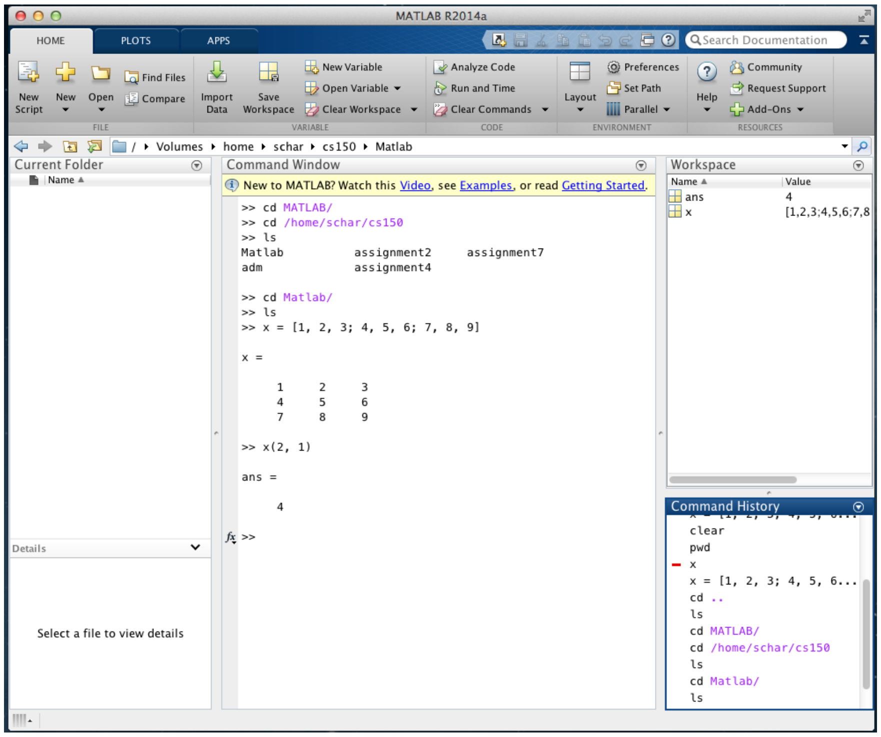
CS 150 - Getting to know Matlab

This in-class lab will walk you through some of the functionality of Matlab. For more information and resources, see the course web page.

1 Getting started

To start Matlab, type "command + spacebar" and then type "matlab" to search for Matlab. The icon for Matlab looks like:

Matlab has its own IDE. The IDE has a number of different parts to the window:



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>> cd MATLAB/
>> cd /home/schar/cs150
>> ls
Matlab          assignment2     assignment7
adm             assignment4

>> cd Matlab/
>> ls
>> x = [1, 2, 3; 4, 5, 6; 7, 8, 9]

x =

     1     2     3
     4     5     6
     7     8     9

>> x(2, 1)

ans =

     4

>>
```

Name	Value
ans	4
x	[1,2,3;4,5,6;7,8

```
clear
pwd
x
x = [1, 2, 3; 4, 5, 6...
cd ..
ls
cd MATLAB/
cd /home/schar/cs150
ls
cd Matlab/
ls
```

The center frame is the interactive shell (with `>>` as the prompt). This is where we will be typing all of our commands. The left frame shows the files in the current working directory. The top right shows all the active variables (to start with, none) and the bottom right shows the history of commands typed (if it doesn't, select "dock" for "Command History" from the Layout menu).

2 Basics

Matlab's behavior in the interactive shell is very similar to the behavior we've seen with Python. Play with the shell some and try to discover the answers to the following questions:

- What mathematical operators does matlab support? (mod/remainder is not an operator, but a function called `mod`)
- What does Matlab do when dividing two integers that don't divide evenly?
- What is the variable `ans`? Can you use it in an expression?
- How do you assign to a new variable? Notice that when you create new variables in the workspace, they are displayed in the upper right frame.
- Statements can optionally be terminated by a semicolon (`;`). What difference does this make? *Hint: it can make a difference!*

A few other things to try out with Matlab:

- To get help from Matlab type `help <function>`. Try getting the help for a few things (even things like `ans` and operators).
- Comments in Matlab are indicated by a % sign. For example, type:

  ```
  >> x = 5 % this assigns 5 to x
  ```

  Just like in Python, everything following the % is ignored
- Strings in Matlab are indicated with single quotes. What happens if you try to use double quotes?
- To print out things to the screen, use the function `disp`. Notice that unlike Python, there is a difference between printing/displaying a variable or expression and just typing it at the terminal. How does `disp(5)` differ from just typing 5? If you want to get fancy, look at the help documentation for `sprintf`.
- Notice that all of the commands that you type show up in the lower right frame "Command History". You can double click on any of these and it will rerun that command in the shell.

3 Matrices

The real power of Matlab comes from its ability to generate and manipulate matrices. **All** objects in Matlab are matrices or special cases of them like scalars (1 by 1 matrices) and vectors (1 by n or n by 1 matrices).

Creating matrices

There are lots of different ways that you can create matrices. Try out the ones below. Remember that you can use `help` for the functions to figure out what they do:

- Just like Python, you can create a 1-dimensional *row* vector (i.e. a list) using square braces:

  ```
  >> [1, 2, 3, 4, 5]
  ```

  (In Matlab the commas are optional, so you could just type `[1 2 3 4 5]`).

- You can also create a 1 dimensional *column* vector using square braces:

  ```
  >> [1;2;3;4;5]
  ```

  What is the difference between these two?

- Matlab has a built-in operator `:` to create vectors representing a range of values:

  ```
  >> 2:10
  ```

  This has a similar effect to the `range` function in Python.

  If we want to use a different increment (other than 1), we can put in another colon:

  ```
  >> 2:.5:10
  ```

  This can also include negative numbers if you want to count in the other direction:

  ```
  >> 10:-1:2
  ```

- Combining the techniques above for making row and column vectors, we can make a matrix:

  ```
  x = [1, 2, 3; 4, 5, 6; 7, 8, 9]
  ```

- There are also a number of built-in functions for creating matrices:

  - `zeros`
  - `ones`
  - `rand`
  - `eye`
  - `magic`

- You can create more elaborate matrices by combining simpler matrices. For example, what would the following matrix look like:

  ```
  >> [eye(2), zeros(2); zeros(2), eye(2)]
  ```

  A comma concatenates to matrices column-wise, while a semi-colon concatenates them row-wise.

  (Think about your answer first, then check it with Matlab)

Once you're comfortable creating matrices, try to create the following matrices with just a single statement (do NOT just simply type in the matrix as a literal :)

- A 10 by 10 matrix where each entry is 10
- The following matrix:

```
  1     2     3     4     5     6     7     8     9    10
 10     9     8     7     6     5     4     3     2     1
```

- The following matrix:

```
  1     1     2     2
  1     1     2     2
  3     3     4     4
  3     3     4     4
```

- The following matrix:

```
  1     2
  1     2
  1     2
  1     2
  1     2
  1     2
  1     2
  1     2
  1     2
  1     2
```

Matrix manipulation

Most operators and functions work over a matrix. Try these out and discover the answer to the questions.

- *indexing*: You can access individual elements of a matrix or vector using parentheses `()`. This is one of the most common mistakes to make for Matlab vs. Python. If you try to use square braces `[]`, you will get an error. For vectors, you just specify an index. For matrices, you must specify a row and a column, separated by a comma:

  For vectors:

```
>> x = 5:-.5:2
>> x(3)
```

```
ans =

     4
```

Does indexing start at 0 or 1?

For matrices:

```
>> x = magic(3);
>> x(2, 2)

ans =

     5
```

If you index a matrix using just one index the indices start in the first column and work down, then move to the top of the second column, etc. For example, the indices for the 3 by 3 matrix are:

```
1 4 7
2 5 8
3 6 9
```

Try it out on the magic matrix:

```
>> x = magic(3);
>> x(2)
>> x(5)
>> x(8)
```

- *slicing*: We can slice matrices just like in Python with the following differences:
  - Slicing (like indexing) uses parentheses.
  - You cannot use the one-sided colon to mean got to beginning or end (e.g. `[1:]`). However, you can use the keyword `end` when slicing to mean the end of the vector, e.g. `x(3:end)`, grabs everything from 3 to the end.
  - You can slice rows *and* columns

  See if you can figure out what the following would return and then check your answers by typing them in:

```
>> x = 1:20
>> x(5)
>> x(4:end)
>> x(15:-1:5)
>> x = magic(3)

x =

     8     1     6
     3     5     7
     4     9     2
>> x(1:2, :)
>> x(:, 1:2)
>> x(1:2, 1:2)
>> x([1, 3], :)
>> x(2:end, 3)
>> x(:)
>> x(1:2)
>> x(5:end)
```

- *size/length*: The `length` function returns the number of elements in a vector. The `size` function returns two values, the number of rows and columns of a matrix or vector. What does `length` return on a matrix? What does `size` return on a vector?

  You can "unpack" the values returned by `size` in a similar way that we do in Python (except `size` returns a matrix instead of a tuple):

  ```
  >> [nrows, ncols] = size(x)
  ```

- *math operators*: All of the math operators work an entry at a time if you apply some operator between a matrix/vector and a scalar. Try multiplying, dividing and subtracting scalars from matrices and vectors.

- *matrix math*: Try subtracting or multiplying two matrices together. Do they do what you would expect?

  ```
  >> x = ones(3)
  >> x - x
  >> x * x
  ```

  When multiplying two matrices, Matlab does matrix multiplication (not element-wise multiplication). If you want element-wise multiplication use `.*`:

  ```
  >> x = ones(3)
  >> x .* x
  ```

  Similar "dot" operations exist for `^` and `/`.

- *transpose*: Often we want to transpose a matrix (or vector), that is, make the rows the columns and the columns the rows. In Matlab, this is done with the `'` operator, which is put *after* a matrix or vector. Try out the following:

```
>> x = [1, 2, 3]
>> x'
>> magic(3)
>> magic(3)'
```

Note: You won't be able to copy and paste these since the single quote doesn't copy and paste right. Just type them in.

- *misc. functions:* Matlab has a number of other built-in functions that are useful. Figure out what they do and answer the questions below:
  - `max`: How can you get the max of a matrix? (There are actually multiple ways of doing this)
  - `mean`: How can you get the mean of a matrix?
  - `sum`: The rows, columns and diagonals of a `magic` matrix should all sum to the same number. Verify this for yourself.
  - `sqrt`
  - And many more ...

4 Control Structures

Matlab has all of the control constructs that we've seen in other languages (`if/else`, `for` loops and `while` loops). Matlab does NOT use indenting to denote a block of code like Python. Instead, Matlab uses the `end` keyword to denote the end of a block. For example:

```
if x > 0
    % do something here if x > 0
end

% do something here regardless of whether x > 0 or not
```

Even though indenting isn't required, we will still indent it to make it more readable.

Below are some of the other differences for each of the control constructs

- if/else
  - `if` by itself:

    ```
    if x > 0
        % do something here if x > 0
    end

    % do something here regardless of whether x > 0 or not
    ```

  - `if/else` statement:

    ```
    if x > 0
        % do something here if x > 0
    else
        % do something if x > 0 is not true
    end

    % do something here regardless of whether x > 0 or not
    ```

    Notice that the `else` terminates the `if` block and the `end` terminates the `else` block.

  - `if/elseif/else`

    ```
    if x > 0
        % do something here if x > 0
    elseif x < 0
        % do something if x < 0
    else
        % do something if none of the if/elseif statements are true
        % this part is optional, though we'll commonly include it
    end
    ```

    As with the `if/else` statements, the `elseif` acts as a block terminator.

- while

  While works just like it does in Python except it is terminated by an `end`.

  ```
  while x > 0
      % do something as long as x > 0
  end
  ```

  Matlab has variables `true` and `false` to represent true and false. However, Matlab *does not actually have a boolean type.* Instead `true` and `false` are just numbers. Try to figure out what values true and false actually represent.

- for

  The for loop is similar to Python except instead of the keyword `in` we use `=`. Unlike python, we don't need to use a function like `range`, since we can use the colon operator. For example, to print out the numbers 1 through 10:

  ```
  for i = 1:10
      disp(i)
  end
  ```

  Like our other constructs, we terminate the for loop block with the `end` keyword.

  Loops in matlab are *much* slower than doing manipulations over vectors or matrices, so it's generally a good idea to avoid loops if you can.

5 Logical expressions

Matlab has all of the standard boolean logic operators we've seen and a few others:

- && (instead of `and`)
- || (instead of `or`)
- & (`and` applied to a vector or matrix)
- | (`or` applied to a vector or matrix)
- ~ (instead of `!`)
- `any`
- `xor`
- `all`

Look up these last three using `help`

What would the following return (and why)?

```
>> 10 & 1
>> 10 && 1
>> 1 && ~1
>> true && false
>> true || false
>> true & ones(1,10)
>> true && ones(1,10)
>> ones(1,10) & ~zeros(1,10)
```

Notice that && and || can only be applied to scalars.

Like the mathematical operators, we can also apply the comparison operators (`>`, `>=`, `<`, `<=`, `==`, `~=`) to a matrix:

```
>> x = magic(3)
>> x > 3
>> x == 4
```

Try to write statements that do the following:

- Count how many elements in a 5 by 5 magic square are greater than 5

- Generate 5 random numbers using `rand` and see if they were all greater than 0.1. The statement should return 1 if they are all greater than 0.1 or 0, otherwise. (note there are multiple ways of doing this).
- Generate 5 random numbers using `rand` and see if *any* of them are greater than 0.9. The statement should return 1 if any one of them is greater than 0.1 or 0, otherwise.
- Out of 100 randomly generated numbers between 0 and 1, count how many are less than 0.2.
- Out of 100 randomly generated numbers between 0 and 100, calculate the sum of all of the numbers that are greater than or equal to 50. First, generate the random number vector/matrix as one statement, then get the sum as a second statement. *Hint: the .* operator might be useful.* Also, be careful about order of operations (comparison operators are lower priority than multiplication).

6 The working directory

Matlab has a current working directory where it looks for files you've written. The nice thing about Matlab is that it uses all of the same commands to navigate around as those we used in the terminal:

```
>> pwd

ans =

/home/schar/cs150/Matlab

>> ls   % I don't have any files in this directory
>> cd .. % move up a directory
>> ls
adm  assignment2  assignment4  assignment7  Matlab
```

When you change directories, you'll also notice that the left frame will be updated to show the contents of the directory.

Change directories so that you're on your Desktop.

You can create a new directory/folder (both in Matlab and in Terminal) with the `mkdir` command (short for make directory):

```
>> mkdir matlab
>> cd matlab
```

Now your working directory should be in a folder called `matlab` on your Desktop. Just FYI, if you want to remove/delete a directory, use the `rmdir` command, however, the directory must be empty.