CA641 Biometrics
Notes for Lab #1, 09 February, 2004

John McKenna
john@computing.dcu.ie

1. Introduction

Each week you'll have a set of problems to work through. You won't be spoon-fed and I'll be putting a lot of faith in your intelligence, common sense, resourcefulness, and most importantly, your desire to learn and do. I deliberately leave gaps in the information I hand out, but the ingredients I've just listed should go a long way to filling those gaps. But inevitably, it will occasionally mean you have to ask a lab tutor, me, or classmates, questions -- and please remember: *NO QUESTION IS TOO SILLY*. If you have questions that can't wait till the lab session, email me or a tutor to arrange an appointment, or grab me (figuratively speaking) after a lecture. In this module you are expected to learn *proactively*.

You are encouraged to help each other, but that should involve helping yourself too. Avoid the habit of asking someone else *every* time you have difficulty -- try thinking things thru yourself. If you finding yourself explaining something to someone (an excellent way of checking your own understanding of a topic), be sure to make note of any 'grey' areas you are not sure of yourself - and please bring them to my attention, so we can figure things out. I am also always receptive to feedback, so don't hesitate to mail me or come see me with comments.

During the labs, I generally prefer if the printers are not used or are even kept offline, as the noise can affect both production, perception and recording quality.

Regular lab attenders will be given priority by the tutor. I may not attend the labs myself, but I like to drop by from time to time to see how you're coping.

Good luck with this module. I hope you find it interesting and stimulating.

2. This week

This week we have a number of tasks to tackle. It is not necessary to follow the order given below. Realistically, there is enough material below to cover 2 weeks' lab sessions, but remember you are expected to work outside lab time too. Where questions are asked below, you may not yet be equipped to answer them. Such topics/issues will be covered in forthcoming lectures, but that does *not* mean you can't stop and think about the question, or even use resources outside the module to try to answer the questions. So, this week I would like you to:

2.1. Use a speech analysis tool.

2.2. Start using MATLAB.

3. Analysis

3.1. In the `Computational Linguistics` start menu, open `Praat`. You can close the `Praat Picture` window to avoid desktop clutter.

3.2. Read in any speech file. You will find prerecorded speech of a number of informants in my public drive in directory `G\:Public\john\data\VowelsHD1`. Alternatively, you can download speech files from the web.

3.3. Pressing the `Edit` button opens a window which allows you to view your speech waveform. Make sure that you can also view the spectrogram, pitch and formant tracks. By examining the spectrogram, how many distinct segments are there in your speech recording? Familiarise yourself with zooming in and out of parts of the waveform and note the time/playback bars at the base of the window.

3.4. Has your recording been clipped? Has it been subjected to an excessively high level of quantisation noise?

3.5. Press the Formants & LPC button and choose Autocorrelation Method LP Analysis. Choose an analysis order of 14, an analysis width of 40 ms, and a time step of 20 ms and preemphasis from 50 Hz. Then get a spectral slice from the resulting LPC object at a time in the middle of two different segments (i.e. 2 different slices – not a single slice from between two segments). Open the editor on the Spectrum objects. How do the spectra differ. Measure F1, F2 and F3; do they tally with the formant track on the spectrogram?

3.6. Now perform intra- and inter-speaker comparisons of spectra of the same phonemes.

4. MATLAB

4.1. Introduction

These notes just give a brief flavour of the basic things you can do with matrices in MATLAB. They don't cover matrix arithmetic, as there is plenty of info in the MATLAB documentation (use `help` *command* or `lookfor` *keyword*). The tutors or a classmate will show you how to start up MATLAB, the command line, the editor, and how to set paths correctly. Feel free to look for tutorial help on the web.

4.2. Creating and Initialising Matrices

Sometimes it is easier to create a matrix of known size by filling it with zeros before it is actually used:

`m1 = zeros(`*number_of_rows, number_of_columns*`)`

You could initialise it with ones:

`m2 = ones(`*number_of_rows, number_of_columns*`)`

4.2.1. Exercise

4.2.1.1. Create a matrix of zeros with 4 rows and 10 columns.

4.2.1.2. Create a horizontal vector of 4 ones.

4.2.1.3. Create a vertical vector of 6 twos.

4.2.1.4. What is the effect of appending a semicolon to a MATLAB instruction?

4.3. Matrix Dimensions and Manipulation

4.3.1. Use the functions `size, length` on the matrices and vectors you have created, e.g:

```
m1_dimensions = size(m1)
```

4.3.2. We can join matrices/vectors together to make larger ones, but the dimensions must be compatible:

```
hv3 = [hv1, hv2]

vv3 = [vv1; hv2']  % Note the ' after hv2… what does this
do?
```

4.3.3. We can then alter or access elements, rows and columns in the matrix:

`m1(`*row, column*`) =` *value*

```
m1(:,1) = vv3

m1(3,4:6) = 3*ones(1,3)

m1(6,1:2:7) = [-ones(1,3), 99]
```

4.3.4. Exercise

4.3.4.1. Manipulate the matrices you have created. Be inventive.

Note: When you get an error message saying the dimensions of the matrices you are manipulating are not compatible, you should print out the dimensions of the matrices involved. This will speed up debugging.

4.4. Sequential Manipulation Of Matrix Elements

4.4.1. We usually use a single `for` loop to access a row or column element-by-element. If we wish to access both rows and columns in an element-by-element fashion, we can use two `for` loops – one 'nested' within the other, e.g.:

```
function full_matrix = play_matrix(num_of_rows,
num_of_columns)
full_matrix = zeros(num_of_rows, num_of_columns)
for i = 1:num_of_rows
      for j = 1:num_of_columns
               full_matrix (i,j) = i+j
               pause
      end;
end;
```

4.4.2. Exercise

4.4.2.1. Write a function that creates a triangular version of the matrix example above. Call the output `triang_matrix`.

4.4.2.2. Write a function that reverses what you've just done: i.e. given `triang_matrix`, re-create `full_matrix`.

4.5. Speech handling

4.5.1. Use the `wavread` function to read in any `.wav` file that contains speech.

4.5.2. Print out the sampling frequency and bit precision.

4.5.3. Use the `plot` function to plot the waveform against sample number.

4.5.4. Has the signal been clipped?

4.5.5. Invert the waveform and play it using `sound`. Compare to the original.

4.5.6. Plot both waveforms, using different colours, on the same figure. Use `clf` and `hold`.

4.5.7. Plot a subsection of the waveform, e.g. about 4 pitch periods of a vowel.

4.5.8. Can you make a vector which stores all the sample *times*? Plot your signals against time.

4.6. Challenging exercise (only if you want a challenge)

4.6.1. Take a signal of any bit precision, and output a signal of any given bit precision less than that of the input signal.