Gene Expression Matlab Tutorial

Daniel Jones

May 4, 2011

1 Introduction

In this tutorial, we will demonstrate how to make quantitative gene expression measurements in *E. coli* using fluorescence microscopy. In other words, how to analyze that data you just acquired. Recall that the particular *E. coli* we're looking at have been genetically engineered to express yellow fluorescence protein (YFP). Our assumption is that the brightness of our cells in the fluorescence channel is proportional to the level of YFP expression in the cell.

The analysis can be broken into two major parts:

1. Segmentation of cells using the phase contrast image.
2. Cross-referencing the resulting mask with the fluorescence image to quantify the fluorescence intensity of each cell.

2 Segmentation Using Phase Contrast

The approach outlined in this section should look awfully familiar, since it's basically the same approach that we used for TPM segmentation. First, we'll load the phase contrast image:

```
im_phase = imread('phase.tif');
```

As before, we will perform threshold by binary thresholding. In this case, the cells are darker than the background. We can use *imtool* to explore the pixel values of our image, in order to find an intermediate value between the dark cell and bright background:

```
imtool(im_phase,[])
```

It looks like 700 is a reasonable cutoff. We'll set all pixels below 700 to 1, and all pixels above 500 to 0.

Just like last time, we'll label the mask and use the output of *regionprops* to perform quality control on the segmented cells. Recall that *regionprops* is a function that take a labeled image and computes properties of the labeled regions, such as area, centroid, major/minor axes, and many more. For our

purposes, we'll be content just to look at the areas of detected cells. We'll want to get rid of "cells" that are too small, and are hence junk, and "cells" that are too large, which most likely correspond to cell clumps that were segmented together.

```
maskL = bwlabel(mask);
props = regionprops(maskL,'Area');
areas = [props.Area];
```

The variable *areas* is a vector whose elements are the areas of each of the detected cells. We can use Matlab's *hist* command to explore the distribution of areas of detected cells. From doing so, it appears that the sweet spot for cells areas is between 250 and 550 pixels. So next, we will implement a filter that deletes all detected cells outside of this range. In addition, we'll use Matlab's *imclearborder*

```
area_ub = 500; % upper bound
area_lb = 250; % lower bound
for i=1:length(areas)
    if areas(i)<area_lb || areas(i)>area_ub
        mask(maskL==i) = 0; % what does this do?
    end
end
% While we're at it, clear any cells on the border (so we don't have ones
% that are half cut off)
mask = imclearborder(mask);
figure; imshow(mask) % much better!
```

Finally, relabel the mask. We'll use this in the next section.

```
maskL = bwlabel(mask);
```

Now we have a pretty good looking mask. It's time to use it to compute integrated fluorescence intensities of our cells.

3 Computing Fluorescence Intensities

First, we need to load our fluorescence image.

```
im_fluo = imread('fluorescence.tif');
```

The developers of Matlab, in their wisdom, anticipated the situation in which a labeled mask is cross-referenced to some other image. The syntax

```
STATS = regionprops(L, I,properties)
```

will compute properties of the image I using the label matrix L. Which properties are we interested in? Recall that we are looking for the integrated intensities (i.e., sum of all the pixels) of each of the detected cells. So if we know the mean fluorescence intensity of each cell, and the area of each cell, we can simply multiply the two to get the total (integrated) intensity.

```
% get regionprops of fluorescence image, using previously generated labeled
% mask
fluo_props = regionprops(maskL,im_fluo,'Area','MeanIntensity');
areas = [fluo_props.Area];
means = [fluo_props.MeanIntensity];

integrated_intensities = areas.*means;
```

Why did we write "areas.*means" instead of simply "areas*means"?

The vector *integrated_intensities* essentially quantifies the level of gene expression in each detected cell. It is a single-cell level measurement of gene expression. In the end, however, we'll only be using the mean level of gene expression in all cells from a given pad. To get that, we simply do:

```
mean_intensity = mean(integrated_intensities);
```

There's one important point that we've neglected so far. We've pretended that the fluorescence value of a pixel comes entirely from YFP. But even cells with no YFP are not completely dark in the fluorescence channel. Instead, they exhibit what we call **cellular autofluorescence**. Let I be the fluorescence intensity of a pixel inside a cell. Then

$$I = I_{\text{autofluo}} + I_{\text{YFP}}$$

where I_{autofluo} is the contribution due to autofluorescence, and I_{YFP} is the contribution due to YFP.

In order to correctly compute YFP expression, we need to subtract out this cellular autofluorescence. How do we know what value to subtract? This is what your control sample is for. You can use it to compute the average contribution from autofluorescence by finding the average fluorescence value of pixels inside your control cells. Once you know this autofluorescence value, you can replace the line

```
integrated_intensities = areas.*means;
```

with

```
integrated_intensities = areas.*(means-mean_autofluo);
```

where *mean_autofluo* is a variable containing the mean pixel value for autofluorescence (how does this work?).