**MATLAB for image processing: Session 4 worksheet**

For the final worksheet we’ll take the nuclear segmentation workflow from the past couple of sessions and add user controls and data plotting. At the end of Session 3’s worksheet we’d created a cell array containing Nucleus class objects for all the nuclei detected in the image. In order to use this on different images, it’s current necessary to write the file path in the code. For the final workflow we will add a file selection dialog that asks the user to pick the image to process. We will also test a couple of different methods for representing data visually: overlaying points on an image and histograms.

Note: To ensure everyone starts from the same place, the Session 4 worksheet repository contains the Nucleus class definition (“nucleus.m”) and a pair of pre-prepared functions for labelling a raw image and creating the cell array of Nucleus class objects:

* + Format: *label\_im = labelImage(input\_im)*
  + Description: Applies the filtering, thresholding and labelling steps from Session 2 to an image.
  + Arguments:
    - *input\_im* the image to process as a 2D numeric pixel array
  + Returns:
    - *label\_im* the labelled image as a 2D numeric pixel array. All pixels corresponding to the same nucleus are given the same pixel value. Background pixels are assigned a value of 0.
  + Format: *nuc\_objs* *= createNucleiObjects(lab\_im)*
  + Description: Converts a labelled image to individual objects. Each object stores its initial label (ID) and an Nx2 numeric array of row and column coordinates, where N is the number of pixels corresponding to that object.
  + Arguments:
    - *lab\_im* is the input labelled image to process. This should be a 2D numeric array.
  + Returned:
    - *nuc\_objs* is a cell array containing the Nucleus class objects. Each cell contains a single object.

1. **Getting the path to the input image**

Currently, we need to specify the path to the input image in the code; however, it would be neater to display a window where we can select this file. Doing so means we don’t need to continually alter the code; it also potentially makes the workflow easier to use for other people.

1. Create a new script file and save this to a location accessible to MATLAB.
2. Use the *uigetfile* function to display a file selection dialog. Set a filter, so the dialog only shows files with the extension “.tif”.

Note: You will need the full path to the file, not just its name.

1. Assign the selected file path to a variable.
2. Display the selected file path in the command window to verify this is working correctly.
3. **Creating a Nucleus object cell array for the selected image**

In this exercise we will run the provided functions (*labelImage* and *createNucleiObjects*) on the loaded image. There’s nothing new here, the purpose of this exercise is to ensure we can run the provided code.

1. Use the file selection dialog to read the “Example\_image.tif” file and assign the image pixel array to a variable.
2. Use the two provided functions to generate a cell array of Nucleus class objects.
3. Display the number of detected nuclei in the selected image.

Note: For “Example\_image.tif” there should be 71 nuclei detected at this point.

1. **Displaying plots over an image**

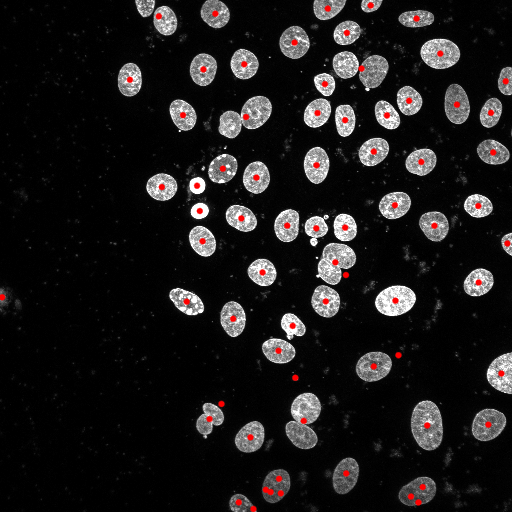
Next, we’ll display the loaded image in a figure window and plot the detected centroids over it. This makes use of the *hold on* feature of MATLAB’s plotting, which allows different plot types to be combined on a single set of axes. To keep our code tidy we will create a separate function, which will do this plotting. The function should take the raw image and nuclei cell array as arguments. There don’t need to be any returned values from this function, as the function will display the image.

1. Create a function file called *showNuclei*, which takes the raw image (i.e. the image loaded from file) and nuclei cell array as arguments. The rest of the code added during this exercise will go inside this function.
2. Use *imshow* to display the raw image.
3. Use *hold on* to allow subsequent plot objects to be added to the same axes.
4. For each nucleus in the cell array acquired during Exercise 2, use the *plot* function to draw a red spot on the axes.

Hint: Nucleus class objects have a *getCentroid* method, which returns the coordinates for the centroid of that object.

1. Test this function using the raw image and cell array created during Exercise 2.

The output from this should look similar to this:



1. **Acquiring values from dialogs**

From the image we just generated we can see there are many instances, especially towards the bottom of the image, where a single nucleus has been detected multiple times. These fragments can arise during segmentation of nuclei with low signal intensity. We can remove any nuclei with areas (number of pixels) lower than a user-defined threshold. Here, we will use a dialog box to ask the user for this threshold value.

Note: At this point, we are simply acquiring the threshold value. It will be implemented in the next exercise. As such, you can test this code in a separate script file, then copy it over during the next exercise.

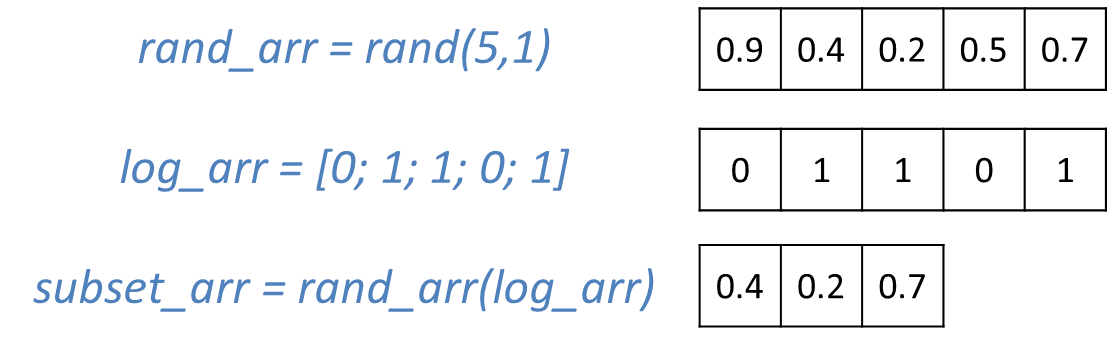
1. Use the *inputdlg* function get ask the user to enter a minimum nucleus area (specified in px2). Set the input dialog, so it has a default value of 50 px2.
2. Assign the returned value to a variable.

Note: The *inputdlg* function returns a cell array of strings, so you will need to extract the value you want.

1. Display the entered value in the command window to check this is being read correctly.
2. **Applying the minimum nucleus area filter**

It’s now time to apply the threshold value entered via the input dialog. To do this we will create another function called *applyAreaFilter*, which will take the nucleus object cell array and user-defined threshold as arguments. The function will output a new cell array containing only nuclei with areas larger than the threshold.

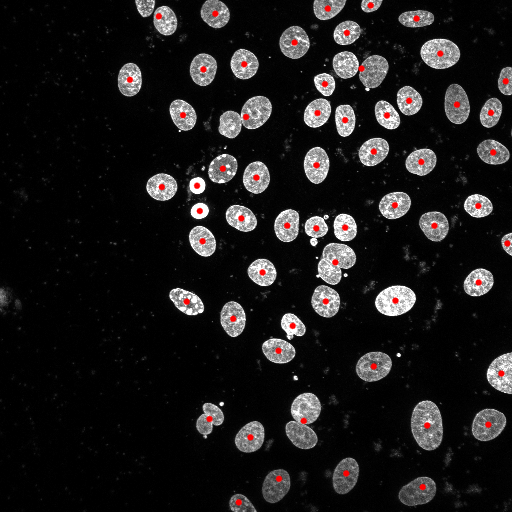
Note: There are many ways to do this, but I will describe steps for using logical arrays. In addition to accessing a subset of an array using indices, we can use a logical array. If we replace the indices with a logical array, MATLAB will return the elements of the array corresponding to a logical true value:



1. Create a new function file called *applyAreaFilter* with the nucleus object cell array and user-defined area threshold as arguments.
2. Initialise a logical array of equal length to the number of nuclei objects.
3. Iterate over all nuclei objects, using the *getArea* method to measure their area. If their area is greater than the user-defined threshold, set the corresponding logical array value to true (or false if it’s less).
4. Use logical indexing to create a new cell array containing only the nuclei objects larger than the threshold.
5. Display the number of nuclei objects to check this is working correctly.

Note: For a threshold of 50 px2 you should be left with 60 nuclei.

The output following the size filter should look like this:



1. **Creating a histogram**

Finally, we will create and display a histogram showing the distribution of nuclei areas. For this, it is necessary to create a numeric array containing all the nucleus area measurements.

1. Initialise a numeric array of equal length to the number of remaining nuclei objects (after the area filter).
2. Iterate over all the remaining nuclei objects, adding their area measurements to the numeric array
3. Use the *histogram* function to create a histogram with bins spaced 100 px2 apart in the range 0 to 1000 px2.
4. Add x and y axis labels to the figure.

The histogram should look like this:

