CSC 535 – Probabilistic Graphical Models

Matlab Primer

This primer will help those who want to learn Matlab. Nothing should be handed in. You should use judgment about which parts you do. This document, which serves dual purposes, focuses on handling images (beyond what is needed for CSC 535).

The purpose of this handout is to become familiar with Matlab. Matlab is useful for exploring ideas and prototyping programs. It is a popular programming environment for computer vision and machine learning, especially for those who do not have lots of experience in C/C++ because it allows one to focus more on the problem and less on the code. It has significant drawbacks for writing large programs or when performance matters. It is also a ″product″ which needs to be purchased and installed before it can be used. Nonetheless, it is a useful tool which computer vision and machine learning students should be exposed to.

Matlab is available for personal use to UA faculty, staff and students for free. It can be downloaded from http://sitelicense.arizona.edu/mathworks-matlab/ by any University of Arizona faculty, staff or student with a netid. The site-license includes MATLAB, Simulink, plus 48 toolboxes. It is also installed in a number of UA machines and labs. If you do not have convenient access to Matlab, please contact the instructor to discuss options.

Make sure you download the image processing toolbox. If Matlab does not know about the command "`imshow`", then you probably do not have this toolbox.

Note that if you download all of Matlab (instead of picking and choosing your toolboxes), it can take a long time, especially on a slow connection. Be prepared for that!

---

1. Become familiar with Matlab.

   Spend some quality time with a Matlab tutorial (there are many on-line), or perhaps just look at the introductory material that is part of Matlab's extensive help system. To do the later, start up Matlab and hunt for a "Help button" (on Matlab R2016a on my mac there is one in the menu bar and one towards the right hand side of home top bar). From there, you probably need to choose "Documentation". Then you need choose help for Matlab itself (instead of one of the tool boxes and other accompanying products). From there, I suggest exploring "Getting started" and "Language Fundamentals".

   The complete documentation for Matlab is also available on online here:

   http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

   but you might notice that these pages are the same as the ones you look at using the Matlab desktop help browsing system (which might explain why you might find the help system bit clunky --- the information is arranged in a page tree instead of a true narrative).

   Regardless of how you do it, it is a good investment to go through some introductory material first to get an overview of how it works. It will save time in the long run.

   Documentation for Matlab commands is also available from the command line, which is especially useful to remind yourself about the details of a function you know the name of, or a function mentioned in this document. For example, if you want to know how the `svd()` function works, type `help svd`. You can also get help on all the built in operators and even the language itself. Typing `help` gives a list of help topics.

   Tip: You may want to turn on paging (`more on`) for reading help pages.

2. Reading and Displaying Images

Place the JPEG color image linked here:

http://kobus.ca/teaching/cs477/data/tent.jpg

in your working directory.

Load the image into a variable using imread. Type help imread to find out how to do this. Create a new figure using figure and display the image in it using imshow.

Tip: Make sure you put a semicolon at the end of your imread *command. The semicolon at the end of a line prevents Matlab from displaying the result of an expression. Most of the time, you want the semicolon.*

3. Writing Images

   Write the image to a file called out.jpg using the imwrite() function. Use some independent image viewer such as display, xv, or a web browser to verify that this worked. If you are a Linux user (Mac and Windows also, but even more likely you need to install), I recommend learning about the ImageMagick suite of tools for converting, and displaying images (do "man convert", and a "man display" to find out more). Also the program import() can be used to get a screen-shot (at least in Linux; "grab" works best on Mac, and I have no idea about Windows).

4. Scripts, files and paths

   The interactivity of Matlab is great for debugging and experimenting, but often one wants to type code into a file. As you develop code for the deliverables, put them into a file called hw0.m. You can then type hw0 at the Matlab prompt to execute the commands in the file. A file of this sort is known as a script.

   Tip: You might prefer the output if you use format compact.

   Tip: My scripts often have close all *in the beginning, so when I re-run them the old images being displayed go away. Please do this for assignments in this class to make the TA's life easier.*

   Tip: Matlab has pwd, ls, *and* cd *commands that do what you expect.*

   Tip: If your script is in some other directory than pwd, *then you can add that other directory to Matlab's search path with the* addpath *command. The current directory is in Matlab's search path by default.*

5. Functions

   You can define a new Matlab function by simply putting code in a file (as we have just done) and placing a function declaration at the top. Write the following as the first line of your hw0.m file:

   ```
   function [num_questions] = hw0(infile)
   ```

   *Tip: The name of the dot-m file (without the dot-m) and the name of the function should always match. A file can export only one function, but the file may contain internal helper functions. If you use such functions (likely you will want to in future assignments), you will need to end them with "*end*"---see "*help function*" for details. Matlab will read the file to find such internal functions before it starts executing instructions within the file.*

   Modify your code so that the imread command use the file provided by the infile variables instead of the hard-coded "tent.jpg". Also, make it so that every question that you implement (starting with the next one) adds one to a variable num_questions so that this number can be returned by the function. The TA should be able to execute this function by typing hw0('tent.jpg'), or n=hw0('tent.jpg') if they want the number of questions done returned in a variable (The TA will check this).

   *Tip. Notice that Matlab strings are specified using **single** quotes.*

   Tip: If you want to return more than one variable from a function, add it to the list in square brackets in the function's declaration. Simply setting these variables inside the function will cause them to be returned.

Tip: As is common with many languages, functions do not change the values of input parameters (call by value). They can be changed (as copies) within the function, but the version of the variables outside do not change. To get values out, use the output parameters. (For pedants: there are omitted details here that are not relevant to getting started).

Tip: Functions can also have optional input and output arguments. To determine the actual number of input and output arguments a function was called with, look at the special variables nargin and nargout.

6. Documenting Functions

   When you create a new exported function, it should always be documented so that help returns something informative. The convention in Matlab is to place the help message in comments after the function declaration (the comment character is "%").

   For example, you can look at some of the code in the Matlab library. Some of it is implemented as .m files, and some of it is "built-in". Regardless, there should be a .m file for at least the documentation. Using the which command, see if you can find the .m file for your favorite function so far. On my mac, the path provided was not exact, but gave me a good idea where to look.

   If you look at some of these examples, you will see that the first line of the comment contains a one-line description of the command. All subsequent contiguous comment lines are included in the help message. Add a few lines of text about your program in this style, and verify that typing "help hw0" does what you expect. Update the description of what your program does as you work on it.

7. Basic image data structures

   Make sure you understand the data structure being used to represent the image that you have read in. Type whos to get a list of active variables along with their types and sizes. You can see that your image is a 3D array of bytes. This is row by column by "channel" where the channels are red, green, and blue intensity values. Now use the whos() function to show the information for only the variable containing the image.

   The function size() is used to get array dimensions of an array for further processing. Show off your ability to read help files to put the number of rows, columns, and channels, into variables num_rows, num_cols, and num_channels respectively, using only one statement.

   What is the range of values contained in the image array? Use the min and max functions to output this information for red, green, blue and overall, and report it into your writeup.

   *Hint: These commands by default operate on only **one** dimension of their argument. You can convert your image into a 1D arrays (a vector) using the (:) notation before you pass it to min and max. There are many other ways to do this. I encourage you to find at least one other way.*

   Hint for the hint: help colon.

   Convert the image to grayscale using rgb2gray. Check the representation of the image now. Use size or whos to find this out. You will see that we now have a 2D array, or a matrix. Create a new figure and display the black and white image.

8. Image channels

   Create three grayscale images based on our color image by extracting the red, green, and blue pixels, respectively, into an array. Display the three images. Are they what you expect? Can you explain why some areas that are bright in the color image are dark in some of the grayscale "slices"?. Don't forget that while your program needs to display images, they also need to be part of your narrative in your PDF, and some commentary about them should there as well.

Tip: While Matlab has C style loops, you should try to avoid doing this sort of thing with low level loops. Instead, use the colon notion (`help colon`).

Create a new color image that has the green channel from the original image as its red channel, the blue channel from the original as its green, and the red channel from the original as its blue. Create a new figure and display the result.

Hint: Again, you might find the colon notation helpful (`help colon`).

9. Visualizing Matrices

   Convert the grayscale image into a double-precision type, and scale the values so that they lie in the range [0,1].

   Hint: Use the `double` function to do the type conversion, and then divide by the maximum allowable value for a byte, which is 255.

   Create a new figure using `figure()`, and use `imagesc()` to display the grayscale image in it. Why does it look so strange? Type `colorbar` to find out. You can see that 0 maps to blue, 0.5 to green, and 1 to red. This is the default `parula` colormap that is useful for data visualization.. Moving on, for this example, we might prefer the grayscale colormap. Use `colormap(gray)` to do this.

   Tip: Use `help gray` to see what default colormaps are available. Note that you can also make your own colormaps.

   Tip: The imagesc *command takes a second argument that lets you specify the range of values. By default,* imagesc *scales the data to use the full colormap, but this is not always what you want. In this example, we could add the argument* [0 1] *to the* imagesc *command would be a NOP ("no operation") because that is the range we already scaled it to.*

   The image is probably distorted, i.e., the pixels aren't square. Use the `axis` command to fix this, and display a new figure.

   Tip: When you display matrices as images, you usually want the axes to be scaled so that the pixels are square and the image is not distorted. See `help axis` to determine how to do this. You can also turn the display of the axes on and off with the `axis` command. Note that some (all?) axis operations require the figure to have data, so you likely need to apply this after `imagesc()`.

   Finally, display the image using `imshow()`. This should convince you that you will often want to avoid using imagesc(), which is designed to visualize data as images, to display images when that is the object of study. No deliverables here, but this is related to the last bit of the next question.

10. Manipulating Matrices

    For this question, we will work with the converted grayscale image with double values in the range [0,1].

    Tip: Array indices in Matlab start at 1, not at 0 like C.

    Tip: As in standard mathematical notation, the first index of a matrix is the row, and the second index the column. When viewing a matrix as an image, this means that the first index is the y direction going down, and the second the x direction going to the right. As is common with image manipulation tools, the origin is at the top left corner, the positive x axis points right, and the positive y axis points down.

    Use nested `for` loops to set every 5th pixel, horizontally and vertically, to 1. This should set 1/25th of the pixels to white in a square lattice pattern. Create another figure and display this result in it with `imagesc()`, and then do the same with `imshow()`.

Hint: Use the colon operator to define the limits of the for *loops. See the help for* colon *and* for *to see how to do this. Specifically, you want the* minval:interval:maxval *form.*

11. Histograms

    A histogram divides up your data space into boxes, and puts counts of occurrences into them. They are visualizations of the empirical probability distribution of your data (e.g., how likely is it to come across a very red pixel?). If you are not familiar with histograms, you should read the Wikipedia article and/or some other resource about them as we will assume that you know what they are.

    Convert each of the three color channel "slice" images into 1D vectors and provide histograms for them with 20 bins for each of them) using the histogram() function. Again, remember to put this in your document with a caption explaining how the figures are a (limited) representation of the image).

    Tip: You need to give histogram() 1D vectors, and if you give it something else (like a matrix), it does something different. Make sure you give it 1D vectors. This can be done using colons.

12. Plotting

    Explore the plot() command. Plot the sin function over the domain -pi:pi. Use the linspace command to define the domain x and then do plot(x,sin(x)). Use the hold on command to plot another function on the same graph. Do this to add cos(). Use a different color. Even if Matlab does this for you automatically, your reading of the , e.g. plot(x,cos(x),'r'). The running of hw0.m should produce a plot along these lines.

13. Playing with Linear Algebra

    Matlab is a great tool to for experimenting with linear algebra. The next few questions are about doing so.

    Use the fact that inv() inverts a matrix to solve for X=(x,y,z):

    ```
    3*x + 4*y +   z = 9
    2*x -   y + 2*z = 8
    x   +   y -   z = 0
    ```

    Verify that your "solution" works. Make sure that your program outputs the answer, and also the "proof" that it is correct.

    While you should know how to invert a matrix using Matlab, using matrix inversion to solve equations is not the best way. Matlab provides a faster, and potentially more robust, method through the function linsolve(). Check that you get a similar (but not necessarily **exactly** the same answer using that function, and that you can invoke linsolve()implicitly with the "\" operator. To see if linsolve() gives exactly the same answer, subtract the two and report the result. Note that this is not the same as simply observing that they are the same when reported with a few decimal places of accuracy. If it is exactly the same, then the difference should be zero. If it is not, can you explain why some difference is reasonable?.

14. Playing with Linear Algebra II

    If there are more equations than unknowns, then, in the general case, "classically" the problem is over constrained and there is no solution. However, in this course, we will often be assuming that such equations are approximations and have errors due to noise or other reasons, and that an exact solution cannot be found regardless. Thus we will want to find the "best" solution. This is known as solving the equations in the least squares sense. The solution for AX=b, where A has more rows than columns, is given by X=inv(A'*A)*A'*b, where inv(A'*A)*A' is known as the Moore-Penrose inverse of A. (Note that the single quote in this context takes the transpose of what is before it). Use this to solve for X=(x,y,z) in:

    ```
    3.0*x + 4.0*y +  1.0*z = 9
    3.1*x + 2.9*y +  0.9*z = 9
    2.0*x - 1.0*y +  2.0*z = 8
    2.1*x - 1.1*y +  2.0*z = 8
    1.0*x + 1.0*y -  1.0*z = 0
    1.1*x + 1.0*y -  0.9*z = 0
    ```

Your program should output the solution and the magnitude of the error vector.

15. Playing with Linear Algebra III

    Recall that an eigenvector of a matrix A is a vector v, so that Av=kv, for some scalar constant k. If A is real and symmetric, then A has real eigenvalues and eigenvectors. Note that for a random matrix, R, R*R' is symmetric. (Try it!). The Matlab function eig() gives you eigenvectors and eigenvalues. Use these hints to create a 4x4 matrix A, and a corresponding vector v, that satisfies the eigenvector equation Av=kv. Show that your A and v have this relation by printing out the value of A*v./v.

16. Matrix manipulations without explicit loops**.**

    For this problem, we will use the same converted grayscale image with double values in the range [0,1] that we used in problem 10. Similar to problem 10, set every 5th pixel, horizontally and vertically, to 0 (instead of 1), so that 1/25th of the pixels are black in a square lattice pattern. However, do it this time without using any for loops in the code. This can be done in one, relatively short, statement. Create yet another figure and display this result in it.

    Hint: You can index arrays in Matlab with vectors as well as with scalars, so im(Y,X)=0 will set multiple entries of im to zero when either X or Y are vectors, such as the vectors returned by the colon operator.

    Hint on hint: What is described in the hint might take some getting used to. Like many things in Matlab, it is easiest to combine reading of the documentation with experimentation. It is helpful to try simple things on random matrices. To get a random square matrix of size 10x10 use rand(10), and to get one of size 4x12 use rand(4,12). A specific vector can be created by v=[2 4 6 8] (and a matrix by m=[1 2; 3, 4]). Given such tools in an interactive environment makes it easy to create vectors with integer values and see what happens when you use them as matrix indices in assignment statements.

    Now set all the pixels whose values are greater than 0.5 to zero without resorting to loops. It is possible to do this in one statement. The find()function is likely your best bet here. Create a new figure and display the result. You should understand how this works.

    *Hint: An expression like (im>0.5) evaluates to a Boolean matrix, which is true where the condition holds. The find function returns a vector containing the indices of the true values. This vector can be used to index the image. You might ask how does this work when the matrix is 2D and indexes are arranged as 1D? Matlab lets you treat matrices as 1D vectors too, linearizing the matrix in **column-major** order.*

    *More on **column-major** order. Think for a moment how 2D arrays are stored in memory. There are a number of options. One options is that the array is stored as a linear sequence of numbers, i.e., a 1D vector. This still leaves two alternatives. One is that the first row is followed by the second row (row-major), which is how C/C++ handle fixed arrays. The second is that the first column is stored in order, followed by the second column, and so on. This is **column-major** order which is used by Fortran and inherited by Matlab. This is an important issue to understand for those that might want to call Fortran routines from C which is a useful skill.*

    Note: It is better to avoid loops where possible as Matlab is usually used as an interpreted language (you can compile it, but at this point I think you should consider using C/C++). If we use the built in matrix manipulations and functions to do the expensive work, Matlab can be reasonably fast.

More tricks for those that are interested (no deliverables)

Matlab makes it easy to write "vectorized" expressions without having to write for loops or if statements. For example, this will add all the values of the image:

```
sum(im(:))
```

The following will count the number of values greater than 0.9:

```
numel(find(im>0.9))
```

So will this:

```
sum(sum(im>0.9))
```

This will halve only those values greater than 0.9 (note the use of the **.*** *operator to do element-size multiplication of matrices):*

```
im = im - 0.5*im.*(im>0.9);
```

And so will this:

```
mask = (im>0.9);
im = im.*~mask + im.*mask*0.5;
```

This will set 100 unique random pixels to zero:

```
p = randperm(numel(im));
im(p(1:100)) = 0;
```

See `help elmat` *for a list of interesting matrix manipulation and creation routines.*