Matlab Lecture 5: Basics of Image Processing

Last Week in Review

* 30 Review previous day's homework. Type the code onscreen, discuss each part of the problem, and explore other methods for reaching the same answer.

Practical Script Workflow: Thinking in Matlab

* The 7 Common Sections of a Typical Analysis Script:
  1. Help comment: Includes a Statement of Purpose and Authorship
  2. Workspace Housekeeping: **clc**, **clear all**
  3. Set Paths: **addpath()** for necessary toolboxes, and define path strings for things like your raw data directory (where you read/load data from), your processed data directory (where you save data to), and your figures directory (where you save figures to)
     + It’s a bad habit to mix different types of files in the same directory. Even when it’s a small project, you can accidentally delete or modify data and slow your work down when using only one path!
  4. Import Data and Set Constants: by defining everything (yes, everything!) near the top of your script, you make your script much easier to use and modify.
     + Effect: Almost all of the “numbers” will be found at the top of the script--almost everything else should be “words”---referring to variable names and functions.
  5. Various Analysis Sections: Make a major section for each big idea.
  6. Plot Data
  7. Save/Output Data

Random Information

* Throughout the class, there are some things that we may have accidentally left out. Although it would have been better to have learned them at the same time as the other related information, we still want you to have that information.
* Browsing the Documentation: Everything we’ve taught in the class (and more!) can be found from the main documentation page and clicking the Matlab topic hyperlink.
* Keyboard Shortcuts
  + **F5**: Runs a script (Mac: Command-R)
  + **Highlight text and press F9**: Runs the highlighted text
  + **Ctrl-I**: Automatically indents code.
  + **Ctrl-C**: Stops Matlab. This is essential if a script is taking longer than expected, or if your script is caught in an “Infinite Loop”.
* Generating Code from a Figure
  + In the figure editor window, click File → Generate Code. This will make a script that has the commands needed to make the figure you made, complete with all changes in the property editor.
* Subplots: You can plot multiple figures in a window (technically, it is considered to be plotting multiple “axes” within a “figure”) using the **subplot()** function.
* Common Tricks
  + Incrementing Numbers: **x = x + 1**
  + Reverse an Array: **reversedArray = array(end:-1:1)**
  + Convert Numbers to Strings and vice-versa: **num2str()**, and **str2num()**
    - Often used for putting new numbers into file names.
  + Extending an Array (2 ways):
    - **myArray = [myArray myScaler]**
      * This method is the slower of the two, but it works in more situations
    - **myArray(end+1) = myScaler**
  + “Vectorizing” Your Code: Matlab is much faster when you do operations on the entire array or matrix, rather than using “for loops”. When your code is running slowly, ask yourself whether you can get rid of a for loop or two!
  + Timing sections of your code: **tic** and **toc** functions. **toc** outputs the amount of time since **tic** was called.

Matrices with more than 2 dimensions

* Multi-dimensional matrices can be useful for handling data of many types of experiments
  + multivariate time series with many trials/blocks
    - row - time/sample points
    - columns - variable type
    - 3rd dimension - Trial/Block number

The Book Analogy

* The 2 dimensional matrix is like a page in a book: rows and columns
* The 3 dimensional matrix is like a bunch pages bound together in a book
* The 4 dimensional matrix is like a bunch of books on a shelf
* The 5 dimensional matrix is like a bunch of shelves in a book case **. . .**

Operations with Multi-Dimensional Matrices

* Concatenation is when you combine multiple matrices to create a larger matrix
  + Note: before we showed some examples where we used the square brackets ‘[ ]’ to concatenate an array to itself.
  + The function **cat** can be used in the exact same way but now you can choose which dimension to concatenate. This is very useful for N-dimensional matrices since through the square brackets only the first two dimensions can be concatenated.
  + Class Exercise:
    - Create a 5 by 5 magic matrix and a 5 by 5 matrix of ones.
    - Concatenate the two in the 1st dimension using the **cat** function
    - Concatenate the two in the 2st dimension using the **cat** function
* Replication of matrices is when you copy a template matrix and tile it in a set of dimensions
  + The function **repmat** will copy a matrix and concatenate it any number of time in the specified dimensions.
  + Specify how many copies should be included in each dimension (i.e. repmat(myMat,1,2) or repmat(myMat,[1,2])
  + Class Exercise:
    - Create a 3 by 3 magic matrix
    - Replicate it along the first dimension to create a 6 by 3 matrix
    - Replicate it along the second dimension to create a 3 by 6 matrix
    - Replicate it along both dimensions to create a 6 by 6 matrix

Opening Image Files in Matlab

* Opening an Image file in matlab is a two step process.
  + Put the image in matlab’s path
  + Open the file with the function **imread**. (i.e. imdata = imread(‘myImage.jpg’) ; )
  + Try loading the test file into a variable named imdata
* The default matrix returned by the **imread** function will be an N by M by 3 matrix
  + The first two dimensions are the image face: rows (bottom to top), and columns (left to right)
  + The third dimension are the color values
    - imdata( : , : , 1 ) = red intensity
    - imdata( : , : , 2 ) = green intensity
    - imdata( : , : , 3 ) = blue intensity

Display an Image in a Figure

* The functions **image** and **imagesc** are both used to display a matrix in a figure
* Class Exercise:
  + Plot the image 'ngc6543a.jpg'. This image comes with matlab.
  + Tile the image so you can see 4 copies using **repmat**
  + Use indexing to create an Andy Warhol style image
    - Change the upper right copy to be red
    - Change the lower left copy to be green
    - Change the lower right copy to be blue

Changing the ColorMap

* A colormap is needed to apply colors to values stored in your image matrix.
* Note: If the size of third dimension of the image matrix is 3, then **image** and **imagesc** will use the 3rd dimension as the color map.
* There are many different colormaps to chose from and the easiest way to choose and apply a color map is to use the **colormapeditor**.
  + Under tools in the menu bar select **Standard Colormaps** to see a list of commonly used colormaps.
  + Class Exercise:
    - Use the **mean** function along the third dimension of your image from the previous exercise to get the average RGB intensity values for the image
    - Plot the averaged image and change the colormap to a gray scale with the **colormapeditor**

Homework 5: Image downsampling function

**Homework:**  Create a function **newImage = imDownsample(filename, n)** that:

1) Loads an image with the file name **filename**.

2) Makes a new image that is smaller than the original, by taking every n’th pixel in the

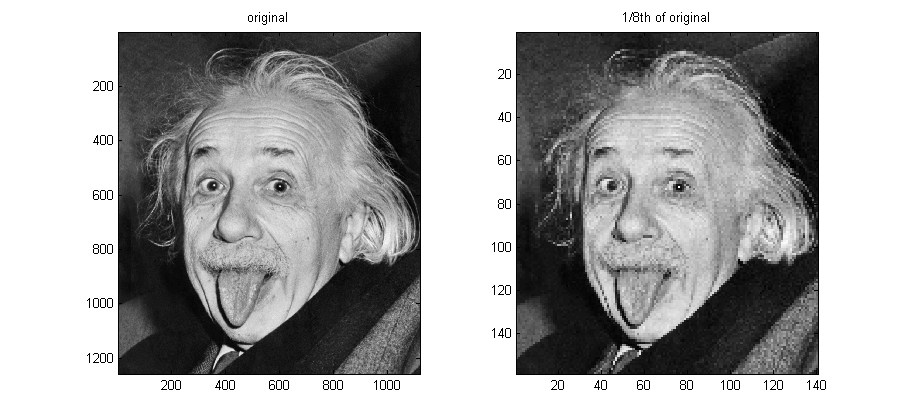
image. So, if **n** is 2, then the image should be ½ the size of the original. If **n** is 3, then

the new image should be ⅓ the size of the original.

3) Plots the new image.

4) Saves the new version of the image.

5) Outputs the new image matrix.



**Extra Credit:** create a second function which has the same functionality but estimates the new pixel values by taking a weighted mean of the surrounding pixels within the original.