Single molecule restriction digest
Tracking Tethers

Hernan Garcia

April 27, 2009

1 Assignment

Follow the tutorial described in the remainder of this write up for one frame of your choice taken during the single-molecule session. Create a gallery of some of the images generated in intermediate steps of the analysis: `Im, ImThresh, ImMask,` and `ImOverlay`. Also, make sure to show the before and after pictures if you're cropping the images.

Once you've worked out a strategy to find the beads, analyze your whole movie (or skip every two or three frames if your movie is too long) and generate a plot of the number of beads found as a function of time. You might want to show some frames of the movie corresponding, for example, to the first, last and some middle frame.

2 A little tutorial

Our objective is to find the number of tethered beads remaining as a function of time after flowing in restriction enzyme. Even though we don't expect each group to have comparable results in terms of the time scales obtained, we think of this exercise as a good way to deepen your knowledge of image analysis using Matlab.

First, we need to decide on a strategy to work with the images. You should move all the TIF files acquired with Micro-Manager to a folder, where we'll be working from. Before writing the part of the code that deals with loading and analyzing multiple images, it is always a good idea to load only one in order to optimize your choice of parameters.

We then start by loading one of the images. In the case of the little movie I took, the image I choose to optimize my analysis with is called `img_000000000_Snap_000.tif`. Let's load it and display it as follows

```
Im=imread('img_000000000_Snap_000.tif');
imshow(Im,[])
```

In some cases you might decide that you have too many beads or you might realize that some part of your field of view is out of focus. If you want to keep just one section of your image this can be easily done remembering that an image in Matlab is just a matrix. If you want to find out the size of this matrix you just have to do

```
size(Im)
```

In my case I got two numbers: `520 696`. The first number tells you the number of rows and the second number tells you the number of columns. I have a good mnemonic rule for remembering that it's rows first and then columns, but it only makes sense in Spanish. You can ask him if you want to find out what it is. In any case, let's say I only want to keep the lower-right quadrant of the image. This would mean that I want to keep rows $520/4 * 3 = 390$ through 520 and columns $696/4 * 3 = 522$ through 696. I can, for example, redefine the image

```
Im=Im(390:520,522:696);
```

What does $390 : 520$ do here? Try executing it in Matlab and you'll see that what you get is all integers between those two numbers. It's telling Matlab to start a new image with the elements of the old `Im` given by the matrix $390 : 520$ - $522 : 696$. Keep in mind that this step is optional and might not be necessary.

Now you need to experiment with the threshold. Some people will have the beads focussed such that they're darker than the background and for some others they will be brighter. If, for example, we're dealing with bright beads we can try the threshold

```
ImThresh=Im>2500;
imshow(ImThresh,[])
```

Experiment a little bit with setting different thresholds. You'll always have a little bit of noise. It's all about finding a compromise between getting good regions for the beads and not thresholding too much on the noise. The closer your threshold to the background the bigger the areas associated with beads will be, but also the more false positives you'll get!

We now run `bwlabel`, which labels each region in the threshold image

```
ImLabel=bwlabel(Im>2500);
```

In my case, this gave me 9 different regions. Since the regions in `ImLabel` are numbered you can find out how many you got by doing

```
max(max(ImLabel))
```

In class we talked about discriminating the regions according to their areas. You can manually select an area and add up all the pixels that make it by doing `sum(sum(ImLabel==2))`. We choose to use the fancier function `regionprops`. We use it to tell it to calculate the area for each one of the regions found in ImLabel

```
ImRegion=regionprops(ImLabel,'Area');
```

You can see all the other properties this function can calculate by typing `help regionprops`.

`ImRegion` is a "Structure". This is Matlab's name for a data structure that can have multiple types of data for each entry. In this case ImRegion has as many elements as ImLabel had. Each element has the variable Area associated with it. If I want to see the the area of region 2 we can just do `ImRegion(2).Area`. We can also generate a vector with the areas of all the regions by doing

```
Areas=[ImRegion.Area];
```

Finally, we have to draw a second threshold. This has to do with the area corresponding to real beads. In my case, this is around 5 pixels. For example, running `Areas>5` gives a vector of the same length as `Areas` with zeros where the condition is not met and with ones where the condition is met. So, if we just want to know how many beads we found we have to do

```
Beads=sum(Areas>5);
```

How can we check how good our detection was? One idea is to overlay the original image and an image with the beads we found. In order to do that we first need to create a mask. This is an image that has ones only in areas where there was an approved bead. We use the function `find` to find the areas in the vector `Areas` that are above the threshold

```
Approved=find(Areas>5);
```

`find` returns the positions of the vector `Areas` where the condition `Areas>5` was met. This is basically the index numbers of `ImLabel` corresponding to the beads that have been approved. In order to generate a mask of the approved beads we first make a blank image

```
ImMask=zeros(size(Im));
```

This creates an image of the size of `Im` with zeros everywhere. Now we need to go through each approved bead. We'll go to each one of these beads and pull out their corresponding pixels from `ImLabel`

```
for i=1:length(Approved)
    ImMask=ImMask + (ImLabel==Approved(i));
end
```

Notice that we're adding pixels to `ImMask` in each round of the for loop. Now, let's try to overlay this mask with the original image. In order to do that we'll first rescale the original `Im` using

```
ImRescaled = mat2gray(Im);
```

Now, we'll create a composite just like we did with the fluorescent cells. We'll put `ImRescaled` on all channels and `ImMask` only on the red channel

```
ImOverlay = cat(3,ImMask+ImRescaled,ImRescaled,ImRescaled);
imshow(ImOverlay)
```

You should notice that the beads that you've detected are red.

2.1 Doing this for a movie

When dealing with a lot of images it's a good idea to analyze one at a time. After you're happy with the settings you obtained following the previous section of the tutorial now you're ready to go for the full movie. First, we get information on the images in the current folder using

```
D = dir('*,tif');
```

`D` is a structure with different fields regarding the files in the folder. The one we care about is `name`. Now, for each element in `D` we can repeat the analysis. We can start with a for loop

```
for i = 1:length(D)
    Im=imread(D(i).name);
```

you can repeat all the analysis coming in the end to the calculation of the beads found

```
    Beads(i)=sum(Areas>5);
end
```

Here I used my threshold of 5 pixels, but you might use something different. We stored the number of beads found in each frame in the vector `Beads`. You might want to overlay some of the images with their approved beads to make sure the algorithm is working well, but you probably won't want to do it for all the images you've taken!

Finally, we can plot our results. Let's say you took 100 frames (which would also be the length of the vector `Beads`) each one every 20 ms. We can create a `Time` vector with Matlab

```
Time = 0:20:19980;
```

This tells Matlab to create a vector that brings you from 0 to 19980 with a spacing of 20. We plot this using

```
plot(Time,Beads,'.-')
```

The option `'.-'` tells the function `plot` to mark your data with points and to join it using lines.

You can always get fancier with the analysis. You're encouraged to play with Matlab. It's really powerful and it will undoubtedly be useful beyond the course. For example, in our lab we use it to acquire data by controlling the microscope and to analyze at the same time. If you have any doubts or want to chat about other Matlab related stuff just send us an email or come to office hours.