Lecture 15: Expectation Maximization & Mixture Models
Notes

Konstantinos Chatzilygeroudis
costashatz@upatras.gr

December 22, 2025

Contents

1 Latent Variable Models

Many signal and data models are naturally described through *unobserved* (latent) variables. These latent variables encode hidden structure such as the regime that generated a measurement, the source identity in a mixture, a cluster label, or a discrete state in a temporal model. A typical motivating example is a *mixture / multi-regime* signal: each observation y_i is produced by one of K underlying sources (or regimes), but the identity of the active source is not directly observed. We represent that hidden identity with a latent variable

$$z_i \in \{1, \dots, K\}.$$

If the latent variables were observed, learning would be straightforward because we could write the *complete-data* model (joint density) in a factorized form:

$$p(y_i, z_i \mid \boldsymbol{\theta}) = p(z_i \mid \boldsymbol{\theta})\, p(y_i \mid z_i, \boldsymbol{\theta}),$$

and for an i.i.d. dataset $(y_1, \dots, y_N)$ the complete-data likelihood becomes

$$p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} p(z_i \mid \boldsymbol{\theta})\, p(y_i \mid z_i, \boldsymbol{\theta}), \qquad \boldsymbol{Y} = \{y_i\}_{i=1}^{N},\ \boldsymbol{Z} = \{z_i\}_{i=1}^{N}.$$

In reality, however, we only observe $\boldsymbol{Y}$, while $\boldsymbol{Z}$ is missing. Therefore we must *marginalize* over the latent variables to obtain the *incomplete-data* likelihood:

$$p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) \qquad (\text{or } \int p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})\, d\boldsymbol{Z} \text{ if } \boldsymbol{Z} \text{ is continuous}).$$

Parameter estimation is then posed as maximum likelihood (ML):

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{Y} \mid \boldsymbol{\theta}) \quad \Leftrightarrow \quad \arg\max_{\boldsymbol{\theta}}\ \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}).$$

The core difficulty appears immediately when we write the incomplete-data log-likelihood:

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \log \sum_{\boldsymbol{Z}} p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}).$$

This is a *log of a sum*, which does not simplify into a sum of logs:

$$\log \sum_{\boldsymbol{Z}} (\cdot) \neq \sum_{\boldsymbol{Z}} \log(\cdot).$$

As a consequence, direct maximization is typically hard: the objective is often non-convex, the marginalization couples variables in an inconvenient way, and gradients tend to involve ratios of sums (numerically delicate and algebraically messy). In short, latent variables make modeling *easier* and more expressive, but they make ML optimization *harder* because we must marginalize over missing structure.

A concrete instance of this difficulty is the mixture likelihood. Suppose each observation comes from one of K components, with mixing probabilities π_k and component likelihoods $p_k(\cdot \mid \boldsymbol{\theta}_k)$:

$$p(z_i = k) = \pi_k, \qquad p(y_i \mid z_i = k, \boldsymbol{\theta}) = p_k(y_i \mid \boldsymbol{\theta}_k).$$

Then the marginal likelihood for a single sample is

$$p(y_i \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \, p_k(y_i \mid \boldsymbol{\theta}_k),$$

and the dataset log-likelihood becomes

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \Big(\sum_{k=1}^{K} \pi_k \, p_k(y_i \mid \boldsymbol{\theta}_k) \Big),$$

which is exactly the "sum inside a log" structure that motivates the Expectation–Maximization (EM) algorithm.

The EM idea is to handle the missing $\boldsymbol{Z}$ by alternating between (i) inferring *soft* latent assignments using the current parameters, and (ii) updating the parameters using those soft assignments. Concretely, EM iterates:

E-step: compute $p(z_i = k \mid y_i, \boldsymbol{\theta})$, **M-step:** update $\boldsymbol{\theta}$ using these assignments.

This alternating strategy converts the difficult marginal-likelihood optimization into a sequence of simpler subproblems that exploit the tractable complete-data factorization.

2 Expectation–Maximization (EM) Algorithm

We now derive the Expectation–Maximization (EM) algorithm as a principled method to maximize the incomplete-data (marginal) log-likelihood in latent-variable models. Recall the setting: we observe $\boldsymbol{Y}$, the latent variables $\boldsymbol{Z}$ are unobserved, and the joint model is $p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})$. The quantity we would like to maximize is the incomplete-data log-likelihood

$$\ell(\boldsymbol{\theta}) \;:=\; \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \log \sum_{\boldsymbol{Z}} p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) \quad (\text{or } \log \int p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) \, d\boldsymbol{Z} \text{ if continuous}).$$

The difficulty is structural: the latent variables appear inside a *log-sum* (or log-integral), which typically prevents closed-form optimization.

2.1 Step 1: Introduce an auxiliary distribution

Let $q(\boldsymbol{Z})$ be *any* distribution over $\boldsymbol{Z}$ with support contained in that of $p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})$ (so that ratios below are well-defined). Insert q into the marginal likelihood:

$$p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}.$$

This expression can be interpreted probabilistically by recalling the definition of expectation with respect to a discrete random variable. If $\boldsymbol{Z}$ is distributed according to $q(\boldsymbol{Z})$, then for any function $g(\boldsymbol{Z})$ we have

$$\mathbb{E}_q[g(\boldsymbol{Z})] = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \, g(\boldsymbol{Z}).$$

By identifying

$$g(\boldsymbol{Z}) = \frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})},$$

the marginal likelihood can therefore be written compactly as

$$p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \mathbb{E}_q\left[\frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right].$$

Taking the logarithm of both sides yields

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \log \mathbb{E}_q\left[\frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right].$$

Thus:

$$p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \mathbb{E}_q\left[\frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right], \qquad \Rightarrow \qquad \ell(\boldsymbol{\theta}) = \log \mathbb{E}_q\left[\frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right].$$

2.2 Step 2: Jensen gives a lower bound (ELBO)

Because $\log(\cdot)$ is concave, Jensen's inequality yields

$$\log \mathbb{E}_q[f(\boldsymbol{Z})] \;\geq\; \mathbb{E}_q[\log f(\boldsymbol{Z})].$$

Apply this with $f(\boldsymbol{Z}) = \dfrac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}$. Then

$$\ell(\boldsymbol{\theta}) = \log \mathbb{E}_q\left[\frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right] \;\geq\; \mathbb{E}_q\left[\log \frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z})}\right].$$

Define the lower bound (also called ELBO in variational inference):

$$\boxed{\mathcal{L}(q, \boldsymbol{\theta}) := \mathbb{E}_q[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})] - \mathbb{E}_q[\log q(\boldsymbol{Z})]\,.}$$

Equivalently, writing expectations as sums (discrete $\boldsymbol{Z}$):

$$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) - \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \log q(\boldsymbol{Z}).$$

This exhibits the two canonical pieces:

$$\underbrace{\mathbb{E}_q[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})]}_{\text{expected complete-data log-likelihood}} \;+\; \underbrace{\mathsf{H}(q)}_{\text{entropy of } q}, \qquad \text{since } \mathsf{H}(q) := -\mathbb{E}_q[\log q(\boldsymbol{Z})].$$

2.3 Step 3: Bound tightness via KL divergence

The bound is not arbitrary; it differs from the true log-likelihood by a KL divergence. Start from Bayes' rule:

$$p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})}{p(\boldsymbol{Y} \mid \boldsymbol{\theta})}.$$

Take logs and rearrange:

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}).$$

Now take expectation with respect to $q(\boldsymbol{Z})$:

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \mathbb{E}_q[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})] - \mathbb{E}_q[\log p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta})].$$

Add and subtract $\mathbb{E}_q[\log q(\boldsymbol{Z})]$:

$$\begin{aligned}\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) &= \Big(\mathbb{E}_q[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})] - \mathbb{E}_q[\log q(\boldsymbol{Z})]\Big) + \Big(\mathbb{E}_q[\log q(\boldsymbol{Z})] - \mathbb{E}_q[\log p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta})]\Big) \\ &= \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q(\boldsymbol{Z}) \parallel p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}))\,,\end{aligned}$$

because

$$\mathrm{KL}(q\|p) = \mathbb{E}_q\left[\log \frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z})}\right] = \mathbb{E}_q[\log q(\boldsymbol{Z})] - \mathbb{E}_q[\log p(\boldsymbol{Z})] \;\geq\; 0.$$

Therefore,

$$\boxed{\mathcal{L}(q, \boldsymbol{\theta}) \leq \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}), \qquad \text{and equality holds iff } q(\boldsymbol{Z}) = p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}) \text{ (a.e.).}}$$

So the best (tightest) bound for a fixed $\boldsymbol{\theta}$ is achieved by choosing q equal to the posterior of the latent variables.

2.4 Step 4: Coordinate ascent on the bound (EM)

EM is simply coordinate ascent on $\mathcal{L}(q, \boldsymbol{\theta})$:

$$\max_{\boldsymbol{\theta}} \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) \quad \text{is approached by iterating} \quad \max_{q} \mathcal{L}(q, \boldsymbol{\theta}) \;\text{ and }\; \max_{\boldsymbol{\theta}} \mathcal{L}(q, \boldsymbol{\theta}).$$

E-step (optimize over q for fixed $\boldsymbol{\theta}^{(t)}$). For fixed $\boldsymbol{\theta}^{(t)}$, we maximize $\mathcal{L}(q, \boldsymbol{\theta}^{(t)})$ w.r.t. q. Using the identity $\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t)}) = \mathcal{L}(q, \boldsymbol{\theta}^{(t)}) + \mathrm{KL}(q\|p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)}))$, and noting the left-hand side does not depend on q, maximizing $\mathcal{L}$ is equivalent to minimizing the KL divergence. The minimum KL is 0, achieved by setting

$$\boxed{q^{(t+1)}(\boldsymbol{Z}) = p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)}).}$$

In mixture models, this corresponds to computing *responsibilities* (soft assignments).

M-step (optimize over $\boldsymbol{\theta}$ for fixed $q^{(t+1)}$). Now fix $q^{(t+1)}$ and maximize the bound over parameters:

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(q^{(t+1)}, \boldsymbol{\theta}).$$

Since the entropy term $-\mathbb{E}_{q^{(t+1)}}[\log q^{(t+1)}(\boldsymbol{Z})]$ does not depend on $\boldsymbol{\theta}$, the M-step reduces to

$$\boxed{\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{q^{(t+1)}}[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})]\,.}$$

It is common to define the *Q-function*:

$$\boxed{Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) := \mathbb{E}_{\boldsymbol{Z} \sim p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)})}\Big[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta})\Big],}$$

so that the M-step is simply

$$\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}).$$

2.5 Monotonic improvement guarantee

EM guarantees that the incomplete-data log-likelihood does not decrease. One clean way to see this is through the bound:

$$\ell(\boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta}) \quad \text{for any } q.$$

At iteration t:

$$\ell(\boldsymbol{\theta}^{(t)}) = \mathcal{L}\Big(q^{(t+1)}, \boldsymbol{\theta}^{(t)}\Big) \quad \text{(E-step makes the bound tight at } \boldsymbol{\theta}^{(t)}).$$

Then the M-step increases (or leaves unchanged) the bound:

$$\mathcal{L}\Big(q^{(t+1)}, \boldsymbol{\theta}^{(t+1)}\Big) \geq \mathcal{L}\Big(q^{(t+1)}, \boldsymbol{\theta}^{(t)}\Big).$$

Finally, since $\ell(\boldsymbol{\theta}^{(t+1)})$ upper-bounds the same quantity,

$$\ell(\boldsymbol{\theta}^{(t+1)}) \geq \mathcal{L}\Big(q^{(t+1)}, \boldsymbol{\theta}^{(t+1)}\Big).$$

Chaining the inequalities yields

$$\boxed{\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t+1)}) \geq \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t)}).}$$

Thus EM performs ascent in the observed-data likelihood, converging to a stationary point (typically a local maximum) in non-convex problems.

2.6 Practical notes

In practice, EM is attractive because the E-step often has a closed form (posterior computations), and the M-step often decomposes into weighted maximum-likelihood updates that resemble the complete-data case. However, because the objective is generally non-convex, initialization matters; common strategies include random restarts, k-means initialization for mixture models, and monitoring convergence via the improvement in $\log p(\boldsymbol{Y} \mid \boldsymbol{\theta})$ (or in $\mathcal{L}(q, \boldsymbol{\theta})$).

3 Gaussian Mixture Models (GMMs)

A Gaussian Mixture Model (GMM) assumes that each observation is generated by first selecting one of K Gaussian components and then sampling from that Gaussian. This introduces a latent discrete assignment variable $z_i \in \{1, \ldots, K\}$ per sample. Let $\boldsymbol{y}_i \in \mathbb{R}^d$ denote the i-th data point and let the full dataset be $\boldsymbol{Y} = \{\boldsymbol{y}_i\}_{i=1}^N$. The model parameters are

$$\boldsymbol{\theta} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K, \qquad \pi_k \geq 0, \ \sum_{k=1}^K \pi_k = 1, \qquad \boldsymbol{\Sigma}_k \succ 0.$$

The generative mechanism is:

$$z_i \sim \text{Categorical}(\pi_1, \ldots, \pi_K), \qquad \boldsymbol{y}_i \mid (z_i = k) \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Equivalently, the joint distribution for a single sample factorizes as

$$p(\boldsymbol{y}_i, z_i = k \mid \boldsymbol{\theta}) = p(z_i = k \mid \boldsymbol{\theta})\, p(\boldsymbol{y}_i \mid z_i = k, \boldsymbol{\theta}) = \pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

3.1 Observed-data (incomplete) likelihood

Since z_i is unobserved, the likelihood of $\boldsymbol{y}_i$ is obtained by marginalization:

$$p(\boldsymbol{y}_i \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} p(\boldsymbol{y}_i, z_i = k \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Thus the incomplete-data log-likelihood of the dataset is

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{y}_i \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \Big(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \Big).$$

This is the characteristic *log-sum* form that prevents direct closed-form maximization.

3.2 Complete-data likelihood (indicator form)

If assignments were known, the likelihood would factor nicely. Define indicator variables

$$\mathbb{I}[z_i = k] = \begin{cases} 1, & z_i = k, \\ 0, & \text{otherwise.} \end{cases}$$

Then the complete-data likelihood can be written as

$$p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \Big(\pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \Big)^{\mathbb{I}[z_i = k]},$$

and the complete-data log-likelihood becomes

$$\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \mathbb{I}[z_i = k] \Big(\log \pi_k + \log \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \Big).$$

EM replaces these hard indicators by their posterior expectations (responsibilities).

3.3 E-step: responsibilities (posterior assignment probabilities)

At iteration t, EM sets

$$q^{(t+1)}(\boldsymbol{Z}) = p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)}).$$

Because the samples are i.i.d. and each z_i only governs $\boldsymbol{y}_i$, this posterior factorizes:

$$p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)}) = \prod_{i=1}^{N} p(z_i \mid \boldsymbol{y}_i, \boldsymbol{\theta}^{(t)}).$$

Define the *responsibility* of component k for point i as

$$\gamma_{ik} := p(z_i = k \mid \boldsymbol{y}_i, \boldsymbol{\theta}^{(t)}).$$

Using Bayes' rule,

$$p(z_i = k \mid \boldsymbol{y}_i, \boldsymbol{\theta}^{(t)}) = \frac{p(z_i = k \mid \boldsymbol{\theta}^{(t)})\, p(\boldsymbol{y}_i \mid z_i = k, \boldsymbol{\theta}^{(t)})}{\sum_{j=1}^{K} p(z_i = j \mid \boldsymbol{\theta}^{(t)})\, p(\boldsymbol{y}_i \mid z_i = j, \boldsymbol{\theta}^{(t)})}.$$

Substituting $p(z_i = k \mid \boldsymbol{\theta}) = \pi_k$ and $p(\boldsymbol{y}_i \mid z_i = k, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ gives

$$\boxed{\gamma_{ik} = \frac{\pi_k^{(t)} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k^{(t)}, \boldsymbol{\Sigma}_k^{(t)})}{\sum_{j=1}^{K} \pi_j^{(t)} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)})}.}$$

These satisfy $\gamma_{ik} \in [0, 1]$ and $\sum_{k=1}^{K} \gamma_{ik} = 1$, i.e., they form a *soft* assignment of each sample to components.

3.4 M-step: maximize the expected complete-data log-likelihood

The M-step maximizes the Q-function

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) := \mathbb{E}_{\boldsymbol{Z} \sim p(\boldsymbol{Z} \mid \boldsymbol{Y}, \boldsymbol{\theta}^{(t)})} \Big[\log p(\boldsymbol{Y}, \boldsymbol{Z} \mid \boldsymbol{\theta}) \Big].$$

Using $\mathbb{E}[\mathbb{I}[z_i = k]] = \gamma_{ik}$,

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_{ik} \Big(\log \pi_k + \log \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \Big).$$

Let

$$N_k := \sum_{i=1}^{N} \gamma_{ik} \qquad \text{(effective number of points assigned to component } k\text{)}.$$

We now maximize Q w.r.t. $\{\pi_k\}$, $\{\boldsymbol{\mu}_k\}$, $\{\boldsymbol{\Sigma}_k\}$.

(1) Update for mixing weights π_k

The π-dependent part is

$$Q_\pi = \sum_{k=1}^{K} N_k \log \pi_k, \qquad \text{subject to } \sum_{k=1}^{K} \pi_k = 1.$$

Using a Lagrangian $\mathcal{J} = \sum_k N_k \log \pi_k + \lambda(\sum_k \pi_k - 1)$, the stationarity condition gives

$$\frac{\partial \mathcal{J}}{\partial \pi_k} = \frac{N_k}{\pi_k} + \lambda = 0 \quad \Rightarrow \quad \pi_k = -\frac{N_k}{\lambda}.$$

Enforcing $\sum_k \pi_k = 1$ yields $\lambda = -N$ (since $\sum_k N_k = \sum_i \sum_k \gamma_{ik} = N$), hence

$$\boxed{\pi_k^{(t+1)} = \frac{N_k}{N}.}$$

(2) Update for means $\boldsymbol{\mu}_k$

The $\boldsymbol{\mu}_k$-dependent part of Q is

$$Q_\mu = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_{ik} \log \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Using the Gaussian log-density and keeping only terms that depend on $\boldsymbol{\mu}_k$,

$$\log \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = -\frac{1}{2}(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^\mathsf{T} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{y}_i - \boldsymbol{\mu}_k) + \text{const},$$

so (up to irrelevant constants)

$$Q_\mu = -\frac{1}{2}\sum_{i=1}^{N} \gamma_{ik}(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^\mathsf{T} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{y}_i - \boldsymbol{\mu}_k) \quad \text{(for each } k \text{ separately)}.$$

Differentiate w.r.t. $\boldsymbol{\mu}_k$ and set to zero:

$$\sum_{i=1}^{N} \gamma_{ik} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{y}_i - \boldsymbol{\mu}_k) = \mathbf{0} \quad \Rightarrow \quad \sum_{i=1}^{N} \gamma_{ik}(\boldsymbol{y}_i - \boldsymbol{\mu}_k) = \mathbf{0},$$

which implies

$$\boxed{\boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik} \boldsymbol{y}_i.}$$

(3) Update for covariances $\boldsymbol{\Sigma}_k$

For the covariance, expand the Gaussian log-density (dropping constants independent of $\boldsymbol{\Sigma}_k$):

$$\log \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = -\frac{1}{2}\log|\boldsymbol{\Sigma}_k| - \frac{1}{2}(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^\mathsf{T} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{y}_i - \boldsymbol{\mu}_k) + \text{const}.$$

Define $\boldsymbol{S}_{ik} := (\boldsymbol{y}_i - \boldsymbol{\mu}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^\mathsf{T}$. Then the $\boldsymbol{\Sigma}_k$-dependent part of Q for a fixed k is

$$Q_{\Sigma_k} = -\frac{1}{2}\sum_{i=1}^{N} \gamma_{ik}\Big(\log|\boldsymbol{\Sigma}_k| + \operatorname{tr}(\boldsymbol{\Sigma}_k^{-1}\boldsymbol{S}_{ik})\Big).$$

Setting the derivative to zero yields the standard weighted sample covariance update:

$$\boxed{\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k}\sum_{i=1}^{N} \gamma_{ik}(\boldsymbol{y}_i - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{y}_i - \boldsymbol{\mu}_k^{(t+1)})^\mathsf{T}.}$$

In implementations, it is common to regularize:

$$\boldsymbol{\Sigma}_k \leftarrow \boldsymbol{\Sigma}_k + \epsilon \boldsymbol{I}, \quad \epsilon > 0,$$

to prevent singular covariances.

3.5 GMM-EM algorithm (summary)

- **Initialize** $\{\pi_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{\Sigma}_k^{(0)}\}_{k=1}^K$ (often via k-means).
- Repeat for $t = 0, 1, 2, \ldots$ until convergence:
  - **E-step:** compute γ_{ik} for all i, k.

– **M-step:** compute $N_k = \sum_i \gamma_{ik}$ and update

$$\pi_k^{(t+1)} = \frac{N_k}{N}, \qquad \boldsymbol{\mu}_k^{(t+1)} = \frac{1}{N_k} \sum_i \gamma_{ik} \boldsymbol{y}_i,$$

$$\boldsymbol{\Sigma}_k^{(t+1)} = \frac{1}{N_k} \sum_i \gamma_{ik} (\boldsymbol{y}_i - \boldsymbol{\mu}_k^{(t+1)})(\boldsymbol{y}_i - \boldsymbol{\mu}_k^{(t+1)})^\mathsf{T}.$$

- **Stop** when $\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t+1)}) - \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t)}) < \delta$.

Each iteration increases (or leaves unchanged) the observed-data likelihood, and the algorithm converges to a stationary point (typically a local maximum) of the GMM log-likelihood.

3.6 Worked EM Examples for GMMs (1D and 2D)

3.6.1 Worked Example 1: 1D GMM with $K = 2$ (one EM iteration)

We illustrate one full EM iteration for a simple one-dimensional mixture of two Gaussians. Assume data points $y_i \in \mathbb{R}$ and the model

$$p(y_i \mid \theta) = \sum_{k=1}^{2} \pi_k \mathcal{N}(y_i \mid \mu_k, \sigma_k^2), \qquad \theta = \{\pi_k, \mu_k, \sigma_k^2\}_{k=1}^2.$$

Consider the dataset

$$\{y_i\}_{i=1}^6 = \{-2.0,\ -1.0,\ -0.5,\ 1.0,\ 2.0,\ 3.0\}.$$

Initialize (at iteration $t = 0$):

$$\pi_1^{(0)} = \pi_2^{(0)} = 0.5, \qquad \mu_1^{(0)} = -1,\ \mu_2^{(0)} = 2, \qquad (\sigma_1^2)^{(0)} = (\sigma_2^2)^{(0)} = 1.$$

E-step (compute responsibilities). For each data point y_i, compute

$$\gamma_{i1} = p(z_i = 1 \mid y_i, \theta^{(0)}) = \frac{\pi_1^{(0)} \mathcal{N}(y_i \mid \mu_1^{(0)}, (\sigma_1^2)^{(0)})}{\pi_1^{(0)} \mathcal{N}(y_i \mid \mu_1^{(0)}, (\sigma_1^2)^{(0)}) + \pi_2^{(0)} \mathcal{N}(y_i \mid \mu_2^{(0)}, (\sigma_2^2)^{(0)})}, \qquad \gamma_{i2} = 1 - \gamma_{i1}.$$

Using $\mathcal{N}(y \mid \mu, 1) \propto \exp\{-\frac{1}{2}(y - \mu)^2\}$ and the equal priors $\pi_1 = \pi_2$, we can compute numerically:

y_i	-2.0	-1.0	-0.5	1.0	2.0	3.0
γ_{i1}	0.9994	0.9890	0.9560	0.1824	0.0110	0.0006
γ_{i2}	0.0006	0.0110	0.0440	0.8176	0.9890	0.9994

(These values come directly from the responsibility formula above.)

Define effective counts

$$N_1 = \sum_{i=1}^{6} \gamma_{i1} \approx 3.1384, \qquad N_2 = \sum_{i=1}^{6} \gamma_{i2} = 6 - N_1 \approx 2.8616.$$

M-step (update parameters). Mixing weights:

$$\pi_1^{(1)} = \frac{N_1}{6} \approx 0.5231, \qquad \pi_2^{(1)} = \frac{N_2}{6} \approx 0.4769.$$

Means:

$$\mu_1^{(1)} = \frac{1}{N_1} \sum_{i=1}^{6} \gamma_{i1} y_i \approx \frac{-3.9810}{3.1384} \approx -1.268,$$

$$\mu_2^{(1)} = \frac{1}{N_2} \sum_{i=1}^{6} \gamma_{i2} y_i \approx \frac{6.9810}{2.8616} \approx 2.439.$$

Variances:

$$(\sigma_1^2)^{(1)} = \frac{1}{N_1} \sum_{i=1}^{6} \gamma_{i1} (y_i - \mu_1^{(1)})^2 \approx 0.547, \qquad (\sigma_2^2)^{(1)} = \frac{1}{N_2} \sum_{i=1}^{6} \gamma_{i2} (y_i - \mu_2^{(1)})^2 \approx 0.818.$$

This completes one EM iteration: E-step computes soft assignments, and M-step updates parameters via responsibility-weighted averages.

Takeaway (1D): EM behaves like "soft k-means" with probabilistic weighting, updating means and variances based on how strongly each point belongs to each component.

3.6.2 Worked Example 2: 2D GMM with $K = 2$ (one EM iteration)

We now illustrate a two-dimensional case where each observation is $\boldsymbol{y}_i \in \mathbb{R}^2$. The model is

$$p(\boldsymbol{y}_i \mid \theta) = \sum_{k=1}^{2} \pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Consider six 2D points forming two clusters:

$$\boldsymbol{y}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{y}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{y}_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{y}_4 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \quad \boldsymbol{y}_5 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}, \quad \boldsymbol{y}_6 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}.$$

Initialize:

$$\pi_1^{(0)} = \pi_2^{(0)} = 0.5, \qquad \boldsymbol{\mu}_1^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \boldsymbol{\mu}_2^{(0)} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \qquad \boldsymbol{\Sigma}_1^{(0)} = \boldsymbol{\Sigma}_2^{(0)} = \boldsymbol{I}.$$

E-step. Responsibilities are

$$\gamma_{i1} = \frac{\pi_1^{(0)} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_1^{(0)}, \boldsymbol{I})}{\pi_1^{(0)} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_1^{(0)}, \boldsymbol{I}) + \pi_2^{(0)} \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_2^{(0)}, \boldsymbol{I})}, \qquad \gamma_{i2} = 1 - \gamma_{i1}.$$

Since both covariances are $\boldsymbol{I}$ and priors are equal, the Gaussian densities are proportional to

$$\exp\left(-\frac{1}{2} \|\boldsymbol{y}_i - \boldsymbol{\mu}_k^{(0)}\|_2^2 \right).$$

Thus points near $(0, 0)$ will have $\gamma_{i1} \approx 1$ and points near $(4, 4)$ will have $\gamma_{i1} \approx 0$. Numerically, the squared distances are:

	$\boldsymbol{y}_1$	$\boldsymbol{y}_2$	$\boldsymbol{y}_3$	$\boldsymbol{y}_4$	$\boldsymbol{y}_5$	$\boldsymbol{y}_6$
$\|\boldsymbol{y}_i - \boldsymbol{\mu}_1^{(0)}\|^2$	0	1	1	32	41	41
$\|\boldsymbol{y}_i - \boldsymbol{\mu}_2^{(0)}\|^2$	32	25	25	0	1	1

From the responsibility formula, for example for $\boldsymbol{y}_2 = (1,0)^\top$:

$$\gamma_{21} = \frac{\exp\left(-\frac{1}{2}\cdot 1\right)}{\exp\left(-\frac{1}{2}\cdot 1\right) + \exp\left(-\frac{1}{2}\cdot 25\right)} \approx 1 - 7\times 10^{-6},$$

so it is essentially assigned to component 1. Similarly, $\boldsymbol{y}_5 = (5,4)^\top$ is essentially assigned to component 2.

Hence, to an excellent approximation in this toy setup,

$$\gamma_{i1} \approx 1 \text{ for } i \in \{1,2,3\}, \qquad \gamma_{i1} \approx 0 \text{ for } i \in \{4,5,6\},$$

and the reverse for γ_{i2}.

M-step. Effective counts:

$$N_1 = \sum_{i=1}^{6} \gamma_{i1} \approx 3, \qquad N_2 \approx 3.$$

Mixing weights:

$$\pi_1^{(1)} \approx \frac{3}{6} = 0.5, \qquad \pi_2^{(1)} \approx 0.5.$$

Means (responsibility-weighted averages):

$$\boldsymbol{\mu}_1^{(1)} \approx \frac{1}{3}\Big(\boldsymbol{y}_1 + \boldsymbol{y}_2 + \boldsymbol{y}_3\Big) = \frac{1}{3}\begin{bmatrix}0+1+0\\0+0+1\end{bmatrix} = \begin{bmatrix}\frac{1}{3}\\\frac{1}{3}\end{bmatrix},$$

$$\boldsymbol{\mu}_2^{(1)} \approx \frac{1}{3}\Big(\boldsymbol{y}_4 + \boldsymbol{y}_5 + \boldsymbol{y}_6\Big) = \frac{1}{3}\begin{bmatrix}4+5+4\\4+4+5\end{bmatrix} = \begin{bmatrix}\frac{13}{3}\\\frac{13}{3}\end{bmatrix}.$$

Covariances:

$$\boldsymbol{\Sigma}_1^{(1)} \approx \frac{1}{3}\sum_{i=1}^{3}(\boldsymbol{y}_i - \boldsymbol{\mu}_1^{(1)})(\boldsymbol{y}_i - \boldsymbol{\mu}_1^{(1)})^\mathsf{T}, \qquad \boldsymbol{\Sigma}_2^{(1)} \approx \frac{1}{3}\sum_{i=4}^{6}(\boldsymbol{y}_i - \boldsymbol{\mu}_2^{(1)})(\boldsymbol{y}_i - \boldsymbol{\mu}_2^{(1)})^\mathsf{T}.$$

(You may optionally regularize $\boldsymbol{\Sigma}_k^{(1)} \leftarrow \boldsymbol{\Sigma}_k^{(1)} + \epsilon \boldsymbol{I}$.)

Takeaway (2D): the E-step compares Gaussian densities using Mahalanobis distances; the M-step updates each component by computing responsibility-weighted sample mean and covariance in $\mathbb{R}^2$.

3.7 Python Implementation

We now describe how the EM algorithm for Gaussian Mixture Models is implemented in practice, and how each part of the code directly corresponds to the mathematical steps derived above. The goal of this subsection is not to introduce new theory, but to *connect equations to executable code*, so that the algorithmic flow of EM becomes completely transparent.

Overall structure. A typical GMM–EM implementation follows the same high-level loop:

`initialize parameters` $\rightarrow$ `repeat E-step and M-step` $\rightarrow$ `check convergence`.

In code, this is usually a `for`- or `while`-loop over EM iterations. The parameters stored and updated at each iteration are

$$\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K},$$

and the intermediate quantities computed in the E-step are the responsibilities

$$\gamma_{ik} = p(z_i = k \mid \boldsymbol{y}_i, \boldsymbol{\theta}).$$

Initialization. The code typically begins by initializing:

- mixing weights $\pi_k^{(0)}$ (often uniformly, $\pi_k = 1/K$),
- means $\boldsymbol{\mu}_k^{(0)}$ (random samples from data or k-means centroids),
- covariances $\boldsymbol{\Sigma}_k^{(0)}$ (identity matrices, diagonal matrices, or empirical covariances).

This corresponds exactly to choosing the starting point $\boldsymbol{\theta}^{(0)}$ for EM. As emphasized earlier, initialization is crucial due to non-convexity.

E-step in code (responsibility computation). The E-step computes the posterior assignment probabilities

$$\gamma_{ik} = \frac{\pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

In code, this usually appears as:

- a loop (or vectorized operation) over components k computing Gaussian densities,
- multiplication by mixing weights π_k,
- normalization across components so that $\sum_k \gamma_{ik} = 1$.

Numerically stable implementations compute these quantities in log-space (using `log-sum-exp`) to avoid underflow, especially in higher dimensions.

Effective counts. Once responsibilities are available, the code computes

$$N_k = \sum_{i=1}^{N} \gamma_{ik},$$

which represents the *effective number of samples* assigned to component k. This quantity appears repeatedly in the M-step updates and is typically computed once per iteration.

M-step in code (parameter updates). The M-step corresponds to maximizing the Q-function and is implemented via closed-form updates:

- **Mixing weights:**
$$\pi_k \leftarrow \frac{N_k}{N}.$$
- **Means:**
$$\boldsymbol{\mu}_k \leftarrow \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \boldsymbol{y}_i.$$
In code, this is a weighted average of data points.
- **Covariances:**
$$\boldsymbol{\Sigma}_k \leftarrow \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} (\boldsymbol{y}_i - \boldsymbol{\mu}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^\mathsf{T}.$$
This corresponds to computing a responsibility-weighted sample covariance matrix.

Many implementations add a small regularization term $\epsilon \boldsymbol{I}$ to $\boldsymbol{\Sigma}_k$ to ensure numerical stability and invertibility.

Log-likelihood evaluation and convergence. After each EM iteration, the code often evaluates the incomplete-data log-likelihood

$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log \Big(\sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \Big),$$

which serves two purposes:

- verifying the monotonic increase guaranteed by EM,
- providing a stopping criterion, e.g.,
$$\log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t+1)}) - \log p(\boldsymbol{Y} \mid \boldsymbol{\theta}^{(t)}) < \delta.$$

Connection to the worked examples. The 1D and 2D worked examples presented earlier are exactly what the code is doing numerically:

- computing distances (scalar or Mahalanobis) inside Gaussian densities,
- converting them into soft assignments (responsibilities),
- updating means as weighted averages and covariances as weighted scatter matrices.

The only difference in higher dimensions or larger datasets is scale and numerical care; conceptually, the algorithm remains identical.

Key takeaway. The EM algorithm for GMMs is not a black box: each line of code corresponds directly to a specific probabilistic or optimization step derived earlier. Understanding this mapping makes it easier to debug implementations, modify covariance structures (diagonal, spherical), and extend GMMs to more complex latent-variable models.

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

# ----------------------------
# Generate 1D mixture data
# ----------------------------
N = 600
true_pi = np.array([0.65, 0.35])
true_mu = np.array([-1.2, 1.8])
true_sigma = np.array([0.45, 0.7])

z = np.random.choice([0, 1], size=N, p=true_pi)
y = np.random.randn(N) * true_sigma[z] + true_mu[z]
y = y.reshape(-1, 1)  # (N,1) for uniformity

# ----------------------------
# Helper: 1D Gaussian pdf
# ----------------------------
def gauss_pdf_1d(x, mu, var):
    return (1.0 / np.sqrt(2*np.pi*var)) * np.exp(-0.5*(x-mu)**2 / var)

# ----------------------------
# EM for 1D GMM (K=2)
# ----------------------------
K = 2
# init (simple): random means from data, equal weights, same variances
mu = np.random.choice(y.flatten(), K, replace=False)
var = np.full(K, np.var(y))
pi = np.full(K, 1.0/K)

def e_step(y, mu, var, pi):
    N = y.shape[0]
    gamma = np.zeros((N, K))
    for k in range(K):
        gamma[:, k] = pi[k] * gauss_pdf_1d(y.flatten(), mu[k], var[k])
    gamma /= np.sum(gamma, axis=1, keepdims=True)
    return gamma

def m_step(y, gamma):
    Nk = np.sum(gamma, axis=0)
    pi_new = Nk / y.shape[0]
    mu_new = np.sum(gamma * y, axis=0) / Nk
    var_new = np.sum(gamma * (y - mu_new)**2, axis=0) / Nk
    return mu_new, var_new, pi_new

# EM loop
max_iter = 13
for t in range(max_iter):
    gamma = e_step(y, mu, var, pi)
    mu, var, pi = m_step(y, gamma)
```

```
53
54      if t % 2 == 0 or t == max_iter - 1:
55          print(f"Iteration {t+1}: mu={mu}, var={var}, pi={pi}")
56          # ---------------------------
57          # Plot fitted model
58          # ---------------------------
59          x = np.linspace(y.min()-1, y.max()+1, 1200)
60
61          p1 = pi[0] * gauss_pdf_1d(x, mu[0], var[0])
62          p2 = pi[1] * gauss_pdf_1d(x, mu[1], var[1])
63          pmix = p1 + p2
64
65          # responsibilities as function of x (for component 1)
66          gamma1_x = p1 / (pmix + 1e-12)
67          gamma2_x = p2 / (pmix + 1e-12)
68
69          plt.figure(figsize=(7,4))
70
71          plt.hist(y.flatten(), bins=40, density=True, alpha=0.35, label=
                "Data histogram")
72          plt.plot(x, p1, linewidth=2, label="Fitted component 1")
73          plt.plot(x, p2, linewidth=2, label="Fitted component 2")
74          plt.plot(x, pmix, linewidth=2, linestyle="--", label="Fitted
                mixture")
75
76          # scale responsibility to sit on same axis nicely
77          plt.plot(x, gamma1_x * pmix.max(), linewidth=2, linestyle="-.",
78                   label=r"Responsibility $\gamma_{1}(y)$ (scaled)")
79          plt.plot(x, gamma2_x * pmix.max(), linewidth=2, linestyle="-.",
80                   label=r"Responsibility $\gamma_{2}(y)$ (scaled)")
81
82          plt.xlabel(r"$y$")
83          plt.ylabel("Density / scaled responsibility")
84          plt.title("1D GMM fit with EM (K=2) - Iteration " + str(t))
85          plt.legend()
86          plt.tight_layout()
87          plt.show()
```

Listing 1: GMMs in 1D

3.8 Python Implementation of GMMs in 2D

We now walk through a concrete Python implementation of EM for a 2D Gaussian Mixture Model (GMM) with $K = 3$ components. The purpose of this example is to show how the theoretical EM steps map *line-by-line* to code: (i) generate mixture data, (ii) implement the Gaussian density, (iii) implement E-step and M-step, and (iv) visualize convergence through fitted contours and estimated means.

3.8.1 Step 0: Generating synthetic data from a 2D mixture

The script first creates a dataset $\boldsymbol{Y} \in \mathbb{R}^{N\times 2}$ sampled from a known (ground-truth) mixture model with $N = 800$ and $K = 3$:

$$p(\boldsymbol{y}) = \sum_{k=1}^{3} \pi_k^\star \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\mu}_k^\star, \boldsymbol{\Sigma}_k^\star).$$

In code:

- `true_pi` stores $\boldsymbol{\pi}^\star = (0.4, 0.35, 0.25)$,
- `true_mu` stores the true component means $\boldsymbol{\mu}_k^\star \in \mathbb{R}^2$,
- `true_Sigma` stores the true covariance matrices $\boldsymbol{\Sigma}_k^\star \in \mathbb{R}^{2\times 2}$.

Then, latent assignments are sampled:

$$z_i \sim \text{Categorical}(\pi_1^\star, \pi_2^\star, \pi_3^\star),$$

implemented by:

`z = np.random.choice(np.arange(K), size=N, p=true_pi)`.

Conditioned on $z_i = k$, the data are sampled from $\mathcal{N}(\boldsymbol{\mu}_k^\star, \boldsymbol{\Sigma}_k^\star)$:

$$\boldsymbol{y}_i \mid (z_i = k) \sim \mathcal{N}(\boldsymbol{\mu}_k^\star, \boldsymbol{\Sigma}_k^\star).$$

In the script, this is implemented using `np.random.multivariate_normal` and concatenated into the final data matrix $\boldsymbol{Y}$.

Why two seeds? The outer loop `for seed in [1, 78]` repeats the experiment with two random initializations. Since GMM likelihood is non-convex, EM can converge to different local maxima depending on initialization. Running multiple seeds is a minimal demonstration of this phenomenon.

3.8.2 Step 1: Implementing the multivariate Gaussian density

The helper function `gauss_pdf_nd(X, mu, Sigma)` computes

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\right),$$

for all rows $\boldsymbol{x}$ in a matrix $X \in \mathbb{R}^{N\times d}$ (here $d = 2$).

The code computes:

- `diff = X - mu` $\leftrightarrow$ $(\boldsymbol{x} - \boldsymbol{\mu})$ for each row,
- `invS = np.linalg.inv(Sigma)` $\leftrightarrow$ $\boldsymbol{\Sigma}^{-1}$,
- `expo = ...` computes the quadratic form

$$(\boldsymbol{x}-\boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu}),$$

  vectorized using `einsum`:

`np.einsum("ni,ij,nj->n", diff, invS, diff)`.

- `norm = sqrt((2pi)^d det(Sigma))` computes the normalization constant.

Finally, it returns `exp(-0.5*expo)/norm` for all rows.

Numerical note. This implementation computes densities in standard space. In higher dimensions or with many iterations, it is often more stable to compute responsibilities using log-densities and the `log-sum-exp` trick. For $d = 2$, this explicit form is usually fine.

3.8.3 Step 2: EM initialization

Before running EM, the parameters are initialized as:

$$\pi_k^{(0)} = \frac{1}{K}, \qquad \boldsymbol{\mu}_k^{(0)} = \text{randomly chosen data points}, \qquad \boldsymbol{\Sigma}_k^{(0)} = \widehat{\text{Cov}}(\boldsymbol{Y}) \text{ (shared, copied for all } k).$$

In code:

- `mu = Y[np.random.choice(N, K, replace=False)]` initializes means by sampling K distinct points from the dataset.
- `Sigma = np.array([np.cov(Y.T) for _ in range(K)])` sets all covariances to the empirical covariance of the entire dataset at initialization.
- `pi = np.full(K, 1.0/K)` initializes uniform mixing weights.

This is a common baseline initialization, though k-means is often superior in practice.

3.8.4 Step 3: E-step in code (responsibilities)

The E-step computes responsibilities

$$\gamma_{ik} = p(z_i = k \mid \boldsymbol{y}_i, \boldsymbol{\theta}) = \frac{\pi_k \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{y}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}.$$

The function `e_step(Y, mu, Sigma, pi)` implements this as follows:

- It builds an $N \times K$ matrix `gamma` with entries

  `gamma[:,k] = pi[k] * gauss_pdf_nd(Y, mu[k], Sigma[k])`.

- Then it normalizes each row to sum to 1:

  `gamma /= np.sum(gamma, axis=1, keepdims=True)`.

After normalization, each row $(\gamma_{i1}, \ldots, \gamma_{iK})$ forms a categorical posterior distribution for z_i.

3.8.5 Step 4: M-step in code (closed-form updates)

The M-step maximizes the expected complete-data log-likelihood and yields the standard closed-form GMM updates:

$$N_k = \sum_{i=1}^{N} \gamma_{ik}, \qquad \pi_k \leftarrow \frac{N_k}{N}, \qquad \boldsymbol{\mu}_k \leftarrow \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \boldsymbol{y}_i,$$

$$\boldsymbol{\Sigma}_k \leftarrow \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \, (\boldsymbol{y}_i - \boldsymbol{\mu}_k)(\boldsymbol{y}_i - \boldsymbol{\mu}_k)^{\mathsf{T}}.$$

The function `m_step(Y, gamma)` implements exactly these:

- `Nk = np.sum(gamma, axis=0)` computes $(N_1, \ldots, N_K)$.

- `pi_new = Nk / Y.shape[0]` updates mixing weights.
- `mu_new = (gamma.T @ Y) / Nk[:, None]` updates means via weighted averages.
- For covariances, for each component k:
  - `diff = Y - mu_new[k]` corresponds to $(\boldsymbol{y}_i - \boldsymbol{\mu}_k)$,
  - `Sk = (gamma[:,k][:,None] * diff).T @ diff / Nk[k]` computes the weighted scatter matrix divided by N_k, i.e.

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{i=1}^{N} \gamma_{ik} \, \texttt{diff}_i \, \texttt{diff}_i^{\mathsf{T}}.$$

3.8.6 Step 5: Iteration, logging, and visualization

The loop

```
for t in range(max_iter):
```

runs EM for `max_iter = 121` iterations, alternating:

$$\gamma \leftarrow \texttt{e\_step}(\cdot), \qquad (\mu, \Sigma, \pi) \leftarrow \texttt{m\_step}(\cdot).$$

Every 20 iterations (and at the final iteration), the code prints the current estimates of (μ, π) and generates a plot:

- Scatter plot of the raw data in $\mathbb{R}^2$.
- Contours of each fitted Gaussian component, computed on a grid:

$$Z_k(\boldsymbol{y}) = \pi_k \, \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

  and plotted with `plt.contour`.
- The current means $\boldsymbol{\mu}_k$ are shown as large `x` markers.

This visualization is pedagogically useful: as EM iterates, the means move toward dense regions of the data and covariances rotate/scale to match the shape of clusters.

3.8.7 Summary: what this code demonstrates

This code example demonstrates three key points about EM for GMMs:

1. **E-step = soft clustering:** responsibilities γ_{ik} assign each point fractionally to each component.
2. **M-step = weighted Gaussian fitting:** each component is updated using responsibility-weighted means and covariances.
3. **Initialization sensitivity:** different seeds can lead to different local optima or different convergence speeds, even on the same dataset.

In short, the script is a direct computational instantiation of the EM derivation: it alternates between computing the posterior over latent assignments and re-estimating mixture parameters from those posteriors, while providing a geometric visualization (contours) of how the fitted density evolves over iterations.

```python
import numpy as np
import matplotlib.pyplot as plt

for seed in [1, 78]:
    np.random.seed(seed)

    # ----------------------------
    # Generate 2D mixture data
    # ----------------------------
    N = 800
    K = 3

    true_pi = np.array([0.4, 0.35, 0.25])
    true_mu = np.array([
        [-2.0, -1.0],
        [ 1.5,  0.7],
        [ 0.0,  2.5]
    ])
    true_Sigma = np.array([
        [[0.6, 0.2],[0.2, 0.5]],
        [[0.5,-0.1],[-0.1,0.4]],
        [[0.3, 0.0],[0.0, 0.8]]
    ])

    z = np.random.choice(np.arange(K), size=N, p=true_pi)
    Y = np.vstack([
        np.random.multivariate_normal(true_mu[k], true_Sigma[k], size=(
            z==k).sum())
        for k in range(K)
    ])

    # ----------------------------
    # Helpers: multivariate Gaussian pdf
    # ----------------------------
    def gauss_pdf_nd(X, mu, Sigma):
        d = X.shape[1]
        diff = X - mu
        invS = np.linalg.inv(Sigma)
        expo = np.einsum("ni,ij,nj->n", diff, invS, diff)
        norm = np.sqrt((2*np.pi)**d * np.linalg.det(Sigma))
        return np.exp(-0.5*expo) / norm

    # ----------------------------
    # EM for 2D GMM
    # ----------------------------
    # init: random means from data, equal weights, shared covariance
    mu = Y[np.random.choice(N, K, replace=False)]
    Sigma = np.array([np.cov(Y.T) for _ in range(K)])
    pi = np.full(K, 1.0/K)

    def e_step(Y, mu, Sigma, pi):
        N = Y.shape[0]
        gamma = np.zeros((N, K))
        for k in range(K):
            gamma[:, k] = pi[k] * gauss_pdf_nd(Y, mu[k], Sigma[k])
        gamma /= np.sum(gamma, axis=1, keepdims=True)
        return gamma
```

```
57
58      def m_step(Y, gamma):
59          Nk = np.sum(gamma, axis=0)
60          pi_new = Nk / Y.shape[0]
61          mu_new = (gamma.T @ Y) / Nk[:, None]
62          Sigma_new = []
63          for k in range(K):
64              diff = Y - mu_new[k]
65              Sk = (gamma[:, k][:, None] * diff).T @ diff / Nk[k]
66              Sigma_new.append(Sk)
67          return mu_new, np.array(Sigma_new), pi_new
68
69      max_iter = 121
70      for t in range(max_iter):
71          gamma = e_step(Y, mu, Sigma, pi)
72          mu, Sigma, pi = m_step(Y, gamma)
73
74          if t % 20 == 0 or t == max_iter - 1:
75              print(f"Iteration {t}: mu={mu}, pi={pi}")
76              # ----------------------------
77              # Plot data + fitted contours
78              # ----------------------------
79              plt.figure(figsize=(6.5,5))
80              plt.scatter(Y[:,0], Y[:,1], s=12, alpha=0.5, label="Data")
81
82              # grid for contours
83              x1 = np.linspace(Y[:,0].min()-1, Y[:,0].max()+1, 250)
84              x2 = np.linspace(Y[:,1].min()-1, Y[:,1].max()+1, 250)
85              X1, X2 = np.meshgrid(x1, x2)
86              grid = np.column_stack([X1.ravel(), X2.ravel()])
87
88              # plot each fitted Gaussian contour
89              for k in range(K):
90                  Zk = pi[k] * gauss_pdf_nd(grid, mu[k], Sigma[k]).
                        reshape(X1.shape)
91                  plt.contour(X1, X2, Zk, levels=6, linewidths=2)
92
93              # mark means
94              plt.scatter(mu[:,0], mu[:,1], s=120, marker="x", linewidths
                    =3, label="Fitted means")
95
96              plt.xlabel(r"$y_1$")
97              plt.ylabel(r"$y_2$")
98              plt.title("2D GMM fit with EM (K=3): contours + means -
                    Iteration " + str(t))
99              plt.legend()
100             plt.tight_layout()
101             plt.show()
```

Listing 2: GMMs in 2D