Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2024

Physics 410/510 Image Analysis: Homework 7

Due date: Wednesday November 13 by 11:59 pm, submitted through Canvas. You'll see in "Assignments" a place to submit a **PDF**.

Reading: [Optional] If you want to read about morphological image processing, or morphological operations, there's a lot you can easily find by Googling. (For example, https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm, and http://graphics.ics.uci.edu/CS111/Slides/woodsandgonzalez.pdf.) Regardless of whether you use MATLAB, its online documentation is very well done and useful: https://www.mathworks.com/help/images/morphological-filtering.html .

Note #1: There's a lot to explore related to deconvolution and to morphological image processing, but I decided to include just one problem about each to make sure you have time to work on your project.
Note #2: Problems #1-3 should involve lots of recycling of past code.

1 Gaussian MLE and number of photons (12 pts.)

(a) (4 pts.) As in HW5 and HW6, simulate images of a point source: 7 × 7 px images with scale = 100 nm/px (i.e. 0.1 μm/px), λ = 510 nm, NA = 0.9, background mean = 10. The true center (x_0, y_0) should be a random number from a uniform distribution between -0.05 μm and 0.05 μm, that is [-0.5, 0.5] pixels, in each dimension. For N_{photon} = 1000, calculate the error in x_{MLE}, $\Delta x = x_{MLE} - x_0$, where (x_{MLE} , y_{MLE}) is your MLE estimate of the particle position. Plot Δx vs. the true x_0. Is there bias? (Just assess this by eye – you don't need to quantify any bias.) [***Hint***: You should find the RMS error, i.e. sqrt($\Delta x^2+\Delta y$ 2) to be less than 0.02 μm. If it isn't, ask for help!]

(b) (6 pts.) Consider a number of photons N_{photon} from 40 to 40,000, making a logarithmically spaced set of at least 10 N_{photon} values in this range. For each N_{photon} value, make M = 100 images again with the true center (x_0, y_0) being a random number uniform over [-0.5, 0.5] in each dimension. Calculate RMS error of the MLE localization (i.e. the square root of the average of $(\Delta x^2+\Delta y^2)$. Submit a log-log plot of RMS error vs. N_{photon}. Include on the graph a plot of $\sigma / \sqrt{N_{photon}}$ vs. N_{photon}, where $\sigma = \lambda / (2\,NA)$.

(c) (2 pts.) Explain why comparing the RMS error to $\sigma / \sqrt{N_{photon}}$ makes sense.

2 Radial-symmetry-based particle localization. (5 pts.) Test the radial symmetry based localization method I noted in class. You don't have to write it yourself – I'll supply a function; see below. As in #1, test this with simulated images.

(a) (1 pt.) Time how long per image the radial-symmetry localization takes. (Again, don't count the image creation time.) State the result, and compare to your results for centroids and MLE from the previous assignment.

(b) (2 pts.) Repeat #1a, …calculate the error in x_c, $\Delta x = x_c - x_0$. Plot Δx vs. the true x_0. . Is there bias?

(c) (2 pts.) Repeat#1b, … N_{photon} from 40 to 40,000 … Calculate RMS error o… Submit a log-log plot of RMS error vs. N_{photon}. Include on the graph a plot of $\sigma / \sqrt{N_{photon}}$ vs. N_{photon}, where $\sigma = \lambda / (2\,NA)$

A RADIAL SYMMETRY LOCALIZATION FUNCTION

Note: This isn't the fastest version of the algorithm – it can be improved by avoiding redundant calls to meshgrid, etc., if one is considering multiple images, but this version is very short and the code is easy to read.

Python

Download the file `radialcenter_ImAnClass.py` from Canvas \Files\Homework and then type , `radialcenter(I);` download the file `radialcenter_ImAnClass.py` and then type:

```
from radialcenter_ImAnClass import radialcenter
```

Calling it is simple:

```
xRS, yRS = radialcenter(im)
```

returns the x and y locations for 2D image `im`, relative to the image center.

MATLAB

Download the file `radialcenter_ImAnClass.m` from Canvas \Files\Homework and then type

Calling it is simple:

```
xRS, yRS = radialcenter_ImAnClass(im)
```

3 Assessing deconvolution. (15 pts.) If we take an image, convolve it with a PSF, and deconvolve it, how well do we recover the original image? Let's try this, using the electron microscope image "mouse_glial_cells_RBurdan_crop.png" that I've posted on Canvas[1]. This is just a 2D image, not 3D, but the idea would be the same for assessing 3D deconvolution. (It would be even slower, however, to calculate in 3D.) We'll pretend the original image is "true," convolve with a Gaussian PSF, incorporate noise, and then **deconvolve** with either the same Gaussian PSF or one that's "wrong."

(a) (1 pt.) Write a function that returns an *N*x*N* PSF array whose intensity is a Gaussian function of distance to the center pixel, with width sigma. Yes, this should look familiar! Submit an image of the PSF with sigma=7 and $N = 35$.

[1] This is a scanning electron microscope image of mouse glial cells, taken by Robert Burdan. I cropped it and removed the scale bar. Source:
https://www.canadiannaturephotographer.com/rberdan_scanning_electron_microscopy2017.html

(b) (3 pts.) Load the original image. If its intensity range is 0-1, multiply by 255 to make the range 0-255. Convolve the image with your Gaussian PSF with sigma=7. (See HW3.) Then replace each pixel with a Poisson random variable of the same mean as the original pixel intensity value (0-255) – i.e. we're pretending that the intensity scale "is" the number of photons. Submit the resulting image.

(c) (3 pts.) Deconvolve! See the note below on "Implementing deconvolution in Python." Use the Richardson-Lucy algorithm with 20 iterations and make sure you get something sensible. (Look at the image.) Calculate the RMSE (root mean square error) between the deconvolved image and the **original** image. Unfortunately, as you'll see from the image, we get "ringing" near the edges and an overall change of scale, so it's not straightforward to calculate the RMSE. Do the following: (i) Consider a "cropped" version of the original and the deconvolved images, ignoring the *N* pixels close to each edge. (*Hint:* what elements of array `x` does "`x[4:-2]`" return?) (ii) Rescale the intensity of the cropped, deconvolved image so its min and max are equal to the min and max of the cropped original image (which are 0-255). Finally, calculate the RMSE between the images, and submit the RMSE value and the deconvolved image. ***Suggestion:*** first do this without worrying about cropping and rescaling, and make sure your code for (c) and (d) works; only then implement cropping and rescaling.

(d) (7 pts 410 students; 4 pts. 510 students) Calculate the RMSE for a number of iterations 5, 15, 25, ..., 205 . Submit a plot of RMSE vs. number of iterations, and comment. *Note:* This will take many minutes to run!

(e) (1 pt.) Submit the most accurate deconvolved image (i.e. the image with the lowest RMSE).

(f) **[510 students]** (3 pts.) How does it look if we use the wrong PSF for deconvolution? Repeat the above (c-e) using sigma = 5 for the deconvolution – i.e. we think the PSF is narrower than it actually is!

I'm moving this to Homework #8

[MOVED] 4 Spot removal. (8 pts.) Download the file "Lichtenstein_imageDuplicator_1963.png" – a painting called "Image Duplicator" by Roy Lichtenstein (1963). (If you're unfamiliar with Roy Lichtenstein's comic-strip-inspired work, see https://en.wikipedia.org/wiki/Roy_Lichtenstein or Google him.) Also download the gray version, which I made by averaging the color channels: "Lichtenstein_imageDuplicator_1963_gray.png". Note the large number of dots! First, play around with various morphological operations, especially dilation, erosion, opening, and closing, applied either to the entire color image or just one of the color channels. See the note below about toolboxes. Your task: Remove the dots on the man's face! (Leaving the rest of the image as intact as possible. Note that perfection is impossible.)

(a) (8 pts. 410 students, 6 pts. 510 students) First, remove the dots as best you can with the grayscale version. For now, don't worry if you also disturb the dots in the eyes or eyebrows, or if the lettering is messed up. Can you do this in one line of code? Write the relevant lines of code and also show your output image.

(b) (2 pts.) **[510 students]** Now consider the color image. See if you can figure out a way to remove *only* the red dots. This takes some thinking about what's different between the

different color channels. I'm just assigning this only 2 points not because it's easy (it isn't!), but because it's not important.

Implementing deconvolution in Python

We'll use the "restoration" function in the scikit image library:

```
from skimage import restoration
```

To deconvolve using Richardson-Lucy, we need to specify the input image (im), the psf (psf), and the number of iterations. Specifying "clip=False" avoids problems with intensity ranges being 0-255 rather than 0-1:

```
im_deconvolve = restoration.richardson_lucy(im, psf,
    iterations=number_of_iterations, clip=False)
```

If this doesn't work, try:

```
im_deconvolve = restoration.richardson_lucy(im, psf,
    num_iter=number_of_iterations, clip=False)
```

(There are different versions of scikit image with different syntax.)

Implementing morphological operations in Python

As noted in class, morphological operations are in the scikitimage package:

```
from skimage.morphology import (erosion, dilation, disk, closing)
```

etc.

A neighborhood that's a disk of radius 15 would be

disk of radius 4

```
ste = disk(15)
```

Erosion with that "structuring element" on image "im" would be

```
im_erode = erosion(im, ste)
```