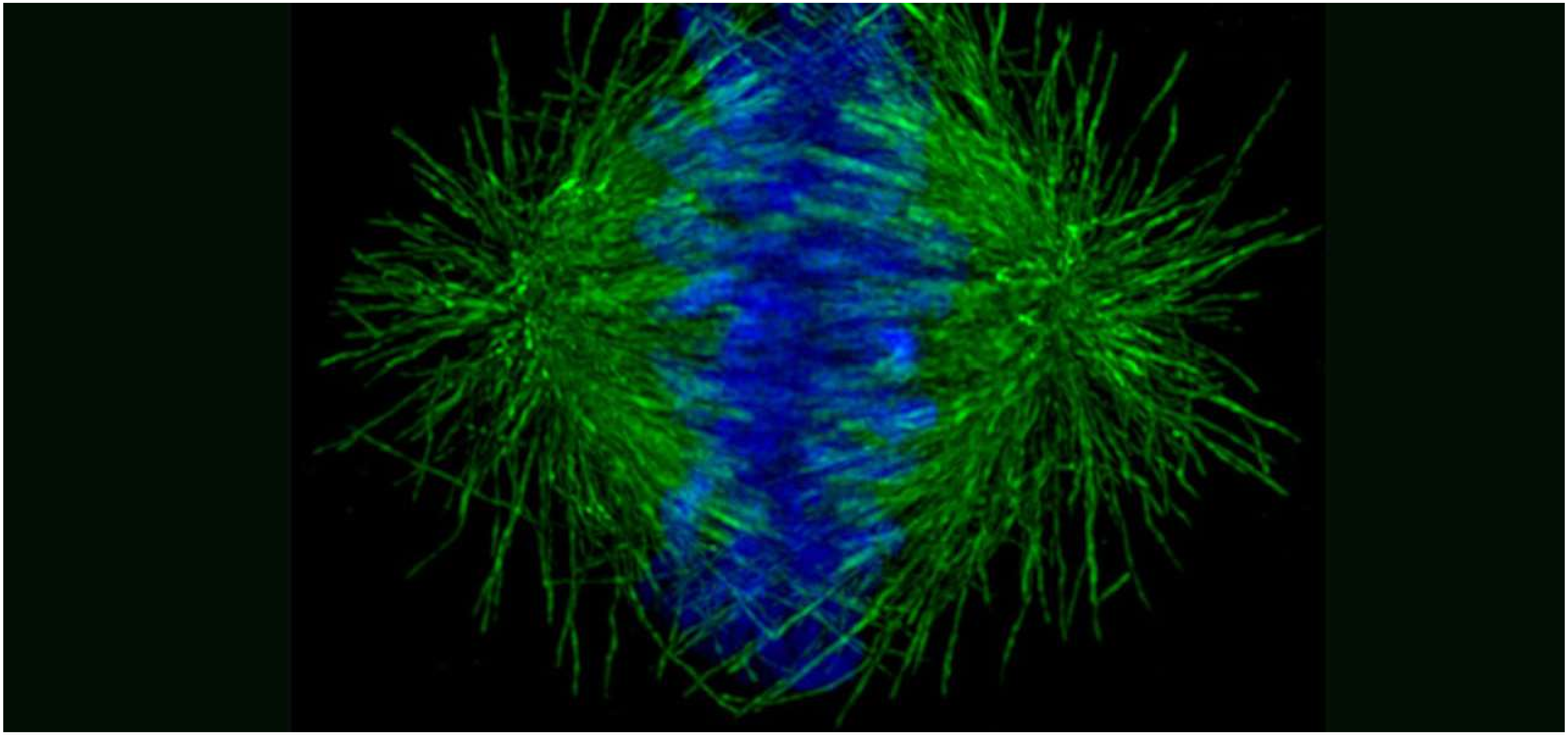


Localizing Single Particles and STORM

MCDB/BCHM 4312/5312

Please download

`hw9_STORMdata.tif`

and

`hw9_example.m`

Learning Goals

- Finding approximate locations of single particles
  - Difference of Gaussians filter
  - Extended maxima filter

- Sub-pixel localization by 2D curve-fitting (surface fitting)

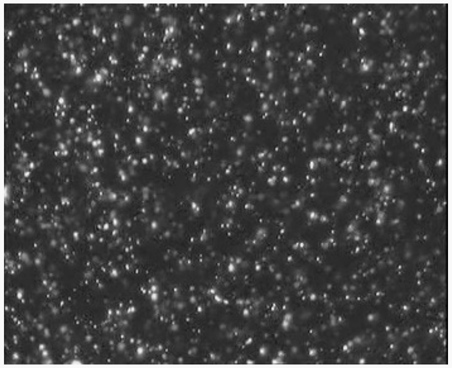
- Single molecule localization microscopy - STORM

Quick recap on STORM

- STORM = STochastic Optical Reconstruction Microscopy

- An imaging technique with a resolution that goes below the standard diffraction limit of light (i.e. below the width of the PSF)

Imaging small objects

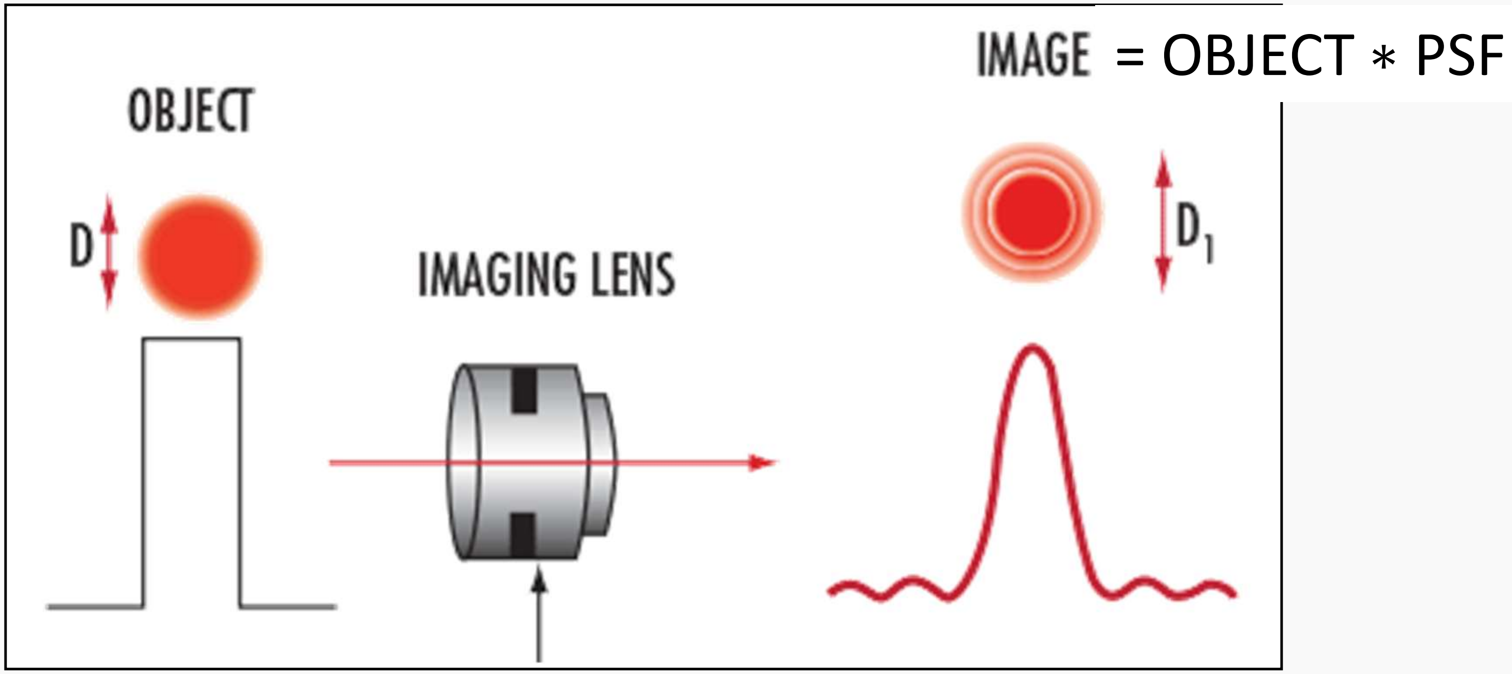


What is the imaged size of a 50 nm object, imaged using an objective lens with a point spread function width of 100 nm?

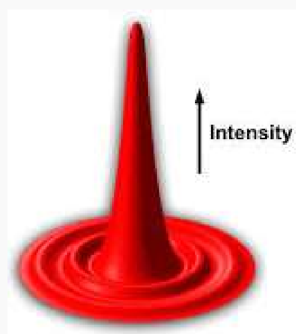
A. 25 nm
B. 50 nm
C. 100 nm
D. 200 nm

http://www.microscopy-uk.org.uk

Image is the convolution of the object intensity with the objective point spread function (PSF)

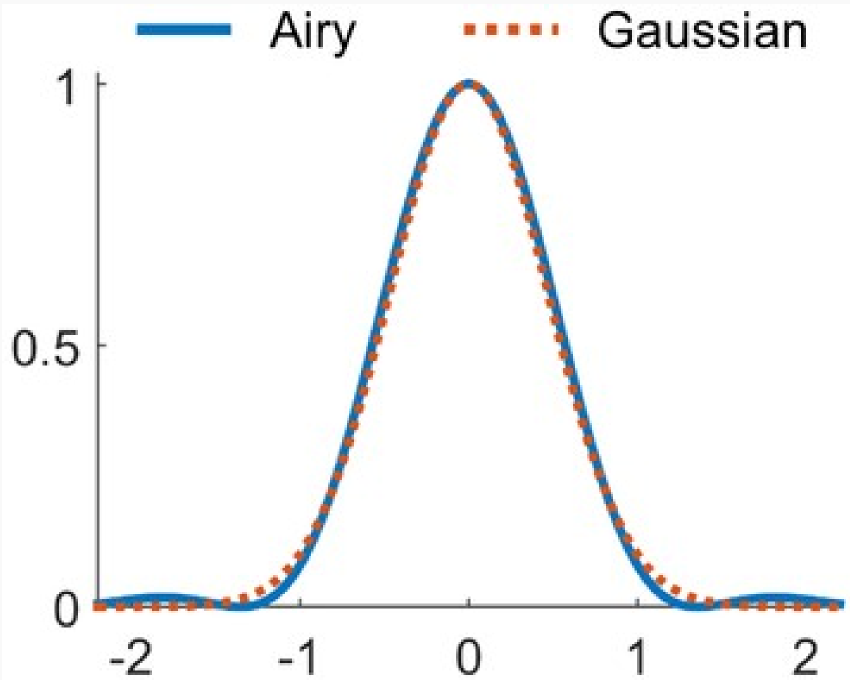


PSF
(Point Spread Function)
or Airy disk



https://www.edmundoptics.com/resources/application-notes/imaging/limitations-on-resolution-and-contrast-the-airy-disk/

Gaussian function approximates an Airy disk



Why use a Gaussian?

- The Airy disk equation is computationally difficult to fit because it has Bessel functions which are complex
- The Gaussian equation is a good estimate of the central peak of the Airy disk (within a few %)

https://www.researchgate.net/figure/Point-spread-function-and-resolution-a-Airy-pattern-representing-the-2D-transverse_fig9_323835351

Quick recap on STORM

- STORM = STochastic Optical Reconstruction Microscopy

- An imaging technique with a resolution that goes below the standard diffraction limit of light (i.e. below the width of the PSF)

- Basic principle is **single molecule localization** – i.e. if there was only a single fluorescent particle, you could fit a Gaussian to get its location at a resolution below a single pixel

Optical imaging

Computational **reconstruction**

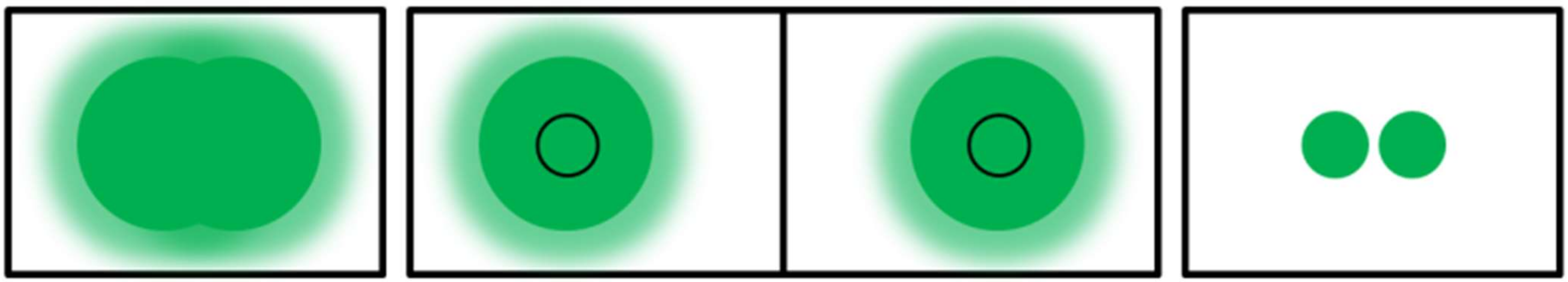


Image of two spots in a normal light microscope

At times when only the left or right spot is “on", we find the center

We plot just the centers to make a STORM image

http://sustainable-nano.com/2013/11/07/zooming-in-on-the-nano-world-limitations-and-breakthroughs-in-light-microscopy/

1D Gaussian equation

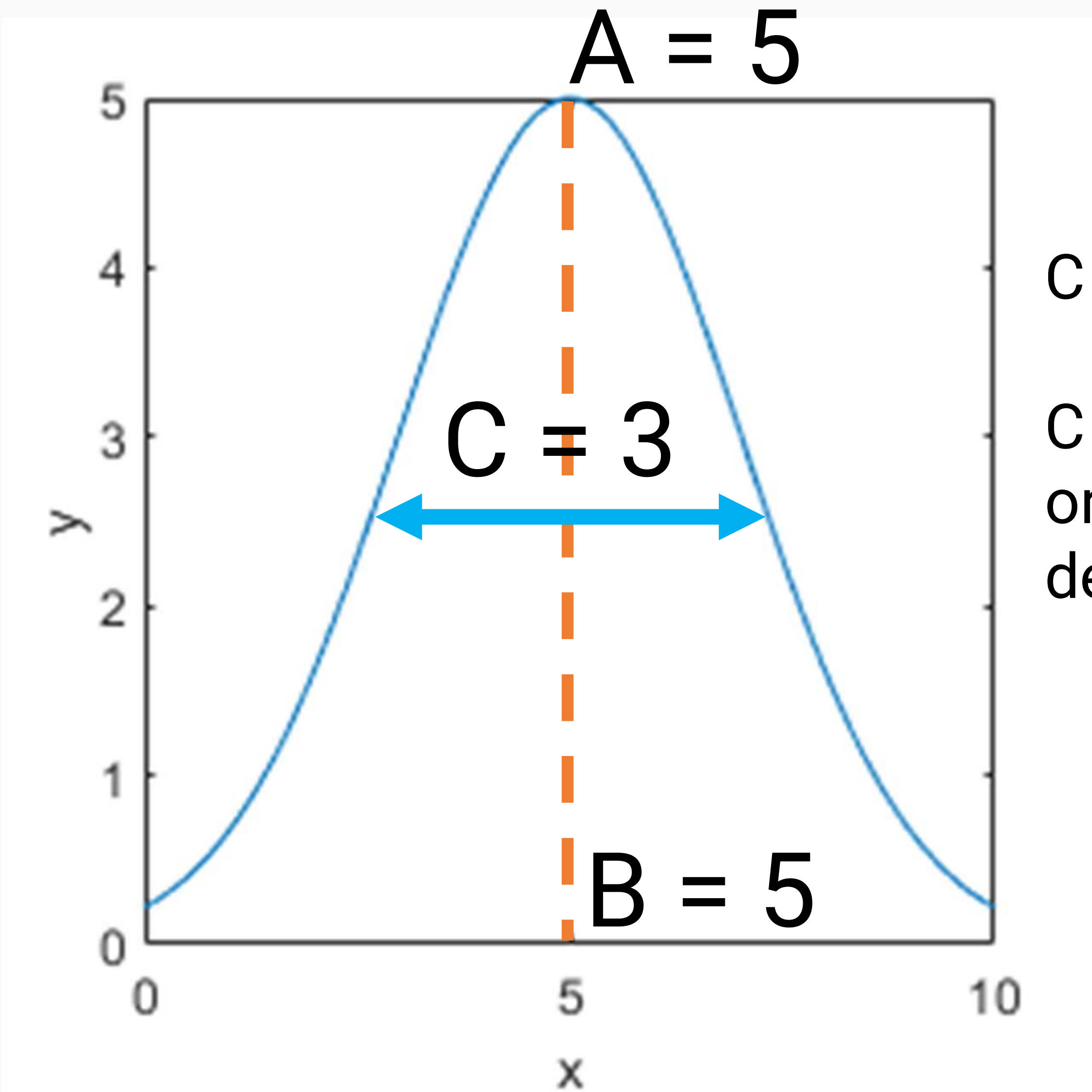
$$y = A \exp\left(-\frac{(x-B)^2}{2C^2}\right)$$

A = amplitude (maximum value of y)

B = x-offset (position along the x-axis the maximum y occurs)

C = width (related to the FWHM)

$$y = A \exp\left(-\frac{(x-B)^2}{2C^2}\right)$$



C related to the FWHM

C is also related to σ or the standard deviation

2D Gaussian equation

$$z = A \exp\left(-\frac{(x-B)^2+(y-C)^2}{2D^2}\right)$$

A = amplitude

B = x-offset

C = y-offset

D = width (related to the FWHM)

Drawing a 2D Gaussian (or any 2D surface)

1. Define x- and y- grid

For this example, generate a vector from -10 to 10, with 1000 steps

```
xx = linspace(-10, 10, 1000);
yy = linspace(-10, 10, 1000);
```

Compute the grid matrix (makes the next step more efficient)

```
[xx, yy] = meshgrid(xx, yy);
```

meshgrid calculates matrices of coordinates

```
x = 1:3;
y = 1:5;
[X,Y] = meshgrid(x,y)
```

```
X = 5×3                    Y = 5×3

     1     2     3              1     1     1
     1     2     3              2     2     2
     1     2     3              3     3     3
     1     2     3              4     4     4
     1     2     3              5     5     5
```

We are working in pixels here, but x and y could be converted to real units

Drawing a 2D Gaussian

1. Define x- and y- grid

2. Generate the z-data (i.e. evaluate the 2D Gaussian function as a matrix operation)

$$z = A \exp\left(-\frac{(x-B)^2+(y-C)^2}{2D^2}\right)$$

A = 5; B = C = 1; D = 3

```
zz = 5 * exp (-((xx - 1).^2 + ...
         (yy - 1).^2) / (2 * 3^2));
```

Drawing a 2D Gaussian

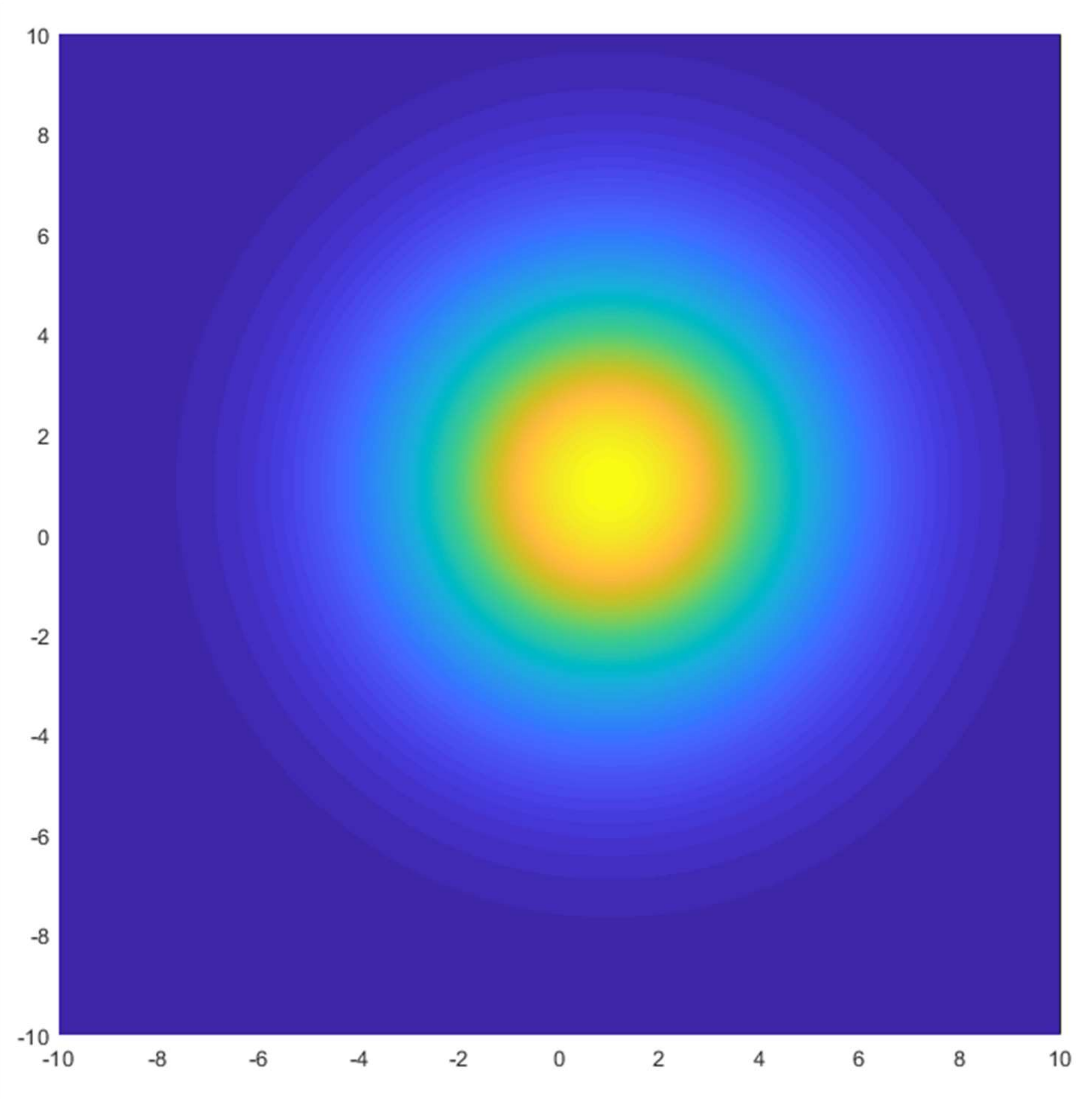
1. Define the x- and y- grid

2. Generate the z-data (i.e. evaluate the 2D function as a matrix operation)

3. Display the image

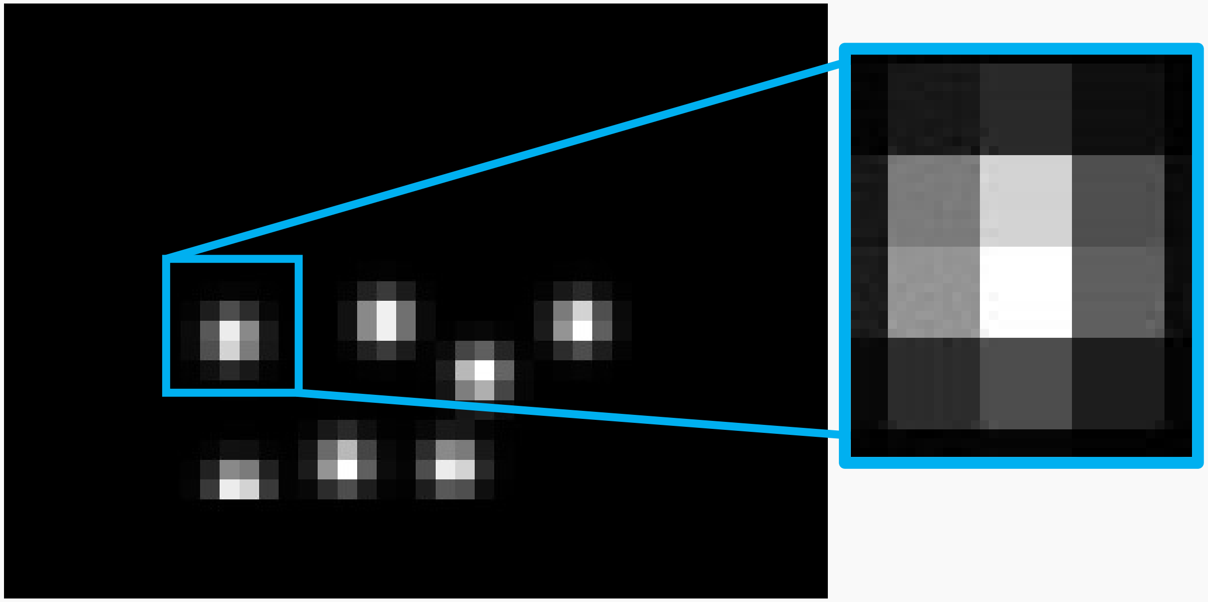
```
pcolor(xx, yy, zz)

shading interp

axis image
```

Use these commands when to plot an image to scale


10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10

Analyzing STORM images

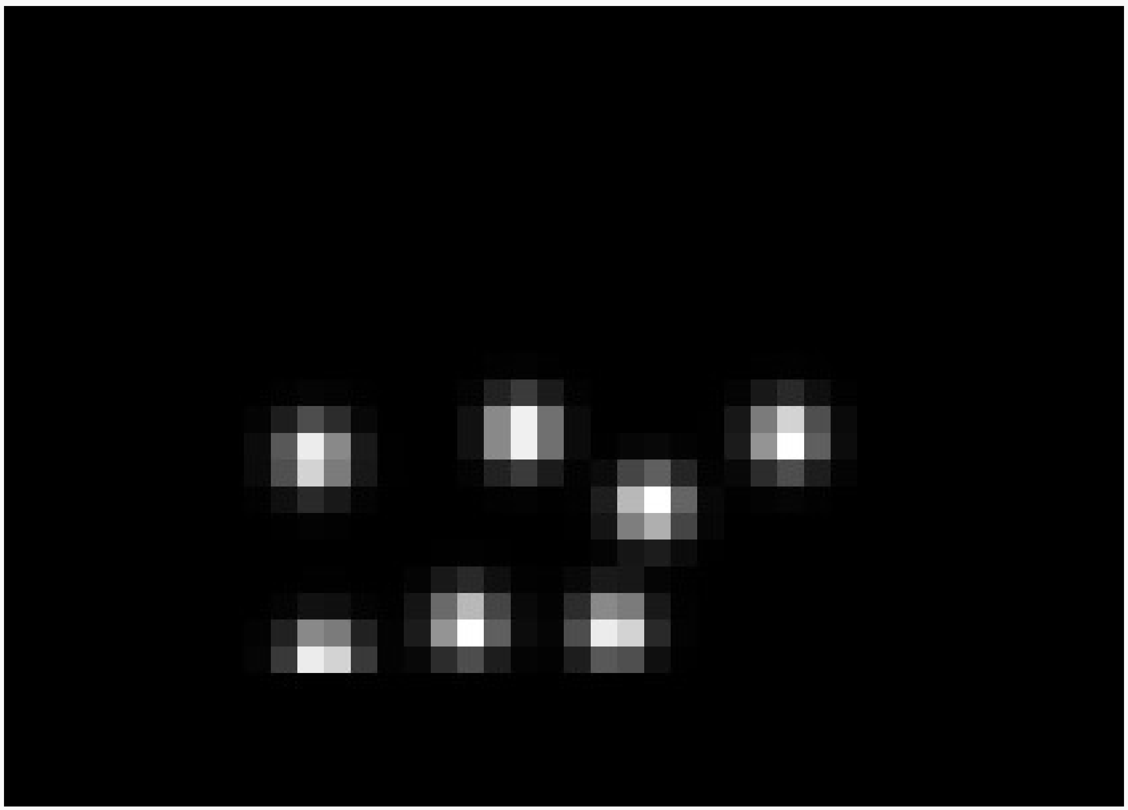
1. Detect approximate particle positions
2. Crop the image to get a sub-image containing only the particle
3. Fit the sub-image to the 2D Gaussian equation to get its position (particle localization)
4. Reconstruct the image by drawing at the particle positions at a higher resolution (smaller grid spacing)



Read in the first frame of the STORM data set

Please start a new script

```
I = imread('hw9_STORMdata.tif')
```



Info about image:

- 5 – 8 particles
- No noise and no background
- A small border has been added around the image to help with indexing the sub-images

Detecting the approximate position of particles using the regional maxima transform

```
BW = imregionalmax(I)
```

`imregionalmax` identifies regional maxima in the grayscale image

Local search region

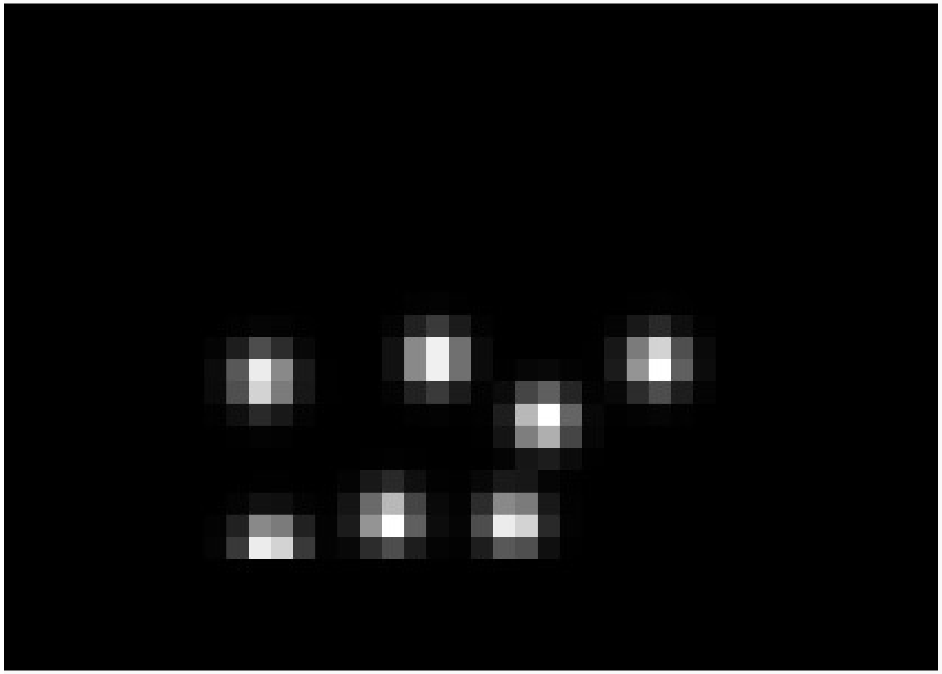
Input

10	10	10	10	10	
10	10	44	10	10	10
10	10	10	45	10	10
10	10	10	10	44	10
10	10	10	10	10	10
10	33	33	33	10	10
10	33	33	33	10	10
10	33	33	33	10	10
10	10	10	10	10	10
10	10	10	10	10	10

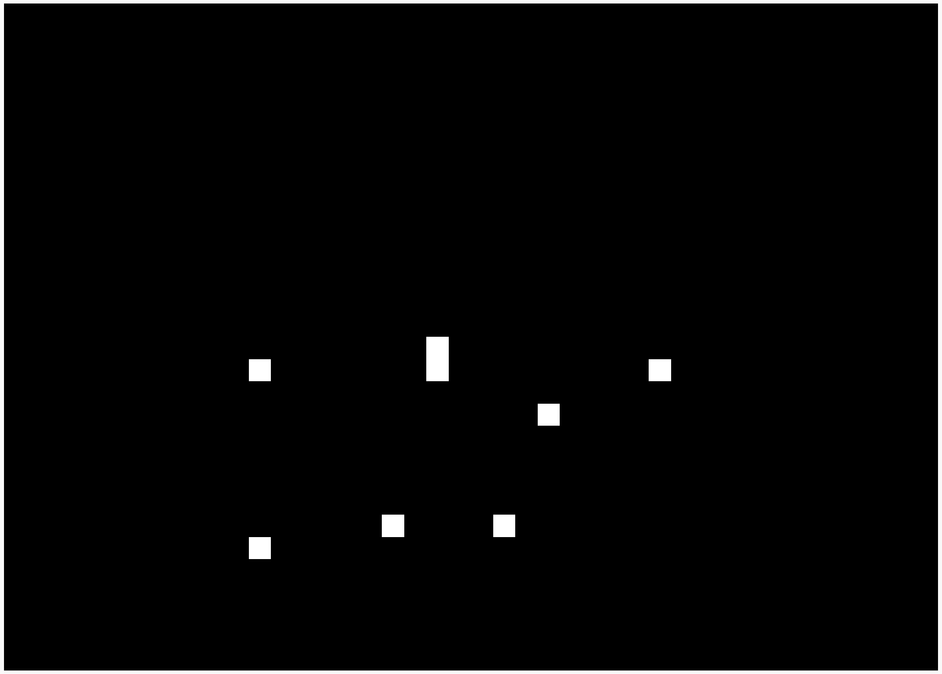
Output (logical array)

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	1	1	1	0	0
0	1	1	1	0	0
0	1	1	1	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Image `I`



Regional maxima mask `BW`



Finding the row and column of true pixels

```
[row, col] = find(BW)
```

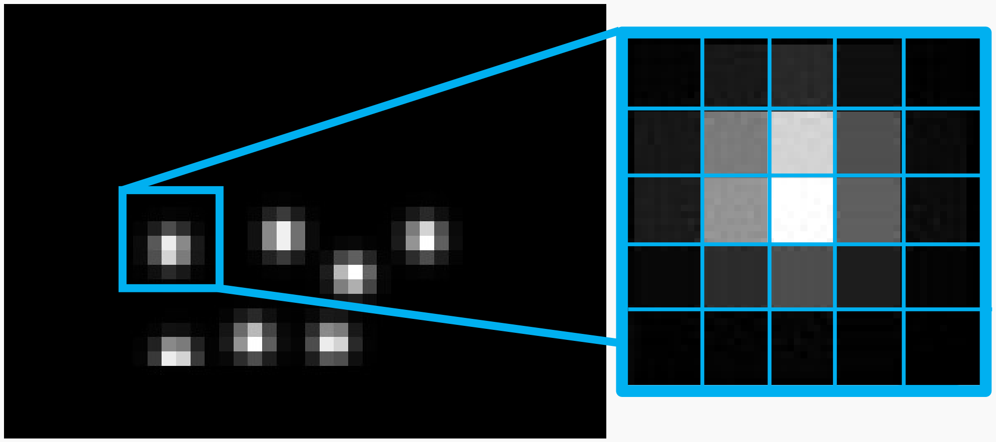
`find` returns the row and column indices of true elements in the logical array BW

`row and col` are the approximate locations of the particles. We will use curve-fitting to get a more accurate position

How many particles were found?

```
numParticles = numel(row)
```

Indexing a sub-image of the bead



- How to index a 5x5 sub-matrix from the image `I` around the first particle?

```
Icrop = I((row(1)-2):(row(1)+2), (col(1)-2):(col(1)+2))
```

- Why index the sub-image?

  Fitting is more accurate as it is not affected by the error from the rest of the image – could lead to wrong estimates

Defining a fitting model

- MATLAB does not have a built-in model for a 2D Gaussian so we have to define one

$$z = A \exp\left(-\frac{(x-B)^2+(y-C)^2}{2D^2}\right)$$

```
gauss2Dmodel = ...

     fittype('A * exp(-((xx - B).^2 + (yy - C).^2) /

(2*D.^2))', 'independent', {'xx', 'yy'});
```

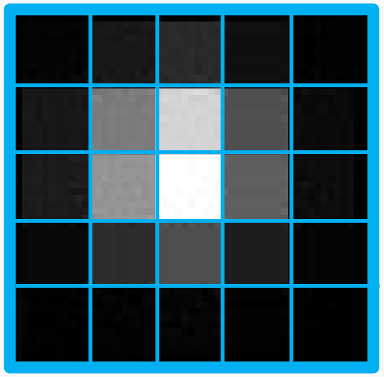
`xx` and `yy` are independent variables, i.e. will not be fit

`fit` function will search for A, B, C, and D

2D curvefitting or surface fitting

```
fitObj = fit([x, y], z, model)
```

x, y, and z are column vectors



What are x and y?

```
xdata = 1:5;
ydata = 1:5;
[xdata, ydata] = meshgrid(xdata, ydata);
```

What is z ?

The cropped image values (Icrop)

2D curvefitting or surface fitting

```
fitObj = fit([x, y], z, model)
```

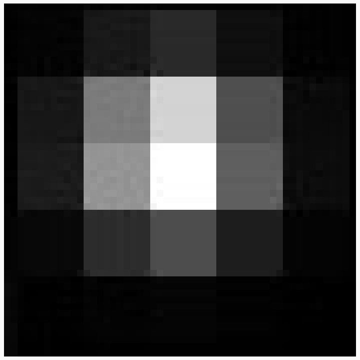
How to convert a matrix into a column vector?

```
x(:)
```

```
fitObj = fit([xx(:), yy(:)], Icrop(:), gauss2Dmodel)
```

Check: You should get values for B and C between 2 - 4

Improve fitting by adding starting guess



$$z = A \exp\left(-\frac{(x-B)^2+(y-C)^2}{2D^2}\right)$$

What are good initial guesses for A, B, C, and D?

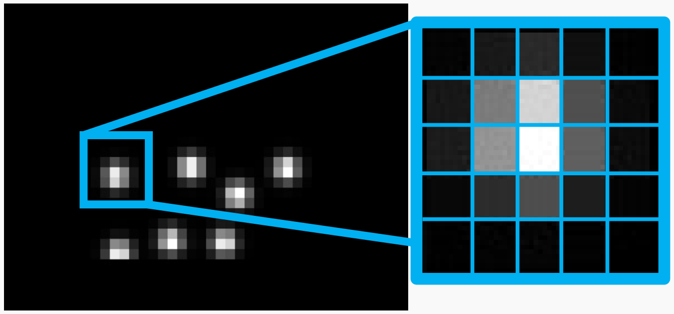
```
A = max(Icrop(:))
B = 3
C = 3
D = 2
```

Improve fitting by adding starting guess

```
fitObj = fit([xdata(:), ydata(:)], Icrop(:),...
     gauss2Dmodel, ...
    'StartPoint', [max(Icrop(:)), 3, 3, 2]);
```

- Check the fit again

What is the particle position?



```
x = fitObj.B
y = fitObj.C
```

WAIT! Remember we cropped the image, so we have to add an offset back in

```
Icrop = I((row(1)-2):(row(1)+2), (col(1)-2):(col(1)+2))
```

Actual particle positions:

```
x = fitObj.B + col(1) - 2
y = fitObj.C + row(1) - 2
```

Modify the code to loop over all particles

```
%Read in the image
I = imread('hw9_STORMdata.tif');

%Find approximate positions of the particle
BW = imregionalmax(I);
[row, col] = find(BW);

%Generate the x-data and y-data axes
xdata = 1:5;
ydata = 1:5;
[xdata, ydata] = meshgrid(xdata, ydata);

%Declare the 2D Gaussian surface model
gauss2Dmodel = fittype('A * exp(-((xx - B).^2 + (yy - C).^2) /  (2*D.^2))',...
    'independent', {'xx', 'yy'});

%Crop a 5x5 image around each particle
Icrop = double(I((row(1) - 2):(row(1) + 2), (col(1) - 2):(col(1) + 2)));

%Fit the surface - with a guess to the starting values
fitObj = fit([xdata(:), ydata(:)], Icrop(:), gauss2Dmodel, ...
    'StartPoint', [max(Icrop, [], 'all'), 3, 3, 2]);
```

Modify the code to loop over all particles

```
%Read in the image
I = imread('hw9_STORMdata.tif');

%Find approximate positions of the particle
BW = imregionalmax(I);
[row, col] = find(BW);

%Generate the x-data and y-data axes
xdata = 1:5;
ydata = 1:5;
[xdata, ydata] = meshgrid(xdata, ydata);

%Declare the 2D Gaussian surface model
gauss2Dmodel = fittype('A * exp(-((xx - B).^2 + (yy - C).^2) /  (2*D.^2))',...
    'independent', {'xx', 'yy'});

for iP = 1:numel(row)

    %Crop a 5x5 image around each particle
    Icrop = double(I((row(iP) - 2):(row(iP) + 2), (col(iP) - 2):(col(iP) + 2)));

    %Fit the surface - with a guess to the starting values
    fitObj = fit([xdata(:), ydata(:)], Icrop(:), gauss2Dmodel, ...
        'StartPoint', [max(Icrop, [], 'all'), 3, 3, 2]);

end
```

numel(row) = number of objects

Remember to use the for loop index variable

Modify the code to save the fitted positions

```
for iP = 1:numel(row)

   %Crop a 5x5 image around each particle
   Icrop = double(I((row(iP) - 2):(row(iP) + 2), (col(iP) -
                2):(col(iP) + 2)));

   %Fit the surface - with a guess to the starting values
   fitObj = fit...

end
```

```
%Initialize a matrix of NaNs (not-a-numbers) to store the position data
storePos = nan(16000, 2);
nP = 0;  %Counter of number of found particles


for iP = 1:numel(row)

    %Crop a 5x5 image around each particle
    Icrop = double(I((row(iP) - 2):(row(iP) + 2), (col(iP) - 2):(col(iP) + 2)));

    %Fit the surface - with a guess to the starting values
    fitObj = fit([xdata(:), ydata(:)], Icrop(:), gauss2Dmodel, ...
        'StartPoint', [max(Icrop, [], 'all'), 3, 3, 2]);

    %Save the fitted positions (remember to correct for the offset since we
    %cropped the image)
    storePos(nP + 1, :) = [fitObj.B, fitObj.C] + [col(iP) - 2, row(iP) - 2];

    %Increment the counter
    nP = nP + 1;

end

%Remove the NaNs
storePos(all(isnan(storePos), 2), :) = [];
```

Code is available on Canvas: hw9_example.m

Explanation in slides at end of lecture notes

Reconstructing the final image

- To reconstruct the final image, draw at each localized particle position
- This is similar to the last question in homework 7 (plotting different Gaussians)
- A few different ways to draw STORM images:
  - Draw crosses/circles at each particle location
  - Bin the locations in a spatial histogram
  - **Draw Gaussians of fixed width**
  - Draw Gaussians with widths depending on localization accuracy (most accurate, but can be difficult)

Reconstructing the final image

1. Convert the stored positions from pixels to microns. Original image has steps of 0.1 μm (100x objective, 1.4 NA)

2. Since we have a model of the data, the final image can have higher resolution (finer grid spacing) than the original image

- Draw the output image with a grid spacing of 0.001 μm

- Assume the localization accuracy is ~0.01 μm (roughly from the confidence intervals after fitting)

- So output Gaussians should have A = 1, B and C from fit data, and D = 0.01

Example drawing code

```
storePos = storePos * 0.1;
```

Convert the stored positions to microns

```
xxOut = 0:0.001:(size(I, 2) * 0.1);
yyOut = 0:0.001:(size(I, 1) * 0.1);
[xxOut, yyOut] = meshgrid(xxOut, yyOut);
```

Define the grid in microns
Grid should have physical limits from the original image

```
imgOut = zeros(size(xxOut));
```

Initialize a matrix of zeros for output image

```
for iP = 1:size(storePos, 1)

    imgOut = imgOut + ...
        exp(- ((xxOut - storePos(iP, 1)).^2 + ...
              (yyOut - storePos(iP, 2)).^2) / (2 * 0.01^2));

end
```

Our friend, the 2D Gaussian equation

Example code from today is on Canvas

You will need to update the code to loop over every image in the STORM dataset **Add the loop BEFORE the drawing code**

Advice for writing the code

- It is always a good idea to check if things are working on a small subset of data before running the full code

- When developing your code to analyze the STORM dataset, start with just the first image.

- Check that your code runs and the correct output image is generated.

- Then check the first 30 frames.

- Run the whole code once everything looks like it is working (it took my computer ~30 mins to process the whole dataset)

Localization accuracy

$$\langle(\Delta x)^2\rangle = \frac{s^2 + a^2/12}{N} + \frac{8\pi s^4 b^2}{a^2 N^2}$$

N = number of photons collected

s = sigma (C) of the Gaussian

a = size of the pixel

b = background noise

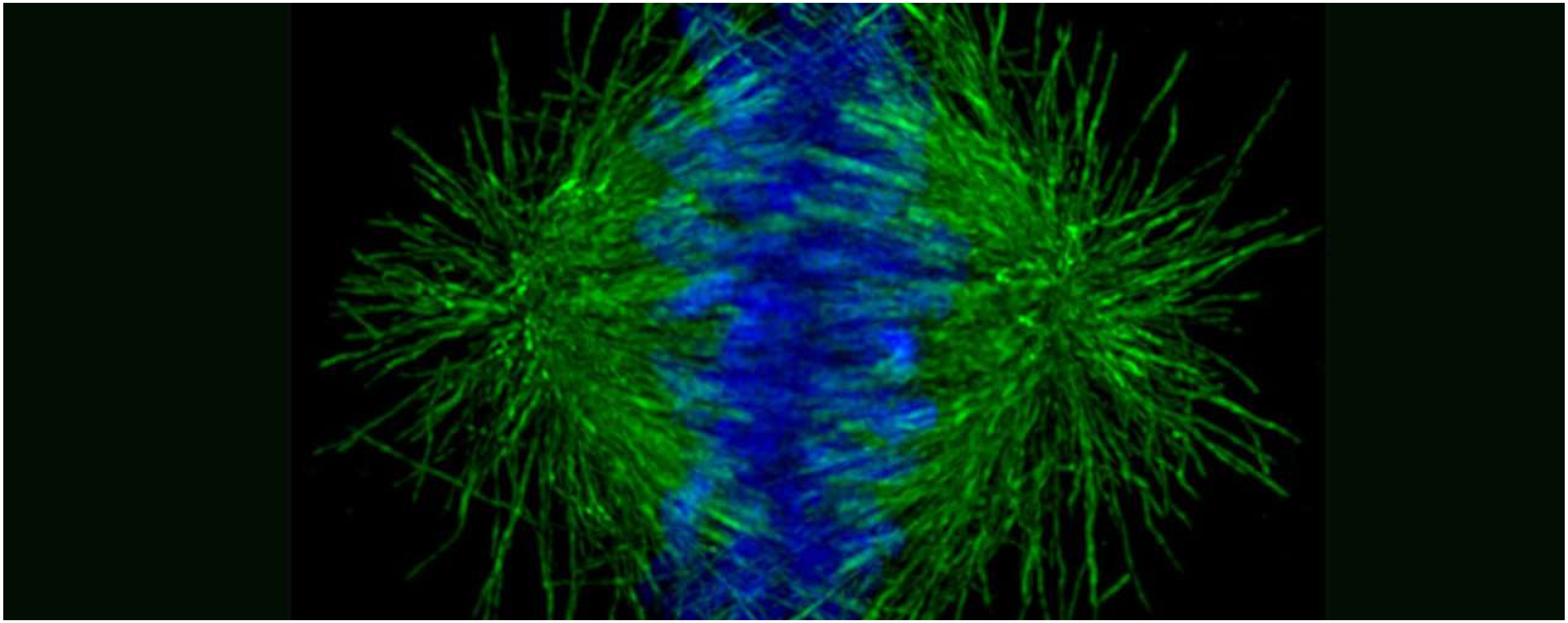
If s >> b (i.e. no noise):

$$\Delta x \approx s/\sqrt{N}$$

Precise nanometer localization analysis for individual fluorescent probes. Thompson, Larson and Webb. Biophys. J. 82 2775-2783 (2002).

Localization accuracy

- Using Δx as the width of the Gaussian gives you an image of the probability density of the object
- The probability density shows the likelihood that the object is actually at the location specified
- The width of objects gives you the upper bound of what the actual size is



Localizing Single Particles and STORM

MCDB/BCHM 4312/5312

The following slides are additional reading/examples

Material marked optional will not be in exam

Storing data of known size from for loops

Example: Compute distance between the two bees

```
%Initialize a vector to store the data
dist_between_bees = zeros(1, 39);

for idx = 1:39
    dist_between_bees(idx) = sqrt((beePos1(idx, 1) -
        beePos2(idx, 1)).^2) % ... truncated
end
```

Storing data of UKNOWN ssize from for loops

If you do not know the size, you can grow the matrix (remove the initialization step)

```
for idx = 1:39
    dist_between_bees(idx) = sqrt((beePos1(idx, 1) -
        beePos2(idx, 1)).^2) % ... truncated
end
```

Storing data of UKNOWN size from for loops

Better solution: make a larger matrix than you think you need and remove the unassigned rows later

```
%Initialize a matrix of NaNs (not-a-number)
dist_between_bees = nan(1, 100);
for idx = 1:39
    dist_between_bees(idx) = sqrt((beePos1(idx, 1) -
        beePos2(idx, 1)).^2) % ... truncated
end

%Remove unassigned rows (they will be NaNs)
dist_between_bees(isnan(dist_between_bees)) = [];
```

Difference of Gaussians filter (Optional material)

- To compute the difference of Gaussians, perform a Gaussian blur (imguassfilt) on the image twice

- The sigma of the first filter should be smaller than the sigma of the second filter (i.e. the second filter should blur more)

- The TrackMate toolbox in ImageJ uses:

$$\sigma_1 = 1/(1+\sqrt{2}) \times d$$
$$\sigma_2 = \sqrt{(2)} \times \sigma_1$$

Note: σ2 > σ1

d = expected diameter of particle in pixels

- The result from the second filter is subtracted from the result of the first filter

- Threshold the resulting image (particles will be negative)

- This website has a good description of the math:
  - http://fourier.eng.hmc.edu/e161/lectures/gradient/node9.html

Difference of Gaussians filter (Optional material)

Example code

```
I = double(imread('hw9_STORMdata.tif'));

sigma1 = 1/(1 + sqrt(2)) * 10;  %Assume diameter of 10 px
sigma2 = sqrt(2) * sigma1;

Igauss1 = imgaussfilt(I, sigma1);
Igauss2 = imgaussfilt(I, sigma2);

Idiff = Igauss2 - Igauss1;

mask = Idiff < -10;

imshowpair(I, mask)
```

Difference of Gaussians filter (Optional material)

Example output (green = image, magenta = mask)