MATLAB Cheat Sheet

Miscellaneous useful notes

1. Text after a percent sign (%) represent **comments**, and are ignored by the code.
2. Text in parentheses is evaluated by MATLAB prior to other text, in a manner identically to how parentheses are treated in math.
3. The two basic windows in MATLAB for the user to work in are the **Command Window** and the **Editor**. The Command Window is for quick commands that the user wishes to see executed; it acts out one line of code at a time, and prints the result unless that result is suppressed by the user. The Editor is for writing larger pieces of code in the form of functions or scripts, both of which are described toward the end of this document.
4. To suppress printing of command output, a simicolon (;) can be placed after the command. E.g., `x=4` will assign the value 4 to `x`, and print `x=4` to the screen. `x=4;` will assign the value 4 to `x`, without printing `x=4` to the screen.
5. To clear variable assignments, type `clear nameOfVariable`. E.g., `clear x` will make it so that `x` no longer has any assigned value. To do this for all variables, type `clear all`.

Data types

The most important data types in MATLAB are **doubles**, **strings**, and **Booleans**; these are MATLAB's default numeric, text, and logical data types. Below, examples of each of these are given - `doub1` and `doub2` are doubles, `str1` and `str2` are strings, and `boo1` and `boo2` are Booleans.

```
doub1=3.0;doub2=1;
str1='hello';str2='h';
boo1=false;boo2=5>2;
```

Arrays

An **array** is a list of data types. It can be multidimensional, though MATLAB works most easily with 1- and 2-dimensional arrays. Arrays can be defined with in several ways. Below, `r1` and `r2` each represent a single-row array - `r1` is defined explicitly, and `r2` is defined with a handy MATLAB notation. `c1` and `c2` are similarly defined single-column arrays. `M1` is a 2-dimensional array.

```
r1=[1,2,3,4,5]%Commas to separate entries in an array within a row.
r2=1:2:9%This is equivalent to writing r2=[1,3,5,7,9]. The notation is start:step:stop.
    %A notation start:stop can also be used, which simply means start:1:stop.
c1=[1;2;3;4;5]%semicolons to separate rows.
c2=(1:2:9)'%The ' symbol means "transpose", i.e., switch the role of rows and columns.
M1=[1,2,3;4,5,6;7,8,9]%M1 is a 2-dimensional array.
```

To select an element of an array, the format is `ArrayName(index)` for a 1D array, and `ArrayName(rowIndex,colIndex)` for a 2D array. The lowest array **index** in MATLAB arrays is the number 1.

```
r2(2)%Gives 3, since the second entry in r2 is 3.
c2(4)%Gives 7, since the fourth entry in r2 is 3.
M1(3,2)%Gives 8, since M1(3) is [7,8,9] and [7,8,9](2) is 8.
```

Finally, arrays can be **concatenated**, which means added onto. In the other direction, **subarrays** of a given array can be selected. There are variations on both of these operations, but the most basic form of each is demonstrated below - more detail can be found online.

```
rbig=cat(2,r1,r2)%cat(2,array1,array2) concatenates array1 and array2 row-wise.
    %This gives rbig=[1,2,3,4,5,1,3,5,7,9].
cbig=cat(1,c1,c2)%cat(1,array1,array2) concatenates the two specified arrays column-wise.
    %This gives cbig=[1;2;3;4;5;1;3;5;7;9].
r2small=r2(2:4)%r2small=[3,5,7].
```

Operators

MATLAB's basic **arithmetic operators** are addition (+), subtraction (-), multiplication (*), division (/), and exponentiation (^), and modulo (`mod(a,b)`). These are used to combine several numbers or arrays to get a new number or array. By default, these operations act on arrays as they would act on matrices for matrix multiplication. This does not matter for addition, subtraction, or modulo, but does matter for the others. To perform the same operation element-wise, put a dot in front of the operation (`.*`, `./`, `.^`). Examples are demonstrated below.

```
2+3%outputs 5.
2*3%outputs 6.
r1+r1%outputs [2,4,6,8,10].
r1*r1%outputs error, since dimensions aren't consistent for matrix multiplication.
r1.*r1%outputs [1,4,9,16,25] by element-wise multiplication.
r1*c1%outputs 55, since the matrix multiplication gives 1*1+2*2+3*3+4*4+5*5=55
c1*r1%outputs [1,2,3,4,5;2,4,6,8,10;3,6,9,12,15;4,8,12,16,20;5,10,15,20,25],
    %since this is what the matrix multiplication gives.
[1,1;1,1]^2%outputs [2,2;2,2] by matrix multiplication.
[1,1;1,1].^2%outputs[1,1;1,1] by element-wise exponentiation.
```

MATLAB's basic **relational operators** are equals (==), greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), not equal (=), and is equal(`isequal(M1,M2)`). All of these give a Boolean of true (1) or false (0) as they would evaluate to mathematically. The `isequal()` function is defined specifically for arrays, and gives true if its array arguments are equal to one another. Examples are demonstrated below.

```
4==2*2%evaluates to true, since 4=2*2.
4==2*3%evaluates to false, since 2*3 does note equal 4.
4>2%evaluates to true, since 4 is greater than 2.
2<2%evaluates to false, since 2 is not less than itself.
2<=2%evaluates to true, since 2 is less than or equal to itself.
4~=2%evaluates to true, since 4 is not equal to 2.
2~=2%evaluates to false, since 2 is not not equal to itself.
isequal(M1,M1)%evaluates to false, since M1 is the same array as itself.
isequal(r1,r2)%evaluates to false, since these two arrays are different.
```

MATLAB's basic **logical operators** are and (&), or (|), and not (~). & and | take two or more Booleans and output a new Boolean. ~ takes a single Boolean and outputs a new Boolean. They all play the same role as they do in language. Examples are demonstrated below.

```
true&true%outputs true.
true&false%outputs false.
false&true%outputs false.
false&false%outputs false.
true|true%outputs true.
true|false%outputs true.
false|true%outputs true.
false|false%outputs false.
~true%outputs false.
~false%outputs true.
2<2|4>2%outputs true, since this is the same as false|true.
~2<3%outputs false, since this is the same as ~true.
(2<2|4>2)&(~2<3)%outputs false, since this is the same as true&false.
```

Functions

A function is anything that takes one or more data types as an input, and either performs a task based on that input, or gives one or more data types as an output. Some examples of functions already built into MATLAB are:

```
sin(x) %takes x as an input and gives the sine of x as an output.
isequal(x,y) %takes two doubles x and y as inputs and gives true or false as an output,
    %depending on whether they are equal.
plot(x,y) %takes two arrays x and y as inputs and performs a task, namely, plots y against x.
```

It is extremely useful to be able to create your own functions. To do so in MATLAB, you create a text file that is formatted in a way that MATLAB understands, and save it with the extension ".m" in a folder where MATLAB

knows to look (The command `path` in MATLAB will display all folders that MATLAB looks in. The command `userpath` will display the current working directory for MATLAB, which is one of the folders that MATLAB looks in and is the default location where MATLAB saves files. The text document can be written using any basic text editor, including Notepad. MATLAB also has a built-in text editor that can be accessed through New->function from MATLAB.

The basic format of the text document for any MATLAB function is as follows (there are variations on this):

```
function outputVariableName=functionName(inputVariable1,inputVariable2,...)
body%Code maninpulating input variables
outputVariableName=somethingFromBody
end
```

There should be only one function per .m file, and the name of the .m file should be the same as the function"s name. An example function is given below.

```
%Function: finalHeight(initialHeight,initialVel,grav,time) returns the final position of a particle with known
    %initial position, initial velocity, and acceleration after a given time,
    %assuming that that particle obeys the kinematic equation for constant acceleration.
%This file would have to be saved in one of the MATLAB path folders, with the name, "finalPos.m"
function fin=finalHeight(initialHeight,initialVel,grav,time)
y=initialHeight+initialVel*time+(1/2)*grav*time^2
fin=y
end
```

If the above function is saved as indicated, and in MATLAB you now type:

```
x0=10;v0=2;a=3;t=5;
x=finalPos(x0,v0,a,t)
```

MATLAB will use the programmed function, and store `x=57.5`, since 10+2*5+(1/2)*3*5^2=10+10+37.5=57.5. Note that as you program write more functions and save them in the MATLAB folders, your copy of MATLAB effectively has an ever-growing catalogue of functions that you can use.

Conditionals You will frequently want your functions to behave differently depending on what conditions your inputs satisfy. For instance, in the finalHeight function above, if the ground is represented by y=0, I may want my function to reflect that my particle will stop dropping if it hits the ground. The keywords `if()`, `elseif()`, and `else` are used to put conditionals in functions. The format is:

```
if (statement that can evaluate to true or false)
    body to execute if statement is true
elseif (other statement that can evaluate to true or false)%There can be any number >=0 of elseif
    %statements and bodies.
    body to execute if statement is true
else
    body to execute if all preceding statements are false.%else statements are not always needed,
        %but are usually a good thing to include with if statements.
end
```

The finalHeight function above is modified below to include an if statement:

```
function fin=finalHeight(initialHeight,initialVel,grav,time)
y=initialHeight+initialVel*time+(1/2)*grav*time^2
if (y<=0)
    fin=0;
else
    fin=y;
end
end
```

Loops You will frequently want your functions to perform a given task multiple times. Loops cause a chosen portion of your program to do this. There are two types of loops - for loops cause a portion of your program to

repeat over a set number of iterations, and while loops cause your program to repeat while a given condition is true. The basic format for both of these loops are given below:

```
%%FOR LOOP FORMAT%%
for LoopVariable=start:step:stop%if step=1, format is for LoopVariable=start:stop
    body%executes, starting with LoopVariable=start, ending when LoopVariable>stop.
end

%%WHILE LOOP FORMAT%%
while (statement that can evaluate to true or false)%Parentheses not necessary
    body
end
```

A function using an example of each is given below. Both functions do the same thing, namely, give an array containing the squares of all integers between the integers `low` and `high` that are input into the function.

```
function sqArray = sqArrayFor(low,high)%for version
arr=[];
for i=low:high
    arr=cat(2,arr,i^2);
end
sqArray=arr;
end

function sqArray = sqArrayWhile(low,high)%while version
i=low;
arr=[];
while (i<=high)
    arr=cat(2,arr,i^2);
    i=i+1;
end
sqArray=arr;
end
```

Scripts

A script in MATLAB is simply a sequence of commands that could be executed from the command window, but which are instead saved to a text file with the extension ".m". As with functions, the .m file can be created using any text editor, and MATLAB has a built-in editor for this purpose which can be accessed by clicking on "New Script". The .m file must be saved in one of the MATLAB path folders.
To run a saved script from the Command Window, type the command:

```
run ScriptName.m
```

Doing this is equivalent to typing all of the commands in the script one-at-a-time from the command window. That's all there is to it. An example script is written below.

```
%Save this script as MySquareArray.m. Once this is done, can run it by typing
    %"run MySquareArray.m" in the Command Window.
%Note that script assigns values to variables just as the Commmand Window does.
%Note also that it has access to functions that you have written.
a=3;b=5;
MyArray=sqArrayFor(a,b)
```