PRA1004 Scientific Computing 2013

Frans Oliehoek
<frans.oliehoek@maastrichtuniversity.nl>

Lab Assignments Week 2

Overview Lab 1: Introduction Matlab

In this lab session, you will get acquainted with Matlab. Like mathematica, Matlab is a very high level programming environment that offers many predefined functions to use. There are also differences, though were Mathematica focuses on symbolic computation and offers unlimited precision numbers, Matlab uses 'floating point' numbers (which have a limited precision, as you will see) and focuses on high speed numerical computations, with a focus on 'matrix algebra'.

At the end of this lab you should...

- ...be able to understand the Matlab user interface.
- ...be able to import data into Matlab and create simple plots.
- ...understand Matlab as a advanced calculator.
- ...be able to define variables, and use Matlab functions.
- ...know how Matlab can execute scripts, called *m-files*.
- ...have a basic understanding of floating point numbers and their limitations.

NOTE: if you are familiar with Matlab and programming, it is also possible to do some more advanced exercises. Please indicate it if you are interested in this.

1 Getting to know Matlab

First, it is important to get to know the MATLAB user interface.

- Start Matlab.

You will be presented with a windows that looks like Figure 2.1 in the book. (In fact, Section 2.1 in the book gives a pretty good description of the user interface, so you may want to browse through that if you are in doubt.) The main window you are presented with is called the command window. Additionally, there may be windows showing the 'current folder', the 'workspace' and 'command history'.

- Try typing some computations in the command window (e.g. `4*2+3`).
- What is the current folder shown? Can you navigate to a different folder? E.g., your home folder, or a folder that you can use as 'home' (for instance create a folder such as c:\mytempdir and navigate to it).
- Also try the '`pwd`' and '`ls -l`' commands in the command window. What do they do?

If you have typed some arithmetic commands, the workspace will show an entry 'ans'. This is a special variable that stores the output of commands you type, but do not assign to any variable.

- type '`ans`' to see what the value of the variable is.

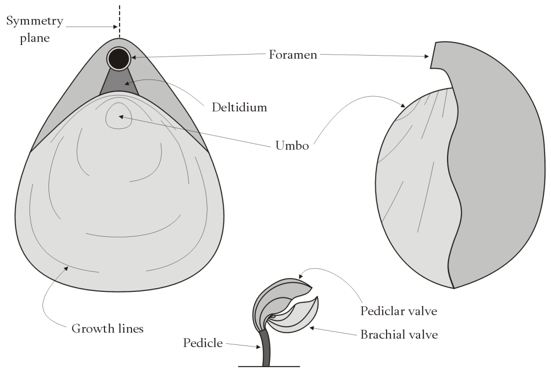

Figure 1: Left: Illustration of a particular type of Brachiopods (Neospirifer condor, source Wikipedia — foto by Dlloyd). Right: Brachiopod morphology. (source Wikipedia — Drawn by Muriel Gottrop in May 2005.)

- type a new arithmetic command, see if `ans` is updated.
- also type `b=3*2`, now `b` should also appear as a variable. Is `ans` updated?
- double click `ans`, this opens up the variable editor, modify the value.
- type `c=ans`, what value does the new variable `c` get?

Further information:

- Section 2.1 in the book gives a pretty good overview.
- If you like getting a overview of Matlab from a video, you can watch the "MATLAB On-Ramp" tutorial located at
  `http://www.mathworks.nl/academia/student_center/tutorials/mltutorial_launchpad.html`
  and try to answer the questions. Of course, just watching these videos it a bit boring. . . try out the things that are demonstrated while watching!

2 Analysis Data Using Matlab and PCA

In this part of the lab, we will load and analyze measurement taken from different types of *Brachiopods*: marine animals that have hard "valves" (shells) on the upper and lower surfaces.

- Clear the workspace.
- Download the labkit and extract it in your working ('home') directory. Make sure that the required files are in your working directory using `ls -l`.
- Now we can load the data
  ```
  >> data = load('Brachiodata.csv')
  ```

You are now presented with 12 data points (12 individuals, the rows) of 10 dimensions (i.e., 10 measurements per individual, each one in a different column). You can see that some colums correspond to 'categorial data' (such as types, or 'low', 'medium' and 'high'), this can be recognized by the fact that a column only contains a small number of integer values. These data are measurements of 6 types of Brachiopods. Which measurements corresponds to what type is stored in a different file.

- You can also load that data via:
  ```
  >> labels = load('Brachiolabels.csv')
  ```

A first good step in exploring any data set is to try and visualize it. However, vizualizing a 10-dimensional data set is not straightforward. Of course is is possible to select some columns and plot the against each other. We will investigate this.

- In order to select a column from the data set, you can use the following command:

  `col3=data(:,3)`

  here,

  - the parenthesis indicate that we are selecting a subset of `data`
  - the colon (:) indicates all rows
  - the 3 indicates the 3rd column
  - `col3` is just the name of a variable (we could also have omitted this '3')
- Select two categorial data colums (e.g., column 1 and 2) and store them in two variables called `colX,colY` (we will plot these on the X and Y axis respectively).
- Now, you can create a scatter plot with the following command
  `>> scatter(colX, colY)`

You may notice that plotting these categorial columns is not very informative. Some data point lie on top of each other, and are hard to discriminate, moreover, there is only a small number of combinations. This is not useful in trying to characterize the 6 types of Brachiopods. Perhaps using the non-categorial data is more useful.

- Create the scatter plot of two non-categorical columns (e.g. 6 and 7).
- This may look more useful, but it is still not clear if it is, since we do not know what point corresponds to what type. We can add the labels to te plot, using the provided function `plotLabels`:
  `>> plotLabels(colX,colY,labels)`

The resulting figure should already look much more informative. However, it still just shows information with respect to two types of measurements. It is unclear whether there are other measurements that may completely change our understanding of this data. One idea could be to try plotting all combinations, but of course this would mean plotting and inspecting all $10 \times 9 = 90$ plots. Instead we examine using *principle component analysis (PCA)*, to reduce the dimensionality of the data, and then plot it. The advantage is that this will give us a overall idea of all the measurements in just one plot.

- You can use PCA reduction to 2 dimensions (the 2 largest principle components) using
  `>> Z = pca(data,2)`
- This results in `Z` being a transformed 2 dimensional data set. Make a scatter plot of it as above, also plot in the labels.
- Take some time to study the plot, does it make sense? Can you identify clearly which parts of the 'transformed' Z-space correspond to what types of Brachiopods? Which types of Brachiopods seem to be more closely related to type 2: type 1 or type 5?

3 Matlab as an Advanced Calculator

Here we return to some more of the basics. We already saw that Matlab can be used as an advanced calculator. Here we elaborate on on this a bit further.

Operators & Priorities

In particular, it is important to know all the basic operators, and their priorities. The basic operators are `+,-,*,/,^` (see Table 2.1 in the book), and they take te same priorities as is normal (`http://en.wikipedia.org/wiki/Order_of_operations`). If you want to evaluate the operators in a different order, you need to group them using parentheses, e.g. `(3+5)/4`.

- Do excercises 16–19, 22–24 of chapter 2.

(If things are unclear, try reading Sect. 2.2.2 in the book.)

Variable Names

So far, we have defined a few Matlab variables. For instance, when we said `b=3*2`, we defined a variable `b` and assigned it the value of `3*2`. Note that the '=' is called the *assignment operator*, and that it is quite different from the mathematical definition of '=' (equality is also known to Matlab, but written as '`==`'. We will see this in a later lab).

It is important to note that the variable names are case sensitive. Also, not all variable names are valid: they must start with a letter, and can only contain letters, digits, and the underscore '_'. Also, certain reserved words (called keywords) are not allowed; see section 2.1.1 in the book.

- Show this by defining `B` and showing it has a different value than `b`.
- Say we would want to compute with some prices of products, e.g., product 1, 2, and 3. Think of a valid name for a variable that should contain the price of the first product. Also think of an invalid name.
- Test your answers using the Matlab function `isvarname`. For instance to test if 'my_var_name' is valid, you could type `isvarname('my_var_name')`. (Note that in Matlab 'False' is encoded by 0 and 'True' by 1) . Also try typing `help isvarname`, try to understand the explanation.
- Delete all current variables by typing `clear`.

Using Matlab Functions

Like Mathematica, Matlab has *many* predefined functions. Here we will introduce a number of important functions, and practice using them.

`help` This is probably the most important function in Matlab. It will print information about the other functions. (For instance, try it on the other functions mentioned in this section.)

`disp` This function allows you to display the value of variables, as well as printing 'strings' to the command window.

- For instance try defining `p1=42`, then type `disp(p1)`, this should display the value of the variable `p1`.
- A 'string' is a computer science term for a piece of text. If you think about it, a piece of text is just a string of characters, which explains the terminology. Strings are encoded using single quotes (') in Matlab, so `'this is a string'` is a string, as well as `'42'`. Try typing `disp('the answer is'); disp(42)`.
- **NOTE:** There is a difference between straight quotes (') and curly quotes (‘ or ’). In Matlab you should always use straight quotes. (even if the excercise documents might not always do this consistently.)

`abs` Used to get the absolute value.

- Try `abs(-4)`

`sqrt` Computes the square root.

- Try `sqrt(2)`.

Section 3.2 lists a number of other useful functions. Have a quick look such that you know of the existence of these functions.

Using Scripts

Using the Matlab functionality by directly typing commands in the command window is useful, but it makes it hard to repeatedly run sequences of commands, or to share your analysis. Scripts overcome this problem. Essentially, a script is just a text file which lists the commands that Matlab should execute. Therefore, to edit scripts, you can just use a simple text editor (such as notepad). Matlab, however, includes a text editor with some useful functionality. For instance, it highlights certain keywords, takes care of nicely 'indenting' your code, and has functionality to 'debug' your code (don't worry if you don't know what these things are).

In this part of the assignment, we will create a simple script and show how it can be run.

- Select file->new->script. This should open up the Matlab editor.
- Save your file as script1.m (in your 'home' directory).
- edit the script by adding the line `disp('This is script1.m')` at the top.
- You can now run the script in various ways:
  - press F5
  - click the green 'play' icon.
  - in the command window type `script1` (make sure that the current working directory is the one containing the script!)

  try all of these.
- Extend the script to (again) compute the number of inches in one light year.

In scripts, it is common to write 'comments': text that is not meant to be executed, but that explains the lines of code. You can indicate that a line is a comment by starting the line with a percentage (%) sign.

- Extend your script such that it computes and displays (use `disp`) the following things:
  1. an estimate of the volume of the class room
  2. an estimate of the volume of air per person in the class room
  3. an estimate of the time that all the air in the class room has been breathed in when each person breaths 8 litres per minute.

  Remember to create variables with meaningful names. Also add comments (using '%') to complex lines.
- Save your script, you should show it for examination.

4 Numbers in Matlab: The Floating Point Number System

As you may have noticed, Matlab represents numbers in a different way than Mathematica. In particular, Matlab uses floating point numbers, as illustrated in figure 2. These can 'overflow' if the become to big and 'underflow' they become to small. (See also p. 40 in the book.) While for many practical purposes, floating point numbers are just as convenient as the (computationally much more costly) arbitrary precision representation offered by Mathematica, it is good to know when thing may go bad. Therefore we will walk through a few such examples.

- First we have a look at *overflows.* Overflow means that the number is just too large to represent. For instance try typing `3e15092` in Matlab. Looking at figure 2, we can imagine that this happens when the exponent e is larger than 1025. However, now try `2e200 * 3e200`. Can you explain why this already overflows?

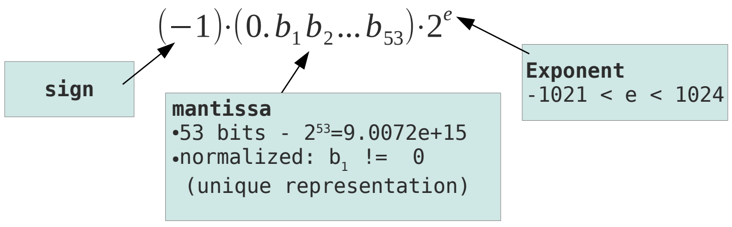

Figure 2: Floating point representation in Matlab.

- Next, we consider *underflows*. These happen when a number is too small to represent. For instance `142.2 / 3e400` underflows. Try it to see what its result it.
- To understand what is going on, it is good to think a bit more about the actual representation of the numbers. For instance, think about the floating point representation of "0.5". What are $b_1 \dots b_{53}$? What is e?
- Matlab can display more digits of the floating point numbers. To get it to do so, type `format long` (see section 2.3 in the book for other format options).
- Compute the 15 digits (in decimal) of the smallest non-normalized floating point number greater than 0. You can do this by raising 2 the the appropriate power.
  (the first few digits are 4.9407 and the exponent is $e = -324$.)
- Compute
  - `1e-4 + 1 - 1`
  - `1e-20 + 1 - 1`

  and observer the differences, think a bit about why this happens. Is this an instance of overflow or underflow?
- Of course, you can imagine that computations like `-1e-20+1-1` may lead to suprises, but that for such small numbers you typically may not notice. It still is important to be aware of these things, however. For instance, in Matlab `==` tests whether the left hand side and right hand side are equal.
  - E.g., try `3==1` and `4==4`. (Matlab follows the standard convention that 0 means false and nonzero (e.g., 1) means true)
  - Now try the following:
    ```
    a=1/49
    b=49*a
    b==1
    ```
    Can you explain what is going on?