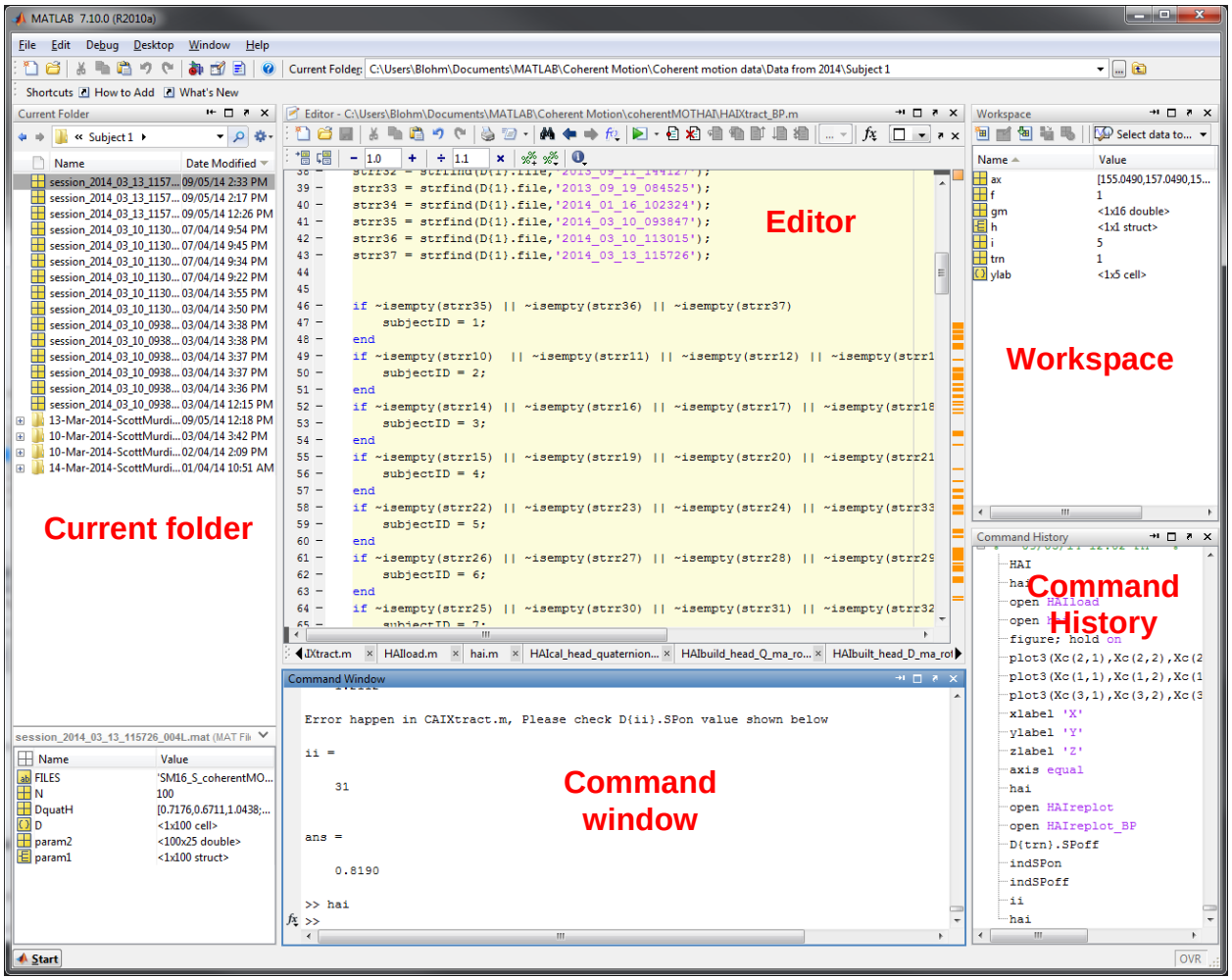
Getting started with Matlab

Components of the desktop:



Current folder: this is where Matlab will load and save files from if you don't specify the full file path with these commands. It is also useful for previewing file contents (lower pane) when loading files manually. For example, if I wanted to load the "session_2014_03_13_1157..." .MAT file highlighted in the Current folder, I could first preview what objects the file contains by single-clicking, then load it into the workspace by double-clicking. You can see that this file contains the objects/variables "FILES", "N", "DquatH", "D", "param2" and "param1". The yellow symbols next to each name tells you what type of object each is – whether it's a string (of text) like "FILES", a double (like a matrix) like "N", a cell like "D" or a structure like "param1". Each object has its specific advantages, which are unimportant for now. We'll focus mostly on doubles.

Workspace: This shows what variables are being held in Matlab's memory for use in computations. Objects contained in files you load and those you generate using commands, functions and scripts will appear here.

Command window: This is one interface between the user and Matlab. Like the name implies, you tell Matlab to do different commands and the it carries out the required computations, line by line. Simply enter commands such as “spy” or “why” for a nice surprise :-)

Editor: This is another useful interface between the user and Matlab. Here, you can enter commands exactly like you would in the command window, but you can save your work. This is where you’ll be creating analysis programs to manipulate and visualize your data.

Executing commands

Basic calculation operators:

+ addition

- subtraction

* multiplication

/ division

^ exponentiation

. element-by-element operation (use only in conjunction with matrix multiplication, division and exponentiation operators)

Expression language

- Expressions typed by the user are interpreted and evaluated by the Matlab system
- Variables are names used to store values
- Variable names allow stored values to be retrieved for calculations or permanently saved
- E.g.: “*variable = expression”* or “*expression”* are acceptable syntax
    1. User types: x = 6
        - Matlab returns: x = 6
    2. User types: y = 2
        - Matlab returns: y = 2
    3. User types: x + y
        - Matlab returns: ans = 8
    4. User types: x * y
        - Matlab returns: ans = 12
    5. User types: x / y
        - Matlab returns: ans = 3
    6. User types: x ^ y
        - Matlab returns: ans = 36

Working with matrices

- Matlab works with matrices
- A matrix is a collection of numerical values that are organized into a specific configuration of rows and columns
- The number of rows and columns can be any number (e.g. 3 rows and 4 columns define a 3 x 4 matrix having 12 elements)
- When describing a matrix, always state the **R**OWs first, then the **C**OLUMNs
    - Just think of generic US soda brand **RC Cola**!

- A **scalar** is a single number and is represented by a 1x1 matrix (e.g. c = 5)
- A **vector** is a one dimensional array of numbers and is represented by an nx1 column vector or a 1xn row vector of n elements
    - E.g. x = [4 5 2 6] (row vector)
    - E.g. x = [4
      5
      2
      6] (column vector)

    - E.g. X = [1 2 5 6
      3 6 3 8
      7 8 2 1] (3x4 matrix)

- Spaces, commas and semicolons are used to separate elements of a matrix
- Spaces or commas separate elements of a row
    - [1 2 3 4] or [1,2,3,4]
- Semicolons separate columns
    - [1,2,3,4; 5,6,7,8; 9,8,7,6] = [1 2 3 4
      5 6 7 8
      9 8 7 6]
- **Matrix multiplication:** built-in matrix multiplication in Matlab is either (1) algebraic dot product or (2) element-by-element multiplication (use the .* operator!!!)
- These same rules apply for **matrix division** and **matrix exponentiation** as well (i.e. that there's the algebraic version and then the element-by-element version using the . operator!)

Indexing matrices

- An **m x n** matrix is defined by the number of m rows and number of n columns
- An individual element of a matrix can be specified with the notation X(i,j) for the generalized element, or by X(3,1)=9 for a specific element.
- E.g.
  >> X = [1,2,3,4; 5,6,7,8; 9,8,7,6]
  >> X(1,2)
  ans = 2

- The colon operator can be used to index a range of elements
  >> X(1:3,2)
  ans = [2 6 8]
- You can also overwrite elements in specific locations within a matrix this way
  - E.g.
    >> X(1,2) = 8
    X = [1 **8** 3 4; 5 6 7 8; 9 8 7 6]
- You can transpose a matrix simply by adding ‘ to the end of your matrix definition
  - (the transpose of a matrix is formed by interchanging the rows and columns of a given matrix)
  - E.g.
    >> X’
    ans = [1 5 9; 8 6 8; 3 7 7; 4 8 6]
    (so X went from being 3x4 to 4x3)
- **The FIND function**
  - The ‘find’ function is extremely helpful for finding the elements of a matrix meeting a certain criteria
    - E.g.
      >> find(X < 5)
      ans = [1 7 10]
      (meaning that the first, seventh and tenth elements of matrix X are less than 5. The elements are counted along each column, starting in top left and working down towards the bottom right)

Matlab scripts

Entering multiple lines without running them

- Matlab can execute sequences of commands stored in files
- Files that contain Matlab commands have the extension ‘*.m’
- *.m files are written and saved in the built-in Matlab m-file editor (remember the Editor window from the desktop?)
- M-Files are executed from the M-file editor or the command prompt (remember the Command window from the desktop??)
- M-files can call other M-files, given that the called M-file is somewhere in Matlab’s path (see File->Set Path)
- **Advantages of M-Files:**
  - Easy editing and saving of work
  - Undo changes

- Readability/portability – non executable comments can be added using the '%' symbol to make commands easier to understand
- Saving M-files is far more memory efficient than saving a workspace (remember the Workspace window??!)

Conditional statements and loops

- **Conditional statements**
  - It is often useful to only perform Matlab operations when certain conditions are met
  - In these conditions, 'if', 'else' and 'elseif' statements might be useful
  - E.g.

```
A = 2;
if A > 2
   A = 5;
elseif A < 0
   A = 8;
elseif A == 3
   A = 0;
else
   A = 500;
end
```

    (results in A = 500, get it? NOTE: '==' is the logical way of saying something is equal to something else, and is used in conditional statements)
- **Loops**
  - Loops enable Matlab to repeat multiple statements in specific and controllable ways
  - Mainly use 'for' and 'while' loops (we won't cover 'while' here)
  - **For:** the for loop executes a statement or group of statements a predetermined number of times
  - Basic form:

```
for index = start:incremend:end
     statements
end
```

    ** if '*increment'* is not specified, an increment of 1 is assumed by Matlab
  - E.g.

```
for i = 1:1:100
     x(i) = 0;
end
```

    (assigns 0 to the first 100 elements of vector x; if x does not exist or has fewer than 100 elements, additional space will automatically be allocated)
  - Loops can be nested in other loops

Functions

- In Matlab, each function is a .m file

- Basic syntax:
    - When naming a function in an .m file:
      function *output_arguments = func_name(input_arguments)*;
    - when calling a function, simply type:
      output_arguments = func_name(input_arguments);
- When you don't know exactly what inputs or outputs a function takes or gives, access its Help menu entry either by typing 'help *functionname'* in the command window, right clicking on the function and choosing 'help' or by highlighting the function and pressing F1. Unlike much other software (*cough* MS Office *cough*), the Help menu in Matlab is EXTREMELY useful. When I'm using Matlab it is constantly open.

Plotting

- One of the most useful things Matlab can do is help visualize your data.
- For this, the **figure, subplot, hold on** and **plot** commands are invaluable
- First, create a figure window by typing 'figure'
- Within this window, you can plot your data using 'plot', or you can create multiple plots using 'subplot'
- If you're plotting multiple datasets on one set of axes, you will want to keep what you've already plotted on the axes by typing 'hold on' (otherwise new plot commands will simply overwrite the old data)
- E.g.

```
X = 0:2:100;
Y(:,1) = X*5;
Y(:,2) = X*3;
Y(:,3) = X;
figure;
for i = 1:size(Y,2)          % size(Y,2) gets the number of columns in matrix Y
    plot(X,Y(:,i));
    hold on
end
```

- And it's also useful to stylize your plots by modifying the marker types, color, lines, etc. when you plot. To do this using a built-in shortcut, simply type "plot(X,Y,*modifier*)" where *modifier* can be any combination of things from the following list (there may be more that I'm missing too, see the Help menu!!):
    - Colors: r,g,b,k,w,y,c,m
    - Lines: -,--,:
    - Markers: .,o,v,x,s,^,+,d
- Because each thing you plot is saved as an object (with a 'handle' *obj_handle*), you can access every property of the objects you've plotted using the get(*obj_handle*) command.
- Now, using our example from above:
- E.g.

```
X = 0:2:100;
Y(:,1) = X*5;
Y(:,2) = X*3;
```

```
Y(:,3) = X;
modifier = {‘ro-’ ‘g:s’ ‘b--v’};
figure;
for i = 1:size(Y,2)              % size(Y,2) gets the number of columns in matrix Y
        obj_handle(i) = plot(X,Y(:,i),modifier{i});
        hold on
end
```

- Now if you type ‘get(obj_handle(1))’ in the command window you can see all the editable properties of the first line you plotted!

One last thing:
Use **ctrl+c** to abort any currently running code!

Quick 3D plotting tutorial:
Once you've gone through the above exercise, try the following exercise, which creates a 3-D surface plot of pseudo-random numbers (then try it again, with a 100x100 matrix!!):

*** Use the Help menu within Matlab to see the proper syntax for using functions; for example, the Help menu says that the function "ones", which is used to create a matrix of only ones, takes at least one input, representing the length of each dimension. ones(3) will create a 3x3 matrix of 1s, while ones(3,1) will create a 3x1 vector array of 1s. ***

*** Another note: to assign the output of a command line to a variable, you must write "x = ... " prior to entering the command. This will assign the output of the command line to the variable "x". You can either do this exercise in the command window line by line (hit enter to run each command), or you can create a m-script using the editor (click 'run' or press F5 to run the script when ready). Both are identical command-wise, but the m-script will let you save your progress. ***

1. One of the 'specialized' matrices that the tutorial mentions is called "magic". Using this command, create a 10x10 matrix called A.

2. Square each element of the matrix A.

3. Multiply each element of A by a 10x10 matrix containing values randomly (and uniformly) distributed between 0 and 5. HINT: You'll have to search the Help menu for the command used to generate a uniform, random matrix, then increase the width of its output interval (the usual interval is between 0 and 1).

4. Using the "meshgrid" command, generate a 2-D grid of points between 1 and 10. HINT: this function takes vectors representing the points along each axis that you want to plot. Because we are going to plot A in 3-D space, each input vector for the meshgrid

command must have the same number of elements as either the horizontal or vertical dimension of A.

5. Type "figure" to define a new figure window. Follow with "hold on" to tell Matlab to keep plotting in this window.

6. Use the "surf" command to plot the outputs of meshgrid and A on a 3-D surface.

7. Type "view(3)" to get an orthogonal view.

8. Add an x-label, y-label and z-label to the plot using a command for each.

9. Pretty picture! You can rotate the view using the rotate tool on the figure toolbar near the top of the window.

Extra Matlab help:
Youtube: Matlab tutorials for beginners, parts 1 and 2 with Jake Blanchard
Part 1: https://www.youtube.com/watch?v=l6fzNfwCpdA
Part 2: https://www.youtube.com/watch?v=TK_TC3pG0Vc

Youtube: General plotting
https://www.youtube.com/watch?v=cyxFsSJSxwE

UMass intro to Matlab tutorial
https://www.math.umass.edu/~kevrekid/math697/mattutorial.pdf

Matlab Central: Cody Challenges – work through exercises with a community to find the most optimal solutions to common problems!
http://www.mathworks.ca/matlabcentral/cody