Prof. Raghuveer Parthasarathy
University of Oregon; Fall 2024

Physics 410/510 Image Analysis: Homework 7

Solutions

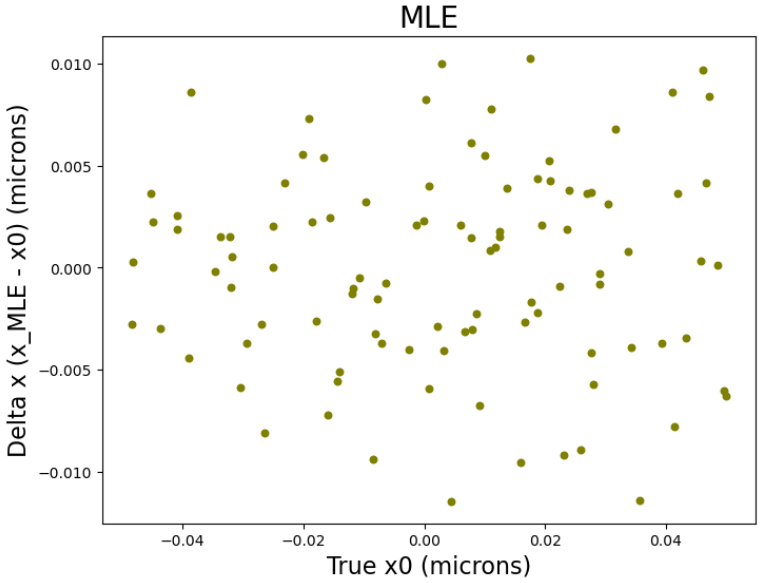
1 Gaussian MLE and number of photons (12 pts.)

(a) (4 pts.) … 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 positions Plot Δx vs. the true x_0. Is there bias?

(a)

Solution

The MLE and simulated image functions are the same as before. You should find something like:



There's no apparent bias!

(b)

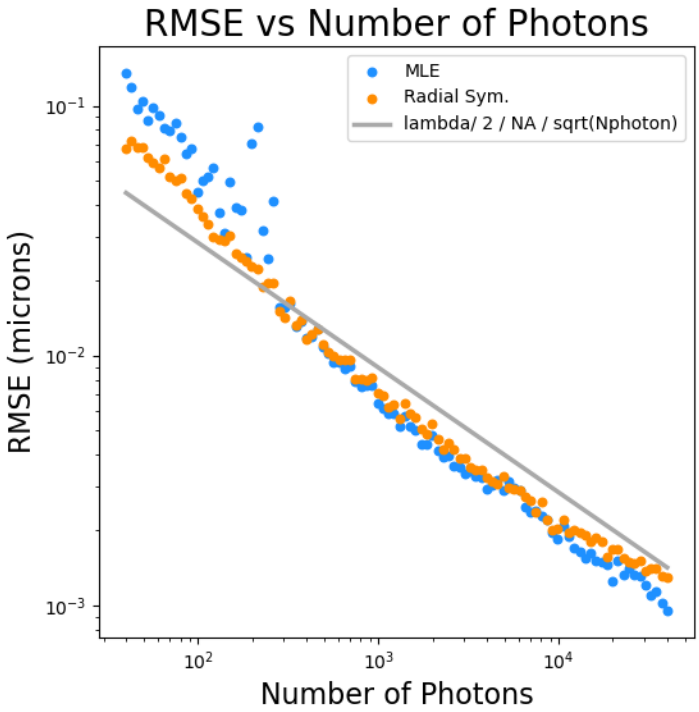
Solution

Again, we re-use our code, now looping over photon number. Your program should look something like this:

```
NNp = 50 # number of "Nphoton" to examine
Nphoton_array = np.logspace(np.log10(40), np.log10(40000), NNp)
bkgPoissMean = 10
M = 100 # number of simulated images
bkgPoissMean = 10 # Mean of additive Poisson background
# Other parameters as above

x1D = np.arange(Nx) - (Nx-1)/2
```

```
y1D = np.arange(Ny) - (Ny-1)/2
x, y = np.meshgrid(x1D, y1D)
# Bounds on the parameters: 0 to Infinity except for position,
bnds = ((0, None), (-N/2.0, N/2.0), (-N/2.0, N/2.0), (0, None), (0, None))

RMSE_MLE = np.zeros((NNp,))
RMSE_RS = np.zeros((NNp,))
for k in range(NNp):
    print(f'Number of photons: {Nphoton_array[k]:.1f}')
    # True positions
    x0  =  np.random.uniform(low=-0.05,  high=0.05,  size=(M,))  #  x-offset,
microns
    y0  =  np.random.uniform(low=-0.05,  high=0.05,  size=(M,))  #  y-offset,
microns
    im =  np.zeros((N, N, M), dtype=float)
    for j in range(M):
        im[:,:,j] = simPointSource(N, scale, fineScale = scale/50,
                        NA=NA, lam=lam, Nphoton=Nphoton_array[k], xc=x0[j],
                        yc=y0[j], bkgPoissMean=bkgPoissMean)
    xMLE = np.zeros((M,))
    yMLE = np.zeros((M,))
    xRS = np.zeros((M,))
    yRS = np.zeros((M,))
    # initial guesses: smallest z; position 0,0; quarter of image shape (y);
max z - min z
    params0  =  np.array((np.min(im),  0,  0,  0.25*im.shape[0],  np.max(im)-
np.min(im)))
    for j in range(M):
        xMLE[j],  yMLE[j]  =  calc_xy_MLE(objfun,  params0,  x,  y,  im[:,:,j],
bnds,
                                          xc=0.0, yc=0.0, checkNonsense=True)
        xRS[j], yRS[j] = radialcenter(im[:,:,j])

    # convert to real units; get RMSE: MLE
    xMLE = xMLE*scale # microns
    yMLE = yMLE*scale
    RMSE_MLE[k] = calc_RMSE(x0, y0, xMLE, yMLE)
    xRS = xRS*scale # microns
    yRS = yRS*scale
    RMSE_RS[k] = calc_RMSE(x0, y0, xRS, yRS)
```

You should get something like the blue points in the graph below for the RMSE of the MLE. Note that it can be poor for low N, failing "catastrophically" for noisy data, not finding the true maximum of the probability.



(c)
Solution

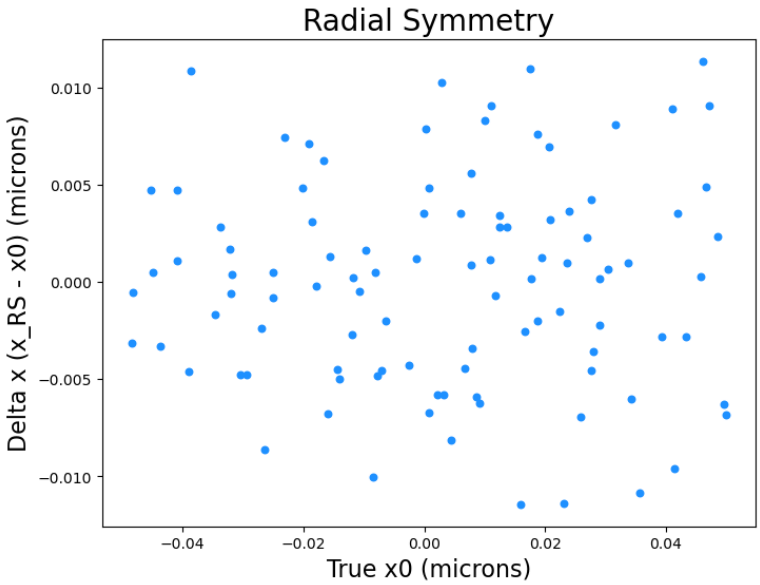
For *N* measurements of a Gaussian function of standard deviation σ, the lowest possible uncertainty in determining the mean is, as noted in class, σ /sqrt(*N*). Our particle image is somewhat like a Gaussian of width σ = λ / (4 NA). (The full width is λ / (2 NA); the σ if we fit it to a Gaussian is about half this. Don't worry about this factor of 2, though.) We'd therefore expect a limit on localization uncertainty to be roughly σ /sqrt(*N*) = = λ / (4 NA) /sqrt(*N*). We see that the MLE estimator and the radial symmetry estimator are fairly close to this, and both, more importantly, have the same, expected, 1/sqrt(*N*) scaling.

2 Radial-symmetry-based particle localization....

(a) Timing
Solution
On my laptop: MLE time per image = 0.0018 seconds . About 10x faster than MLE!

(b)
Solution
It's convenient to do this at the same time as #1b, using the same images!

Let's look at bias



Again, seems unbiased!

(b)

Solution

See the plot for 1(b), above.

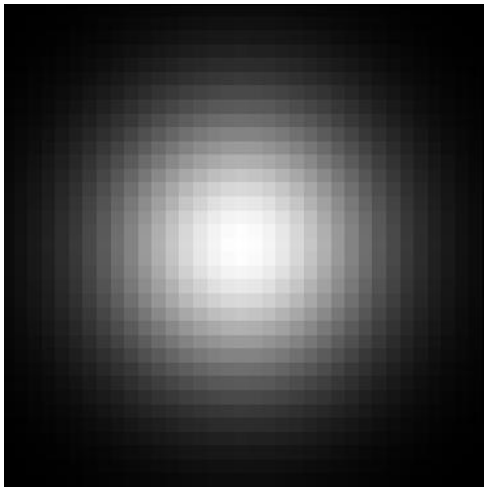
3 Assessing deconvolution. If we take an image, convolve it with a PSF, and deconvolve it, how well do we recover the original image? ...

(a) 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 $N = 35$.

Answer.

Something like this:

```
def make2DGaussian(sigma = 3, N = 19):
    # 2D Gaussian of width sigma, and normalized to sum==1
    x = np.arange(N) - (N-1)/2
    xx, yy = np.meshgrid(x, x)
    gauss = np.exp(-(xx**2 + yy**2)/2/sigma/sigma)
    gauss = gauss/np.sum(gauss)
    return gauss
```

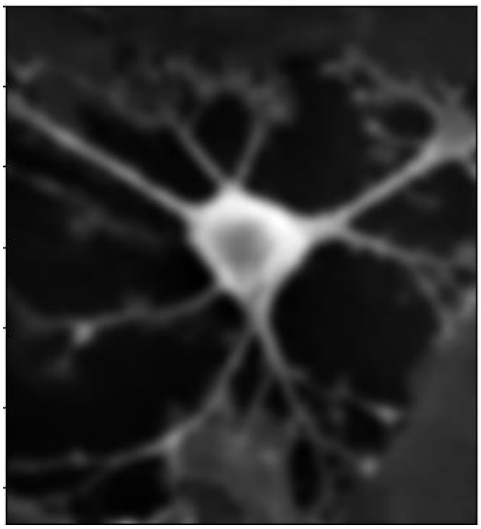


(b) ... Convolve the image with your Gaussian PSF with sigma=7. (See HW2 and HW3.) Then replace each pixel with a Poisson random variable of the same mean as the original pixel intensity value (0-255) .

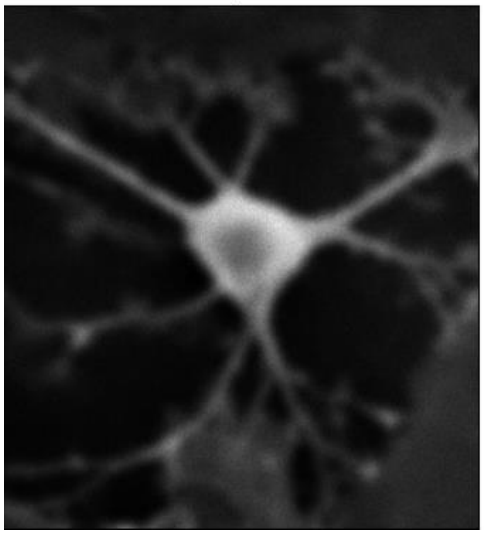
Answer.

```
      # Convolution!
      im_convolve = ndi.convolve(im_original, gaussC, mode='nearest') # As in
HW2, HW3
      plt.figure()
      plt.imshow(im_convolve, cmap = 'gray')
      plt.title('Convolved')

      # Incorporate noise
      im_noise = np.random.poisson(im_convolve)
      plt.figure()
      plt.imshow(im_noise, cmap='gray')
      plt.title('Convolved, with Noise')
```

Convolved

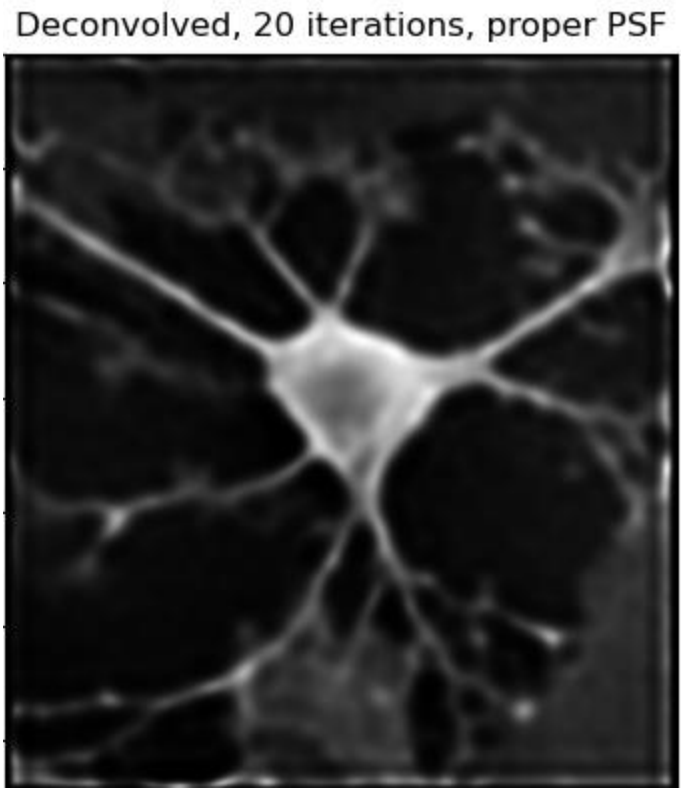


Convolved, with Noise



(c) 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 between the deconvolved image and the **original** image. ...

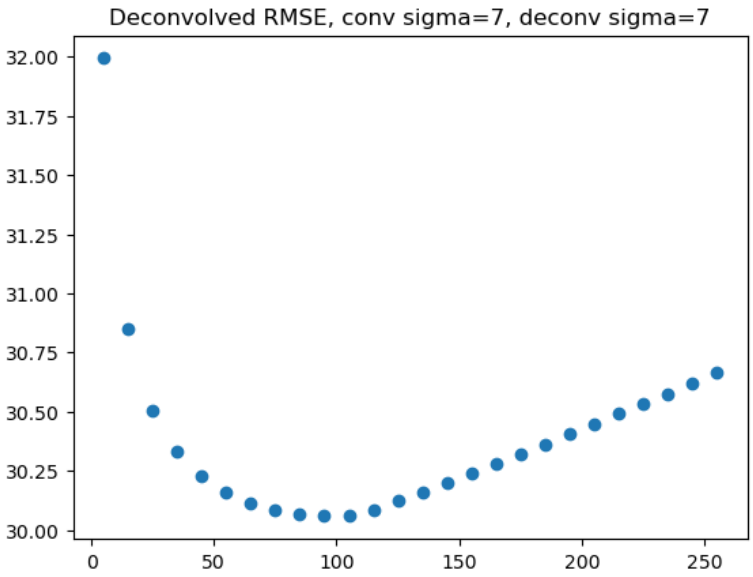
Answer.



RMSE = 30.65
(Note: if you got 30.50, you didn't consider the interior, cropped region.)

(d) 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!

Answer.

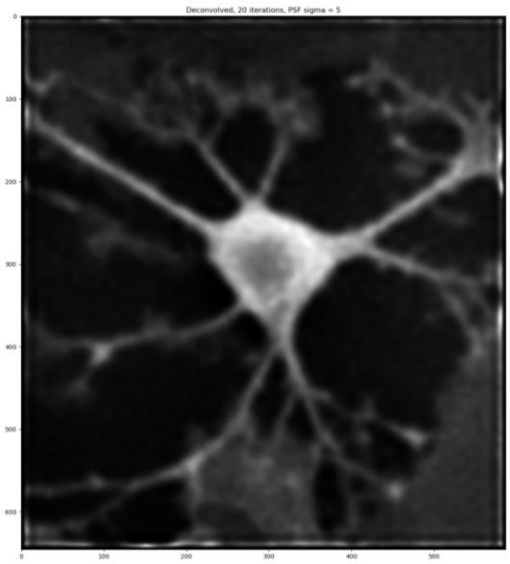


We see that there's a minimum error around 100 iterations.

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

Solution

You should see a deconvolved image that's pretty similar to the "proper sigma" image, but strangely splotchier.



The RMSE vs iteration graph looks much different: more iterations make it much worse!