A Basic Introduction to Matlab

Liana Engie 12 February 2011

Matlab (**mat**rix **lab**oratory) is primarily-numerical-computing program used by mostly physicists and engineers. It is a powerful tool for mathematical programming, and has additional packages and toolboxes for just about everything, including symbolic engines (something akin to Maple), model-based design, statistics and data analysis, image processing, and more.

There are not many ways of formally learning image processing computation on the 5Cs. My current knowledge is the result of what I have happened to learn while working at a variety of physics and mathematical biology labs as well as wading through the help files and documentation of Matlab. I hope to give you a good overview of what I think will be helpful to you so that you don't have to build up from scratch in the same manner. You will probably get pretty familiar with the help files though. I'll say right now that I'm definitely not the best at Matlab; my knowledge is probably pretty patchy and fairly specific. Looking at this document so far it seems that it's just rather stream-of-conciousness, especially the functions list. If you think of something that should be on here, or see that there's something that you'd really want to know how to do that isn't on the list, let me know.

This document contains some basic programming instructions and tips, then gets into more specific image processing functions. Here are some other resources that will probably prove useful as you get deeper into projects like this.

Resources:

Mathworks' Demos & Webinars:
http://www.mathworks.com/products/matlab/demos.html

(The first professor who taught me Matlab had me go over all of the demos, and it was pretty useful. I'd recommend perusing "Matlab Overview" at least - you can ignore the stuff about C. "Getting Started" is very useful if you have the time.)

Mathworks' Matlab Support Page:

http://www.mathworks.com/support/product/product.html?product=ML

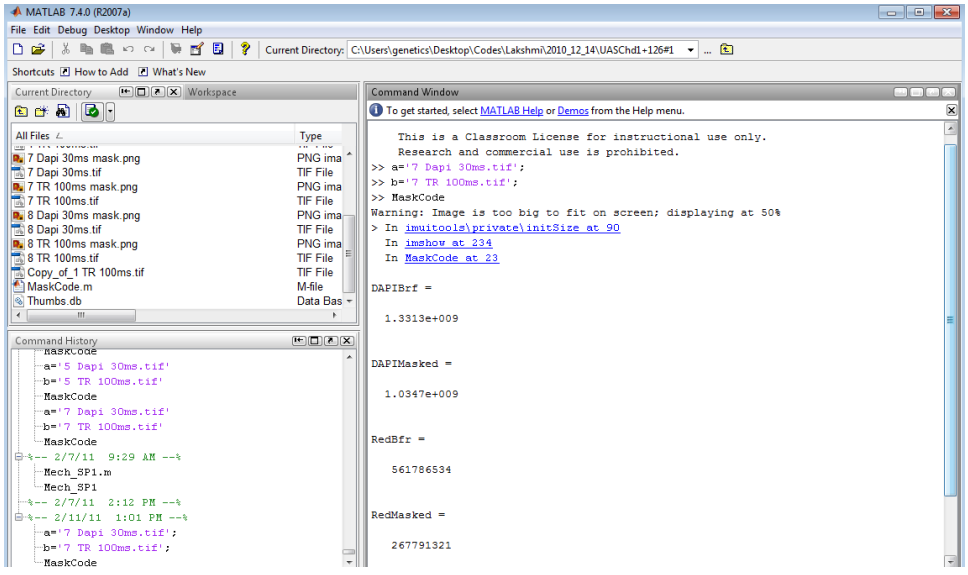
Steve's Image Processing Blog: http://blogs.mathworks.com/steve/

There are several profs who are familiar with Matlab. Prof. Higdon teaches Phys 100, Computational Biology, using Matlab. As far as I recall there is no image processing in the class, but he has a firm grounding in Matlab and can help you with whatever questions you have. Prof. Landsberg also uses Matlab fairly often and you can consult him if you have questions.

Basics:

This is what an intro CS class is for. There's no way I can impart all of this on you guys - there are some things that just need practice (aka lots of assignments and labs) and instruction. Things like learning how to code efficiently and elegantly will take time. They are good to learn, but I won't be concentrating on them here.

- Matlab Layout



The Command Window is where you'll be doing most of your work. Use this to input things, call functions, etc. Similar to the main window in

Maple.

Previously executed commands are saved in the Command History window. All current variables are displayed in the Workspace and the current folder you're working in is shown in the Current Directory. If you are opening any images or using any m-files (see below) they have to be in your Current Directory. THere are some other windows but I think that those are the only ones you'll really need.

- Functions and m-files

  In Matlab, you can work purely in the Command Window (like Maple, you've just got that one main window and you do your calculations in there) or also make your own programs/algorithms in separate files which you can call in the Command Window or in other m-files.

- Commenting

  Commenting is super important! Using a % will signal Matlab to ignore everything to the right of the percent sign, and thus you can write little notes about what a certain line or section of code does, make notes to go back and check something, etc.

  Documenting your code well with comments is critical. When you go back to old codes it will be much easier to use and edit, as you can remember what you were thinking when you wrote it, and how some less-often-used functions work. Also when you share codes with other people it helps orient them and is convenient.

  Going along with that, I often write a section of comments at the top of a code, documenting who wrote it, when, and what it does (just a sentence or two). I often also note what the required inputs are, and what the outputs will be. This is also great for people who do not code, as they can look at it and at least understand what it should do, even if they don't understand coding.

  Tip: Use {% and }%, on separate lines, to frame a series of lines of code (else you have to write % at the beginning of each line). This is also convenient for commenting out whole sections of code that you don't want to just delete as you may want to use it later.

Aside: Also, to make things clearer for yourself and others, label your variables with useful names. Use 'mask' or 'totalLength' rather than 'm' and 'L'.

- For, if, and while, loops

  For loops will take a variable and run through each line of the loop over and over again, setting the variable to a different value each time.

```
list = [] %an empty set
for i = 0:10 %the : means every value between the range of 0 and 10.
%1:.1:2 would use steps of .1 in the range 1 to 2
    list = [list,i]
end
list
```

  Should output

```
list = [0,1,2,3,4,5,6,7,8,9,10]
```

  If loops will go into effect only if a certain condition is met.

```
list = [1,2,3,4,1]
for column=1:5
   if list(1,column) <= 2
      list(1,column) = 10
   else
      list(1,column) = 5  %having an else clause is not necessary
   end
end
list
```

  yields

```
list = [10,10,5,5,10]
```

  While loops are sort of like repeated if loops. It may take you some time to realize the usefulness of while versus if loops, but they can definitely

be advantageous in different situations. If you have a particular liking for one over the other they can often be used interchangably. Placement is often critical in which you choose to use.

- Matrices and arrays

  Matrices are set using square brackets. Elements in the same row are separated by commas and columns separated by semi colons.

  Arrays are 1-dimensional matrices.

- Editing figures and graphs

  Instead of coding everything up, you can manipulate an image using the Plot Tools, which you can open from any figure window (rightmost option in the toolbar at the top of the figure).

Specific Functions:

- help/doc/lookfor

  This will be useful in accessing and searching Matlab's help files.

  ```
  >> help(function_name)
  ```

  ```
  >> lookfor im
  ```

  This will result in a list of built-in functions Matlab has that contain the string 'im' in the name. There's no use reinventing the wheel; there are many things that we could spend weeks coding that someone else has already programmed Matlab to do.

- imread, imshow

  Stands for "image read" and "image show".To display or work with an image in Matlab, the program has to first read the image, then it can work on it. After reading the image, Matlab can display it to you (*ex.*

  ```
  imshow(image.tif)
  ```

) or perform a variety of operations on it, as though it were a matrix.

Black and white images are read as matrices with values from 0 (black) to $2^{bitnumber}$. For instance, a 16-bit b&w image has values from 0 (black) to 65,536 (white).

- imagesc

  like imshow, but instead of showing the image as is it uses a scaled map - from red to blue. It looks at the range of values in the picture and assigns them to the spectrum of the map rather than black to white.

  Using this helps us see how evenly distributed the values are and differentiate features, especially as the black and white images can sometimes be very dim.

- plot

  When plotting some 2-dimensional graph,

```
p = plot(x,y,LineSpec,'PropertyName', PropertyValue)
xlabel('X axis')
ylabel('Y axis')
title('My Figure')
```

- figure(#), hold on/off

  When you plot multiple things, it's good to specify how they should appear and how many figures and windows there should be. To make sure that each image shows up in a separate window use figure(#). Otherwise the image may replace the last one on the same window, making it hard to compare them. Also it becomes messy when you are manipulating many images at the same time.

  See also: plotting more than one image in a window ().

  To make sure that you are imposing images on the same figure (say you want to draw a line on a previous figure, or put on a false color mask) use 'hold on' to stack images on the same figure, and turn it off with 'hold off'.

- find

Will run through the matrix and find all elements equal to the value. The result is a matrix listing the locations of the desired value within the matrix. If you don't define a number, Matlab gives you back the locations of all nonzero values.

```
>> find(list = 1)
```

- threshold

  Takes all of the values under a certain number to 0.

- size/length

```
>>size(list}
ans = [5 1]
```

  Thus the matrix 'list' is a 1×5 matrix. When I'm going for length I often use

```
max(size(list))
```

- Sum

  This function adds all of the values of an array together, or all of the values of a matrix's columns together.

```
>> A = [1,1,1,1;2,3,4,5;6,7,8,9];
>>sum(A)
ans = [9,11,13,15]

>>sum(sum(A))
ans = [48]
```

  I have to check something about cumulative sums. Will get back to you about it.

- Creating logical matrices

  Logical matrices are just matrices with only binary values - everything is either 0 or 1 (true or false does the same thing). Also known as a Boolean

matrix. It makes things easier, in a way - there are many functions that logical matrices have that you can manipulate.

In Matlab and many programming languages, a matrix of 0s and 1s is not necessarily a logical matrix, and can't do everything a logical matrix can. You will have to convert it.

zeros(rows,columns)

ones(rows,colums) makes a logical matrix of all ones.

You can change the values of specific elements by setting them equal to true or false.

```
>>A = zeros(1,2);
>> A(1,1) =
ans = 0
>> A(1,1) = true;
>>A
A = [1 0]
```

- saveas

  Automatically save your generated matrices or graphs as images or .csv files.

- Watershedding

  These next few things I'm still messing with.

- labeling

  Still trying to perfect this. Will get back to you.

Random other things: I've picked some of these things up here and there, and I think these are some useful little functions or tips.

- Diary

  This is a useful function when you're building an m-file and testing out functions.

```
>> a = 'file.tif';
>> diary on

>> imread(a);
>> imshw(a);
>> imshow(a);
>> diary off
```

A m-file will then appear that has every command that you've typed between turning the diary on and off. Nice for when you're experimenting a lot, or typing up an algorithm for the first time, and don't want to rewrite everything into an m-file.

You can do the same thing by selecting previously executed commands in the Command HIstory window and choosing "Make m-file" in the right-click options.

- Double %

  Separating code with a double comment breaks the code into pieces that you can compile separately, to test individual aspects of the code. Makes dealing with large m-files easier. Also if your code takes a bit of time to run, instead of having to rerun the entire thing every time you want to test it, you can focus on specific lines of code.

```
%%Opening the files
threshold = 1000; %Set the threshold here
q = imread(a); %DAPI picture
p = imread(b); %Chd1 picture
figure(1), imshow(q)
mask = true(size(q)); %creates a matrix of 1s

%%Seting up the mask
m = find(q<=threshold);
mask(m) = false;
mask2 = zeros(size(mask));
mask2(mask) = 0.1; %transparency
```

```
%%Make the false color mask
green = cat(3,zeros(size(q)),ones(size(q)),zeros(size(q)));
hold on
h = imshow(green);
hold off
set(h, 'AlphaData', mask2);

%%
```

- Adding counters

  A good way to check if certain parts of your code are working correctly, or if the code is getting stuck anywhere.

  Choose a random letter and make a variable that does something irrelevant to the rest of the code and is printed.

```
for column=0:max(size(list))
   if list(1,column) = 0
      somefunction(something)
      q = 1 %do not suppress
   end
end
```

  Say I input:

```
>> list = [1,2,0,4,0]
```

  and I get some error, or $q = 1$ is not outputted, I know that something is wrong. Specifically, the counter will tell you whether or not that part of the code was read by Matlab. If no $q = 1$ pops out of Matlab, then obviously the problem with the code is before the placement of the counter.

- Indentation

  Indentation is pretty important, especially in Matlab. It will help you organize, and see what is nested in what. Also helps keep your code cleaner and more elegant.

  ctrl+i will automatically indent your code.

  When dealing with

- Creating colors:

  In Matlab, colored images are represented using pixels that have matrix values of [red, green, blue] randing from 0 to 255.

  I've made the false color masks using only green, set slightly transparent so that the chromosome shines though from underneath. The mask shows up best with yellow, actually.

  Yellow = [255,255,0]

  Purple = [255,0,167]

  RGB Color Charts are all over the internet; search for them to mess around with colors.

That's all I can think of for the moment. Plus this is getting really long, which isn't the point of it, so I'll stop here and make more edits later, as necessary. Give me any comments you can think of.

In my lab notebook, I wrote pretty thorough notes while writing code and examining how to manipulate with certain images in different ways. I wrote down a bunch of functions I was investigating and ideas I have for ways to approach the problem, and how I persued that. It will be more helpful for the various specific projects than this broad overview, but those pages also assume a certain familiarity with Matlab and writing code.