cs24 Lab: Introduction to Matlab

Out: Jan. 23rd 2003

1 Foreword

This handout is a crash introduction to Matlab. It provides basic knowledge that you will need in order to complete future Matlab assignments. Recommended alternative (or additional) sources are:

- “A Practical Introduction to Matlab”, Mark Gockenbach
  (http://www.cs.unb.ca/courses/cs3113/Gocken/intro.html)
- “MATLAB Primer”, Kermit Sigmon
  (http://math.ucsd.edu/ driver/21d-s99/matlab-primer.html)
- “The Matlab Manuals”, Mathworks
  (http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml)

2 Getting Started

Copy this intro into your 224 directory and run Matlab:

$ cd ~/course/cs224
$ cp -r /course/cs224/asgn/matlab_tutorial/ .
$ cd matlab_tutorial
$ matlab &

If you dislike the fancy (slow) GUI you can run Matlab in your shell by doing the following:

$ matlab -nojvm

The directory you are in when you run Matlab is important because Matlab will automatically load any .m files that are in your current directory. Still, if you started from the wrong directory, no problem: Matlab will execute Unix commands (like cd or ls).

To quit Matlab, type:

>> quit

The intro.m file that you copied over contains all of the introductory matlab commands that we suggest you run. Some people may learn the concepts more quickly by typing in the commands themselves but if you would rather just look at the output than you can type the following in the matlab command window:

>> intro
3 Help!

Always remember your best source for Matlab help is not online tutorials, not the Matlab manuals, but the two following modest scripts:

\[ \texttt{help <function_name>} \]

and

\[ \texttt{lookfor <keyword>} \]

Try typing in “help sum” and “lookfor logarithm”. The lookfor command may take around 10 seconds to finish.

4 Matrix Algebra

Matlab was especially designed for matrix computations (MATrix LABoratory). In some 224 assignments we will be processing images (and images are matrices, after all), so this section should be particularly useful. This is how you define a matrix:

\[ \texttt{A = \begin{bmatrix} 1 & 2 & 3; 4 & 5 & 6 \end{bmatrix}} \]

As you see, a semicolon separates rows; space (or comma) separates elements on the same row. Square brackets go around the matrix. Here comes another matrix

\[ \texttt{B = \begin{bmatrix} 2 & 3; 3 & 4 \end{bmatrix}} \]

Let’s multiply them:

\[ \texttt{C = B*A} \]

Here’s a good time to see what other operators are available. Type:

\[ \texttt{help *} \]

Good place to notice that \( .^2 \) and \(^2 \) do different things. This one you’d better remember, it may come in handy when computing norms:

\[ \texttt{B^2} \]

\[ \texttt{B.^2} \]

Again, let’s look at \( A, B, \) and \( C \):

\[ \texttt{A} \]

\[ \texttt{B} \]

\[ \texttt{C} \]

Let’s concatenate (horizontally) \( C \) with \( B \) (try “help horzcat”):

\[ \texttt{CB = [C B]} \]

Let’s concatenate (vertically) \( A \) with \( C \) (try “help vertcat”):

\[ \texttt{AC = [A; C]} \]
And this is how you transpose a matrix (put a quote after the matrix):

\[ AC_t = AC' \]

Let’s look at A again:

\[ A \]

Here is how you access the elements of a matrix (mind you, Matlab will accept positive indices only):

\[ el = A(1,2) \]

Here is how you select a whole row or a whole column:

\[ row_1 = A(1,:) \]
\[ col_2 = A(:,2) \]

Here is how you select columns 2 through 4:

\[ cols_234 = CB(:,2:4) \]

Here is how you find the dimension of a matrix:

\[ mysize = size(A) \]

Here is how you make a 3x4 matrix of zeros:

\[ my_zero = zeros(3,4) \]

B.t.w., here is how you make a matrix of ones or the identity matrix (not that you’ll need them):

\[ my_one = ones(3,4) \]
\[ my_id = eye(3) \]

Pretty useful: reshaping a matrix into a vector

\[ my_A_vec = \text{reshape}(A,1, \text{mysize}(1) \times \text{mysize}(2)) \]

### 5 Other Useful Functions

Just run “help” on each of the following:

- sum, cumsum,
- std, mean,
- min, max,
- disp, rand, clear,
- floor, ceil

For example, to find the standard deviation of a group of numbers you can do the following:

\[ \text{std}(\text{my}_A_{\text{vec}}) \]

(see? reshape is useful!)
6 Plots, Histograms, Images

To make a 2D graph you can use the `plot` command. The plot command takes a vector of X coordinates and a vector of Y coordinates. Type “help plot” for more graphical options.

Matlab will automatically clip the axes to crop the plot data. The `axis` command gives you some control over the axes.

```
>> inputs = [ 0.2 0.3 0.4 0.5];
>> outputs = [ 300 500 100 900];
>> plot(inputs, outputs);
>> axis([ 0 1 0 1000]);
```

By default, all graphic outputs will go to the current graphics window (default is 1). To make another window, use the `figure` command. For example, type:

```
>> figure(2);
```

To make a histogram in the new window:

```
>> samples = rand( 1, 1000);
>> hist(samples, 20);
>> axis([ 0 1 0 1000]);
```

To display a (randomly generated, in our case) image, try:

```
>> figure(3);
>> im = rand(300,400);
>> imagesc(im);
>> colormap(gray);
```

Matlab will attempt to display images in color. If you want to display greyscale images you can trick Matlab by using a colormap that goes from black to white (see `colormap` example above). You may find this useful in the tone mapping assignment.

Finally, to see how to load an image file in Matlab, try “help imread”.

7 Basic Matlab Programming

You can actually glue Matlab expressions together in a function and put them in a .m file. Remember to use the same name for the function and the .m file.

I’m afraid all you need to know for now is how to write a “for” loop and an “if” statement. Run “help for” and “help if”. (Also take a look at the examples in the `calc_fibb.m` and `fibb.m` files.)

There are two short Matlab examples of functions in this directory:

- `calc_fibb.m` returns a vector containing the first n Fibonacci numbers, plus their mean, plus their standard deviation.
- `fibb.m` calls the above function and displays the numbers in the vector, their mean and std.

For example, run

```
>> fibb(10)
```
Now look at the .m code. Yes, it is that easy. The first line of a function is always:

```matlab
function <return_values> = <function_name>(<arguments>)
```

Everything that starts with a % is a comment (hey, try a “help fibb”). Add a ; at the end of each instruction if you do not want Matlab to echo it. That’s all.

One thing to be careful of is that when indexing into vectors and matrices within Matlab the first element is at index 1 not index 0.

If you try to pass a function as a parameter you will probably run into trouble. What you need to do is prepend the '@' sign to create a “function handle”:

```matlab
>> sample(@sample_func, 40);
```

Now in the above example if you want to invoke sample_func while inside of the sample procedure you can use the function “feval”:

```matlab
>> feval(sample_func, input);
```

One final comment on programming in Matlab: in general, avoid writing “for”-s and “if”-s whenever you can. Matlab’s strength lies in its ability to perform efficiently matrix and vector computations. For example, min( A ) will be much faster than looping and comparing every element of the matrix A to find its minimum value. We recommend the following exercise:

Write a Matlab script to multiply two matrices (use “for” loops). Run it on two random 100 x 100 matrices A and B. How long did it take? Now type:

```matlab
>> A*B
```

Point made, we hope.