Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2024

Physics 410/510 Image Analysis: Homework 6

Solutions

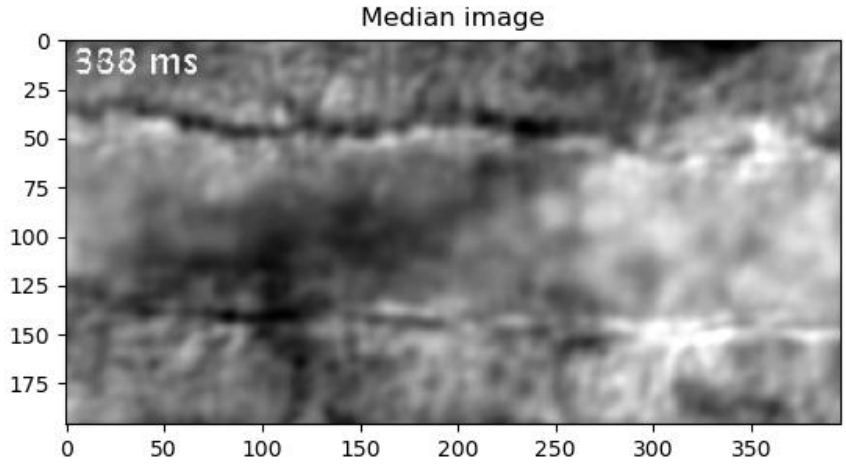
1 Cilia and temporal filtering. (18 pts) ....

(a) (4 pts.) ...Subtract the median from each pixel in each frame. ... Submit (i) the median image, and (ii) a histogram of all the pixel values in the subtracted 3D image array.

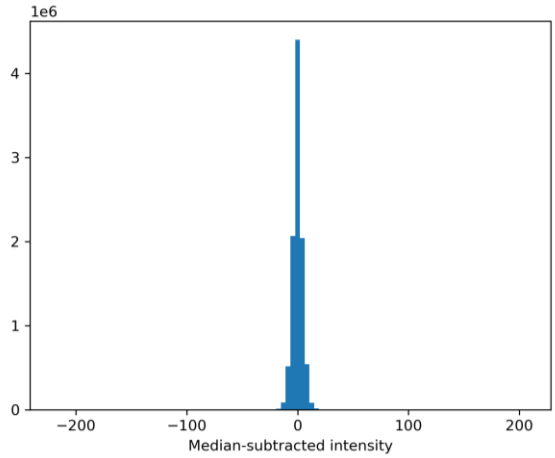
We calculate the median along the stack dimension, which is axis 0 of the 3D array:

```
im_med = np.median(im, 0)  # median along the stack dimension
```

The median image looks like:



Subtracting, we get the following histogram:



Note that there are "extreme" pixels, mostly due to the timestamp, but most pixels are in a narrow range around zero.

(b) (2 pts.) Looking at the histogram, pick (by eye) the range of intensities that seems relevant, and scale this to [0, 255]. ...Convert the values to integers and export your new image sequence as a multipage TIFF file. Submit this image file.

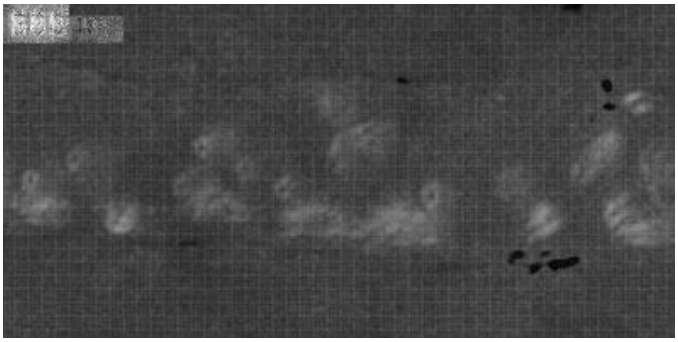
By eye, roughly [-15, 15] seems like where most of the intensity is. Scale so that this -> [0, 255]. A simple linear transformation

```
im_min = -15
im_max = 15
im_rescale = 255*(im_sub-im_min) / (im_max-im_min)
```

(c) Calculate the standard deviation of each (x, y) pixel over time, creating a single image that shows this. Submit this image.

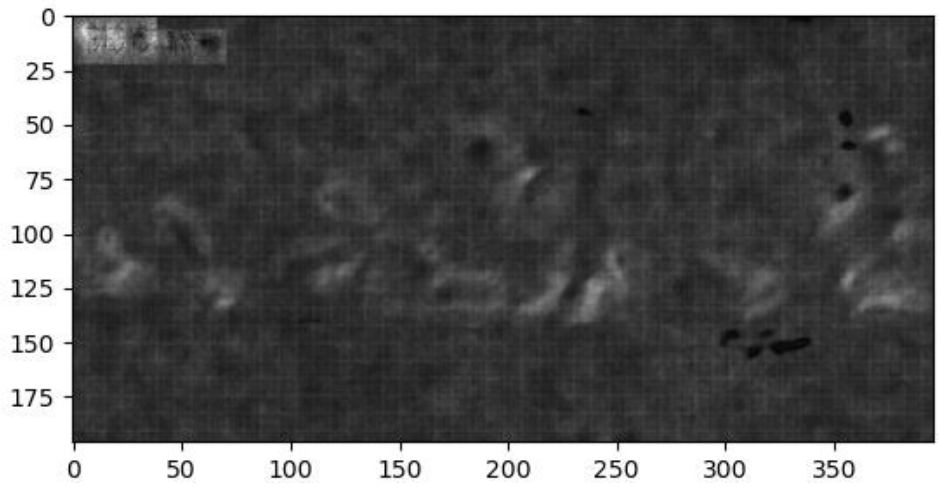
```
im_stdDev = np.std(im_rescale,0)
```

This looks like:



(d) (4 pts.) Hopefully from watching the result of (a) you can see filament-like things are moving. Can you enhance this further?

There are many possibilities... I did a spatial median filtering (9 px) and then weighted the intensities by the standard deviation across time, to accentuate the moving parts.
Here's an example frame:



(e) Do the cilia look like they are moving "back and forth" with the same form, or not?

We'll discuss this in class.

2 MLE practice. Here are a few problems for practice with numerical maximum likelihood estimation.

a) (2 pts.) Consider some random numbers x = {x[0], x[1], x[2], ..., x[N-1]} generated from a Poisson distribution of mean A. Write code to numerically calculate the maximum likelihood estimate for A using the expression for -log(p) we derived in class, and verify that this estimate is sensible.

For example:

```
N = 200
A = 12

x = np.random.poisson(A, (N,1))
print('Mean of x: ', np.mean(x), 'A: ', A)

# Here's the function we want to minimize:
def objfun(params, x):
    # Sum of -log(p) for each measurement
    nsum_logp = np.sum(params[0] - x*np.log(params[0]))
    return nsum_logp

# Initial guess: mean of x
params0 = np.array(np.mean(x))

# Bounds on the parameters: 0 to Infinity
bnds = ((0, None), ) # Oddly, comma is necessary
```

```
results = spoptimize.minimize(objfun, params0, args = (x), bounds =
bnds)
# Here is the best fit parameter value for A
best_A = results.x[0]
print(f'True A: {A}, Best fit A: {best_A:.2f}')
```

(b) Multiplying by some constant C, the MLE estimate for A is now C*A, as we'd expect.
(c) Here's the important part of the code:

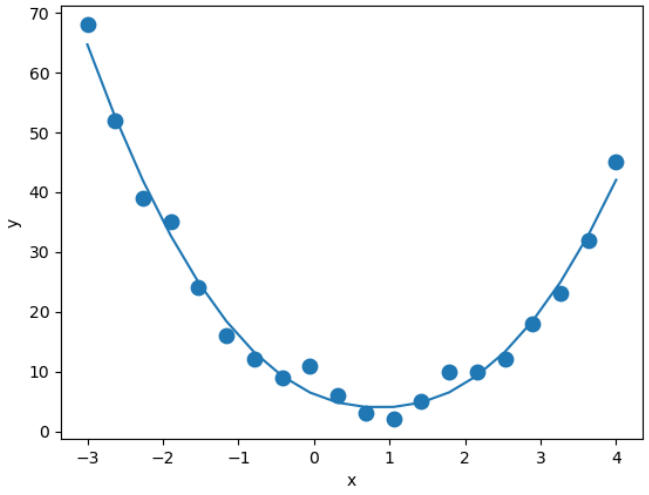
```
def weirdFunction(x, x0, b):
    A = 3*(np.abs(x-x0)**b) + 4
    return A

# Here's the function we want to minimize:
def objfun(params, x, y):
    # Sum of -log(p) for each measurement
    A = weirdFunction(x, params[0], params[1])
    minus_logp = A - y*np.log(A)
    nsum_logp = np.sum(minus_logp)
    return nsum_logp

params0 = np.array([np.mean(x), 1])
# Bounds on the parameters: -Infinity to Infinity, 0 to Inf
bnds = ((None, None),(0.1, None))

results = spoptimize.minimize(objfun, params0, args = (x, y), bounds =
bnds)
```

You should find the best-fit x0 to be between around 0.6 and 0.9, and b between about 2.1 and 2.3. Plot:



3 Centroid localization timing. …. Assess how long per image the centroid localization takes – in other words, simulate *M* images and note total time / *M*. (… Submit one number: the centroid computation time per image.

Solution

I forgot to specify Background = 10. In any case, you should get something like:
Centroid time per image = 0.000029 seconds (on my laptop) – i.e. very small!

4 Gaussian MLE for particle localization! (14 pts.) Write a 2D MLE particle localization function!

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

Solution

```
# Here's the function we want to minimize
def objfun(params, x, y, z):
     # Log-likelihood for Poisson-distributed intensity values, with a
     #  Gaussian spatial distribution
     gaussprob = params[4]*np.exp(-((x - params[1])**2 + (y -
params[2])**2)/2/params[3]/params[3]) + params[0];
     Lk = z*np.log(gaussprob) - gaussprob  # log-likelihood; Poisson-
distr.
     return -1.0*np.sum(Lk);

# parameters [B x0 y0 sigma A0]
# initial guesses: smallest z; position 0,0; quarter of image shape (y); max z - min z (or anything
reasonable)
params0 = np.array((np.min(im), 0, 0, 0.25*im.shape[0], np.max(im)-
np.min(im)))
# Bounds on the parameters: 0 to Infinity except for position,
bnds = ((0, None), (None, None), (None, None), (0, None), (0, None))
```

Call with (for one image, in a loop):

```
results = spoptimize.minimize(objfun, params0, args = (x, y, im[:,:,j]),
bounds = bnds)
```

(b) Timing

Solution

On my laptop: MLE time per image = 0. 00404. You should find something around 100x slower than the Centroid calculation.

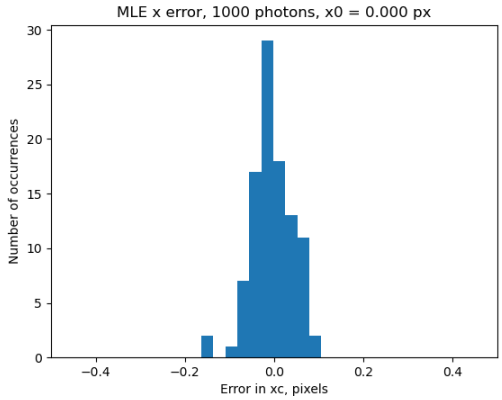
(c)

(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?

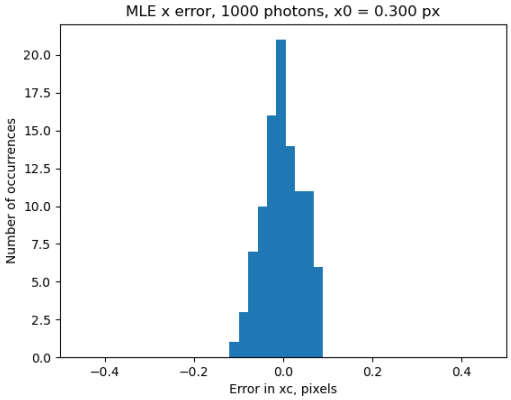
Solution

Here are the histograms:

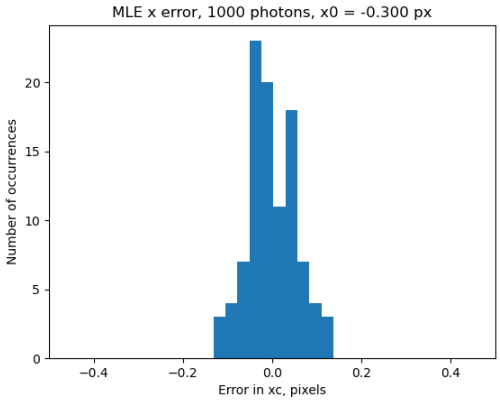
(x_0, y_0) = (0.0, 0.0) px. We see that the error is roughly centered at 0:



(x_0, y_0) = (0.3, 0.0) px: We see that the error is roughly centered at 0!



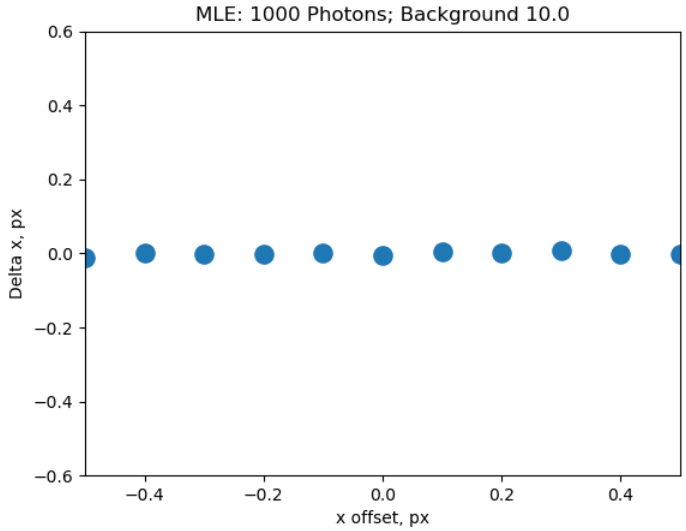
(x_0, y_0) = (-0.3, 0.0) px. We see that the error is roughly centered at 0!



(d) [510 students] (5 pts.) Repeat HW5 #3d, now using MLE localization. For N_{photons} = **1000,** plot a graph of the mean **Δ**x 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.)

Solution

Your plot should look like this:



The error is quite independent of the true object position!