

Mixture Models and the EM Algorithm

Lecture Notes Consolidated with Claude AI — CS6140 Machine Learning

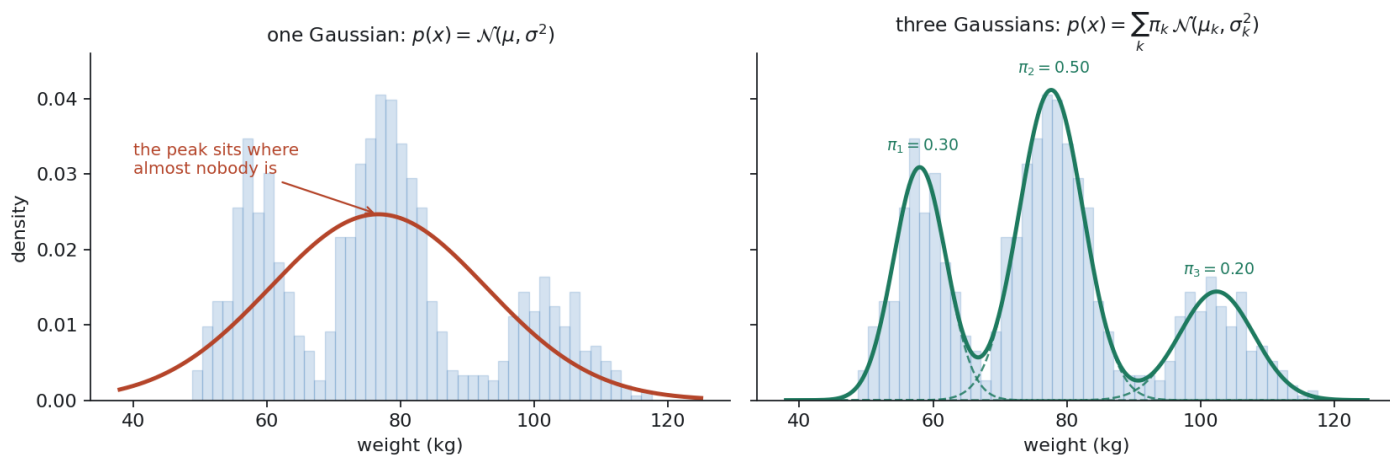
1 Mixture Models

Two things to set up before any algorithm: why one Gaussian is sometimes hopeless, and what exactly goes missing from the data the moment we use several.

1.1 One Bump Is Not Enough

The previous note fit *one* Gaussian per class. That works when a class really is one blob. It fails — quietly, and sometimes badly — when it is not.

Take the weight of adults in a mixed population: children, adult women, adult men. Plot the histogram and you see three humps. Fit a single $\mathcal{N}(\mu, \sigma^2)$ to it and maximum likelihood does exactly what you asked: it finds the one bell curve closest to the whole pile. The result places its peak in a valley between humps and assigns high density to weights that are in fact uncommon.



Left: the best single Gaussian. It is not a bad fit by any mechanical criterion — it has the right mean and the right variance — but its mode sits where the data has a dip. **Right:** three Gaussians, each with its own μ_k, σ_k , added together with weights π_k that sum to 1. The recovered weights 0.30/0.50/0.20 match the three sub-populations the data was built from (300/500/200 points).

The fix is to let the density be a weighted sum of simple pieces:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k p_k(\mathbf{x} \mid \theta_k), \quad \pi_k \geq 0, \quad \sum_{k=1}^K \pi_k = 1. \quad (1)$$

This is a **mixture model**. The π_k are the **mixing weights** (how much of the population each piece accounts for) and the p_k are the **components**. If the components are Gaussians we call it a **Gaussian**

Mixture Model (GMM); if they are Binomials, a mixture of Binomials (§7.1); Poissons, Bernoullis, anything — the machinery below does not care.

Key point: Equation 1 is still a density: non-negative, integrates to 1 (because each p_k does and the π_k sum to 1). So everything the GDA_NB note did with a density you can now do with a mixture — including plugging it into Bayes’ rule as $p(\mathbf{x} \mid y = c)$ to get a classifier whose per-class density has several bumps. What changes is only how we *fit* it, and that is the subject of this whole note.

Remark. Mixtures are used in two quite different ways, and it is worth keeping them apart. (i) *As a flexible density:* you do not believe in the components, you just want a curve that tracks the data, and K is a knob. (ii) *As a model of real sub-populations:* you believe there genuinely are K kinds of thing and you want to find out which point is which. Recovering the parameters of a known generator, as in §9.1, is case (ii). Most applications are case (i), where “did I get the right components?” is not even a meaningful question.

1.2 The Generative Story, and the Variable We Cannot See

Equation 1 is an algebraic statement. It also has a procedural reading, and the procedural one is what makes EM possible. To draw one datapoint from a mixture:

The mixture’s generative recipe

1. Roll a K -sided die with face probabilities $\pi_1, \dots, \pi_K$; say it comes up k .
 2. Draw $\mathbf{x}_i \sim p_k(\cdot \mid \theta_k)$ — i.e. from the component the die chose.
 3. **Throw away which face came up.** Record only $\mathbf{x}_i$.
-

Step 3 is the whole problem. The recipe says $\mathbf{x}_i$ came from exactly one component, but the dataset does not say which. That “which” is a **latent** (hidden) variable: part of the model, never observed. Write it as an **indicator**, one bit per (point, component) pair:

$$z_{ik} = \begin{cases} 1 & \text{if datapoint } i \text{ came from component } k, \\ 0 & \text{otherwise,} \end{cases} \quad \text{so} \quad \sum_{k=1}^K z_{ik} = 1 \quad \text{for every } i. \quad (2)$$

This indicator is the *only* latent symbol in the note; everything else is built from it. Two sum identities get used constantly below, and both follow from Eq. 2:

$$\sum_k z_{ik} = 1 \quad (\text{each point is in exactly one component}), \quad \sum_i z_{ik} = N_k \quad (\text{the size of component } k).$$

With the indicator, the mixture density is a *marginal*: start from the joint over the observed and the hidden,

$$p(\mathbf{x}_i, z_{ik}=1) = \underbrace{p(z_{ik}=1)}_{\pi_k} \underbrace{p(\mathbf{x}_i \mid z_{ik}=1)}_{p_k(\mathbf{x}_i \mid \theta_k)}, \quad (3)$$

and sum the hidden variable away: $p(\mathbf{x}_i) = \sum_k p(\mathbf{x}_i, z_{ik}=1) = \sum_k \pi_k p_k(\mathbf{x}_i \mid \theta_k)$, which is Eq. 1 again.

Key point: A mixture is not a new kind of object — it is an ordinary joint distribution over $(\mathbf{x}, z)$ with z summed out. Everything hard about fitting mixtures traces back to that sum: the logarithm of a sum does not split, so the log-likelihood does not decompose into per-component pieces the way every maximum-likelihood problem so far in this course has.

Concretely, the log-likelihood we actually want to maximize is

$$\ell(\theta) = \sum_{i=1}^N \log p(\mathbf{x}_i) = \sum_{i=1}^N \log \left(\sum_{k=1}^K \pi_k p_k(\mathbf{x}_i | \theta_k) \right). \quad (4)$$

Compare this with the fully-observed case. If somebody handed us the true z_{ik} 's, the data would split into K labelled groups and we would fit each component separately — exactly the per-class fit of the `GDA_NB` note, nothing new. The $\log \sum$ in Eq. 4 is the price of not knowing them. Setting $\partial \ell / \partial \mu_k = 0$ gives an equation in which μ_k appears on both sides, and no closed-form solution falls out.

2 Chicken and Egg: k -Means as EM

There is a chicken-and-egg problem at the centre of this, and seeing it clearly is most of the work.

If we knew the parameters, the memberships would be easy. Given the component means $\mu_1, \dots, \mu_K$, deciding which component produced $\mathbf{x}_i$ is just a matter of looking at which one it sits near. Call this computing the memberships from the parameters the **E step**.

If we knew the memberships, the parameters would be easy. Given every z_{ik} , the data splits into K labelled groups and each μ_k is the average of its own group — the ordinary fit from `GDA_NB`, done K times. Call this computing the parameters from the memberships the **M step**.

We know neither. But each would be easy *given* the other, so: guess one, and alternate. That is the entire algorithm, and every remaining section of this note is one of those two steps done properly.

Key point: E step: parameters $\rightarrow$ memberships. **M step:** memberships $\rightarrow$ parameters. Alternate. Everything that follows — the Bayes' rule in §3, the three weighted averages in §4.2, even the ELBO in §6.1 — is detail hung on these two sentences. Write them on a corner of the page and keep them there.

2.1 The familiar case: hard k -means

You have already run this algorithm. Make both steps as crude as possible — memberships that are 0 or 1, components that are only means — and the two steps are the two halves of k -means:

k -means: the two steps, at their crudest

Initialize centres $\mu_1, \dots, \mu_K$ (e.g. K random datapoints). Repeat until nothing moves:

1. **E step.** $z_{ik} = 1$ for $k = \arg \min_{k'} \|\mathbf{x}_i - \mu_{k'}\|^2$, and 0 otherwise — each point goes wholly to its nearest centre.
2. **M step.** $\mu_k \leftarrow \frac{1}{N_k} \sum_{i: z_{ik}=1} \mathbf{x}_i = \frac{\sum_i z_{ik} \mathbf{x}_i}{\sum_i z_{ik}}$ — each centre moves to the average of the points it owns.

Remark. Strictly, calling k -means’s step 1 an “*expectation* step” is generous: the membership it produces is a 0 or a 1, so the expectation is degenerate — $\mathbb{E}[z_{ik}]$ is just z_{ik} itself. That degeneracy is exactly why this is the easy case, and why it needs no probability at all. The real E step (§3) returns a genuine average instead, and that single change is the whole difference between k -means and EM.

k -means is a complete algorithm and it works. Its weakness is visible in one picture: a point sitting exactly between two centres is forced to commit, and the centre it commits to is then pulled by it as if the point belonged there with certainty. Information is being discarded.

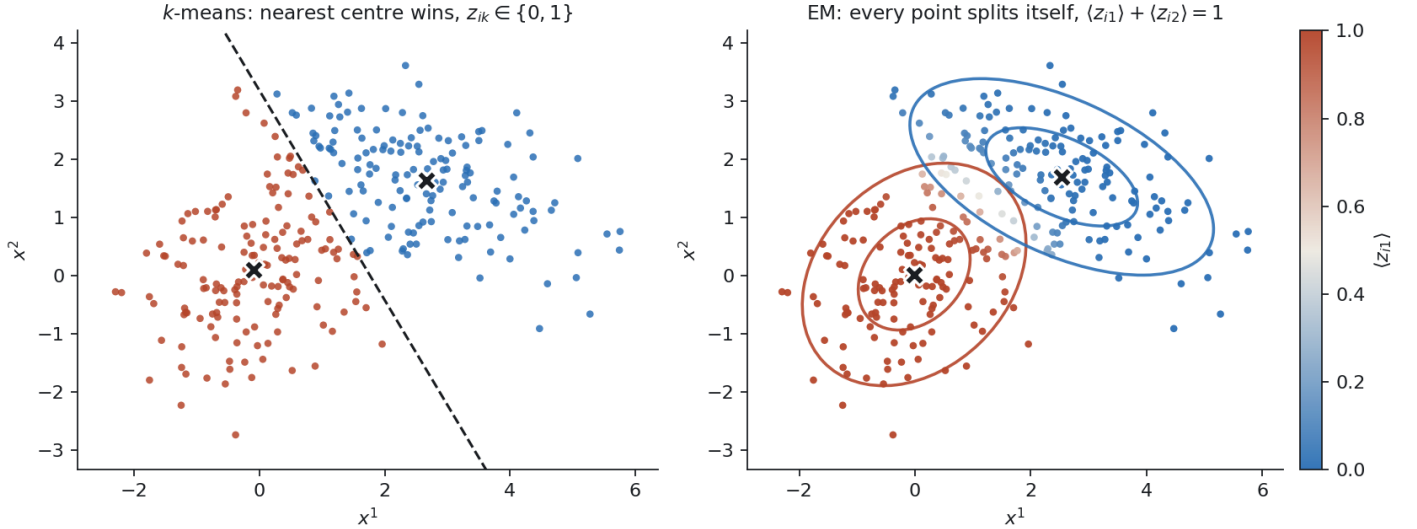
2.2 Softening the E step

Soft k -means changes the E step and nothing else: replace the hard 0/1 membership with a number between 0 and 1, obtained by exponentiating the negative distance.

$$\text{E step:} \quad \langle z_{ik} \rangle = \frac{\exp(-\|\mathbf{x}_i - \mu_k\|^2/2\sigma^2)}{\sum_{\ell=1}^K \exp(-\|\mathbf{x}_i - \mu_\ell\|^2/2\sigma^2)}, \quad \sum_k \langle z_{ik} \rangle = 1. \quad (5)$$

$$\text{M step:} \quad \mu_k \leftarrow \frac{\sum_i \langle z_{ik} \rangle \mathbf{x}_i}{\sum_i \langle z_{ik} \rangle}$$

The M step is the same average as before, now *weighted*: each point contributes to every centre, in proportion to how plausible that centre is for it.



The same 300 points, deliberately overlapping. **Left:** k -means draws a single line (the perpendicular bisector of the two centres) and every point on one side is wholly blue. **Right:** EM gives the points in the contested band intermediate responsibilities — the pale points have $\langle z_{i1} \rangle$ near 0.5 — and they pull on both components in proportion. The fitted components also have their own shapes, which k -means has no notion of.

Key point: Look again at Eq. 5: the numerator $\exp(-\|\mathbf{x}_i - \mu_k\|^2/2\sigma^2)$ is an unnormalized spherical Gaussian density, and the denominator is the sum over components. So soft k -means already *is* a Gaussian mixture, with every $\Sigma_k = \sigma^2 I$ and every $\pi_k = 1/K$. The next two sections put the missing pieces back: a real E step (§3) that also accounts for component shape and size, and a real M step (§4.2) that estimates the covariances and the mixing weights rather than fixing them.

Remark. Running the limit the other way is also instructive. In Eq. 5, let $\sigma^2 \rightarrow 0$. The largest exponent dominates completely and $\langle z_{ik} \rangle \rightarrow$ the hard 0/1 assignment. So k -means is the zero-variance limit of EM on a GMM — which is why it is often used to *initialize* EM (§5). The reverse reading is the useful one for this note: EM is k -means that knows how uncertain it is.

3 The E Step: Bayes’ Rule

Now do it properly. Fix the current parameters $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ and ask: given that I observed $\mathbf{x}_i$, what is the probability it came from component k ? This is a posterior over the latent variable, and Bayes’ rule answers it immediately from Eq. 3:

$$\langle z_{ik} \rangle \equiv p(z_{ik}=1 \mid \mathbf{x}_i, \theta) = \frac{p(z_{ik}=1) p(\mathbf{x}_i \mid z_{ik}=1)}{\sum_{\ell=1}^K p(z_{i\ell}=1) p(\mathbf{x}_i \mid z_{i\ell}=1)} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)}{\sum_{\ell=1}^K \pi_\ell \mathcal{N}(\mathbf{x}_i; \mu_\ell, \Sigma_\ell)} \quad (6)$$

The quantity $\langle z_{ik} \rangle$ is called the **responsibility** of component k for point i — also its **membership**. It is an $N \times K$ matrix whose rows sum to 1.

Both π_k and $\langle z_{ik} \rangle$ are “the probability of source k ”, and the whole of the E step lives in the difference between them. Say it in English:

- π_k is the probability that source k produced a datapoint **in general** — for a random point you have not looked at. It is the share of the population that source k accounts for, and it is what you would bet before seeing anything.
- $\langle z_{ik} \rangle$ is the probability that source k produced **this particular, observed** datapoint $\mathbf{x}_i$. It is the same question asked again *after* looking at $\mathbf{x}_i$, and the answer is now specific to that one point.

So π_k carries one number per source; $\langle z_{ik} \rangle$ carries one number per source *per point*. Looking at $\mathbf{x}_i$ is what turns the first into the second — which is exactly what Eq. 6 does, and why it is Bayes’ rule and nothing more.

Three readings of Eq. 6, all worth having:

- **As Bayes’ rule.** π_k is the *prior* over components — what you believe before seeing $\mathbf{x}_i$. The likelihood $\mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)$ is the evidence $\mathbf{x}_i$ provides. $\langle z_{ik} \rangle$ is the *posterior*. Note how the prior matters: a component with $\pi_k = 0.02$ needs to fit $\mathbf{x}_i$ about fifty times better than a component with $\pi_k = 0.98$ before it gets equal responsibility.
- **As an expectation — which is what the notation says.** Since z_{ik} is 0 or 1,

$$\mathbb{E}[z_{ik} \mid \mathbf{x}_i, \theta] = 1 \cdot p(z_{ik}=1 \mid \mathbf{x}_i, \theta) + 0 \cdot p(z_{ik}=0 \mid \cdot) = p(z_{ik}=1 \mid \mathbf{x}_i, \theta) = \langle z_{ik} \rangle.$$

For a 0/1 variable the posterior probability *is* the expectation, so the angle brackets are not a second object: $\langle z_{ik} \rangle$ is the average value of the missing bit z_{ik} , given this point. That is the reading that makes §4.2 work — wherever the fully-observed formula would use z_{ik} , we will use $\langle z_{ik} \rangle$.

- **As soft k -means.** With $\pi_k = 1/K$ and $\Sigma_k = \sigma^2 I$, Eq. 6 collapses to Eq. 5 exactly.

Key point: Keep π_k and $\langle z_{ik} \rangle$ apart; confusing them is the single most common conceptual error here. π_k is one number per *component* — the population share, the prior. $\langle z_{ik} \rangle$ is one number per *(point, component) pair* — the posterior for that specific point. They are connected by averaging: π_k is what you get when you average $\langle z_{ik} \rangle$ over all points (see Eq. 14).

Remark. The denominator of Eq. 6 is $p(\mathbf{x}_i)$ — the very quantity whose log appears in the observed-data log-likelihood, Eq. 4. So the E step hands you the per-iteration log-likelihood for free: accumulate $\sum_i \log(\text{denominator}_i)$ while normalizing. Do this. It costs nothing and it is how you will monitor convergence and detect bugs (§6.2).

4 The M Step

Memberships in hand, we do the other half of the chicken and egg: parameters from memberships. It takes two moves — write down what we *would* maximize if the memberships were real labels, then maximize it.

4.1 The Complete-Data Likelihood, and Why We Take Its Expectation

What would we do if the z_{ik} were known? Using the one-hot trick, the joint density of a single datapoint can be written as a product over components in which all but one factor equals 1 (we write a bare z when we mean all of point i 's indicators at once):

$$p(\mathbf{x}_i, z \mid \theta) = \prod_{k=1}^K [\pi_k \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)]^{z_{ik}}. \quad (7)$$

(Check: if $z_{i3} = 1$ and the rest are 0, every exponent is 0 except the third, and anything to the power 0 is 1, so the product is $\pi_3 \mathcal{N}(\mathbf{x}_i; \mu_3, \Sigma_3)$ — correct.) Taking the log of the product over all N points gives the **complete-data log-likelihood**:

$$\ell_c(\theta) = \log \prod_{i=1}^N \prod_{k=1}^K [\pi_k \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)]^{z_{ik}} = \sum_{i=1}^N \sum_{k=1}^K z_{ik} \left[\log \pi_k + \log \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k) \right]. \quad (8)$$

Compare Eq. 8 with Eq. 4. The sum over k has moved *outside* the logarithm. That is the entire difference, and it is everything: Eq. 8 is a sum of separate terms, one per component, each of which is the ordinary log-likelihood of a single Gaussian weighted by z_{ik} . Each μ_k appears in exactly one group of terms, so differentiating and setting to zero gives closed forms — the same ones as in **GDA_NB**.

We do not know z_{ik} . But §3 gives us its expectation under the current parameters. So take the expectation of Eq. 8 over z , holding the current θ fixed inside the posterior and treating the θ we are optimizing as free. Because Eq. 8 is *linear* in z_{ik} , the expectation passes straight through onto the z_{ik} 's and replaces each one by $\langle z_{ik} \rangle$:

$$Q(\theta \mid \theta^{\text{old}}) \equiv \mathbb{E}_{z \sim p(z \mid X, \theta^{\text{old}})} [\ell_c(\theta)] = \sum_{i=1}^N \sum_{k=1}^K \langle z_{ik} \rangle \left[\log \pi_k + \log \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k) \right]. \quad (9)$$

Q is called the **expected (complete-data) log-likelihood**, or the **ELL**. The M step maximizes it.

Key point: The algorithm in one sentence: *the thing we want to maximize (Eq. 4) is too hard because of the $\log \sum$; the thing we could maximize easily (Eq. 8) needs labels we do not have; so we fill in the labels with their expected values (Eq. 6) and maximize that instead (Eq. 9), repeatedly.* Why repeating this converges — and converges on the right thing — is §6.1 and §6.2.

Remark. A fair objection: why not just assign each point to its most likely component (take $\arg \max_k \langle z_{ik} \rangle$), which *would* give real labels, and then fit each component? That is exactly k -means, and it is not maximizing Eq. 4 — it is maximizing a different objective in which borderline points are declared certain. EM’s soft version is not a heuristic softening; it is the step that makes the procedure provably hill-climb the actual likelihood.

4.2 Three Weighted Averages

Maximize Eq. 9 over θ with the $\langle z_{ik} \rangle$ held fixed. Define the **effective count** of component k :

$$N_k = \sum_{i=1}^N \langle z_{ik} \rangle, \quad \text{so} \quad \sum_{k=1}^K N_k = N. \quad (10)$$

N_k is how many points component k owns, counting fractions. It need not be an integer.

① **The means.** Only the Gaussian term involves μ_k , and inside it only the quadratic form. Writing it with row vectors as in `GDA_NB`,

$$\log \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k) = -\frac{1}{2}(\mathbf{x}_i - \mu_k) \Sigma_k^{-1} (\mathbf{x}_i - \mu_k)^T + \text{const},$$

so the μ_k -dependent part of Q is $-\frac{1}{2} \sum_i \langle z_{ik} \rangle (\mathbf{x}_i - \mu_k) \Sigma_k^{-1} (\mathbf{x}_i - \mu_k)^T$. The one matrix-calculus fact needed is that for a symmetric A ,

$$\frac{\partial}{\partial \mu} [(\mathbf{x} - \mu) A (\mathbf{x} - \mu)^T] = -2(\mathbf{x} - \mu) A \quad (11)$$

(a $1 \times D$ row, matching μ ; it is the matrix version of $\frac{d}{d\mu}(x - \mu)^2 = -2(x - \mu)$). The two minus signs and the two halves cancel, leaving

$$\frac{\partial Q}{\partial \mu_k} = \sum_i \langle z_{ik} \rangle (\mathbf{x}_i - \mu_k) \Sigma_k^{-1} = 0.$$

Now right-multiply by Σ_k . Because Σ_k is invertible this changes nothing about where the zero is, and $\Sigma_k^{-1} \Sigma_k = I$ disappears:

$$\sum_i \langle z_{ik} \rangle (\mathbf{x}_i - \mu_k) = 0 \quad \implies \quad \sum_i \langle z_{ik} \rangle \mathbf{x}_i = \mu_k \sum_i \langle z_{ik} \rangle = \mu_k N_k,$$

giving

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^N \langle z_{ik} \rangle \mathbf{x}_i \quad (12)$$

— the *responsibility-weighted mean*. Set every $\langle z_{ik} \rangle$ to 0 or 1 and this is the plain class mean from `GDA_NB`.

② **The covariances.** Same trick as in GDA.NB: differentiate with respect to the *inverse*. Write $\Lambda_k = \Sigma_k^{-1}$, so that $\log |\Sigma_k| = -\log |\Lambda_k|$ and the Λ_k -dependent part of Q becomes

$$\sum_i \langle z_{ik} \rangle \left[\frac{1}{2} \log |\Lambda_k| - \frac{1}{2} (\mathbf{x}_i - \mu_k) \Lambda_k (\mathbf{x}_i - \mu_k)^T \right] + \text{const.}$$

This is now *linear* in Λ_k apart from the log-determinant, and two standard identities finish it:

$$\frac{\partial \log |\Lambda|}{\partial \Lambda} = \Lambda^{-1}, \quad \frac{\partial}{\partial \Lambda} [\mathbf{a} \Lambda \mathbf{a}^T] = \mathbf{a}^T \mathbf{a} \quad (\mathbf{a} = \mathbf{x}_i - \mu_k, \text{ a } 1 \times D \text{ row}).$$

(Both hold for symmetric Λ . Strictly, Λ_k has only $D(D+1)/2$ free entries rather than D^2 ; differentiating as if all D^2 were free introduces a factor of 2 on the off-diagonals that appears on both sides and cancels, so the stationary point is the same.) Differentiating term by term and setting the result to zero,

$$\frac{\partial Q}{\partial \Lambda_k} = \frac{1}{2} \sum_i \langle z_{ik} \rangle \left[\Lambda_k^{-1} - (\mathbf{x}_i - \mu_k)^T (\mathbf{x}_i - \mu_k) \right] = 0 \implies N_k \Lambda_k^{-1} = \sum_i \langle z_{ik} \rangle (\mathbf{x}_i - \mu_k)^T (\mathbf{x}_i - \mu_k),$$

using $\sum_i \langle z_{ik} \rangle = N_k$ on the first term. Since $\Lambda_k^{-1} = \Sigma_k$, this is already the answer — the responsibility-weighted scatter matrix:

$$\boxed{\Sigma_k = \frac{1}{N_k} \sum_{i=1}^N \langle z_{ik} \rangle (\mathbf{x}_i - \mu_k)^T (\mathbf{x}_i - \mu_k)} \quad (13)$$

with μ_k the value just computed in Eq. 12. Note the shape: $\mathbf{x}_i - \mu_k$ is a row vector ($1 \times D$), so $(\cdot)^T(\cdot)$ is $D \times D$ — same convention as the previous note.

③ **The mixing weights.** The Gaussian term has no π in it, so the only π -dependent part of Q is $\sum_i \sum_k \langle z_{ik} \rangle \log \pi_k$; and since $\log \pi_k$ does not depend on i , the inner sum collapses by Eq. 10:

$$\sum_i \sum_k \langle z_{ik} \rangle \log \pi_k = \sum_k \left(\sum_i \langle z_{ik} \rangle \right) \log \pi_k = \sum_k N_k \log \pi_k.$$

Maximizing this alone would send every $\pi_k \rightarrow \infty$, so the constraint $\sum_k \pi_k = 1$ is doing real work and has to be carried explicitly — hence a Lagrange multiplier:

$$\mathcal{L}(\pi, \lambda) = \sum_k N_k \log \pi_k - \lambda \left(\sum_k \pi_k - 1 \right), \quad \frac{\partial \mathcal{L}}{\partial \pi_k} = \frac{N_k}{\pi_k} - \lambda = 0 \implies \pi_k = \frac{N_k}{\lambda}.$$

That is K equations in $K+1$ unknowns; the missing one is $\partial \mathcal{L} / \partial \lambda = 0$, which just restates the constraint. Substituting $\pi_k = N_k / \lambda$ into it,

$$1 = \sum_k \pi_k = \frac{1}{\lambda} \sum_k N_k = \frac{N}{\lambda} \implies \lambda = N,$$

using $\sum_k N_k = N$ from Eq. 10. So the multiplier is not an artefact to be discarded — it comes out as the dataset size. Hence

$$\boxed{\pi_k = \frac{N_k}{N} = \frac{1}{N} \sum_{i=1}^N \langle z_{ik} \rangle} \quad (14)$$

— component k 's share of the (fractional) points. This is the promised connection: π_k is the column average of the responsibility matrix.

Key point: Every M-step formula is the formula you already knew, with counts replaced by responsibility-weighted counts. Mean $\rightarrow$ weighted mean; covariance $\rightarrow$ weighted covariance; class proportion $\rightarrow$ weighted proportion. That substitution — $z_{ik} \rightsquigarrow \langle z_{ik} \rangle$ — is the content of the E step, and it is why Eq. 9 had to be linear in z for any of this to work.

5 The Algorithm, Start to Finish

EM for a Gaussian mixture

Input: data X ($N \times D$), number of components K , tolerance ε .

1. **Initialize** $\pi_k = 1/K$; $\mu_k = K$ distinct datapoints drawn at random (or k -means centres); $\Sigma_k =$ the covariance of the whole dataset. Set $\ell^{(0)} = -\infty$.

2. **Repeat** for $t = 1, 2, \dots$:

E step. For every i and k compute, *in logs*, $a_{ik} = \log \pi_k + \log \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)$. Then $\log p(\mathbf{x}_i) = \text{logsumexp}_k a_{ik}$ and $\langle z_{ik} \rangle = \exp(a_{ik} - \log p(\mathbf{x}_i))$. Accumulate $\ell^{(t)} = \sum_i \log p(\mathbf{x}_i)$.

M step. $N_k = \sum_i \langle z_{ik} \rangle$; then π_k by Eq. 14, μ_k by Eq. 12, and Σ_k by Eq. 13 *using the new* μ_k .

Stop when $\ell^{(t)} - \ell^{(t-1)} < \varepsilon |\ell^{(t)}|$ (relative), or at a maximum iteration count.

3. **Return** $\{\pi_k, \mu_k, \Sigma_k\}$ and the responsibility matrix R ($N \times K$).

Numerics — the three that actually bite.

- **Work in logs, normalize with log-sum-exp.** In D dimensions a Gaussian density is routinely 10^{-300} and underflows to exactly 0; then $\langle z_{ik} \rangle = 0/0$. Compute a_{ik} as above and use $\text{logsumexp}(a) = m + \log \sum_k e^{a_{ik}-m}$ with $m = \max_k a_{ik}$, so the largest exponential is $e^0 = 1$ and nothing overflows. (`scipy.special.logsumexp` does this.)
- **Singular Σ_k .** If a component collects too few points — or, in the pathological case, exactly one — its covariance becomes singular, the density at that point goes to $+\infty$, and the likelihood diverges. This is a real, well-known degeneracy of the GMM likelihood, not a bug in your code. Guard it the same way `GDA_NB` guarded QDA: add ϵI to every Σ_k in the M step, with ϵ around 10^{-6} times the data's overall variance. Use `slogdet`, never `log(det(.))`.
- **Empty components.** If $N_k \rightarrow 0$, Eqs. 12–13 divide by zero. Detect it ($N_k < 10^{-8}$, say) and either re-seed that component at a random datapoint or drop it and report $K - 1$ components. Silently producing `nan` is the alternative.

Initialization and stopping. EM is sensitive to where it starts (§8). Standard practice: run it R times (say $R = 10$) from different random seeds and keep the run with the highest final ℓ . Initializing μ_k from k -means usually reduces the number of restarts you need. For stopping, use a *relative* tolerance — ℓ scales with N , so an absolute threshold that works for $N = 1000$ is meaningless for $N = 10^6$.

Key point: Before you trust this on real data, run it on four points. §9.1 works one full iteration by hand and gives the numbers your code must reproduce, together with the parameters to expect on `2gaussian.txt` and `3gaussian.txt`. An implementation that cannot match four points by hand will not be debuggable at $N = 10^4$.

6 Why It Works: the ELBO and Monotonicity

Everything so far has been a recipe. This section says why alternating two easy steps climbs a hard likelihood at all, and exactly what that buys you — which is also the vocabulary the VAE note assumes.

6.1 What EM Is Really Optimizing: the ELBO

The derivation above is a recipe; it does not yet say *why* alternating should work. The answer introduces a bound that will matter again in Week 10, so it is worth stating cleanly even though the proof lives in `EM_ADVANCED`.

Let $q(z)$ be *any* distribution over the latent variable. For a single datapoint, with a little algebra (Jensen’s inequality, or the KL decomposition — see `EM_ADVANCED`):

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(z|\mathbf{x})}[\log p(\mathbf{x} | z)] - \text{KL}(q(z | \mathbf{x}) \| p(z))}_{\text{ELBO}(q, \theta)}. \quad (15)$$

The right-hand side is the **evidence lower bound (ELBO)**. The name: $\log p(\mathbf{x})$ is the “evidence”; the right-hand side is a bound on it from below. Read the two terms:

- $\mathbb{E}_q[\log p(\mathbf{x} | z)]$ — **reconstruction**. If I guess the latent using q , how well does the model explain the data I actually saw?
- $\text{KL}(q(z | \mathbf{x}) \| p(z))$ — **regularization**. How far has my point-specific guess about z drifted from the prior I would use knowing nothing about the point? KL is zero when the two agree and positive otherwise, so this term is a penalty.

For a Gaussian mixture, both pieces are things you have already computed. The latent z takes K values, so q and p are just lists of K numbers:

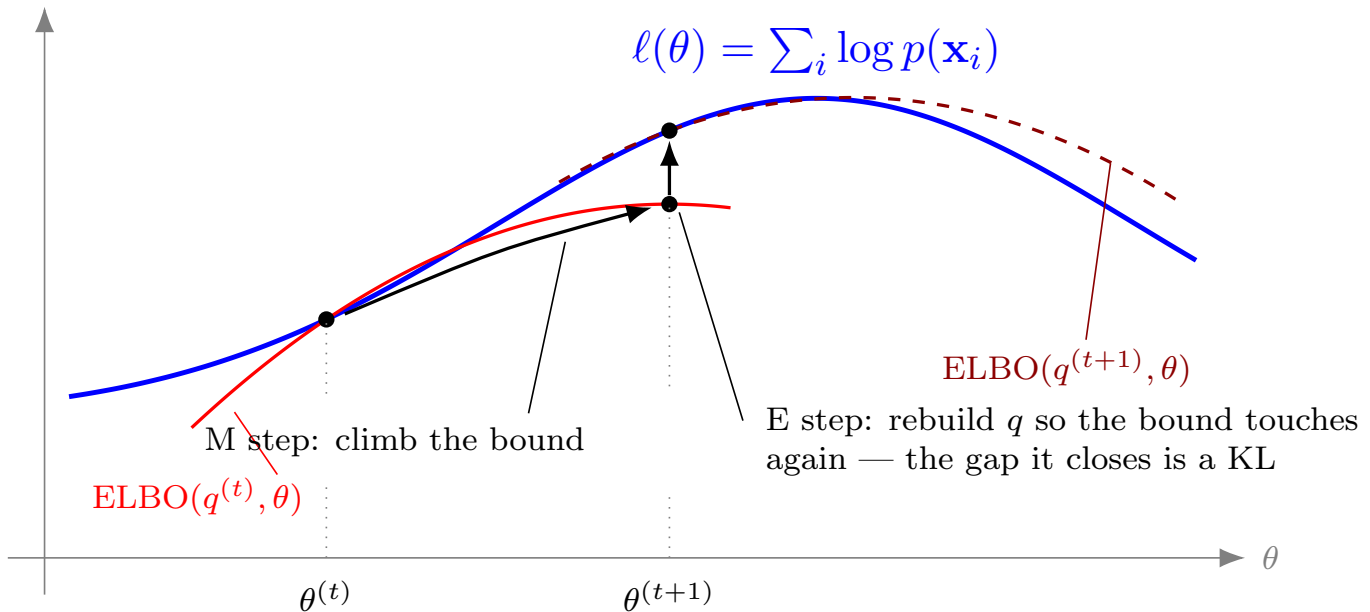
In Eq. 15	In your GMM	Where it came from
$p(z)$, the prior over the latent	$(\pi_1, \dots, \pi_K)$	M step, Eq. 14
$q(z \mathbf{x}_i)$, the guess for point i	$(\langle z_{i1} \rangle, \dots, \langle z_{iK} \rangle)$	E step, Eq. 6
$p(\mathbf{x} z = k)$, the decoder	$\mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)$	the component density

so, written out for the whole dataset,

$$\text{ELBO} = \sum_{i=1}^N \left[\sum_{k=1}^K \langle z_{ik} \rangle \log \mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k) - \sum_{k=1}^K \langle z_{ik} \rangle (\log \langle z_{ik} \rangle - \log \pi_k) \right]. \quad (16)$$

Expanding the second sum and recombining gives exactly $Q(\theta)$ from Eq. 9 plus the entropy of q — which is to say, **maximizing the ELL is maximizing the ELBO**. The recipe of §4.1–4.2 was a bound-optimization method all along.

The picture to carry away is the two-step staircase:



The E step chooses q to make the bound *touch* the true log-likelihood at the current θ — that is what $q(z | \mathbf{x}_i) = \langle z_{ik} \rangle$ achieves, and the gap it closes is exactly a KL divergence. The M step then climbs the bound. Since the bound touched the curve before the climb and lies below it after, the true log-likelihood must have gone up by at least as much as the bound did.

Key point: HW4 Problem 3(C) asks for the first column of the table above: identify $q(z | \mathbf{x})$ and $p(z)$ for your Gaussian mixture. The answer is $q(z | \mathbf{x}_i) = (\langle z_{i1} \rangle, \dots, \langle z_{iK} \rangle)$, your responsibilities — one distribution per datapoint, computed exactly by Bayes' rule — and $p(z) = (\pi_1, \dots, \pi_K)$, your mixing weights. No derivation required; the mapping is the point.

Remark. *Why this bound and not another.* Two features of Eq. 15 make it the right object. First, it is *tight*: the E step can drive the gap to zero, so optimizing the bound is not a permanent compromise. Second, it is *tractable*: $\log p(\mathbf{x})$ contains the intractable $\log \sum_k$, while the ELBO contains only $\mathbb{E}_q[\log p(\mathbf{x} | z)]$ — a weighted sum of logs of single components. Trading log-of-sum for weighted-sum-of-logs is the same move as in §4.1, viewed from one level up.

6.2 Monotonicity: the Log-Likelihood Must Not Decrease

The staircase picture has a consequence you can check numerically, and should:

Theorem (EM monotonicity). Let $\theta^{(t)}$ be the parameters after t EM iterations and $\ell(\theta) = \sum_i \log p(\mathbf{x}_i | \theta)$ the observed-data log-likelihood of Eq. 4. Then

$$\ell(\theta^{(t+1)}) \geq \ell(\theta^{(t)}) \quad \text{for every } t,$$

with equality only at a stationary point. (Proof: EM_ADVANCED.)

Note what the theorem does *and does not* say. It says ℓ never goes down. It does not say ℓ reaches the global maximum — EM is a hill-climber and the likelihood surface of a mixture has many local maxima (§8). It does not say the *parameters* converge quickly, or at all, to anything interpretable. And it says nothing about ℓ on held-out data, which can and does go down while training ℓ rises.

Key point: This is the best debugging tool you have. Plot $\ell(\theta^{(t)})$ after every iteration — the E step already computed it (see the remark in §3). If the curve ever dips, you have a bug, full stop. It is not a tolerance issue, not bad luck, and not a property of the data. **HW4 Problem 6** asks exactly this.

When the curve does dip, the cause is almost always one of these:

- **Stale parameters in the E step.** You updated μ_k but computed $\langle z_{ik} \rangle$ with the old Σ_k , or vice versa. The theorem assumes the E step uses the *complete* current θ .
- **Hard assignments in the E step.** Rounding $\langle z_{ik} \rangle$ to 0/1 — “it is basically certain anyway” — and then reporting the *soft* log-likelihood. You are now running k -means while plotting EM’s objective, and the two disagree; this one dips visibly and often.
- **Unnormalized responsibilities.** If $\sum_k \langle z_{ik} \rangle \neq 1$ for some i — a forgotten denominator, or a row that underflowed to all-zeros — every weighted average is wrong.
- **Measuring the wrong quantity.** Plotting Q (the ELL) instead of ℓ . Q is also non-decreasing across M-steps, but the two are different numbers and Q drops at the E step; if you record them in the wrong order you will see a sawtooth.
- **A variance floor or regularizer applied inconsistently** — added to Σ_k when scoring but not when fitting, say. Then the ℓ you plot is not the ℓ the algorithm is climbing.
- **What does *not* break it, despite looking like it should:** updating Σ_k with the *previous* iteration’s μ_k . Measured over two datasets and six random starts each, this never produced a single decrease. The reason is worth knowing: Σ_k taken about the old mean equals the correct one plus $(\mu_k^{\text{old}} - \mu_k)^T (\mu_k^{\text{old}} - \mu_k)$, so it is merely *inflated*, not wrong — still a valid covariance, and still an *increase* in Q . That makes it a legal generalized-EM step (EM_ADVANCED §1.3), so monotonicity survives. Nor does it cost anything: measured over both data files and three seeds, the final log-likelihood matches the correct run to ten decimal places, because at the fixed point $\mu_k^{\text{old}} = \mu_k$ and the two updates coincide in the limit — the bug perturbs only the transient. So here is code that is genuinely wrong, with a clean likelihood curve *and* the right answer. A monotone plot is a *necessary* condition for a correct implementation, never a sufficient one.
- **Genuine but tiny decreases** of order 10^{-10} relative, late in the run. These are floating-point noise once the algorithm has converged, not a bug. A real bug produces a visible drop early on. Judge by relative size.

7 Mixtures of Other Things

Nothing in §3–5 was specific to Gaussians. Swapping the component changes exactly one thing — the M step’s formula — and leaves the E step and the loop untouched. This section does the swap twice, then says how to choose.

7.1 Same Machinery, Different Component: the Coins Problem

Nothing in §3–4.2 was specific to Gaussians. Replace $\mathcal{N}(\mathbf{x}_i; \mu_k, \Sigma_k)$ by any likelihood $p_k(\mathbf{x}_i | \theta_k)$ and the E-step (Eq. 6) is unchanged; only the M step changes, and it changes into the weighted version of that family’s own maximum-likelihood estimator. The coins problem below is the standard instance.

The setup. There are K biased coins with unknown head-probabilities $p_1, \dots, p_K$ and unknown selection probabilities $\pi_1, \dots, \pi_K$. N times, somebody picks a coin according to π and flips it n times, reporting only the sequence of heads and tails — never which coin was used. Let $h_i \in \{0, 1, \dots, n\}$ be the number of heads in sequence i . The latent indicator z_{ik} now says which coin, exactly as before.

The component likelihood. Given coin k , the count h_i is Binomial:

$$p(h_i | z_{ik}=1) = \binom{n}{h_i} p_k^{h_i} (1 - p_k)^{n-h_i}. \quad (17)$$

E step. Substitute Eq. 17 into Eq. 6:

$$\langle z_{ik} \rangle = \frac{\pi_k \binom{n}{h_i} p_k^{h_i} (1 - p_k)^{n-h_i}}{\sum_{\ell} \pi_{\ell} \binom{n}{h_i} p_{\ell}^{h_i} (1 - p_{\ell})^{n-h_i}} = \frac{\pi_k p_k^{h_i} (1 - p_k)^{n-h_i}}{\sum_{\ell} \pi_{\ell} p_{\ell}^{h_i} (1 - p_{\ell})^{n-h_i}}. \quad (18)$$

The binomial coefficient $\binom{n}{h_i}$ depends on i but not on k , so it appears in every term of the numerator and denominator and **cancels**. You never need to compute it. (It does not cancel in the log-likelihood, where it contributes an additive constant — which is harmless for monitoring monotonicity, but means your ℓ differs from a library’s by that constant.)

M step. Maximize $\sum_i \sum_k \langle z_{ik} \rangle [\log \pi_k + h_i \log p_k + (n - h_i) \log(1 - p_k)]$ over p_k . Differentiating the p_k terms,

$$\frac{\partial}{\partial p_k} \sum_i \langle z_{ik} \rangle [h_i \log p_k + (n - h_i) \log(1 - p_k)] = \frac{\sum_i \langle z_{ik} \rangle h_i}{p_k} - \frac{\sum_i \langle z_{ik} \rangle (n - h_i)}{1 - p_k} = 0,$$

which rearranges to

$$\boxed{p_k = \frac{S_k}{n R_k}, \quad S_k = \sum_{i=1}^N \langle z_{ik} \rangle h_i, \quad R_k = \sum_{i=1}^N \langle z_{ik} \rangle} \quad \text{and} \quad \pi_k = \frac{R_k}{N} \quad (19)$$

(R_k is the N_k of Eq. 10 under a different name; π_k is Eq. 14 verbatim). Read Eq. 19 in words: *coin k ’s head-rate is the fraction of heads among the flips it is responsible for* — heads weighted by responsibility, divided by flips weighted by responsibility. Which is, of course, what you would compute if you knew which coin was which.

Worked example: one coin-EM iteration, by hand Two coins, $n = 10$ flips per sequence, $N = 4$ sequences with head counts $h = 9, 8, 2, 1$. Start at $p_1 = 0.6$, $p_2 = 0.4$, $\pi_1 = 0.5$.

The log-odds in Eq. 18 reduce to $h \log \frac{0.6}{0.4} + (10 - h) \log \frac{0.4}{0.6} = (2h - 10) \log 1.5 = 0.4055(2h - 10)$, so $\langle z_{i1} \rangle = \sigma(0.4055(2h_i - 10))$:

h_i	9	8	2	1
$\langle z_{i1} \rangle$	0.9624	0.9193	0.0807	0.0376

Then $R_1 = 2.0000$ (exactly, by the symmetry of this dataset), $S_1 = 9(0.9624) + 8(0.9193) + 2(0.0807) + 1(0.0376) = 16.215$, so

$$p_1 = \frac{16.215}{10 \times 2.0000} = \boxed{0.811}, \quad p_2 = \frac{(20 - 16.215)}{10 \times 2.0000} = \boxed{0.189}, \quad \pi_1 = \frac{2}{4} = 0.5.$$

One iteration has pushed the coins from $(0.6, 0.4)$ out to $(0.811, 0.189)$, and the log-likelihood from -25.394 to -19.893 . Use these numbers as your unit test before generating data.

Key point: The entire difference between HW4 Problem 3 and Problem 4 is which density goes inside Eq. 6 and which weighted average comes out of the M step. If you write your EM loop with the component likelihood as an argument, Problem 4 is a dozen lines on top of Problem 3. If you write two separate programs, you will debug the same algorithm twice.

7.2 Mixtures of Poissons

To make the pattern unmistakable, here is a third instance in a paragraph. Suppose each datapoint is a count $x_i \in \{0, 1, 2, \dots\}$ — calls to a help desk per hour, say — and you suspect two regimes, busy and quiet. Model it as a mixture of Poissons, $p_k(x \mid \lambda_k) = e^{-\lambda_k} \lambda_k^x / x!$.

E step: Eq. 6 with that likelihood; the $1/x_i!$ is constant in k and cancels, just as $\binom{n}{h_i}$ did. **M step:** the Poisson MLE is the sample mean, so the weighted version is

$$\lambda_k = \frac{\sum_i \langle z_{ik} \rangle x_i}{\sum_i \langle z_{ik} \rangle}, \quad \pi_k = \frac{1}{N} \sum_i \langle z_{ik} \rangle.$$

Which is Eq. 12 and Eq. 14 again.

Key point: The EM recipe, independent of family: **(E)** responsibilities by Bayes' rule, using the current parameters; **(M)** for each parameter, write down the ordinary MLE for that family and replace every count/sum over a group by the responsibility-weighted version. Any factor of the likelihood that does not depend on k cancels in the E step.

7.3 Choosing a Component Family

The component family is a modelling choice, and it is made the same way as the per-feature choice in GDA_NB §4.2: *look at what one datapoint actually is*. A datapoint that is a real vector wants a Gaussian; one that is a count out of a known number of trials wants a Binomial; one that is an unbounded count wants a Poisson. Getting this wrong is the mixture-model version of fitting a Gaussian to binary data.

One datapoint is...	Component	Parameters	M-step update
a real vector in $\mathbb{R}^D$	$\mathcal{N}(\mu_k, \Sigma_k)$	μ_k, Σ_k	weighted mean, weighted scatter
a real number	$\mathcal{N}(\mu_k, \sigma_k^2)$	μ_k, σ_k^2	weighted mean, weighted variance
h heads out of n flips	$\text{Bin}(n, p_k)$	p_k	$S_k/(nR_k)$: weighted head-rate
a single yes/no	$\text{Bern}(p_k)$	p_k	weighted fraction of yes
an unbounded count	$\text{Poisson}(\lambda_k)$	λ_k	weighted mean count
word counts over V terms	$\text{Mult}(\phi_k)$	$\phi_k \in \Delta^V$	weighted term frequencies

Every M-step update in the last column is “the ordinary maximum-likelihood estimator for that family, with each datapoint counted $\langle z_{ik} \rangle$ times instead of once.” Only the middle two columns change between rows.

What changes, and what does not. Across all six rows the E step is *the same formula*, Eq. 6, with a different density plugged in; and in every case the part of the density that does not involve k — $\binom{n}{h_i}$, $1/x_i!$, the multinomial coefficient — cancels out of it. The M step is the only place the family is visible, and even there it is only ever the family’s own MLE with weights attached. This is why §5’s algorithm box can be written once: pass the component likelihood in as an argument and the loop does not care what it is.

Remark. **Where mixtures actually earn their keep.** Three honest use cases, since “flexible density” can sound like a licence to use them everywhere. (i) *Genuine sub-populations.* Height in a mixed-sex population; two kinds of spam (phishing and advertising); two failure modes of a machine. Here the components mean something and K is set by domain knowledge. (ii) *A density with the wrong shape for one bump.* Skewed, heavy-tailed or multi-modal data where you want $p(\mathbf{x})$ for its own sake — anomaly detection scores a new point by its density under the fitted mixture, and a single Gaussian would score the valley between two modes as typical. (iii) *Soft clustering.* When you want not just a label per point but a confidence, $\langle z_{ik} \rangle$ is the answer and k -means has nothing to offer. Against that: a mixture with the wrong K or badly separated components tells you very little (§8), and if your real goal is a decision boundary rather than a density, a discriminative model will usually beat it with the same data (GDA_NB_ADVANCED §4).

8 What Goes Wrong: Local Optima, Label Switching, Identifiability

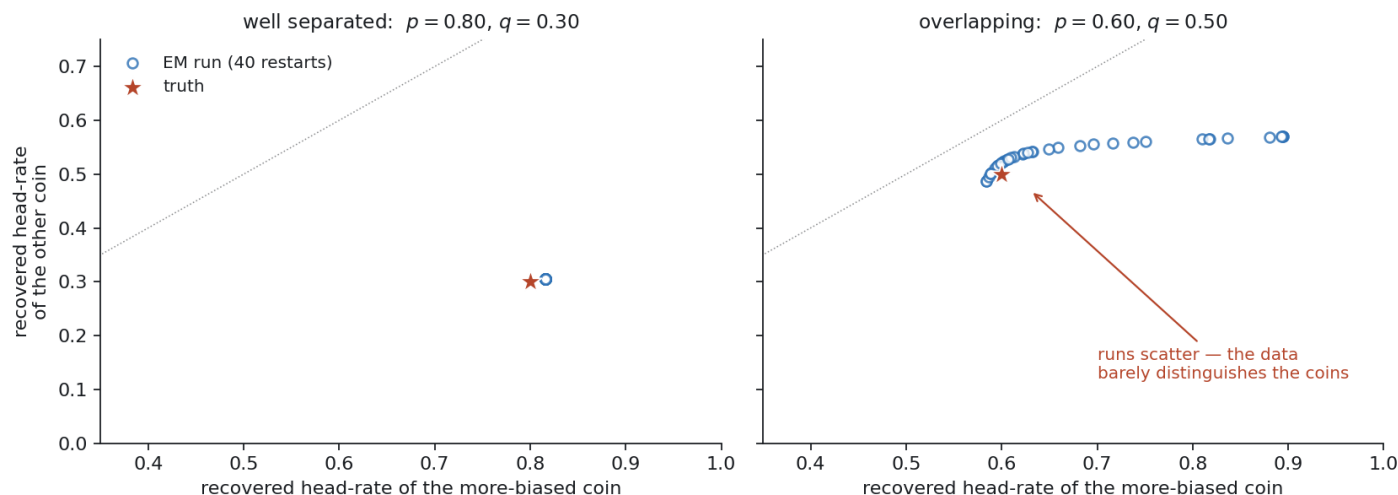
Three distinct things go wrong with mixtures, and they are often confused with each other.

(1) Label switching (harmless). Permute the component labels — swap (π_1, μ_1, Σ_1) with (π_2, μ_2, Σ_2) — and Eq. 1 is numerically identical. With K components there are $K!$ parameter settings giving exactly the same density, so the likelihood has (at least) $K!$ global maxima and the model is **non-identifiable** in the strict sense. Nothing is wrong; it just means “component 1” has no meaning across runs. Deal with it by canonicalizing before you compare anything — sort components by μ_k , or match them to a reference by nearest mean.

(2) Local optima (real, manageable). The likelihood surface genuinely has distinct maxima of different heights. Figure 1.1’s three-hump data illustrates it: from some starting points, two components merge onto one hump and the third stretches across the other two, and EM converges there happily — monotonically, as promised, to a worse answer. The mitigation is restarts: run from R random initializations and

keep the highest final ℓ . (The right-hand panel of §1.1’s figure is the best of twenty such runs. Over 300 random starts on that data, 11% converge to a merged-hump solution, so a single start is wrong about one time in nine and twenty starts make it negligible — the measurement is in EM_ADVANCED.)

(3) Weak identifiability (not fixable by restarts). If the components genuinely overlap, the data simply does not contain the information needed to tell them apart, and no algorithm can supply it. HW4 Problem 4 is a good laboratory for this, because you choose the coins.



Binomial-mixture EM, $n = 10$ flips, $N = 400$ sequences, forty random restarts. **Left:** coins at 0.80 and 0.30.

Every restart lands on the same point, essentially on the truth. **Right:** coins at 0.60 and 0.50. The restarts spread along an arc — many of them report a confident, badly separated pair — because a 10-flip sequence from a 0.60 coin and one from a 0.50 coin look nearly the same. The problem is the data, not the optimizer.

Key point: When you report HW4 Problem 4, try at least one well-separated setting and one overlapping setting, and say which is which. “EM recovered (0.81, 0.19) when the truth was (0.80, 0.30)” and “EM returned anything from (0.59, 0.49) to (0.89, 0.57) when the truth was (0.60, 0.50)” are both correct results, and the second one is the more interesting finding. More flips per sequence (n) helps far more than more sequences (N): n is how much evidence each latent decision gets.

9 HW4: Worked Numbers, Map, and Common Bugs

Everything needed to write the code and tell whether it is right: numbers to check against, a map from problems to sections, and the mistakes that actually happen.

9.1 Numbers You Can Check by Hand, and What to Expect on the HW Data

Worked example: one EM iteration in 1-D, by hand Take four points $x = 1, 2, 3, 6$ and $K = 2$. Start at $\pi_1 = \pi_2 = 0.5$, $\mu_1 = 0$, $\mu_2 = 4$, $\sigma_1^2 = \sigma_2^2 = 4$.

E step. With equal π ’s and equal variances, the log-odds simplify:

$$\log \frac{\pi_1 \mathcal{N}(x; 0, 4)}{\pi_2 \mathcal{N}(x; 4, 4)} = \frac{(x - 4)^2 - x^2}{2 \cdot 4} = \frac{16 - 8x}{8} = 2 - x, \quad \text{so} \quad \langle z_{i1} \rangle = \sigma(2 - x_i) = \frac{1}{1 + e^{x_i - 2}}.$$

x_i	1	2	3	6
$\langle z_{i1} \rangle$	0.7311	0.5000	0.2689	0.0180
$\langle z_{i2} \rangle = 1 - \langle z_{i1} \rangle$	0.2689	0.5000	0.7311	0.9820

Sanity checks: the point at $x = 2$ is exactly halfway between the two means, so it splits 50/50; the point at $x = 6$ is far from μ_1 and gets essentially no weight from component 1. The log-likelihood at this starting point is $\ell^{(0)} = -9.1331$.

M step. $N_1 = 0.7311 + 0.5 + 0.2689 + 0.0180 = 1.5180$, so $N_2 = 4 - 1.5180 = 2.4820$ and

$$\pi_1 = \frac{1.5180}{4} = 0.3795, \quad \mu_1 = \frac{0.7311(1) + 0.5(2) + 0.2689(3) + 0.0180(6)}{1.5180} = \frac{2.6462}{1.5180} = \boxed{1.743},$$

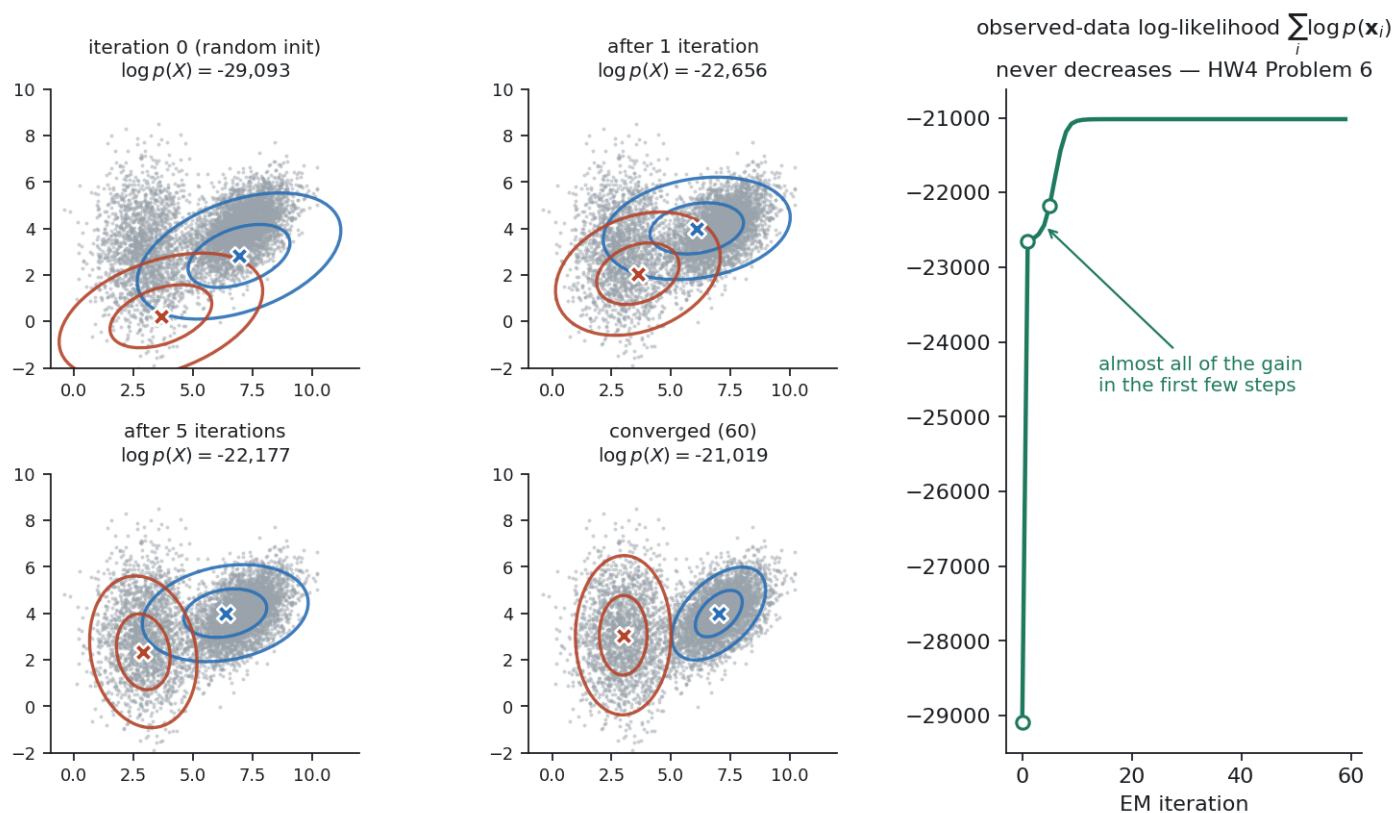
and likewise $\mu_2 = 9.3538/2.4820 = 3.769$, $\sigma_1^2 = 0.782$, $\sigma_2^2 = 3.605$. The means have moved toward each other, which is right: component 1 started at 0 with no data below 1, and component 2 started at 4 but owns the point at 6 only partially.

Monotonicity check. Three iterations give $\ell = -9.1331, -7.6209, -7.5144$. Non-decreasing, with the bulk of the gain in the first step — the shape you should see on the real data too. If your code reproduces the $\langle z_{i1} \rangle$ row and these three numbers, your E step, M step and likelihood accumulator are all correct, and any remaining problem on Spambase-sized data is numerical rather than algorithmic.

What convergence looks like. Running the algorithm above on `2gaussian.txt` ($N = 6000$) and `3gaussian.txt` ($N = 10000$), best of ten random restarts, recovers:

		true π_k	found π_k	true μ_k	found μ_k
2gaussian	comp. A	0.333	0.335	[3, 3]	[2.994, 3.052]
	comp. B	0.667	0.665	[7, 4]	[7.013, 3.983]
3gaussian	comp. A	0.200	0.206	[3, 3]	[3.040, 3.048]
	comp. B	0.300	0.298	[7, 4]	[7.022, 4.015]
	comp. C	0.500	0.496	[5, 7]	[5.012, 7.001]

The covariances come back equally well — e.g. for **2gaussian** comp. B the true $\Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$ is recovered as $\begin{bmatrix} 0.975 & 0.497 \\ 0.497 & 1.001 \end{bmatrix}$. Convergence (relative tolerance 10^{-4}) takes about 20 iterations in both cases.



EM on `2gaussian.txt` from a random start. The four panels are the fitted components at iterations 0, 1, 5 and 60 (one and two standard deviations). The right panel is the observed-data log-likelihood: it rises monotonically, almost all of it in the first ten iterations, then flattens. This is the plot to produce whenever you run EM, for the reason given in §6.2.

Remark. Do not expect component A of your output to be component A of the table. The labelling is arbitrary (see §8); match components to the truth by proximity of their means before computing any “absolute parameter difference”. Mismatched ordering is the most common reason a correct implementation appears to have failed.

9.2 Problem-by-Problem Map, and Common Bugs

Problem 3 (A,B) — EM for a Gaussian mixture (50 points). Implement §5 and run it on `2gaussian.txt` ($K = 2$) and `3gaussian.txt` ($K = 3$). The E and M update logic must be yours; evaluating a Gaussian density with a library call is fine. Compare against the true parameters as in §9.1 — match components by nearest mean first. Library baseline: `sklearn.mixture.GaussianMixture` (use `covariance_type='full'`, and note that it defaults to k -means initialization and ten EM iterations’ worth of warm-up, so its result will look better than one random restart of yours; that is a fair thing to point out, not a defect).

Problem 3(C) — the ELBO (10 points). §6.1. You are asked to identify $q(z | \mathbf{x})$ and $p(z)$, not to derive the bound. One short paragraph with the table’s mapping, plus a sentence on why q is computed exactly here (closed-form Bayes’ rule over K values) is a complete answer. HW5 Problem 3(C) picks this up again when q has to be *learned* instead.

Problem 4 — mixture of Binomials (50 points). §7.1. Generate the data yourself so you know the truth; the generative step is §1.2’s recipe with Binomial components. Verify against the hand-worked iteration in §7.1 before trusting anything. Try both a well-separated and an overlapping parameter setting (§8). There is no library baseline for this one — validation is against your own known ground truth.

Problem 6 — (theory) can the log-likelihood decrease? (25 points). §6.2. The answer is no, and the expected response names at least one concrete bug and says where to look for it. The bullet list in §6.2 is the menu; pick one and explain the mechanism, do not just name it.

Problem 11 (optional) — Naive Bayes with K Gaussians per feature. Combine this note with GDA_NB §4 (Naive Bayes): keep the conditional-independence factorization, but model each per-feature, per-class likelihood $p(x^j \mid y = c)$ as a 1-D K -component mixture fitted by EM, instead of a single Gaussian. That is $D \times C$ independent little EM runs. It is option 2 of GDA_NB’s three applied one feature at a time, and it fixes exactly the failure that note’s likelihood figure illustrates.

Common bugs.

- Computing $\langle z_{ik} \rangle$ with densities instead of log-densities (§5). In 2-D on the HW data you may get away with it; you will not on anything higher-dimensional.
- Updating Σ_k with the *previous* μ_k . Breaks monotonicity — see §6.2.
- Shape errors in Eq. 13: $(\mathbf{x}_i - \mu_k)^T (\mathbf{x}_i - \mu_k)$ must be $D \times D$, not a scalar. If your Σ_k comes out as a number, you have the transposes backwards.
- Forgetting that $\langle z_{ik} \rangle$ depends on π_k . Dropping π_k from Eq. 6 gives an algorithm that still converges, to the wrong thing, whenever the components have unequal sizes — which is the case in both HW data files.
- Comparing recovered to true parameters without first matching components (§8).
- Declaring convergence on an absolute change in ℓ .
- Computing $\binom{n}{h_i}$ in Problem 4’s E step. It cancels (§7.1); computing it with factorials overflows for modest n .

10 Notation, and Where This Goes Next

Two short references to close on: a dictionary between this note’s symbols and the ones used in lecture and in the textbooks, and where the latent variable goes after this.

10.1 Notation Bridge

The board, the textbooks and these notes use different letters for the same few objects. Here is the dictionary, so that nothing in lecture, in the linked handwritten notes, or in anything you read outside the course looks like a different algorithm.

These notes	On the board	Bishop, Murphy	Meaning
π_k	w_k	π_k	mixing weight of component k ; prior $p(z = k)$
z_{ik}	Π_{ik}	z_{nk}	indicator: did point i come from component k
$\langle z_{ik} \rangle$	$\langle \Pi_{ik} \rangle, \hat{\Pi}_{ik}$	$\gamma(z_{nk}), r_{ik}$	responsibility (membership) = $\mathbb{E}[z_{ik} \mid \mathbf{x}_i, \theta]$
$k = 1, \dots, K$	$\ell = 1, \dots, L$	$k = 1, \dots, K$	component index
$N_k = \sum_i \langle z_{ik} \rangle$	—	N_k	effective count of component k
$Q(\theta \mid \theta^{\text{old}})$	ELL	Q	expected complete-data log-likelihood

Angle brackets $\langle \cdot \rangle$ mean $\mathbb{E}[\cdot]$, here and on the board. The data conventions are unchanged from every previous note in this course: X is $N \times D$; $\mathbf{x}_i$ is the i -th row, a row vector; x_i^j is its j -th feature; quadratic forms are written $(\mathbf{x} - \mu)\Sigma^{-1}(\mathbf{x} - \mu)^T$; class index is c , component index is k .

Key point: Two symbols carry the latent variable in this note, not three: the *indicator* z_{ik} , which is 0 or 1 and which we never observe, and its *expectation* $\langle z_{ik} \rangle$, which we compute. The notation is deliberately literal — $\langle z_{ik} \rangle$ is not a new quantity with its own name, it is the average of z_{ik} . The *word* for it, however, is standard and worth knowing: most of the literature writes this as r_{ik} or $\gamma(z_{nk})$ and calls it the **responsibility**. Search for that word, not for the brackets.

Remark. One collision worth flagging: K in this note is the number of mixture *components*, while in some of the linked coin-problem write-ups K denotes the number of *flips* per sequence. This note uses n for the number of flips and N for the number of sequences (§7.1), so that k and K keep their meaning across both problems.

10.2 Where This Goes Next

- **The proofs.** Why Eq. 15 holds (Jensen; the KL decomposition), why the E step makes the bound tight, and the monotonicity theorem of §6.2 with its proof, plus EM as coordinate ascent on the ELBO: EM_ADVANCED.
- **Structured latent variables.** In this note the latent variable was one indicator per datapoint, independent across i . Let the z 's form a chain — z_t depends on z_{t-1} — and the same E/M alternation becomes the **Baum–Welch** algorithm for **hidden Markov models**, with the E step done by forward–backward instead of a single Bayes' rule. See EM_ADVANCED (optional). More generally this is option 3 of GDA_NB's three: “not-so-naïve Bayes” $\rightarrow$ Bayes networks.
- **When q cannot be computed.** Everything here rested on the E step being exact: K values of z , so Eq. 6 is a finite sum. Make z a continuous vector and let $p(\mathbf{x} \mid z)$ be a neural network, and the posterior $p(z \mid \mathbf{x})$ has no closed form. The response is to *learn* an approximation $q_\phi(z \mid \mathbf{x})$ with a second network, and to optimize the same ELBO of Eq. 15 with respect to both networks at once. That is the **variational autoencoder** — Week 10, and **HW5 Problem 3**, whose part (C) asks you to line it up against what you did in HW4 Problem 3(C). The objective does not change; only the tractability of q does.