DD2434 Machine Learning, Advanced Course
Assignment 1A

Philip Rettig, Wilhelm Öberg

2023-12-11

1 1A

1.1 Exponential Family

$$p(x|\theta) = h(x) \exp\left(\eta(\theta) \cdot T(x) - A(\eta)\right) \tag{1}$$

Question 1.1.1

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

1.1.1 Solution

$$p(x|\lambda) = \frac{1}{x!} \exp\left(\ln \lambda \cdot x - \exp \ln \lambda\right) \tag{2}$$

$$p(x|\lambda) = \frac{1}{x!} \exp\left(\ln \lambda^x - \lambda\right) \tag{3}$$

$$p(x|\lambda) = \frac{\lambda^x \exp\left(-\lambda\right)}{x!} \tag{4}$$

Equation 4 correspond to the Poisson distribution.

Question 1.1.2

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

1.1.2 Solution

$$p(x|\alpha, \beta) = 1 \cdot \exp\left([\alpha - 1, \beta] \cdot [\log x, x] - \log \Gamma(\alpha) - \alpha \log \beta\right) \tag{5}$$

$$p(x|\alpha, \beta) = \exp\left((\alpha - 1) \log x - \beta x + \log \beta^{\alpha} - \log \Gamma(\alpha)\right) \tag{6}$$

$$p(x|\alpha, \beta) = \exp\left(\log(x^{(\alpha-1)} \cdot \beta^{\alpha}) - \beta x - \log \Gamma(\alpha)\right) \tag{7}$$

$$p(x|\alpha, \beta) = \frac{x^{(\alpha-1)} \cdot e^{-\beta x} \cdot \beta^{\alpha}}{\Gamma(\alpha)} \tag{8}$$

Equation 8 correspond to the Gamma distribution.

Question 1.1.3

- $\boldsymbol{\theta} = [\mu, \sigma^2]$
- $\eta(\theta) = [\frac{\theta_1}{\theta_2}, -\frac{1}{2\theta_2}]$
- $h(x) = \frac{1}{\sqrt{2\pi}}$
- $T(x) = [x, x^2]$
- $A(\eta) = -\frac{n_1^2}{4n_2} - \frac{1}{2}\log(-2n_2)$

1.1.3 Solution

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}} \exp\Big(\Big[\frac{\theta_1}{\theta_2}, -\frac{1}{2\theta_2}\Big] \cdot [x, x^2] - (-\frac{n_1^2}{4n_2} - \frac{1}{2}\log(-2n_2))\Big) \tag{9}$$

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}} \exp\Big(\frac{\mu x}{\sigma^2} - \frac{x^2}{2\sigma^2} + \frac{2\sigma^2}{-4} + \frac{1}{2}\log\Big(\frac{1}{\sigma^2}\Big)\Big) \tag{10}$$

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}} \exp\Big(\frac{2\mu x - x^2}{2\sigma^2} - \frac{\mu^2}{2\sigma^2} + \frac{1}{2}\log\Big(\frac{1}{\sigma^2}\Big)\Big) \tag{11}$$

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}} \exp\Big(\frac{-(x^2 - 2\mu x + \mu^2)}{2\sigma^2} + \log\Big(\frac{1}{\sigma}\Big)\Big) \tag{12}$$

$$p(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(-\frac{1}{2}\Big(\frac{x-\mu}{\sigma}\Big)^2\Big) \tag{13}$$

Equation 13 correspond to the Normal distribution.

Question 1.1.4

- $\theta = \lambda$
- $\eta(\theta) = -\theta$
- $h(x) = 2$
- $T(x) = x$
- $A(\eta) = -\log(-\eta/2)$

1.1.4 Solution

$$p(x|\lambda) = 2\exp\left(-\lambda x + \log(\lambda/2)\right) \tag{14}$$

$$p(x|\lambda) = 2\exp\left(-\lambda x + \log\lambda - \log 2\right) \tag{15}$$

$$p(x|\lambda) = \frac{2e^{-\lambda x}\lambda}{2} \tag{16}$$

$$p(x|\lambda) = \lambda e^{-\lambda x} \tag{17}$$

Equation 17 correspond to the Exponential distribution.

Question 1.1.5

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

1.1.5 Solution

$$\begin{aligned} p(x|\psi_1, \psi_2) =& 1 \cdot \exp\left([\psi_1 - 1, \psi_2 - 1] \cdot [\log x, \log(1-x)]\right. \\ & \left. - (\log\Gamma(\psi_1) + \log\Gamma(\psi_2) - (\log\Gamma(\psi_1 + \psi_2)))\right) \end{aligned} \tag{18}$$

$$\begin{aligned} p(x|\psi_1, \psi_2) =& \exp\left((\psi_1 - 1)\log x + (\psi_2 - 1)\log(1-x)\right. \\ & \left. + \log\Gamma(\psi_1 + \psi_2) - (\log\Gamma(\psi_1) + \log\Gamma(\psi_2))\right) \end{aligned} \tag{19}$$

$$p(x|\psi_1, \psi_2) = \exp\left(\log x^{\psi_1 - 1} + \log(1-x)^{\psi_2 - 1} + \log\Big(\frac{\Gamma(\psi_1 + \psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)}\Big)\right) \tag{20}$$

$$p(x|\psi_1, \psi_2) = x^{\psi_1 - 1}(1-x)^{\psi_2 - 1} \cdot \Big(\frac{\Gamma(\psi_1 + \psi_2)}{\Gamma(\psi_1)\Gamma(\psi_2)}\Big) \tag{21}$$

Equation 21 correspond to the Beta distribution.

1.2 Dependencies in a DGM

Question 1.2.6: In figure 1

- $W_{d,n} \perp W_{d,n+1} | \Lambda_d, \beta_{1:K}$?

Answer: Yes

Question 1.2.7: In figure 1

- $\theta_d \perp \theta_{d+1} | Z_{d,1:N}$?

Answer: No

Question 1.2.8: In figure 1

- $\theta_d \perp \theta_{d+1} | \alpha, Z_{1:D,1:N}$?

Answer: Yes

Question 1.2.9: In figure 2

- $W_{d,n} \perp W_{d,n+1} | \Lambda_d, \beta_{1:K}$?

Answer: No

Question 1.2.10: In figure 2

- $\theta_d \perp \theta_{d+1} | Z_{d,1:N}, Z_{d+1,1:N}$?

Answer: No

Question 1.2.11: In figure 2

- $\Lambda_d \perp \Lambda_{d+1} | \phi, Z_{1:D,1:N}$?

Answer: No

1.3 CAVI

Question 1.3.12:

Solution:

```
import numpy as np
def generate_data(mu, tau, N):
    D = np.random.normal(loc = mu, scale = (1/tau), size = N)
    return D
```

The histograms for each dataset can be found in the jupyter notebook 1A-3-CAVI.ipynb.

Question 1.2.13:

Solution:

```
def ML_est(data):
    mu_ml = np.mean(data)
    tau_ml = (1/(np.var(data)))
    return mu_ml, tau_ml
```

The ML estimates can be found in the jupyter notebook.

Question 1.2.14:

Firstly, Bayes' rule is used to obtain the posterior distribution:

$$posterior = \frac{likelihood \times prior}{evidence}$$

Calculating the evidence, $p(X) = \iint p(X, \mu, \tau)\, d\mu\, d\tau$, is analytically intractable in many cases. Therefore the posterior is said to be proportional to the likelihood and priors. Where the joint distribution equals the posterior distribution up to a normalizing constant.

$$posterior \propto likelihood \times prior$$

Secondly, express and substitute the joint distribution as likelihood and priors.

$$p(X, \mu, \tau) = p(X|\mu, \tau) \cdot p(\mu, \tau) \cdot p(\tau) \tag{22}$$

$$p(X|\mu, \tau) = \prod_{n=1}^{N} \sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2}(x_n - \mu)^2\right) \qquad \text{, where } N = |X| \tag{23}$$

$$p(\mu|\tau) = \sqrt{\frac{\lambda_0 \tau}{2\pi}} \exp\left(-\frac{\lambda_0 \tau}{2}(\mu - \mu_0)^2\right) \tag{24}$$

$$p(\tau) = \frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\tau^{\alpha_0-1}\exp\left(-\beta_0\tau\right) \tag{25}$$

Equation 23 is the likelihood function, 24 is the μ prior and 25 is the τ prior.

$$\begin{aligned}
p(\mu,\tau|X) &\stackrel{\pm}{=} p(X|\mu,\tau)\cdot p(\mu,\tau)\cdot p(\tau) \qquad (26)\\
\log p(\mu,\tau|X) &\stackrel{\pm}{=} \log p(X|\mu,\tau) + \log p(\mu,\tau) + \log p(\tau)\\
&\stackrel{\pm}{=} \sum_{n=1}^{N}\log\Big(\sqrt{\frac{\tau}{2\pi}}\exp\left(\frac{-\tau(x_n-\mu)^2}{2}\right)\Big) + \log\Big(\sqrt{\frac{\lambda_0\tau}{2\pi}}\exp\left(\frac{-\lambda_0\tau(\mu-\mu_0)^2}{2}\right)\Big)+\\
&+\log\Big(\frac{\beta_0^{\alpha_0}}{\Gamma(\alpha_0)}\tau^{\alpha_0-1}\exp\left(-\beta_0\tau\right)\Big)\\
&\stackrel{\pm}{=} \sum_{n=1}^{N}\log\frac{1}{2}\log\tau - \frac{\tau}{2}\log(x_n-\mu)^2 + \alpha_0\log\tau - \frac{\lambda_0\tau}{2}(\mu-\mu_0)^2 - \beta_0\tau\\
&\stackrel{\pm}{=} \frac{N}{2}\log\tau - \frac{\tau}{2}\sum_{n=1}^{N}(x_n^2 - 2x\mu + \mu^2) + \alpha_0\log\tau - \beta_0\tau - \frac{\lambda_0\tau}{2}(\mu^2 - 2\mu\mu_0 + \mu_0^2)\\
&\stackrel{\pm}{=} \Big(\frac{N}{2}+\alpha_0\Big)\log\tau - \beta_0\tau - \frac{\tau}{2}(N+\lambda_0)\Big(\mu^2 - \frac{2\mu}{N+\lambda_0}\sum_{n=1}^{N}(x_n+\mu_0\lambda_0)\\
&+\frac{\sum_{x=1}^{N}x_n^2+\lambda_0\mu_0^2}{N+\lambda_0}\Big)\\
&\stackrel{\pm}{=} \Big(\frac{N}{2}+\alpha_0\Big)\log\tau - \tau(\beta_0 + \frac{1}{2}\Big(\sum_{n=1}^{N}x_n^2 + \lambda_0\mu_0^2 - \frac{\Big(\sum_{n=1}^{N}x_n+\mu_0\lambda_0\Big)^2}{N+\lambda_0}\Big)\\
&-\frac{\tau}{2}(N+\lambda_0)\Big(\mu - \frac{\Big(\sum_{n=1}^{N}x_n+\mu_0\lambda_0\Big)^2}{N+\lambda_0}\Big)
\end{aligned}$$

$$\begin{aligned}
p(\mu,\tau|X) \stackrel{\pm}{=} \exp\Bigg(&\Big(\frac{N}{2}+\alpha_0\Big)\log\tau - \tau\Big(\beta_0 + \frac{1}{2}\Big(\sum_{n=1}^{N}x_n^2+\lambda_0\mu_0^2 - \frac{\Big(\sum_{n=1}^{N}x_n+\mu_0\lambda_0\Big)^2}{N+\lambda_0}\Big)\\
&-\frac{\tau}{2}(N+\lambda_0)\Big(\mu - \frac{\Big(\sum_{n=1}^{N}x_n+\mu_0\lambda_0\Big)^2}{N+\lambda_0}\Big)\Big)\Bigg)
\end{aligned} \tag{27}$$

Equation 27 is the exact posterior distribution of the model up to a normalizing constant.

Question 1.2.15:

See 1A-3-CAVI.ipynb for CAVI implementation and contour plots of the inferred variational distribution and the exact distribution.

There is an issue with the size of the contors of the posteriors. Where a larger number of observations cause a smaller contour plot of the posterior. We believe this issue is caused by the lack of a normalization factor. By looking at the plots describing parameter values, it is clear that parameters sclale linearly with the length of the data vector, causing this behaviour.

However, in all cases the inferred posterior align well with the exact posterior. The maximum likelihood estimates increase in precision when a larger dataset is used.

1.4 SVI

1.4.1 Definition of local hidden variables

Local hidden variable typically refers to unobserved variables associated with each document. In the case of our model we get the following:

$$p(w, Z, \beta, \theta|\alpha, \eta) = \prod_{k=1}^{K} p(\beta_k|\eta)\Big(\prod_{d=1}^{D} p(\theta_d|\alpha) \prod_{n=1}^{N} p(Z_{n,d}|\theta_d)p(w_{n,d}|Z_{n,d}, \beta_{1:k})\Big) \tag{28}$$

1.4.2 Global and local hidden

Local hidden: $Z_{d,n}, \theta_d$
Global hidden: β_k

1.4.3 ELBO

$$
\begin{aligned}
L(\gamma,\phi;\alpha,\beta) = {} & \underbrace{\log\Gamma\left(\textstyle\sum_{j=1}^{k}\alpha_j\right) - \sum_{i=1}^{k}\log\Gamma(\alpha_i) + \sum_{i=1}^{k}(\alpha_i - 1)\left(\Psi(\gamma_i) - \Psi\left(\textstyle\sum_{j=1}^{k}\gamma_j\right)\right)}_{E_q[\log p(\theta\,|\,\alpha)] \text{ where } \theta\sim\mathrm{Dir}(\alpha)} \\
& + \underbrace{\sum_{n=1}^{N}\sum_{i=1}^{k}\phi_{ni}\left(\Psi(\gamma_i) - \Psi\left(\textstyle\sum_{j=1}^{k}\gamma_j\right)\right)}_{E_q[\log p(\mathbf{z}\,|\,\theta)] \text{ where } z_n\sim\mathrm{Multinomial}(\theta)} \\
& + \underbrace{\sum_{n=1}^{N}\sum_{i=1}^{k}\sum_{j=1}^{V}\phi_{ni}w_n^j\log\beta_{ij}}_{E_q[\log p(\mathbf{w}\,|\,\mathbf{z},\beta)]} \\
& \underbrace{-\log\Gamma\left(\textstyle\sum_{j=1}^{k}\gamma_j\right) + \sum_{i=1}^{k}\log\Gamma(\gamma_i) - \sum_{i=1}^{k}(\gamma_i - 1)\left(\Psi(\gamma_i) - \Psi\left(\textstyle\sum_{j=1}^{k}\gamma_j\right)\right)}_{E_q[\log q(\theta)]} \\
& \underbrace{-\sum_{n=1}^{N}\sum_{i=1}^{k}\phi_{ni}\log\phi_{ni},}_{E_q[\log q(\mathbf{z})]}
\end{aligned}
$$

Modified from source

where Ψ is

$$\psi(x) = \frac{d}{dx}\ln\big(\Gamma(x)\big) = \frac{\Gamma'(x)}{\Gamma(x)}$$

Figur 1: Source: https://jonathan-hui.medium.com/machine-learning-latent-dirichlet-allocation-lda-1d9d148f13a4

1.4.4 Implemented SVI algorithm

The SVI perform slightly worse than CAVI with respect to time and ELBO calculations, taking longer time and returning a more unstable and lower ELBO.

See jupyter notebook LDA-SVI.ipynb for the implementation.

1.5 BBVI

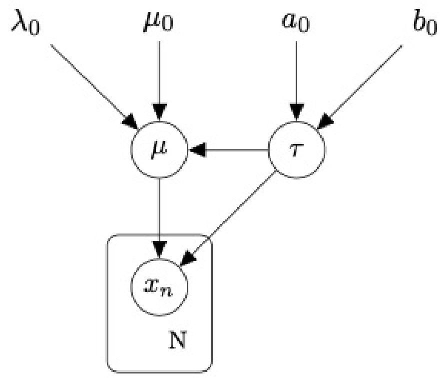
1.5.1 Simple model

1.5.2 Control Variates

Control variates in BBVI are used to reduce the variance of gradient estimates, improving the efficiency and stability of the optimization process by incorporating auxiliary terms based on known expectations.

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:

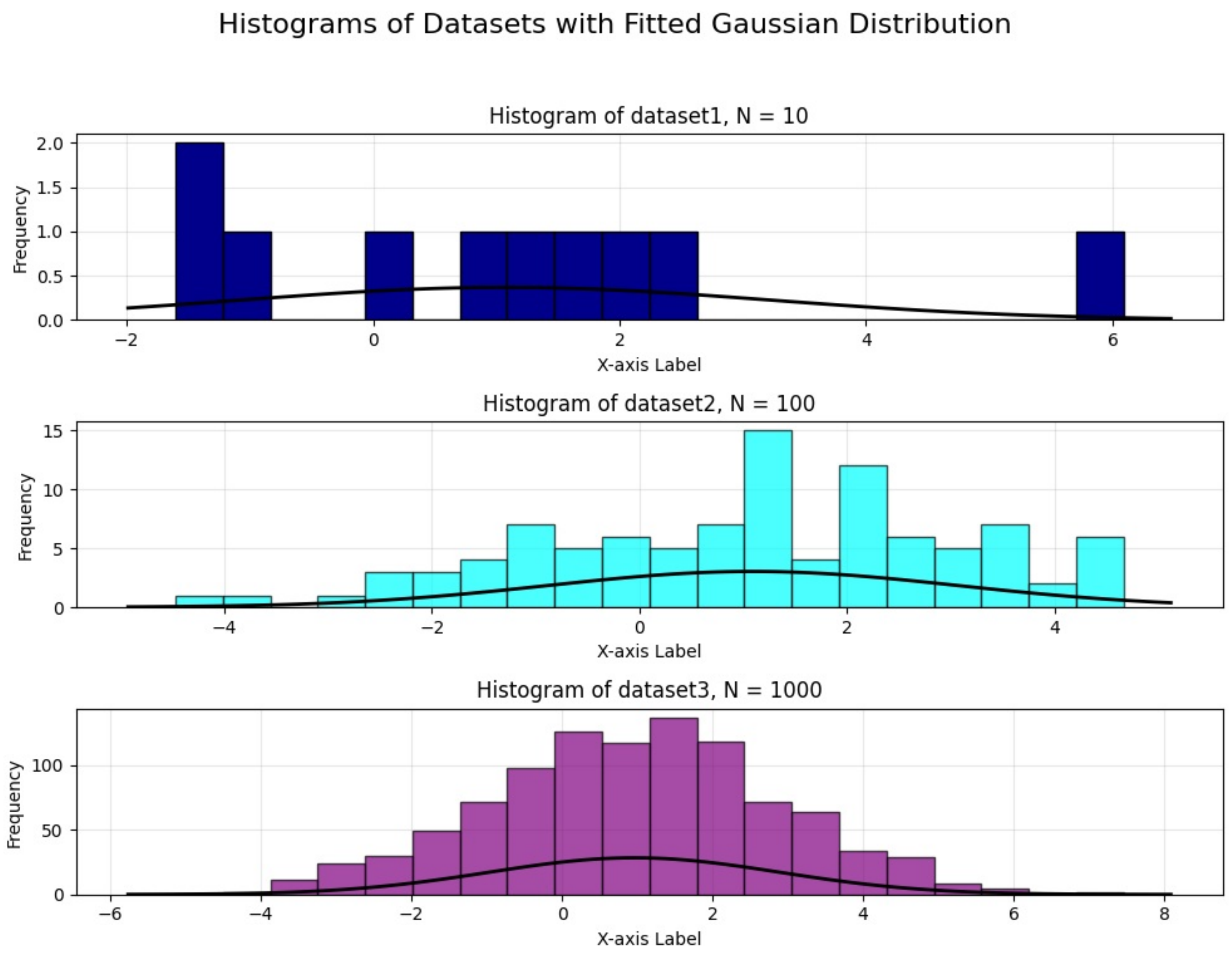
Implement a function that generates data points for the given model.

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm, gamma
```

In [2]:
```
def generate_data(mu, tau, N):
    D = np.random.normal(loc = mu, scale = (1/tau), size = 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)

datasets = [dataset_1,dataset_2,dataset_3]

# Visualize the datasets via histograms
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(10, 8))

# Plot histograms and add line along the distribution
for i, dataset in enumerate([dataset_1, dataset_2, dataset_3]):
    bins = 20
    color = ['darkblue', 'cyan', 'purple'][i]
    alpha = [1.0, 0.7, 0.7][i]

    # Histogram
    n, bins, _ = axes[i].hist(dataset, bins=bins, color=color, alpha=alpha, edgecolor='black', zorder=2)
    axes[i].set_title(f'Histogram of dataset{i+1}, N = {len(dataset)}')
    axes[i].set_xlabel('X-axis Label')
    axes[i].set_ylabel('Frequency')

    # Fit a normal distribution to the data
    mu_fit, std_fit = norm.fit(dataset)

    # Plot the fitted Gaussian distribution
    xmin, xmax = axes[i].get_xlim()
    x = np.linspace(xmin, xmax, 100)
    p = norm.pdf(x, mu_fit, std_fit) * np.max(n)
    axes[i].plot(x, p, 'k', linewidth=2, zorder=3)

    # Set grid below histograms
    axes[i].grid(True, alpha=0.3, zorder=1)
    axes[i].set_axisbelow(True)

# Overall title
plt.suptitle('Histograms of Datasets with Fitted Gaussian Distribution', fontsize=16)

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Question 1.3.13:

Find ML estimates of the variables μ and τ

In [4]:
```
def ML_est(data):
    mu_ml = np.mean(data)
    tau_ml = (1/(np.var(data)))
    return mu_ml, tau_ml
```

In [5]:
```
# ML estimates for the datasets
for dataset in datasets:
    mu, tau = ML_est(dataset)
    print(f"mu = {mu}, tau = {tau}, N = {len(dataset)}")
```

```
mu = 1.073714690563369, tau = 0.21339503539023952, N = 10
mu = 1.0921865141266263, tau = 0.25656863810597763, N = 100
mu = 0.9607410652001522, tau = 0.26923588619543337, N = 1000
```

In [6]:
```
# prior parameters
mu_0 = 0
lambda_0 = 1
a_0 = 1
b_0 = 1
```

Continue with a helper function that computes ELBO:

In [7]:
```
from scipy.special import digamma, gamma, gammaln
def compute_elbo(D, a_0, b_0, mu_0, lambda_0, a_N , b_N, mu_N, lambda_N):
    # expected values
    E_tao = a_N / b_N
    E_log_tao = np.log(b_N) - digamma(a_N)
    E_mu = mu_N
    mse = np.sum((D-np.mean(D))**2)
    E_mu_sq = mu_N**2

    # pdfs
    E_log_p_x_mu_tao = (1/lambda_0) - (1/2 * E_tao * np.sum((D**2)-D*2*E_mu+E_mu_sq)) + (len(D)*E_log_tao/2) -
    E_log_p_mu_tao = 0.5*(np.log(lambda_0)+digamma(a_N)-np.log(b_N)-np.log(2*np.pi) -lambda_0*((np.mean(D)**2)+
    E_log_p_tao = a_0*np.log(b_0)-gammaln(a_0)+(a_0-1)*(digamma(a_N)-np.log(b_N))-((b_0*a_N)/(b_N))
```

```
    # entropies
    E_log_q_mu = 1/2 * np.log((2*np.pi/(E_tao*lambda_N+len(D))))
    E_log_q_tao = a_N-np.log(b_N)+gammaln(a_N)+(1-a_N)*digamma(a_N)

    elbo = E_log_p_x_mu_tao + E_log_p_mu_tao + E_log_p_tao + E_log_q_mu + E_log_q_tao
    return elbo
```

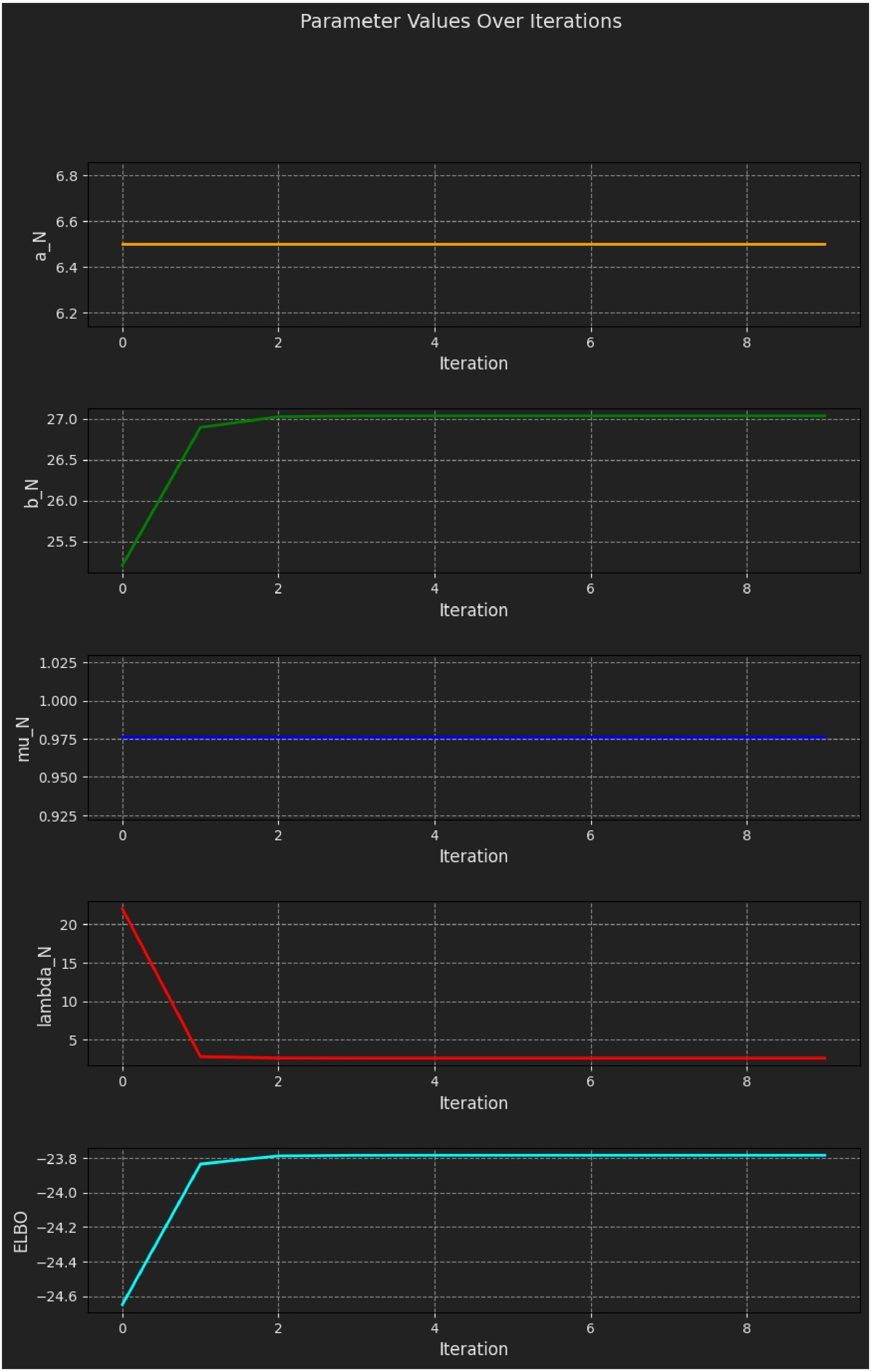
Now, implement the CAVI algorithm:

In [8]:
```
def CAVI(D, a_0, b_0, mu_0, lambda_0):
    # make an initial guess for the expected value of tau
    initial_guess_exp_tau = 2
    iterations = 10
    x_bar = np.mean(D)
    N = len(D)

    # initialize res arrays
    elbos = np.zeros(iterations)
    mu_N = np.zeros(iterations)
    lambda_N = np.zeros(iterations)
    a_N = np.zeros(iterations)
    b_N = np.zeros(iterations)

    # CAVI iterations
    for i in range(iterations):

        # update parameters
        a_N[i] = a_0 + ((N + 1)/2)
        mu_N[i] = ((lambda_0*mu_0) + (N*x_bar))/(lambda_0 + N)

        # deal with lambdas and bs circular dependency
        if i == 0:
            lambda_N[i] = (lambda_0+N)*initial_guess_exp_tau
        else:
            lambda_N[i] = (lambda_0+N)*(a_N[i]/b_N[i-1])

        b_N[i] = b_0 + (1/2) * ((lambda_0 + N) * (1/lambda_N[i] + mu_N[i]**2) - 2 * ((lambda_0 * mu_0 + np.sum(I

        # save the elbo values
        elbos[i] = compute_elbo(D, a_0, b_0, mu_0, lambda_0, a_N[i], b_N[i], mu_N[i], lambda_N[i])

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

In [9]:
```
def plot_parameters(a_N, b_N, mu_N, lambda_N, elbos):
    for param in ['figure.facecolor', 'axes.facecolor', 'savefig.facecolor']:
        plt.rcParams[param] = '#222222'
    for param in ['text.color', 'axes.labelcolor', 'xtick.color', 'ytick.color']:
        plt.rcParams[param] = '0.9'

    fig, axes = plt.subplots(5, 1, figsize=(10, 15))

    # Plot a_N
    axes[0].plot(a_N, color='orange', linewidth=2)
    axes[0].set_ylabel('a_N', fontsize=12)
    axes[0].set_xlabel('Iteration', fontsize=12)
    axes[0].grid(True, linestyle='--', alpha=0.7)

    # Plot b_N
    axes[1].plot(b_N, color='green', linewidth=2)
    axes[1].set_ylabel('b_N', fontsize=12)
    axes[1].set_xlabel('Iteration', fontsize=12)
    axes[1].grid(True, linestyle='--', alpha=0.7)

    # Plot mu_N
    axes[2].plot(mu_N, color='blue', linewidth=2)
    axes[2].set_ylabel('mu_N', fontsize=12)
    axes[2].set_xlabel('Iteration', fontsize=12)
    axes[2].grid(True, linestyle='--', alpha=0.7)

    # Plot lambda_N
    axes[3].plot(lambda_N, color='red', linewidth=2)
    axes[3].set_ylabel('lambda_N', fontsize=12)
    axes[3].set_xlabel('Iteration', fontsize=12)
    axes[3].grid(True, linestyle='--', alpha=0.7)

    # Plot ELBO
    axes[4].plot(elbos, color='cyan', linewidth=2)
    axes[4].set_ylabel('ELBO', fontsize=12)
    axes[4].set_xlabel('Iteration', fontsize=12)
    axes[4].grid(True, linestyle='--', alpha=0.7)

    # Adjust spacing between subplots
    plt.subplots_adjust(hspace=0.5)
```

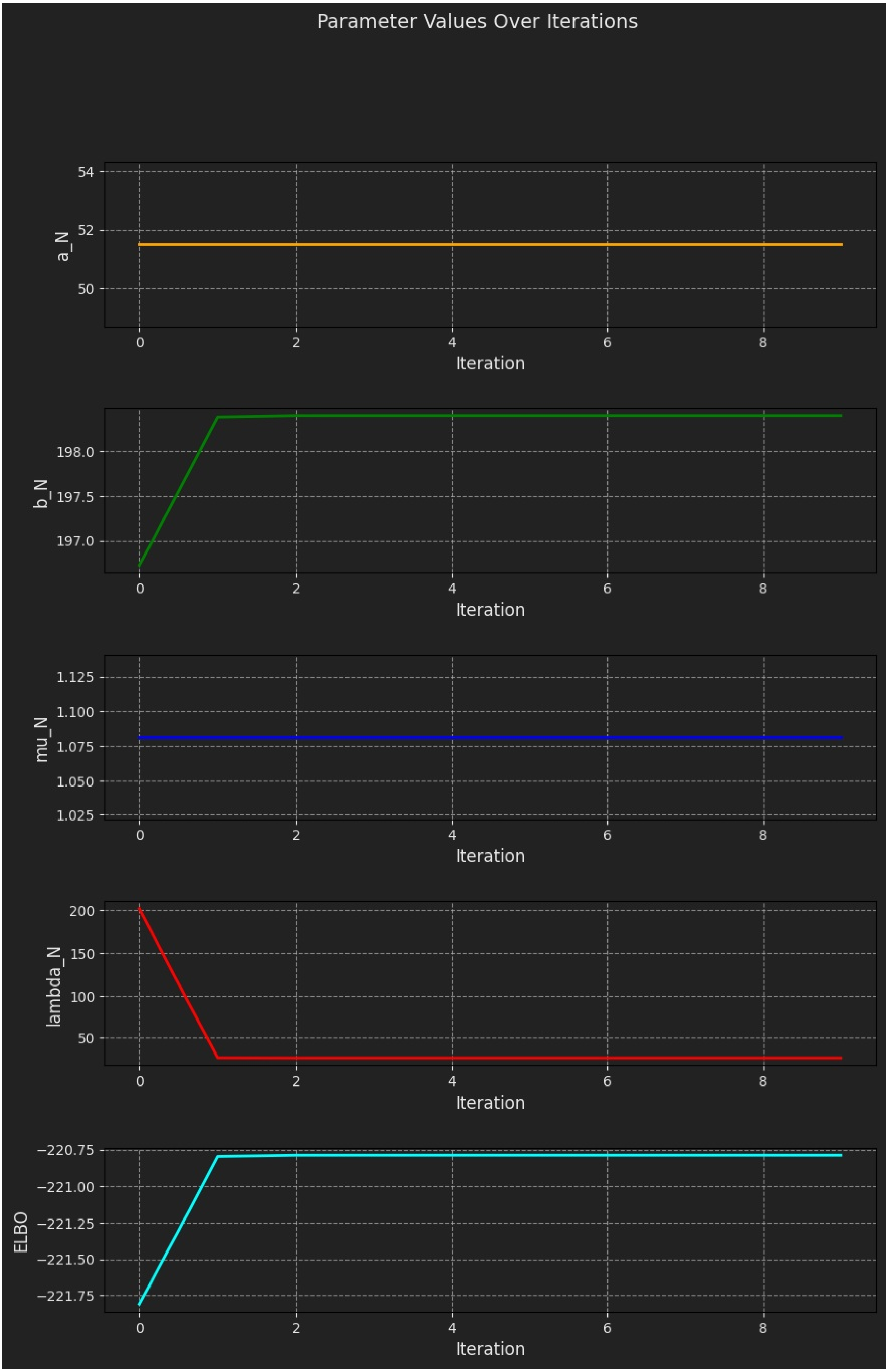
```python
    plt.suptitle('Parameter Values Over Iterations', fontsize=14)
    plt.show()
```

In [10]:
```python
a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset_1, a_0, b_0, mu_0, lambda_0)
plot_parameters(a_N, b_N, mu_N, lambda_N, elbos)
```



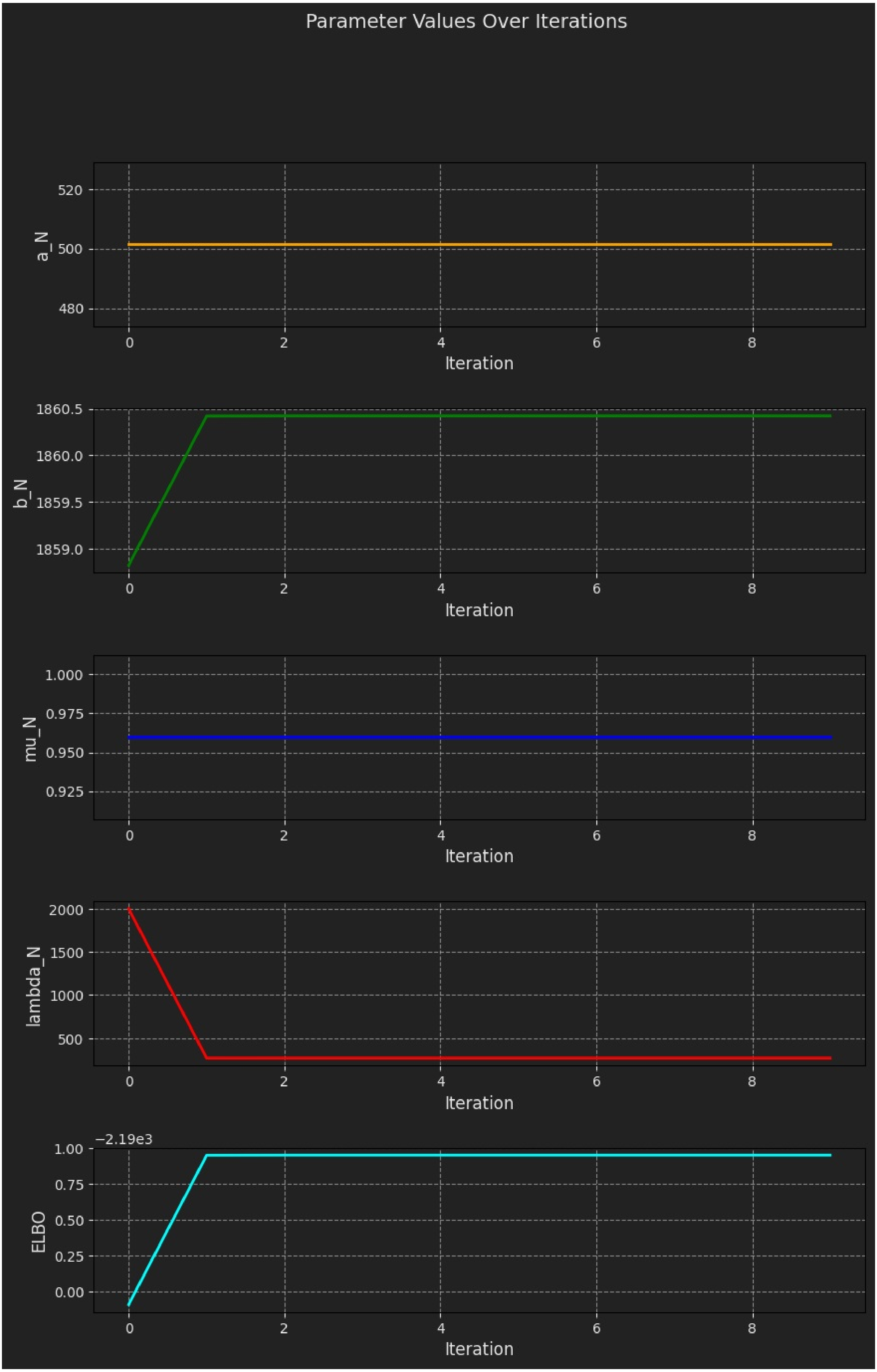
In [11]:

```
a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset_2, a_0, b_0, mu_0, lambda_0)
plot_parameters(a_N, b_N, mu_N, lambda_N, elbos)
```



In [12]:

```
a_N, b_N, mu_N, lambda_N, elbos = CAVI(dataset_3, a_0, b_0, mu_0, lambda_0)
plot_parameters(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:

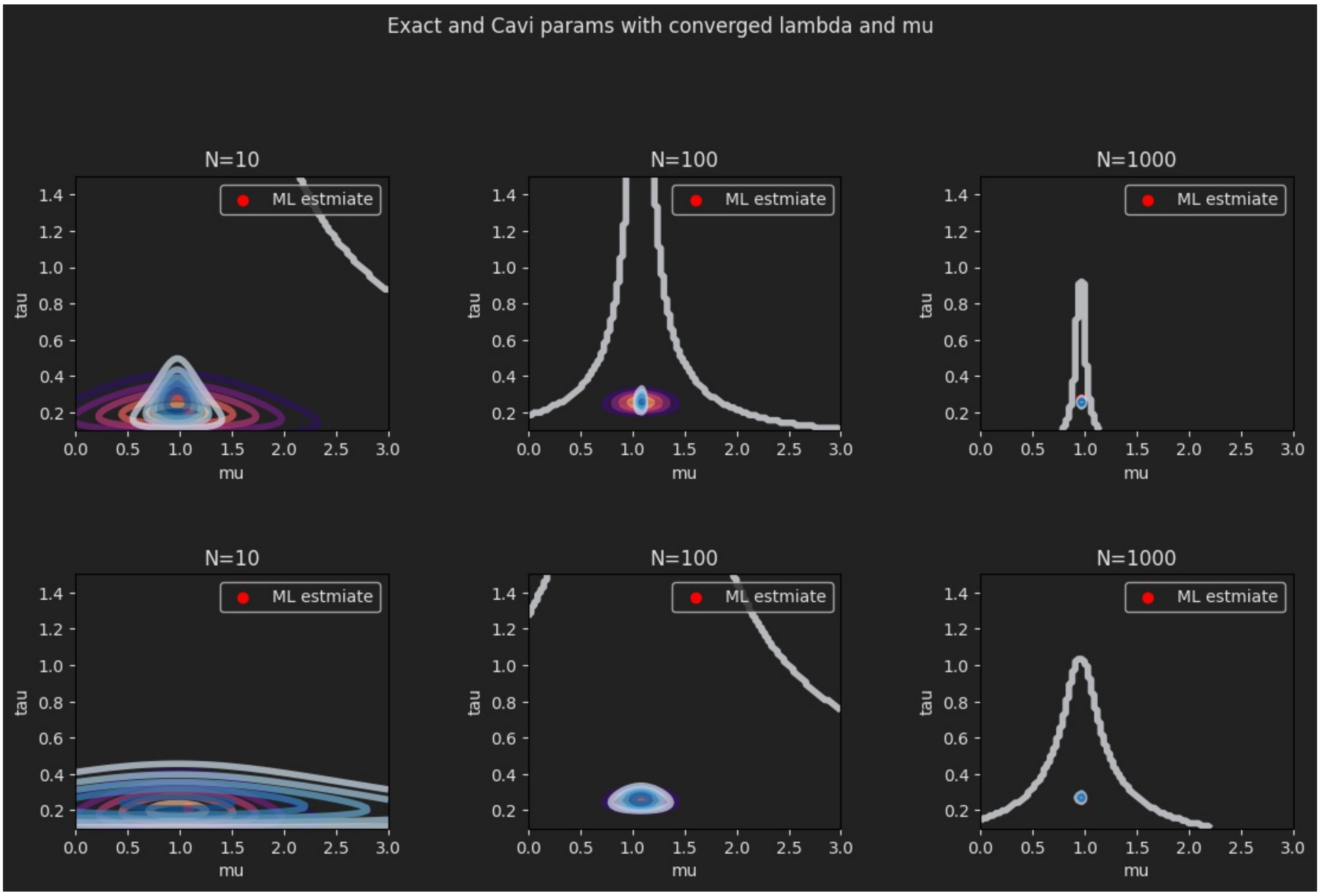
In [13]:
```
import scipy.stats as stats
def compute_exact_posterior(mu, tau, D, a_0, b_0, mu_0, lambda_0):
    # Calculate updated parameters
    exact_a = (len(D) / 2) + a_0
    exact_b = (
        b_0
        + 0.5 * (np.sum(D**2)
        + (lambda_0 * mu_0**2)
        - (((np.sum(D) + (mu_0 * lambda_0))**2) / (len(D) + lambda_0)))
    )
    exact_lambda = lambda_0 + len(D)
    exact_mu = (np.sum(D) + mu_0 * lambda_0) / (len(D) + lambda_0)

    exact_posterior = np.exp((exact_a-1)*np.log(tau) - tau*(exact_b)-(tau/2)*(exact_lambda*(mu-exact_mu)**2))
    return exact_posterior
```

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 [14]:
```
# get maximum likelihood estimates
ml_mu1, ml_tau1 = ML_est(dataset_1)
ml_mu2, ml_tau2 = ML_est(dataset_2)
ml_mu3, ml_tau3 = ML_est(dataset_3)

ML = [(ML_est(x)) for x in datasets]

# creating a grid for plotting
mu_values = np.linspace(0,3, 100)
tau_values = np.linspace(0.1,1.5, 100)
mu_grid, tau_grid = np.meshgrid(mu_values, tau_values)
pos = np.dstack((mu_grid, tau_grid))

# exact posterior for datasets 1-3
ep1 = compute_exact_posterior(pos[:,:,0], pos[:,:,1], dataset_1, a_0, b_0, mu_0, lambda_0)
ep2 = compute_exact_posterior(pos[:,:,0], pos[:,:,1], dataset_2, a_0, b_0, mu_0, lambda_0)
ep3 = compute_exact_posterior(pos[:,:,0], pos[:,:,1], dataset_3, a_0, b_0, mu_0, lambda_0)

exact_posteriors = [ep1,ep2,ep3]

# inferred variational parameters for dataset 1
cavi = CAVI(dataset_1, a_0, b_0, mu_0, lambda_0)
# prior parameters
aN = a_0
bN = b_0
muN = mu_0
lambdaN = lambda_0

# converged
aN = cavi[0][-1]
bN = cavi[1][-1]
muN = cavi[2][-1]
lambdaN = cavi[3][-1]

# inferred variational parameters for dataset 2
cavi2 = CAVI(dataset_2, a_0, b_0, mu_0, lambda_0)
aN2 = cavi2[0][-1]
bN2 = cavi2[1][-1]
muN2 = cavi2[2][-1]
lambdaN2 = cavi2[3][-1]

# inferred variational parameters for dataset 3
cavi3 = CAVI(dataset_3, a_0, b_0, mu_0, lambda_0)
aN3 = cavi3[0][-1]
bN3 = cavi3[1][-1]
muN3 = cavi3[2][-1]
lambdaN3 = cavi3[3][-1]

# cavi posterior for prior parameters for datasets 1-3
q_mu_tau1_prior = stats.norm.pdf(pos[:,:,0], loc=muN, scale=1.0 / (cavi[3][0]*pos[:,:,1])) * stats.gamma.pdf(po
q_mu_tau2_prior = stats.norm.pdf(pos[:,:,0], loc=muN2, scale=1.0 / (cavi2[3][0]*pos[:,:,1])) * stats.gamma.pdf(
q_mu_tau3_prior = stats.norm.pdf(pos[:,:,0], loc=muN3, scale=1.0 / (cavi3[3][0]*pos[:,:,1])) * stats.gamma.pdf(

# cavi posterior for converged parameters for datasets 1-3
q_mu_tau1 = stats.norm.pdf(pos[:,:,0], loc=muN, scale=1.0 / (lambdaN*pos[:,:,1])) * stats.gamma.pdf(pos[:,:,1],
q_mu_tau2 = stats.norm.pdf(pos[:,:,0], loc=muN2, scale=1.0 / (lambdaN2*pos[:,:,1])) * stats.gamma.pdf(pos[:,:,1
q_mu_tau3 = stats.norm.pdf(pos[:,:,0], loc=muN3, scale=1.0 / (lambdaN3*pos[:,:,1])) * stats.gamma.pdf(pos[:,:,1
```

In [15]:
```
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(12,8))
ax1 = axes[0,0]
ax2 = axes[0,1]
ax3 = axes[0,2]
ax4 = axes[1,0]
ax5 = axes[1,1]
ax6 = axes[1,2]

ax1.contour(mu_grid, tau_grid, ep1, levels=6, cmap='magma', alpha=0.7, linewidths=4.)
ax1.contour(mu_grid, tau_grid, q_mu_tau1_prior, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax1.scatter(ml_mu1,ml_tau1,color="red", label="ML estmiate")

ax2.contour(mu_grid, tau_grid, ep2, levels=5, cmap='magma', alpha=0.7, linewidths=4.)
ax2.contour(mu_grid, tau_grid, q_mu_tau2_prior, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax2.scatter(ml_mu2,ml_tau2, color="red", label="ML estmiate")

ax3.contour(mu_grid, tau_grid, ep3, levels=5, cmap='magma', alpha=0.7, linewidths=4.)
ax3.contour(mu_grid, tau_grid, q_mu_tau3_prior, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax3.scatter(ml_mu3,ml_tau3, color="red", label="ML estmiate")

ax4.contour(mu_grid, tau_grid, ep1, levels=6, cmap='magma', alpha=0.7, linewidths=4.)
ax4.contour(mu_grid, tau_grid, q_mu_tau1, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax4.scatter(ml_mu1,ml_tau1,color="red", label="ML estmiate")

ax5.contour(mu_grid, tau_grid, ep2, levels=5, cmap='magma', alpha=0.7, linewidths=4.)
ax5.contour(mu_grid, tau_grid, q_mu_tau2, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax5.scatter(ml_mu2,ml_tau2, color="red", label="ML estmiate")

ax6.contour(mu_grid, tau_grid, ep3, levels=5, cmap='magma', alpha=0.7, linewidths=4.)
ax6.contour(mu_grid, tau_grid, q_mu_tau3, levels=5, cmap='Blues', alpha=0.7, linewidths=4.)
ax6.scatter(ml_mu3,ml_tau3, color="red", label="ML estmiate")

ax1.legend()
ax2.legend()
ax3.legend()
ax4.legend()
ax5.legend()
ax6.legend()
fig.suptitle("Exact and Cavi params with converged lambda and mu")
ax1.set_title("N=10")
ax2.set_title("N=100")
ax3.set_title("N=1000")
ax4.set_title("N=10")
ax5.set_title("N=100")
ax6.set_title("N=1000")


ax1.set_xlabel('mu')
ax1.set_ylabel('tau')
ax2.set_xlabel('mu')
ax2.set_ylabel('tau')
ax3.set_xlabel('mu')
ax3.set_ylabel('tau')
ax4.set_xlabel('mu')
ax4.set_ylabel('tau')
ax5.set_xlabel('mu')
ax5.set_ylabel('tau')
ax6.set_xlabel('mu')
ax6.set_ylabel('tau')
fig.tight_layout(pad=4.0)
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

In [1]:
```python
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 [2]:
```python
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
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: [0.66368254 0.33631746]
 Mean z of doc 0: [0.7 0.3]
Beta of topic 0: [0.13024721 0.48180651 0.19941248 0.         0.1885338 ]
Beta of topic 1: [0.12208308 0.64327644 0.16446489 0.         0.07017558]
Word to topic assignment, z, of document 0: [0 1 1 0 0 0 1 1 0 0]
Observed words, w, of document 0: [1 1 1 2 4 0 1 2 4 4]
Unique words and count of document 0: ['0: 6', '1: 24', '2: 12', '4: 8']
```

In [7]:
```python
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 [8]:
```
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 [33]:
```
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]
```

```
[[8 6 7 4 1]
 [1 3 8 5 6]]
```

In [10]:

```
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=Tru
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), p
print(f"Final E[beta] k=1: {np.round(final_lmbda0[1, :] / np.sum(final_lmbda0[1, :], axis=-1, keepdims=True), p
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:  [0.38 0.62]
True theta of doc 0:          [0.664 0.336]
----- Recall label switching - e.g. E[beta_0] could be fit to true theta_1. -----
Final E[beta] k=0: [0.102 0.793 0.101 0.    0.003]
Final E[beta] k=1: [0.151 0.3   0.273 0.    0.276]
True beta k=0: [0.13  0.482 0.199 0.    0.189]
True beta k=1: [0.122 0.643 0.164 0.    0.07 ]
```

SVI Implementation

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

In [154…

```
def update_q_Z_svi(batch, w, gamma, lmbda):
    """
    TODO: rewrite to SVI update
    """
    samples = w[batch]
    D, N, W = samples.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 = samples.argmax(axis=-1)
    for d in range(len(samples)):
        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_svi(batch, phi, alpha):
    """
    TODO: rewrite to SVI update
    """
    E_Z = phi
    D, N, K = phi.shape
    gamma = np.ones((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_svi(batch, w, phi, eta):
    """
    TODO: rewrite to SVI update
    """
    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 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))

  step_size = [1/(x**0.5) for x in range(1,n_iter+1)]

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

    phi_prev = phi[batch].copy()
    gamma_prev = gamma[batch].copy()
    for j in range(20):
        phi[batch] = update_q_Z_svi(batch,w,gamma_prev,lmbda)
        gamma[batch] = update_q_theta_svi(batch, phi_prev, alpha)

        if (np.sum(np.abs(phi_prev - phi[batch])) < 0.1*S and (np.sum(np.abs(gamma_prev - gamma[batch])) < 0.1*S
            break

        gamma_prev = gamma[batch].copy()
```

```
        phi_prev = phi[batch].copy()

    lmbda_intermediate = update_q_beta_svi(batch, w, phi, eta)

    lmbda_nxt = (1-rho)*lmbda + rho*lmbda_intermediate

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

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

  return phi_out, gamma_out, lmbda_out, elbo
```

CASE 1

Tiny dataset

In [155…

```
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, a
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, al
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]
```

```
[[4 7 8 4 3]
 [4 6 8 8 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.

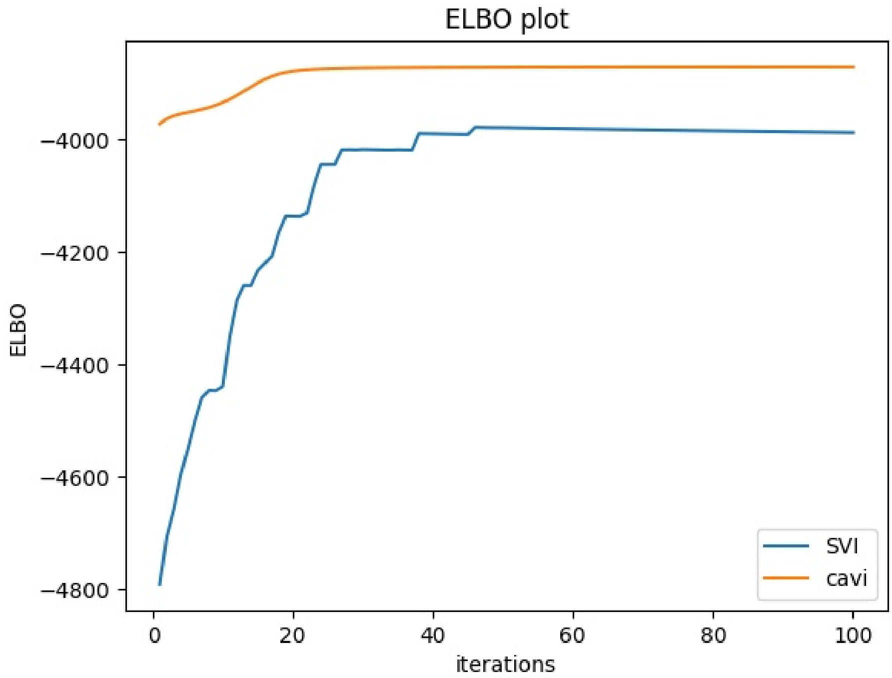
In [156…

```
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:  [0.882 0.118]
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:    [0.201 0.151 0.269 0.320 0.058]
E[beta] SVI k=1:    [0.091 0.268 0.112 0.249 0.280]
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 [157…

```
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, label="cavi")
plt.title("ELBO plot")
plt.legend()
plt.xlabel("iterations")
plt.ylabel("ELBO")
plt.show()
```



In [15]:

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

CASE 2

Small dataset

In [158…

```
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, a
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, al
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]
```

```
[[5 2 3 7 4 7 2 8 2 4]
 [9 8 7 2 8 5 6 1 1 7]
 [6 2 5 9 5 9 1 2 5 6]]
```

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.

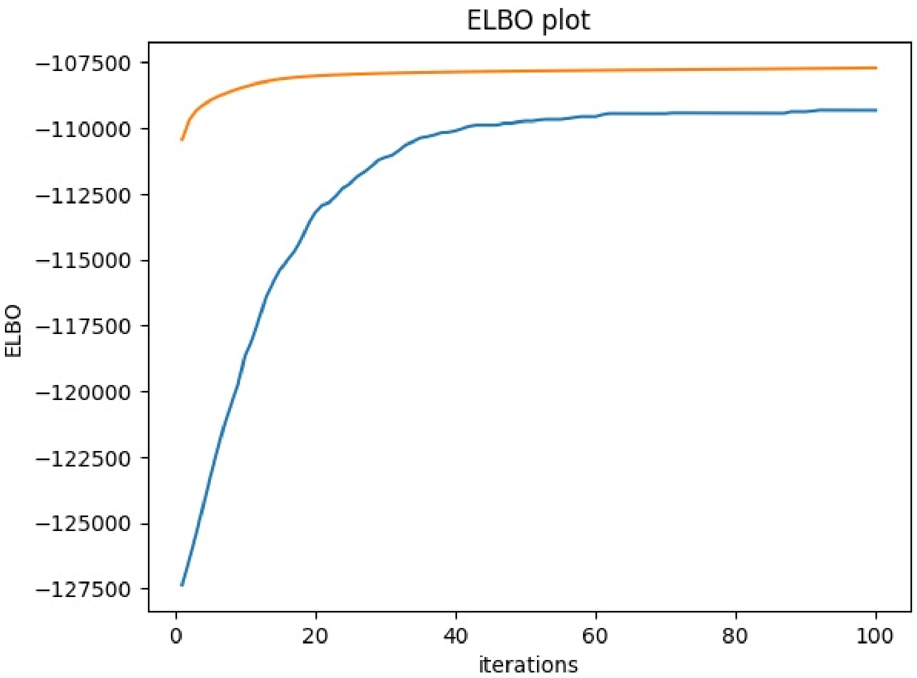
In [159…

```
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:      [0.404 0.327 0.269]
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:    [0.077 0.062 0.055 0.257 0.041 0.010 0.041 0.021 0.353 0.084]
E[beta] k=1:    [0.167 0.142 0.063 0.107 0.017 0.068 0.005 0.288 0.072 0.072]
E[beta] k=2:    [0.309 0.032 0.112 0.023 0.004 0.114 0.044 0.148 0.071 0.142]
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: 58.699244022369385
Time CAVI: 48.015001104904175
```

In [160…

```
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 [161…

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

CASE 3

Medium small dataset, one iteration for time analysis.

In [162…

```
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_
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_
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]
```

```
[[2 3 4 1 1 2 5 1 5 5]
 [1 9 3 1 8 3 6 1 6 8]
 [8 9 7 6 1 5 2 9 2 5]
 [3 2 6 9 5 8 7 4 6 7]
 [3 1 7 5 8 1 7 8 4 2]]
```

In [163…

```
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: 68.96532678604126
Time CAVI: 74.91650700569153
```

In [22]:

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

In [ ]: