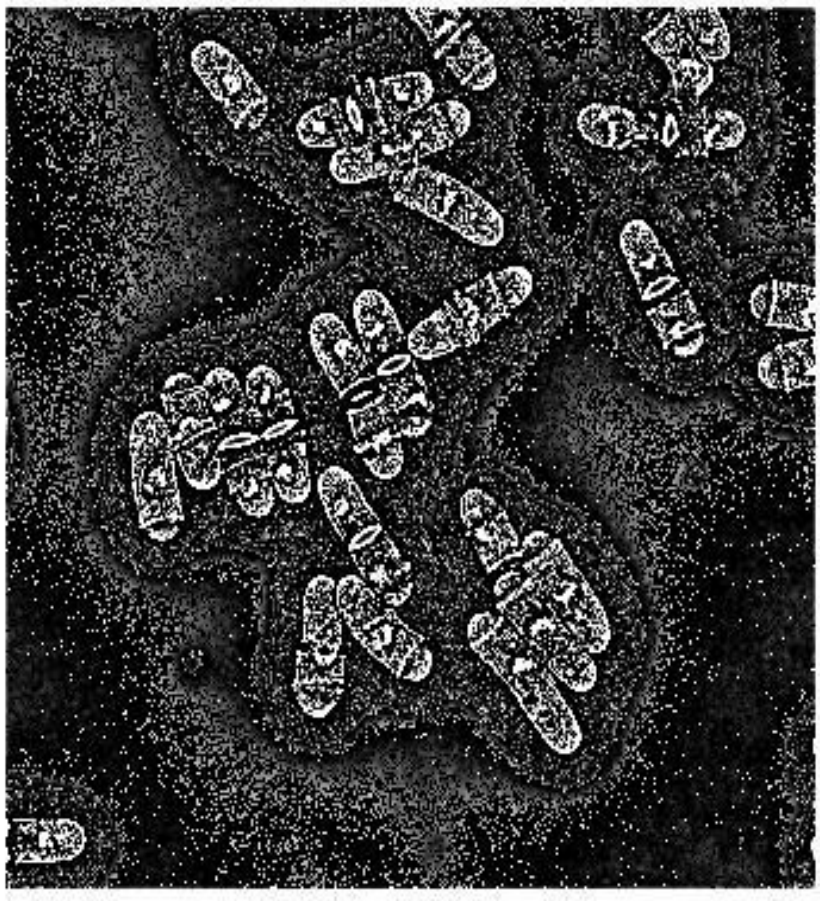
Gaussian filtering, Noise models and the Airy Disk

Programming Assignment # 2



3/4/2016

Phillip Lee

Guruprasad Raghavan

Priyanka J .Raja

Introduction

This assignment is subdivided into three parts and each of these parts allow us to dig deep and understand topics pertaining to 1) Spatial filtering by employing a Gaussian kernel and studying the effects increased values of standard deviation (sigma) on filtering / image smoothing, accompanied by the calculation of derivatives of the smoothed image to visualize the edges in the X and Y direction respectively; 2) Analysis of noise distribution, illumination uniformity, pixel calibration; 3) Plotting Airy disks to understand the significance of illumination wavelength and numerical aperture; Fitting a Gaussian function with each of these airy disks, in order to optimize sampling frequency, and with the future aim of performing subpixel detection.

Part I

Code Instructions

1. Implementation of the Gaussian Filter
    a. The code for this section is found in the *Question1_1.m* file and it takes two *inputs:*
        i. *path _to _the _image* and the values of *sigma* as a vector.
        ii. The *path_to_the_image has* to be specified as a string(in quotes) and if the image is not in the same folder as the code, kindly ensure to include the complete path to the image.
    b. The code displays the original image and the filtered images for the different values of sigma respectively. The function does not return any value/variable.
    c. The code contains the manual implementation of the Spatial filtering approach using the Gaussian filter using for loops, the conv2 function can be used alternatively to speed up the approach. This portion of the code needs to be uncommented if there exists a requirement to use the conv2 function.
    d. Both implementations assure correct results with same pixel values after filtering upto a significant degree of precision.
2. Calculate Image intensity derivatives
    a. The code for this section is found in the *Question1_2.m* file and it takes two *inputs:*
        i. *path _to _the _image* and the values of *sigma* as a vector.
        ii. The *path_to_the_image has* to be specified as a string (in quotes) and if the image is not in the same folder as the code, kindly ensure to include the complete path to the image. The sigma variable can take as input just one value or a list of values as a vector..
    b. The code displays the original image ,filtered image for the different value of sigma and the x and y derivatives of the filtered image along with its gradient magnitude. The function does not return any value/variable.

3. Implementation of anisotropic Gaussian Filter

    a. The code for this section is found in the *Question1_3.m* file and it takes four *inputs:*
        i. *path _to _the _image*
        ii. *sigma* values in the *longitudinal* and *lateral* axis
        iii. *phi* values in a vector

    b. The code displays a series of images for the *phi* values .

Results

a) Implementation of the Gaussian Filter :

   a. The images shown in Figure 1 represent the original image taken from the previous assignment and the corresponding filtered images with the Gaussian filter of different values of sigma = 1,2,5,7 .
   b. The size of the filter N is evaluated for every value of sigma, where **N = 2*(3*sigma) +1.**
   c. The filter is normalized by the sum of the value of the elements of the filter.
   d. The code is manually implemented by using for loops [1]and conv2 which is built in Matlab function.

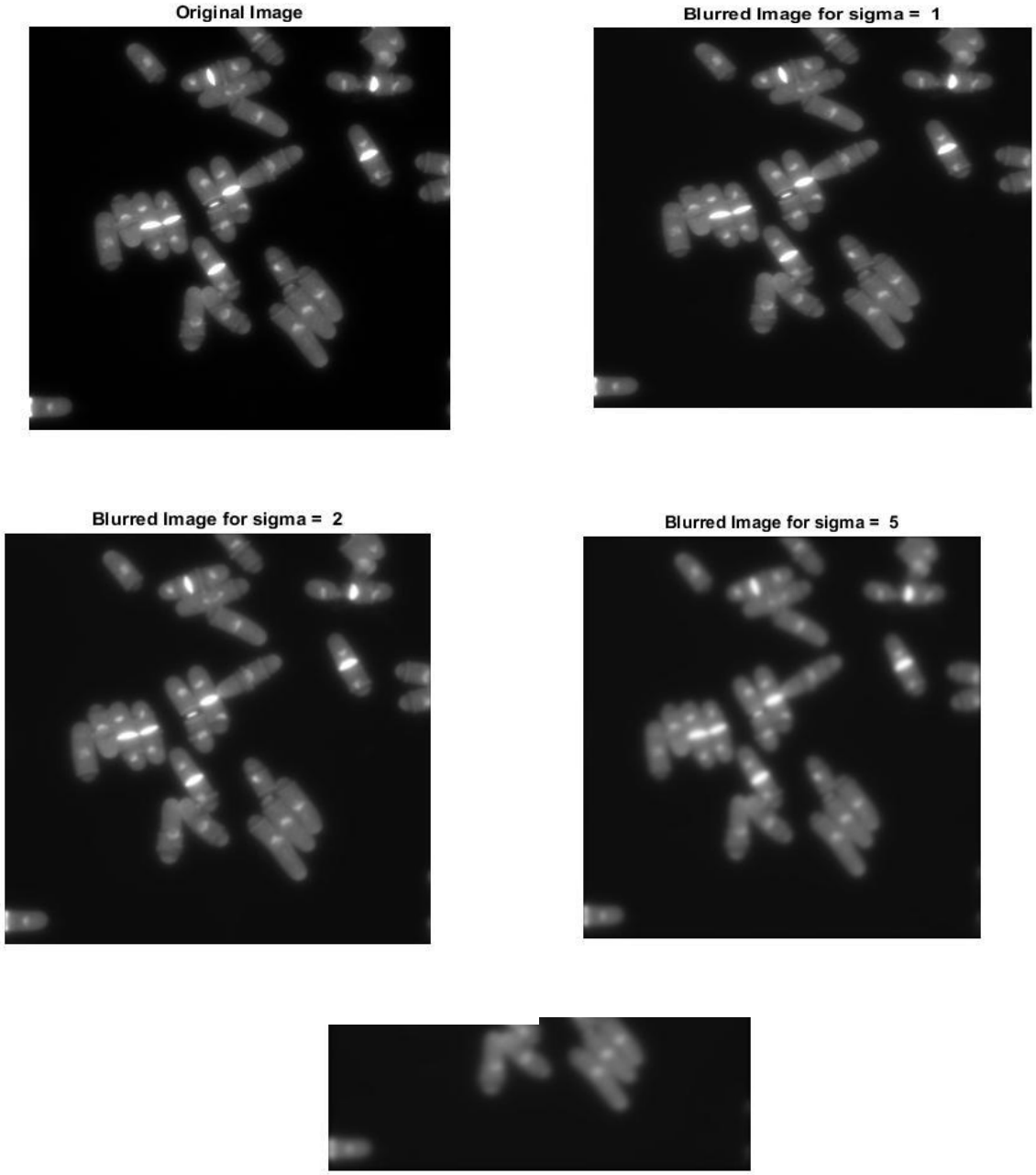
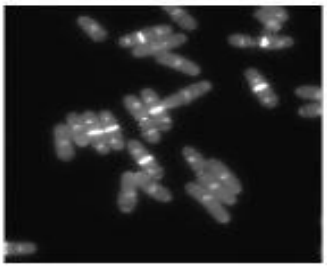


Figure 1: The original image without spatial filtering and the remaining images are images after spatial filtering with different values of sigma = 1,2,5,7.

b) Calculate Image intensity derivatives :

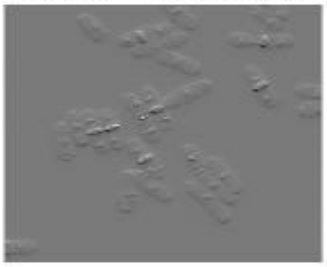
a. The images shown below are represent the x and y derivatives of the Gaussian blurred image obtained by running Question1_2.m using sigma = [1, 2, 5].
b. The central difference is used for every pixel in the image along the X and Y directions to compute the derivative [2].
c. As evident from the images below as the sigma value increases the edges become more and more prominent/distinct in the derivative images in their respective direction.

Sigma = 1

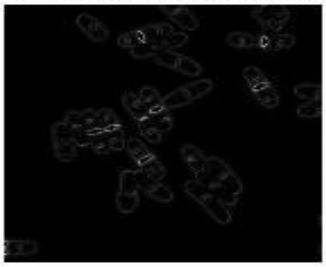


The X - derivative

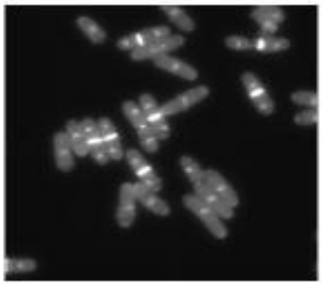
The Y - derivative



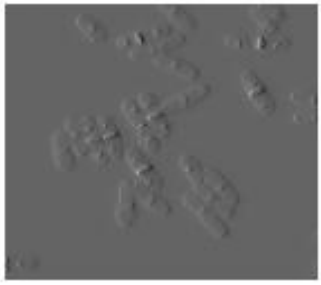
Gradient Magnitude



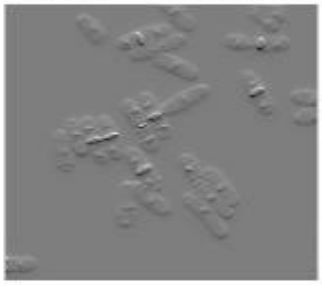
Sigma = 2



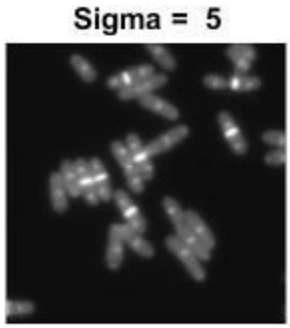
The X - derivative

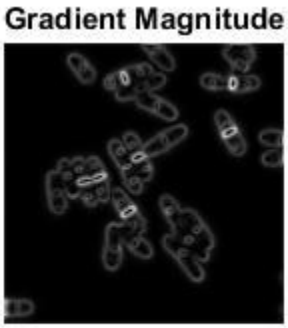


The Y - derivative



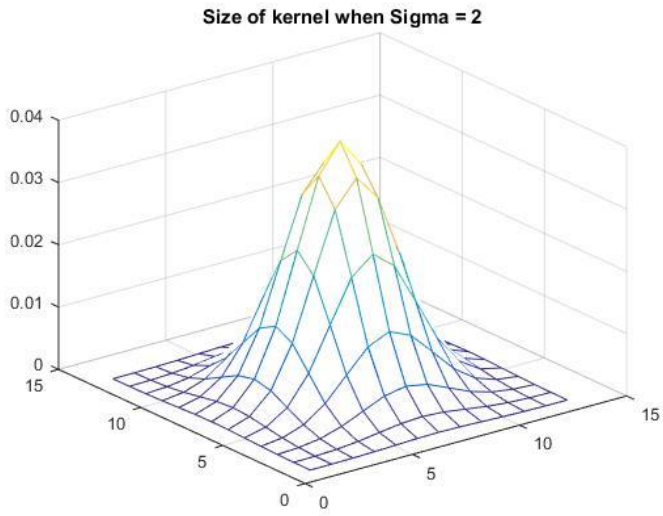
Gradient Magnitude

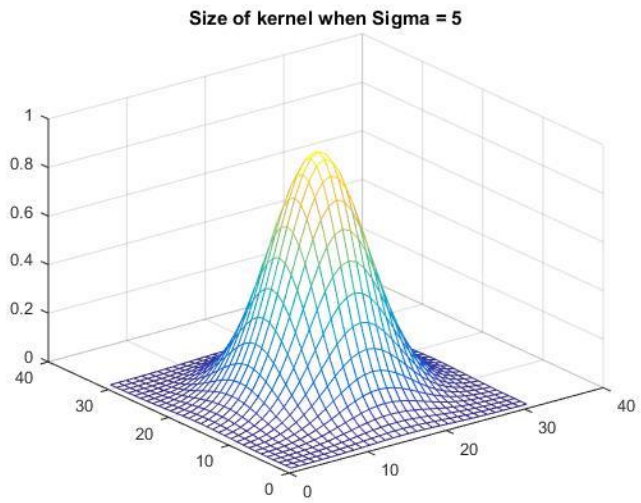




<u>Remarks/</u>Discussion :

- As we filter the image with a Gaussian kernel we filter out noise and some of the sharp features from the image.
- As the sigma increases the image becomes more smooth/blurred and therefore we lose more information/detail from the image and only retain low frequency components of the image.
- As we increase sigma the size of the kernel increases by a relatively large amount as shown below. Therefore the pixel at the center is the average pixel value obtained by the kernel over large area which leads to loss of fine features.





c) *Fast Anisotropic Filtering*

The images shown below are images filtered along two different axis, the x-axis and the t-axis respectively. Here, the standard deviation along the x-axis is 5 units, while along the t-axis is 10 units.

Figure 1: Convolving along x-direction

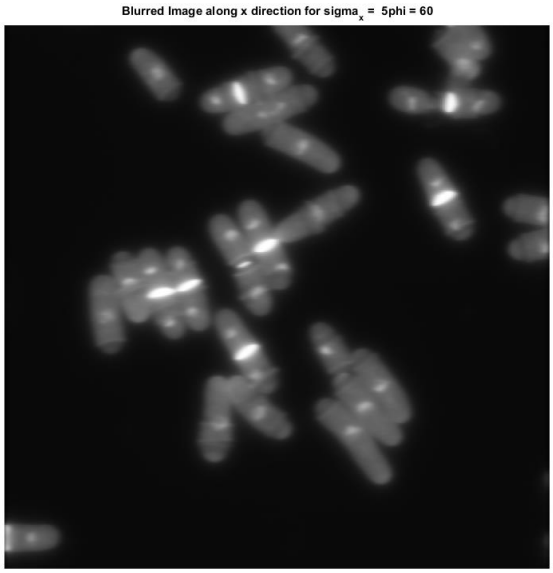


Figure 2: Phi = 30°

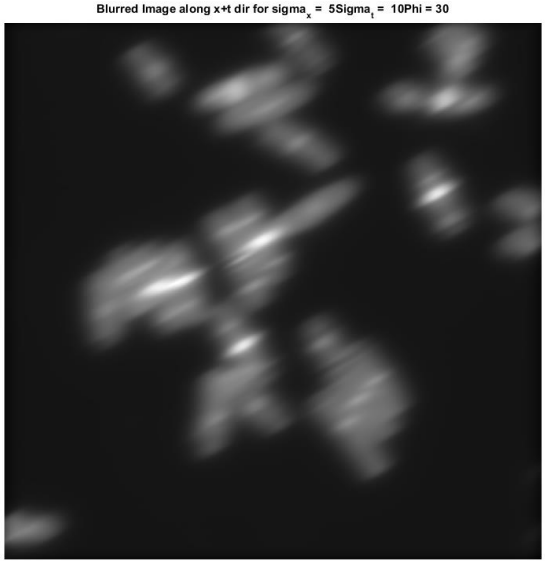


Figure 3: Phi = 60°

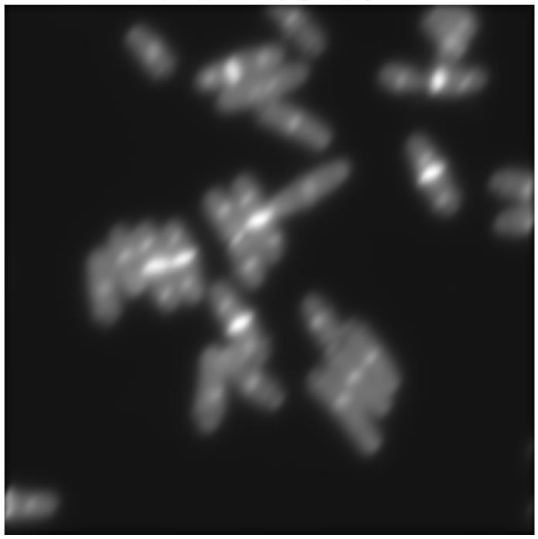


Figure 4: Phi = 90°

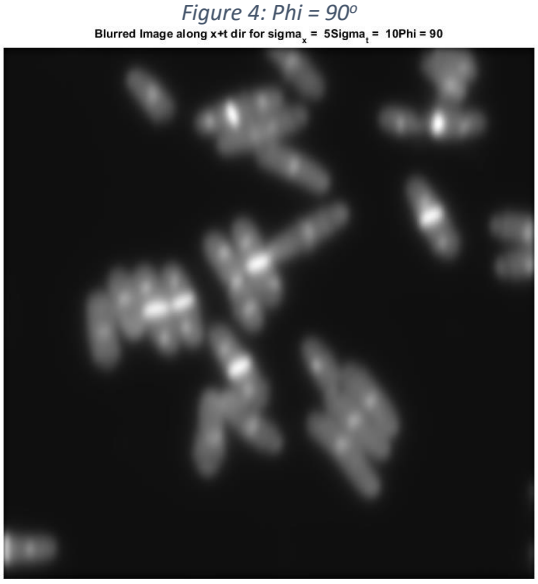


Figure 5: Phi = 120°

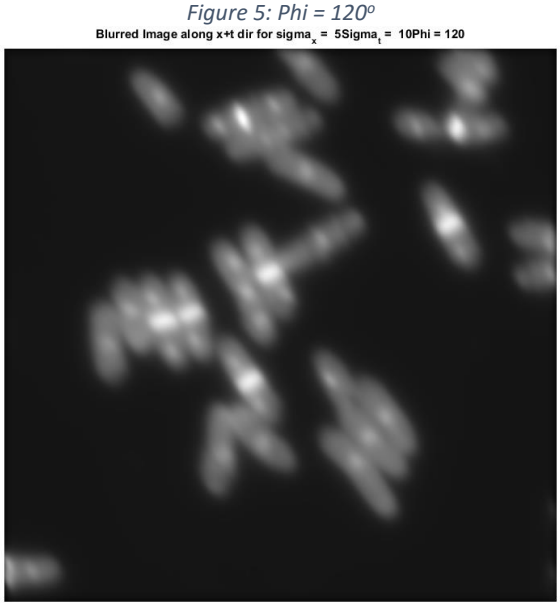
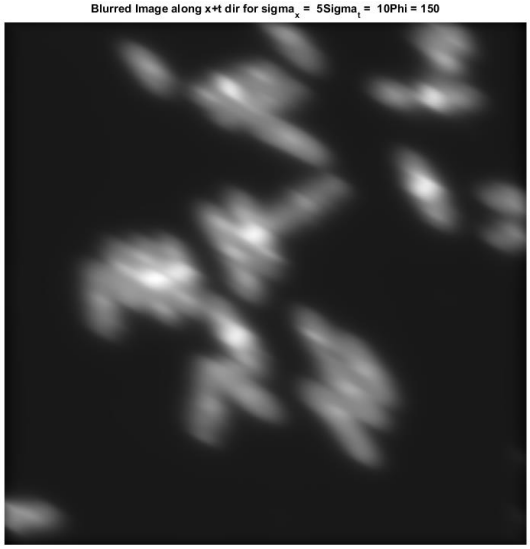


Figure 6: Phi = 150°



PART 2

Code Execution Instructions

Code execution instruction

2.2

Make Matlab's current directory to q2

Crop background

Type:

seq_cr2('DrosophilaVesicleTransport')

Select a region in the background that you want to analyze for noise signal

Wait for message, “cropping and saving ROI in the image sequence is completed.”

Files will be saved in new folder called DrosophilaVesicleTransport_cropped as background1.tif, background2.tif, and so on.

Noise analysis of cropped background

Type:

s_noise(‘DrosophilaVesicleTransport_cropped‘)

A figure that shows noise signal (averaged over space) across time will be displayed.

Type:

t_noise(‘DrosophilaVesicleTransport_cropped‘)

A figure that shows noise signal (averaged over time) across x- and y-direction will be displayed.

Crop background of the 2 images from JCB viewer that were used in proj1 assignment

<For neuron image>

Type:

sTiff_bgcr(‘proj1_neuron.tiff’)

Select a region in the background that you want to analyze for noise signal. The selected ROI will be cropped and saved for all images in the image file.

Cropped images will be saved as ‘background_1_proj1_neuron.tiff’ and ‘background_2_proj1_neuron.tiff’ in folder named ‘background_1_proj1_neuron’ and ‘background_2_proj1_neuron’, respectively

To select different ROI for each image in the image file, type:

multiTiff_bgcr(‘proj1_neuron.tiff’)

Select a region in the background that you want to analyze for noise signal for each image.

Cropped images will be saved as ‘ind_background_1_proj1_neuron.tiff’ and ‘ind_background_2_proj1_neuron.tiff’ in folder named ‘ind_background_1_proj1_neuron’ and ‘ind_background_2_proj1_neuron’, respectively.

Noise analysis of cropped background #2

<For neuron image>

1. If sTiff_bgcr function was used, type:

s_noise(‘background_1_proj1_neuron.tiff’)

A figure that shows noise signal (averaged over time) across x- and y-direction for the first neuron image will be displayed.

Type:

s_noise('background_2_proj1_neuron.tiff')

A figure that shows noise signal (averaged over time) across x- and y-direction for the second neuron image will be displayed.

2. If multiTiff function was used, type:

s_noise('ind_background_1_proj1_neuron.tiff')

A figure that shows noise signal (averaged over time) across x- and y-direction for the first neuron image will be displayed.

Type:

s_noise('ind_background_2_proj1_neuron.tiff')

A figure that shows noise signal (averaged over time) across x- and y-direction for the second neuron image will be displayed.

<For yeast image>

Type:

s_noise('background_yeast.tiff')

A figure that shows noise signal (averaged over time) across x- and y-direction for the yeast image will be displayed.

<For neuron image>

Type:

s_noise('background_2_proj1_neuron.tiff')

For

2.3

Type:

il_u('20x_02.tif')

Select a region to check for illumination uniformity.

2.4

2.4.1

Type:

manula_cal('20x_02.tif')

Enter microscope pixel size and magnification manually

Select a region with the most number of ruler marks in the y-direction.

2.4.2

Type:

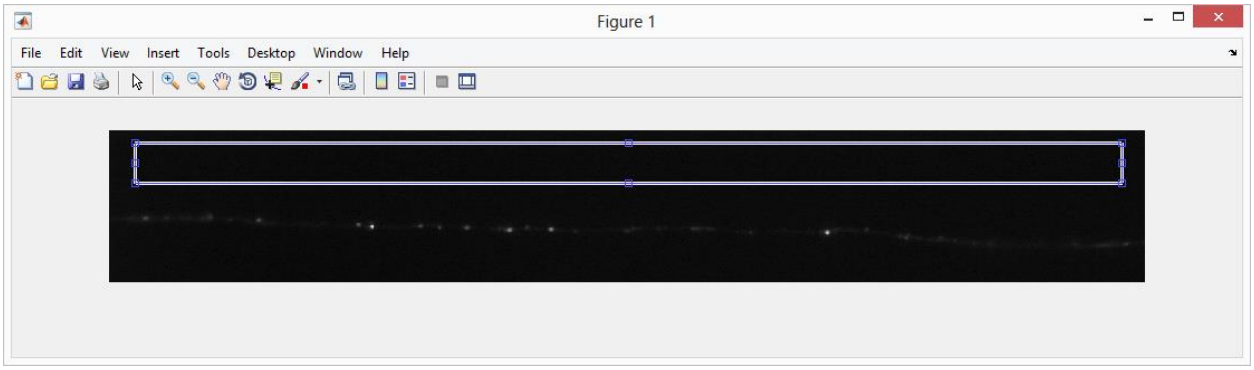
auto_cal('20x_02.tif')

Enter microscope pixel size.

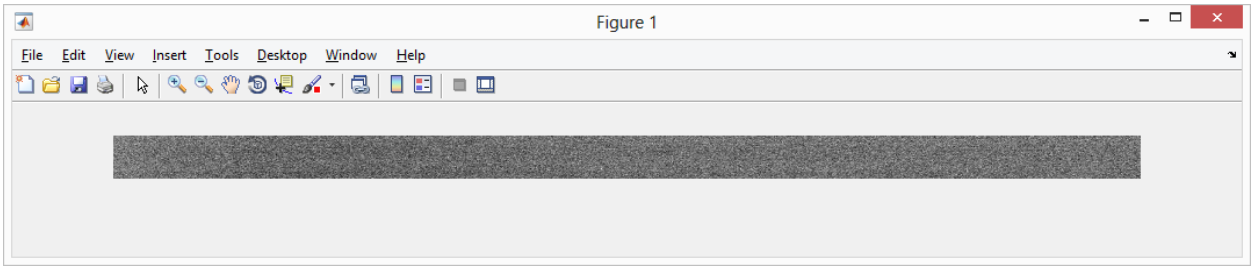
Make sure that the image file has the magnification number and X in the front.

Results

2.2

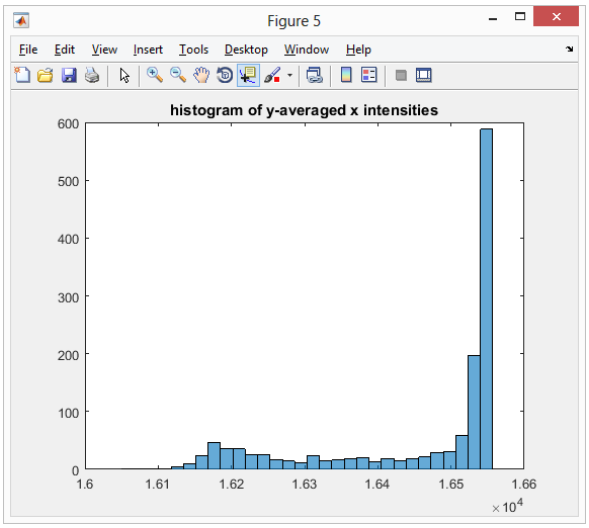


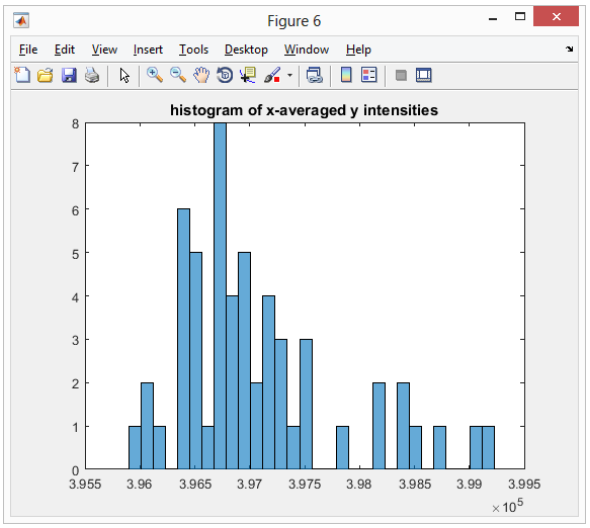
Cropped region in the background for noise analysis



Contrast adjusted

2.2.1





Histogram of Time-averaged spatial noise signals.

y-averaged x intnesity data -> highly biased noise distribution

x-averaged y intensity data -> possibly normal noise distribution

- Checking for normal distribution:

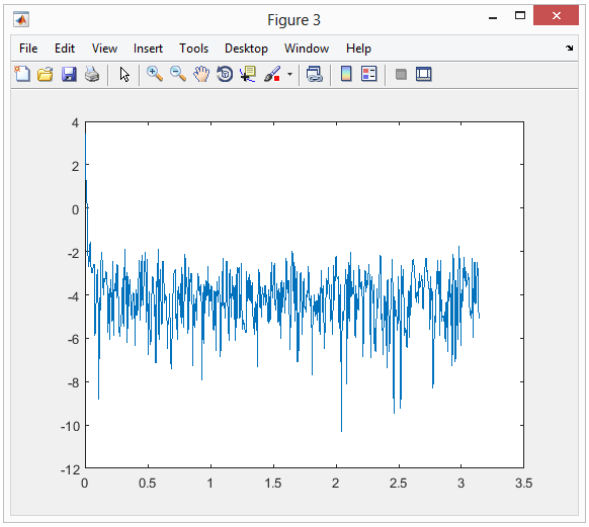
Use Normality test: Jarque-Bera test. Tests whether data came from a normal distribution with unknown mean and variance. The results are as followed:

The probability that the y-averaged x intensity data came from a normal distribution is less than 5% according to Jarque-Bera test.

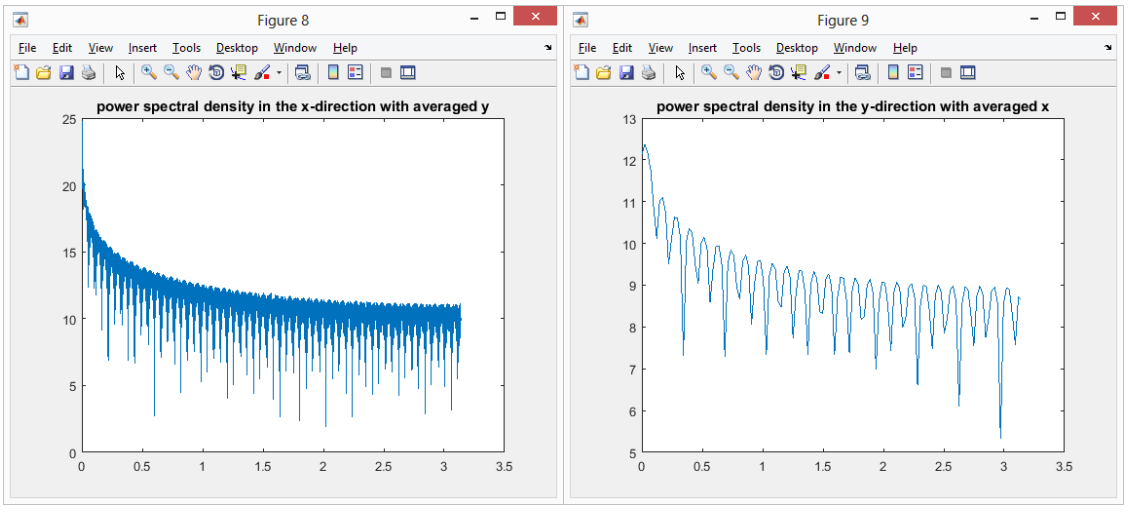
The probability that the x-averaged y intensity data came from a normal distribution is less than 5% according to Jarque-Bera test.

Signal unlikely a normal distribution.

- Checking if the noise is white:



If the noise is white, we should get the above power spectral density which was generated by 1000 random numbers (from rand(1,1000)), with spectral density is approximately constant across different frequencies.



Not constant density across -> not white noise.

2.2.2

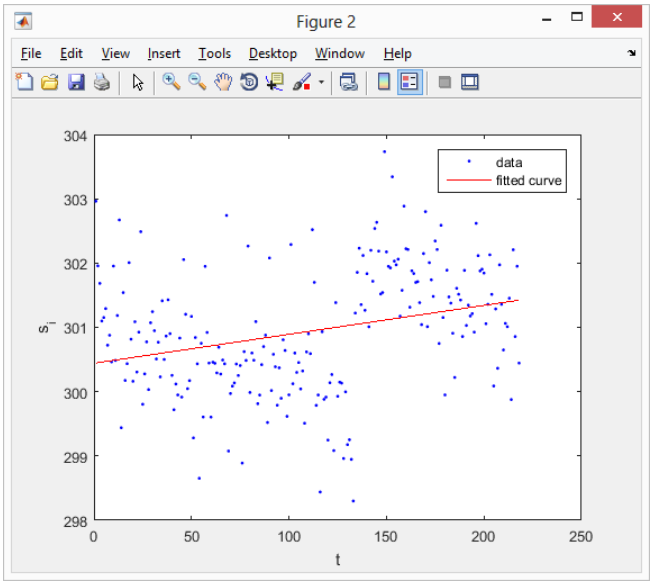


Figure 3: Noise signal (averaged over space) across time

Change in spatially averaged noise signal over time. The noise seems to show a sinusoidal behavior over time.

2.2.3

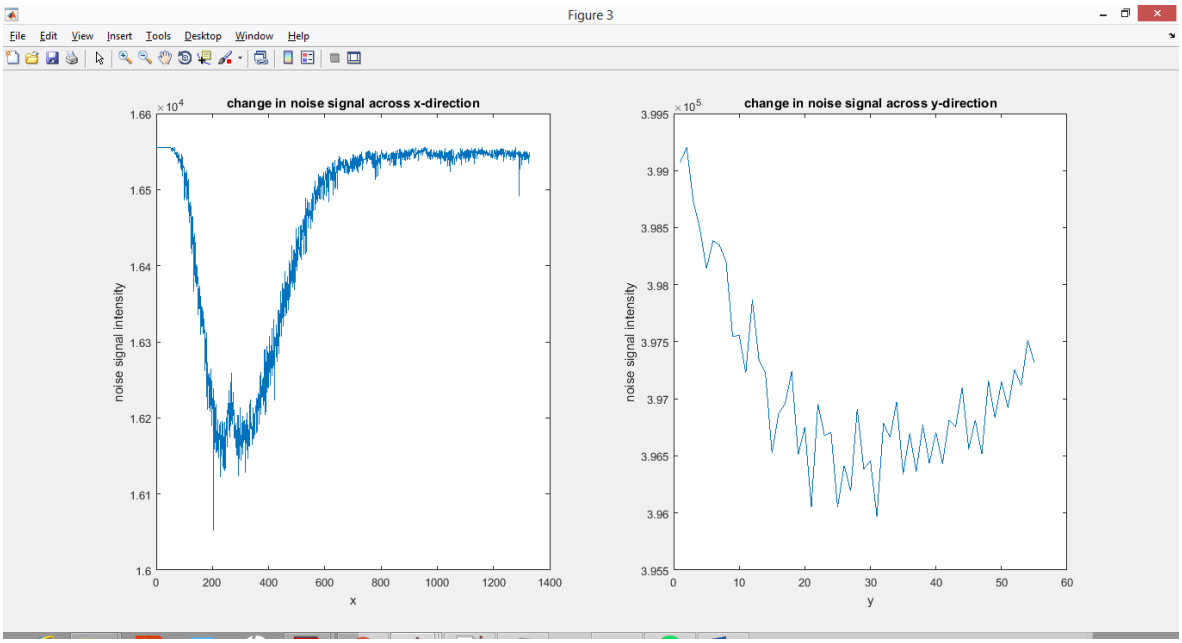


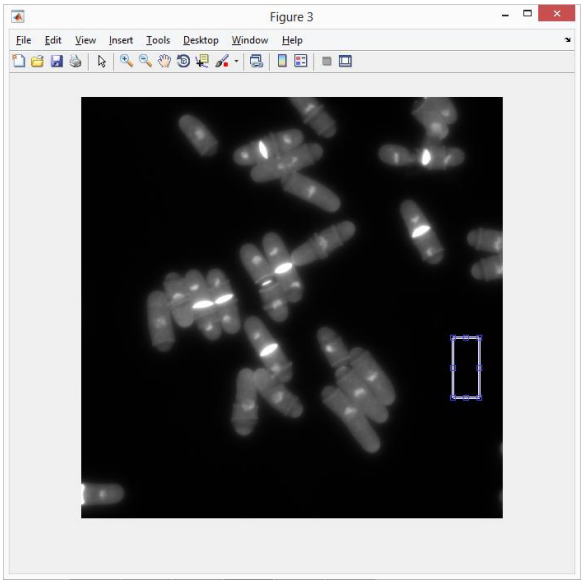
Figure 4: Noise signal (averaged over time) across x- or y-direction.

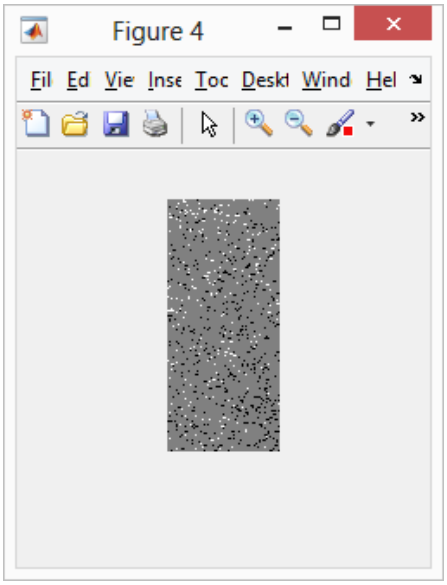
Noise signal across x-direction decreases then increases.

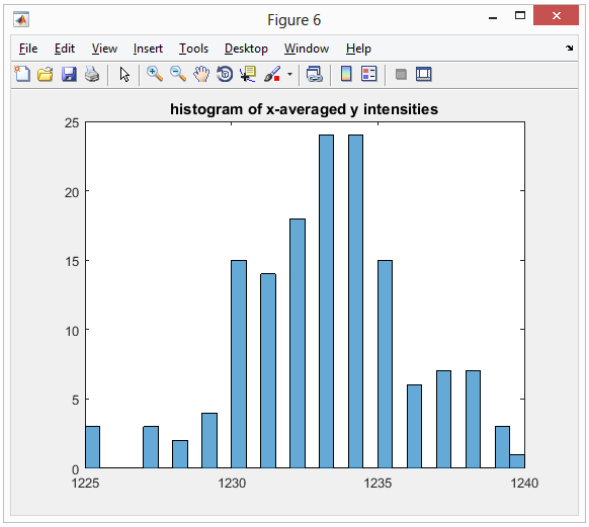
Noise signal across y-direction decreases then increases.

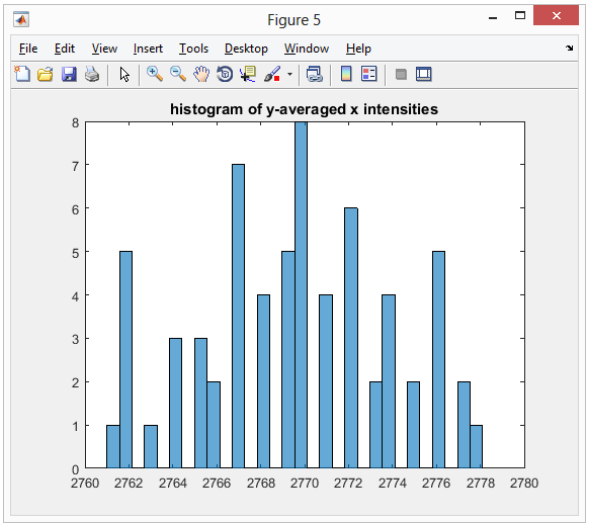
Repeat with two images from JCB viewer: Yeast and Neuron

2.2.1-2 (Yeast)









Histogram of Time-averaged spatial noise signals.

Both y-averaged x intnesity data and x-averaged y intensity data -> possibly normal noise distribution

Checking if the noise signal is a normal distribution.

Use Jarque-Bera test that tests if the signal comes from a normal distribution with an unknown mean and variance.

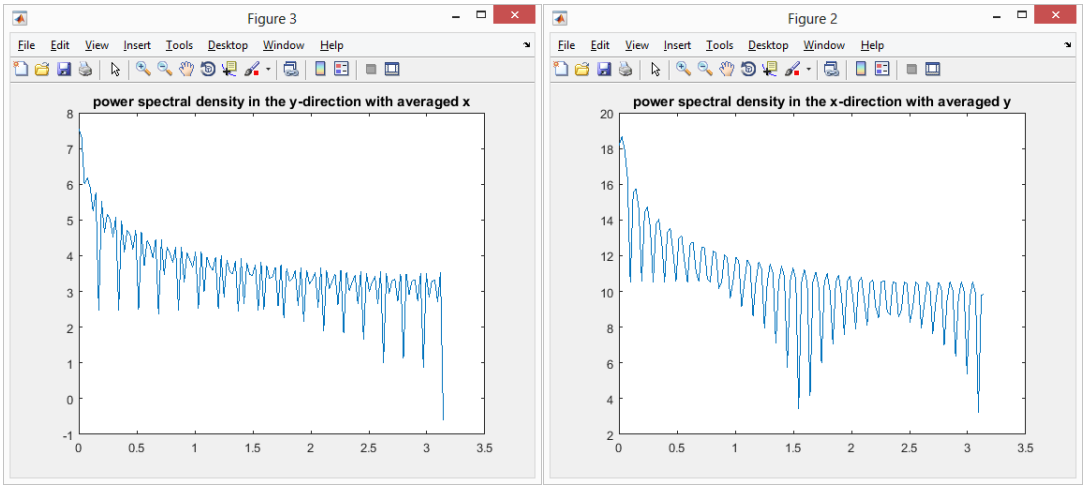
This test, which is integrated in the code, gives the following result:

The probability that the y-averaged x intensity data came from a normal distribution is more than 5% according to Jarque-Bera test.

The probability that the x-averaged y intensity data came from a normal distribution is more than 5% according to Jarque-Bera test.

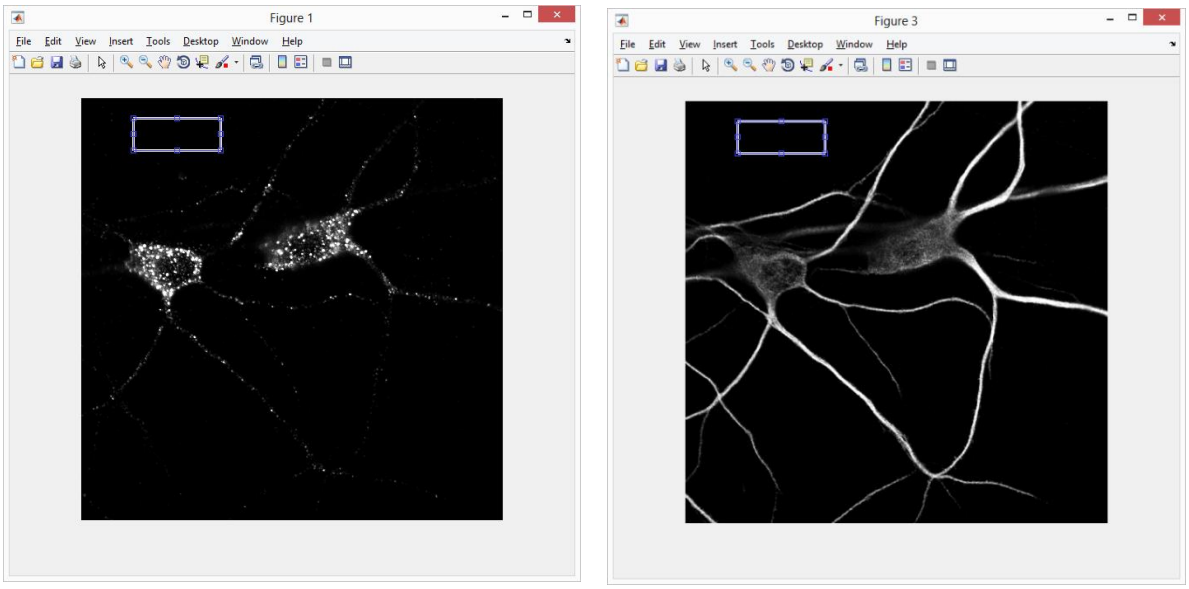
Based on this, we conclude that the noise signal may have a normal distribution.

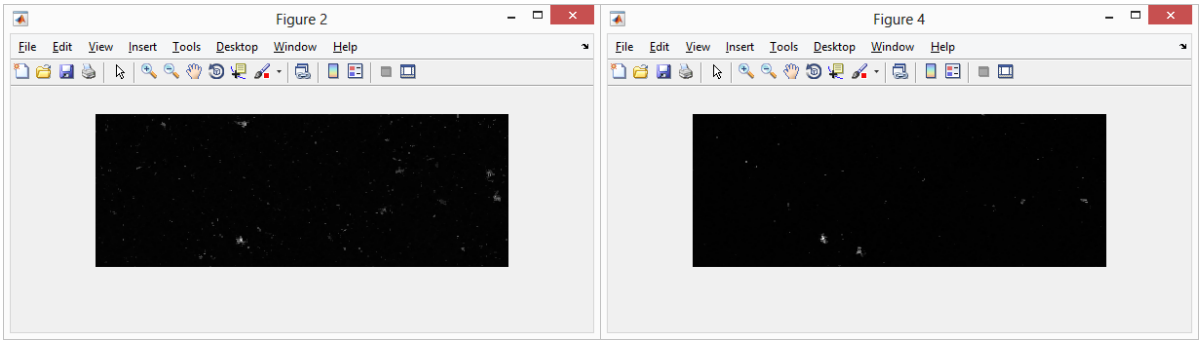
Checking for White noise



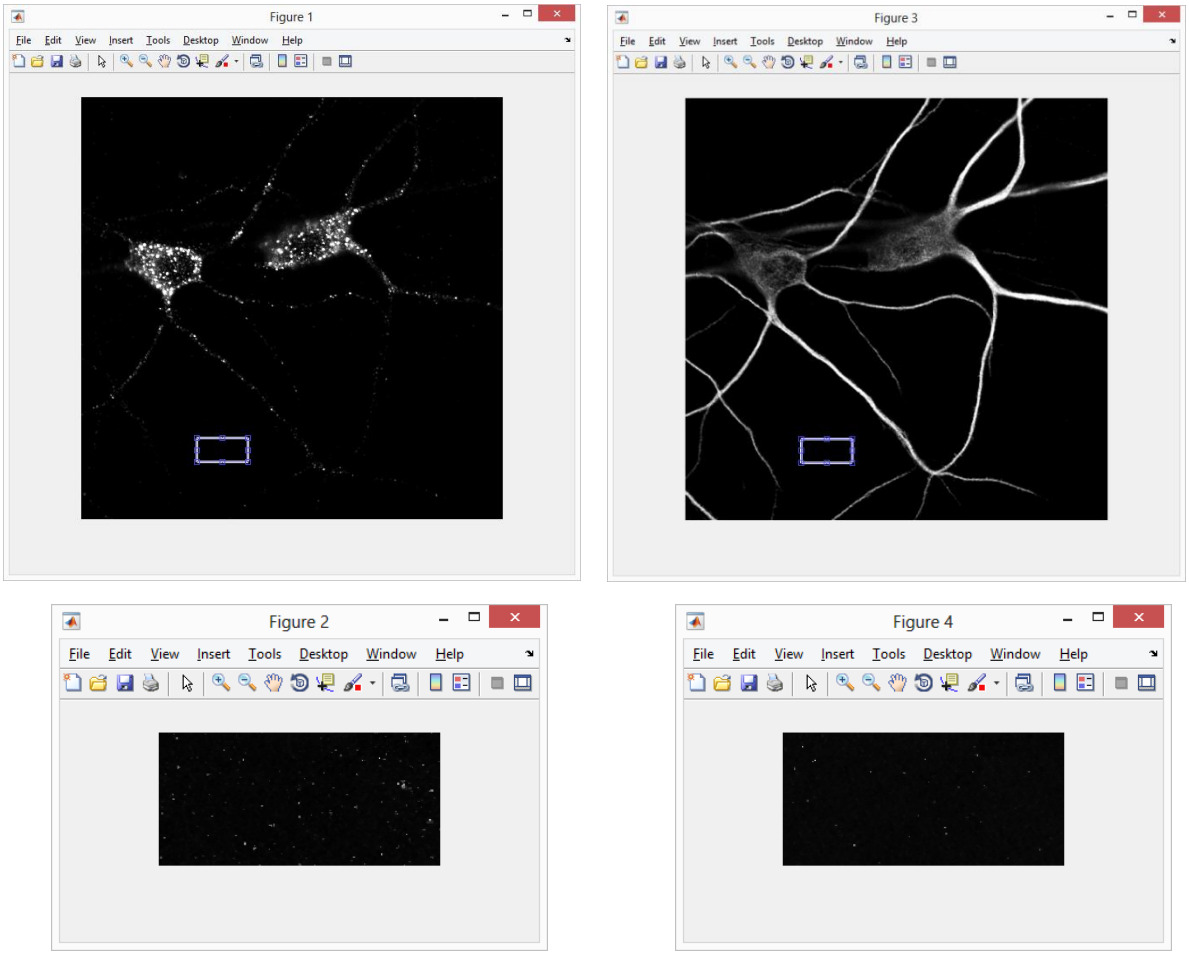
Definitely not constant density across frequency -> non-white noise.

2.2.1-3 (Neuron)



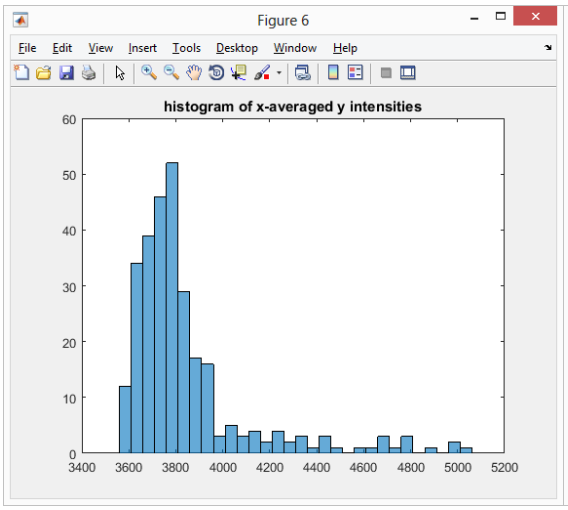


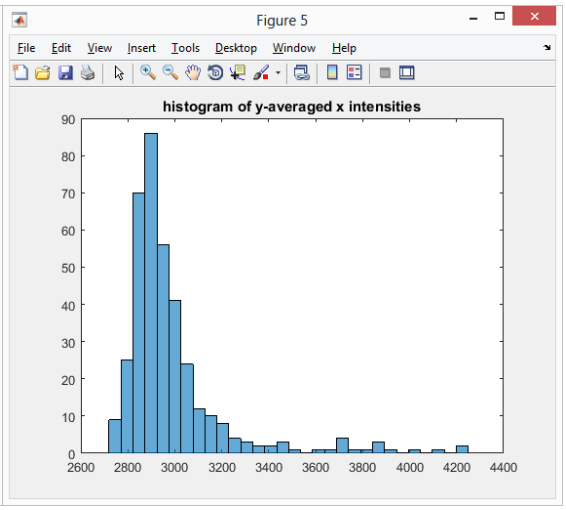
Seems capturing small particles. Analyze a different region in the background.



Capturing a different region in the background. Same result when checking additional regions.

Go ahead and analyze this background. The resulting noise distribution is:





Checking if the noise signal is a normal distribution.

Use Jarque-Bera test that tests if the signal comes from a normal distribution with an unknown mean and variance.

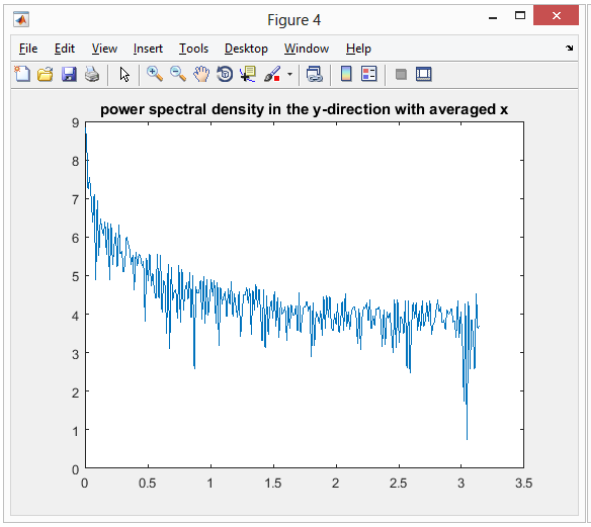
This test, which is integrated in the code, gives the following result:

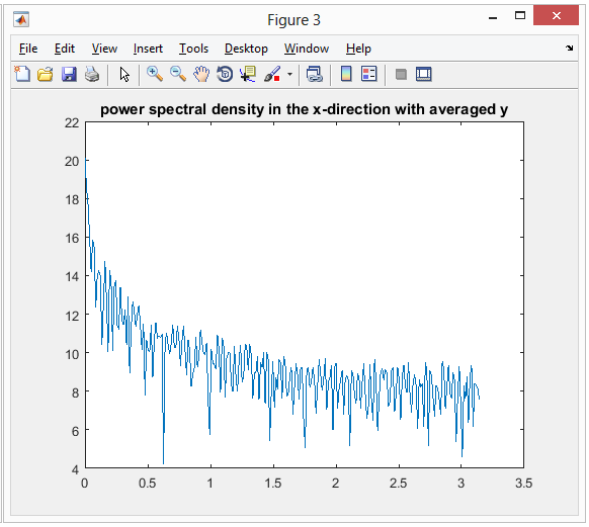
The probability that the y-averaged x intensity data came from a normal distribution is less than 5% according to Jarque-Bera test.

The probability that the x-averaged y intensity data came from a normal distribution is less than 5% according to Jarque-Bera test.

Based on this, we conclude that the noise signal does not have a normal distribution.

Checking for white noise:





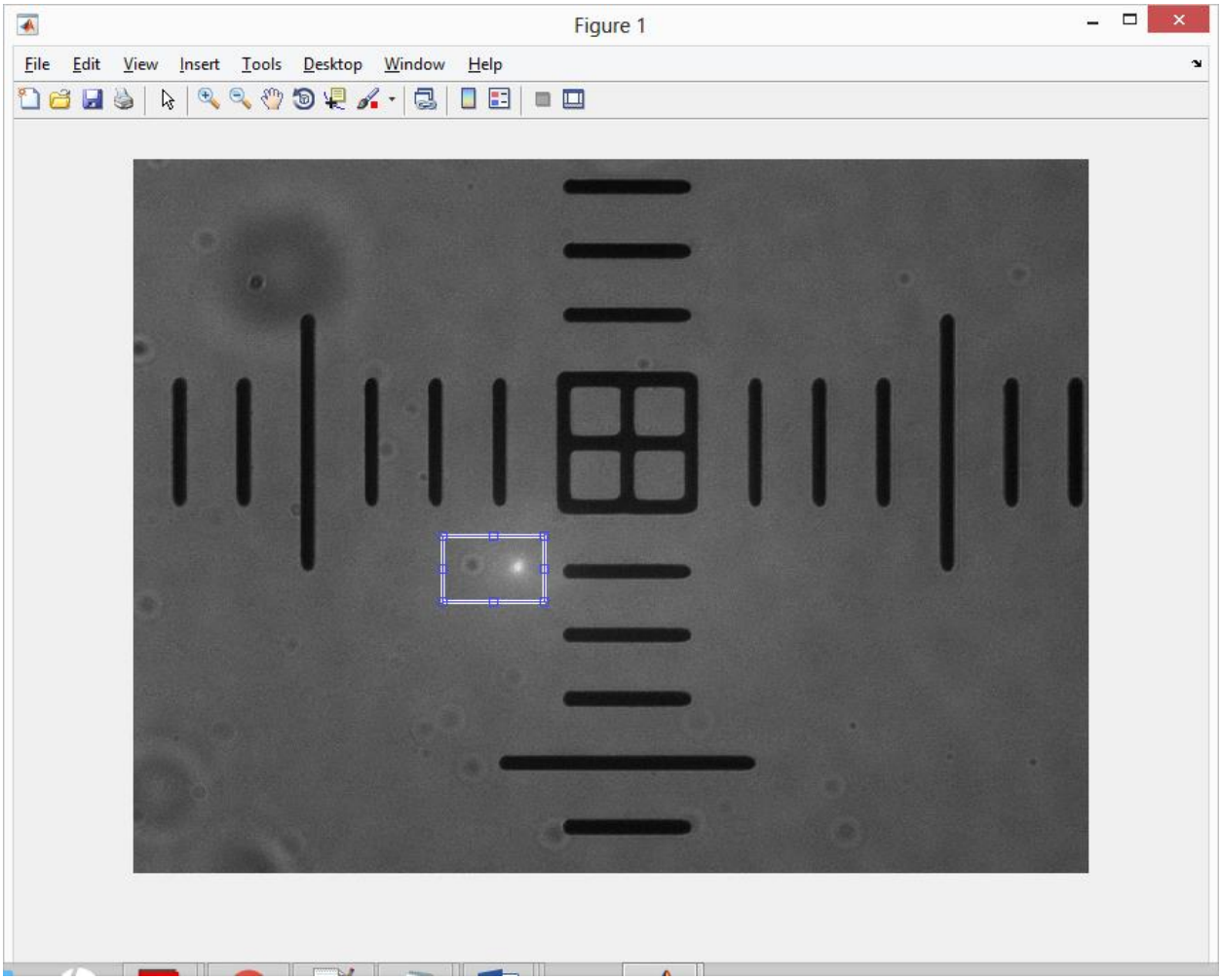
Decreasing spectral density over frequency -> not white noise

2.3

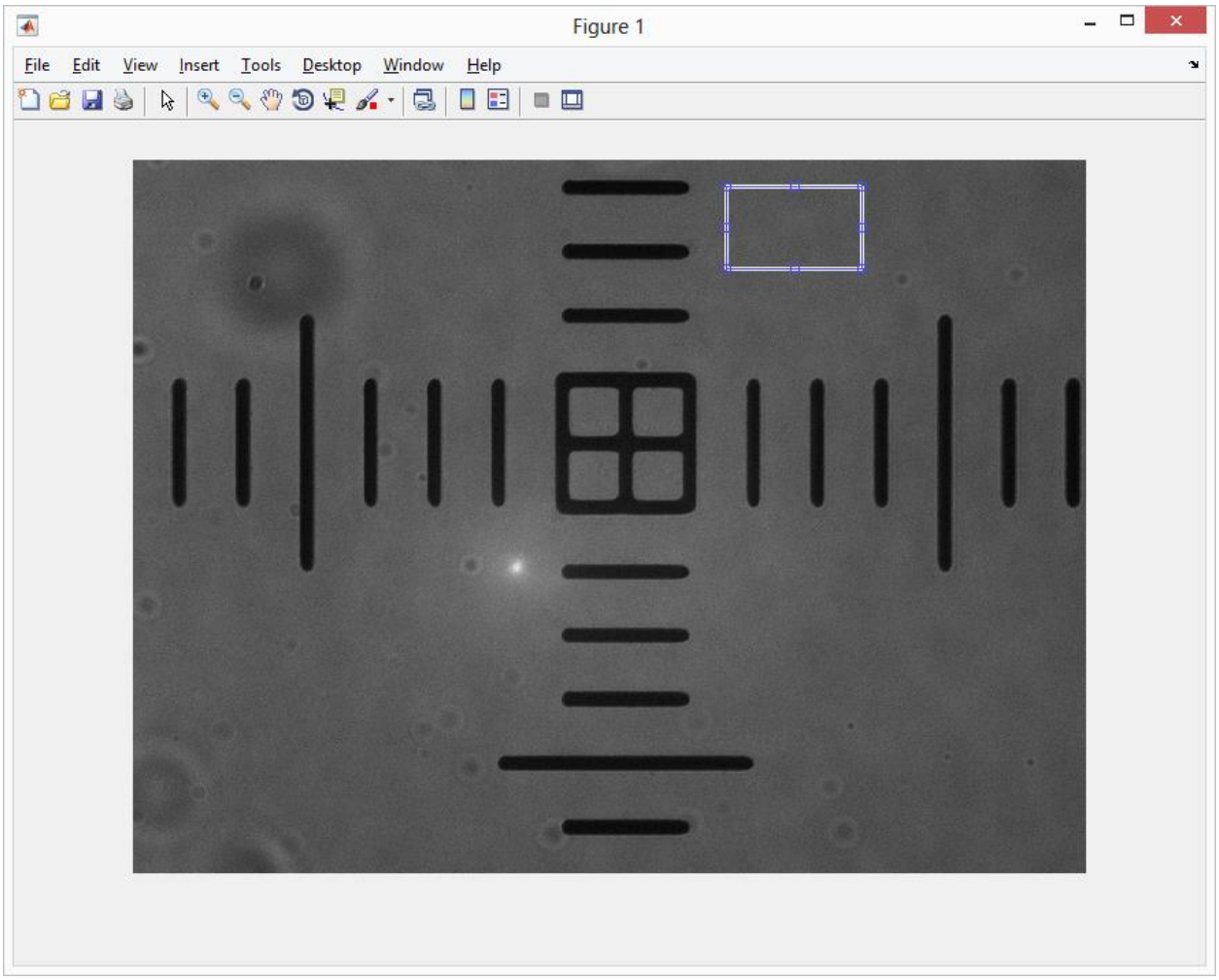
Own Quantitative descriptor of illumination uniformity: standard deviation/mean.

This can capture significant gradual changes relative to mean in illumination intensity. This also captures noise due to unexpected features that will produce relatively higher intensity value difference and the lack of illumination uniformity thereof.

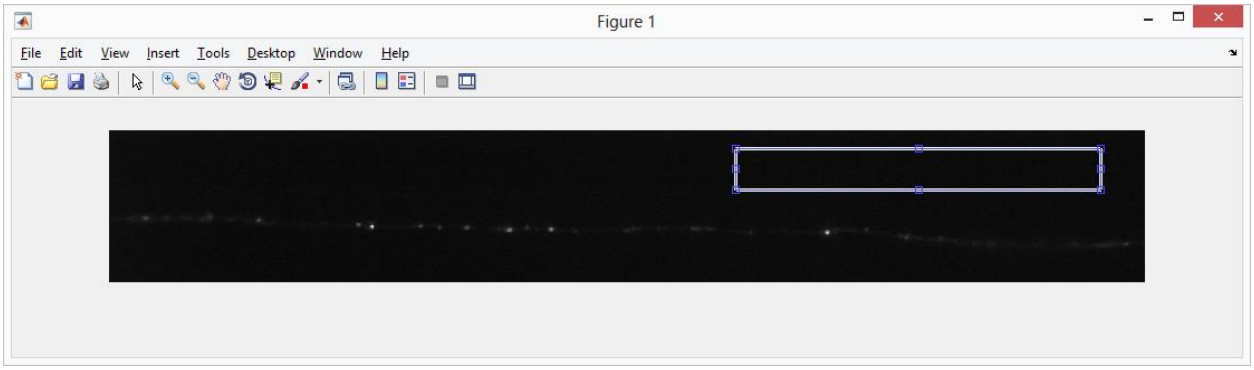
The closer the value is to 0, the more uniform the illumination is.



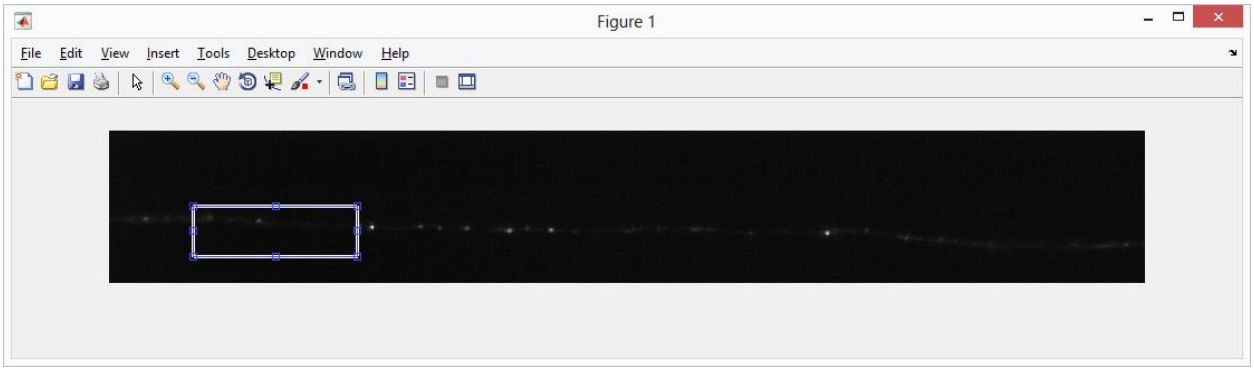
Intensities from region analyzed for mean and standard deviation gives the quantitative descriptor value (Standard deviation over mean) 1.435926e-01.



The standard deviation/mean is 1.262467e-01, which is lower than when features were selected in the previous figure. The value is still relatively high, which is apparent by textile appearance of the background.



The standard deviation is 8.100343e-02, indicating is a more uniform illumination compared to the pixel calibration image.

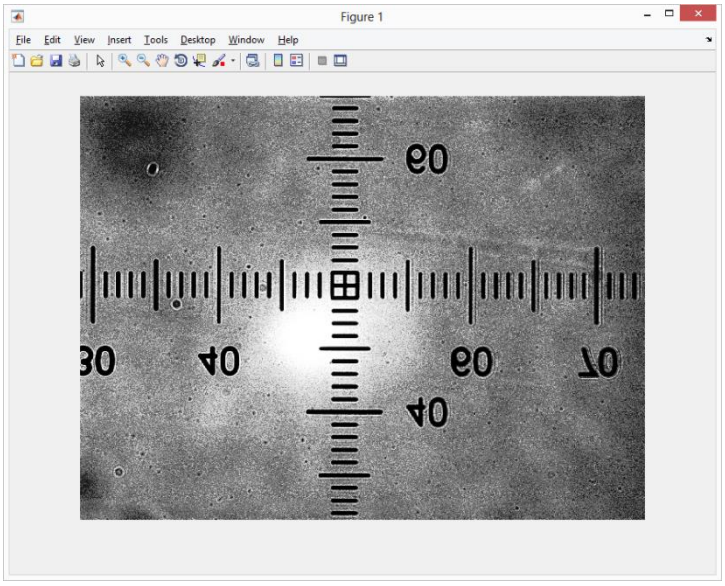


The standard deviation/mean is 1.826078e-01 when the features area included. This gives additional confidence of our illumination uniformity quantitative descriptor.

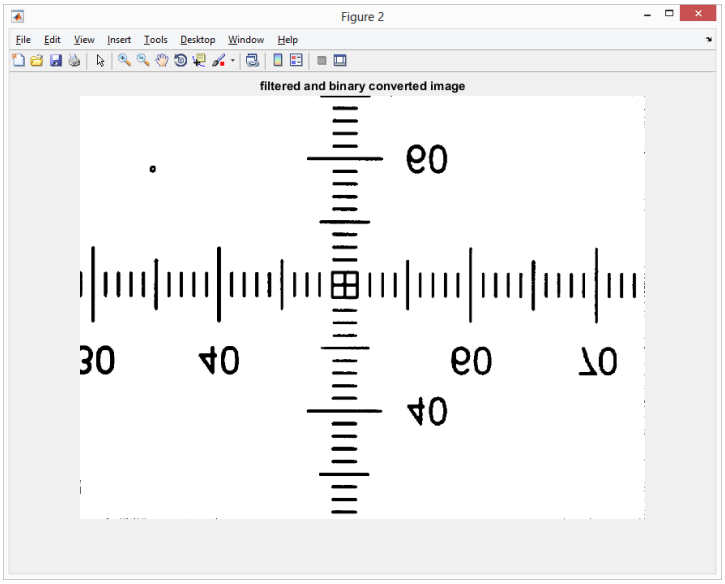
2.4

2.4.1

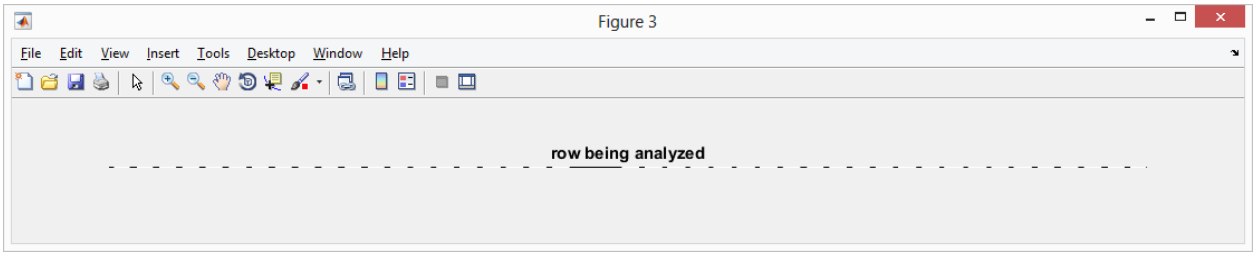
The image is first auto-contrasted, Gaussian-filtered, then automatically rotated such that the ruler marks are perpendicular to the x-direction. Then, the user selects a region that contains the most number of ruler marks going in the y-direction. That region is then converted to binary, and the distances between each edge is calculated and averaged to find the pixel values between each mark that represent a true 10um distance.

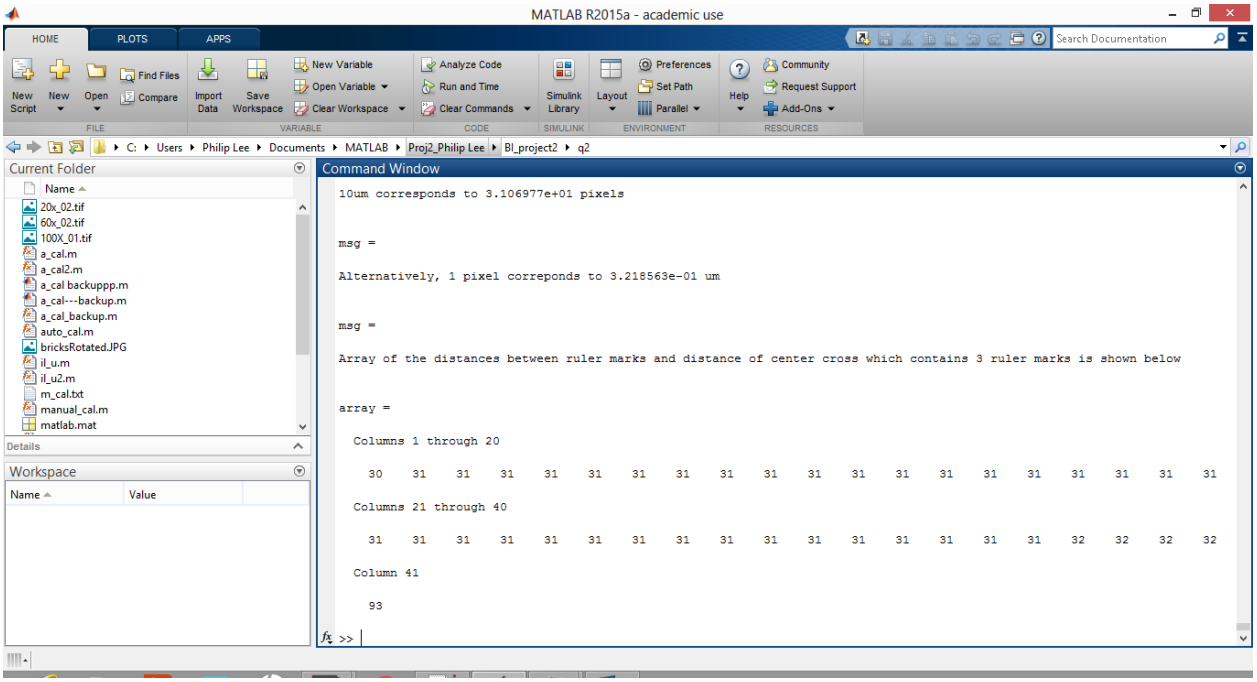


Auto-contrasted image



Gaussian filtered and converted to binary





Resulting messages. Note distances between the ruler marks come out quite uniformly (~31 pix).

2.4.2

With the automated calibration, the same procedure is executed, except that the region with the most number of ruler marks going the y-direction is selected automatically.

Part 3

3.1 Implement a MATLAB function to plot the Airy disk.

Code execution Instruction:

1. Run MATLAB script titled (qsn3part1.m) to generate all the Airy disk plots on a single figure handle (in different colors).

   Note: The function "PlotAirydisk(lambda, NA)" computes the spatial intensity values after passing through a circular aperture, with the wavelength of light (lambda) and Numerical aperture(NA) as the inputs.

Mathematical background:

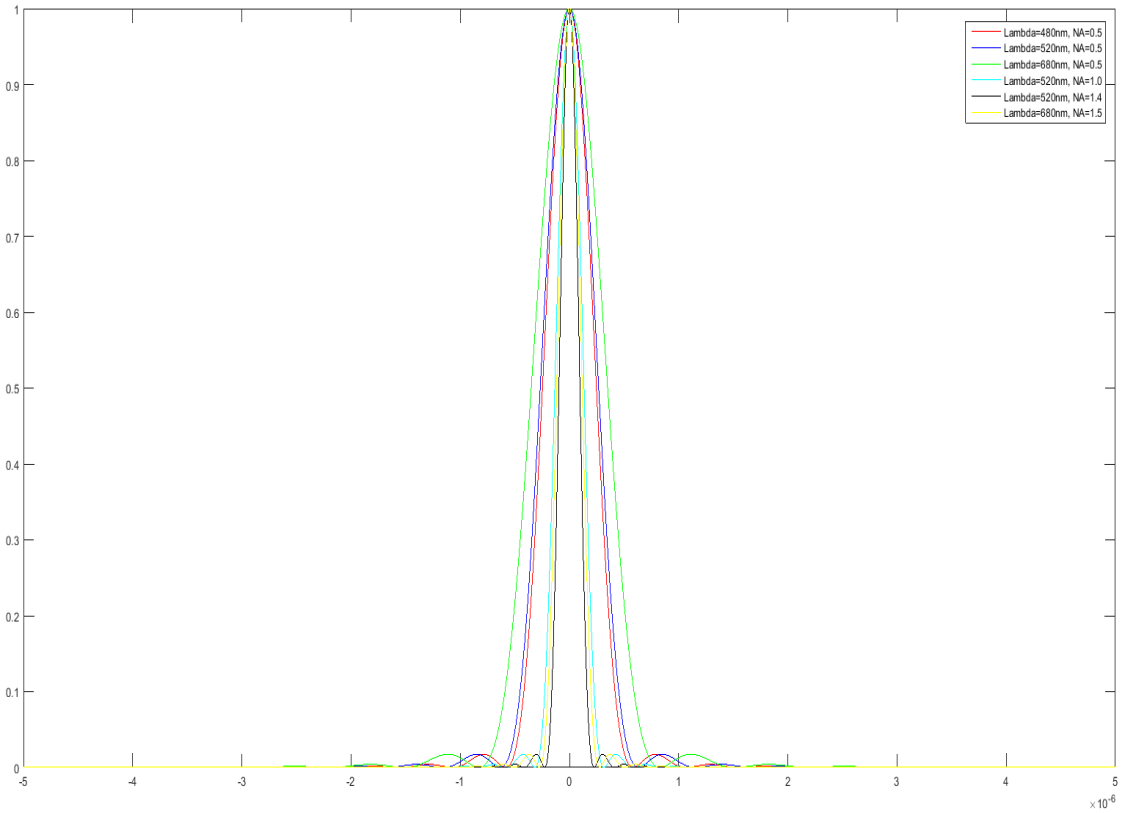
The intensity of an airy disk is calculated by solving the first order Bessel's function. [3]

$$I = \left(\frac{2J_1(x)}{x}\right)^2$$

where, $x = \frac{2*\pi}{\lambda} * a * \sin(\theta)$;

$\lambda\ is\ wavelength$;

$a\ is\ aperture,$



θ is the angle between the distance between the screen to circular aperture and the point on the screen

Results: The plot of airy disks with different wavelength and Numerical aperture combinations are as follows:

From the plot above, it's clearly evident that the incident light wavelength and numerical aperture of the microscope, influences the radius of the airy disk.

On comparing the first 3 cases (as depicted in the legend), we can see that radius of the airy disk increases, with an increase in wavelength of light (while keeping the Numerical aperture constant) [from red to blue to green]. Along with that, we can see that radius of the airy disk increases linearly with an increase in wavelength (can be proved quantitatively, as well). This shows that the radius of the airy disk is directly proportional to the wavelength of light.

On comparing Case 2, Case 4 and Case 5, where the incident wavelength is 520nm, while the Numerical aperture increases (from 0.5 to 1.0 to 1.4), there is a notable decrease in the radius of the airy disk. This implies that the radius of the airy disk is inversely proportional to the Numerical aperture used.

From the above figure and analysis, we can say, that $\boldsymbol{r_{airydisk} \propto \lambda/NA}$

3.2 Fitting of the Airy disk using a Gaussian kernel

Code execution Instruction:

1. Run MATLAB script titled (qsn3part2.m) to evaluate the radius of the airy disk as well as the standard deviation of the best fitted Gaussian kernel.

   Note: The "findradius" function calculates the radius of the Airy disk, and the "gaussianFit" function determines the best fitted Gaussian kernel (for each of the 6 cases).

For each of the Airy disks plotted, a Gaussian kernel has been fitted.

From the earlier plot, it is clear that the mean of the Gaussian must be centered at distance of screen (y) = 0. The sigma is varied to obtain the best fit, by minimizing the mean-square error between the fitted Gaussian kernel and the Airy disk intensity profile.

For the 6 cases, the best Gaussian kernel fit (sigma) and the radius of the airy disk are tabulated below:

Sno	**Input parameters**	**Radius of Airy disk**	**Best Gaussian Kernel Fit (sigma)**
1	$\lambda = 480nm, NA = 0.5$	$0.58\ \mu m$	$0.2045\ \mu m$
2	$\lambda = 520nm, NA = 0.5$	$0.62\ \mu m$	$0.2215\ \mu m$
3	$\lambda = 680nm, NA = 0.5$	$0.82\ \mu m$	$0.2895\ \mu m$
4	$\lambda = 520nm, NA = 1.0$	$0.31\ \mu m$	$0.1110\ \mu m$
5	$\lambda = 520nm, NA = 1.4$	$0.22\ \mu m$	$0.0795\ \mu m$
6	$\lambda = 680nm, NA = 1.5$	$0.27\ \mu m$	$0.0965\ \mu m$

On comparing the radius of the Airy disk to the Sigma of the Gaussian Kernel, we can see that that the best fit Gaussian (Standard deviation (sigma)) is one third the radius of the airy disk, while the mean is centered at 0.

The fitting is optimal, and this can be verified by calculating the value of the Gaussian function at 3*sigma, which equals 0.011 (Peak amplitude) (ie very close to zero, first minima of the airy disk).

This standard deviation – radius relation dictates the optimal sampling frequency. Here, in order to be able to get reliable sub-pixel detection, we would require 3 pixels to cover the radius of one airy disk.

Bibliography

[1] "http://www.mathworks.com/matlabcentral/answers/81689-how-to-implement-convolution-instead-of-the-built-in-imfilter," [Online].

[2] G. Yang, "BioimageInformatics_Spring2016_Lecture_07.pdf".

[3] E. Hecht, Optics.