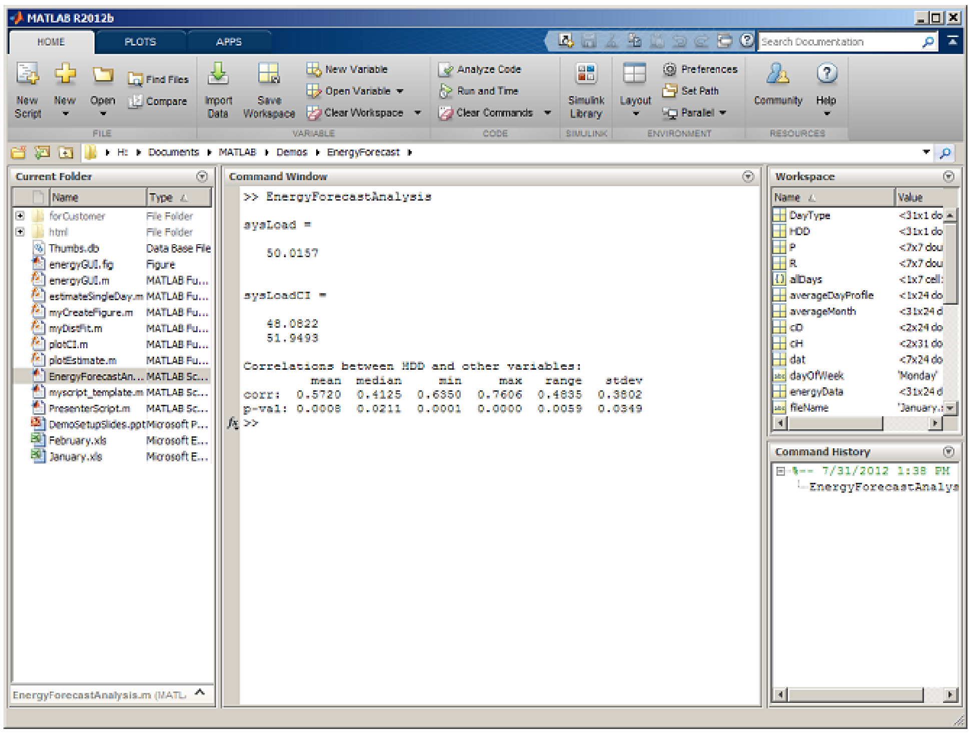
Introduction to Computational Neuroscience
Practice I: Introduction to Matlab/Octave

Ilya Kuzovkin, Raul Vicente

February 4, 2015

A request: Please track how long it will take to complete this set of exercises. Add this time to your final report.

In this course we will use Matlab programming language. The choice is due to the popularity of this language (and the environment) in the neuroscience community. You can use any other programming language if you want to, but some practice session will have basecode in Matlab and data files we will use often come in Matlab format. There are plenty of tutorials online, this practice covers some basics and the most common tasks we will encounter.



```
>> EnergyForecastAnalysis

sysLoad =

   50.0157


sysLoadCI =

   48.0822
   51.9493

Correlations between HDD and other variables:
          mean   median     min     max   range   stdev
corr:   0.5720   0.4125  0.6350  0.7606  0.4835  0.3802
p-val:  0.0008   0.0211  0.0001  0.0000  0.0059  0.0349
>>
```

Figure 1: Matlab main window

On Figure 1 you can see Matlab main window. For Matlab it is very important to know which directory is the current *working directory*. So the first order of business for you is to pick a directory (in the upper part of the screen) where you will complete this practice exercises. Once your unpack the `01 - Matlab.zip` archive you will have `01 - Matlab/code` folder in it. I suggest you point Matlab to that folder.

The main part of the screen occupies the command line, where you can run you commands.

The second essential part of the Matlab is a text editor (which is not shown in default layout). There are also such panels as command log, list of variables in your current workspace and file browser. You can realign the panels to your own liking. The most common way to do work in Matlab is to write you code in editor and sometimes run and test it in the command line.

Now you are all set and we briefly go through basic things you do in any programming language. Read the code and run the pieces which are not immediately clear.

Listing 1 Comments

```
% This is a comment, write comments to explain tricky parts of code and outline
% what is happening
```

Listing 2 Variables

```
% variable can hold any kind of data, you do not need to specify the type
x = 5;

% semicolon at the end of a line suppresses the output, run and compare
>> x = 5
>> x = 5;

% an array is a bunch of numbers in one variable
v = [1, 2, 3, 4, 5];

% a matrix can be defined as
>> B = [[1, 2, 3]; [4, 5, 6]; [7, 8, 9]; [10, 11, 12]; [13, 14, 15]];
% you can take one element form the matrix by specifying row and column
>> B(1, 1)
% or you can take the whole second row
>> B(2, :)
% or the second column
>> B(:, 2)
% or rows 2, 3, 4 and 5 from columns 2 and 3
>> B(2:5, 2:3)
% to transpose a matrix do
>> B'

% You can also have matrixes with more dimensions than two
>> M = repmat(magic(3), 2, 3, 4, 5);
% will produce a 4D matrix of size
>> size(M)
ans =     6     9     4     5
% and you can navigate along each dimension, for example
>> M(2, 1, 4, 3)
% which is powerful, but it is easy to loose track of those dimensions, so
% people often use cells

% Cells are most generic stuctures to store anything
>> C = {'here', 'we', 'add', [1, 2, 3, 4, 5], 'whatever we like', 5};
% and then you can access a certain element as
>> C{4}
ans =     1     2     3     4     5
```

```
% indexing starts form 1
```

Listing 3 Arithmetics

```
% You can add, subtract, multiply, divide numbers
>> 2 + 3 - 4 * 5 / 6
% raise them to power
>> 2^8
% add and subtract vectors
>> [1, 2, 3] + [4, 5, 6] - [7, 8, 9]
% multiply and divide vectors element-wise
>> [1, 2, 3] .* [4, 5, 6] ./ [7, 8, 9]
% multiply matrices
>> [[1, 2]; [3, 4]] * [[5, 6]; [7, 8]]
```

Listing 4 `IF - THEN - ELSE`

```
number = 1;
if number == 1                 % as you can see comparison is done using ==
  disp('The number is one')  % this way you can produce output messages
elseif number == 2
  disp('The number is two')
else
  disp('The number is something else')
end
```

Listing 5 Loops

```
% FOR loop is the main guy
for i = 1:100
  disp(i)
end

% it can also go over elements in a variable (the 'foreach' behaviour)
C = {'here', 'we', 'add', [1, 2, 3, 4, 5], 'whatever we like', 5};
for c = C
  disp(c)
end

% the WHILE loop is also available
i = 0;
while i < 5
    disp(i)
    i = i + 1;
end

% it is often much more efficient to use vectorised code instead of loops
% for example to raise numbers 1 to 100 to power 2 and sum them we can do
total = 0;
for i = 1:100
  total = total + i^2;
end
```

```
disp(total)

% but the Matlab way to do it is
total = sum([1:100] .^ 2)
```

You will see weird functions now and then, like this `sum` in the last example. And you may wonder *what does it do exactly?*. To answer that there is a Matlab documentation `http://se.mathworks.com/help/` where you can enter `sum` and read all about it. Or just Google `matlab sum`.

Listing 6 Creating a function, file `dothings.m`

```
% Each function in Matlab lives in a separate file (you can have several
% functions in one file, but only the first one will be callable from the outside)
% So this is an example of a function, which should be saved as
% a separate file 'dothings.m'

% Matlab function can return several values
function [s, d, p, q] = add(a, b)
  s = a + b;
  d = a - b;
  p = a * b;
  q = a / b;
end

% this function will compute sum, difference, product and quotient of two
% numbers a and b
```

Listing 7 Calling a function

```
% Now once we have our function defined and saved we can call it
>> [s, d, p, q] = dothings(10, 20)
```

Exercise 1: Language basics
Write a function which multiplies two matrices A and B and returns the resulting matrix as an output. If matrix dimensions do not match your program should output appropriate message. In your report please give the full code and the example of how to call the function.

(1pt)

Listing 8 Working with data

```
% There are two great functions in Matlab: 'save' and 'load'.
% Suppose you have very important piece of information
>> important = 'I had a cookie today';
% and you want to save it
>> save('mydata.mat', 'important')
% it will save the variable 'important' into file named 'mydata.mat'
>> clear important     % deletes our variable
>> load('mydata.mat')  % loads it back
>> important           % will print out the content of the variable

```

```
% load() can also deal with formats other that .mat, such as .csv or .dat
>> load ../data/erptrials.csv % assuming 01 — Matlab/code is working directory
% will load a matrix in your workspace
>> size(erptrials)
ans =         79       2000
% where we have 79 recordings of neural data 2000 data points per each
```

Exercise 2: Manipulating data
Load the `erptrials.csv` dataset as described in Listing 8. Create a subsample of observations by taking first 30 rows (and all of the columns) of the `erptrials` matrix. Save the resulting matrix of size 30×2000 as `one.mat` file. The remaining 49 samples save as `two.mat`. Use `clear` command to remove both subsets from your workspace. Now load them both back using `load`. Glue two subsets together to be one whole set as it was before. Assuming that the original dataset is called `erptrials` and the resulting one is called `reconstructed` confirm that they are identical by running `all(erptrials(:)==reconstructed(:))`. Include the code in your PDF file.

(1pt)

Listing 9 Plotting

```
% One of the most important abilities in any data analysis task is the ability to
% visualise the results.

% generate some a random vector
>> data = rand(1, 100) * 10;

% the simplest way to plot is
>> plot(data)

% there are various ways to plot your data depending on what you have there
>> bar(data)
>> hist(data)

% for two dimensional data
>> data = rand(100, 100) * 10;
% you might find useful
>> area(data)
>> imagesc(data)
>> boxplot(data(:, 1:20))
>> scatter(data(1, :), data(2, :))
% etc...

% by the way if some operation is taking too long you can interrupt
% it by pressing Ctrl + C

% By default Matlab will create one canvas (called figure) and plot everything
% on it, if you want to produce several plots and open them each separately you
% should call 'figure(N)' before each one
>> figure(1)
>> plot(data(1, :))
>> figure(2)
>> plot(data(2, :))

% You can also put several plots onto one canvas
>> subplot(3, 2, 1)
% will create a grid with 3 rows and 2 columns and will plot the next picture
% into cell 1 on this grid
>> plot(data(1, :))
% next we want to put into the second cell
>> subplot(3, 2, 2)
>> plot(data(2, :))
% and so on (better done with a FOR cycle)
>> subplot(3, 2, 3)
>> plot(data(3, :))
>> subplot(3, 2, 4)
>> plot(data(4, :))
>> subplot(3, 2, 5)
>> plot(data(5, :))
>> subplot(3, 2, 6)
>> plot(data(6, :))
```

Exercise 3: Plotting

Create a plot of 17th row (all columns) of our `erpdata` dataset. Whenever you produce a plot make sure that it is explained what your plot demonstrates. Go through `plot` function documentation [1] and make the following changes to your plot:

- Add caption, so that everybody would know what this plot is about
- Set X-axis label (try to guess what it is or come up with something)
- Set Y-axis label (try to guess what it is or come up with something)
- Change the color of the line
- Change the type of the line

(1pt)

Please submit a `pdf` report with answers to the questions and comments about your solutions. Include figures, explanations and essential pieces of code. Do not include the code itself as a separate file, your report should give good understanding of what you have done. Please mark how long it took to complete this set of exercises. Upload the `pdf` to the practice session page on the course website.

[1] `http://se.mathworks.com/help/matlab/ref/plot.html`