Assignment 2 — (Non-) Parametric Density Estimation

Image Processing and Pattern Recognition

Deadline: December 19, 2024

1. Goal

In this exercise you will become familiar with parametric and non-parametric density estimation. In particular, you will implement the expectation-maximization algorithm to train a Gaussian mixture model for patch-based denoising. Additionally, you will utilize kernel density estimation for image segmentation.

2. Bayesian Denoising (15P)

Let $X \in \mathbb{R}^{M\times M}$ be an unknown clean image patch. For notational convenience, we denote with $x \in \mathbb{R}^m$ ($m = M^2$) the patch flattened in lexicographical order (e.g. $M = 2$, $x = (X_{11}, X_{12}, X_{21}, X_{22})^\top$). We assume access to an image patch y corrupted by Gaussian noise, modelled by

$$y = x + \nu \tag{1}$$

where $\nu \sim \mathcal{N}(\mathbf{0}, \sigma_n \mathrm{Id}_m)$. Here, $\mathcal{N}(\mu, \Sigma)$ denotes the Gaussian distribution on $\mathbb{R}^m$ with mean μ, covariance matrix Σ, and Id_k the $k \times k$ identity matrix. In other words, each entry in y is corrupted by i.i.d. zero-mean Gaussian noise with variance σ_n^2. Viewing y as a random vector, it follows a distribution with density (called the *likelihood*)

$$p_{\mathrm{li}}(y \mid x) = (2\pi\sigma_n^2)^{-\frac{m}{2}} \exp\Big(-\frac{1}{2\sigma_n^2} \|x - y\|_2^2\Big). \tag{2}$$

By Bayes theorem, the *posterior distribution* $p_{\mathrm{po}}(x \mid y)$ has density

$$p_{\mathrm{po}}(x \mid y) = \frac{p_{\mathrm{li}}(y \mid x) p_{\mathrm{pr}}(x)}{p_{\mathrm{ev}}(y)} \tag{3}$$

where p_{pr} encodes *prior* knowledge of the distribution of the clean patches. Given p_{po}, a classical point estimator for the unknown clean patch x is the patch x_{MAP} that maximizes the posterior distribution, i.e.

$$x_{\mathrm{MAP}} \in \arg\max_x p_{\mathrm{po}}(x \mid y). \tag{4}$$

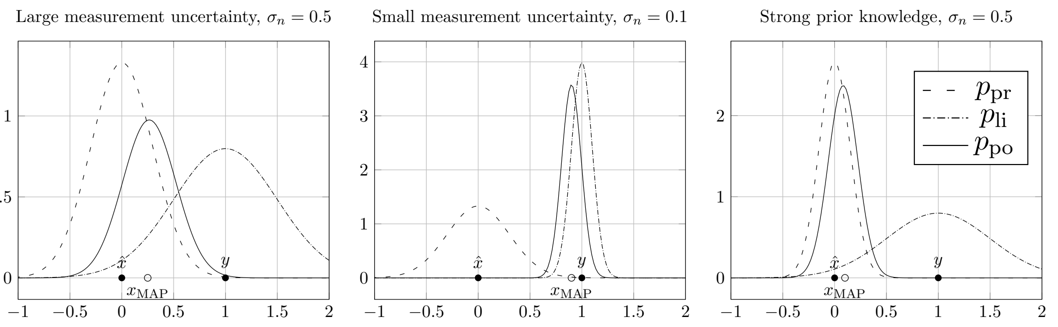

Figure 1: Trading off prior knowledge and measurement uncertainty in Bayesian inference: Prior knowledge tells us x is "close" to some $\hat{x}$ with standard deviation 0.3 (left, middle) or 0.15 (right). A measurement $y = 1$ was acquired with different associated uncertainty. With large measurement uncertainty or a strong prior, x_{MAP} shifts towards $\hat{x}$ (left and right respectively). With high measurement certainty, the influence of the prior vanishes (middle).

We show a one-dimensional toy example in Fig. 1. Exploiting the monotonicity of the logarithm (and noting that the denominator in Eq. (3) is independent of x), we can equivalently write

$$x_{\mathrm{MAP}} \in \arg\min_{x} \frac{1}{2\sigma_n^2} \|x - y\|_2^2 - \log p_{\mathrm{pr}}(x). \tag{5}$$

2.1. Gaussian Mixture Models

In this assignment we utilize a particular parametric density estimator as a prior: the Gaussian mixture model (GMM). With slight abuse of notation, the density function of a GMM with $K \in \mathbb{N}$ components is

$$p(x, \theta) = \sum_{k=1}^{K} \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k), \tag{6}$$

where the trainable parameters can be summarized as $\theta = \{\alpha_k, \mu_k, \Sigma_k\}_{k=1}^{K}$. Here, $\alpha_k \in \mathbb{R}^+$ weights the k-th component, and μ_k, Σ_k are the mean and covariance of the k-th component respectively. For Eq. (6) to represent a proper density, we require $\alpha_k \geq 0$, $\sum_{k=1}^{K} \alpha_k = 1$ and Σ_k symmetric pos. def. $\forall\, k = 1, \dots, K$. The density of the multivariate Gaussian distribution is

$$\mathcal{N}(x; \mu, \Sigma) = \left((2\pi)^m \det \Sigma\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right). \tag{7}$$

2.1.1. Training

To train the GMM, we assume access to a database of N clean datapoints $(x_i)_{i=1}^{N}$. In the well known maximum-likelihood framework, training the GMM amounts to solving the optimization problem

$$\min_{\theta} \sum_{i=1}^{N} -\log\left(\sum_{i=1}^{K} \alpha_k \mathcal{N}(x_i; \mu_k, \sigma_k)\right). \tag{8}$$

Eq. (8) can be solved by the iterative expectation-maximization algorithm: Given the estimates $\alpha_k^{(j)}, \mu_k^{(j)}, \Sigma_k^{(j)}$ at the j-th iteration, the expectation step computes the *responsibilities* $\gamma_{k,i}^{(j)}$ of the k-th component w.r.t the i-th datapoint as

$$\gamma_{k,i}^{(j)} = \frac{\alpha_k^{(j)} \mathcal{N}(x_i; \mu_k^{(j)}, \Sigma_k^{(j)})}{\sum_{k=1}^{K} \alpha_k^{(j)} \mathcal{N}(x_i; \mu_k^{(j)}, \Sigma_k^{(j)})}. \tag{9}$$

We detail how to implement this equation in a numerically stable manner in Appendix A. With the current responsibilities $\gamma_{k,i}^{(j)}$, the log-likelihood is maximized by updating the parameters as

$$\begin{aligned} \alpha_k^{(j+1)} &= \frac{1}{N} \sum_{i=1}^{N} \gamma_{k,i}^{(j)}, \\ \mu_k^{(j+1)} &= \frac{\sum_{i=1}^{N} \gamma_{k,i}^{(j)} x_i}{\sum_{i=1}^{n} \gamma_{k,i}^{(j)}}, \\ \tilde{\Sigma}_k^{(j+1)} &= \frac{\sum_{i=1}^{N} \gamma_{k,i}^{(j)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^\top}{\sum_{i=1}^{N} \gamma_{k,i}^{(j)}}. \end{aligned} \tag{10}$$

To make the algorithm more stable (components might collapse to single data points), we regularize the covariance matrices by adding a small diagonal matrix:

$$\Sigma_k^{(j+1)} = \tilde{\Sigma}_k^{(j+1)} + \epsilon \mathrm{Id}_m, \tag{11}$$

where $\epsilon \ll$ is a tunable parameter, which should be set as small as possible while retaining a stable algorithm.

2.2. Inference

To be invariant with respect to radiometric shifts (see [2] for why this is desirable), we now assume a mixture trained on zero-mean patches. Given noisy patches $(y_i)_{i=1}^{J}$, solving the inference problem Eq. (5) translates to

$$x_{i,\mathrm{MAP}} \in \arg\min_{x} \frac{1}{2\sigma_n^2} \|x - y_i\|_2^2 - \log\Bigl(\sum_{k=1}^{K} \alpha_k \mathcal{N}(Ex; \mu_k, \Sigma_k)\Bigr) \tag{12}$$

where $\mathbb{R}^{m\times m} \ni E = \mathrm{Id}_m - \frac{1}{m}\mathbf{1}\mathbf{1}^\top$ ($\mathbf{1} \in \mathbb{R}^m$ is the one-vector) is a matrix that projects onto $\{x \in \mathbb{R}^m \colon \sum_{i=1}^{m} x_i = 0\}$, i.e. the set of zero-mean patches. We solve this problem using the approximate MAP estimation procedure proposed by [3], summarized in Algorithm 1. Fig. 2 shows a full-image (y, x_{MAP}, x) triplet using $\{K=10, m=25\}$.

Algorithm 1: Fast approximate zero-mean patch-based denoising.

```
  Data: Noisy patch $y$, number of iterations $N_{\text{iter}}$
  Result: Denoised patch $x^{(N_{\text{iter}})}$
1 $x^{(0)} = y$
2 for $j = 0, \dots, N_{\text{iter}} - 1$ do
3     $k_{\max} = \arg\max_k \log\left(\alpha_k \mathcal{N}(Ex^{(j)}; \mu_k, \Sigma_k)\right)$   // Responsible component
4     $\tilde{x} = \left(\lambda \mathrm{Id}_m + E^T \Sigma_{k_{\max}}^{-1} E\right)^{-1} (\lambda y + \Sigma_{k_{\max}}^{-1} E \mu_{k_{\max}})$   // Wiener Filter
5     $x^{(j+1)} = \alpha x^{(j)} + (1 - \alpha)\tilde{x}$   // Overrelaxed update
```

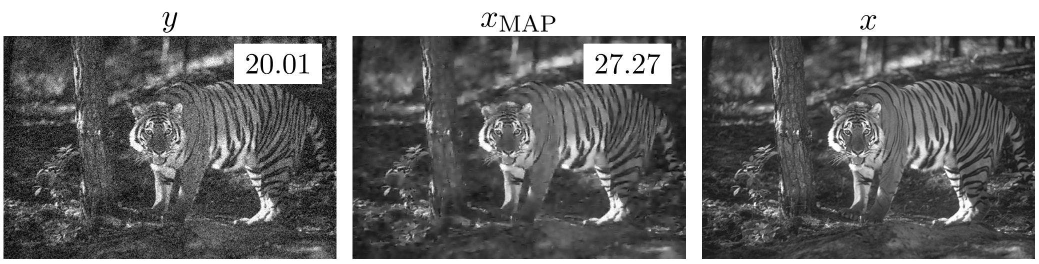

Figure 2: Denoising using a Gaussian mixture model. The inset shows the PSNR value.

2.3. Framework & Tasks

The file `denoising.py` (along with the utilities in `utils.py`) provide a framework for you to work in. The parts that have to be extended are marked, in particular you have to

1. implement the expectation-maximization algorithm Eqs. (9) to (11) (6P) and
2. (parts of the) fast approximate MAP algorithm (4P) (Algorithm 1).

To help you implement the expectation-maximization algorithm, we have provided a toy dataset consisting of 1000 points with 2 features drawn from 2 Gaussians. In addition, the utilities provide a function to visualize the GMM when working with the toy dataset. We provide a trained model (for the toy dataset, $K = 2$, $\epsilon = 1 \times 10^{-6}$, `max_iter = 50`) for reference, which your implementation should match up to numerical precision. If you do not manage to implement the expectation-maximization algorithm, we provide pre-trained models for the rest of the tasks.

In the report, you should discuss the following (5P):

- Use Algorithm 1 to denoise the images in the validation dataset:
  - Try the algorithm for $\sigma_n \in \{0.05, 0.1\}$ and report PSNR values for each image (σ_n is an argument to the `denoise` function).
  - For $\sigma_n = 0.1$, show the noisy, denoised and uncorrupted ground truth images of the validation dataset (the total of 15 images should easily fit in one figure on one page).

Table 1: Reference performance on the validation set using $K = 10$, $m = 25$.

Validation index	1	2	3	4	5
PSNR	27.27	28.54	31.39	27.00	31.39

State the hyper-parameters of the GMM that you used to get this output. The parameters in the skeleton file ($K = 10$, $m = 25$) are good starting points (assuming an efficient implementation).

- Experiment with the hyper-parameters of the GMM: K and m. Show the impact of the changes on the validation set. You should have at least 2 examples for the number of Gaussians and 2 examples for the filter size. Illustrate the impact on one noisy example of the validation dataset. If the expectation-maximization algorithm becomes unstable, increase ϵ (you should not have to increase it over 1×10^{-6} for $m = 25$).
- What are the advantages and drawbacks of increasing K and m?

In addition, there is a challenge for bonus points: Use your best model and apply it to the noisy test set (the function `benchmark` handles all the book-keeping). We will compare your results with the reference images by PSNR. We award $\{5, 4, 3\}$ bonus points for the $\{\text{first}, \text{second}, \text{third}\}$-best group.

Performance The reference method finishes 50 expectation-maximization iterations for a GMM with $K = 10$ components and $m = 25$ features in 179.01 s, using 1.4 GB of memory on an AMD Ryzen 9 3900X. With the same hyper-parameters, the 30 iterations of the fast approximate MAP finish in 9.45 s. We show the quantitative performance of the reference model and implementation in Table 1.

3. Kernel Density Estimation and Mean-Shift (10P)

Let $(x_i)_{i=1}^N$, $x_i \in \mathbb{R}^d$ be a collection of N datapoints in d-dimensional space, drawn from some unknown distribution we wish to estimate. A popular way to construct a probability density from these datapoints — that does not make any model assumptions — is kernel density estimation (KDE). Given a kernel function $\phi\colon \mathbb{R}^d \to \mathbb{R}^+$ satisfying $\phi \geq 0$ and $\int_{\mathbb{R}^d} \phi = 1$, the KDE takes the form

$$f_h(x) = \frac{1}{Nh^d} \sum_{i=1}^{N} \phi\left(\frac{x_i - x}{h}\right), \tag{13}$$

where $h \in \mathbb{R}^+$ is the *bandwidth*, defining the fidelity of the KDE. Here, we consider the Epanechnikov kernel $u \mapsto \frac{3}{4}(1 - u^2)\mathbb{1}_{|\cdot| \leq 1}(u)$ ($\mathbb{1}_{\mathcal{A}}$ is the indicator function of the set $\mathcal{A}$), as shown in Fig. 3. Assuming a radially symmetric kernel, the kernel function can be

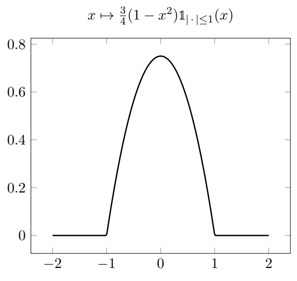

Figure 3: The Epanechnikov kernel in one dimension.

written as $\phi(u) = ck(\|u\|^2)$, where $c \in \mathbb{R}^+$ is a normalization constant and $k\colon \mathbb{R}^+ \to \mathbb{R}^+$ is the *profile*.

The mean shift algorithm [1] is an algorithm for mode-finding in a KDE. The iterations proceed by

$$x^{(j+1)} = \frac{\sum_{i=1}^N x_i g\left(\left\|\frac{x^{(j)}-x_i}{h}\right\|^2\right)}{\sum_{i=1}^N g\left(\left\|\frac{x^{(j)}-x_i}{h}\right\|^2\right)}, \tag{14}$$

where $g = -k'$. Observe that in the Epanechnikov case, k takes on the particularly easy form $k(l) = (1-l)\mathbb{1}_{|\cdot|\leq 1}(l)$. Additionally, $k'(l) = -\mathbb{1}_{|\cdot|\leq 1}(l)$ and Eq. (14) reduces to

$$x^{(j+1)} = \frac{\sum_{i\in\mathcal{I}(x^{(j)})} x_i}{\operatorname{card}\mathcal{I}(x^{(j)})}, \tag{15}$$

where $\mathcal{I}(x^{(j)}) = \{i\colon \left\|\frac{x^{(j)}-x_i}{h}\right\|^2 \leq 1\}$ and card denotes the cardinality.

3.1. Image Segmentation

We utilize the mean shift algorithm to perform image segmentation as follows: Similar to the last assignment, let $F\colon \Omega \to [0,1]^3$ denote an image from the (square) image domain $\Omega = \{1,\dots,M\} \times \{1,\dots,M\}$ to an RGB tuple. For each pixel $p \in \Omega$, we define the features as

$$p \mapsto \left((F(p))^\top, o_\zeta(p_1), o_\zeta(p_2)\right)^\top \in [0,1]^3 \times [-\zeta,\zeta]^2 \tag{16}$$

where $o_\zeta(x) = (2\frac{x-1}{M-1} - 1)\zeta$, and $\zeta \in \mathbb{R}^+$ scales the positional features. In other words, our features consist of the original image features (RGB tuples) concatenated with the normalized position. The mean shift algorithm is finished, when for all datapoints

$$\left\|x^{(j+1)} - x^{(j)}\right\|^2 < 1 \times 10^{-6}. \tag{17}$$

We show an example using $\zeta = 1$, $h = 0.3$ in Fig. 4.

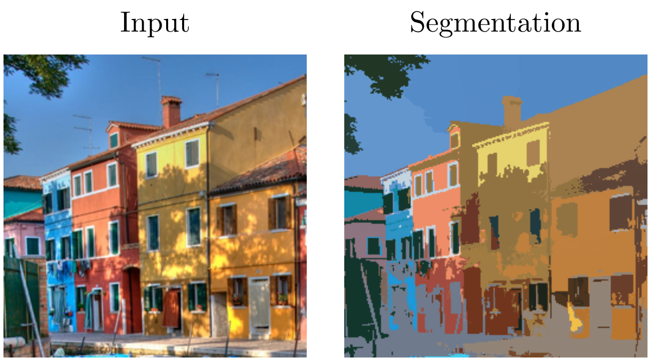

Figure 4: Mean shift image segmentation using $\zeta = 1$ and $h = 0.3$.

3.2. Framework & Tasks

The file `mean_shift.py` sets up the feature space for you. You have to implement the mean shift iterations Eq. (15) as well as the stopping criterion Eq. (17) (5P).

In the report, you should discuss the following points (5P):

- Show and discuss the results for $\zeta \in \{1, 4\}$ and $h \in \{0.1, 0.3\}$
- Consider the following practical segmentation task:
  - Distinguish 8 classes: Sky, the six clearly visible houses, and the pavement.
  - Distinguish 2 classes: Sky and non-sky.

  Try to find optimal parameters ζ and h for these tasks, show and discuss the results.
- Discuss the advantages and drawbacks of parametric and non-parametric density estimation in a general sense.

3.2.1. Notes

We provide two input images with $M \in \{128, 256\}$ respectively, you can prototype your implementation with the smaller image but should complete the task for $M = 256$. To help with performance, we implemented the code skeleton in `pytorch`, such that you can choose whether to run the implementation on the CPU or the GPU. We highly recommend to set up `pytorch` such that it has access to the GPU, if you have one available to you. In addition, we provide reference images for the standard parameters mentioned above for both resolutions, which you are expected to match up to numerical precision.

Performance The reference implementation finishes all four standard parameter choices described in Section 3.2 for $M = 128$ in 5.8 s (`simul = M ** 2`) and for $M = 256$ in 95.3 s (`simul = M ** 2 // 3`) on an Nvidia TITAN RTX.

References

[1] D. Comaniciu and P. Meer. “Mean shift: a robust approach toward feature space analysis”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.5 (2002), pp. 603–619. DOI: `10.1109/34.1000236`.

[2] Sonia Tabti et al. “Building invariance properties for dictionaries of SAR image patches”. In: *2014 IEEE Geoscience and Remote Sensing Symposium*. 2014, pp. 4918–4921. DOI: `10.1109/IGARSS.2014.6947598`.

[3] Daniel Zoran and Yair Weiss. “From learning models of natural image patches to whole image restoration”. In: *2011 International Conference on Computer Vision*. 2011, pp. 479–486. DOI: `10.1109/ICCV.2011.6126278`.

A. Implementation

In this appendix, we detail how to implement the concepts discussed in Section 2 in a numerically stable and fast way.

A.1. Responsibilities

Recall the definition of the responsibility of the k-th component w.r.t. the i-th datapoint at iteration j:

$$\gamma_{k,i}^{(j)} = \frac{\alpha_k^{(j)} \mathcal{N}(x_i; \mu_k^{(j)}, \Sigma_k^{(j)})}{\sum_{k=1}^K \alpha_k^{(j)} \mathcal{N}(x_i; \mu_k^{(j)}, \Sigma_k^{(j)})}. \tag{18}$$

In the following we omit the iteration-superscript as well as the subscript indicating the i-th datapoint for clarity. It is well known that this operation is numerically delicate since the exponential terms can become large and intermediate computations lose precision. The key to a stable implementation is to transform the equation into the log-domain as

$$\log \gamma_k = \log \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k) - \log \sum_{k=1}^K \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k). \tag{19}$$

Let $\xi(x) = \log \sum_{k=1}^K \alpha_k \mathcal{N}(x; \mu_k, \Sigma_k)$, and denote $\|x\|_{\Sigma^{-1}}^2 = x^\top \Sigma^{-1} x$. Using exponential rules, we find

$$\begin{aligned} \xi(x) &= \log \sum_{k=1}^K \alpha_k ((2\pi)^m \det \Sigma_k)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \|x - \mu_k\|_{\Sigma_k}^2\right) \\ &= \log \sum_{k=1}^K \exp\left(-\frac{1}{2}\left(\|x - \mu_k\|_{\Sigma_k}^2 + \log\det\Sigma_k + m\log(2\pi)\right) + \log \alpha_k\right). \end{aligned} \tag{20}$$

Before advancing, let $\Lambda_k = \Sigma_k^{-1}$ denote the precision matrix of the k-th component, and let L_k denote the Cholesky decomposition of Λ_k, i.e. $\Lambda_k = L_k L_k^\top$. Using determinant

Table 2: `numpy` API references for useful functions. All of these functions can handle batch dimensions.

$\Sigma^{-1} = \Lambda$	`numpy.linalg.inv`
$\Lambda = LL^\top$	`numpy.linalg.cholesky`
$\log\det L$	`numpy.linalg.slogdet`
AB (Matrix mult.)	`numpy.matmul` (`@` operator)

rules, we find $\det\Sigma_k = (\det\Lambda_k)^{-1} = (\det L_k)^{-2}$. In addition, the Cholesky decomposition allows to compute the (squared) Mahalanobis distance efficiently:

$$\begin{aligned} \|x-\mu_k\|^2_{\Sigma_k^{-1}} &= (x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) = (x-\mu_k)^T L_k L_k^T(x-\mu_k) \\ &= (L_k^T(x-\mu_k))^T L_k^T(x-\mu_k) = \left\|L_k^T(x-\mu_k)\right\|_2^2 . \end{aligned} \tag{21}$$

Substituting yields

$$\xi(x) = \log\sum_{k=1}^K \exp\underbrace{\left(-\frac{1}{2}\big(\|L_k(x-\mu_k)\|_2^2 + m\log(2\pi)\big) + \log\det L_k + \log\alpha_k\right)}_{:=\beta_k(x)} \tag{22}$$

This can be computed in a numerically stable way by utilizing the well-known LogSumExp-trick

$$\begin{aligned} \xi(x) &= \log\sum_{k=1}^K \exp\beta_k(x) \\ &= \max_k \beta_k(x) + \log\sum_{k=1}^K \exp(\beta_k(x) - \max_k\beta_k(x)) =: \operatorname{LogSumExp}(\beta(x)), \end{aligned} \tag{23}$$

where $\beta(x) = (\beta_1(x),\dots,\beta_K(x))^\top$, which ensures that the largest exponentiated term is 0. Summarizing, we have

$$\gamma_k = \exp\big(\log\alpha_k\mathcal{N}(x;\mu_k,\Sigma_k) - \operatorname{LogSumExp}(\beta(x))\big), \tag{24}$$

or, even more succinctly by noting that β_k are nothing else than the (weighted) log-probabiliteis of the k-th component,

$$\gamma_k = \exp\big(\beta_k(x) - \operatorname{LogSumExp}(\beta(x))\big). \tag{25}$$

Table 2 summarizes functions that might be helpful in your implementation.

A.2. Wiener Filter

For implementing the fast approximate MAP (Algorithm 1), first observe that finding the responsible component (Line 3), amounts to (using previously introduced notation) $\arg\max_k \beta_k(Ex^{(j)})$, where you can reuse your implementation for β. Second, observe that essentially everything needed in Line 4 can be precomputed:

$$\tilde{x} = (\lambda \mathrm{Id}_m + E^T \Sigma_{k_{\max}}^{-1} E)^{-1} (\lambda y + \Sigma_{k_{\max}}^{-1} E \mu_{k_{\max}}) = A_{k_{\max}}(\lambda y + b_{k_{\max}}). \tag{26}$$

You can construct a tensor $A \in \mathbb{R}^{K \times m \times m}$ (i.e. $A_k = (\lambda \mathrm{Id}_m + E^T \Sigma_k^{-1} E)^{-1} \in \mathbb{R}^{m \times m}\ \forall\, k$) and a matrix $b \in R^{K \times m}$ outside of the algorithm loop, and index them properly inside the loop.