Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2024

Physics 410/510 Image Analysis: Homework 6

Due date: Wednesday November 6 by 11:59 pm, submitted through Canvas. You'll see in "Assignments" a place to submit a **PDF**.

Reading: *None.*

1 Cilia and temporal filtering. (14 pts) This exercise doesn't involve parameter estimation, simulated images, or any of the recent topics. It's intended to provide practice with the "old" topic of treating images as arrays of numbers. Download "**cilia_movie_crop.avi"** or "**.tif**" from Canvas, in \Images. This is a movie of the area around what will become the spine in a zebrafish larva, from Beth Bearce in Dan Grimes' lab here at UO. (The scale is 9 px/μm; the total width is 44 μm.) The movement is due to cilia, whiplike cellular appendages that bend and sway to move fluid. Cilia in your esophagus drive mucus up to your throat, for example, and cilia-driven flows early in animal development seem crucial for enabling animals like you and me to have the proper shape. It's probably convenient to *watch* the .avi using whatever viewer you want (FIJI/ImageJ is great), but use the multipage TIFF for analysis[1]. (A multipage TIFF is *one* file that contains a stack of images.) Load the image stack this movie as a 3D array of numbers – see the note at the end of the assignment on how to load it. Notice that it's hard to see the cilia and to figure out how they're moving. What can we do to make their motion more visible? This is a very "real-world" problem for which there's no standard recipe. Try a few things, described below. Please make your filenames self-explanatory.

(a) (4 pts.) Notice that most of the pixels are boring – their value doesn't change throughout the movie. What if we subtracted the median from each pixel – not the median in a *spatial* neighborhood, but the median value in *time*? The boring (x, y) locations are always similar to their median, so subtracting the median would give something near zero. The interesting (x, y) locations vary in intensity, and so will be at times quite positive and at times quite negative when the median is subtracted. Subtract the median from each pixel in each frame. (*Note:* you needn't loop through each pixel! Median functions can act on a specified array dimension, and you can subtract one 2D array from another, all at once.) Submit (i) the median image, and (ii) a histogram of all the pixel values in the subtracted 3D image array.

(b) (2 pts.) Looking at the histogram, pick (by eye) the range of intensities that seems relevant, and scale this to [0, 255]. In other words, apply the linear transformation $I_{out} = A + B\, I_{in}$ such that the minimum relevant value of the "input" I_{in} becomes zero, and the max becomes 255. Convert the values to integers and export your new image sequence as a multipage TIFF file. Submit this image file in addition to your PDF – call it `cilia_sub_rescale_[YourName].tiff`. See the note below on reading and writing multipage TIFFs in Python.

[1] I think the .avi has been compressed twice, with lossy compression.

(c) (2 pts.) Calculate the standard deviation of each (x, y) pixel over time, creating a single 2D image that shows this. In other words, pixel intensity [i,j] of the 2D image should be the standard deviation of the pixels intensities at position [i,j] in the 3D image series. Submit this image. (*Hint:* Making the array should be one line of code.)

(d) (4 pts.) Hopefully from watching the result of (a) you can see filament-like things are moving. Can you enhance this further? How could you weight different pixels differently, or do some sort of averaging, or otherwise make things clearer? Think about this, play around with the arrays, and submit the best thing you can come up with as multipage TIFF image, along with a short description. Obviously, there is no "correct answer."

(e) (2 pts.) Do the cilia look like they are moving "back and forth" with the same form, or not? (In other words, does the shape of a cilium look the same on the forward and reverse stroke? Does the movie look the same if played forward and backward?) I found a region near the center to be clearest to see this. Answer as best you can -- showing some snapshots might help, but you don't have to.

NOTE – read this before doing Problem 2: Problem 2 involves numerical minimization of a function – in other words, searching through parameter space to find the parameters that give the minimal value of the function. I have written an example of this in Python, which we saw in class: "`example_numerical_MLE.py`". I also wrote a MATLAB version, "`example_numerical_MLE.m`". Download either one from Canvas; read and run it.

2 MLE practice. (6 pts.) Let's practice numerical maximum likelihood estimation. Suppose `x` is an array of *N* linearly spaced values from -3 to 4 (Python: `x = np.linspace(-3,4,N)`), `A` is an array: $A = 3|x\text{-}x_0|^b + 4$ (Python: `A = 3*(np.abs(x-x0)**b) + 4`), and `y` is an array of the same size as A such that y[j] = a Poisson-distributed random variable with mean A[j]. Write code to calculate the MLE estimator for the two parameters x_0 and b, minimizing your objective function starting from some initial parameter guess. *Note:* your objective function will require both `x` and `y` as inputs, and calling it will use "`args = (x, y)`". Run this on simulated data with *N* = 20 and true values $x_0 = 0.75$ and b = 2.2.

Submit your code.

a) What did you choose for initial guesses for the parameters? Why?

b) How close are the estimated x_0 and b to the true values? Run your code a few times, and "by eye" comment on the accuracy. (Or if you want, run it a lot and calculate the RMSE.)

c) Plot the simulated y vs. x datapoints together with the function $3|x\text{-}x_0|^b + 4$ for both the true and estimated parameter values. Submit the plot.

d) 510 students: Calculate the objective function for a grid of parameter values $x_0 = 0.1$ to 10 in steps of 0.1, and b = 0.1 to 10 in steps of 0.1. Submit a plot of this – it could be a surface plot, or a 2D plot with color indicating the value of the objective function.

(By the way: this is *almost* the same as MLE for an image!)

Images for #3- For the following problems, you'll use the same simulated single-particle image function that you wrote in Homework 5. Be sure that it works, with the true center position input in some sensible units, and ask questions if it doesn't! For all of these exercises, create simulated images that, unless otherwise specified, have;

- 7×7 px images with scale = 100 nm/px (i.e. 0.1 μm/px), λ = 510 nm, NA = 0.9
- Number of photons = 1000
- M = 100 simulated images for each condition

3 Centroid localization timing. (2 pts.) Re-run your center-of-mass (centroid) localization code on simulated 2D single-molecule images as in Homework 5. Use the parameters described above, and have the true positions be random numbers between -0.05 μm and 0.05 μm in x and y. Assess how long per image the centroid localization takes – in other words, simulate *M* images and note total time / *M*. (Note the timing functions we used in HW3) Note that you should **not** include the time needed to *create* the simulated images. First create a (3D) array of simulated images and then loop through these, running your centroid localization function on each "slice." Submit one number: the centroid computation time per image.

4 Gaussian MLE for particle localization! (14 pts.) Write a 2D MLE particle localization function! In other words, write a function that returns the MLE estimate for the particle center (xc, yc) and other parameters given a 2D image in which each pixel has Poisson-distributed random values with a mean that depends on position as:

$$A(x,y) = A_0 exp\left[-\frac{(x-x_c)^2+(y-y_c)^2}{2\sigma^2}\right] + B$$

(In other words, a Gaussian approximation to the PSF.) Then apply Gaussian MLE localization to the same sorts of simulated images that you used earlier.

(a) (3 pts.) Show the important parts of your code (the objective function and the lines that call for minimizing it).

(b) (1 pt.) Repeat #3, now using MLE localization. Time how long per image it takes.

(c) (10 pts. For 410, 5 pts. For 510). Repeat HW5 #3c, now using MLE localization, making histograms for $N_{photons}$ = **1000,** (x_0, y_0) = (0.0, 0.0) px, (x_0, y_0) = (0.3, 0.0) px, and (x_0, y_0) = (-0.3, 0.0) px. Does the MLE estimate look unbiased? (You don't have to do $N_{photons}$ = 100,000.) You should be able to re-use a lot of your code from HW5!

(d) [510 students] (5 pts.) Repeat HW5 #3d, now using MLE localization. For $N_{photons}$ = **1000,** plot a graph of the mean Δx as a function of position *p*, … Describe what you find: how does the error depend on *p*? (You only need to do mean background = 10.)

Next: As in HW5, you'll consider RMSE error vs. number of photons. I'll put this into HW7, but I suggest starting it as soon as that's posted, because it will be a straightforward extension of these problems!

NOTE: Reading and writing multipage TIFFs in Python.

To read or write a multipage TIFF (i.e. *one* file that has all the images in the image stack), you can use the scikitimage toolkit.

```
    from skimage import io # input output sub-package
```

Reading multipage TIFFs:

```
    im = io.imread('cilia_movie_crop.tif', plugin="tifffile")
```

Caution: What are the dimensions of the resulting array?

Saving the 3D stack "im_rescale" to a multipage TIFF:

```
    io.imsave('cilia_movie_sub_rescaled.tif', im_rescale)
```

Writing AVIs in Python can be tricky, by the way.