Homework 3: The Expectation Maximization (EM) algorithm

STAT 348, UChicago, Spring 2025

Your name here: Zijiang Yang

Hours spent: 20

(Please let us know how many hours in total you spent on this assignment so we can calibrate for future assignments. Your feedback is always welcome!)

Open in Colab

Instructions

The purpose of this assignment is to explore the ideas in weeks 2-3:

- mixture models
- probabilistic graphical models
- conditional conjugacy
- the EM algorithm

Assignment is due **Sunday April 20, 11:59pm** on GradeScope.

Zero-inflated Poisson Mixture Model

The following generative process defines the model we will be working with:

$$\boldsymbol{\pi} \sim \mathrm{Dir}(\boldsymbol{\alpha}_0) \tag{1}$$
$$\omega_{k,d} \overset{\text{iid}}{\sim} \mathrm{Beta}(a_0, b_0) \tag{2}$$
$$\lambda_{k,d} \overset{\text{iid}}{\sim} \mathrm{Gamma}(e_0, f_0) \tag{3}$$
$$z_i \overset{\text{iid}}{\sim} \mathrm{Cat}(\boldsymbol{\pi}) \tag{4}$$
$$b_{i,d} \overset{\text{ind.}}{\sim} \mathrm{Bernoulli}(\omega_{z_i,d}) \tag{5}$$
$$x_{i,d} \overset{\text{ind.}}{\sim} \begin{cases} \delta_0 & \text{if } b_{i,d} = 0 \\ \mathrm{Pois}(\lambda_{z_i,d}) & \text{if } b_{i,d} = 1 \end{cases} \tag{6}$$

- Each observation is a D-dimensional count vector $\boldsymbol{x}_i = (x_{i,1}, \dots, x_{i,D}) \in \mathbb{N}_0^D$.
- Each $x_{i,d}$ is associated with a latent Bernoulli variable $b_{i,d}$. When $b_{i,d} = 1$, $x_{i,d}$ is sampled from a Poisson. When $b_{i,d} = 0$, $x_{i,d} = 0$ deterministically.
- Each observation is assigned to one of K mixture components $z_i \in \{1, \dots, K\}$.
- We will often use the following transformation of z_i

$$\zeta_{ik} \triangleq 1(z_i = k), \;\; \boldsymbol{\zeta}_i = (\zeta_{i1}, \dots, \zeta_{iK})$$

- An equivalent way to sample the cluster assignment is:

$$\boldsymbol{\zeta}_i \sim \mathrm{Multinomial}(1, \boldsymbol{\pi})$$

- Each mixture component k is associated with two D-dimensional parameter vectors, $\boldsymbol{\omega}_k$ and $\boldsymbol{\lambda}_k$.
- The vector $\boldsymbol{\omega}_k = (\omega_{k,1}, \dots, \omega_{k,D})$ gives the probabilities for the zero-inflated part of the likelihood where each $\omega_{k,d} \in (0, 1)$ is Beta distributed.
- The vector $\boldsymbol{\lambda}_k = (\lambda_{k,1}, \dots, \lambda_{k,D})$ gives the rates for the Poisson part of the likelihood where each $\lambda_{k,d} > 0$ is Gamma distributed.
- The weights $\boldsymbol{\pi} \in \Delta_K$ give the prior probability of each component and are Dirichlet distributed.
- We will refer to the full set of model parameters as $\Theta = \{\Lambda, \Omega, \boldsymbol{\pi}\}$.
- We will refer to the full set of hyperparameters as $\boldsymbol{\eta}_0 = \{a_0, b_0, e_0, f_0, \boldsymbol{\alpha}_0\}$.

The **likelihood** of the model can be expressed as:

$$P(X \mid B, Z, \Theta, \boldsymbol{\eta}_0) = \prod_{i=1}^{n} \prod_{k=1}^{K} \prod_{d=1}^{D} \mathrm{Pois}(x_{i,d}; \lambda_{k,d})^{\zeta_{i,k}\, b_{i,d}} \delta_0(x_{i,d})^{\zeta_{i,k}(1-b_{i,d})} \tag{7}$$

The **"complete data likelihood"** of the model is then $p(X, B, Z \mid \Theta, \boldsymbol{\eta}_0) = p(X \mid B, Z, \Theta, \boldsymbol{\eta}_0)\, p(B, Z \mid \Theta, \boldsymbol{\eta}_0)$ where

$$p(B, Z \mid \Theta, \boldsymbol{\eta}_0) = \prod_{i=1}^{n} \prod_{k=1}^{K} \left[p(z_i = k \mid \boldsymbol{\pi}) \prod_{d=1}^{D} p(b_{i,d} \mid \omega_{k,d}) \right]^{\zeta_{i,k}} \tag{8}$$
$$= \prod_{i=1}^{n} \prod_{k=1}^{K} \pi_k^{\zeta_{i,k}} \prod_{d=1}^{D} \omega_{k,d}^{\zeta_{i,k} b_{i,d}} (1 - \omega_{k,d})^{\zeta_{i,k}(1-b_{i,d})} \tag{9}$$

The **prior** over parameters $\Lambda, \Omega, \boldsymbol{\pi}$ is:

$$p(\Theta \mid \boldsymbol{\eta}_0) = p(\boldsymbol{\pi} \mid \boldsymbol{\alpha}) \prod_{k=1}^{K} \prod_{d=1}^{D} p(\omega_{k,d} \mid a_0, b_0)\, p(\lambda_{k,d} \mid e_0, f_0) \tag{10}$$
$$= \mathrm{Dir}(\boldsymbol{\pi}; \boldsymbol{\alpha}) \prod_{k=1}^{K} \prod_{d=1}^{D} \mathrm{Beta}(\omega_{k,d}; a_0, b_0)\, \mathrm{Gamma}(\lambda_{k,d}; e_0, f_0) \tag{11}$$

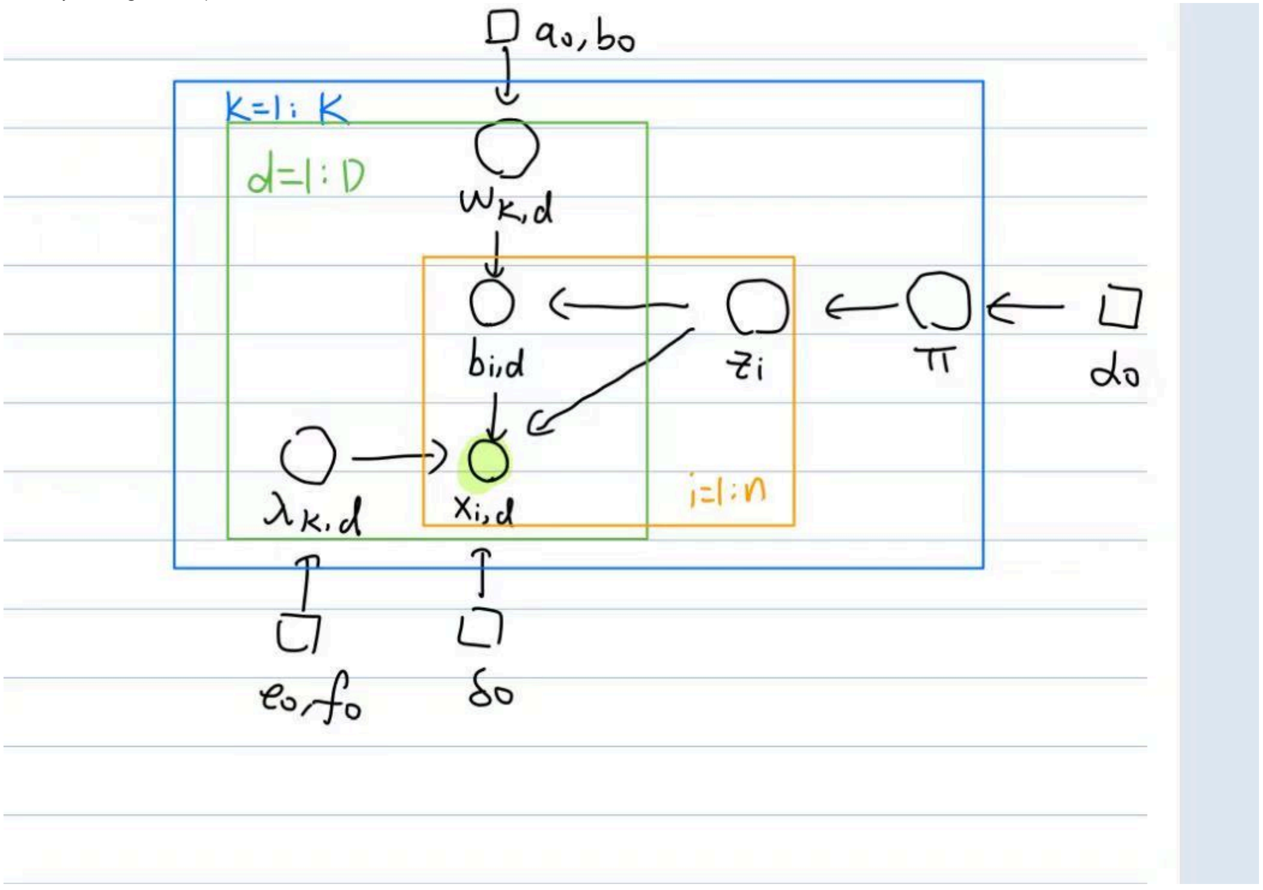
Problem 0: Draw the graphical model [Visual].

Using either sofware (e.g., TikZ, Keynote) or **very neat handwriting**, create a probabilistic graphical model that describes the generative process above.

Your PGM should:

- Use plate notation to denote repeated sampling.
- Use shaded circular nodes to denote observed variables.
- Use un-shaded circular nodes to denote latent variables.
- Use square or bullet-nodes to denote hyperparameters.

Include your image in the space below.



Problem 1: Derive the complete conditionals [Math]

1a): Derive the complete conditional distribution $p(\boldsymbol{\pi} \mid -) \equiv p(\boldsymbol{\pi} \mid X, B, Z, \Theta_{\setminus\pi}, \boldsymbol{\eta}_0)$.

$$\begin{aligned}
p(\pi|-) &= p(\pi|z_{1:n}, \alpha_0) \\
&\propto [\prod_{i=1}^{n} p(z_i|\pi, \alpha_0)] \cdot p(\pi|\alpha_0) \\
&\propto [\prod_{i=1}^{n}\prod_{k=1}^{K} \pi_k^{\zeta_{ik}}] \cdot \frac{1}{B(\alpha_0)} \prod_{k=1}^{K} \pi_k^{\alpha_{0,k}-1} \\
&\propto \prod_{k=1}^{K} \pi_k^{\sum_{i=1}^{n}\zeta_{ik}+\alpha_{0,k}-1} \\
&\propto Dir(\sum_{i=1}^{n} \zeta_i + \alpha_0) \\
p(\pi|-) &= \frac{1}{B(\sum_{i=1}^{n}\zeta_i + \alpha_0)} \prod_{k=1}^{K} \pi_k^{\sum_{i=1}^{n}\zeta_{ik}+\alpha_{0,k}-1}
\end{aligned}$$

Here B is the multivariable Beta function.

1b): Derive the complete conditional distribution $p(\omega_{k,d} \mid -)$.

Denote $\mathcal{I}_k = \{i|z_i = k\}$

$$
\begin{aligned}
p(\omega_{k,d}|-) &= p(\omega_{k,d}|a_0, b_0, b_{\mathcal{I}_k,d}, z_{\mathcal{I}_k}) \\
&\propto \prod_{i\in\mathcal{I}_k} p(b_{i,d}|\omega_{k,d}) \cdot p(\omega_{k,d}|a_0, b_0) \\
&\propto \prod_{i=1}^{n} p(b_{i,d}|\omega_{k,d})^{\zeta_{i,k}} \cdot p(\omega_{k,d}|a_0, b_0) \\
&\propto \prod_{i=1}^{n} \omega_{k,d}^{\zeta_{i,k}b_{i,d}}(1-\omega_{k,d})^{\zeta_{i,k}(1-b_{i,d})} \cdot \frac{1}{B(a_0,b_0)} w_{k,d}^{a_0-1}(1-w_{k,d})^{b_0-1} \\
&\propto \omega_{k,d}^{\sum_{i=1}^{n}\zeta_{i,k}b_{i,d}+a_0-1}(1-\omega_{k,d})^{\sum_{i=1}^{n}\zeta_{i,k}(1-b_{i,d})+b_0-1} \\
&\propto Beta(\sum_{i=1}^{n}\zeta_{i,k}b_{i,d}+a_0, \sum_{i=1^m}\zeta_{i,k}(1-b_{i,d})+b_0)
\end{aligned}
$$

Let $a_k = \sum_{i=1}^{n}\zeta_{i,k}b_{i,d}+a_0$, $b_k = \sum_{i=1}^{n}\zeta_{i,k}(1-b_{i,d})+b_0$

$$
\begin{aligned}
p(\omega_{k,d}|-) &= Beta(a_k, b_k) \\
&= \frac{1}{B(a_k,b_k)} w_{k,d}^{a_k-1}(1-w_{k,d})^{b_k-1}
\end{aligned}
$$

1c): Derive the complete conditional distribution $p(\lambda_{k,d} \mid -)$.

Denote $\mathcal{I}_{k,1} = \{i|z_i = k, b_{i,d} = 1\}$.

$$
\begin{aligned}
p(\lambda_{k,d}|-) &= p(\lambda_{k,d}|x_{\mathcal{I}_{k,1},d}, e_0, f_0) \\
&\propto \prod_{i\in\mathcal{I}_{k,1}} p(x_{i,d}|\lambda_{k,d})p(\lambda_{k,d}|e_0, f_0) \\
&\propto \prod_{i=1}^{n}(\frac{\lambda_{k,d}^{x_{i,d}}e^{-\lambda_{k,d}}}{x_{i,d}!})^{\zeta_{i,k}b_{i,d}} \cdot \frac{f_0^{e_0}}{\Gamma(e_0)}\lambda_{k,d}^{e_0-1}e^{-f_0\lambda_{k,d}} \\
&\propto \lambda_{k,d}^{\sum_{i=1}^{n}\zeta_{i,k}b_{i,d}x_{i,d}+e_0-1}e^{-(\sum_{i=1}^{n}\zeta_{i,k}b_{i,d}+f_0)\lambda_{k,d}}
\end{aligned}
$$

Let $e_k = \sum_{i=1}^{n}\zeta_{i,k}b_{i,d}x_{i,d}+e_0$, $f_k = \sum_{i=1}^{n}\zeta_{i,k}b_{i,d}+f_0$.

$$
\begin{aligned}
p(\lambda_{k,d}|-) &= Gamma(e_k, f_k) \\
&= \frac{f_k^{e_k}}{\Gamma(e_k)}\lambda_{k,d}^{e_k-1}e^{-f_k\lambda_{k,d}}
\end{aligned}
$$

1d): Derive the complete conditional distribution $p(b_{i,d} = 1 \mid z_i = k, -)$.

- In this case, when conditioning on z_i, assume that $z_i = k$.

When $b_{i,d} = 0$, $x_{i,d}$ can only be 0, so we can derive that when $x_{i,d} > 0$,

$$
p(b_{i,d} = 1|z_i = k, x_{i,d} > 0, -) = 1
$$

Then we consider the situation that $x_{i,d} = 0$.

$$
\begin{aligned}
p(b_{i,d}|z_i = k, x_{i,d} = 0, -) &\propto p(x_{i,d}|b_{i,d}, \lambda_{k,d}, z_i = k)p(b_{i,d}|w_{k,d}, z_i = k) \\
&\propto (1(z_{i,d} = 0))^{1-b_{i,d}}(Pois(x_{i,d};\lambda_{k,d}))^{b_{i,d}} \cdot w_{k,d}^{b_{i,d}}(1-w_{k,d})^{1-b_{i,d}} \\
&\propto (1(z_{i,d} = 0))^{1-b_{i,d}}(\frac{\lambda_{k,d}^{x_{i,d}}e^{-\lambda_{k,d}}}{x_{i,d}!}w_{k,d})^{b_{i,d}} \cdot w_{k,d}^{b_{i,d}}(1-w_{k,d})^{1-b_{i,d}} \\
&\propto (1-w_{k,d})^{1-b_{i,d}} \cdot (Pois(x_{i,d};\lambda_{k,d}) \cdot \omega_{k,d})^{b_{i,d}}
\end{aligned}
$$

Denote $p_{i,k,d} = \frac{Pois(x_{i,d};\lambda_{k,d})\cdot\omega_{k,d}}{(1-w_{k,d})\cdot 1\{x_{i,d}=0\}+Pois(x_{i,d};\lambda_{k,d})\cdot\omega_{k,d}}$.

$$
\begin{aligned}
p(b_{i,d}|z_i = k, x_{i,d} = 0, -) &= Bernoulli(p_{i,k,d}) \\
&= p_{i,k,d}^{b_{i,d}}(1-p_{i,k,d})^{1-b_{i,d}}
\end{aligned}
$$

Therefore, \$\$

$$
\begin{aligned}
p(b_{i,d} = 1|z_i = k, -) &= p_{i,k,d} \\
&= \frac{Pois(x_{i,d};\lambda_{k,d})\cdot\omega_{k,d}}{(1-w_{k,d})\cdot 1\{x_{i,d} = 0\}+Pois(x_{i,d};\lambda_{k,d})\cdot\omega_{k,d}}
\end{aligned}
$$

\$\$

1e): Derive the complete conditional distribution $p(z_i = k \mid -)$.

$$
\begin{aligned}
p(z_i = k|-) &\propto \prod_{d=1}^{D} p(x_{i,d}|b_{i,d}, w_{k,d}) \cdot p(b_{i,d}|w_{k,d}) \cdot p(z_i = k|\pi) \\
&\propto [\prod_{d=1}^{D}(1(x_{i,d} = 0))^{1-b_{i,d}}(Pois(x_{i,d};\lambda_{k,d}))^{b_{i,d}}\omega_{k,d}^{b_{i,d}}(1-\omega_{k,d})^{1-b_{i,d}}] \cdot \pi_k
\end{aligned}
$$

Denote

$$
\begin{aligned}
\tilde{\pi}_{post,k} &= [\prod_{d=1}^{D}(1(x_{i,d} = 0))^{1-b_{i,d}}(Pois(x_{i,d};\lambda_{k,d}))^{b_{i,d}}\omega_{k,d}^{b_{i,d}}(1-\omega_{k,d})^{1-b_{i,d}}] \cdot \pi_k \\
\pi_{post,k} &= \frac{\tilde{\pi}_{post,k}}{\sum_{k'=1}^{K}\tilde{\pi}_{post,k'}} \\
p(z_i = k|-) &= \pi_{post,k}
\end{aligned}
$$

1f): Derive the "**incomplete**" conditional distribution $p(z_i = k \mid -\backslash_{\boldsymbol{b}_i})$.

- This is the distribution of z_i that conditions on everything **except $\boldsymbol{b}_i$**.
- **Hint:** One way to proceed is to first write down the marginal likelihood $P(x_{i,d} \mid z_i, \Omega, \Lambda)$, with $b_{i,d}$ marginalized out, and then proceed to derive the complete conditional of z_i as if that were the likelihood.

If $x_{i,d} > 0$,

$$\begin{aligned} p(x_{i,d}|z_i = k, -\backslash b_i) &= p(x_{i,d}|z_i = k, b_{i,d} = 1, -)p(b_{i,d} = 1|-) + p(x_{i,d}|z_i = k, b_{i,d} = 0, -)p(b_{i,d} = 0|-) \\ &= Pois(x_{i,d}; \lambda_{k,d})w_{k,d} \end{aligned}$$

If $x_{i,d} = 0$,

$$\begin{aligned} p(x_{i,d}|z_i = k, -\backslash b_i) &= p(x_{i,d}|z_i = k, b_{i,d} = 1, -)p(b_{i,d} = 1|-) + p(x_{i,d}|z_i = k, b_{i,d} = 0, -)p(b_{i,d} = 0|-) \\ &= Pois(x_{i,d}; \lambda_{k,d})w_{k,d} + 1 - w_{k,d} \end{aligned}$$

Thus,

$$p(x_{i,d}|z_i = k, -\backslash b_i) = Pois(x_{i,d}; \lambda_{k,d})w_{k,d} + 1(x_{i,d} = 0)(1 - w_{k,d})$$

Therefore, we can derive the incomplete conditional distribution:

$$\begin{aligned} p(z_i = k| - \backslash b_i) &\propto \prod_{d=1}^{D} p(x_{i,d}|z_i = k, -\backslash b_i)p(z_i = k|\pi) \\ &\propto \prod_{d=1}^{D} [Pois(x_{i,d}; \lambda_{k,d})w_{k,d} + 1(x_{i,d} = 0)(1 - w_{k,d})] \cdot \pi_k \end{aligned}$$

Denote

$$\begin{aligned} \tilde{\pi}_{icpost,k} &= \prod_{d=1}^{D} [Pois(x_{i,d}; \lambda_{k,d})w_{k,d} + 1(x_{i,d} = 0)(1 - w_{k,d})] \cdot \pi_k \\ \pi_{icpost,k} &= \frac{\tilde{\pi}_{icpost,k}}{\sum_{k'=1}^{K} \tilde{\pi}_{icpost,k'}} \\ p(z_i = k| - \backslash b_i) &= \pi_{icpost,k} \end{aligned}$$

Problem 2: Derive the EM algorithm

In this part, we will implement the EM algorithm to do **MAP estimation** for the parameters:

$$\widehat{\Theta}^{\text{MAP}} = \arg\max_{\boldsymbol{\Theta}} p(\Theta \mid X, \boldsymbol{\eta}_0) \tag{12}$$

As a reminder, the EM algorithm finds a local mode of the posterior by performing coordinate ascent on the **evidence lower bound (ELBO)**:

$$B(q, \Theta) = \mathbb{E}_{q(B,Z)} \left[\log \frac{p(X, B, Z, \Theta \mid \boldsymbol{\eta}_0)}{q(B, Z)} \right] \tag{13}$$

More specifically, EM iterates maximizing the ELBO with respect to q (E-step) and then with respect to Θ (M-step). The optimal updates for both steps can be written as:

$$\begin{aligned} &(\text{E-step}) & q &= \arg\max_q B(q, \Theta^{\text{old}}) = p(B, Z \mid \Theta^{\text{old}}, \boldsymbol{\eta}_0) & (14) \\ &(\text{M-step}) & \Theta^{\text{new}} &= \arg\max_{\Theta} B(q, \Theta) = \arg\max_{\boldsymbol{\Theta}} \mathbb{E}_q \left[\log p(X, B, Z, \Theta \mid \boldsymbol{\eta}_0) \right] & (15) \end{aligned}$$

We will help you by telling you that for the E-step, it will suffice to compute the following posterior expectations:

$$\begin{aligned} q(z_i = k) &\equiv \mathbb{E}_q[\zeta_{i,k}] & &\text{for all } i, k & (16) \\ q(b_{i,d} = 1 \mid z_i = k) &\equiv \mathbb{E}_q[b_{i,d} \mid z_i = k] & &\text{for all } i, d, k & (17) \end{aligned}$$

We will further help you telling you that the M-step in this model can be equivalently written as performing the following separate steps:

$$\begin{aligned} \boldsymbol{\pi} &\leftarrow \arg\max_{\boldsymbol{\pi}} \mathbb{E}_q \left[\log P(\boldsymbol{\pi} \mid -) \right] & & & (18) \\ \omega_{k,d} &\leftarrow \arg\max_{\omega_{k,d}} \mathbb{E}_q \left[\log P(\omega_{k,d} \mid -) \right] & &\text{for all } k, d & (19) \\ \lambda_{k,d} &\leftarrow \arg\max_{\lambda_{k,d}} \mathbb{E}_q \left[\log P(\lambda_{k,d} \mid -) \right] & &\text{for all } k, d & (20) \end{aligned}$$

2a) Derive the E-step.

Derive and provide forms for the posterior expectations $\mathbb{E}_q[\zeta_{i,k}]$ and $\mathbb{E}_q[b_{i,d} \mid z_i = k]$ described above.

- Your answers can be in terms of known distributions---e.g., $\text{Pois}(x_{i,d};\ \lambda_{k,d})$---you do not need to simplify beyond this.
- Show the steps of your derivation, but be concise, and make sure your final answer is clearly visible.

<1> $\mathbb{E}_q[\zeta_{i,k}]$

According to Problem 1e)

$$\begin{aligned} \mathbb{E}_q[\zeta_{i,k}] &= p(z_i = k|\Theta^{old}, -\backslash_{b_{i,d}}) \\ &= \frac{\prod_{d=1}^{D} [Pois(x_{i,d}; \lambda_{k,d}^{old})w_{k,d}^{old} + 1(x_{i,d} = 0)(1 - w_{k,d}^{old})] \cdot \pi_k^{old}}{\sum_{k'=1}^{K} \prod_{d=1}^{D} [Pois(x_{i,d}; \lambda_{k',d}^{old})w_{k',d}^{old} + 1(x_{i,d} = 0)(1 - w_{k',d}^{old})] \cdot \pi_{k'}^{old}} \end{aligned}$$

<2> $\mathbb{E}_q[b_{i,d} \mid z_i = k]$

According to 1d)

$$\mathbb{E}_q[b_{i,d} \mid z_i = k, \Theta^{old}] = \frac{Pois(x_{i,d}; \lambda_{k,d}^{old}) \cdot \omega_{k,d}^{old}}{(1 - \omega_{k,d}^{old}) \cdot 1\{x_{i,d} = 0\} + Pois(x_{i,d}; \lambda_{k,d}^{old}) \cdot \omega_{k,d}^{old}}$$

2b) Derive the M-step.

Derive and provide forms for the M-step updates to the parameters.

- Show the steps of your derivation, but be concise, and make sure your final answer is clearly visible.
- **Hint:** Most common distributions (e.g., Gamma) have closed-form formulas for their mode. If the mode is only defined for certain settings of the hyperparameters, you can assume they will hold in your setting.

<1> π

$$\begin{aligned}\pi &= \arg\max_{\pi} E_q[logp(\pi|-)]\\ &= \arg\max_{\pi} E_q[log(\frac{1}{B(\sum_{i=1}^n \zeta_i+\alpha_0)}\prod_{k=1}^K \pi_k^{\sum_{i=1}^n \zeta_{ik}+\alpha_{0,k}-1})]\\ &= \arg\max_{\pi} E_q[\sum_{k=1}^K(\sum_{i=1}^n \zeta_{i,k}+\alpha_{0,k}-1)log\pi_k]\\ &= \arg\max_{\pi} \sum_{k=1}^K(\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1)log\pi_k\end{aligned}$$

Since we have a constraint $\sum_{k=1}^K \pi_k = 1$, we create a Lagrange function to derive the maximum arguments.

$$\begin{aligned}L(\pi) &= \sum_{k=1}^K(\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1)log\pi_k + \lambda(\sum_{k=1}^K \pi_k - 1)\\ \frac{\partial L}{\partial \pi_k} &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1}{\pi_k}+\lambda = 0\\ \lambda &= \sum_{k=1}^K(\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1)\\ \pi_k &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1}{\sum_{k=1}^K(\sum_{i=1}^n E_q[\zeta_{i,k}]+\alpha_{0,k}-1)}\end{aligned}$$

<2> $w_{k,d}$

$$\begin{aligned}w_{k,d} &= \arg\max_{w_{k,d}} E_q[logp(w_{k,d}|-)]\\ &= \arg\max_{w_{k,d}} E_q[(\sum_{i=1}^n \zeta_{i,k}b_{i,d}+a_0-1)logw_{k,d} + (\sum_{i=1}^n \zeta_{i,k}(1-b_{i,d})+b_0-1)log(1-w_{k,d})]\\ &= \arg\max_{w_{k,d}}(\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+a_0-1)logw_{k,d} + (\sum_{i=1}^n E_q[\zeta_{i,k}](1-E_q[b_{i,d}|z_i=k])+b_0-1)log(1-w_{k,d})\end{aligned}$$

Denote $L(w_{k,d}) = (\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+a_0-1)logw_{k,d} + (\sum_{i=1}^n E_q[\zeta_{i,k}](1-E_q[b_{i,d}|z_i=k])+b_0-1)log(1-w_{k,d})$.

$$\begin{aligned}\frac{\partial L}{\partial w_{k,d}} &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+a_0-1}{w_{k,d}} - \frac{\sum_{i=1}^n E_q[\zeta_{i,k}](1-E_q[b_{i,d}|z_i=k])+b_0-1}{1-w_{k,d}}\\ &= 0\\ w_{k,d} &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+a_0-1}{\sum_{i=1}^n E_q[\zeta_{i,k}]+a_0+b_0-2}\end{aligned}$$

<3> $\lambda_{k,d}$

$$\begin{aligned}\lambda_{k,d} &= \arg\max_{\lambda_{k,d}} E_q[logp(\lambda_{k,d}|-)]\\ &= \arg\max_{\lambda_{k,d}} E_q[(\sum_{i=1}^n \zeta_{i,k}b_{i,d}x_{i,d}+e_0-1)log\lambda_{k,d} - (\sum_{i=1}^n \zeta_{i,k}b_{i,d}+f_0)\lambda_{k,d}]\\ &= \arg\max_{\lambda_{k,d}}(\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]x_{i,d}+e_0-1)log\lambda_{k,d} - (\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+f_0)\lambda_{k,d}\end{aligned}$$

Denote $L(\lambda_{k,d}) = (\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]x_{i,d}+e_0-1)log\lambda_{k,d} - (\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+f_0)\lambda_{k,d}$

$$\begin{aligned}\frac{\partial L}{\lambda_{k,d}} &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]x_{i,d}+e_0-1}{\lambda_{k,d}} - (\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+f_0)\\ &= 0\\ \lambda_{k,d} &= \frac{\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]x_{i,d}+e_0-1}{\sum_{i=1}^n E_q[\zeta_{i,k}]E_q[b_{i,d}|z_i=k]+f_0}\end{aligned}$$

2c) Derive the ELBO.

Derive an analytic form for the ELBO. Because this can be tricky, we encourage you to break down the derivation in to four separate pieces:

$$B(q,\Theta) = \mathbb{E}_{q(B,Z)}\left[\log\frac{p(X,B,Z,\Theta\mid\boldsymbol{\eta}_0)}{q(B,Z)}\right] \tag{21}$$

$$= \mathbb{E}_q\Big[\log p(X\mid B,Z,\Theta,\boldsymbol{\eta}_0)\Big] + \mathbb{E}_q\Big[\log p(B,Z\mid\Theta,\boldsymbol{\eta}_0)\Big] - \mathbb{E}_q\Big[\log q(B,Z)\Big] + \log p(\Theta\mid\boldsymbol{\eta}_0) \tag{22}$$

- Show the steps of your derivation, but be concise, and make sure your final answer is clearly visible.
- Your answers can be in terms of known distributions---e.g., $\text{Pois}(x_{i,d};\ \lambda_{k,d})$---you do not need to simplify beyond this.
- You do not need to provide a form for the fourth term $\log p(\Theta\mid\boldsymbol{\eta}_0)$ since we have given that earlier.

Denote following notations:

$$\begin{aligned}E_q[\zeta_{i,k}] &= q(z_i=k) = \eta_{i,k}\\ E_q[b_{i,d}|z_i=k] &= q(b_{i,d}=1|z_i=k) = \phi_{i,d,k}\end{aligned}$$

$$\mathbb{E}_q\Big[\log p(X \mid B, Z, \Theta, \boldsymbol{\eta}_0)\Big] = E_q[\sum_{i=1}^{n}\sum_{k=1}^{K}\sum_{d=1}^{D}\zeta_{i,k}b_{i,d}logPois(x_{i,d};\lambda_{k,d}) + \zeta_{i,k}(1-b_{i,d})log\delta_0(x_{i,d})]$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}E_q[\zeta_{i,k}](\sum_{d=1}^{D}E_q[b_{i,d}|z_i=k]logPois(x_{i,d};\lambda_{k,d}) + (1-E_q[b_{i,d}|z_i=k])log\delta_0(x_{i,d}))$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}(\sum_{d=1}^{D}\phi_{i,d,k}logPois(x_{i,d};\lambda_{k,d}) + (1-\phi_{i,d,k})log\delta_0(x_{i,d}))$$

$$\mathbb{E}_q\Big[\log p(B, Z \mid \Theta, \boldsymbol{\eta}_0)\Big] = E_q[\sum_{i=1}^{n}\sum_{k=1}^{K}(\zeta_{i,k}log\pi_k + \sum_{d=1}^{D}\zeta_{i,k}b_{i,d}logw_{k,d} + \zeta_{i,k}(1-b_{i,d})log(1-w_{k,d}))]$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}E_q[\zeta_{i,k}][log\pi_k + \sum_{d=1}^{D}[E_q[b_{i,d}|z_i=k]logw_{k,d} + (1-E_q[b_{i,d}|z_i=k])log(1-w_{k,d})]]$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}[log\pi_k + \sum_{d=1}^{D}[\phi_{i,d,k}logw_{k,d} + (1-\phi_{i,d,k})log(1-w_{k,d})]]$$

$$\mathbb{E}_q\Big[\log q(B, Z)\Big] = E_q[log(\prod_{i=1}^{n}q(z_i)\cdot(\prod_{d=1}^{D}q(b_{i,d}|z_i)))]$$

$$= E_q[\sum_{i=1}^{n}logq(z_i) + \sum_{i=1}^{n}\sum_{d=1}^{D}logq(b_{i,d}|z_i)]$$

$$= \sum_{i=1}^{n}E_{q(z_i)}logq(z_i) + \sum_{i=1}^{n}E_{q(z_i)}\sum_{d=1}^{D}E_{q(b_{i,d}|z_i)}logq(b_{i,d}|z_i)$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}q(z_i=k)logq(z_i=k) + \sum_{i=1}^{n}\sum_{k=1}^{K}q(z_i=k)\sum_{d=1}^{D}[q(b_{i,d}=1|z_i=k)logq(b_{i,d}=1|z_i=k) + q(b_{i,d}=0|z_i=k)logq(b_{i,d=0}|z_i=k)]$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}log\eta_{i,k} + \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}\sum_{d=1}^{D}[\phi_{i,d,k}log\phi_{i,d,k} + (1-\phi_{i,d,k})log(1-\phi_{i,d,k})]$$

$$B(q,\Theta) = \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}(\sum_{d=1}^{D}\phi_{i,d,k}logPois(x_{i,d};\lambda_{k,d}) + (1-\phi_{i,d,k})log\delta_0(x_{i,d}))$$

$$+ \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}[log\pi_k + \sum_{d=1}^{D}[\phi_{i,d,k}logw_{k,d} + (1-\phi_{i,d,k})log(1-w_{k,d})]]$$

$$- \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}log\eta_{i,k}$$

$$- \sum_{i=1}^{n}\sum_{k=1}^{K}\eta_{i,k}\sum_{d=1}^{D}[\phi_{i,d,k}log\phi_{i,d,k} + (1-\phi_{i,d,k})log(1-\phi_{i,d,k})]$$

$$+ \log p(\Theta \mid \boldsymbol{\eta}_0)$$

Question 3: Implement EM [Code]

In this question you will implement the full EM algorithm using PyTorch for MAP estimation in the zero-inflated Poisson mixture model.

We have provided a skeleton structure for the code you should implement below. The function `em` below will run EM on the inputs. This function is fully implemented, and you do not need to edit it. However, this function calls the following functions:

- `initialize_em` : initializes the parameters and posterior expectations
- `e_step` : runs the E-step and returns an updated version of the posterior expectations
- `m_step` : runs the M-step and returns an updated version of the parameters
- `elbo` : calculates the evidence lower bound (ELBO)

For your convenience, we have also fully implemented `initialize_em` . Your job is therefore to implement the other three functions (i.e., `e_step` , `m_step` , `elbo`) to get a fully working implementation of `em` .

By default, the functions you will implement take the following four dictionaries as arguments:

- `data` : contains X
- `belief_state` : contains the posterior expectations $\mathbb{E}_q[\zeta_{i,k}]$ and $\mathbb{E}_q[b_{i,d} \mid z_i = k]$
- `params_state` : contains the parameters Θ
- `hyperparams` : contains the hyperparameters $\boldsymbol{\eta}_0$

Each function may or may not require all of the information in all dictionaries. You can also see the names assigned to the different variables in `initialize_em` , and we encourage you not to change those.

In [ ]:
```
import torch
from tqdm import tqdm

def initialize_em(data, hyperparams, seed=617):
    """Initialize the parameters and beliefs for the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        hyperparams (dict): dictionary of hyperparameters

    Returns:
        params_state (dict): dictionary of parameters
        belief_state (dict): dictionary of expectations of latent variables
    """
    # set the random seed for torch
    torch.manual_seed(seed)

    # get hyperparameters and sizes
    n, D = data['X_ND'].shape
    K = hyperparams['alpha_K'].shape[0]
    a0, b0, e0, f0, alpha_K = [hyperparams[key] for key in ['a0', 'b0', 'e0', 'f0', 'alpha_K']]

    # initialize the parameters with a draw from the prior
    params_state = {}
    params_state['pi_K'] = torch.distributions.Dirichlet(alpha_K).sample()
    params_state['omega_KD'] = torch.distributions.Beta(a0, b0).sample((K, D))
    params_state['lambd_KD'] = torch.distributions.Gamma(concentration=e0, rate=f0).sample((K, D))

    # initialize the "beliefs" (posterior expectations) uniformly
    # (this is not important because the e-step is run first)
    belief_state = {}
    # this will store all the q(zi=k) = E[zeta_ik] expectations
    belief_state['Ezeta_NK'] = torch.ones((n, K)) / K
    # this will store all the q(bid=1 | zi=k) = E[bid | zi=k] expectations
    belief_state['Eb_NKD'] = torch.ones((n, K, D)) * 0.75
    return params_state, belief_state

def e_step(data, belief_state, params_state, hyperparams):
    """Perform the E-step of the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        belief_state (dict): dictionary of expectations of latent variables
        params_state (dict): dictionary of parameters
        hyperparams (dict): dictionary of hyperparameters

    Returns:
        belief_state (dict): updated dictionary of expectations of latent variables
    """

    return belief_state

def m_step(data, belief_state, params_state, hyperparams):
    """Perform the M-step of the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        belief_state (dict): dictionary of expectations of latent variables
        params_state (dict): dictionary of parameters
        hyperparams (dict): dictionary of hyperparameters

    Returns:
        params_state (dict): updated dictionary of parameters
    """
    # Your code
    return params_state

def elbo(data, params_state, belief_state, hyperparams):
    """
    Compute the Evidence Lower Bound (ELBO) for the Zero-Inflated Poisson Mixture Model.

    Args:
        data (dict): Dictionary containing observed data.
        params_state (dict): Dictionary containing model parameters.
        belief_state (dict): Dictionary containing expectations of latent variables.
        hyperparams (dict): Dictionary containing hyperparameters.

    Returns:
        torch.Tensor: The computed ELBO value.
    """
    # Your code here

def em(data, hyperparams, max_iter=100, tol=1e-7, seed=617, verbose=True):
    """Run the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        hyperparams (dict): dictionary of hyperparameters
        max_iter (int): maximum number of iterations
        tol (float): tolerance for convergence
    """
    params_state, belief_state = initialize_em(data, hyperparams, seed=seed)
    elbo_vals = [elbo(data, params_state, belief_state, hyperparams)]

    # only display a progress bar if verbose=True
    iterator = tqdm(range(max_iter)) if verbose else range(max_iter)
    for _ in iterator:
        belief_state = e_step(data, belief_state, params_state, hyperparams)
        params_state = m_step(data, belief_state, params_state, hyperparams)
        elbo_vals.append(elbo(data, params_state, belief_state, hyperparams))

        curr_elbo_val, last_elbo_val = elbo_vals[-1], elbo_vals[-2]
        assert curr_elbo_val >= last_elbo_val

        # assess convergence (can also be done by looking at the param values)
        if (curr_elbo_val - last_elbo_val).abs() < tol:
            break

    return params_state, belief_state, elbo_vals
```

3a) Implement the E-step.

- **Do not use for loops.** All calculations can and should be done via broadcasting and indexing.
- Make use of torch.logaddexp or torch.logsumexp to **compute (products of) probabilities in log-space** for numerically stable computation. (If your implementation is generating NaNs or INFs, it is often due to not properly doing things in log-space.)
- For your convenience, we have provided a **PyTorch implementation of the Poisson's log PMF** in the function `poisson_logpmf`. This function will **broadcast** (i.e., you can pass in tensors).

In [ ]:
```
def poisson_logpmf(x, rate):
    """
    Compute the log probability mass function of the Poisson distribution.

    Args:
        x (torch.Tensor): The observed data.
        rate (torch.Tensor): The rate parameter (lambda) of the Poisson distribution.

    Returns:
        torch.Tensor: The log probability mass function of the Poisson distribution.
    """
    return x * torch.log(rate) - rate - torch.lgamma(x + 1)

def e_step(data, belief_state, params_state, hyperparams):
    """Perform the E-step of the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        belief_state (dict): dictionary of expectations of latent variables
        params_state (dict): dictionary of parameters
        hyperparams (dict): dictionary of hyperparameters

    Returns:
        belief_state (dict): updated dictionary of expectations of latent variables
    """
    X_ND = data['X_ND']
    pi_K = params_state['pi_K']
    omega_KD = params_state['omega_KD']
    lambd_KD = params_state['lambd_KD']
    n, D = X_ND.shape
    K = pi_K.shape[0]

    X_NKD = X_ND.unsqueeze(1).expand(-1, K, -1)  # (n, K, D)
    lambd_NKD = lambd_KD.unsqueeze(0).expand(n, -1, -1)  # (n, K, D)
    omega_NKD = omega_KD.unsqueeze(0).expand(n, -1, -1)  # (n, K, D)

    pois_logpmf = poisson_logpmf(X_NKD, lambd_NKD)
    p_x_given_z = omega_NKD * torch.exp(pois_logpmf) + (1 - omega_NKD) * (X_NKD == 0).float()
    log_pi_K = torch.log(pi_K).unsqueeze(0).expand(n, -1) #n, K
    log_joint = log_pi_K + torch.log(p_x_given_z).sum(dim=2)
    Ezeta_NK = torch.softmax(log_joint, dim=1)

    numer = torch.exp(pois_logpmf) * omega_NKD
    denom = numer + (1 - omega_NKD)
    Eb = torch.ones_like(numer)
    mask = (X_NKD == 0)
    Eb[mask] = (numer / denom)[mask]


    belief_state['Ezeta_NK'] = Ezeta_NK
    belief_state['Eb_NKD'] = Eb
    assert belief_state['Ezeta_NK'].shape == (n, K)
    assert belief_state['Eb_NKD'].shape == (n, K, D)

    return belief_state
```

3b) Implement the M-step.

- **Do not use for loops.** All calculations can and should be done via broadcasting and indexing.

In [ ]:
```
def m_step(data, belief_state, params_state, hyperparams):
    """Perform the M-step of the EM algorithm for the Zero-Inflated Poisson Mixture Model

    Args:
        data (dict): dictionary of observed variables
        belief_state (dict): dictionary of expectations of latent variables
        params_state (dict): dictionary of parameters
        hyperparams (dict): dictionary of hyperparameters

    Returns:
        params_state (dict): updated dictionary of parameters
    """

    X = data["X_ND"]  # shape (n, D)
    Ez = belief_state["Ezeta_NK"]  # shape (n, K)
    Eb = belief_state["Eb_NKD"]  # shape (n, K, D)
    n, D = X.shape
    K = Ez.shape[1]

    alpha0 = hyperparams["alpha_K"]  # (K,)
    a0 = hyperparams["a0"]
    b0 = hyperparams["b0"]
    e0 = hyperparams["e0"]
    f0 = hyperparams["f0"]

    # ---- Update pi (K,)
    sum_Ez = Ez.sum(dim=0)  # (K,)
    pi_numerator = sum_Ez + alpha0 - 1
    pi = pi_numerator / pi_numerator.sum()

    # ---- Update w (K, D)
    Ez_expanded = Ez.unsqueeze(2).expand(-1,-1,D)  # (n, K, D)
    Eb_given_z = Eb  # (n, K, D)
    numerator_w = torch.sum(Ez_expanded * Eb_given_z, dim=0)  # (K, D)
    denominator_w = sum_Ez.unsqueeze(1).expand(-1, D)  # (K, D)
    a0_w = a0 * torch.ones_like(numerator_w)
    b0_w = b0 * torch.ones_like(numerator_w)
    w = (numerator_w + a0_w - 1) / (denominator_w + a0_w + b0_w - 2)

    # ---- Update lambda (K, D)
    x_expanded = X.unsqueeze(1).expand(-1,K,-1)  # (n, K, D)
    numerator_lambda = torch.sum(Ez_expanded * Eb_given_z * x_expanded, dim=0)  # (K, D)
    denominator_lambda = torch.sum(Ez_expanded * Eb_given_z, dim=0)  # (K, D)
    e0_lambda = e0 * torch.ones_like(numerator_lambda)
    f0_lambda = f0 * torch.ones_like(numerator_lambda)
    lambda_ = (numerator_lambda + e0_lambda - 1) / (denominator_lambda + f0_lambda)

    # ---- Update parameters
    params_state["pi_K"] = pi
    params_state["omega_KD"] = w
    params_state["lambd_KD"] = lambda_

    return params_state
```

3c) Implement the ELBO.

- As in the derivation part, we encourage you to break down your implementation into the following four pieces:

$$B(q,\Theta) = \mathbb{E}_{q(B,Z)}\left[\log \frac{p(X,B,Z,\Theta \mid \boldsymbol{\eta}_0)}{q(B,Z)}\right] \tag{23}$$

$$= \mathbb{E}_q\Big[\log p(X \mid B,Z,\Theta,\boldsymbol{\eta}_0)\Big] + \mathbb{E}_q\Big[\log p(B,Z \mid \Theta,\boldsymbol{\eta}_0)\Big] - \mathbb{E}_q\Big[\log q(B,Z)\Big] + \log p(\Theta \mid \boldsymbol{\eta}_0) \tag{24}$$

- For your convenience, **we have implemented the last piece** $\log p(\Theta \mid \boldsymbol{\eta}_0)$, which makes use of our PyTorch implementations of the log PDFs of Gamma, Beta, Dirichlet. Your job is to finish the implementation of the three other pieces.
- And again, **do not use for loops.**

In [ ]:
```
def dirichlet_logpdf(x, alpha):
    """
    Compute the log probability density function of the Dirichlet distribution.

    Args:
        x (torch.Tensor): The observed data.  Should be a probability vector (sums to 1).
        alpha (torch.Tensor): The concentration parameters of the Dirichlet distribution.

    Returns:
        torch.Tensor: The log probability density function of the Dirichlet distribution.
    """
    return torch.lgamma(alpha.sum()) - torch.lgamma(alpha).sum() + ((alpha - 1) * torch.log(x)).sum()

def gamma_logpdf(x, concentration, rate):
    """
    Compute the log probability density function of the Gamma distribution.

    Args:
        x (torch.Tensor): The observed data.
        concentration (torch.Tensor): The shape parameter of the Gamma distribution.
        rate (torch.Tensor): The rate parameter (1/scale) of the Gamma distribution.

    Returns:
        torch.Tensor: The log probability density function of the Gamma distribution.
    """
    return (concentration - 1) * torch.log(x) - x * rate - torch.lgamma(concentration) + concentration * torch.log(rate)

def beta_logpdf(x, alpha, beta):
    """
    Compute the log probability density function of the Beta distribution.

    Args:
        x (torch.Tensor): The observed data.  Should be in the range [0, 1].
        alpha (torch.Tensor): The alpha parameter of the Beta distribution.
        beta (torch.Tensor): The beta parameter of the Beta distribution.

    Returns:
        torch.Tensor: The log probability density function of the Beta distribution.
    """
    return torch.lgamma(alpha + beta) - torch.lgamma(alpha) - torch.lgamma(beta) + (alpha - 1) * torch.log(x) + (beta - 1) * torch.log(1 - x)

def elbo(data, params_state, belief_state, hyperparams):
    """
    Compute the Evidence Lower Bound (ELBO) for the Zero-Inflated Poisson Mixture Model.

    Args:
        data (dict): Dictionary containing observed data.
        params_state (dict): Dictionary containing model parameters.
        belief_state (dict): Dictionary containing expectations of latent variables.
        hyperparams (dict): Dictionary containing hyperparameters.

    Returns:
        torch.Tensor: The computed ELBO value.
    """
    n, D = data['X_ND'].shape
    K = hyperparams['alpha_K'].shape[0]
    a0, b0, e0, f0, alpha_K = [hyperparams[key] for key in ['a0', 'b0', 'e0', 'f0', 'alpha_K']]

    # Parameters
    pi_K = params_state['pi_K']  # (K,)
    omega_KD = params_state['omega_KD']  # (K, D)
    lambd_KD = params_state['lambd_KD']  # (K, D)

    # Beliefs
    eta_nK = belief_state['Ezeta_NK']  # (n, K)
    phi_nKD = belief_state['Eb_NKD']  # (n, K, D)

    # Data
    X_ND = data['X_ND']  # (n, D)
    X_NKD = X_ND.unsqueeze(1).expand(-1,K,-1) #n, K, D

    # Log Poisson likelihood for each data point
    log_pois = poisson_logpmf(X_NKD, lambd_KD.unsqueeze(0).expand(n,-1,-1))  # (n, K, D)

    # Log delta term
    log_delta = (X_ND == 0).float().unsqueeze(1) * 0.0 + (X_ND != 0).float().unsqueeze(1) * (-1e10)  # (n, ,1,D)
    log_delta = log_delta.expand(-1, K, -1)  # (n, D, K)

    # Eq[ log p(x | b, z) ]
    Elogpx = eta_nK * (
        (phi_nKD * log_pois + (1 - phi_nKD) * log_delta).sum(dim=2)
    )

    # Eq[ log p(b, z) ]
    log_omega_KD = torch.log(omega_KD).unsqueeze(0).expand(n, -1, -1)  # (n, K, D)
    log_omega_KD2 = torch.log(1-omega_KD).unsqueeze(0).expand(n, -1, -1)  # (n, K, D)
    log_pi_K_extend = torch.log(pi_K).unsqueeze(0).expand(n,-1) # n,K
    term1 = (phi_nKD*log_omega_KD+(1-phi_nKD)*log_omega_KD2).sum(dim=2)
    Elogpbz = eta_nK *(log_pi_K_extend+term1) #n,k

    # Eq[ log q(b, z) ]
    Elogqbz = eta_nK * torch.log(eta_nK) + eta_nK*(phi_nKD*torch.log(phi_nKD)+(1-phi_nKD+1e-10)*torch.log(1-phi_nKD+1e-10)).sum(dim=2)

    # Log prior for model parameters (Theta)
    log_pi_K = dirichlet_logpdf(pi_K, alpha_K)  # log p(pi_K | alpha_K)
    log_omega_KD = beta_logpdf(omega_KD, a0, b0)  # log p(omega_KD | a0, b0)
    log_lambd_KD = gamma_logpdf(lambd_KD, e0, f0)  # log p(lambd_KD | e0, f0)

    # ELBO computation
    elbo_val = Elogpx.sum() + Elogpbz.sum() - Elogqbz.sum()+ log_pi_K.sum() + log_omega_KD.sum() + log_lambd_KD.sum()
    return elbo_val
```

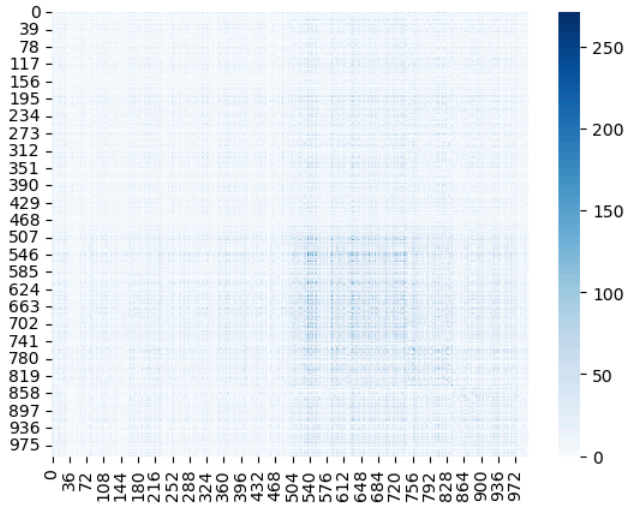
Question 4: Results on "real" data [Code, results]

In this question, you will run your EM algorithm on a data set with provided hyperparameters. First, load in the data and hyperparameters as dictionaries.

In [ ]:
```
import pickle

with open('data.pkl', 'rb') as f:
    data = pickle.load(f)

with open('hyperparams.pkl', 'rb') as f:
    hyperparams = pickle.load(f)
```

The data is synthetic and has been generated by the model itself. We have re-ordered the rows of the data matrix X so that observations that were part of the same (true) cluster are adjacent. (In a realistic setting, we would not know the true clusters and would not be able to order the matrix in this way.) By doing this, we can explore similarity between observations by glimpsing "block structure" in the $n \times n$ matrix $X^\top X$, as below:

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

X_ND = data['X_ND']
_ = sns.heatmap(X_ND @ X_ND.T, cmap='Blues')
```



As you can see above, the block structure is very faint. We can instead use our model to get a clearer picture of the similarity between observations.

4a): Implement the affinity matrix.

In this question, you will run your EM algorithm on the data with the provided hyperparameters. Once converged, you will then use the results to compute the following **affinity matrix** $A \in (0,1)^{n \times n}$, where an entry A_{ij} contains the following posterior expectation:

$$A_{ij} = \mathbb{E}\left[z_i = z_j \mid X, \widehat{\Theta}^{\text{MAP}}\right] = \sum_{k=1}^{K} p(z_i = k, z_j = k \mid X, \widehat{\Theta}^{\text{MAP}})$$

Before running EM, **implement** `compute_affinity_matrix`.

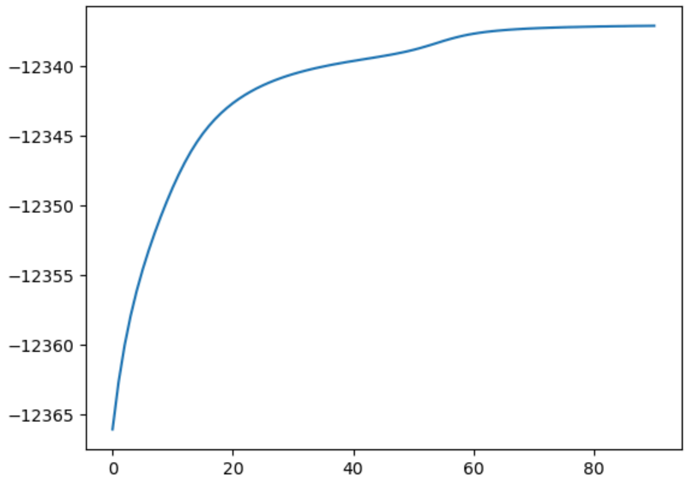
In [ ]:
```
def compute_affinity_matrix(params_state, belief_state):
    """Compute the expectation of the similarity matrix given MAP estimates of parameters.

    Args:
        params_state (dict): Dictionary containing the MAP estimates of the parameters.
        belief_state (dict): Dictionary containing the posterior expectations of latent variables.

    NOTE: Both dictionaries are passed in by default, but you might not need everything in them.

    Returns:
        torch.Tensor: The computed similarity matrix A_NN of shape (n, n), where A_NN[i, j] represents
                      the posterior expectation that observations i and j belong to the same cluster.
    """
    # YOUR CODE HERE
    # A_NN = ...
    eta_nK = belief_state["Ezeta_NK"]  # shape: (n, K)

    # Compute A_NN = eta @ eta^T
    A_NN = torch.matmul(eta_nK, eta_nK.T)  # shape: (n, n)
    return A_NN
```

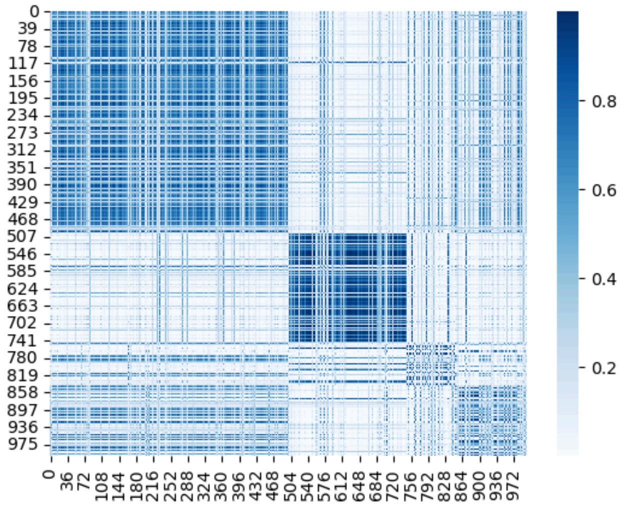
4b): Run EM on the data with the provided hyperparameters.

- Run the cell below.
- The output should display a **heatmap of A and a plot of the ELBO over iteration**.
- **Note:** The `em` function contains an assert statement which will crash if the ELBO ever decreases. **Do not comment this out.** If your ELBO is decreasing, there is a problem with your implementation, and you should return to previous steps and fix the issues.

In [ ]:
```
# run EM
params_state, belief_state, elbo_vals = em(data, hyperparams, max_iter=100, tol=1e-7)

# compute affinity matrix
A_NN = compute_affinity_matrix(params_state, belief_state)

# plot the ELBO and the affinity matrix
plt.plot(elbo_vals[10:]); plt.show()
sns.heatmap(A_NN, cmap='Blues'); plt.show()
```

```
100%|██████████| 100/100 [00:00<00:00, 212.64it/s]
```



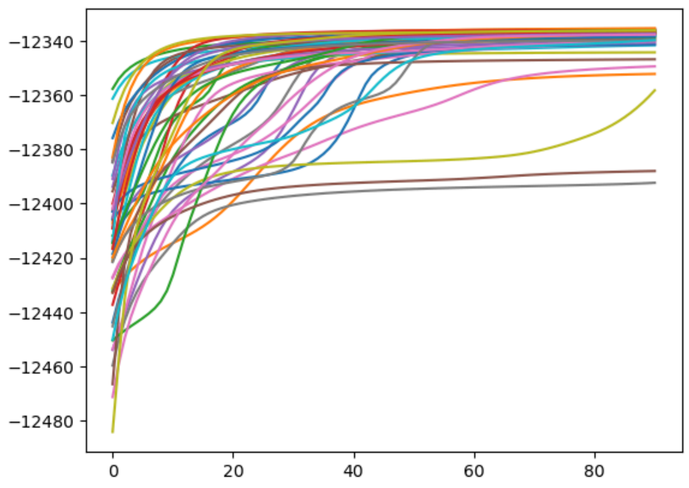


4c): Run EM with random restarts.

- Run the cell below.
- This will run EM fifty times with random restarts.
- Our implementation does this in under a minute. If yours is taking much longer, you should look at the bottlenecks in your code.
- The output should display 50 plots of the ELBO for the 50 random restarts.

In [ ]:
```
best_params_state = {}
best_belief_state = {}
best_elbo_vals = [-torch.inf]
n_random_restarts = 50
for _ in tqdm(range(n_random_restarts)):
    seed = torch.randint(0, 10000, (1,)).item()
    params_state, belief_state, elbo_vals = em(data, hyperparams, max_iter=100, tol=1e-8, seed=seed, verbose=False)
    plt.plot(elbo_vals[10:])
    if elbo_vals[-1] > best_elbo_vals[-1]:
        best_belief_state = belief_state
        best_params_state = params_state
        best_elbo_vals = elbo_vals

plt.show()
```

```
100%|██████████| 50/50 [00:25<00:00,  1.96it/s]
```



Question 5: Matrix completion [Code, results]

In this final problem, you will use your estimate $\widehat{\Theta}^{\text{MAP}}$ to do matrix completion on a test matrix Y.

The test matrix Y is $m \times D$ where the m rows correspond to $i = n, n+1, \ldots, n+m$ new observations.

The first three columns of Y are missing. The full matrix is $Y = [Y^{(\text{obs})}, Y^{(\text{miss})}]$ where $Y^{(\text{obs})}$ is $m \times (D-3)$ and $Y^{(\text{miss})}$ is $m \times 3$.

You will only observe $Y^{(\text{obs})}$. Your job is to use the MAP estimate of the parameters that you obtained in the previous problem (by fitting to X) to calculate the following posterior predictive expectation:

$$\mathbb{E}[Y^{(\text{miss})} \mid Y^{(\text{obs})}, X, \widehat{\Theta}^{\text{MAP}}]$$

- Your solution should not call `m_step` or any other function that updates the parameters.
- **Hint:** Your solution should make use of the `e_step` function you previously implemented. You do not need to change that function at all if you are clever about what input you pass to it.
- **Hint:** We have created for you `params_state_obs`, a dictionary that contains portions of $\widehat{\Theta}^{\text{MAP}}$ which may be useful.
- Your code should ultimately create a tensor called `EY_MD_miss`, which contains the predictions of $Y^{(\text{miss})}$.
- After implementing your solution, run the cell so that it prints the RMSE on the missing portion of the test matrix and compares it to the RMSE of a naive estimator (which is based only on the empirical mean of $Y^{(\text{obs})}$).

In [ ]:
```
with open('test_data.pkl', 'rb') as f:
    test_data = pickle.load(f)

Y_MD_obs = test_data['Y_MD_obs']  # shape (m, D-3)
Y_MD_miss_true = test_data['Y_MD_miss']
m, D1 = Y_MD_obs.shape
D = D1 + 3

K = best_params_state['pi_K'].shape[0]

Y_MD_full = torch.cat([torch.zeros((m, 3)), Y_MD_obs], dim=1)
test_data_dict = {'X_ND': Y_MD_full}

params_state_full = {
    'omega_KD': best_params_state['omega_KD'],
    'lambd_KD': best_params_state['lambd_KD'],
    'pi_K': best_params_state['pi_K']
}

belief_state_test = {
    'Ezeta_NK': torch.ones((m, K)) / K,
    'Eb_NKD': torch.ones((m, K, D)) * 0.75
}

belief_state_test = e_step(test_data_dict, belief_state_test, params_state_full, hyperparams)

Ezeta = belief_state_test['Ezeta_NK']
Eb = belief_state_test['Eb_NKD']
lambda_KD = params_state_full['lambd_KD']

EY_MD = (Ezeta.unsqueeze(2) * Eb * lambda_KD.unsqueeze(0)).sum(dim=1)  # shape (m, D)
EY_MD_miss = EY_MD[:, :3]

rmse_model = torch.sqrt(torch.mean((EY_MD_miss - Y_MD_miss_true)**2))
rmse_naive = torch.sqrt(torch.mean((Y_MD_obs.mean() - Y_MD_miss_true)**2))

print(f'RMSE of model-based estimator: {rmse_model:.3f}')
print(f'RMSE of naive estimator: {rmse_naive:.3f}')
```

```
RMSE of model-based estimator: 1.321
RMSE of naive estimator: 1.329
```

Submission Instructions

Formatting: check that your code does not exceed 80 characters in line width. You can set *Tools → Settings → Editor → Vertical ruler column* to 80 to see when you've exceeded the limit.

Download your notebook in .ipynb format and use the following commands to convert it to PDF. Then run the following command to convert to a PDF:

```
jupyter nbconvert --to pdf <yourlastname>_hw1.ipynb
```

(Note that for the above code to work, you need to rename your file `<yourlastname>_hw1.ipynb`)

Possible causes for errors:

- the "Open in colab" button. Just delete the code that creates this button (go to the top cell and delete it)
- Latex errors: many latex errors aren't visible in the notebook. Try binary search: comment out half of the latex at a time, until you find the bugs

Getting this HW into PDF form isn't meant to be a burden. One quick and easy approach is to open it as a Jupyter notebook, print, save to pdf. Just make sure your latex math answers aren't cut off so we can grade them.

Please post on Ed or come to OH if there are any other problems submitting the HW.

Installing nbconvert:

If you're using Anaconda for package management,

```
conda install -c anaconda nbconvert
```

Upload your .pdf file to Gradescope. Please tag your questions!