Lab Assignment 1 - CSE 377/591, Fall 2011

Due: Wednesday, October 12, 2011, 11:59pm

This lab is meant to (i) introduce you to matlab, if you did not know it yet, and (ii) implement routines for some of the functionality you will need later in the course. You do not need to develop a GUI (Graphical User Interface), but if you want to, you can. Submit all your work in a zip file on blackboard. Matlab is available in the CS department's UG and G labs, and in the University sinc sites. Please let me know if you have trouble accessing it. Note, please start early, as matlab takes some time to get used to. For the next lab assignment you will then have "warmed up" to this computing environment and you will be able to do more involved tasks quite comfortably.

What you need to submit:
a) All .m files
b) A report that shows, for each question, the respective matlab code, the appropriate output (numbers, plots, images, spectra), and a narration of these (that is, a discussion of your solution and your findings, and any observations you may have made). Any questions asked in the text below should also be answered in the report. This report will form the basis for grading. Be professional about it.

First have a look at the matlab tutorials discussed in class and linked to on the class website to get an introduction to matlab. Try a few of the examples, and see if you understand the answers. There are a lot more tutorials on the web, if you have doubts. Finally, Matlab also has help facilities. Use them!

Make sure you understand the difference between a row vector, a column vector, and a 2D matrix. Also understand how you can create an input vector and use it to compute an output vector. For example, try to create a vector x = [0 1 2 3 ... 100] and compute another vector y from it that contains the squares of the x-elements. Here, make sure you understand the important meaning of the '.' when you generate y = x .^2. Make sure you understand when the '.' is used and when not. Check that this also works for 2D matrices.

Also, learn how you can graph the vectors and matrices you produced above. This should produce a graph of the function $y=x^2$. Make sure you can do this.

Finally, locate a few images on the web you find interesting under the goal of image processing. Make sure to include some with medical imaging content, such as radiographs (X-ray images), CT slices, MRI images, or even ultrasound. You could use the google image search engine. But get the original high-res image, not the thumbnail stored at google. Right-click on the image in your browser and choose "save as". Note, for the following capabilities, your matlab version must have the image processing toolbox installed. To read an image, use *my_img=imread(filename)*. Note, the filen ame must appear in single quotes, such as *imread('some_image.jpg')*. You can read in images of any format -- matlab will know how to conevrt it. Then convert it to grayscale with *myGrayImg=rgb2gray(my_img)*. Finally display it with *imshow(myGrayImg)*. Now you are ready to take on the following exercises, each of which you need to document in your report. See also the section 'Expected results' at the end of this assignment.

Create functions for your matlab code. On the command line you first create and then go to your assignment directory where you want to save all your files. You need to change the current working directory in matlab. There is a box for this on the top bar of the application. To bring up the matlab editor type 'edit <filename.m>

1. Write two Matlab functions, *function* **[*sum*]=mySum(a)** and *function* **[*avg*]=myAvg(a)** (note, in matlab the function outputs, here *sum* and *avg*, are entered into [ ]) that compute the sum and the average value, respectively, for the elements of a vector **v** and a matrix **m**. You could just use the matlab functions sum and avg for this (inside your own function). Create the vector as described above.

2. Write a function **simpleBox(L,H)** where L is the width of the 1D box and H is its value. This is a vector of length L which each element set to H. You can create a constant y-vector of the same length than another vector x-vector using the command *ones(size(x))*. Then you multiply this vector of all 1's by your chosen value H. Graph that box in an x-y plot (using the matlab *plot()* utility -- in this context also check out the *subplot()* utility which allows you to place several plots into a more manageable plot matrix). In this case x-vector is the driving vector and y-vector is the data vector. If you want a true box (meaning H values surrounded by zeros on each side), you can edit the y-vector by assigning parts of it to zero, such as *y(1:10)=0*. So in this case you would make the x-vec-

tor longer than L to make room for the zeros. A good strategy is to put the same number of zeros on each size of the box to make it centered.

3. Write a function **simpleGauss(sigma)** that returns a 1D Gaussian filter (a row vector) for a given value of sigma. To generate this vector, create a suitable x-vector that covers the negative and positive range of the gaussian, such as x=[-3*sigma : increment : 3*sigma]. This will always be an odd-length vector. Using x, each value of the filter can then be computed from the Gaussian function, exp(- x^2/(2*sigma^2))). You could use the exp() function in matlab: g = exp(-(x.^2) / (2*(sigma^2))). Note the '.' after the 'x', which will make sure that each element of x is transformed. We shall call this vector the g-vector. The parameter *'increment'* in the x-vector should be set to 0.1, or even lower, for plotting (this creates a smooth curve). Graph both g-vectors (the one with increment set to 0.1 and set to 1) in an x-y-plot.

4. The above formula for the Gaussian ignores a constant factor. Modify the previous functions to a function **gaussNorm(sigma)** which normalizes the values in the filter so they sum to 1. Hint: compute the sum of the elements of the g-vector and divide the g-vector (elements) by it (recall the '.'). Also write a function **boxNorm(L,H)** for the box with similar functionality. Graph the results in x-y plots. In order to prove that the new vector is normalized, its sum should be 1.

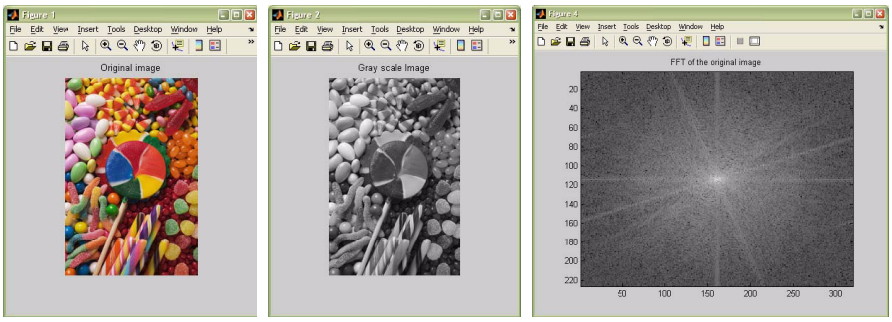
5. Now display some (grey-level) images. For this, write a function **showimg(filename)**. Use the matlab function *m=imread('<filename>')* to read the image into a 2D matrix, here called m. If the image is in color, convert it to a greyscale float image using the *mgray=rgb2gray(m)* and display it using *imshow(mgray)*.

For display you can also use the function i*magesc(m)*. It has the added benefit that it will normalize the image before display, that is, the array will always appear at maximum contrast. But you need a few extra statements. First you type *colormap(gray)* to make sure the image is displayed in greyscale (if is not already). Then you use *imagesc()* as follows: *imagesc(m); axis equal; axis tight;* This makes sure the image does appear unstretched.

6. Let's do some Fourier transforms. Plot the (1D) Fourier spectra for the gaussian and the box with various sigma and L, respectively. For the transforms use the functions with the 0.1 or 0.01 increments to get smooth curves for the spectra. Create a function *myFFT()* for this, which calls the matlab function *fft()*. Use the matlab function *fftshift()* to center the frequency spectrum. Use *abs()* to get the magnitude, since the FFT produces a complex number.

For the box you should see a *sinc*-function develop. The wider you make the box, the more lobes you will see. For example, if you have a box of size 41 (in a sufficiently large vector of zeros, say 512), then you should see 41 lobes, falling off in a sinc-function on each side. What happens to the width of the main lobe as you increase the box size, and what effect does this have if you used this box for convolution? Also, how tall is the center lobe, and is there a difference for the simple box and the normalized one? What effect does the normalization have, and what effect does the box height have in the unnormalized case? Finally, are there any such patterns for the Gaussian filter?

7. Now find the (2D) Fourier transforms of the (grey-scale) images. You can use matlab's *fft2()* (2D Fast Fourier Transform) function for this. Then, use *fftshift()* to put the origin of the computed spectrum (the zero-frequency band, the DC component, the average term) into the center of the plot. Following, use the *abs()* function to compute the magnitude of each (complex) frequency term before plotting. Using *log()* will bring out smaller values better, or alternatively you could bracket the values using specialized parameters in the display command (see the matlab help files). Use the routine *imagesc()* for this display (see also item 5 above) since your value range will be outside [0, 255]. In your report put the spectra images next to the corresponding spatial images.



A color image (left), converted to greyscale (center), and the Fourier spectrum of this grey-scale image (right).