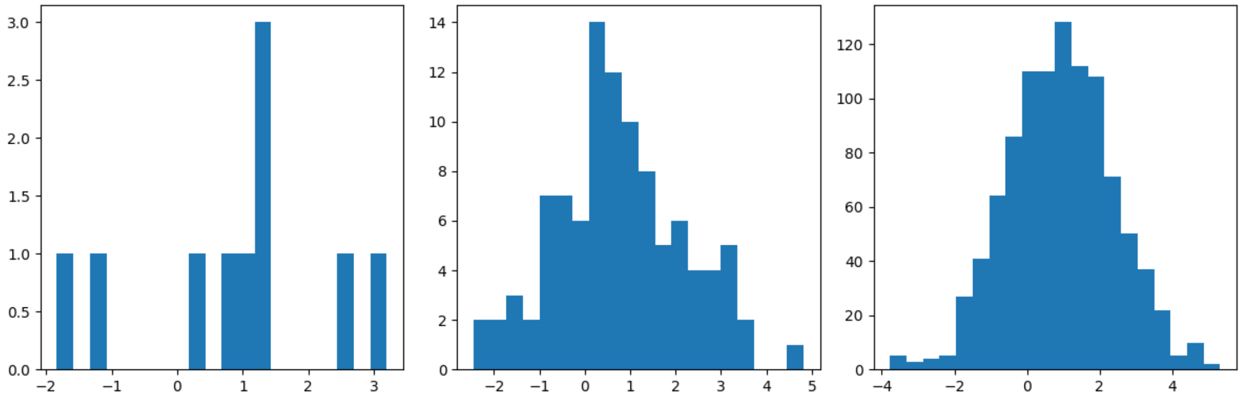
Question 1.3.12:

Implement a function that generates data points for the given model.

In [ ]:
```
import numpy as np
from scipy.stats import gamma, norm
from scipy.special import psi
from scipy.special import gamma as gamma_func
np.random.seed(14)

def generate_data(mu, tau, N):
  # Insert your code here
  D = np.random.normal(mu, np.sqrt(1/tau), N)

  return D
```

Set μ = 1, τ = 0.5 and generate datasets with size N=10,100,1000. Plot the histogram for each of 3 datasets you generated.

In [ ]:
```
MU = 1
TAU = 0.5

dataset_1 = generate_data(MU, TAU, 10)
dataset_2 = generate_data(MU, TAU, 100)
dataset_3 = generate_data(MU, TAU, 1000)

# Visulaize the datasets via histograms
# Insert your code here
import matplotlib.pyplot as plt
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].hist(dataset_1, bins=20)
axs[1].hist(dataset_2, bins=20)
axs[2].hist(dataset_3, bins=20)
plt.tight_layout()
plt.savefig('../images/12_data.png')
plt.show()
```



Question 1.3.13:

Find ML estimates of the variables μ and τ

In [ ]:
```
def ML_est(data):
    # insert your code
    N = len(data)
```

```python
    x_mean = np.mean(data)
    x_var = np.var(data)

    tau_ml = 1 / x_var
    mu_ml = x_mean

    return mu_ml, tau_ml
```

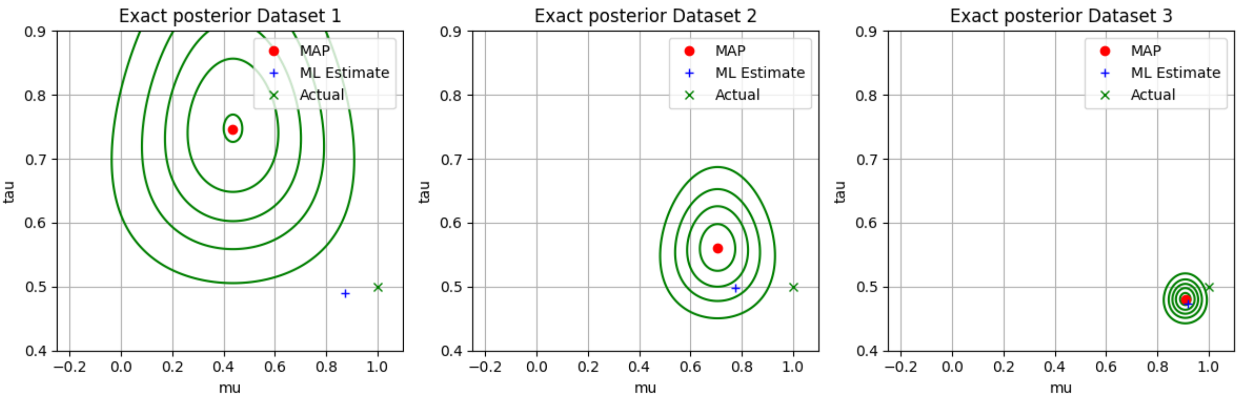
Question 1.3.14:

What is the exact posterior? First derive it in closed form, and then implement a function that computes it for the given parameters:

In [ ]:
```python
def compute_exact_posterior(D, a_0, b_0, mu_0, lambda_0):
    # your implementation
    x_mean = np.mean(D)
    N = len(D)

    mu_prime = (lambda_0 * mu_0 + N * x_mean) / (lambda_0 + N)
    lambda_prime = lambda_0 + N
    a_prime = a_0 + (N-1)/ 2
    b_prime = b_0 + 0.5 * (np.sum(D**2) +
                            lambda_0 * mu_0**2 - lambda_prime * mu_prime**2)

    exact_post_distribution = (a_prime, b_prime, mu_prime, lambda_prime)

    return exact_post_distribution
```

In [ ]:
```python
# prior parameters
mu_0 = 0
lambda_0 = 10
a_0 = 20
b_0 = 20
```

In [ ]:
```python
mus = np.linspace(-0.25, 1.1, 200)
taus = np.linspace(0.4, 0.9, 200)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))
for i, dataset in enumerate([dataset_1, dataset_2, dataset_3]):
  mu_ml, tau_ml = ML_est(dataset)

  a_T, b_T, mu_T, lambda_T = compute_exact_posterior(
      dataset, a_0, b_0, mu_0, lambda_0)

  Z_exact = np.zeros((len(mus), len(taus)))
  pTau = gamma(a=a_T, loc=0, scale=1/b_T)
  for j, tau in enumerate(taus):
      pMu = norm(loc=mu_T, scale=1/np.sqrt(lambda_T*tau))
      Z_exact[:, j] = pMu.pdf(mus) * pTau.pdf(tau)
  # Finding the maximum of the exact posterior
  mu_max_exact = mus[np.argmax(np.max(Z_exact, axis=1))]
  tau_max_exact = taus[np.argmax(np.max(Z_exact, axis=0))]
  # Plotting the results
  axs[i].contour(*np.meshgrid(mus, taus), Z_exact.T,
                 levels=5, colors=['green'])
  axs[i].plot(mu_max_exact, tau_max_exact, 'ro', label='MAP')
  axs[i].plot(mu_ml, tau_ml, 'b+', label='ML Estimate')
  axs[i].plot(MU, TAU, 'gx', label='Actual')
  axs[i].legend()
  axs[i].grid()
  axs[i].set_xlabel('mu')
  axs[i].set_ylabel('tau')
  axs[i].set_title('Exact posterior Dataset {}'.format(i+1))
plt.tight_layout()
plt.savefig('../images/14_contours.png')
plt.show()
```



Question 1.3.15:

You will implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop. Start with introducing the prior parameters:

Continue with a helper function that computes ELBO:

In [ ]:
```python
def compute_E_tau(a_N, b_N):
    E_tau = a_N / b_N

    return E_tau

def compute_E_mu_2(mu_N, lambda_N):
    E_mu_2 = mu_N**2 + 1/lambda_N

    return E_mu_2

def compute_E_log_tau(a_N, b_N):
    E_log_tau = psi(a_N) - np.log(b_N)

    return E_log_tau
```

In [ ]:
```python
def compute_elbo(D, a_0, b_0, mu_0, lambda_0, a_N, b_N, mu_N, lambda_N):
    N = len(D)
    x_mean = np.mean(D)
    x_2_sum = np.sum(D**2)
    E_tau = compute_E_tau(a_N, b_N)
    E_mu_2 = compute_E_mu_2(mu_N, lambda_N)
    E_log_tau = compute_E_log_tau(a_N, b_N)

    # compute the elbo
    # E[log p(D|mu, tau)]
    E_log_p_D = N/2 * E_log_tau - 0.5*E_tau * (x_2_sum - 2*N*x_mean*mu_N + N*E_mu_2)

    # E[log p(mu, tau)]
    E_log_p_mu_tau = (a_0-0.5)*E_log_tau - b_0*E_tau - 0.5*lambda_0*E_tau*(E_mu_2 + mu_0**2 - 2*mu_0*mu_N)

    # Entropy of mu
    entropy_mu = norm.entropy(loc=mu_N, scale=1/np.sqrt(lambda_N))
    # Entropy of tau
    entropy_tau = gamma.entropy(a=a_N, scale=1/b_N)

    elbo = E_log_p_D + E_log_p_mu_tau + entropy_mu + entropy_tau

    return elbo
```

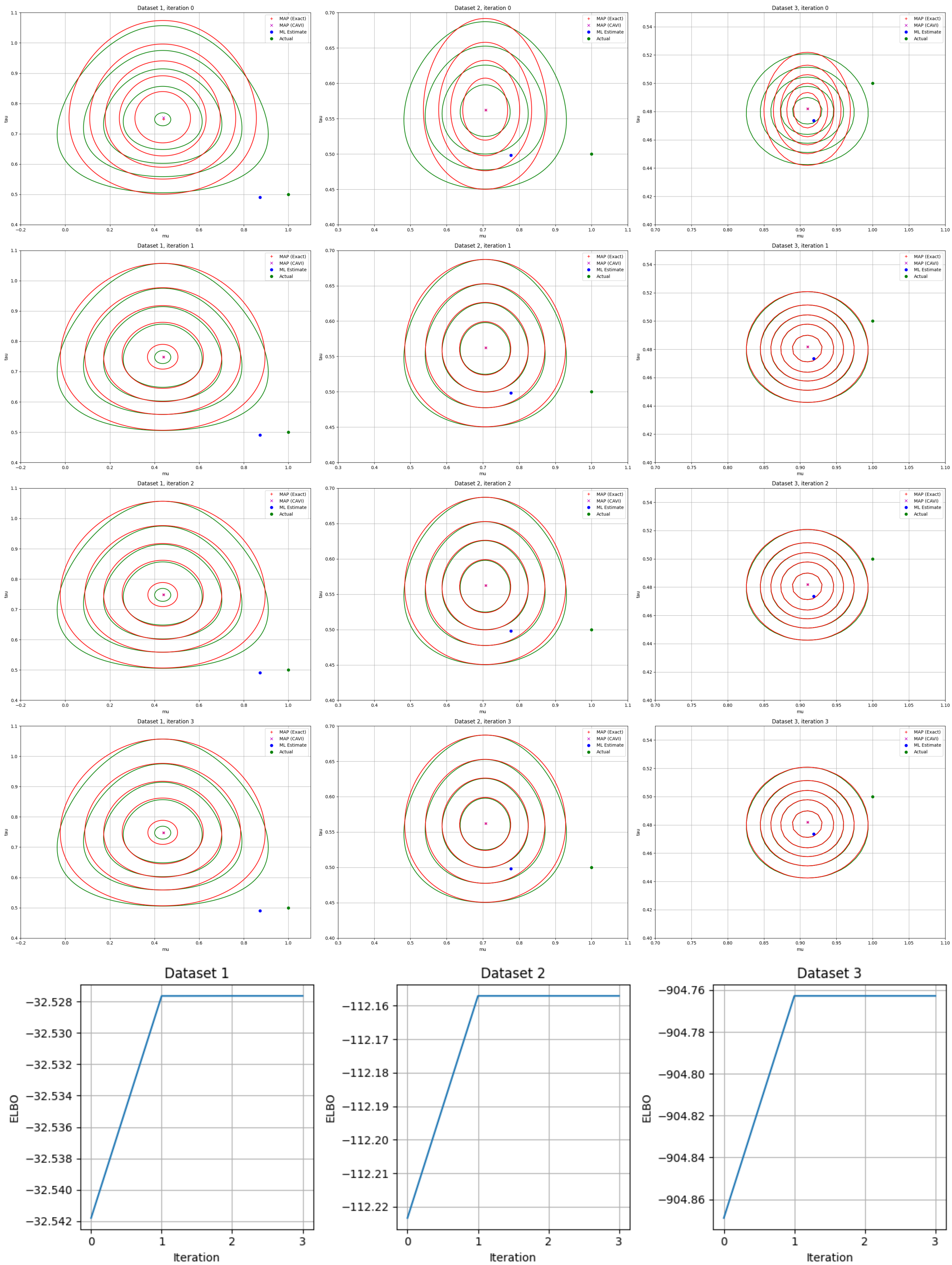
Now, implement the CAVI algorithm:

In [ ]:
```python
def CAVI(D, a_0, b_0, mu_0, lambda_0, iter=5):
  # make an initial guess for the expected value of tau
  E_tau = 1

  N = len(D)
  x_mean = np.mean(D)
  x_2_sum = np.sum(D**2)

  # Constants
  a_N = a_0 + (N+1) / 2
  mu_N = (lambda_0 * mu_0 + N * x_mean) / (lambda_0 + N)
  E_mu = mu_N

  # Variables
  b_Ns = []
  lambda_Ns = []

  # ELBO
  elbos = []

  # CAVI iterations ...
  for i in range(iter):
    # update the values for the variational parameters
    lambda_N = (lambda_0 + N) * E_tau

    E_mu_2 = compute_E_mu_2(mu_N, lambda_N)
    b_N = b_0 + 0.5 * (x_2_sum + N*E_mu_2 - 2*N*E_mu*x_mean + lambda_0*(E_mu_2 - 2*E_mu*mu_0 + mu_0**2))

    E_tau = compute_E_tau(a_N, b_N)

    b_Ns.append(b_N)
    lambda_Ns.append(lambda_N)
    # save ELBO for each iteration, plot them afterwards to show convergence
    elbos.append(compute_elbo(D, a_0, b_0, mu_0, lambda_0, a_N , b_N, mu_N, lambda_N))

  return a_N, b_N, mu_N, lambda_N, elbos, b_Ns, lambda_Ns
```

Run the VI algorithm on the datasets. Compare the inferred variational distribution with the exact posterior and the ML estimate. Visualize the results and discuss your findings.

In [ ]:
```
def compute_z_exact(mus, taus, a_, b_, mu_, lambda_):
    z = np.zeros((len(mus), len(taus)))
    pTau = gamma(a=a_, loc=0, scale=1/b_)
    for j, tau in enumerate(taus):
        pMu = norm(loc=mu_, scale=1/np.sqrt(lambda_*tau))
        z[:, j] = pMu.pdf(mus) * pTau.pdf(tau)

    return z

def compute_z_cavi(mus, taus, a_, b_, mu_, lambda_):
    pTau = gamma(a=a_, loc=0, scale=1/b_)
    pMu = norm(loc=mu_, scale=1/np.sqrt(lambda_))
    z = np.outer(pMu.pdf(mus), pTau.pdf(taus))
    return z
```

In [ ]:
```
iter = 4 # number of iterations for CAVI
mus = np.linspace(-0.2, 1.1, 200)
taus = np.linspace(0.1, 1.1, 200)

xlims = [[-0.2, 1.1], [0.3, 1.1], [0.7, 1.1]]
ylims = [[0.4, 1.1], [0.4, 0.7], [0.4, 0.55]]

elbos_list = []

fig, axs = plt.subplots(iter, 3, figsize=(30, 30))
for i, dataset in enumerate([dataset_1, dataset_2, dataset_3]):
  mu_ml, tau_ml = ML_est(dataset)
  a_N, b_N, mu_N, lambda_N, elbos, b_Ns, lambda_Ns = CAVI(dataset, a_0, b_0, mu_0, lambda_0, iter=iter)
  a_T, b_T, mu_T, lambda_T = compute_exact_posterior(
      dataset, a_0, b_0, mu_0, lambda_0)

  elbos_list.append(elbos)

  for j in range(iter):
    Z_exact = compute_z_exact(mus, taus, a_T, b_T, mu_T, lambda_T)
    Z_cavi = compute_z_cavi(mus, taus, a_N, b_Ns[j], mu_N, lambda_Ns[j])
    # Finding the maximum of the exact posterior
    mu_max_exact = mus[np.argmax(np.max(Z_exact, axis=1))]
    tau_max_exact = taus[np.argmax(np.max(Z_exact, axis=0))]
    # Finding the maximum of the CAVI approximation
    mu_max_cavi = mus[np.argmax(np.max(Z_cavi, axis=1))]
    tau_max_cavi = taus[np.argmax(np.max(Z_cavi, axis=0))]
    # Plotting the results
    axs[j, i].contour(*np.meshgrid(mus, taus), Z_exact.T,
                  levels=5, colors=['green'])
    axs[j, i].contour(*np.meshgrid(mus, taus), Z_cavi.T,
                    levels=5, colors=['red'])
    axs[j, i].plot(mu_max_exact, tau_max_exact, 'r+', label='MAP (Exact)')
    axs[j, i].plot(mu_max_cavi, tau_max_cavi, 'mx', label='MAP (CAVI)')
    axs[j, i].plot(mu_ml, tau_ml, 'bo', label='ML Estimate')
    axs[j, i].plot(MU, TAU, 'go', label='Actual')
    axs[j, i].legend()
    axs[j, i].grid()
    axs[j, i].set_xlabel('mu')
    axs[j, i].set_ylabel('tau')
    axs[j, i].set_title(f'Dataset {i+1}, iteration {j}')
    axs[j, i].set_xlim(xlims[i])
    axs[j, i].set_ylim(ylims[i])
plt.tight_layout()
plt.savefig('../images/15_contours.png')
plt.show()

# Plot ELBOs
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
for i in range(3):
  axs[i].plot(elbos_list[i])
  axs[i].set_xlabel('Iteration')
  axs[i].set_ylabel('ELBO')
  axs[i].set_title(f'Dataset {i+1}')
  axs[i].grid()
plt.tight_layout()
plt.savefig('../images/15_elbo.png')
plt.show()
```


Dataset 1, iteration 0
Dataset 2, iteration 0
Dataset 3, iteration 0
Dataset 1, iteration 1
Dataset 2, iteration 1
Dataset 3, iteration 1
Dataset 1, iteration 2
Dataset 2, iteration 2
Dataset 3, iteration 2
Dataset 1, iteration 3
Dataset 2, iteration 3
Dataset 3, iteration 3
MAP (Exact)
MAP (CAVI)
ML Estimate
Actual
mu
tau
Dataset 1
Dataset 2
Dataset 3
ELBO
Iteration

A.2 SVI

In [ ]:
```
import time

import numpy
import matplotlib.pyplot as plt
import numpy as np
import scipy.special as sp_spec
import scipy.stats as sp_stats
```

Assignment 1A. Problem 1.4.19 SVI.

Generate data

The cell below generates data for the LDA model. Note, for simplicity, we are using N_d = N for all d.

In [ ]:
```
def generate_data(D, N, K, W, eta, alpha):
    # sample K topics
    beta = sp_stats.dirichlet(eta).rvs(size=K)  # size K x W

    theta = np.zeros((D, K))   # size D x K

    w = np.zeros((D, N, W))
    z = np.zeros((D, N), dtype=int)
    for d in range(D):
        # sample document topic distribution
        theta_d = sp_stats.dirichlet(alpha).rvs(size=1)
        theta[d] = theta_d
        for n in range(N):
            # sample word to topic assignment
            z_nd = sp_stats.multinomial(n=1, p=theta[d, :]).rvs(size=1).argmax(axis=1)[0]

            # sample word
            w_nd = sp_stats.multinomial(n=1, p=beta[z_nd, :]).rvs(1)

            z[d, n] = z_nd
            w[d, n] = w_nd

    return w, z, theta, beta

D_sim = 500
N_sim = 50
K_sim = 2
W_sim = 5

eta_sim = np.ones(W_sim)
eta_sim[3] = 0.0001  # Expect word 3 to not appear in data
eta_sim[1] = 3.  # Expect word 1 to be most common in data
alpha_sim = np.ones(K_sim) * 1.0
w0, z0, theta0, beta0 = generate_data(D_sim, N_sim, K_sim, W_sim, eta_sim, alpha_sim)
w_cat = w0.argmax(axis=-1)  # remove one hot encoding
unique_z, counts_z = numpy.unique(z0[0, :], return_counts=True)
unique_w, counts_w = numpy.unique(w_cat[0, :], return_counts=True)

# Sanity checks for data generation
print(f"Average z of each document should be close to theta of document. \n Theta of doc 0: {theta0[0]} \n Mean z of doc 0: {counts_z/N_sim}
print(f"Beta of topic 0: {beta0[0]}")
print(f"Beta of topic 1: {beta0[1]}")
print(f"Word to topic assignment, z, of document 0: {z0[0, 0:10]}")
print(f"Observed words, w, of document 0: {w_cat[0, 0:10]}")
print(f"Unique words and count of document 0: {[f'{u}: {c}' for u, c in zip(unique_w, counts_w)]}")
```

```
Average z of each document should be close to theta of document.
 Theta of doc 0: [0.39546146 0.60453854]
 Mean z of doc 0: [0.32 0.68]
Beta of topic 0: [0.24091975 0.49660657 0.00123537 0.         0.26123832]
Beta of topic 1: [0.02307817 0.53687642 0.11790575 0.         0.32213965]
Word to topic assignment, z, of document 0: [0 1 1 1 1 0 1 1 1 0]
Observed words, w, of document 0: [1 1 2 4 2 4 1 2 1 1]
Unique words and count of document 0: ['0: 2', '1: 30', '2: 5', '4: 13']
```

In [ ]:
```
import torch
import torch.distributions as t_dist

def generate_data_torch(D, N, K, W, eta, alpha):
    """
    Torch implementation for generating data using the LDA model. Needed for sampling larger datasets.
    """
    # sample K topics
    beta_dist = t_dist.Dirichlet(torch.from_numpy(eta))
    beta = beta_dist.sample([K])  # size K x W

    # sample document topic distribution
    theta_dist = t_dist.Dirichlet(torch.from_numpy(alpha))
    theta = theta_dist.sample([D])

    # sample word to topic assignment
    z_dist = t_dist.OneHotCategorical(probs=theta)
    z = z_dist.sample([N]).reshape(D, N, K)

    # sample word from selected topics
    beta_select = torch.einsum("kw, dnk -> dnw", beta, z)
    w_dist = t_dist.OneHotCategorical(probs=beta_select)
    w = w_dist.sample([1])
```

```
    w = w.reshape(D, N, W)

    return w.numpy(), z.numpy(), theta.numpy(), beta.numpy()
```

Helper functions

In [ ]:
```
def log_multivariate_beta_function(a, axis=None):
    return np.sum(sp_spec.gammaln(a)) - sp_spec.gammaln(np.sum(a, axis=axis))
```

CAVI Implementation, ELBO and initialization

In [ ]:
```
def initialize_q(w, D, N, K, W):
    """
    Random initialization.
    """
    phi_init = np.random.random(size=(D, N, K))
    phi_init = phi_init / np.sum(phi_init, axis=-1, keepdims=True)
    gamma_init = np.random.randint(1, 10, size=(D, K))
    lmbda_init = np.random.randint(1, 10, size=(K, W))
    return phi_init, gamma_init, lmbda_init

def update_q_Z(w, gamma, lmbda):
    D, N, W = w.shape
    K, W = lmbda.shape
    E_log_theta = sp_spec.digamma(gamma) - sp_spec.digamma(np.sum(gamma, axis=1, keepdims=True))  # size D x K
    E_log_beta = sp_spec.digamma(lmbda) - sp_spec.digamma(np.sum(lmbda, axis=1, keepdims=True))   # size K x W
    log_rho = np.zeros((D, N, K))
    w_label = w.argmax(axis=-1)
    for d in range(D):
        for n in range(N):
            E_log_beta_wdn = E_log_beta[:, int(w_label[d, n])]
            E_log_theta_d = E_log_theta[d]
            log_rho_n = E_log_theta_d + E_log_beta_wdn
            log_rho[d, n, :] = log_rho_n

    phi = np.exp(log_rho - sp_spec.logsumexp(log_rho, axis=-1, keepdims=True))
    return phi

def update_q_theta(phi, alpha):
    E_Z = phi
    D, N, K = phi.shape
    gamma = np.zeros((D, K))
    for d in range(D):
        E_Z_d = E_Z[d]
        gamma[d] = alpha + np.sum(E_Z_d, axis=0)  # sum over N
    return gamma

def update_q_beta(w, phi, eta):
    E_Z = phi
    D, N, W = w.shape
    K = phi.shape[-1]
    lmbda = np.zeros((K, W))
    for k in range(K):
        lmbda[k, :] = eta
        for d in range(D):
            for n in range(N):
                lmbda[k, :] += E_Z[d,n,k] * w[d,n]  # Sum over d and n
    return lmbda

def calculate_elbo(w, phi, gamma, lmbda, eta, alpha):
    D, N, K = phi.shape
    W = eta.shape[0]
    E_log_theta = sp_spec.digamma(gamma) - sp_spec.digamma(np.sum(gamma, axis=1, keepdims=True))  # size D x K
    E_log_beta = sp_spec.digamma(lmbda) - sp_spec.digamma(np.sum(lmbda, axis=1, keepdims=True))  # size K x W
    E_Z = phi  # size D, N, K
    log_Beta_alpha = log_multivariate_beta_function(alpha)
    log_Beta_eta = log_multivariate_beta_function(eta)
    log_Beta_gamma = np.array([log_multivariate_beta_function(gamma[d, :]) for d in range(D)])
    dg_gamma = sp_spec.digamma(gamma)
    log_Beta_lmbda = np.array([log_multivariate_beta_function(lmbda[k, :]) for k in range(K)])
    dg_lmbda = sp_spec.digamma(lmbda)

    neg_CE_likelihood = np.einsum("dnk, kw, dnw", E_Z, E_log_beta, w)
    neg_CE_Z = np.einsum("dnk, dk -> ", E_Z, E_log_theta)
    neg_CE_theta = -D * log_Beta_alpha + np.einsum("k, dk ->", alpha - 1, E_log_theta)
    neg_CE_beta = -K * log_Beta_eta + np.einsum("w, kw ->", eta - 1, E_log_beta)
    H_Z = -np.einsum("dnk, dnk ->", E_Z, np.log(E_Z))
    gamma_0 = np.sum(gamma, axis=1)
    dg_gamma0 = sp_spec.digamma(gamma_0)
    H_theta = np.sum(log_Beta_gamma + (gamma_0 - K) * dg_gamma0 - np.einsum("dk, dk -> d", gamma - 1, dg_gamma))
    lmbda_0 = np.sum(lmbda, axis=1)
    dg_lmbda0 = sp_spec.digamma(lmbda_0)
    H_beta = np.sum(log_Beta_lmbda + (lmbda_0 - W) * dg_lmbda0 - np.einsum("kw, kw -> k", lmbda - 1, dg_lmbda))
    return neg_CE_likelihood + neg_CE_Z + neg_CE_theta + neg_CE_beta + H_Z + H_theta + H_beta

def CAVI_algorithm(w, K, n_iter, eta, alpha):
  D, N, W = w.shape
  phi, gamma, lmbda = initialize_q(w, D, N, K, W)

  # Store output per iteration
  elbo = np.zeros(n_iter)
  phi_out = np.zeros((n_iter, D, N, K))
  gamma_out = np.zeros((n_iter, D, K))
  lmbda_out = np.zeros((n_iter, K, W))
```

```python
    for i in range(0, n_iter):

      ###### CAVI updates #######

      # q(Z) update
      phi = update_q_Z(w, gamma, lmbda)

      # q(theta) update
      gamma = update_q_theta(phi, alpha)

      # q(beta) update
      lmbda = update_q_beta(w, phi, eta)

      # ELBO
      elbo[i] = calculate_elbo(w, phi, gamma, lmbda, eta, alpha)

      # outputs
      phi_out[i] = phi
      gamma_out[i] = gamma
      lmbda_out[i] = lmbda

    return phi_out, gamma_out, lmbda_out, elbo

n_iter0 = 100
K0 = K_sim
W0 = W_sim
eta_prior0 = np.ones(W0)
alpha_prior0 = np.ones(K0)
phi_out0, gamma_out0, lmbda_out0, elbo0 = CAVI_algorithm(w0, K0, n_iter0, eta_prior0, alpha_prior0)
final_phi0 = phi_out0[-1]
final_gamma0 = gamma_out0[-1]
final_lmbda0 = lmbda_out0[-1]
```

In [ ]:

```python
precision = 3
print(f"----- Recall label switching - compare E[theta] and true theta and check for label switching -----")
print(f"Final E[theta] of doc 0 CAVI:  {np.round(final_gamma0[0] / np.sum(final_gamma0[0], axis=0, keepdims=True), precision)}")
print(f"True theta of doc 0:          {np.round(theta0[0], precision)}")

print(f"----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----")
print(f"Final E[beta] k=0: {np.round(final_lmbda0[0, :] / np.sum(final_lmbda0[0, :], axis=-1, keepdims=True), precision)}")
print(f"Final E[beta] k=1: {np.round(final_lmbda0[1, :] / np.sum(final_lmbda0[1, :], axis=-1, keepdims=True), precision)}")
print(f"True beta k=0: {np.round(beta0[0, :], precision)}")
print(f"True beta k=1: {np.round(beta0[1, :], precision)}")
```

```
----- Recall label switching - compare E[theta] and true theta and check for label switching -----
Final E[theta] of doc 0 CAVI:  [0.807 0.193]
True theta of doc 0:          [0.395 0.605]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
Final E[beta] k=0: [0.    0.556 0.12  0.    0.324]
Final E[beta] k=1: [0.253 0.482 0.    0.    0.264]
True beta k=0: [0.241 0.497 0.001 0.    0.261]
True beta k=1: [0.023 0.537 0.118 0.    0.322]
```

SVI Implementation

Using the CAVI updates as a template, finish the code below.

In [ ]:

```python
def update_q_Z_svi(batch, w, gamma, lmbda):
    """
    TODO: rewrite to SVI update
    """
    D, N, W = w.shape
    K, W = lmbda.shape
    S = batch.shape[0]
    E_log_theta = sp_spec.digamma(
        gamma) - sp_spec.digamma(np.sum(gamma, axis=1, keepdims=True))  # size D x K
    E_log_beta = sp_spec.digamma(
        lmbda) - sp_spec.digamma(np.sum(lmbda, axis=1, keepdims=True))   # size K x W
    log_rho = np.zeros((S, N, K))
    w_label = w.argmax(axis=-1)
    for i, d in enumerate(batch):
        for n in range(N):
            E_log_beta_wdn = E_log_beta[:, int(w_label[d, n])]
            E_log_theta_d = E_log_theta[d]
            log_rho_n = E_log_theta_d + E_log_beta_wdn
            log_rho[i, n, :] = log_rho_n

    phi = np.exp(log_rho - sp_spec.logsumexp(log_rho, axis=-1, keepdims=True))
    return phi


def update_q_theta_svi(batch, phi, alpha):
    """
    TODO: rewrite to SVI update
    """
    E_Z_batch = phi[batch, :, :]
    D, N, K = phi.shape
    S = batch.shape[0]
    gamma = np.zeros((S, K))
    for i, d in enumerate(batch):
        E_Z_d = E_Z_batch[i]
        gamma[i] = alpha + np.sum(E_Z_d, axis=0)   # sum over N
    return gamma
```

```python
def update_q_beta_svi(batch, w, phi, eta):
    """
    TODO: rewrite to SVI update
    """
    E_Z = phi[batch, :, :]
    D, N, W = w.shape
    K = phi.shape[-1]
    S = batch.shape[0]
    lmbda = np.zeros((K, W))
    for k in range(K):
        lmbda[k, :] = eta
        for i, d in enumerate(batch):
            for n in range(N):
                lmbda[k, :] += E_Z[i, n, k] * w[d, n]  # Sum over d and n
    return lmbda


def SVI_algorithm(w, K, S, n_iter, eta, alpha):
    """
    Add SVI Specific code here.
    """
    D, N, W = w.shape
    phi, gamma, lmbda = initialize_q(w, D, N, K, W)

    # Store output per iteration
    elbo = np.zeros(n_iter)
    phi_out = np.zeros((n_iter, D, N, K))
    gamma_out = np.zeros((n_iter, D, K))
    lmbda_out = np.zeros((n_iter, K, W))

    delay = int(n_iter/10)
    if delay < 1:
        delay = 1

    forgetting_rate = 0.6
    def rho(t): return (t + delay)**(-forgetting_rate)

    for i in range(0, n_iter):
        # Sample batch and set step size, rho.
        batch = np.random.randint(0, D, size=S)
        rho_t = rho(i)
        gamma[batch, :] = 1.0

        bool = True
        count = 0
        ###### SVI updates #######

        while bool:
            phi_batch_prev = phi[batch, :, :]
            gamma_batch_prev = gamma[batch, :]
            phi[batch, :, :] = update_q_Z_svi(batch, w, gamma, lmbda)
            gamma[batch, :] = update_q_theta_svi(batch, phi, alpha)
            if (np.sum(np.abs(phi_batch_prev - phi[batch, :, :])) < 0.1*S and np.sum(np.abs(gamma_batch_prev - gamma[batch, :])) < 0.1*S) \
                                                                                                                            or count > 20:
                bool = False
            count += 1

        lmbda_batch = update_q_beta_svi(batch, w, phi, eta)
        lmbda = (1 - rho_t) * lmbda_batch + rho_t / \
            S * np.sum(lmbda_batch, axis=0)

        # ELBO
        elbo[i] = calculate_elbo(w, phi, gamma, lmbda, eta, alpha)

        # outputs
        phi_out[i] = phi
        gamma_out[i] = gamma
        lmbda_out[i] = lmbda

    return phi_out, gamma_out, lmbda_out, elbo
```

CASE 1

Tiny dataset

In [ ]:
```python
np.random.seed(0)

# Data simulation parameters
D1 = 50
N1 = 50
K1 = 2
W1 = 5
eta_sim1 = np.ones(W1)
alpha_sim1 = np.ones(K1)

w1, z1, theta1, beta1 = generate_data(D1, N1, K1, W1, eta_sim1, alpha_sim1)

# Inference parameters
n_iter_cavi1 = 100
n_iter_svi1 = 100
eta_prior1 = np.ones(W1) * 1.
alpha_prior1 = np.ones(K1) * 1.
S1 = 5 # batch size

start_cavi1 = time.time()
```

```
phi_out1_cavi, gamma_out1_cavi, lmbda_out1_cavi, elbo1_cavi = CAVI_algorithm(w1, K1, n_iter_cavi1, eta_prior1, alpha_prior1)
end_cavi1 = time.time()

start_svi1 = time.time()
phi_out1_svi, gamma_out1_svi, lmbda_out1_svi, elbo1_svi = SVI_algorithm(w1, K1, S1, n_iter_svi1, eta_prior1, alpha_prior1)
end_svi1 = time.time()

final_phi1_cavi = phi_out1_cavi[-1]
final_gamma1_cavi = gamma_out1_cavi[-1]
final_lmbda1_cavi = lmbda_out1_cavi[-1]
final_phi1_svi = phi_out1_svi[-1]
final_gamma1_svi = gamma_out1_svi[-1]
final_lmbda1_svi = lmbda_out1_svi[-1]
```

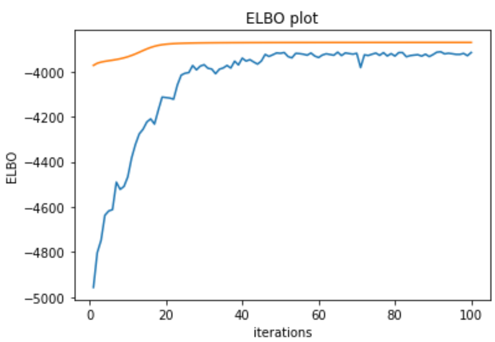
Evaluation

Do not expect perfect results in terms expectations being identical to the "true" theta and beta. Do not expect the ELBO plot of your SVI alg to be the same as the CAVI alg. However, it should increase and be in the same ball park as that of the CAVI alg.

In [ ]:
```
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
print(f"----- Recall label switching - compare E[theta] and true theta and check for label switching -----")
print(f"E[theta] of doc 0 SVI:  {final_gamma1_svi[0] / np.sum(final_gamma1_svi[0], axis=0, keepdims=True)}")
print(f"E[theta] of doc 0 CAVI: {final_gamma1_cavi[0] / np.sum(final_gamma1_cavi[0], axis=0, keepdims=True)}")
print(f"True theta of doc 0:    {theta1[0]}")

print(f"----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----")
print(f"E[beta] SVI k=0:    {final_lmbda1_svi[0, :] / np.sum(final_lmbda1_svi[0, :], axis=-1, keepdims=True)}")
print(f"E[beta] SVI k=1:    {final_lmbda1_svi[1, :] / np.sum(final_lmbda1_svi[1, :], axis=-1, keepdims=True)}")
print(f"E[beta] CAVI k=0:   {final_lmbda1_cavi[0, :] / np.sum(final_lmbda1_cavi[0, :], axis=-1, keepdims=True)}")
print(f"E[beta] CAVI k=1:   {final_lmbda1_cavi[1, :] / np.sum(final_lmbda1_cavi[1, :], axis=-1, keepdims=True)}")
print(f"True beta k=0:      {beta1[0, :]}")
print(f"True beta k=1:      {beta1[1, :]}")
```

```
----- Recall label switching - compare E[theta] and true theta and check for label switching -----
E[theta] of doc 0 SVI:  [0.529 0.471]
E[theta] of doc 0 CAVI: [0.475 0.525]
True theta of doc 0:    [0.676 0.324]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
E[beta] SVI k=0:    [0.117 0.076 0.282 0.451 0.073]
E[beta] SVI k=1:    [0.255 0.282 0.152 0.166 0.144]
E[beta] CAVI k=0:   [0.276 0.347 0.129 0.095 0.154]
E[beta] CAVI k=1:   [0.075 0.011 0.351 0.503 0.059]
True beta k=0:      [0.185 0.291 0.214 0.183 0.128]
True beta k=1:      [0.136 0.075 0.291 0.434 0.063]
```

In [ ]:
```
plt.plot(list(range(1, n_iter_cavi1 + 1)), elbo1_svi[np.arange(0, n_iter_svi1, int(n_iter_svi1 / n_iter_cavi1))])
plt.plot(list(range(1, n_iter_cavi1 + 1)), elbo1_cavi)
plt.title("ELBO plot")
plt.xlabel("iterations")
plt.ylabel("ELBO")
plt.show()
```



In [ ]:
```
# Add your own code for evaluation here (will not be graded)
```

CASE 2

Small dataset

In [ ]:
```
np.random.seed(0)

# Data simulation parameters
D2 = 1000
N2 = 50
K2 = 3
W2 = 10
eta_sim2 = np.ones(W2)
alpha_sim2 = np.ones(K2)

w2, z2, theta2, beta2 = generate_data(D2, N2, K2, W2, eta_sim2, alpha_sim2)

# Inference parameters
n_iter_cavi2 = 100
n_iter_svi2 = 100
eta_prior2 = np.ones(W2) * 1.
alpha_prior2 = np.ones(K2) * 1.
S2 = 100 # batch size
```

```
start_cavi2 = time.time()
phi_out2_cavi, gamma_out2_cavi, lmbda_out2_cavi, elbo2_cavi = CAVI_algorithm(w2, K2, n_iter_cavi2, eta_prior2, alpha_prior2)
end_cavi2 = time.time()

start_svi2 = time.time()
phi_out2_svi, gamma_out2_svi, lmbda_out2_svi, elbo2_svi = SVI_algorithm(w2, K2, S2, n_iter_svi2, eta_prior2, alpha_prior2)
end_svi2 = time.time()

final_phi2_cavi = phi_out2_cavi[-1]
final_gamma2_cavi = gamma_out2_cavi[-1]
final_lmbda2_cavi = lmbda_out2_cavi[-1]
final_phi2_svi = phi_out2_svi[-1]
final_gamma2_svi = gamma_out2_svi[-1]
final_lmbda2_svi = lmbda_out2_svi[-1]
```

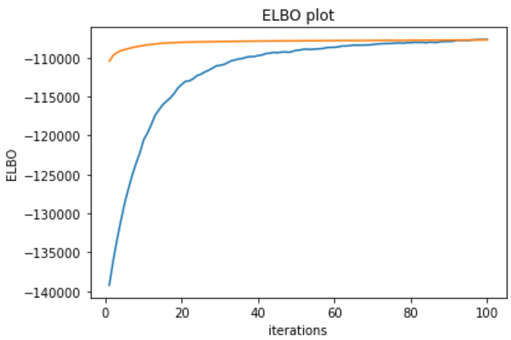
Evaluation

Do not expect perfect results in terms expectations being identical to the "true" theta and beta. Do not expect the ELBO plot of your SVI alg to be the same as the CAVI alg. However, it should increase and be in the same ball park as that of the CAVI alg.

In [ ]:
```
np.set_printoptions(formatter={'float': lambda x: "{0:0.3f}".format(x)})
print(f"----- Recall label switching - compare E[theta] and true theta and check for label switching -----")
print(f"E[theta] of doc 0 SVI:      {final_gamma2_svi[0] / np.sum(final_gamma2_svi[0], axis=0, keepdims=True)}")
print(f"E[theta] of doc 0 CAVI:     {final_gamma2_cavi[0] / np.sum(final_gamma2_cavi[0], axis=0, keepdims=True)}")
print(f"True theta of doc 0:        {theta2[0]}")

print(f"----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----")
print(f"E[beta] k=0:    {final_lmbda2_svi[0, :] / np.sum(final_lmbda2_svi[0, :], axis=-1, keepdims=True)}")
print(f"E[beta] k=1:    {final_lmbda2_svi[1, :] / np.sum(final_lmbda2_svi[1, :], axis=-1, keepdims=True)}")
print(f"E[beta] k=2:    {final_lmbda2_svi[2, :] / np.sum(final_lmbda2_svi[2, :], axis=-1, keepdims=True)}")
print(f"True beta k=0:  {beta2[0, :]}")
print(f"True beta k=1:  {beta2[1, :]}")
print(f"True beta k=2:  {beta2[2, :]}")

print(f"Time SVI: {end_svi2 - start_svi2}")
print(f"Time CAVI: {end_cavi2 - start_cavi2}")
```

```
----- Recall label switching - compare E[theta] and true theta and check for label switching -----
E[theta] of doc 0 SVI:      [0.288 0.077 0.635]
E[theta] of doc 0 CAVI:     [0.238 0.338 0.424]
True theta of doc 0:        [0.128 0.619 0.253]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
E[beta] k=0:    [0.011 0.052 0.095 0.093 0.049 0.034 0.039 0.121 0.458 0.049]
E[beta] k=1:    [0.262 0.183 0.043 0.024 0.012 0.119 0.028 0.296 0.027 0.006]
E[beta] k=2:    [0.201 0.062 0.062 0.290 0.003 0.003 0.002 0.141 0.023 0.213]
True beta k=0:  [0.067 0.105 0.077 0.066 0.046 0.087 0.048 0.186 0.277 0.040]
True beta k=1:  [0.139 0.067 0.074 0.230 0.007 0.008 0.002 0.158 0.134 0.181]
True beta k=2:  [0.295 0.123 0.047 0.116 0.010 0.078 0.012 0.222 0.057 0.041]
Time SVI: 16.92941117286682
Time CAVI: 56.193533420562744
```

In [ ]:
```
plt.plot(list(range(1, n_iter_cavi2 + 1)), elbo2_svi[np.arange(0, n_iter_svi2, int(n_iter_svi2 / n_iter_cavi2))])
plt.plot(list(range(1, n_iter_cavi2 + 1)), elbo2_cavi)
plt.title("ELBO plot")
plt.xlabel("iterations")
plt.ylabel("ELBO")
plt.show()
```



In [ ]:
```
# Add your own code for evaluation here (will not be graded)
```

CASE 3

Medium small dataset, one iteration for time analysis.

In [ ]:
```
np.random.seed(0)

# Data simulation parameters
D3 = 10**4
N3 = 500
K3 = 5
W3 = 10
eta_sim3 = np.ones(W3)
alpha_sim3 = np.ones(K3)

w3, z3, theta3, beta3 = generate_data_torch(D3, N3, K3, W3, eta_sim3, alpha_sim3)

# Inference parameters
```

```python
n_iter3 = 1
eta_prior3 = np.ones(W3) * 1.
alpha_prior3 = np.ones(K3) * 1.
S3 = 100 # batch size

start_cavi3 = time.time()
phi_out3_cavi, gamma_out3_cavi, lmbda_out3_cavi, elbo3_cavi = CAVI_algorithm(w3, K3, n_iter3, eta_prior3, alpha_prior3)
end_cavi3 = time.time()

start_svi3 = time.time()
phi_out3_svi, gamma_out3_svi, lmbda_out3_svi, elbo3_svi = SVI_algorithm(w3, K3, S3, n_iter3, eta_prior3, alpha_prior3)
end_svi3 = time.time()

final_phi3_cavi = phi_out3_cavi[-1]
final_gamma3_cavi = gamma_out3_cavi[-1]
final_lmbda3_cavi = lmbda_out3_cavi[-1]
final_phi3_svi = phi_out3_svi[-1]
final_gamma3_svi = gamma_out3_svi[-1]
final_lmbda3_svi = lmbda_out3_svi[-1]
```

In [ ]:

```python
print(f"Examine per iteration run time.")
print(f"Time SVI: {end_svi3 - start_svi3}")
print(f"Time CAVI: {end_cavi3 - start_cavi3}")
```

```
Examine per iteration run time.
Time SVI: 7.3243772983551025
Time CAVI: 95.36939764022827
```

In [ ]:

```python
# Add your own code for evaluation here (will not be graded)
```