Machine Learning Advanced Course: Assignment 1A

Michel Le Dez, Corentin Flandre

December 05, 2023

Contents

1 Assignment 1A

1.1 Exponential Family

An exponential-family distribution with natural parameters is in the following form:

$$p(x|\theta) = h(x)\exp(\eta(\theta) \cdot T(x) - A(\eta))$$

Question 1.1.1:

We have:

- $\theta = \lambda$
- $\eta(\theta) = \log(\theta) = \log(\lambda)$
- $h(x) = \frac{1}{x!}$
- $T(x) = x$
- $A(\eta) = e^{\eta} = e^{\log \lambda} = \lambda$

Therefore:

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

We recognize the probability mass function of the **Poisson distribution**.

Question 1.1.2:

We have:

- $\theta = [\alpha, \beta]$
- $\eta(\theta) = [\theta_1 - 1, -\theta_2] = [\alpha - 1, -\beta]$
- $h(x) = 1$
- $T(x) = [\log x, x]$
- $A(\eta) = \log\Gamma(\eta_1 + 1) - (\eta_1 + 1)\log(-\eta_2) = \log\Gamma(\alpha) - \alpha\log(\beta) = \log(\frac{\Gamma(\alpha)}{\beta^{\alpha}})$

Therefore:

$$\begin{aligned} & p(x|\theta) = h(x)\exp(\eta(\theta) \cdot T(x) - A(\eta)) \\ \iff \quad & p(x|\alpha, \beta) = \exp((\alpha - 1)\log x - \beta x - \log(\frac{\Gamma(\alpha)}{\beta^{\alpha}})) \\ & \boxed{p(x|\alpha, \beta) = \frac{\beta^{\alpha}}{\Gamma(\alpha)}x^{\alpha-1}e^{-\beta x}} \end{aligned}$$

We recognize the probability density function of the **Gamma distribution**.

Question 1.1.3:

We have:

- $\theta = [\mu, \sigma^2]$

- $\eta(\theta) = [\frac{\theta_1}{\theta_2}, -\frac{1}{2\theta_2}] = [\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}]$
- $h(x) = \frac{1}{\sqrt{2\pi}}$
- $T(x) = [x, x^2]$
- $A(\eta) = -\frac{\eta_1^2}{4\eta_2} - \frac{1}{2}\log(-2\eta_2) = \frac{\mu^2}{2\sigma^2} - \log(\frac{1}{\sigma})$

Therefore:

$$p(x|\theta) = h(x)\exp(\eta(\theta) \cdot T(x) - A(\eta))$$
$$\iff \quad p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}}\exp(\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 - \frac{\mu^2}{2\sigma^2} + \log(\frac{1}{\sigma}))$$
$$\boxed{p(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-\frac{1}{2\sigma^2}(x-\mu)^2)}$$

We recognize the probability density function of the **Normal distribution**.

Question 1.1.4:

We have:

- $\theta = \lambda$
- $\eta(\theta) = -\theta = -\lambda$
- $h(x) = 2$
- $T(x) = x$
- $A(\eta) = -\log(-\frac{\eta}{2}) = -\log(\frac{\lambda}{2})$

Therefore:

$$p(x|\theta) = h(x)\exp(\eta(\theta) \cdot T(x) - A(\eta))$$
$$\iff \quad p(x|\lambda) = 2\exp(-\lambda x + \log(\frac{\lambda}{2}))$$
$$\boxed{p(x|\lambda) = \lambda e^{-\lambda x}}$$

We recognize the probability density function of the **Exponential distribution**.

Question 1.1.5:

We have:

- $\theta = [\psi_1, \psi_2]$
- $\eta(\theta) = [\theta_1 - 1, \theta_2 - 2] = [\psi_1 - 1, \psi_2 - 2]$
- $h(x) = 1$
- $T(x) = [\log x, \log(1-x)]$
- $A(\eta) = \log\Gamma(\eta_1 + 1) + \log\Gamma(\eta_2 + 1) - \log\Gamma(\eta_1 + \eta_2 + 2) = -\log\frac{\Gamma(\psi_1+\psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)}$

Therefore:

$$p(x|\theta) = h(x)\exp(\eta(\theta) \cdot T(x) - A(\eta))$$
$$\iff \quad p(x|\psi_1, \psi_2) = \exp((\psi_1 - 1)\log x + (\psi_2 - 1)\log(1-x) + \log\frac{\Gamma(\psi_1 + \psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)})$$
$$\boxed{p(x|\psi_1, \psi_2) = \frac{\Gamma(\psi_1 + \psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)}x^{\psi_1 - 1}(1-x)^{\psi_2 - 1}}$$

We recognize the probability density function of the **Beta distribution**.

1.2 Dependencies in a Directed Graphical Model

Question 1.2.6: Yes.

Question 1.2.7: No.

Question 1.2.8: Yes.

Question 1.2.9: No.

Question 1.2.10: No.

Question 1.2.11: No.

1.3 CAVI

We have the following distribution:

$$p(\tau) = Gam(\tau|a_0, b_0) = \frac{b_0^{a_0}}{\Gamma(a_0)}\tau^{a_0-1}e^{-b_0\tau} \tag{1}$$

$$p(\mu|\tau) = \mathcal{N}(\mu|\mu_0, (\lambda_0\tau)^{-1}) = \frac{\sqrt{\lambda_0\tau}}{\sqrt{2\pi}}\exp(-\frac{\lambda_0\tau}{2}(\mu-\mu_0)^2) \tag{2}$$

$$p(D|\mu,\tau) = \prod_{n=1}^{N}\frac{\sqrt{\tau}}{\sqrt{2\pi}}\exp(-\frac{\tau}{2}(x_n-\mu)^2) = (\frac{\tau}{2\pi})^{\frac{N}{2}}\exp(-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2) \tag{3}$$

Question 1.3.12: The function implementation for generating data points, as well as the code for displaying the histogram is in the annex. Here are the the histograms we obtained:

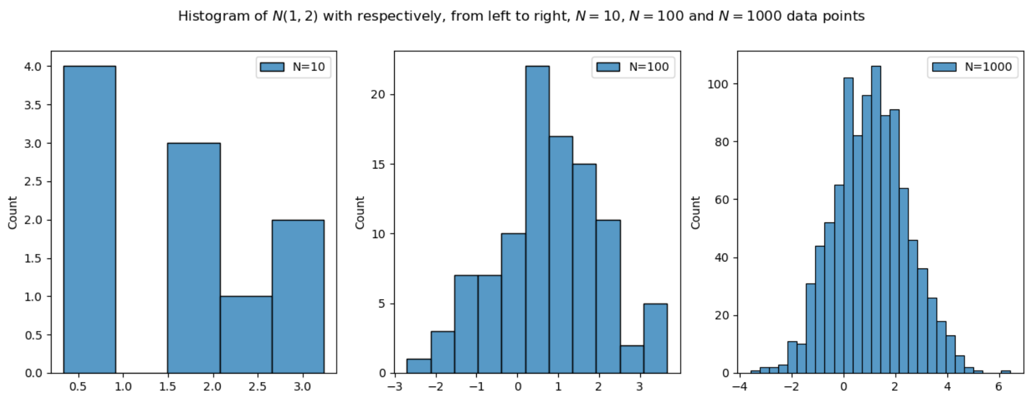


Figure 1: Histogram of $\mathcal{N}(\mu, \frac{1}{\tau})$ with $\mu = 1$ and $\tau = 0.5$ with respectively, from left to right, $N = 10$, $N = 100$, $N = 1000$ data points.

We observe that the more data we have, the closer the histogram is to the normal distribution that generated it.

Question 1.3.13: The likelihood of the data points $D = x_{1:N}$ given the parameter μ, τ is as follows:

$$l(\mu,\tau) := p(D|\mu,\tau) = (\frac{\tau}{2\pi})^{\frac{N}{2}}\exp(-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2)$$

$$\iff \log(l(\mu,\tau)) = \frac{N}{2}\log\tau - \frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2 + const$$

We are looking for the parameters μ and τ which maximise the likelihood $l(\mu,\tau)$, which is equivalent to maximising the log-likelihood given that the log function is a monotonically increasing one. The constant term above gathers all the terms that do not depend on μ or τ. Deriving the gradient of $l(\mu,\tau)$ and setting it to 0 at (μ_{MLE}, τ_{MLE}) yields the following system of equations:

$$\begin{cases} \tau_{MLE}\sum_{n=1}^{N} x_n - \tau_{MLE}N\mu_{MLE} = 0 \\ \frac{N}{2\tau_{MLE}} - \frac{1}{2}\sum_{n=1}^{N}(x_n-\mu_{MLE})^2 = 0 \end{cases}$$

which yields to the following solution:

$$\boxed{\begin{cases} \mu_{MLE} = \bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \\ \tau_{MLE} = \frac{1}{\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^2} \end{cases}}$$

To verify that the point (μ_{MLE}, τ_{MLE}) definitely maximises the likelihood we can compute the hessian of the log-likelihood at this point, which yields to:

$$\begin{bmatrix} -\frac{N^2}{\sum_{n=1}^{N}(x_n-\bar{x})^2} & 0 \\ 0 & -\frac{1}{2N}(\sum_{n=1}^{N}(x_n-\bar{x})^2)^2 \end{bmatrix}$$

We can see the eigenvalues of the hessian are strictly negative, therefore (μ_{MLE}, τ_{MLE}) definitely maximises the likelihood.

Question 1.3.14: To compute the posterior, we are going to use the Bayes' theorem and gather in the constant term all the terms that do not depend on μ or τ. Here is the posterior:

$$\begin{aligned} p(\mu,\tau|D) =& p(D|\mu,\tau)p(\mu|\tau)p(\tau)/p(D) \\ \iff \log p(\mu,\tau|D) =& \log p(D|\mu,\tau) + \log p(\mu|\tau) + \log p(\tau) + const \\ =& \frac{N}{2}\log\tau - \frac{\tau}{2}\sum_{n=1}^{N}(x_n^2+\mu^2-2x_n\mu) + \frac{1}{2}\log\tau - \frac{\lambda_0\tau}{2}(\mu^2+\mu_0^2-2\mu\mu_0) \\ &+ (a_0-1)\log\tau - b_0\tau + const \\ =& (a_0+\frac{N}{2}-\frac{1}{2})\log\tau - (b_0+\frac{1}{2}\sum_{n=1}^{N}x_n^2+\frac{\lambda_0\mu_0^2}{2})\tau + (\sum_{n=1}^{N}x_n+\lambda_0\mu_0)\tau\mu \\ &- \frac{\tau}{2}(\lambda_0+N)\mu^2 + const \end{aligned}$$

However, we know that for $\mu,\tau \sim NormalGamma(\mu_0^*, \lambda_0^*, a_0^*, b_0^*)$, the logarithm of the probability density function is as follows:

$$\begin{aligned} \log p(\mu,\tau|\mu_0^*,\lambda_0^*,a_0^*,b_0^*) &= (a_0^*-\frac{1}{2})\log\tau - b_0^*\tau - \frac{\lambda_0^*\mu_0^{2*}}{2} + \lambda_0^*\mu_0^*\tau\mu - \frac{\tau}{2}\lambda_0^*\mu^2 + const \\ &= (a_0^*-\frac{1}{2})\log\tau - b_0^*\tau - \frac{\tau\lambda_0^*}{2}(\mu-\mu_0^*)^2 + const \end{aligned}$$

By identification, we have $a_0^* = a_0 + \frac{N}{2}$, $\lambda_0^* = \lambda_0 + N$ and $\mu_0^* = \frac{\sum_{n=1}^{N}x_n+\lambda_0\mu_0}{\lambda_0+N}$. For b_0^*, let's rewrite the log posterior in the form of the second equality above by adding and subtracting the missing term $\frac{1}{2}\frac{(\sum_{n=1}^{N}x_n+\lambda_0\mu_0)^2}{\lambda_0+N}\tau$ for completing the square, which yields to:

$$\begin{aligned} \log p(\mu,\tau|D) =& (a_0+\frac{N}{2}-\frac{1}{2})\log\tau - (b_0+\frac{1}{2}\sum_{n=1}^{N}x_n^2+\frac{\lambda_0\mu_0^2}{2}-\frac{1}{2}\frac{(\sum_{n=1}^{N}x_n+\lambda_0\mu_0)^2}{\lambda_0+N})\tau \\ &- \frac{\tau(\lambda_0+N)}{2}(\mu-\frac{\sum_{n=1}^{N}x_n+\lambda_0\mu_0}{\lambda_0+N})^2 + const \end{aligned}$$

Here, we can easily identify b_0^* and we summarise the results below:

$$\boxed{\begin{aligned} &\mu,\tau|D \sim NormalGamma(\mu_0^*,\lambda_0^*,a_0^*,b_0^*) \text{ with the following parameters} \\ &\qquad \mu_0^* = \frac{\sum_{n=1}^{N}x_n+\lambda_0\mu_0}{\lambda_0+N} \\ &\qquad \lambda_0^* = \lambda_0+N \\ &\qquad a_0^* = a_0+\frac{N}{2} \\ &\qquad b_0^* = b_0+\frac{1}{2}\sum_{n=1}^{N}x_n^2+\frac{\lambda_0\mu_0^2}{2}-\frac{1}{2}\frac{(\sum_{n=1}^{N}x_n+\lambda_0\mu_0)^2}{\lambda_0+N} \end{aligned}}$$

Question 1.3.15: The mean field approximation for the variational distribution is the following:

$$q(\mu,\tau) = q_\mu(\mu)q_\tau(\tau).$$

The log of the joint distribution can be written as follows:

$$\log p(x, \mu, \tau) = \log p(x|\mu, \tau) + \log p(\mu|\tau) + \log p(\tau),$$

with:

$$\log p(x|\mu, \tau) = \frac{N}{2}\log\tau - \frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2 + const$$
$$\log p(\mu|\tau) = \frac{1}{2}\log\tau - \frac{\lambda_0\tau}{2}(\mu - \mu_0)^2 + const$$
$$\log p(\tau) = (a_0 - 1)\log\tau - b_0\tau + const$$

where the constant terms include terms that do not depend on μ or τ. Let's derive now the coordinate ascent update for μ by including terms that do not depend on μ in the constant term (i.e $\log p(\tau)$, $\frac{N}{2}\log\tau$, $\frac{1}{2}\log\tau$, $-\frac{1}{2}\mathbf{E}_{q(\tau)}[\tau]\sum_{n=1}^{N} x_n^2$ and $-\frac{1}{2}\mathbf{E}_{q(\tau)}[\tau]\lambda_0\mu_0^2$):

$$\begin{aligned}\log q^*(\mu) &= \mathbf{E}_{q(\tau)}[\log p(x, \mu, \tau)]\\ &= -\mathbf{E}_{q(\tau)}\left[\frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2 + \frac{\lambda_0\tau}{2}(\mu - \mu_0)^2\right] + const\\ &= -\frac{1}{2}\mathbf{E}_{q(\tau)}[\tau](\sum_{n=1}^{N}(x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2) + const\\ &= \mathbf{E}_{q(\tau)}[\tau](\sum_{n=1}^{N} x_n + \lambda_0\mu_0)\mu - \frac{1}{2}\mathbf{E}_{q(\tau)}[\tau](\lambda_0 + N)\mu^2 + const\end{aligned}$$

However, we know that for $\mu \sim Normal(\tilde{\mu_0}, \tilde{\lambda_0}^{-1})$, the log of the probability density function is as follows:

$$\log p(\mu|\tilde{\mu_0}, \tilde{\lambda_0}^{-1}) = \tilde{\lambda_0}\tilde{\mu_0}\mu - \frac{\tilde{\lambda_0}}{2}\mu^2 + const$$

By identification, we have:

$$\boxed{\begin{aligned}&q^*(\mu) = Normal(\mu|\tilde{\mu_0}, \tilde{\lambda_0}^{-1}) \text{ with the following parameters}\\ &\quad \tilde{\mu_0} = \frac{\sum_{n=1}^{N} x_n + \lambda_0\mu_0}{\lambda_0 + N}\\ &\quad \tilde{\lambda_0} = \mathbf{E}_{q(\tau)}[\tau](\lambda_0 + N)\\ &\quad \text{with}\\ &\quad\quad \mathbf{E}_{q(\tau)}[\tau] = \frac{\tilde{a_0}}{\tilde{b_0}}\end{aligned}}$$

Let's derive now the coordinate ascent update for τ by including terms that do not depend on τ in the constant term:

$$\begin{aligned}\log q^*(\tau) &= \mathbf{E}_{q(\mu)}[\log p(x, \mu, \tau)]\\ &= \frac{N}{2}\log\tau - \frac{\tau}{2}\sum_{n=1}^{N}\mathbf{E}_{q(\mu)}[(x_n - \mu)^2] + \frac{1}{2}\log\tau - \frac{\lambda_0\tau}{2}\mathbf{E}_{q(\mu)}[(\mu - \mu_0)^2] + (a_0 - 1)\log\tau - b_0\tau\\ &= (a_0 + \frac{N+1}{2} - 1)\log\tau - (b_0 + \frac{1}{2}\mathbf{E}_{q(\mu)}[\sum_{n=1}^{N}(x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2])\tau\end{aligned}$$

However, we know that for $\tau \sim Gamma(\tilde{a_0}, \tilde{b_0})$, the log of the probability density function is as follows:

$$\log p(\tau|\tilde{a_0}, \tilde{b_0}) = (\tilde{a_0} - 1)\log\tau - \tilde{b_0}\tau + const$$

By identification, we have:

$$\boxed{\begin{aligned}
&q^*(\tau) = Gamma(\tau|\tilde{a_0}, \tilde{b_0}) \text{ with the following parameters}\\
&\qquad \tilde{a_0} = a_0 + \frac{N+1}{2}\\
&\qquad \tilde{b_0} = b_0 + \frac{1}{2}[\sum_{n=1}^{N} x_n^2 + \lambda_0\mu_0^2 - 2(\lambda_0\mu_0 + \sum_{n=1}^{N} x_n)\mathbf{E}_{q(\mu)}[\mu] + (\lambda_0 + \sum_{n=1}^{N} x_n^2)\mathbf{E}_{q(\mu)}[\mu^2]]\\
&\quad \text{with}\\
&\qquad\quad \mathbf{E}_{q(\mu)}[\mu] = \tilde{\mu_0}\\
&\qquad\quad \mathbf{E}_{q(\mu)}[\mu^2] = \frac{1}{\tilde{\lambda_0}} + \tilde{\mu_0}^2
\end{aligned}}$$

Now that we have derived the variational distributions, we can compute the ELBO $\mathcal{L}$ to monitor its convergence during the CAVI algorithm:

$$\mathcal{L} = \mathbf{E}_{q(\mu,\tau)}\left[\log\frac{p(x,\mu,\tau)}{q(\mu,\tau)}\right] = \mathbf{E}_{q(\mu)q(\tau)}\left[\log\frac{p(x|\mu,\tau)p(\mu|\tau)p(\tau)}{q(\mu)q(\tau)}\right]$$

which finally gives:

$$\boxed{\begin{aligned}
&\mathcal{L} = \mathbf{E}_{q(\mu)q(\tau)}[\log p(x|\mu,\tau)] + \mathbf{E}_{q(\mu)q(\tau)}[\log p(\mu|\tau)] + \mathbf{E}_{q(\tau)}[\log p(\tau)]\\
&\qquad - \mathbf{E}_{q(\mu)}[\log q(\mu)] - \mathbf{E}_{q(\tau)}[\log q(\tau)]\\
&\text{with:}\\
&\quad \mathbf{E}_{q(\mu)q(\tau)}[\log p(x|\mu,\tau)] = -\frac{N}{2}\log(2\pi) + \frac{N}{2}\mathbf{E}_{q(\tau)}[\log q(\tau)] - \frac{1}{2}\mathbf{E}_{q(\tau)}[\tau]\sum_{n=1}^{N}(x_n^2 + \mathbf{E}_{q(\mu)}[\mu^2] - 2x_n\mathbf{E}_{q(\mu)}[\mu])\\
&\quad \mathbf{E}_{q(\mu)q(\tau)}[\log p(\mu|\tau)] = \frac{1}{2}\log(\frac{\lambda_0}{2\pi}) + \frac{1}{2}\mathbf{E}_{q(\tau)}[\log q(\tau)] - \frac{\lambda_0}{2}\mathbf{E}_{q(\tau)}[\tau](\mathbf{E}_{q(\mu)}[\mu^2] + \mu_0^2 - 2\mu_0\mathbf{E}_{q(\mu)}[\mu])\\
&\quad \mathbf{E}_{q(\tau)}[\log p(\tau)] = (a_0 - 1)\mathbf{E}_{q(\tau)}[\log q(\tau)] + a_0\log b_0 - b_0\mathbf{E}_{q(\tau)}[\tau] - \log\Gamma(a_0)\\
&\quad \mathbf{E}_{q(\mu)}[\log q(\mu)] = -H(\mu) = -\frac{1}{2}\log(\frac{2\pi}{\tilde{\lambda_0}}) - \frac{1}{2}\\
&\quad \mathbf{E}_{q(\tau)}[\log q(\tau)] = -(\tilde{a_0} - \log\tilde{b_0} + \log\Gamma(\tilde{a_0}) + (1 - \tilde{a_0})\psi(\tilde{a_0}))\\
&\text{and:}\\
&\quad \mathbf{E}_{q(\tau)}[\log q(\tau)] = \psi(\tilde{a_0}) - \log\tilde{b_0}\\
&\quad \mathbf{E}_{q(\tau)}[\tau] = \frac{\tilde{a_0}}{\tilde{b_0}}\\
&\quad \mathbf{E}_{q(\mu)}[\mu] = \tilde{\mu_0}\\
&\quad \mathbf{E}_{q(\mu)}[\mu^2] = \frac{1}{\tilde{\lambda_0}} + \tilde{\mu_0}^2
\end{aligned}}$$

We implemented the CAVI algorithm using the variational distributions found above, and we monitored the convergence of the ELBO. Finally, we compared the variational distribution $q(\mu, \tau)$ obtained at the end of the CAVI algorithm with the true posterior that we computed at **Question 1.3.14**. Here are the results we obtained for the 3 data sets generated at **Question 1.13.12**. We also display the ML estimate computed at **Question 1.13.13**.

First data set:

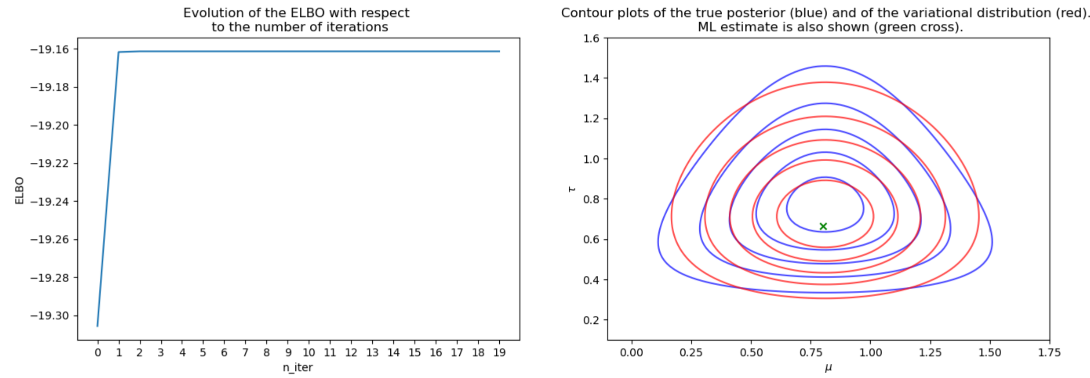


Figure 2: Left: Evolution of the ELBO with respect to the number of iterations. Right: Comparison of the true posterior (blue) with the variational distribution (red) obtained with the CAVI algorithm. The ML estimate (green) is also displayed.

Second data set:

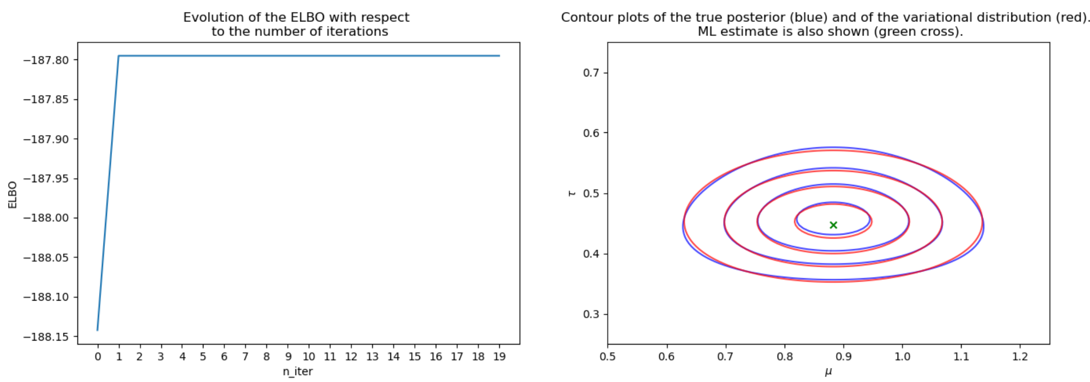


Figure 3: Left: Evolution of the ELBO with respect to the number of iterations. Right: Comparison of the true posterior (blue) with the variational distribution (red) obtained with the CAVI algorithm. The ML estimate (green) is also displayed.

Third data set:

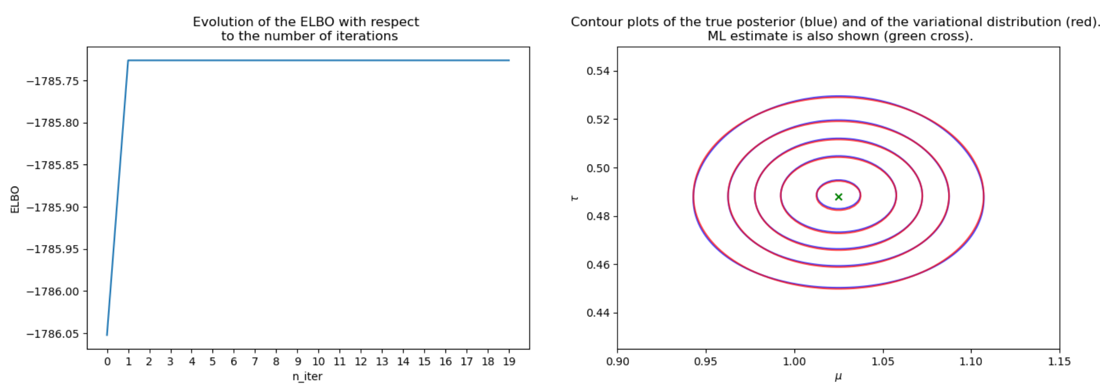


Figure 4: Left: Evolution of the ELBO with respect to the number of iterations. Right: Comparison of the true posterior (blue) with the variational distribution (red) obtained with the CAVI algorithm. The ML estimate (green) is also displayed.

Discussion of the results:

> ***Concerning the ELBO, we notice that its convergence is very fast****, a small number of iterations is enough to converge. Concerning the comparison of the variational distribution obtained at the end of the CAVI algorithm with the true posterior distribution, we notice that* ***the more data points we have in our data set, the more accurate will be the variational distribution to the true posterior*** *as the right plot of Figure 5 shows where they almost overlap perfectly.* ***It is the same observation concerning the ML estimate, the more data points we have in our data set, the more accurate it will be*** *as we can see with the right plot of Figure 5 where the ML estimate is effectively at the center of the true posterior.*

1.4 SVI - LDA

Question 1.4.16: According to the Hoffman paper, local hidden variables $z_{n,j}$ are defined by the fact that their complete conditional probability distribution are determined by β the global hidden variables, α the fixed parameters and the other local variables x_n, $z_{n,-j}$ in the n$^{\text{th}}$ context. Mathematically, the definition is the following:

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

Question 1.4.17: In the LDA model of the Hoffman paper, the global hidden variables are the $\beta_k, k = 1 \cdots K$ and the local hidden variables are the $\theta_d, d = 1 \cdots D$ and $z_{d,n}, d = 1 \cdots D, n = 1 \cdots N$.

Question 1.4.18: To compute the ELBO, we used this source [1].

$$\boxed{\begin{aligned} ELBO = & \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{w=1}^{W} \phi_{dnk}(\psi(\lambda_{kw}) - \psi(\sum_{w'} \lambda_{kw'}))w_{dnw} + \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K} \phi_{dnk}(\psi(\gamma_{dk}) - \psi(\sum_{k'} \gamma_{dk'})) \\ & - D \log B(\alpha) + \sum_{k=1}^{K}\sum_{d=1}^{D}(\alpha_k - 1)(\psi(\gamma_{dk}) - \psi(\sum_{k'} \gamma_{dk'})) \\ & - K \log B(\eta) + \sum_{k=1}^{K}\sum_{w=1}^{W}(\eta_w - 1)(\psi(\lambda_{kw}) - \psi(\sum_{w'} \lambda_{kw'})) \\ & + \sum_{d=1}^{D}\left[\log B(\gamma_d) + (\sum_{k=1}^{K} \gamma_{dk} - K)\psi(\sum_{k=1}^{K} \gamma_{dk}) - \sum_{k=1}^{K}(\gamma_{dk} - 1)\psi(\gamma_{dk})\right] \\ & + \sum_{k=1}^{K}\left[\log B(\lambda_k) + (\sum_{w=1}^{W} \lambda_{kw} - W)\psi(\sum_{w=1}^{W} \lambda_{kw}) - \sum_{w=1}^{W}(\lambda_{kw} - 1)\psi(\lambda_{kw})\right] \\ & - \sum_{d=1}^{D}\sum_{n=1}^{N}\sum_{k=1}^{K} \phi_{dnk} \log \phi_{dnk} \end{aligned}}$$

Question 1.4.19: Implementing the SVI updates in the notebook and running the two algorithms on the three data sets yielded the following results.

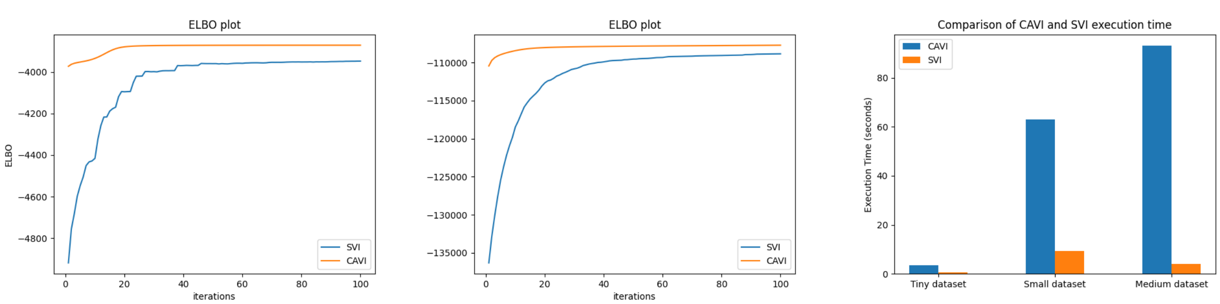


Figure 5: Respectively from left to right: (a) Evolution of the ELBO with respect to the number of iterations for both the CAVI algorithm and the SVI algorithm for the tiny dataset; (b) Same as (a) but for the small dataset; (c) Comparison of the execution time of the CAVI and SVI algorithms for all three datasets.

Discussion of the results:

> *Concerning the ELBO, we notice that both the one of the SVI and CAVI algorithm converge approximately around the same value (in terms of order of magnitude). The one of the CAVI algorithm seems to be faster in term of iterations to converge. However, what is of interest is the execution time and if we look at the bar plot, we can see that* ***the SVI algorithm is definitely faster than the CAVI algorithm for all three datasets.***

1.5 BBVI

Question 1.5.20: First, let's derive the gradient of the ELBO with respect to ν:

$$\begin{aligned}
\nabla_\nu \mathcal{L} &= \nabla_\nu \int q(\theta|\nu,\epsilon^2)\left[\log p(x,\theta) - \log q(\theta|\nu,\epsilon^2)\right] d\theta \\
&= \int \nabla_\nu q(\theta|\nu,\epsilon^2)\left[\log p(x,\theta) - \log q(\theta|\nu,\epsilon^2)\right] d\theta - \int q(\theta|\nu,\epsilon^2)\nabla_\nu \log q(\theta|\nu,\epsilon^2) d\theta \\
&= \int q(\theta|\nu,\epsilon^2)\nabla_\nu \log q(\theta|\nu,\epsilon^2)\left[\log p(x,\theta) - \log q(\theta|\nu,\epsilon^2)\right] d\theta
\end{aligned}$$

where we used $\nabla_\nu \log q(\theta|\nu,\epsilon^2) = \frac{\nabla_\nu q(\theta|\nu,\epsilon^2)}{q(\theta|\nu,\epsilon^2)}$ and $\int q(\theta|\nu,\epsilon^2)\nabla_\nu \log q(\theta|\nu,\epsilon^2) d\theta = \int \nabla_\nu q(\theta|\nu,\epsilon^2) d\theta$ which equals 0 by commuting the gradient and the integral and noticing that the integral equals 1. Therefore, an estimate of the gradient of the ELBO using one sample $z \sim q(\theta|\nu,\epsilon^2)$ is obtained as follows:

$$\nabla_\nu \mathcal{L} \approx \nabla_\nu \log q(z|\nu,\epsilon^2)\left[\log p(x,z) - \log q(z|\nu,\epsilon^2)\right]$$

with

$$\begin{aligned}
\nabla_\nu \log q(z|\nu,\epsilon^2) &= \frac{1}{\epsilon^2}(\log z - \nu) \\
\log q(z|\nu,\epsilon^2) &= -\log(z\epsilon\sqrt{2\pi}) - \frac{1}{2\epsilon^2}(\log z - \nu)^2 \\
\log p(x,z) &= -\log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}(x-z)^2 + \alpha\log\beta - \log\Gamma(\alpha) + (\alpha-1)\log z - \beta z
\end{aligned}$$

Question 1.5.21: Control variates is a variance reduction method used to compute a noisy estimate of the components of the gradient of the ELBO (obtained by the Rao-Blackwellization method) with low variance. Mathematically, if we want to estimate the expected value of f, it introduces a family $\hat{f}(z) = f(z) - a(h(z) - \mathbf{E}[h(z)])$ such that $\mathbf{E}[\hat{f}] = \mathbf{E}[f]$ but $Var[\hat{f}] < Var[f]$.

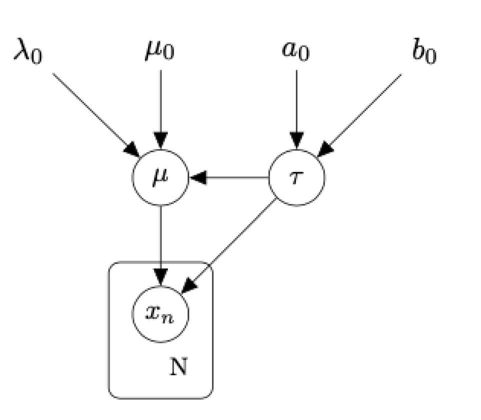
References

1. Blei, D., Ng, A., and Jordan, M. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022. `https://www.cs.columbia.edu/~blei/papers/BleiLafferty2009.pdf`.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import scipy.special as sp_spec
        import pandas as pd
        np.random.seed(3)
```

Assignment 1.3 - CAVI

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



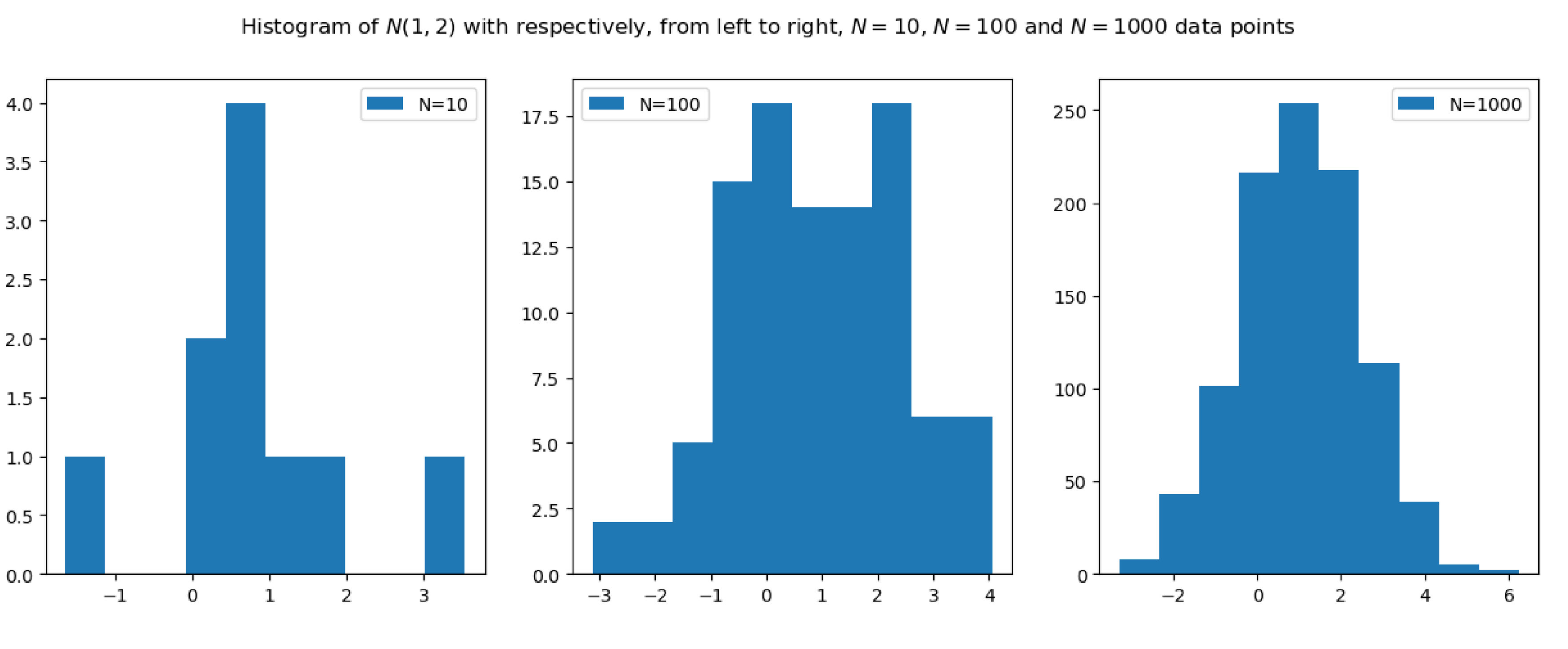
Question 1.3.12:

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

```
In [2]: 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 [3]: 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
        list_dataset = [dataset_1, dataset_2, dataset_3]
        fig, axes = plt.subplots(1, 3, figsize=(15, 5))
        for i in range(len(list_dataset)):
            dataset = list_dataset[i]
            axes[i].hist(dataset, label=f'N={len(dataset)}')
        #     sns.histplot(dataset, label=f'N={len(dataset)}', ax=axes[i])
            axes[i].legend()
        plt.suptitle(r'Histogram of $N(1, 2)$ with respectively, from left to right, $N=10$, $N=100$ and $N=1000$ data points')
        # plt.savefig('histogram.png')
        plt.show()
```



Question 1.3.13:

Find ML estimates of the variables μ and τ

```
In [4]: def ML_est(data):
          # insert your code
            mu_ml = np.mean(data)
            tau_ml = 1/np.var(data, ddof=0)
            return mu_ml, tau_ml
```

```
In [5]: print('For N=10, the Maximum Likelihood estimates for the parameters mu and tau are respectively', np.round(ML_est(dataset_1),2))
        print('For N=100, the Maximum Likelihood estimates for the parameters mu and tau are respectively', np.round(ML_est(dataset_2),2))
        print('For N=1000, the Maximum Likelihood estimates for the parameters mu and tau are respectively', np.round(ML_est(dataset_3),2))
```

```
For N=10, the Maximum Likelihood estimates for the parameters mu and tau are respectively [0.8  0.67]
For N=100, the Maximum Likelihood estimates for the parameters mu and tau are respectively [0.88 0.45]
For N=1000, the Maximum Likelihood estimates for the parameters mu and tau are respectively [1.03 0.49]
```

We observe that the more data we have, the closer the parameters are to the true values: [1.00, 0.50].

Question 1.3.14:

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

```
In [6]: # prior parameters
        mu_0 = 1
        lambda_0 = 0.5
        a_0 = 2
        b_0 = 1
```

Continue with a helper function that computes ELBO:

```
In [7]: def compute_elbo(D, a_0, b_0, mu_0, lambda_0, a_N , b_N, mu_N, lambda_N):
            # given the prior and posterior parameters together with the data,
            # compute ELBO here
            N = len(D)
        #     eps = 0.0001
            E_log_tau = sp_spec.digamma(a_N) - np.log(b_N)
            CR_likelihood = -N/2*np.log(2*np.pi) + N/2*E_log_tau - 0.5*(a_N/b_N)*np.sum(D**2 + 1/lambda_N + mu_N**2 - 2*D*mu_N)
            CR_mu = 0.5*np.log(lambda_0/(2*np.pi)) + 0.5*E_log_tau - lambda_0/2*(a_N/b_N)*(1/lambda_N + mu_N**2 + mu_0**2 - 2*mu_0*mu_N)
            CR_tau = (a_0 - 1)*E_log_tau + a_0*np.log(b_0) - b_0*(a_N/b_N) - np.log(sp_spec.gamma(a_0))
            H_mu = 0.5*np.log(2*np.pi/lambda_N) + 0.5
            H_tau = a_N - np.log(b_N) + sp_spec.gammaln(a_N) + (1 - a_N)*sp_spec.digamma(a_N)
            elbo = CR_likelihood + CR_mu + CR_tau + H_mu + H_tau
            return elbo
```

Now, implement the CAVI algorithm:

```
In [8]: def update_q_mu(D, mu_0, lambda_0, a_N, b_N):
            N = len(D)
            mu_N = (lambda_0*mu_0 + np.sum(D)) / (lambda_0 + N)
            lambda_N = (lambda_0 + N) * (a_N / b_N)
            return mu_N, lambda_N
```

```
In [9]: def update_q_tau(D, a_0, b_0, mu_0, lambda_0, mu_N, lambda_N):
            N = len(D)
            a_N = a_0 + (N+1)/2
            b_N = b_0 + 0.5*(np.sum(D**2) + lambda_0*(mu_0)**2 - 2*(np.sum(D) + lambda_0*mu_0)*mu_N + (lambda_0 + N)*(1/lambda_N + mu_N**2))
            return a_N, b_N
```

```
In [10]: def CAVI(D, a_0, b_0, mu_0, lambda_0, n_iter):
             # make an initial guess for the expected value of tau
         #     initial_guess_exp_tau = 5
             a_N = 20
             b_N = 10

             # CAVI iterations ...
             # save ELBO for each iteration, plot them afterwards to show convergence
             elbos = np.zeros(n_iter)
             for i in range(0, n_iter):
                 mu_N, lambda_N = update_q_mu(D, mu_0, lambda_0, a_N, b_N)
                 a_N, b_N = update_q_tau(D, a_0, b_0, mu_0, lambda_0, mu_N, lambda_N)
                 elbos[i] = 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
```

Question 1.3.15:

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

```
In [11]: def compute_exact_posterior(D, a_0, b_0, mu_0, lambda_0):
             # your implementation
             N = len(D)
             mu_0_star = (np.sum(D)+lambda_0*mu_0)/(lambda_0+N)
             lambda_0_star = lambda_0 + N
             a_0_star = a_0 + N/2
             b_0_star = b_0 + np.sum(D**2)/2 + lambda_0*(mu_0**2)/2 - 0.5*((np.sum(D) + lambda_0*mu_0)**2)/(lambda_0 + N)

             return mu_0_star, lambda_0_star, a_0_star, b_0_star
```

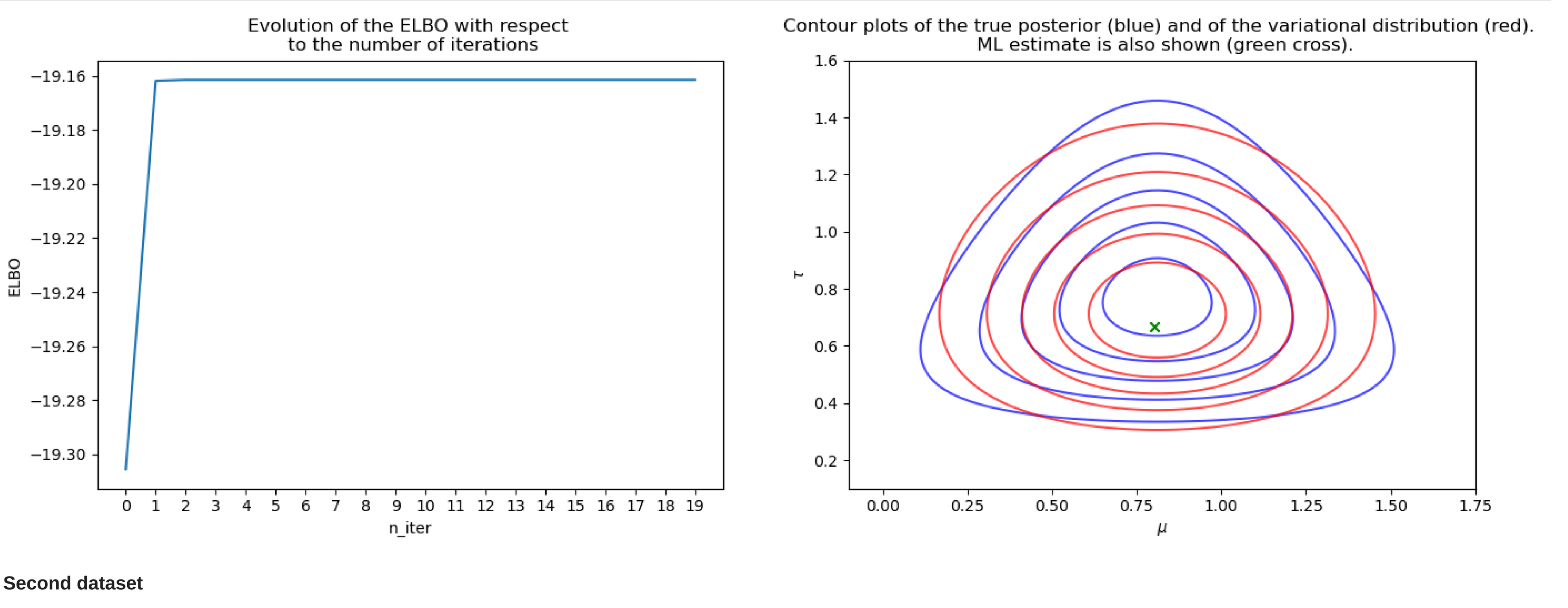
Question 1.3.16:

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 [12]: def log_pdf_exact_post(mu, tau, mu_0_star, lambda_0_star, a_0_star, b_0_star):
             return a_0_star * np.log(b_0_star) - sp_spec.gammaln(a_0_star) + 0.5 * np.log(lambda_0_star / (2 * np.pi)) + (a_0_star - 1/2) * np.log(tau) - b

         def log_pdf_approx_post(mu, tau, mu_N, lambda_N, a_N, b_N):
             return a_N * np.log(b_N) - sp_spec.gammaln(a_N) + 0.5 * np.log(lambda_N / (2 * np.pi)) + (a_N - 1) * np.log(tau) - b_N * tau - 0.5 * lambda_N *
```

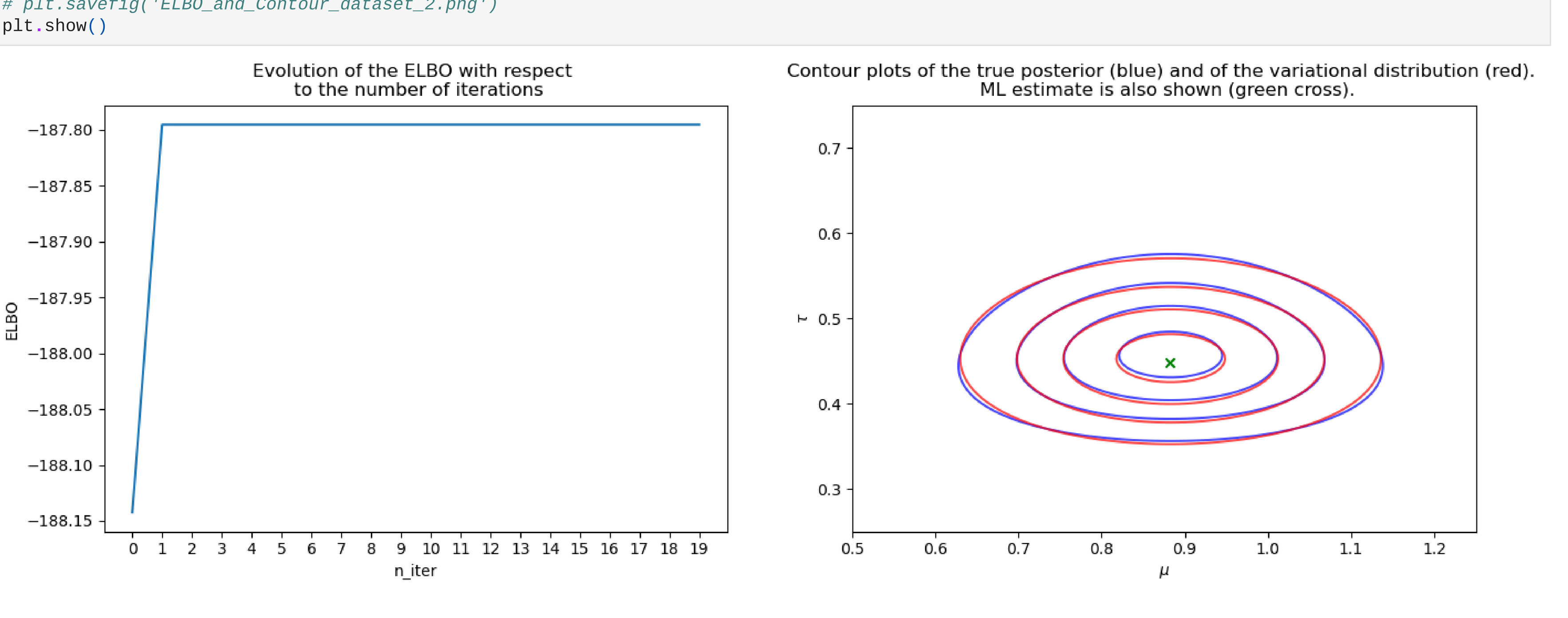
First dataset

```
In [13]: dataset = dataset_1
         n_iter = 20

         # ML estimates, VI estimates, true parameters
         mu_ml, tau_ml = ML_est(dataset)
         a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset, a_0, b_0, mu_0, lambda_0, n_iter)
         mu_0_star, lambda_0_star, a_0_star, b_0_star = compute_exact_posterior(dataset, a_0, b_0, mu_0, lambda_0)

         # plot elbos, show convergence
         fig, ax = plt.subplots(1, 2, figsize=(16, 5))
         ax[0].plot(elbos)
         ax[0].set_title('Evolution of the ELBO with respect \n to the number of iterations')
         ax[0].set_ylabel('ELBO')
         ax[0].set_xlabel('n_iter')
         ax[0].set_xticks(np.arange(0,n_iter))

         #  compare exact_post_dist with the CAVI result ( = q(a_N, b_N, mu_N, lambda_N) ) using for ex. contour plots, show also ML estimate on this plot
         # definition of the grid
         mu = np.linspace(-0.1, 1.75, 100)
         tau = np.linspace(0.1, 1.6, 100)
         muv, tauv = np.meshgrid(mu, tau)

         # compute exact posterior and approx posterior on the grid defined above
         lg_pdf_ex_post = log_pdf_exact_post(muv, tauv, mu_0_star, lambda_0_star, a_0_star, b_0_star)
         pdf_ex_post = np.exp(lg_pdf_ex_post - sp_spec.logsumexp(lg_pdf_ex_post))

         lg_pdf_aprx_post = log_pdf_approx_post(muv, tauv, mu_N, lambda_N, a_N, b_N)
         pdf_aprx_post = np.exp(lg_pdf_aprx_post - sp_spec.logsumexp(lg_pdf_aprx_post))

         # graph contour
         levels = 5
         ax[1].contour(muv, tauv, pdf_ex_post, colors='blue', levels=levels, alpha=0.7)
         ax[1].contour(muv, tauv, pdf_aprx_post, colors='red', levels=levels, alpha=0.7)
         ax[1].scatter(mu_ml, tau_ml, color='green', label='ML Estimates', marker='x')
         ax[1].set_title('Contour plots of the true posterior (blue) and of the variational distribution (red). \n ML estimate is also shown (green cross).'
         ax[1].set_xlabel(r'$\mu$')
         ax[1].set_ylabel(r'$\tau$')
         # ax.set_title('Contours des Distributions Réelle et Approximative')
         # plt.savefig('ELBO_and_Contour_dataset_1.png')
         plt.show()
```



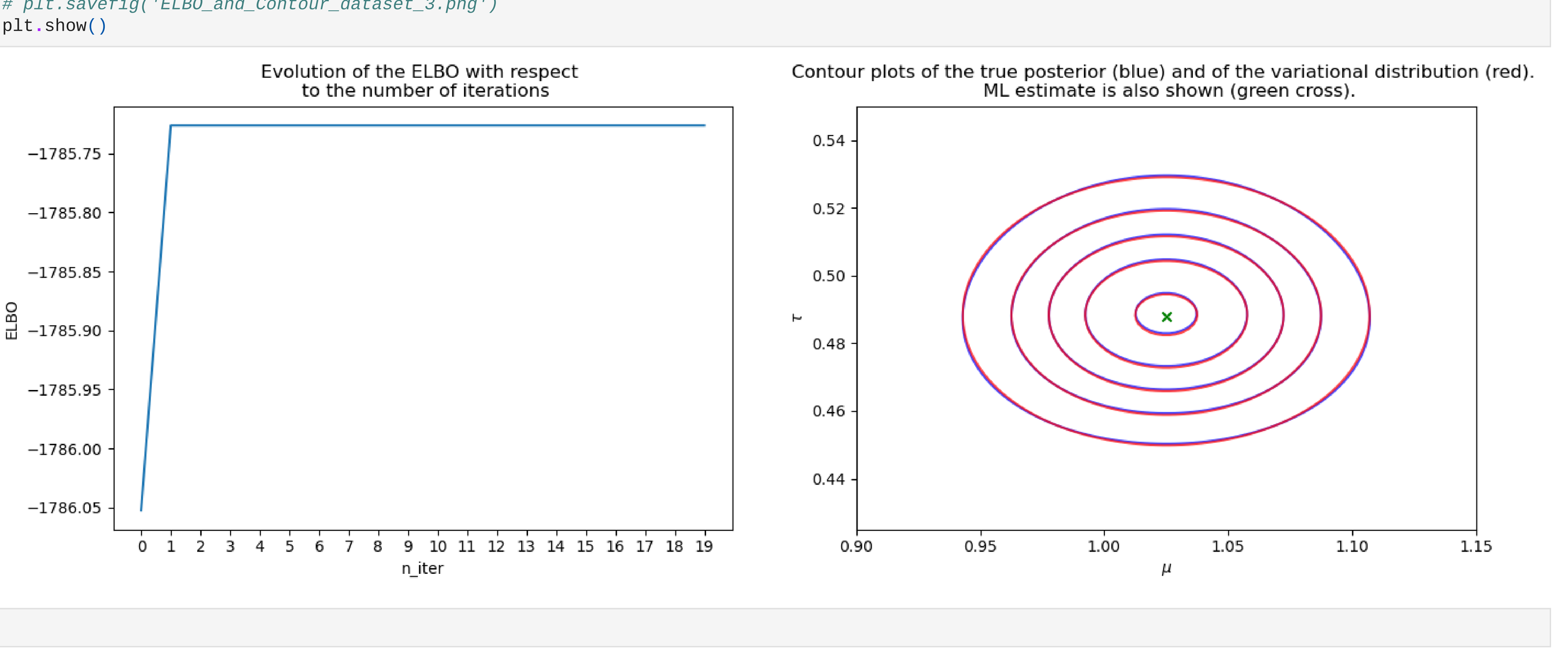
Second dataset

```
In [14]: dataset = dataset_2
         n_iter = 20

         # ML estimates, VI estimates, true parameters
         mu_ml, tau_ml = ML_est(dataset)
         a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset, a_0, b_0, mu_0, lambda_0, n_iter)
         mu_0_star, lambda_0_star, a_0_star, b_0_star = compute_exact_posterior(dataset, a_0, b_0, mu_0, lambda_0)

         # plot elbos, show convergence
         fig, ax = plt.subplots(1, 2, figsize=(16, 5))
         ax[0].plot(elbos)
         ax[0].set_title('Evolution of the ELBO with respect \n to the number of iterations')
         ax[0].set_ylabel('ELBO')
         ax[0].set_xlabel('n_iter')
         ax[0].set_xticks(np.arange(0,n_iter))

         #  compare exact_post_dist with the CAVI result ( = q(a_N, b_N, mu_N, lambda_N) ) using for ex. contour plots, show also ML estimate on this plot
         # definition of the grid
         mu = np.linspace(0.5, 1.25, 100)
         tau = np.linspace(0.25, 0.75, 100)
         muv, tauv = np.meshgrid(mu, tau)

         # compute exact posterior and approx posterior on the grid defined above
         lg_pdf_ex_post = log_pdf_exact_post(muv, tauv, mu_0_star, lambda_0_star, a_0_star, b_0_star)
         pdf_ex_post = np.exp(lg_pdf_ex_post - sp_spec.logsumexp(lg_pdf_ex_post))

         lg_pdf_aprx_post = log_pdf_approx_post(muv, tauv, mu_N, lambda_N, a_N, b_N)
         pdf_aprx_post = np.exp(lg_pdf_aprx_post - sp_spec.logsumexp(lg_pdf_aprx_post))

         # graph contour
         levels = 5
         ax[1].contour(muv, tauv, pdf_ex_post, colors='blue', levels=levels, alpha=0.7)
         ax[1].contour(muv, tauv, pdf_aprx_post, colors='red', levels=levels, alpha=0.7)
         ax[1].scatter(mu_ml, tau_ml, color='green', label='ML Estimates', marker='x')
         ax[1].set_title('Contour plots of the true posterior (blue) and of the variational distribution (red). \n ML estimate is also shown (green cross).'
         ax[1].set_xlabel(r'$\mu$')
         ax[1].set_ylabel(r'$\tau$')
         # ax.set_title('Contours des Distributions Réelle et Approximative')
         # plt.savefig('ELBO_and_Contour_dataset_2.png')
         plt.show()
```



Third dataset

```
In [15]: dataset = dataset_3
         n_iter = 20

         # ML estimates, VI estimates, true parameters
         mu_ml, tau_ml = ML_est(dataset)
         a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset, a_0, b_0, mu_0, lambda_0, n_iter)
         mu_0_star, lambda_0_star, a_0_star, b_0_star = compute_exact_posterior(dataset, a_0, b_0, mu_0, lambda_0)

         # plot elbos, show convergence
         fig, ax = plt.subplots(1, 2, figsize=(16, 5))
         ax[0].plot(elbos)
         ax[0].set_title('Evolution of the ELBO with respect \n to the number of iterations')
         ax[0].set_ylabel('ELBO')
         ax[0].set_xlabel('n_iter')
         ax[0].set_xticks(np.arange(0,n_iter))

         #  compare exact_post_dist with the CAVI result ( = q(a_N, b_N, mu_N, lambda_N) ) using for ex. contour plots, show also ML estimate on this plot
         # definition of the grid
         mu = np.linspace(0.9, 1.15, 100)
         tau = np.linspace(0.425, 0.550, 100)
         muv, tauv = np.meshgrid(mu, tau)

         # compute exact posterior and approx posterior on the grid defined above
         lg_pdf_ex_post = log_pdf_exact_post(muv, tauv, mu_0_star, lambda_0_star, a_0_star, b_0_star)
         pdf_ex_post = np.exp(lg_pdf_ex_post - sp_spec.logsumexp(lg_pdf_ex_post))

         lg_pdf_aprx_post = log_pdf_approx_post(muv, tauv, mu_N, lambda_N, a_N, b_N)
         pdf_aprx_post = np.exp(lg_pdf_aprx_post - sp_spec.logsumexp(lg_pdf_aprx_post))

         # graph contour
         levels = 5
         ax[1].contour(muv, tauv, pdf_ex_post, colors='blue', levels=levels, alpha=0.7)
         ax[1].contour(muv, tauv, pdf_aprx_post, colors='red', levels=levels, alpha=0.7)
         ax[1].scatter(mu_ml, tau_ml, color='green', label='ML Estimates', marker='x')
         ax[1].set_title('Contour plots of the true posterior (blue) and of the variational distribution (red). \n ML estimate is also shown (green cross).'
         ax[1].set_xlabel(r'$\mu$')
         ax[1].set_ylabel(r'$\tau$')
         # ax.set_title('Contours des Distributions Réelle et Approximative')
         # plt.savefig('ELBO_and_Contour_dataset_3.png')
         plt.show()
```




```
In [ ]:
```

```python
In [1]: 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

np.random.seed(42)
```

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.

```python
In [2]: 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: [0.3277851 0.6722149]
 Mean z of doc 0: [0.4 0.6]
Beta of topic 0: [0.15682799 0.41169399 0.01999794 0.         0.41148008]
Beta of topic 1: [0.00397547 0.6158494  0.34144902 0.         0.03872611]
Word to topic assignment, z, of document 0: [1 1 1 1 1 0 0 1 1 1]
Observed words, w, of document 0: [1 1 1 1 1 0 4 1 1 1]
Unique words and count of document 0: ['0: 3', '1: 33', '2: 6', '4: 8']
```

```python
In [3]: 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]) # size D x K

    # 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

```python
In [4]: 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

```python
In [5]: 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))

    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]
```

```python
In [6]: 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:  [0.552 0.448]
True theta of doc 0:          [0.328 0.672]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
Final E[beta] k=0: [0.    0.63  0.367 0.    0.003]
Final E[beta] k=1: [0.15  0.409 0.    0.    0.44 ]
True beta k=0: [0.157 0.412 0.02  0.    0.411]
True beta k=1: [0.004 0.616 0.341 0.    0.039]
```

SVI Implementation

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

```python
In [7]: def update_q_Z_svi(batch, w, gamma, lmbda):
    """
    OK: 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,s in enumerate(batch):
        for n in range(N):
            E_log_beta_wsn = E_log_beta[:, int(w_label[s, n])]
            E_log_theta_s = E_log_theta[s]
            log_rho_n = E_log_theta_s + E_log_beta_wsn
            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):
    """
    OK: rewrite to SVI update
    """
    K = phi.shape[-1]
    S = batch.shape[0]
    gamma = np.zeros((S, K))
    for i,s in enumerate(batch):
        E_Z_s = phi[s]
        gamma[i] = alpha + np.sum(E_Z_s, axis=0)  # sum over N
    return gamma


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

def SVI_algorithm(w, K, S, n_iter, eta, alpha):
    """
    OK: 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))

    # set step size, rho
    delay = 1
    forgetting_rate = 0.6
    rho = lambda t: (t + delay)**(-forgetting_rate)

    # for local convergence
    old_phi = phi
    old_gamma = gamma

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

        ###### SVI updates #######
        # Update locals on batch
        phi_batch = update_q_Z_svi(batch, w, gamma, lmbda)
        phi[batch] = phi_batch
        gamma_batch = update_q_theta_svi(batch, phi, alpha)
        gamma[batch] = gamma_batch
        i = 1
        while not(np.sum(np.abs(old_gamma[batch] - gamma[batch])) < 0.1*S and np.sum(np.abs(old_phi[batch] - phi[batch])) < 0.1*S) and i <= 20:
            # q(Z) update
            old_phi = phi
            phi_batch = update_q_Z_svi(batch, w, gamma, lmbda)
            phi[batch] = phi_batch

            # q(theta) update
            old_gamma = gamma
            gamma_batch = update_q_theta_svi(batch, phi, alpha)
            gamma[batch] = gamma_batch

            i += 1

        # Update intermediate global parameters and take a Robbins-Monro step
        # q(beta) update
        lmbda_batch = update_q_beta_svi(batch, w, phi, eta)
        lmbda = (1 - rho_t) * lmbda + rho_t * 1/S * np.sum(lmbda_batch, axis=0)

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

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

    return phi_out, gamma_out, lmbda_out, elbo
```

CASE 1

Tiny dataset

```python
In [8]: 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]
```

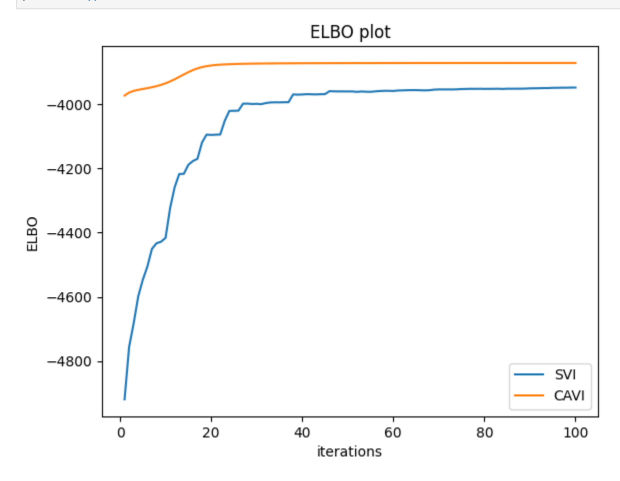
Evaluation

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.

```python
In [9]: 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, :]}")

print(f"Time SVI: {end_svi1 - start_svi1}")
print(f"Time CAVI: {end_cavi1 - start_cavi1}")
```

```
----- Recall label switching - compare E[theta] and true theta and check for label switching -----
E[theta] of doc 0 SVI:  [0.706 0.294]
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:    [0.160 0.130 0.294 0.342 0.074]
E[beta] SVI k=1:    [0.176 0.207 0.199 0.288 0.131]
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]
Time SVI: 0.4023935794830322
Time CAVI: 2.0533666610717773
```

```python
In [10]: plt.plot(list(range(1, n_iter_cavi1 + 1)), elbo1_svi[np.arange(0, n_iter_svi1, int(n_iter_svi1 / n_iter_cavi1))], label='SVI')
plt.plot(list(range(1, n_iter_cavi1 + 1)), elbo1_cavi, label='CAVI')
plt.title("ELBO plot")
plt.xlabel("iterations")
plt.ylabel("ELBO")
plt.legend()
plt.savefig('ELBO_convergence_tiny_dataset.png')
plt.show()
```




```python
In [11]: # Add your own code for evaluation here (will not be graded)
```

CASE 2

Small dataset

```python
In [12]: 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]
```

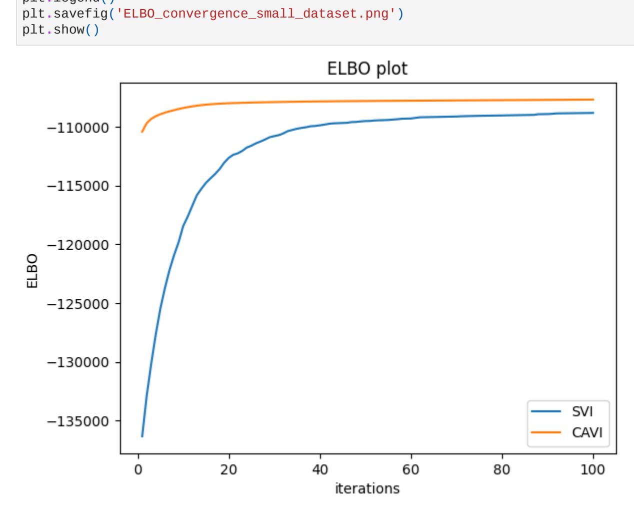
Evaluation

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.

```python
In [13]: 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:      [0.627 0.294 0.078]
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:    [0.079 0.087 0.072 0.211 0.033 0.025 0.029 0.072 0.282 0.111]
E[beta] k=1:    [0.186 0.131 0.062 0.133 0.018 0.063 0.009 0.246 0.079 0.074]
E[beta] k=2:    [0.232 0.074 0.070 0.069 0.010 0.083 0.027 0.236 0.115 0.084]
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: 10.909294843673706
Time CAVI: 68.34232449531555
```

```python
In [14]: plt.plot(list(range(1, n_iter_cavi2 + 1)), elbo2_svi[np.arange(0, n_iter_svi2, int(n_iter_svi2 / n_iter_cavi2))], label='SVI')
plt.plot(list(range(1, n_iter_cavi2 + 1)), elbo2_cavi, label='CAVI')
plt.title("ELBO plot")
plt.xlabel("iterations")
plt.ylabel("ELBO")
plt.legend()
plt.savefig('ELBO_convergence_small_dataset.png')
plt.show()
```




```python
In [15]: # Add your own code for evaluation here (will not be graded)
```

CASE 3

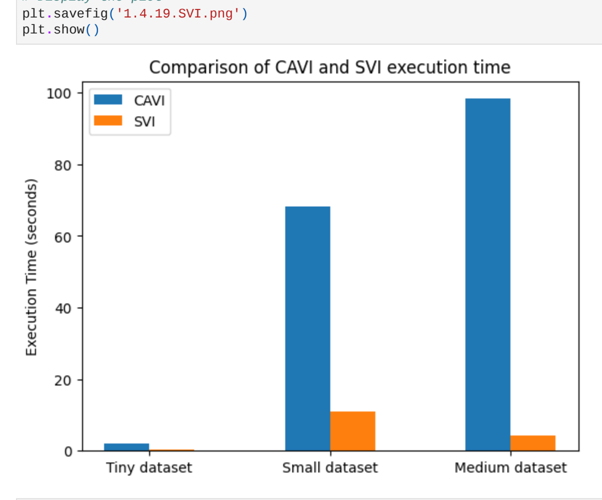
Medium small dataset, one iteration for time analysis.

```python
In [16]: 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
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]
```

```python
In [17]: 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_gamma3_svi[0] / np.sum(final_gamma3_svi[0], axis=0, keepdims=True)}")
print(f"E[theta] of doc 0 CAVI:     {final_gamma3_cavi[0] / np.sum(final_gamma3_cavi[0], axis=0, keepdims=True)}")
print(f"True theta of doc 0:        {theta3[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_lmbda3_svi[0, :] / np.sum(final_lmbda3_svi[0, :], axis=-1, keepdims=True)}")
print(f"E[beta] k=1:    {final_lmbda3_svi[1, :] / np.sum(final_lmbda3_svi[1, :], axis=-1, keepdims=True)}")
print(f"E[beta] k=2:    {final_lmbda3_svi[2, :] / np.sum(final_lmbda3_svi[2, :], axis=-1, keepdims=True)}")
print(f"True beta k=0:  {beta3[0, :]}")
print(f"True beta k=1:  {beta3[1, :]}")
print(f"True beta k=2:  {beta3[2, :]}")


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

```
----- Recall label switching - compare E[theta] and true theta and check for label switching -----
E[theta] of doc 0 SVI:      [0.107 0.286 0.214 0.179 0.214]
E[theta] of doc 0 CAVI:     [0.221 0.119 0.382 0.029 0.249]
True theta of doc 0:        [0.185 0.103 0.409 0.099 0.204]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
E[beta] k=0:    [0.242 0.029 0.144 0.046 0.153 0.136 0.023 0.022 0.111 0.094]
E[beta] k=1:    [0.238 0.118 0.020 0.042 0.171 0.072 0.038 0.090 0.134 0.076]
E[beta] k=2:    [0.150 0.156 0.117 0.188 0.049 0.122 0.036 0.032 0.018 0.133]
True beta k=0:  [0.401 0.002 0.055 0.100 0.047 0.093 0.017 0.065 0.025 0.196]
True beta k=1:  [0.460 0.040 0.189 0.014 0.076 0.061 0.072 0.013 0.027 0.049]
True beta k=2:  [0.074 0.320 0.004 0.073 0.027 0.091 0.035 0.161 0.171 0.044]

Examine per iteration run time.
Time SVI: 4.212661981582642
Time CAVI: 98.33035349845886
```

```python
In [18]: run_names = ['Tiny dataset', 'Small dataset', 'Medium dataset']
runtimes_1 = [end_cavi1 - start_cavi1, end_svi1 - start_svi1]
runtimes_2 = [end_cavi2 - start_cavi2, end_svi2 - start_svi2]
runtimes_3 = [end_cavi3 - start_cavi3, end_svi3 - start_svi3]

runtimes_CAVI = [end_cavi1 - start_cavi1, end_cavi2 - start_cavi2, end_cavi3 - start_cavi3]
runtimes_SVI = [end_svi1 - start_svi1, end_svi2 - start_svi2, end_svi3 - start_svi3]

bar_width = 0.25
index = np.arange(3)

plt.bar(index - bar_width/2, runtimes_CAVI, bar_width, label='CAVI')
plt.bar(index + bar_width/2, runtimes_SVI, bar_width, label='SVI')

plt.ylabel('Execution Time (seconds)')
plt.title('Comparison of CAVI and SVI execution time')
plt.xticks(index, run_names)
plt.legend()

# Display the plot
plt.savefig('1.4.19.SVI.png')
plt.show()
```




```python
In [18]:
```