Assignment 2 — (Non-)Parametric Density Estimation

Deadline: TBD

November 9, 2023

1. Goal

In this exercise you will become familiar with parametric and non-parametric density estimation. In particular, you will implement the expectation-maximization (EM) algorithm to train a Gaussian mixture model (GMM) on patches of natural images, which will be subsequently used as a prior for Bayesian image denoising. Additionally, you will utilize kernel density estimation and the mean shift algorithm for image segmentation.

2. Parametric Density Estimation and Bayesian Inference (15P)

In Section 2.1, we recall the definition of a GMM and the EM algorithm for training. Section 2.2 outlines the Bayesian inference framework and Section 2.3 shows how we can utilize the GMM prior in this setting to perform denoising.

2.1. Gaussian Mixture Models

The parametric density we utilize in this assignment is the popular GMM. The density function of a GMM on $\mathbb{R}^m$ with $K \in \mathbb{N}$ components is

$$f_\theta(x) = \sum_{k=1}^{K} \alpha_k G_{\mu_k, \Sigma_k}(x). \tag{1}$$

Here, $G_{\mu,\Sigma}$ denotes the Gaussian density with mean μ and covariance matrix Σ

$$G_{\mu,\Sigma}(x) = \left((2\pi)^m \det \Sigma\right)^{-\frac{1}{2}} \exp\left(-\|x-\mu\|^2_{\Sigma^{-1}}/2\right), \tag{2}$$

where we use the shorthand $\|a\|^2_M \coloneqq a^\top M a$. The scalar $\alpha_k \in \mathbb{R}_+$ weights the k-th component and we require $\sum_{k=1}^K \alpha_k = 1$ for proper normalization. The subscript θ in (1) indicates that f is parametrized, and we summarize the trainable parameters as $\theta = (\alpha_k, \mu_k, \Sigma_k)_{k=1}^K$.

2.1.1. Training: Expectation-Maximization

Let $(x_i)_{i=1}^N$ be a collection of N samples from the distribution we are trying to approximate. Maximum-likelihood estimation of the parameters θ of the GMM amounts to solving the optimization problem

$$\min_\theta \sum_{i=1}^{N} -\log\left(\sum_{i=1}^{K} \alpha_k G_{\mu_k,\Sigma_k}(x_i)\right). \tag{3}$$

The EM algorithm is a popular algorithm to solve this problem iteratively: 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 data point as

$$\gamma_{k,i}^{(j)} = \frac{\alpha_k^{(j)} G_{\mu_k^{(j)}, \Sigma_k^{(j)}}(x_i)}{\sum_{k=1}^{K} \alpha_k^{(j)} G_{\mu_k^{(j)}, \Sigma_k^{(j)}}(x_i)}. \tag{4}$$

With the current responsibilities $\gamma_{k,i}^{(j)}$, the maximization step amounts to updating

$$\begin{aligned}
\alpha_k^{(j+1)} &= \frac{1}{N} \sum_{i=1}^{N} \gamma_{k,i}^{(j)}, \\
\mu_k^{(j+1)} &= \frac{\sum_{i=1}^{N} \gamma_{k,i}^{(j)} x_i}{\sum_{i=1}^{N} \gamma_{k,i}^{(j)}}, \\
\tilde{\Sigma}_k^{(j+1)} &= \frac{\sum_{i=1}^{N} \gamma_{k,i}^{(j)} (x_i - \mu_k^{(j+1)})(x_i - \mu_k^{(j+1)})^\top}{\sum_{i=1}^{N} \gamma_{k,i}^{(j)}}.
\end{aligned} \tag{5}$$

To avoid degenerate cases, we regularize the covariance matrices by adding a small multiple of the identity matrix Id_m (the subscript indicates the dimensionality):

$$\Sigma_k^{(j+1)} = \tilde{\Sigma}_k^{(j+1)} + \epsilon \mathrm{Id}_m, \tag{6}$$

where $\epsilon > 0$ is a tunable parameter, which should be set as small as possible while retaining stability. We detail how to implement the computation of the responsibilities in a numerically stable manner in Appendix A.

2.2. Bayesian Inference

In this section we detail how we can estimate the underlying clean signal $x \in \mathbb{R}^m$ from corrupted observations $y \in \mathbb{R}^m$ that follow the signal model

$$y = x + \nu, \tag{7}$$

where $\nu \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathrm{Id}_m)$. Here, $\mathcal{N}(\mu, \Sigma)$ denotes the Gaussian distribution on $\mathbb{R}^m$ with mean $\mu \in \mathbb{R}^m$ and covariance matrix $\Sigma \in \mathbb{R}^{m \times m}$. In other words, each entry in y is corrupted by i.i.d. zero-mean Gaussian noise with variance σ^2.

Viewing x and y as instantiations of the random variables X and Y respectively, the conditional *likelihood* density of Y given $X = x$ is

$$f_{Y|X}(y \mid x) = (2\pi\sigma^2)^{-\frac{m}{2}} \exp\Big(-\frac{1}{2\sigma^2} \|x - y\|_2^2\Big). \tag{8}$$

Bayes theorem allows us to utilize the *prior* density f_X to form the *posterior* density

$$f_{X|Y}(x \mid y) = \frac{f_{Y|X}(y \mid x) f_X(x)}{f_Y(y)}. \tag{9}$$

Here, f_Y is the density of the corrupted random variable Y, sometimes called the *evidence*. In this context, a classical point estimator for the unknown clean signal x given $Y = y$ is a signal $\hat{x}_{\mathrm{MAP}}$ that maximizes the posterior distribution, i.e.

$$\hat{x}_{\mathrm{MAP}} \in \arg\max_x f_{X|Y}(x \mid y). \tag{10}$$

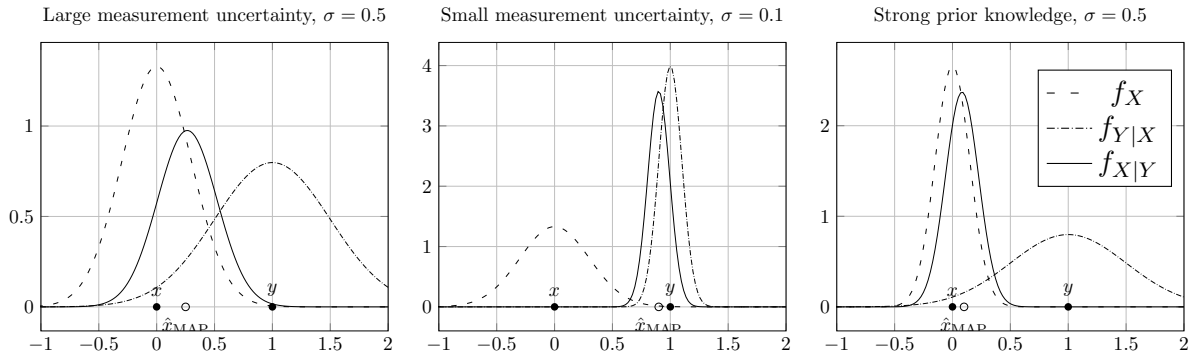


Figure 1: Trading off prior knowledge and measurement uncertainty in Bayesian inference: Prior knowledge tells us the underlying signal follows a Gaussian distribution around x with standard deviation 0.3 (left, middle) or 0.15 (right). A measurement $y = 1$ was acquired with different associated uncertainty. With large measurement uncertainty or a strong prior, $\hat{x}_{\mathrm{MAP}}$ shifts towards 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 disregarding the evidence (which is constant w.r.t. the optimization variable), this can be equivalently written as

$$\hat{x}_{\mathrm{MAP}} \in \arg\min_x \frac{1}{2\sigma^2} \|x - y\|_2^2 - \log f_X(x). \tag{11}$$

In this assignment, the prior f_X will be a GMM that we train on reference data. In the next section, we show how we can efficiently solve the inference problem (11).

2.3. Fast Approximate Inference

Let $y \in \mathbb{R}^m$ be a corrupted observation as defined previously. Utilizing a GMM prior $f_X = \sum_{k=1}^K \alpha_k G_{\mu_k,\Sigma_k}$, solving the inference problem (11) translates to finding

$$\hat{x}_{\mathrm{MAP}} \in \arg\min_x \lambda \|x - y\|_2^2 - \log\left(\sum_{k=1}^K \alpha_k G_{\mu_k,\Sigma_k}(x)\right), \tag{12}$$

where we have replaced $(2\sigma^2)^{-1}$ with a tunable trade-off parameter $\lambda > 0$ to account for modelling inaccuracies (a standard choice would be $\lambda = (2\sigma^2)^{-1}$). This is a challenging optimization problem due to the log-sum-exp structure of the last term. To solve this, in this assignment we use the approximate MAP estimation procedure proposed by [2] and summarized in Algorithm 1, where $\alpha \in [0, 1]$ is a over-relaxation parameter.

Algorithm 1: Fast approximate MAP with a GMM prior.

Data: Corrupted observation y, number of iterations N_{iter}
Result: Denoised signal $\hat{x}^{(N_{\mathrm{iter}})}$

$\hat{x}^{(0)} = y$
for $j = 0, \ldots, N_{\mathrm{iter}} - 1$ **do**
  $k_{\max} = \arg\max_k \log\left(\alpha_k G_{\mu_k,\Sigma_k}(\hat{x}^{(j)})\right)$ `// Responsible component`
  $\tilde{x} = (\lambda \mathrm{Id}_m + \Sigma_{k_{\max}}^{-1})^{-1}(\lambda y + \Sigma_{k_{\max}}^{-1}\mu_{k_{\max}})$ `// Wiener Filter`
  $\hat{x}^{(j+1)} = \alpha\hat{x}^{(j)} + (1 - \alpha)\tilde{x}$ `// Overrelaxed update`

In this assignment, we want to use the above algorithm for patch-based whole-image denoising. Thus, the N-point collection $(x_i)_{i=1}^N$ to train the GMM is given by a training set of N clean

image patches of size $m = w \times w$ (where w is the patch size, i.e. m is always a perfect square). To perform whole-image restoration using the patch-prior, we simply denoise each overlapping patch in the image individually and average the result. In the implementation (for both the EM algorithm and the fast approximate MAP), we work with vectorized image patches.

2.4. Framework & Tasks

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

1. the EM algorithm (4) to (6) (6P) and
2. the fast approximate MAP algorithm (Algorithm 1, 4P).

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

- Use Algorithm 1 to denoise the images in the validation dataset:
  - Try the algorithm for $\sigma \in \{0.05, 0.1\}$ and report PSNR values for each image.
  - For $\sigma = 0.1$, 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).

  State the hyper-parameters of the GMM that you used to get this output.
- Experiment with the hyper-parameters K and m of the GMM (choose at least two each). Illustrate the impact on one noisy image of the validation dataset.
- What are the advantages and drawbacks of increasing K and m?

2.4.1. Notes

EM-Implementation We provide a toy dataset consisting of $N = 1000$ points with $m = 2$ features drawn from $K = 2$ Gaussians, along with a reference model ($\epsilon = 1 \times 10^{-6}$, `max_iter = 50`) and visualization methods to facilitate the implementation of the EM algorithm. If you do not manage to implement the EM algorithm, you can utilize the provided pre-trained models for subsequent tasks.

Performance The reference method finishes 50 EM 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. Qualitative and quantitative denoising performance of the reference model are shown in the appendix.

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

Let $(x_i)_{i=1}^N \subset \mathbb{R}^d$ be a collection of N data points in d-dimensional feature space, drawn from some unknown distribution we wish to estimate. A popular way to construct a probability density from these data points — 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

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

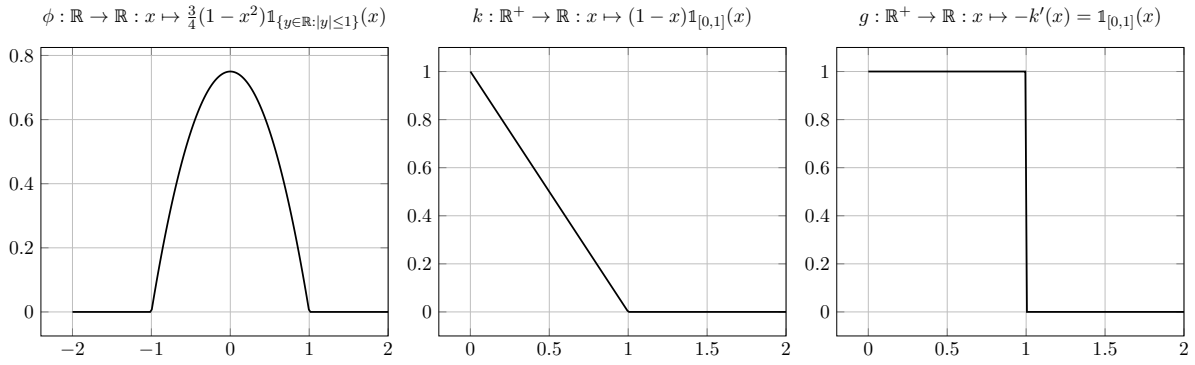


Figure 2: The Epanechnikov kernel ϕ, its profile k and the negative derivative of the profile g in one dimension.

where the *bandwidth* $h \in \mathbb{R}_+$ defines the "fidelity" of the KDE. Here, we consider the Epanechnikov kernel $u \mapsto \frac{3}{4}(1 - \|u\|^2)\mathbb{1}_{\{x\in\mathbb{R}^d:\|x\|\le 1\}}(u)$ that utilizes the indicator function

$$\mathbb{1}_{\mathcal{A}}(x) = \begin{cases} 1 & \text{if } x \in \mathcal{A}, \\ 0 & \text{else.} \end{cases} \tag{14}$$

The radially symmetric Epanechnikov kernel can be 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*. Observe that for our kernel choice, k takes on the particularly easy form $k(l) = (1-l)\mathbb{1}_{[0,1]}(l)$.

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

$$x^{(j+1)} = \frac{\sum_{i=1}^N x_i g\left(\left\|\frac{x^{(j)}-x_i}{h}\right\|^2\right)}{\sum_{i=1}^N g\left(\left\|\frac{x^{(j)}-x_i}{h}\right\|^2\right)}, \tag{15}$$

where $g = -k'$. For the Epanechnikov kernel and its profile, $g(l) = -k'(l) = \mathbb{1}_{[0,1]}(l)$. In other words, g is 1 if its argument is less than or equal to one, and zero otherwise. Thus, the mean shift iterations (15) are

$$x^{(j+1)} = \frac{\sum_{i\in\mathcal{I}(x^{(j)})} x_i}{\operatorname{card}\mathcal{I}(x^{(j)})}, \tag{16}$$

where $\mathcal{I}(x^{(j)}) = \{i\colon \left\|x^{(j)} - x_i\right\|^2 \le h^2\}$ and card denotes the cardinality. The Epanechnikov kernel ϕ (for $d = 1$), the corresponding profile k and its negative derivative g are shown in Fig. 2.

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 mapping from the (square) image domain $\Omega = \{1,\dots,M\} \times \{1,\dots,M\}$ to an RGB triple. For each pixel $p \in \Omega$, we define the features as

$$p \mapsto \left((F(p))^\top, o_\zeta(p_1), o_\zeta(p_2)\right)^\top \in [0,1]^3 \times [-\zeta,\zeta]^2 \tag{17}$$

where $o_\zeta(x) = (2\frac{x-1}{M-1} - 1)\zeta$, and $\zeta \in \mathbb{R}^+$ scales the positional features. In other words, the features space we are operating over are five-tuple consisting of the original image features (RGB triples) concatenated with the normalized (and scaled) position.

The N-point collection $(x_i)_{i=1}^N$ is given by the features of the input image (i.e. $N = M^2$) and stays fixed during the mean shift iterations (i.e., we do not modify the input image). The iteration variables in the mean shift algorithm are also the features of the M^2 pixels. In other

words, for each individual pixel, the mean shift algorithm should find the associated mode in the KDE defined by the features of all pixels. Notice that during the mean shift iterations, the positional features can change and do, in general, not stay on the equidistant grid defined by Ω and o_ζ. However, for the purpose of visualizing the segmented image, we draw the pixels at their original location. The mean shift algorithm is considered to have finished when the condition

$$\left\| x^{(j+1)} - x^{(j)} \right\|^2 < 1 \times 10^{-6} \tag{18}$$

holds.

3.2. Framework & Tasks

The file `mean_shift.py` sets up the feature space you. You have to implement the mean shift iterations (16) as well as the stopping criterion (18) (5P). In the report, you should discuss the following points (5P):

- Show and discuss the results for $\zeta \in \{1, 4\}$ and $h \in \{0.1, 0.3\}$.
- Find optimal parameters ζ and h for the following tasks. Show and discuss the results.
  - Distinguish 8 classes: Sky, the six clearly visible houses, and the pavement.
  - Distinguish 2 classes: Sky and non-sky.
- Discuss the benefits and drawbacks of parametric and non-parametric density estimation in general.

3.2.1. Notes

We provide two input images with $M \in \{128, 256\}$ respectively. 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 available). You can prototype your implementation on $M = 128$ but should strive to complete the task using $M = 256$ (if you have a GPU). 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 3.0 s (`simul = M ** 2`, see code) and for $M = 256$ in 32.9 s (`simul = M ** 2 // 3`) on an NVIDIA TITAN RTX. For the same parameter choice and $M = 128$, it finishes in 29.8 s on an AMD Ryzen 9 3900X. Qualitative performance is shown in the appendix.

References

[1] D. Comaniciu and P. Meer. "Mean shift: a robust approach toward feature space analysis". In: *PAMI* 24.5 (2002), pp. 603–619.

[2] D. Zoran and Y. Weiss. "From learning models of natural image patches to whole image restoration". In: *Proc. ICCV*. 2011, pp. 479–486.

A. Implementation

A.1. Responsibilities

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

$$\gamma_{k,i}^{(j)} = \frac{\alpha_k^{(j)} G_{\mu_k^{(j)}, \Sigma_k^{(j)}}(x_i)}{\sum_{k=1}^{K} \alpha_k^{(j)} G_{\mu_k^{(j)}, \Sigma_k^{(j)}}(x_i)}. \tag{19}$$

In the following we omit the iteration-superscript (j) as well as the point-subscript i 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

$$\log \gamma_k = \log \alpha_k G_{\mu_k, \Sigma_k}(x) - \log \sum_{k=1}^{K} \alpha_k G_{\mu_k, \Sigma_k}(x). \tag{20}$$

Let $\xi(x) = \log \sum_{k=1}^{K} \alpha_k G_{\mu_k, \Sigma_k}(x)$. Using exponential rules, we find that

$$\begin{aligned} \xi(x) &= \log \sum_{k=1}^{K} \alpha_k ((2\pi)^m \det \Sigma_k)^{-\frac{1}{2}} \exp\left(-\frac{1}{2} \|x - \mu_k\|^2_{\Sigma_k^{-1}}\right) \\ &= \log \sum_{k=1}^{K} \exp\left(-\frac{1}{2}\big(\|x - \mu_k\|^2_{\Sigma_k^{-1}} + \log\det \Sigma_k + m \log(2\pi)\big) + \log \alpha_k\right). \end{aligned} \tag{21}$$

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 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:

$$\|x - \mu_k\|^2_{\Sigma_k^{-1}} = \|L_k(x - \mu_k)\|_2^2 . \tag{22}$$

Substituting yields

$$\xi(x) = \log \sum_{k=1}^{K} \exp\Bigg(\underbrace{-\frac{1}{2}\big(\|L_k(x - \mu_k)\|_2^2 + m \log(2\pi)\big) + \log\det L_k + \log \alpha_k}_{:= \beta_k(x)}\Bigg) \tag{23}$$

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

$$\begin{aligned} \xi(x) &= \log \sum_{k=1}^{K} \exp \beta_k(x) \\ &= \max_k \beta_k(x) + \log \sum_{k=1}^{K} \exp(\beta_k(x) - \max_k \beta_k(x)) =: \mathrm{LogSumExp}(\beta(x)), \end{aligned} \tag{24}$$

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

$$\gamma_k = \exp\big(\log \alpha_k G_{\mu_k, \Sigma_k}(x) - \mathrm{LogSumExp}(\beta(x))\big), \tag{25}$$

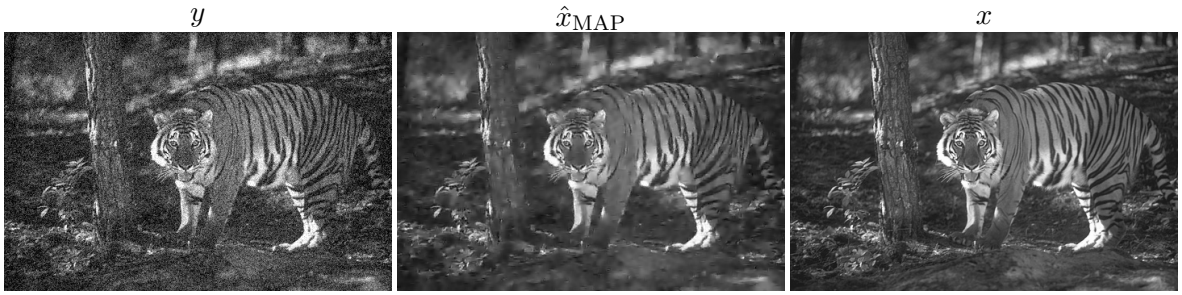
or, even more succinctly by noting that β_k are nothing else than the (weighted) log-probabilities of the k-th component,

$$\gamma_k = \exp\big(\beta_k(x) - \mathrm{LogSumExp}(\beta(x))\big). \tag{26}$$

Table 1 summarizes functions that might be helpful in your implementation.

Table 1: Useful `numpy` API references. 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)



PSNR	26.75	27.15	28.40	31.24	27.07

Figure 3: Qualitative and quantitative Bayesian denoising results. The PSNR table is ordered by the index of the image in the validation dataset.

A.2. Wiener Filter

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

$$\tilde{x} = (\lambda \mathrm{Id}_m + \Sigma_{k_{\max}}^{-1})^{-1}(\lambda y + \Sigma_{k_{\max}}^{-1}\mu_{k_{\max}}) = A_{k_{\max}}(\lambda y + b_{k_{\max}}). \tag{27}$$

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

B. Reference Results

Fig. 3 shows qualitative and quantitative results for Bayesian denoising with a GMM prior. Here, $\sigma = 0.1$ and we used $K = 10$ components in $m = 25$ dimensions. The trade-off parameter was set to $\lambda = \sigma^{-2}$. An example of mean-shift image segmentation using $\zeta = 1$ and $h = 0.3$ is shown in Fig. 4.

Figure 4: Image segmentation using mean shift: Input image (left) and segmented image (right).