Matlab for Psychologists: A Tutorial

Table of Contents

Introduction

Matlab is a language, and like all languages, the best way to learn is by using it. This tutorial starts off covering the basic grammar of Matlab, and then moves on to realistic examples and lots of exercises. It may seem slow to get started, but it really is worth persisting because once you know how to use Matlab, you will be able to:

- analyze your data much quicker, more flexibly and with fewer errors than you ever could in Excel
- Use Cogent to carry out experiments
- Generate stimuli - pictures, sounds, movies, according to precise specifications
- Write scripts for SPM so you can analyze your imaging data quickly and efficiently

There are a couple of things to bear in mind before you start. There is no one right way to do anything in Matlab - lots of pieces of code may have the same effect, but as you get better it is worth looking for ways to make your code neat, then it will run quickly and be easy to debug. Starting from scratch on any project is very intimidating and the one of the best way's to start on Matlab is to take scripts that other people have written and adapt them to what you need. Some of the scripts in this tutorial may help, or ask people for their scripts (especially for SPM and Cogent).

Getting started

These notes assume that the reader uses Windows and has no programming experience. Linux / Unix users and people with experience of C - please be patient.

Before you start, you should know that Matlab hates file names with spaces in (and so do I), so if you are going to use Matlab extensively, get used to using _ (underscore) instead of space in your file and directory names, and avoid keeping important files in the 'My Documents' folder or the Desktop in Windows. Make a folder directly in C: or D: or whatever and use that for your Matlab analysis – it will make your life much easier.

Start Matlab by clicking on the icon on your desktop, and a variety of windows will open up. The main blank window is the command window, which has a prompt **>>** where you type your instructions to Matlab. I find it best to close all the other windows, except maybe the command history window, as the rest are a bit useless.

The current directory is shown at the top of the command window, and you can change directories by clicking on the three dots to the right of it. You can also use the system commands `ls`, `cd`, `dir` and `pwd` to move between directories.

As you read though this tutorial, try typing each line into Matlab, and if you don't understand what it is doing, try something else until you do. The only way to learn Matlab is by using it, so just try stuff. Though out this tutorial, things you can type into Matlab are show in `green` and Matlab's response is shown in `blue`. Words highlighted in red are (in general) defined in the glossary at the end.

Getting help

Matlab has a detailed help section, which you can access by clicking on the question mark at the top of the command window and searching. At any time, you can type `help` to get a list of categories of commands, for example type `help general` for a list of general commands. Type `help` followed by a command name to get more help on that command. For example:

```
>> help length
 LENGTH   Length of vector.
    LENGTH(X) returns the length of vector X.  It is equivalent to
MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.
```

Lesson 1 – The Basics

Matlab is based on a command line where you can see and manipulate variables. You enter commands at the prompt **>>** and Matlab responds.

To create the variable A with the value 1 (N.B. spaces before and after = are optional).

```
>> A=10
A = 10
```

A is now stored as a variable in Matlab's memory or workspace with the value 10. It can be used in sums etc

```
>> A+A
ans = 20
```

You can also put the result of a sum straight into a variable

```
>> B = 5+8
B = 13
```

If you put a semi-colon at the end of the line, Matlab won't show you the output of that command, but the value of the variable has still been changed.

```
>> A = 15;
```

Now type the name of the variable to see its new value:

```
>> A
A = 15
```

The name of a variable must begin with a letter, but can have numbers later in the name. Names are case-sensitive, and can be up to 32 characters long, but cannot have spaces or punctuation in the name. You can use underscore _ if you want a space, and there are certain reserved words like `if` which cannot be used as variable names. If you use a command name as a variable name, that command may not work and you will have to clear your variable to use the command

To get rid of a variable, type `clear` followed by the name:

```
>> clear B
```

You can clear everything in the workspace by typing `clear all`

The command `whos` shows you the variables that are in the workspace. You will see:
Name: the name you use to refer to the variable
Size: the number of rows (first number) and columns (second number)
Bytes: how much memory the variable uses, but you don't need to know that unless you are using very big variables.
Type: all numbers are double arrays, but you can also use text, cell and logical (see later)

```
>> whos
     Name        Size        Bytes      Class
     A           1x1         8          double array
     ans         1x1         8          double array
Grand total is 2 elements using 16 bytes
```

You can clear the variables you have created with the command clear. Use `clear var` to remove just the variable var or use `clear all` to clear everything

```
>> clear all
```

Lesson 2 - Matrices and Punctuation

If you are familiar with Excel, you will be used to the idea of putting all your data into a grid, and then adding up the rows or columns to do your analysis. Matlab lets you have as many grids or arrays as

you like, of any size, and each one is a variable with a name. You can then use the variable name to refer to the array when you want to manipulate it. Some types of array are:

Scalar – a single number

```
A = 10
```

Vector – a row or column of numbers

```
B = 1 2 3
C = 4
    3
    8
```

Matrix – an array of numbers in a grid, its size is the number of rows x the number of columns. D is a 3 by 4 matrix.

```
D = 5 6 7 9
    8 3 5 3
    5 6 3 2
```

Element – a single number within a matrix or vector.

Brackets

To enter most data into matlab, you need to use square brackets [ ]

To put data into a row vector, type the values within square brackets, separated by spaces OR commas (or both).

```
>> C = [6, 5, 8, 10]
C = 6 5 8 10
```

To put data into a column vector, type the values in square brackets, separated by semi colons. In this context, semi-colon means 'start a new line'.

```
>> D = [3; 1; 6; 5]
D = 3
    1
    6
    5
```

To put data into a matrix, use commas for the rows and then a semi-colon to start each new line. It is just like doing a row and column vector at once.

```
>> E = [1, 2, 3; 4, 5, 6]
E = 1 2 3
    4 5 6
```

In general, square brackets are used any time that you want to join things together. See **vertcat** and **horzcat** for more information.

Use round brackets ( ) to get things out of a matrix or to refer to just part of a matrix. For example, E(2,3) means the value in row 2, column 3 of E

```
>> E(2,3)
ans = 6
```

If you try to refer to something outside E, you will get an error message

```
>> E(4,3)
???  Index exceeds matrix dimensions.
```

You can also use round brackets to change part of a matrix. To make the number in the 1st row and 3rd column of E be 10, type

```
>> E(1,3) = 10
E =  1     2    10
     4     5     6
```

And if you want to add a row or column to a matrix, you just need to refer to it and it will be created. Note that this only works if you are adding a row or column of the same size as your original matrix, if you try adding a different size, you will get an error message.

```
>> E(3,:) = [7, 8, 9]
E =  1     2    10
     4     5     6
     7     8     9
```

Colon

The colon character `:` means several different things in Matlab.

In round brackets, a colon means everything in a row or column and is normally used to extract data from a matrix.

E(:,2) means everything in column 2 of E. Read this as 'E, every row, column 2'

```
>> E(:,2)
ans = 2
      5
      8
```

E(2,:) means every column in row 2 of E. Read this as 'E, row 2, every column'

```
>> E(2,:)
ans = 4 5 6
```

A colon in brackets by itself turns anything into a column vector. E(:) rearranges everything in E into one long column

```
>> E(:)
ans = 1
      4
      7
      2
      5
      8
     10
      6
      9
```

Between two numbers, the colon means count from A to B one integer at a time

```
>> F = 5:10
F = 5 6 7 8 9 10
```

You can use a colon like this to extract data from a matrix too. For example, F(:,3:5) means everything in columns 3, 4 and 5 of F

```
>> F(:,3:5)
ans = 7 8 9
```

A set of three numbers with a colon specifies the step to use for counting, e.g. G = 3:4:20 means G counts from 3 to 20 in steps of 4

```
>> G = 3:4:20
ans = 3 7 11 15 19
```

Type `help punct`, `help colon` and `help paren` for more information on matrix punctuation in matlab.

The Array Editor

I never use the array editor in Matlab, but when you are making the transition from Excel, you may find it helpful. Use the menus to select View - Workspace and Matlab will show you a window listing all your variables and their sizes, similar to typing whos. Then if you click on a variable, the Array Editor window opens. This shows you the data in a grid a bit like Excel, and any numbers you change here will be changed in Matlab's memory, while commands at the command prompt which change the variable you are looking at will automatically be shown in the array editor. You can also paste data from another program (like Excel) into a Matlab variable here.

Lesson 3 - Indexing

Indexing means getting the part of a matrix you want, and it is crucial to Matlab. The easiest way to get the data you want is to use subscripts, i.e. to tell Matlab the rows and columns that you want.

For this example, we will use a magic square matrix which is stored in Matlab and accessed with the command magic.

```
>> clear all
>> A = magic(5)
A = 17  24   1   8  15
    23   5   7  14  16
     4   6  13  20  22
    10  12  19  21   3
    11  18  25   2   9
```

We can extract the value in row 4 column 3 from A, using round brackets

```
>> B = A(4,3)
B = 19
```

or just row 4, read this as C is A, row 4 all columns

```
>> C = A(4,:)
C = 10 12 19 21 3
```

We can extract rows 2 and 3 from A. Read this as B is A, rows 2 and 3, all columns. The square brackets and semi-colon are used to join the required rows together and the round brackets to extract the rows from A.

```
>> D = A([2;3],:)
D = 23  5   7  14  16
    4   6  13  20  22
```

And we can extract columns 1 to 3 of D. Read this as E is D, all rows from columns 1 through 3

```
>> E = D(:,1:3)
E = 23  5   7
    4   6  13
```

We can also use indexing to put values into A, e.g. to make row 2 column 2 of A equal 100, use:

```
>> A(2,2) = 100
A = 17  24   1   8  15
    23 100   7  14  16
     4   6  13  20  22
    10  12  19  21   3
    11  18  25   2   9
```

To make all of column 4 of A equal 0, use

```
>> A(:,4) = 0
A = 17  24   1   0  15
    23 100   7   0  16
     4   6  13   0  22
    10  12  19   0   3
    11  18  25   0   9
```

You should now know enough to do Exercise A

Lesson 4 - Basic maths

Type `clear all` before starting this section, and enter enter E = [1, 2, 3; 4, 5, 6] and A=10.

In matlab, you can do sums on numbers or on variables. Variables are treated just like the number (or numbers) they represent. Most things are as you would expect:

\+ means plus
\- means minus
' means transpose (turn rows into columns and vice versa)
( ) round brackets can be used to specify the order of operations according to BODMAS

BUT

* means matrix multiplication
/ means matrix division
You are more likely to want to dot-multiple and dot-divide
.* means element-wise multiplication
./ means element-wise division
.^ means power (i.e. A.^2 means A squared)

You can do sums on single numbers just by typing in the command line. Spaces before and after the signs are optional.

```
>> 174 .* 734
ans =  127716
>> (A*2) / 5
ans = 4
```

You can also do sums between a scalar and a matrix. The sum will be applied to every value of the matrix.

```
>> J = E*A
J =  10   20   30
     40   50   60
```

You can add and subtract matrices if they are the same size:

```
>> K = J - E
ans =     9    18    27
         36    45    54
```

You can multiple or divide elements of two matrices using `.*` and `./` The matrices must both be of the same size, otherwise you will get an error message.

```
>> L = [3 2 1; 9 5 6]
L =   3     2     1
      9     5     6
```

To divide each value of K by the value in the equivalent place from L:

```
>> K./L
ans = 3     9    27
      4     9     9
```

To multiply each value of E by the value in the equivalent place from L:

```
>> E.*L
ans =
     3     4     3
    36    25    36
```

Note that using `*` and `/` without the `.` will perform matrix multiplication and division, where whole rows and columns are multiplied and summed. This is described in linear algebra text books, but you are unlikely to need it for basic data analysis. If you multiply and you get back a matrix of a different shape, you've probably done matrix multiplication. If your matrix is square (has the same number of rows and columns), be extra careful because matrix multiplication won't produce an obvious error.

Type `help elfun` (elementary functions) `help ops` (operators) or `help arith` (arithmetic) for more information on basic maths with Matlab.

Lesson 5 - Basic functions

Matlab has a large number of build in functions which allow you to perform simple maths and to generate matrices quickly and easily. Functions are bits of code (written by you or Matlab or anyone) which receive some inputs and give you some outputs. They use round brackets, and all functions written by Matlab have help files to tell you how to use them. Type `help` followed by the command name for details.

All functions have one or more inputs or arguments and produce one or more outputs. If a function has just one output, the output can be placed straight into a variable, but if a function has several outputs, square brackets are needed to group the output variables.

All functions have the form:

```
[output1, output2, ...] = function (arg1, arg2, ...)
```

All the mathematical functions you need to operate on single values are available, e.g sin, cos, tan, sqrt, log and pi. Matlab has a variety of useful functions which you can apply to the rows or columns of a matrix, including sum, mean, std, max and min.

By default, these work on columns, but you can change the dimension to work on rows. Type `help` followed by the function name to see how to do this.

To sum the columns of K:

```
>> sum(K)
ans =   45    63    81
```

To sum the rows of K, you must tell the sum function to operate in the 2nd dimension (1st = columns, 2nd = rows)

```
>> sum(K,2)
ans =   54
       135
```

To find the mean of J and put it in a variable

```
>> mj = mean(J)
mj =  25    35    45
```

The apostrophe is useful to transpose a matrix or vector, i.e. to turn all the rows into columns and the columns into rows

```
>> K'
ans =   9    36
        18    45
        27    54
```

There are also a set of functions to create vectors and matrices. These include `rand`, `ones`, `zeros`, `repmat`, `ndgrid`, `linspace`, `logspace` and `magic`.

e.g. To create a matrix of random numbers with 3 rows and 5 columns (yours will be different because R is random)

```
>> R = rand(3,5)
R =
    0.6154    0.7382    0.9355    0.8936    0.8132
    0.7919    0.1763    0.9169    0.0579    0.0099
    0.9218    0.4057    0.4103    0.3529    0.1389
```

Many more useful functions are listed in the glossary. type help followed by a function name to find out what each one does.

You should now know enough to do Exercise B.

Lesson 6 - Logical Operators

Type `clear all` before starting this section to clear all the previous variables.

Logical operators are used to assess if a statement is true or false. False is always represented by 0, and Matlab will consider any non-zero value to be true. For neatness, it is best to use 1 as true. Logicals can be used to decide which part of a script to run next or which elements of a matrix to use in a calculation. There are four main logical operators:

```
>    Greater than
<    Less than
==   Is equal to
~=   Is not equal to
```

You can also join logicals together using AND, OR and NOT, together with round brackets

```
&    logical AND
|    logical OR
~    logical NOT
```

In Matlab, you can apply logical operators to whole arrays as well as to individual numbers, and then you obtain a logical array which can be used to index another array.

In this example, B shows which values of A are larger than 2, and C shows which values of A are smaller than 5.

```
>> A = [1 5 3 4 8 3];
>> B = A>2
B = 0 1 1 1 1 1
>> C = A<5
```

```
C = 1 0 1 1 0 1
```

D shows where both B and C are true.

```
>> D = B & C
D = 0 0 1 1 0 1
```

You can see that B, C and D are a logical when you type `whos` and the class is listed as logical. This means Matlab will treat these matrices as a list of True and False values rather than just a list of ones and zeroes.

Now you can use D as a logical index into A:

```
>> E = A(D)
E = 3 4 3
```

E contains only the values of A which are true according to D, that is, only the values which are larger than 2 AND smaller than 5. For these values, there was a one (TRUE) in the corresponding place in D, and all the other values in A are dropped.

Logical indices can often be used in a similar to the subscript indices described above. Both function to select out particular values from an array. You can convert a logical index into a subscript index using find, which tells you the non-zeros values of the logical.

```
>> F = find(D)
F = 3 4 6
```

This means that the 3rd, 4th and 6th values of D are not zero. If you use the subscript index F into A, you will get the same result as using the logical index D into A:

```
>> A(F)
   ans = 3 4 3
```

Logical indexing is most useful when one variable is used to categorise another. Imagine you have some data which falls into three different categories.

```
>> data = [4 14  6 11  3 14  8 17 17 12 10 18];
>> cat =  [1  3  2  1  2  2  3  1  3  2  3  1];
```

To find where cat is 2:

```
>> cat2 = cat==2
cat2 = 0  0  1  0  1  1  0  0  0  1  0  0
```

To find the values of data where cat is 2:

```
>> data2 = data(cat2);
data2 =     6     3    14    12
```

Now we can find the mean of the data for category 2

```
>> mdat2 = mean(data2)
mdat2 = 8.75
```

We could have done all 3 steps in one line:

```
>> mdat2 = mean(data(cat==2))
mdat2 = 8.75
```

Lesson 7 - Missing Data

Real life psychology experiments often suffer from missing data - you may want to exclude trials where reaction times fall outside some limits, or where the subject made an error. In Matlab, you can do this using `NaN`. NaN stands for Not A Number and can be used in any vector or matrix in place of missing data. In general, it is much better to replace your missing data with NaNs than to delete it altogether. When you use NaN, some of the basic functions like sum and mean will no longer work as

you expect, but Matlab provides alternate versions, `nansum`, `nanmean` and `nanstd` which let you ignore NaNs and find the sum, mean and standard deviation of your data.

A quick example using the data set from Lesson 6. An additional vector err has a one at every location where the subject made an error;

```
>> err = [1 0  0 0  0 0  0 1 0 1 0 0];
```

We can use err as an index to replace every error in the data with a NaN. This line should be read as the data where err is one is changed to NaN;

```
>> data(err==1) = NaN
data = [NaN 14  6 11  3 14  8 NaN 17 NaN 10 18]
```

Now when we extract the data in category 2, one of the numbers is NaN

```
>> data2 = data(cat2);
data2 =     6     3    14    NaN
```

So we have to use nanmean to find the mean

```
>> mdat2 = nanmean(data2)
mdat2 = 7.6667
```

Some other useful functions to use with NaNs include `isnan`, which returns a 1 at places where a matrix is a NaN and zero elsewhere, and `isfinite`, which returns a 1 at places where a matrix is a number (not NaN or Inf) and zero elsewhere.

You should now know enough to do Exercise C and D

Lesson 8 - Basic Graphs

Visualising your data is very important, but first we need to generate some data. Type `clear all` before you begin this section. Linspace is a useful function for generating a sequence of numbers of a specific length. Type help linspace to see what it does

```
>> x = linspace(0,2*pi,30);
>> y = sin(x);
>> z = 0.5+cos(x/2);
```

Basic graphs

Plot is the basic function for plotting vectors and matrixes as lines or points. You specify the x data, the y data and the type of line. Type `help plot` for a list of possible lines. First we will plot y against x

```
>> plot(x,y,'b-o')
```

A new window called Figure 1 will appear. Now keep the plot so you can add another line

```
>> hold on
```

The plot z against x

```
>> plot(x,z,'r--')
```

Now you can label the axes, give your plot a title and add a legend. Note that you need to use text within single quotes for the labels and title.

```
>> xlabel('x')
>> title('a couple of lines')
>> legend('y=sin(x)','z=0.5+cos(x/2)')
```

You can specify the axis limits using axis. For example, to set the x axis from 0 to 10 and the y axis from 5 to 15, type `axis([0, 10, 5, 15])`

Plot some data

Make sure the file Lesson2.mat is in your current directory and type

```
>> load lesson2
```

to load some data to plot. If the file doesn't load, type `ls` to see if you can see the file in your current directory. If you can't see lesson2.mat when you type `ls`, you will have to find the file (it is on my website if you don't have it, Google Antonia Hamilton) and put it in your current directory. The current directory is shown at the top of the Matlab window.

Type `whos` to see what you have got. **`data`** is a matrix of test scores on three different subjects, each repeated ten times.

We need to clear the old graph to plot a new one. You can use `close all` to close all figure windows, or `clf` to clear a single figure, or `cla` to clear just one axis in a figure.

```
>> close all
```

First let's just take a look at our data. If you use plot with just one argument, matlab just uses numbers from 1 to the length of the data on the x axis.

```
>> plot(data,'bo')
```

To see if there is a trend, we can plot the mean

```
>> hold on
>> plot(mean(data),'r-^')
```

Or we can use the `errorbar` function to plot the mean with error bars.

```
>> errorbar(mean(data),std(data),'g-')
```

This is all getting a bit messy, so instead we can plot a new figure with several subplots. You have as many figure windows open as you want, and you can put multiple plots within each window:

Create a new figure called figure 2

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

You can put lots of plots in one figure with the subplot command. You must specify the number of rows of subplots, the number of columns of subplots and the number of the current subplot in the form:

```
subplot(rows, columns, current)
```

Create a set of subplots which will have 2 rows and one column, and make the first subplot (counting from top left) active. Then plot data in it and give it a title.

```
>> subplot(2,1,1)
>> plot(data,'b*')
>> title('Raw data')
>> subplot(2,2,1)
>> bar(mean(data),'r')
```

this plots the mean of the data using bars rather than lines

```
>> hold on
>> errorbar(mean(data),std(data),'k')
>> title('Mean data')
```

Some useful plotting functions are listed in the glossary, or type `help graph2d` to see the complete list of graph functions available.

You should now know enough to do Exercise E.

Lesson 9 - Basic Scripts

Typing stuff at the command line quickly gets confusing if you have a lot of commands. To avoid this, any matlab command you write into the command line can also be written into a script, which is basically just a list of commands that are all executed one after another in the order they appear on the screen.

You can edit scripts in any program you like (notepad, emacs etc) but the matlab script editor has a few advantages. In particular, it highlights the script to show you particular words and can automatically indent loops to look nice (select File – Preferences-Editor-Indenting-Emacs style smart indenting to get this feature). Also, if you highlight a chunk of a script and press F9, matlab runs that chunk.

To start making a script, type `edit` to get the Matlab script editor (if you are in Windows). For this section, write the lines in `dark green` or `purple` in the script editor (they won't appear in purple). Then highlight them and press F9 to run your script. If F9 doesn't work, it is because you didn't start the editor by typing `edit`, (or because you aren't using Windows) but you can cut and paste scripts into the command line instead. You can save and load scripts using the menus just like Word, and you can run a whole script by typing its name in the command line.

It is good practice to start every script with the line `clear all` (unless you specifically want to keep previous variables), and a comment line saying what the script does. Comment lines begin with `%`, and are ignored by Matlab as it runs the script but can be very useful to annotate and explain your code, or to cut out lines which aren't working when you are debugging. Comments can go on a line by themselves or at the end of a line of code.

The semicolon `;` is a very useful character for scripts. Normally, when you enter a command into Matlab, it responds by showing you the answer to your sum or the new value of your variable. But when you are running a script and doing the same sum hundreds of times, you don't need to see the result every time. By putting a semi colon at the end of a line, you can suppress the output so that it does not appear on the screen, but Matlab still does the calculation.

Some other useful commands when scripting include `pause` which pauses the script if you do want to see the output on the screen (hit any key to start the script running again), and **Crtl-C** (that is, press the Control key and the letter C at the same time) which stops a crashed script and gives you back your command prompt.

Using the example data from the end of Lesson 6, we could write a very simple script to find the mean of the data in each of 3 categories, put all these into a variable mdat and plot the results. Enter the lines below into your script editor

```
%% this script plots the mean of data for each category
clear all
data = [4 14  6 11  3 14  8 17 17 12 10 18];
cat =  [1  3  2  1  2  2  3  1  3  2  3  1];
mdat(1) = mean(data(cat==1))
mdat(2) = mean(data(cat==2))
mdat(3) = mean(data(cat==3))
figure(1), clf
plot(1:3,mdat,'r-*')
title('Mean of data for each category')
```

Once this is written in the script editor, all the lines can be executed by highlighting them and pressing F9. This gives you a simple and repeatable way to do some analysis. When you have created a plot like this using Matlab, you have a script so you can always look back and see exactly what you did to get the figure. And if you have a new data set, you can often apply the same analysis by just changing a few characters. This is what makes scripting in Matlab so efficient for data processing. Save this example as `lesson8.m` because you will need to edit it below.

Lesson 10 - Flow Control

A script can run any sequence of commands in the order they appear, but to make scripts really useful, we need to be able to tell the script to start or stop at different points. There are 5 basic commands which allow you to do this in Matlab.

If

If is the most basic control element. If an expression is true (or evaluates to a number other than zero), the next command will execute. An **if** can also have an **else**, which executes if the first part is false or equal to zero, and must have an **end** to finish the command.

```
>> A = 10;
if (A>5)
  B = 1
else
  B = 0
end
B = 1
```

Try this one out for different values of A

If statements are often useful for catching errors. For example, you could modify your script **lesson8.m** to include these lines (put them straight after you define cat and data).

```
if( length(cat) ~= length(data) )
     disp('ERROR: Data and categories are not the same length')
     return  %% return means stop running the script
end
```

The script will now give you a helpful error message if you entered the data wrong.

For

For runs a set of commands multiple times in a loop. It has a counter which tells it how many times to run through the loop. The counter can be used within the loop as an index or as a variable but should NOT be changed inside the loop.

In the example below, the loop runs once with i=1, so A(1) = 2, then it runs with i=2, so A(2) = 4 etc. The values of i are specified in the same way as creating a vector with a sequence of values (see colons above). A **for** loop must always have an **end** to tell it where to stop, and can also be stopped prematurely with **break**.

```
for i=1:4
   A(i) = i*2
end
A = 2
A = 2 4
A = 2 4 6
A = 2 4 6 8
```

As a second example, look at lesson8.m again. We could replace the three lines which calculate mdat with a **for** loop. We put the counter i in the place of the number which changed between each line and the loop will calculate the mean for each category and put the results in mdat. Obviously when there are only three categories, writing the loop does not save much time, but if there were 100 categories it would.

```
for i=1:3
```

```
    mdat(i) = mean(data(cat==i))
end
```

Switch

A switch loop is like testing lots of ifs one after the other. It is useful if you want to match a value to several alternatives. Put the value you want to match in brackets after the **switch** statement. Each **case** is a possibility, and if the value matches, then next statement will execute. It none of the cases matches, the lines after **otherwise** will execute. A **switch** must have an **end** to tell it to stop. (disp means display text). Switch loops can also determine if one text string matches another text string, and are useful for classifying text strings.

```
>> A = 3
switch(A)
  case 1
    disp('A is one')
  case 3
    disp('A is three')
  case 5
    disp('A is five')
otherwise
   disp('A is not one or three or five')
end
'A is three'
```

While

While is like a continuous **for** loop, which keeps going until it gets a false value. In this example, the loop keeps increasing y until y is greater than 1. Then x is set to 2, so the loop stops. Again, there must be an **end** to stop the loop. While loops are most useful for reading in text files of an unknown length.

```
x = 1;
y = -5;
while(x==1)
    y = y+1
    if(y>1)
      x = 2
    end
end
y =  -4
y =  -3
y =  -2
y =  -1
y =   0
y =   1
y =   2
x =   2
```

Try ... Catch

This is not really a control element, it is used for catching errors and preventing them from crashing your script. You can ask Matlab to `try` to do something, and if it generates an error, to `catch` the error with another instruction. This is useful when you are doing things like reading in text files or processing data where a few trials might be anomalous. For example:

```
>> A = 1:10;
>> B = 1:5;
for i=1:length(A)
      try
            C(i) = A(i) + B(i)     %% this will give an error when i
                                   %% is greater than the length of B
      catch
            disp('B is too small')
      end
end
C = 2
C = 2 4
C = 2 4 6
C = 2 4 6 8
C = 2 4 6 8 10
B is too small
```

Lesson 11 - Functions

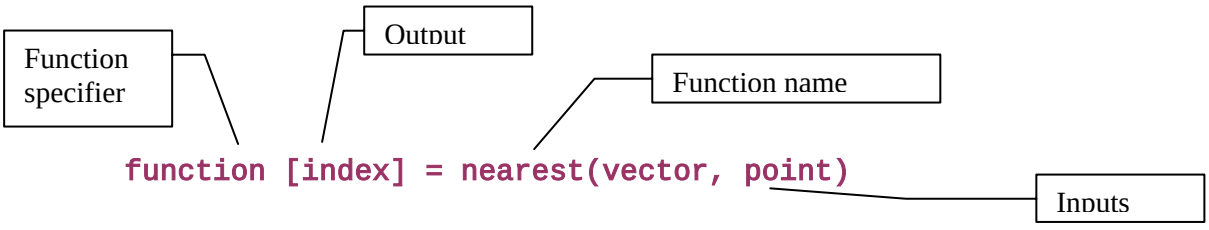
A script is just like typing a long list of commands into the command window all at once, and they can get quite long and complicated. If you have a simple little task which you do often, it is often easier to write a function to do it for you. Functions take inputs and return outputs, but everything that happens within a function is hidden or encapsulated from the main workspace. This means that if variables within a function have the same name as variables in the workspace, they won't interfere with each other or over-write each other's value.

The exception to this rule is global variables. If a variable is declared as global in a function (i.e. the line `global var` is included at the top of the function) AND is declared as global in another function or script, the variables are no longer hidden and share values. This means changes to the variable in one place affect the other. For example, if you use spm, the variable **`defaults`** is global. This means that after starting spm, you can type `global defaults` at the command line to make **`defaults`** visible to you. Then you can see and edit all the default settings for data processing.

A function is written like a script, but it must begin with the word **`function`** and then have the form **`[outputs] = function_name(inputs)`**. There can be as many inputs and outputs of any type (arrays, strings etc) as you want.

It is good practice to put some helpful information after the function name. All lines immediately after the function name which start with a `%` act as the help file for the function, so it is useful to record the format of the inputs and outputs which the function needs.

This is an example of a function which finds the index of the number in the vector which is closest to the point. Type it into a new page in your script editor and save it with the name nearest.m.



```
%%%%%%%%%%%%%%%%%%%%%%%%
%% [index] = nearest(vector, point)
%% this function finds the index of the number in the vector
%% which is closest in absolute terms to the point
%% if there is more than one match, only the 1st is returned
df = abs(vector-point);
ind = find(df==min(df));
index = ind(1);
```

Now you can test your new function by using it at the command line. Type help nearest for a reminder on how it works. Generate some random data r = rand(50,1) and find the data point which is nearest to 0.5. Check that is it right with a plot.

Paths

When you run a script or function in Matlab, or use someone else's function, how does Matlab know which program to run? This may not seem important when you only have a few scripts, but it can quickly become crucial. The general rule is that Matlab looks first in the current directory, and then in every directory in the **path** in order, until it finds a command with the name you asked for. To see what is in the Matlab path, type **path**, and to edit the path, select the option **Set path** on the **file menu**. If you want to know quickly which command Matlab is running, type **which** followed by the command name.

You should now know enough to do Exercise F.

Lesson 12 – More about variables

Saving and Loading your data

Matlab can save any variables you want in a .mat file, so you can load them at a later date. Filenames can be as long as you like, but cannot begin with a number or contain spaces. The save, load and clear commands work in lists separated by spaces, not with brackets. You just type the command, followed by a list of stuff to save, clear or load

To save x, y z and data in the file somestuff.mat

```
>> save somestuff x y z data
```

the file somestuff.mat will appear in the current directory and contains the variables x, y z and data

Now clear everything

```
>> clear all
```

the variables are removed – type **whos** and you will see they have gone.

```
>> load some_stuff
```

x, y z and data are loaded back again!

You can also save stuff in different formats, for example to load into SPSS or Excel. The command dlmwrite writes a matrix as a text file with the numbers separated by tabs. This can be read by SPSS or Excel or anything else.

To write data as a tab delimited text file: The '\t' is the tab delimiter, you can use other things, but tabs are compatible with all programs I've come across.

```
>> dlmwrite('mydata.txt',data,'\t')
```

The file mydata.txt will appear in the current directory

Type **help iofun** for a complete list of input / output functions (i.e. functions to save and load data)

Strings and Cells

So far, all the variables we have dealt with are numbers, but it is often useful to be able to manipulate other types of data. Text (i.e. strings) can be assigned to a variable just like a number, as long as the text is put within single quotes

```
>> firstname = 'Joe';
>> surname = 'Bloggs';
```

Text can be joined together just like two vectors could, though it looks nicer to put a space between using ' ' (that is just a space character in single quotes).

```
>> fullname = [firstname,' ',surname]
     'Joe Bloggs'
```

And you can use `num2str` to turn numbers into text to display them

```
>> age = 25;
>> agestring =[fullname,' is ',num2str(age),' years old']
     'Joe Bloggs is 25 years old'
```

It is generally inconvenient to put text strings into arrays because Matlab will complain if they aren't all the same length, but a cell array behaves like a matrix, without minding what size each entry is. Cells can be referred to in just the same way as matrices, but they use curly brackets {} instead of round ones (). For example, to put the names of a set of stimuli into a cell array:

```
>> stimuli={'dog','cat','horse','rat';'car','train','hammer','van'}
stimuli =
    'dog'    'cat'      'horse'     'rat'
    'car'    'train'    'hammer'    'van'
```

Note that you use commas to separate items on the same row and a semicolon to start a new row, just like with a regular matrix. You can reference individual items from a cell array using curly brackets

```
>> stimuli{1,3}
ans =
horse
```

Or you can move rows and columns just like matrices. We use curly brackets here so the items are moved as a whole array, not 4 separate things, and the result is a new cell array called animals.

```
>> animals = stimuli(1,:)
animals =
    'dog'    'cat'    'horse'    'rat'
```

Cell arrays can contain a mixture of text and numbers, and can even contain other matrices, but I find they are most useful for text. Matlab has a set of string manipulation functions which can be used to do things like find a particular string in a cell array

```
>> strmatch('cat',animals)
ans =
     2
```

To turn numbers from a cell array into a numeric array, use `cat`

```
>> nums = {10, 15, 8, 12};  %% numbers as cells
>> realnums = cat(1,nums{:});  %% turn them into a numeric array
```

Type `help strfun` and `help cell` for more information

Structures

If you've been doing analysis for a while, you are likely to type whos and find you have about 30 variables and can't remember which are which or which are important. When this happens, the best solution is to use a structure to organize your variables. Structures are a hierarchical way to organize your data, where a single variable can have many fields, each of which has a name starting with a dot (full stop). For example, if your data consists of measures of performance on a word task and a picture task, you might have three vectors of data called **word_data** and **pic_data**. Load lesson3.mat to get these variables. You could organize these by putting them in a structure called data, together with the name and age of the subject who did the task

```
>> data.word = word_data;
>> data.pic = pic_data;
>> data.subjectname = 'Joe Bloggs';
>> data.subjectage = 25;
```

Then type data to see what is in the structure data:

You can refer to any of the fields in data just by typing its name. Structures are most useful for grouping different types of data together, allowing you to see and save important variables quickly. Structures can have multiple layers (each field can have subfields) and multiple sets of entries. For example, you could add the data for a second subject to your structure by putting it in data(2).

```
>> data(2).word = word_data2;
>> data(2).pic = pic_data2;
>> data(2).subjectname = 'John Doe';
>> data(2).subjectage = 26;
```

This could be useful if you want to keep all the data from all your subjects in the same format, in order to apply the same process to all of it.

System commands

This isn't really a data type, just another short topic. From the Matlab command prompt, you can control your computer just like at the command line. The system commands `dir` , `ls`, `pwd` and `cd` work directly to show you the files in the current directory and let you change directory, just like at the unix or dos command prompt. You can also use the command `system` (for windows) or `unix` (for unix) to send a command to system. For example

```
>> system('copy data.txt processed_data.txt')
```

copies the data file data.txt to the file processed_data.txt. You can use these commands in for loops to move your data from one directory to another or rename files as you process them (especially useful for imaging data when you have hundreds of files).

The command `eval` is another helpful one. It allows you to evaluate an arbitrary string as a Matlab command, so again you can put together commands (say, for variables with different names) using text strings and then ask Matlab to execute them.

You should now know enough to do Exercises G and H.

Lesson 13 - Advanced Graphs

The basic plot function in Matlab will let you do most things, but sometimes you want to have more control. In Matlab, you can define every aspect of a figure from the command line, but this can be quite complex. For details, see the matlab graphics manual, the interactive help or the documentation on the matlab website, www.mathworks.com. Most things about the figure can also be changed by clicking on the line / text you want to edit and using the Edit and Tools menu in the figure window. Here I will describe just a few basic things.

Graphics Handles

All figures, subplots, axes, lines on plots etc have a handle which you use to tell Matlab which part of a figure you want to edit. Most of the time the handles are completely invisible to the user - you don't need to know or care about them, but there are a couple which are useful

gcf is the handle to the current figure. The command `get(gcf)` will show you all the properties of the current figure. This is a useful way to decide which ones you want to edit. For example, you can tell Matlab exactly where you would like a figure to be placed on the screen with the 'Position' property.

```
set(gcf,'Position',[50,50,600,800])
```

this means put the current figure 50 pixels above and 50 pixels to the left of the bottom left hand corner of the screen, and make the figure 600 pixels wide and 800 pixels high.

gca is the handle to the current axis. Again, `get(gca)` shows you all the properties you can edit. You can change things like the labels on the tick marks of an axis.

```
set(gca,'XTick',1:4','XTickLabel',{'cat','dog','car','boat'})
```

This gives the current axis four tick marks at numbers 1 to 4, and puts the words 'cat', 'dog' etc on each tick mark.

Line handles are obtained by giving a plotting function an output. For example

```
h = plot(1:10, rand(1,10))
```

h is now the handle to your line. You can use h to do things like change the colour or width of the line

```
set(h,'LineWidth',3)
```

Full details of graphics handles are given in the Matlab graphics manuals.

Images

As well as lines, Matlab can plot any matrix as an image. The basic image commands are `image` and `imagesc`. They both plot a matrix as a coloured pattern, but imagesc scales the colour automatically to look nice. The command `colorbar` adds a scale at the side of the image, and `colormap` changes the colours used. `Help graph3d` gives a list of possible colormaps.

```
x = linspace(0,4*pi, 100);
y = linspace(0,2*pi,100);
c = sin(repmat(x,100,1))+cos(repmat(y',1,100));
figure(1), clf
imagesc(x,y,c)
colorbar
```

If you look at c in the command line, it is just a bunch of numbers, but imagesc reveals the pattern in the numbers. I find it is a very useful function for looking at large amounts of data at once - sometimes you can see the anomalous subject just by looking at the data.

3D graphs

Matlab can also plot things in three dimensions, and you can zoom and move around the plot as you like. help plot3d gives you a complete list of 3d plotting functions. The most useful is `surf` (to draw a 3D surface). For example, our colored image could be plotted in 3D

```
x = linspace(0,4*pi, 100);
y = linspace(0,2*pi,100);
c = sin(repmat(x,100,1))+cos(repmat(y',1,100));
figure(2), clf
surf(x,y,c)
```

At the top of the figure window, there is an icon of an arrow going around in a circle. Clicking this gives you the ability to rotate and move your 3D plot so that it looks nice. You can also control a wide range of camera angles, lighting, surface reflectance, in fact anything you need to create a virtual world. Look in the Matlab graphics manual for details.

Pictures

As well as displaying data, Matlab can read and display a range of picture and movie formats, including bitmap, jpeg, gif, avi etc. This feature is useful for viewing / editing pictures or creating fancy images. The script below is an example of how to load a bitmap, add a fixation point and save it with a different name. Scripts like this give you a quick way to generate simple stimuli for experiments. This assumes you have a file called dog.bmp, obviously you can change the name or make one if you don't.

```
clear all, close all
picname = 'dog.bmp'
%% read in the image file
dog = imread(picname)
[m,n,p] = size(dog)
%% show the image
figure(1), clf
image(dog)
%% set the figure properties nicely
set(1,'Position',[50,50,m,n])
set(gca,'Position',[0,0,1,1],'XLim',[0,m],'YLim',[0,n])
axis off
hold on
%% draw a plus for fixation
h=text(m/2,n/2,'+')
set(h,'FontSize',20,'FontWeight','Bold', ...
'HorizontalAlignment', 'center')
frame = getframe(gca);
%% extract the image from the figure
dog_fix = frame2im(frame);
%% write out the image with a new filename
imwrite(['fix_',picname],dog_fix)
```

This whole script could be put into a **for** loop to apply exactly the same transformation to a whole set of images at once, so even if it takes a while to write the script, when you have several hundred images this is quicker and more reliable than editing each of them in photoshop.

Lesson 14 - How do I read this data file into Matlab?

This is one of the questions I am asked most often, and the answer is - it depends on your data file. The easiest way to transfer your data between any programs is to use tab delimited text. This is a basic text file where you have a matrix of data in the format:

```
number TAB number TAB number ... NEWLINE
number TAB number TAB number ... NEWLINE
... etc.
```

where TAB means the tab character and NEWLINE means a carriage return character. Each row must have the same number of columns and not contain any text strings. This format is compatible with

every program I've come across, including Excel, Matlab and SPSS, so if you can get your data in this format, life should be easy. Just use `dlmread` to read it into Matlab and `dlmwrite` to output a matrix as tab delimited data.

However, lots of programs like to give you back data in a much messier form, where each line of your text file has a mixture of words and numbers which mean different things. In this case, you will have to write a custom script to read your file. It is guaranteed to be tricky to write and need lots of trial and error, but once you have it working you will be able to process your data very quickly!

A basic example of a script to read data files is below. It assumes every line starts with an identifier, either TIME or STIM or KEY, followed by some data.

```
%%% this is my script to read my data files
fname = input('Enter the file name ')  %% prompt the user for a
filename
fid = fopen(fname)  %% open the file and get a file ID
fline = fgets(fid)   %% read a line of the file
w = 1;     %% counter
while(fline~=-1) %% fline will be -1 when the file is finished
                 %% keep going until this happens

     switch fline(1:4)  %% switch loop based on first few
characters if the line
     case 'TIME'
          trial_time(w) = str2num(fline(6:10))  %% read time
     case 'STIM'
          stimulus{w} = fline(6:length(fline));  %% read stimulus
     case 'KEY '
          key(w) = str2num(fline(6)) %% read key hit
          w = w+1;  %% increment counter
     end
end
fclose(fid)  %% close the file
```

Most data files are much more complex than this example, but if you start with this, you can adapt it to what you need. Some helpful functions for dealing with text include: `num2str` and `str2num` - convert between numbers and strings. `isspace` - find white space characters, `isletter` - finds letters, `strmatch`, `strfind`, `strcmp` and `findstr` are all variants on finding a string within a line. Type `help iofun` and `help strfun` for more information on file input and text functions.

Lesson 15 - An example of a complete analysis script

will be here one day …

Matlab for Psychologists: Exercises

Exercise A: Data Manipulation

Make sure you type `clear all` before starting

1) Enter the following matrices and vectors

```
a =  9 12 13  0
    10  3  6 15
     2  5 10  3
b =  1  4  2 11
     9  8 16  7
    12  5  0  3
```

2) use `a` and `b` to create these matrices
   a) `c` is the element in the 3rd row and 3rd column of `a`
   b) `d` is column 3 of `a`
   c) `e` is rows 1 and 3 of `b`
   d) `f` is `a` and `b` one above each other
   e) `g` is column 1 of `a` next to column 4 of `b`
3) Change some of the entries in this matrices
   a) make element (2,2) of `e` be 20
   b) make row 1 of `a` be all zeros
   c) make column 3 of `f` be the numbers from one to 6
   d) make column 1 of `a` be the number from column 2 of `b`

Exercise B: Maths and functions

Make sure you type `clear all` before starting

1) Enter the following matrices and vectors:

```
A = 1 5 6
    3 0 8
B = 7 3 5
    2 8 1
C = 10
D = 2
```

2) Do these sums:

```
E = A - B
F = D*B
G = A.*B
H = A'
J = B/D
```

3) Do some maths on parts of matrices
   a) Put the first column of A into M

b) Put the second column of G into N

c) Add them together

d) Multiple ONLY the third column of A by C and put the result back in the third column of A. You can use several steps if you want, but it is possible do use just one line.

e) Find the Dth row of H (i.e. row 2) and sum all the elements in that row

f) Create a new matrix K made up of A in the first 2 rows and B in the next 2 rows.

4) Look at difference between array multiplication and matrix multiplication

a) Try `A*B` (you will get an error)

b) Try `A*B'` (that is, A matrix multiplied by B transpose), and compare to `A.*B` (that is, array multiplication of A and B)

5) Find the maximum values of each column of J, and find the minimum value of each row of B

Type `help max` and `help min` if you need to

Exercise C: Logicals

Make sure you type `clear all` before starting, and load the dataset ex_C.mat. This data set contains 3 vectors:
`rt` is a vector of reaction times of one subject on a visual attention task.
`cue` is a vector describing if the cue was valid (1), invalid (2) or absent (3).
`side` is a vector describing if the target was on the left (1) or right (2) of the screen.

1) Create a logical vector called `valid` which defines all the trials with valid cues

2) Use this to find the mean and standard deviation of reaction time on valid cues

3) Error trials are ones where the reaction time is less than 100 msec or more than 1000 msec. Create a logical vector called `error` defining which trials are errors

4) Using both `error` and `valid` to find the mean and standard deviation of reaction times for correct valid trials

5) Create a logical vector called `side` which you can use to determine the side of stimulus presentation.

6) Does mean reaction time on all trials vary with side? What about reaction time of valid trials only?

Exercise D: A real data set

Fred is doing a pilot study on the perception of attractiveness, so he asked his flatmates to judge the attractiveness of 20 faces. The gender and symmetry of each face was known in advance. Clear everything and load the data set ex1.mat. This data set has 3 vectors and 1 matrix:

`stimno` (stimulus number)
`gender` (1 = male, 2 = female)
`symm` (symmetry on a 1 to 5 scale)
`data` (each column is the attractiveness judgment on a 1-100 scale for one of his 3 subjects)

1) Fred wrote down the wrong responses for subject 2, stimuli 3, 5 and 17. The values should be 30, 62 and 35. Change them.

2) Find the mean, max and min attractiveness for subject 3, then find the range of responses for each subject.

3) Does subject 1 rate male or female faces as more attractive?

4) Are the faces with a high symmetry (4 or 5) rated as more attractive than faces with low symmetry (1 or 2) by subject 3?

5) Do subjects 2 and 3 agree on any of their ratings? How many?
6) Find the mean disagreement between subjects 1 and 2 (use abs)
7) Faces 4, 10, 12 and 18 were much older than the other faces. Were they ranked as less attractive?
8) Sort the data according to the stimulus number (use `sortrows`)

Exercise E: Basic Graphs

Make sure you type `clear all` and `close all` before starting

1) Try plotting some graphs
    a) Generate a vector X with values from 1 to 10
    b) Generate a vector Y containing X squared.
    c) Generate a vector Z containing X*9
    d) In Figure 2, plot Y against X, using a red line with stars at each data point. (type `help plot` if you need to)
    e) Keep that plot, and on the same graph plot Z against X using a green line with squares at each point
    f) Give your figure a title and legend
2) Load Exercise2.mat. This file contains three vectors – `iq`, `scoreA` and `scoreB`, which represent the results test scores obtained by some (fictional) subjects
    a) Plot iq against scoreA using blue circles
    b) On the same plot, plot iq against scoreB using red squares
    c) Label your axes
    d) If you have the statistics toolbox, type `lsline` to fit a least squares line to this data
    e) In a new plot, compare the mean of scoreA and scoreB using a bar graph. You may find it easiest to create a new vector containing both mean values.
    f) Add errorbars to your bar graph
    g) Start a new figure with two subplots. Plot the distribution of scoreA in subplot 1 (use `hist`) and the distribution of scoreB in subplot 2.

Exercise F: Scripts and Functions

1) Look back at exercise C and write a script to perform this exercise. Your script should also plot the mean reaction time in the cue-valid, cue-invalid and no-cue conditions.
2) Try writing loops
    a. Write a for loop to find the mean of 3 random numbers 20 times and place the result in a vector A
    b. Write a second loop to find the mean of 30 random numbers 20 times and place the result in vector B
    c. find the standard deviations of A and B
    d. Clear figure 2 and plot A and B in different subplots so you can compare their distributions (try a histogram!)
3) Test out the switch loop given as an example in the main text
    a) Run the loop for several different values of A
    b) Modify the loop to tell you if A if 0 or 1 or 2

c) Modify the switch statement to give you the remainder of A divided by 3 (use the `rem` command)

d) Put the whole `switch` loop within a `for` loop that counts though A from 50 to 70. Your program should print what remainder of A/3 in words for each value of A.

4) Write a function which will find the standard error of a vector, and test that it works.

Exercise G: Structures and cells

You can use a script to do this exercise if you like.

1) Enter the following words into a cell array, making sure they all go in one column: hat coat gloves shoes socks vest jacket gown.
2) In the second column of your cell array, enter the word frequency class of each word. Hat is high frequency, coat, shoes and jacket are medium frequency and the others are low frequency.
3) load ex_F.mat. This contains a matrix ld which represents the lexical decision times for 30 subjects responding to these 8 words. Write a script to plot the mean and standard error of lexical decision time for each word. Put the words (from your cell array) on the x axis.
4) Sort your data into responses to high, medium and low frequency words. Use a for loop and a switch statement to work through your cell array and classify each word
5) Plot the mean lexical decision time for high, medium and low frequency words with a bar plot and nice heading and axis labels.
6) Put all the useful results of your analysis into a structure and save it. You should save the word lists and the mean and standard error for each word.

Exercise H: More real data

ex2.mat contains data from a single subject performing a stroop task.

`Cong` - 1=congruent, 2=neutral, 3=incongruent
`Stim` - 1=words, 2=colours
`data` - reaction time
`err` - error rate, 1=error

1) Plot the raw data for responses to congruent colors
2) Exclude all errors and reaction times below 200 msec or above 1000 msec
3) Does the exclusion rate differ between conditions?
4) Use a histogram to see if the data is normally distributed
5) Write a for loop to find the mean and standard error of reaction time in each condition and put them in a structure
6) Plot the results of 5 with error bars
7) Load the data from a second subject (ex2b.mat) and use your script to analyse it.

Matlab Glossary

Glossary - Definitions

Scalar	a single number `D = 10`

Vector	a row or column of numbers
	`A = 1 2 3`
	
	`B = 4`
	`3`
	`8`

Matrix	an array of numbers in a grid, its size is the number of rows x the number of columns. C is a 3 x 4 matrix
	`C = 5 6 7 9`
	`8 3 5 3`
	`5 6 3 2`

Element	a single number within a matrix or vector.
Variable	any items stored in matlab's workspace. Type whos to see your variables.
logical array	a vector or matrix of ones and zeroes where 1 means TRUE and 0 means FALSE
cell array	An array of items in curly brackets, which can be any size. Cell arrays are particularly useful for storing lists of words or a mixture of words and numbers. Cell arrays can be referenced and indexed like normal matrices, but obviously you can't do maths on words.
workspace	matlab's memory, containing a number of variables which can be seen with the command whos
argument	The input(s) to a function

function	A piece of code which takes some inputs (arguments), performs some operations on them, and returns some outputs. Functions in matlab are encapsulated, meaning that the variables within a function cannot be seen by the user, and the function cannot see the rest of the workspace. e.g.
	`A = [5 3 4 2 7]`
	`B = rem(A,3)` %% rem is a built in matlab function which returns the remainder of the first argument (A) divided by the second argument (3).
	`B = [2 0 1 2 1]`
	Matlab comes with a lot of built in functions, you may have others from Cogent and SPM, and you can write your own.

script	a set of commands which will be executed one after another in a sequence. Similar to a function, but scripts are not encapsulated and have access to the full workspace.
comment	lines in a script or function which begin with `%` are not executed, they are comments which you can use to remind yourself about what the code is doing.
path	the list of directories where matlab searches for functions. Type `path` to see the path. Matlab will look through each directory in order, and use the first function it finds with the right name. If you have more than one function with the same name, you can type `which` *`name`* to see which version matlab will use. There is a path editor under File-Set Path if you need to change the matlab path.

Glossary - Operators

For more information, type: `help ops, help paren, help punct`

`;` at the end of a line	Suppress the output so that it is not shown to the screen (it is still stored in the matlab workspace)

`;` in the middle of a line	Start a new line of a vector or matrix
	e.g. `A = [2, 3; 5, 7]`
	`A = 2 3`
	`5 7`

`,` in the middle of a line	Break between elements of a matrix (comma or space can be used)
`[ ]` Square brackets	use square brackets to put things into a matrix or to join things together, with commas or spaces between items on the same row, and semi-colons to separate rows
`( )` Round brackets	1) use round brackets to get things out of a matrix or for indexing 2) use round brackets for the arguments of functions 3) use round brackets to determine the order of calculation according to BODMAS
`{ }` curly brackets	{ } use curly brackets for cell arrays which store words etc `B = {'dog', 'cat', 'rabbit'}`
`:` colon	1) On its own, a colon means everything in a row or column `M(:,2)` means everything in column 2 of M `N(3,:)` means row 3 of N, every column A colon in brackets by itself turns anything into a column vector. `M(:)` rearranges everything in M into one long column 2) Between two numbers, the colon means count from A to B `C = 5:8` means C = 5 6 7 8 `M(:,5:8)` means everything in columns 5, 6, 7 and 8 of M 3) A set of three numbers with a colon specifies the step to use for counting `D = 3:4:20` means D counts from 3 to 20 in steps of 4 `D = 3 7 11 15 19`
`...`	continue to the next line without a line break. Useful when lines of code are very long
`.` full stop	1) Decimal Point 2) Element wise maths (see below) 3) Access to fields of a structure
`'hello'` single quotes	Anything between single quotes is treated as a text string by matlab
`=`	equals. Use to assign a value to a variable, e.g. A = 10 puts the variable A into the matlab workspace with the value 10
`+ -`	plus, minus
`* /`	matrix multiply, matrix divide
`.*`	element-wise multiply
`./`	element-wise divide
`^`	matrix power
`.^`	element-wise power (i.e. A.^2 means square each element of A)
`'`	transpose (turn rows into columns and vice-versa)
`==`	Is equal to (logical)
`~=`	Is not equal to (logical)
`~`	not (logical)
`>`	greater than (logical)
`<`	less than (logical)
`&`	and (logical)

`|`	or (logical)
`NaN`	Not a number. Useful for missing data. Use nanmean and nanstd to do statistics which ignore missing data
`Inf`	infinity
`global`	Defines a variable as having global scope, i.e. a global variable in a function is not encapsulated and can be accessed and changed from outside the function. The variable must be declared as global in every function or script where it is to be used.

Glossary - Basic Commands

type help followed by a command name for details

help *command*	show help entry for a particular command
lookfor *word*	search through all help files for a particular word (can be slow)
whos	shows current variables
clear A	remove variable A from memory
clear all	remove everything from memory
size	returns the size of a matrix
max / min	largest / smallest values in each column of a matrix.
mean	average of column
std	standard deviation
sqrt	square root
abs	absolute value of a number, i.e. convert negative numbers to positive
diff	difference between subsequent values of column
sort	sort columns from largest to smallest
sortrows	sort a matrix by the values in a particular column
round	round decimals to nearest integer
find	find the values in a vector which are greater than zero, useful for converting logical indices into subscripts.
rem(a,b)	Find remainder of a divided by b
ones(m,n)	creates a matrix m rows by n columns filled with ones
zeros(m,n)	creates a matrix m rows by n columns filled with zeros
linspace(a,b,n)	generates a vector of n equally spaced numbers where the first is a and the last is b
rand(m,n)	creates a matrix m rows by n columns of random numbers uniformly distributed between one and zero
randperm(m)	puts the integers from 1 to m in a random order. useful for randomising the order of a set of trials
disp(string)	display a line of text to the screen. Useful to tell you what is happening in your script
num2str	convert a number to a string. Useful if you need to display values
repmat	replicate a matrix. Useful to fill a matrix with a particular value

Glossary - Graphics functions

plot	basic plotting of xdata against ydata. See matlab help for line types
title	add title to plot
xlabel / ylabel	add labels to axes
axes	set axis limits
figure	create / select a particular figure
clf	clear a figure (the one most recently selected)

subplot(m,n,p)	create / select subplot p assuming the figure is divided into m by n subplots
cla	clear the currently selected axis / subplot
legend	add a legend to a figure
lsline	add a least squares fit line
text	add text to graph
grid	add grid lines
bar	bar plot
hist	histogram
errorbar	plot with errorbars
image / imagesc	show a matrix as an image or show a picture
colorbar	add a colour scale to your image
colormap	change the colours used in your image
get / set	advanced control of all graphics properties. For example, to set the labels on the x axis of a figure to be 'cat', 'dog' and 'rabbit', use: `set(gca,'XTick',1:3,'XTickLabel',{'cat','dog','rabbit'})` gca means *get current axis* and tells matlab to set the labels on the axis which is currently active XTick is the number of tick marks shown on the x axis XTickLabel is the values attached to each tick mark on the x axis. Use `get(gca)` to see all the things which can be set for an axis

Glossary - Statistics

you need the statistics toolbox for this. Type help stats for a list of all functions

regress	linear regression
ttest	test if the mean of a vector differs from zero
ttest2	test if the means of two samples differ
anova1	basic anova. Matlab in general is not good at complex ANOVAs, it is better to write your data out as tab-delimited text and put it in SPSS to do ANOVAs with repeated measures or more than one factor.