PatternMatching

The purpose of the pattern matching routine is to generate defocused (or focused) model images of single dipole emitters as imaged by a standard wide-field microscope on a CCD, an to use these generate model images for identifying and localizing corresponding images in a measure wide-field image. First at all, the routine reads in the measured image to process. This is done in the first four lines, for example by

```
if exist('im')==0
    im = double(imread('c:\Joerg\example.png'));
    im = im(100:end-101,100:end-100);
end
```

where one first asks whether the image im is already within working memory, and if not, it reads the image from the specified location (here at c:\Joerg\example.png).

Next, the routine calculates the model images of a single molecule emitter for a discrete set of possible orientations. This is done by

```
model = PatternGeneration(z, NA, n0, n, n1, d0, d, d1, lamem, mag,
focus, atf, ring, pixel, nn, be_res, al_res, pic);
```

`PatternGeneration` calculates the images of an oscillating electric dipole emitter (e.g. single molecule) within a planar stratified structure as imaged by a high-aperture microscope onto a CCD chip. The emitter is assumed to be located at the bottom interface of a layer with refractive index n and thickness d that is sandwiched between two stacks of layers with refractive indices $n_{1,1}$ through $n_{1,k1}$ and thickness values $d_{1,1}$ through $d_{1,k1-1}$ (bottom stack, from bottom to top) and refractive indices $n_{2,1}$ through $n_{2,k2}$ and thickness values $d_{2,1}$ through $d_{2,k2-1}$ (top stack, from bottom to top).

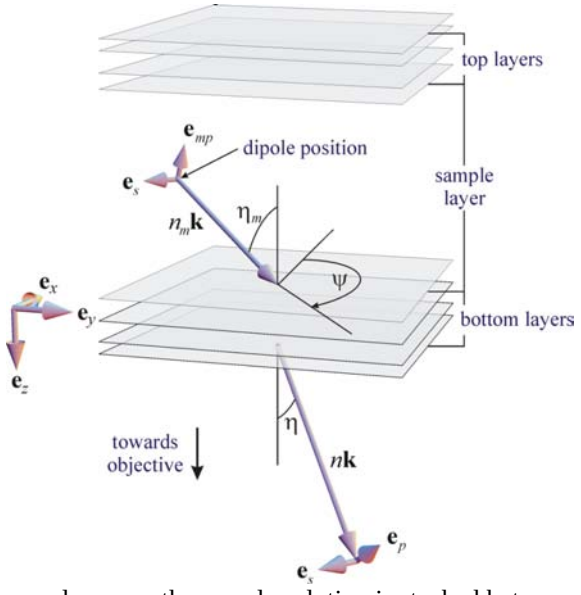


Fig.1: Geometry of he sample space: the sample solution is stacked between stacks of layers with homogeneous optical properties.

Images are calculated for all possible orientations of the emitter defined by the two angles β and α, where β is the inclination towards the optical axis and α the azimuthal

angle around the optical axis. The values of θ and ϕ are varied incrementally by steps defined by be_res (β) and al_res (α). The image is calculated for a CCD camera with assumed pixel size of **`pixel`** in μm, the generated images have square size of (2***nn*** + 1)*(2***nn*** + 1).

The following table gives a detailed description of the input and output variables.

Input variables:

`z`	– position of emitter above bottom of its layer in μm;
`NA`	– numerical aperture;
`n1`	– vector of refractive indices of the stack below the molecule's layer;
`n`	– refractive index of the molecule's layer;
`n2`	– vector of refractive indices of the stack above the molecule's layer;
`d1`	– vector of layer thickness values of the stack below the molecule's layer (length(d1) = length(n1)-1);
`d`	– thickness of molecule's layer;
`d2`	– vector of layer thickness values of the stack above the molecule's layer (length(d2) = length(n2)-1).
`lamem`	– center emission wavelength in μm;
`mag`	– imaging magnification;
`focus`	– defocusing value in μm, giving the distance the objective is moved out of focus ***towards*** the sample;
`atf`	– correction vector for coverslide thickness effects. If **`atf = []`**, no coverslide effects are taken into account. If **`atf`** is a 2-element vector, the first is the refractive index of the coverslide glass, the second the deviation (in μm) of the coverslide thickness from its design value;
`ring`	– optional parameter for a potential phase plate (default []);
`pixel`	– side length of one pixel in μm;
`nn`	– the image is calculated over (2***nn*** +1)*(2***nn*** +1) pixels (default **`nn = 10`**)
`be_res`	– increment in polar angle (default 10°)
`al_res`	– increment in azimuthal angle (default 10°)
`pic`	– if non-zero and non-empty, the calculated images are shown in a figure (default 0)

Output variable:

`model` – structure with the following fields:

`model.rho`	– radial coordinate
`model.theta`	– vector of polar angles
`model.phi`	– vector of azimuthal angles
`model.mask`	– three-dimensional field of images: **`model.mask(:,:,j)`** is the image corresponding to the emitter orientation with **`β = model.theta(j)`** and **`α = model.phi(j)`**;

The next and final step is to locate modeled patterns within the measured image. This is done by the commands

```
bck = Disk((size(model.mask,1)-1)/2);
[err, bim, cim, sim, xc, yc, bc, cc, sc, len, imm] = FindPat-
tern(im,model.mask,bck,bck);
```

where the first line calculates a background pattern (uniform disc with a radius **nn** pixel), and the pattern-finding routine **FindPattern.m.** The general call of this latter routine is

```
[err, bim, cim, sim, xc, yc, bc, cc, sc, len, imm] = FindPat-
tern(im,mask,sup,bck,sze,tsh,fun,flag)
```

Before explaining the meaning of all the variables, I will describe here briefly the pattern matching algorithm. Let x_{jk} be the intensity value of the pixel with co-ordinates (j,k) in the original image. The task is to identify and localize pre-defined patterns within the image. Thus, the algorithm has to compare all possible sub-regions of the image with a discrete set of R predefined patterns $p_{jk}^{(r)}$ which shall have the size (2L+1)× (2L+1) (with $1 \le r \le R$ counting the patterns and $-L \le j \le L$ and $-L \le k \le L$ counting the pixels of the patterns). Additionally, a uniform background pattern b may be super-imposed, and the pattern-matching may be restricted to non-rectangular sub-areas defined by the support matrices $s_{jk}^{(r)}$ of same size as the patterns but having values one and zero only. Furthermore it is assumed that within any given sub-area of the image, only one of the R patterns can be present. Pattern comparison is done in the following way. The algorithm fits, by a least-square method, each of the R patterns (plus a flat background) to all possible sub-areas of the image. For every sub-area, the algorithm chooses the pattern yielding minimal least-square error to be the most likely pattern present in the given sub-area. Thus, for any given sub-area of the image x with pixels $m-L \le j \le m+L$ and $n-L \le k \le n+L$, the algorithm tries to find $2R$ parameters $c_{jk}^{(r)}$ and $d_{nm}^{(r)}$ so that the least-square errors $e_{mn}^{(r)}$,

$$e_{mn}^{(r)} = \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} \left(x_{m+j,n+k} - c_{mn}^{(r)} p_{jk}^{(r)} - d_{mn}^{(r)} b_{jk} \right)^2 , \tag{1}$$

are minimized. Subsequently, it is assumed that the patterns $p^{(r)}{}_{jk}$ as well as the background pattern b_{jk} are all square normalized so that

$$\sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} \left(p_{jk}^{(r)} \right)^2 = 1 \text{ and } \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} \left(b_{jk} \right)^2 = 1. \tag{2}$$

By differentiating with respect to the coefficients $c_{mn}^{(r)}$ and $d_{mn}^{(r)}$, these conditions lead to the $2R$ equations

$$\frac{\partial e_{mn}^{(r)}}{\partial c_{mn}^{(r)}} = \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} \left(x_{m+j,n+k} - c_{mn}^{(r)} p_{jk}^{(r)} - d_{mn}^{(r)} b_{jk} \right) p_{jk}^{(r)} = 0 ,$$

$$\frac{\partial e_{mn}^{(r)}}{\partial d_{mn}^{(r)}} = \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} \left(x_{m+j,n+k} - c_{mn}^{(r)} p_{jk}^{(r)} - d_{mn}^{(r)} b_{jk} \right) b_{jk} = 0 , \tag{3}$$

with $1 \le r \le R$. Introducing the abbreviations

$$X_{mn} \equiv \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} x_{m+j,n+k} b_{jk} ,$$

$$Q_{mn}^{(r)} \equiv \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} x_{m+j,n+k} p_{jk}^{(r)} , \tag{4}$$

$$P^{(r)} \equiv \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} p_{jk}^{(r)} b_{jk}$$

the two equations can be rewritten in matrix form as

$$\begin{pmatrix} 1 & P^{(r)} \\ P^{(r)} & 1 \end{pmatrix} \begin{pmatrix} c_{mn}^{(r)} \\ d_{mn}^{(r)} \end{pmatrix} = \begin{pmatrix} Q_{mn}^{(r)} \\ X_{mn} \end{pmatrix} \tag{5}$$

with the explicit solution

$$\begin{pmatrix} c_{mn}^{(r)} \\ d_{mn}^{(r)} \end{pmatrix} = \begin{pmatrix} 1 & P^{(r)} \\ P^{(r)} & 1 \end{pmatrix}^{-1} \begin{pmatrix} Q_{mn}^{(r)} \\ X_{mn} \end{pmatrix} = \frac{1}{1 - \left(P^{(r)} \right)^2} \begin{pmatrix} 1 & -P^{(r)} \\ -P^{(r)} & 1 \end{pmatrix} \begin{pmatrix} Q_{mn}^{(r)} \\ X_{mn} \end{pmatrix} . \tag{6}$$

The square normalization of the patterns, together with the condition that all pixel values of the patterns are non-negative (and at least one pixel value positive), assures that $0 < P^{(r)} < 1$ and that the solution is well-defined and real. With the coefficients $c_{mn}^{(r)}$ and $b_{mn}^{(r)}$ explicitly known, the R errors $e_{mn}^{(r)}$ are calculated as

$$e_{mn}^{(r)} = \left(X^2 \right)_{mn} - \frac{\left(Q_{mn}^{(r)} \right)^2 + \left(X_{mn} \right)^2 - 2P^{(r)} Q_{mn}^{(r)} X_{mn}}{1 - \left(P^{(r)} \right)^2} \tag{7}$$

where the additional abbreviation

$$\left(X^2 \right)_{mn} \equiv \sum_{j=-L}^{L} \sum_{k=-L}^{L} s_{jk} x_{m+j,n+k}^2 \tag{8}$$

was introduced. The beauty of equations (6) through (8) is that they involve only two-dimensional convolutions of the image x with the patterns $p^{(r)}$ and b, which can be

calculated efficiently by using fast Fourier transforms. Thus, the flow of the image analysis algorithm is summarized as follows:

1. Calculate the $R+1$ two-dimensional convolutions of the image x with the R patterns $p^{(r)}$ and the background pattern b, see Eqs.(4).
2. Calculate the $2R$ "coefficient images" $c^{(r)}$ and $d^{(r)}$ according to Eq.(6).
3. Calculate the R "error images" $e^{(r)}$ according to Eq.(7), involving the calculation of the "squared image" X^2.
4. For all positions (m,n), the pattern with minimum error $e_{mn}^{(r)}$ is chosen as the most likely pattern for that position. Thus, two new images $\tilde{c}$ and $\tilde{e}$ are generated with values $\tilde{c}_{mn}$ and $\tilde{e}_{mn}$ composed by those values $c_{mn}^{(r)}$ and $e_{mn}^{(r)}$ having minimum $e_{mn}^{(r)}$.
5. Finally, pattern positions are localized by asking for positions (m,n) where the ratio $\tilde{c}_{mn}/\sqrt{\tilde{e}_{mn}}$ exceeds some predefined value κ. A good value for κ yielding excellent pattern recognition is close to one.

Now, we can specify the meaning of the input and output variables of **`FindPattern.m`**:

Input variables:

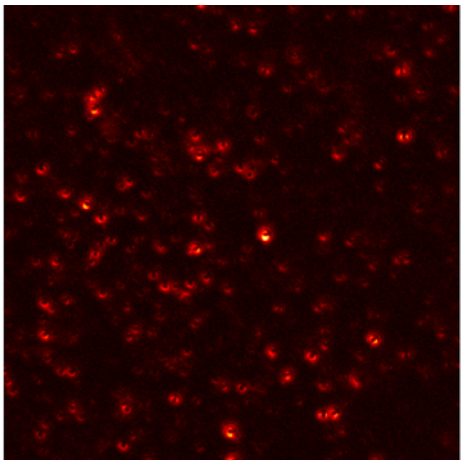
`im`	–	measured image
`mask`	–	3D stack of model patterns
`sup`	–	support of pattern matching, see s_{jk} above; usually a square or circular disk
`bck`	–	background b_{jk}, usually the same as **`sup`**
`sze`	–	a threshold parameter which can be used to make the pattern recognition more discriminating for suppressing false positives (default 1)
`tsh`	–	threshold value κ, see above (default 1).
`fun`	–	string with function for deciding a match, default is **`'cim>tsh*sqrt(err)'`**, where **`cim`** is $c_{mn}^{(r)}$ and **`err`** is $e_{mn}^{(r)}$ from above.
`flag`	–	if flag is 1, then pattern matching is suppressed ate the image borders within a stripe of half the width of the model images; this is important to prevent false recognitions in the border region (default 1).

Output variables:

err	–	error image $\min_r\left(e_{jk}^{(r)}\right)$
bim	–	$\min_{e_{jk}^{(r)}}\left(d_{jk}^{(r)}\right)$
cim	–	$\min_{e_{jk}^{(r)}}\left(c_{jk}^{(r)}\right)$
sim	–	$\min_{e_{jk}^{(r)}}\left(r\right)$
xc, yc	–	center coordinates of found patterns
bc	–	corresponding values of **bim**
cc	–	corresponding values of **cim**
sc	–	corresponding pattern index
len	–	number of found patterns
imm	–	reconstructed image with found model patterns

Example

As example consider the following defocused single-molecule image:

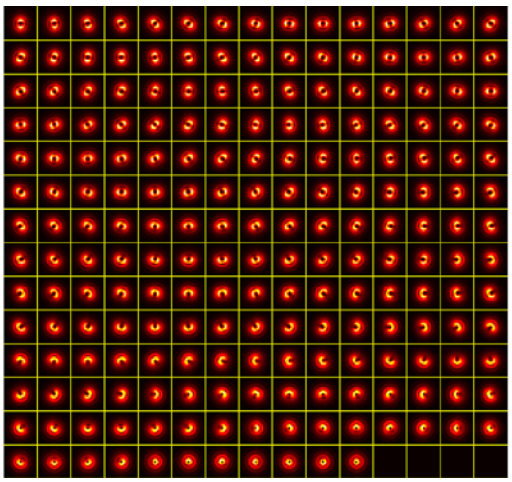


The parameters used for the pattern generation are:

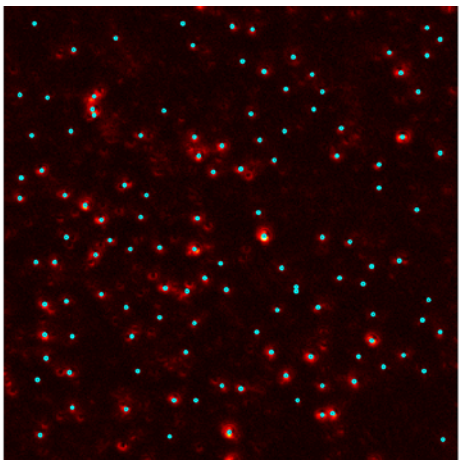
```
z = 0;
NA = 1.40;
n0 = 1.52;
n = 1.49;
n1 = 1.0;
d0 = [];
d = 0.01;
d1 = [];
lamem = 0.57;
mag = 140;
focus = 0.65;
atf = [];
ring = [];
pixel = 16;
pic = 0;
be_res = [];
```

```
al_res = [];
nn = [];
```

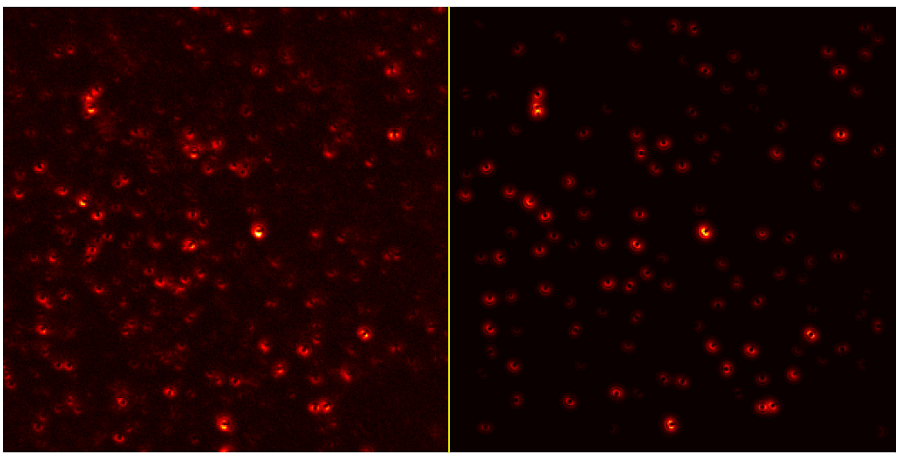
The resulting model patterns are displayed by **`CombineImages(model.mask,15,15)`**:



After applying the pattern matching algorithm, one can display the found coordinates of the patterns with **`mim(im); hold on; plot(xc,yc,'oc'); hold off`**



Or one can display, the original and the reconstructed image by **`CombineImages(cat(3,im,imm),1,2)`:**



The index of the corresponding pattern is stored in the vector **`sc`**, thus for the j_{th} pattern, the best matching angular orientations are **`model.theta(sc(j))`** and **`model.phi(sc(j))`**.