Statistics and Data Analysis in MATLAB
Kendrick Kay, kendrick.kay@wustl.edu

MATLAB Basics I

1. The MATLAB environment

- ***Variables*** hold things, such as numbers, matrices, strings, etc. Variable names are simple alphanumeric strings (e.g. `rec`, `ABC`, or `var123`).
- The ***workspace*** is the collection of variables that currently exist. Variables can be defined by the user, or can be loaded into the workspace from a file. Variables can be saved from the workspace to a file. When the MATLAB session ends (i.e. MATLAB is quit), the workspace is lost.
- The ***command window*** is where you interact with MATLAB. The command prompt (**`>>`**) is where you type a command.
- The ***GUI (graphical user interface)*** is all the pretty stuff that surrounds the command window (menus, other windows, etc.). The GUI is helpful for various things (e.g. debugging, managing files) but is not essential. You can run MATLAB without the GUI.
- **Functions** accept input and return output. There are built-in functions (e.g. `inv`) that have no readable source code, functions that come with MATLAB (e.g. `mean`) that have readable source code, and user-written functions. Importantly, functions cannot access nor modify the workspace except through **inputs** and **outputs**.
- **Scripts** are text files that consist of a series of MATLAB commands. Scripts are similar to user-written functions (which are also text files consisting of a series of MATLAB commands). However, the crucial difference is that scripts do not operate on inputs and outputs. Rather, scripts have direct access to the workspace, so they can freely read variables from the workspace and write variables to the workspace.
- Functions and scripts written by the user are saved as **.m files**. These are text files.
- MATLAB saves variables into **.mat files**. These are binary files that can be read by MATLAB.

2. Interacting with MATLAB

- To interact with MATLAB, just type a command into the command window.
- If a command is not ended with **`;`**, the result of evaluating the command is displayed in the command window.
- If the output is not assigned to a variable, it is automatically assigned to a variable called **`ans`**.
- Typically, a command is either an **expression** (e.g. `1+1`) or an **assignment** (e.g. `a=1+1`).

3. Basic commands for navigating the MATLAB environment

- **`whos`** - provide information on variables that currently exist in the workspace
- **`clear <variable name>`** - remove the variable from the workspace
- **`clear all`** - remove all variables from the workspace
- **`help <function>`** - get help on a function
- **`type <function>`** - show the code that underlies a function
- **`edit <function>`** - edit the code that underlies a function
- **`which <function>`** - show the location of the file that defines the function
- **`cd <path>`** - change directory
- **`pwd`** - print the current working directory

- **`ls`** - list files in current working directory
- **`load <.mat file>`** - load the variables contained in the .mat file into the workspace
- **`save <.mat file>`** - save the variables in the current workspace to the .mat file

4. Basic data types
- The **`double`** data type is the standard MATLAB data type, allowing storage of numbers in double precision (e.g. `5`, `-1.234`, `3.14159265358979`).
- The **`single`** data type has less precision than **`double`** and takes up half as much memory (e.g. `3.141593`). Unless memory is an issue, you do not need to worry about using single.
- The **`logical`** data type consists of **`0`**s and **`1`**s, which mean **`false`** and **`true`**, respectively.
- The **`char`** data type allows characters, such as `'a'`, `'T'`, and `'!'`. Characters can be concatenated together into strings, such as `'abc'` and `'The dog'`.
- The **`cell`** data type allows different data types to be collected together (e.g. `{1 'a'}`).
- There are also integer data types such as **`uint8`** and **`int16`**.
- Most of the time, numbers will be finite. However, there are special keywords that indicate non-finite numbers. These include **`NaN`** (not-a-number), **`Inf`** (infinity), and **`-Inf`** (negative infinity). `NaN`s are often used to indicate missing values.

5. Everything is a matrix
- Data in MATLAB are stored as **matrices** (also known as **arrays**). The individual components that compose a matrix are called **elements**.
- Matrices can be constructed from different data types. For example, you might have a double matrix or a logical matrix.
- A single number, or **scalar**, is a matrix with dimensions 1 x 1 (e.g. `5`).
- A **vector** is a matrix that has a dimensionality of 1 in all but one dimension. Typically, a vector is a **row vector** (e.g. `[5 6 7]` which has dimensions 1 x 3) or a **column vector** (e.g. `[5; 6; 7]` which has dimensions 3 x 1).
- Matrices are often two-dimensional (e.g. `[1 2 3; 4 5 6; 7 8 9]`) but can have arbitrary dimensionality (e.g. `zeros(10,10,10,10)`).
- It is possible for a matrix to be empty (no elements). The **empty matrix** is given by **`[]`**.
- Strings are vectors of characters (e.g. `'abc'` is a char vector of length 3).
- Cells are matrices that consist of heterogeneous elements (e.g. `{1 'abc'}` is a cell vector of length 2).
- Matrices can be created through the use of brackets (**`[ ]`**). Semicolons (**`;`**) indicate the end of a row. Matrices can also be created through the use of functions, such as `zeros`, `ones`, and `rand`. Cell matrices can be created through the use of curly brackets (**`{ }`**).

6. Indexing into matrices
- To access the elements of a matrix, we use **indexing**. Indexing is accomplished through the use of positive integers referring to the various positions in a matrix. MATLAB's convention is to use 1-indexing, meaning that the first element is indexed by 1. For example, `b(5,[1 3])` refers to the first and third elements in the fifth row of `b`; when evaluated, the expression returns a row vector with two elements.
- The colon (**`:`**) is a special symbol that means all indices along a given dimension.
- The keyword **`end`** is a shortcut for the index of the last element along a given dimension.

- For the purposes of indexing, MATLAB can automatically treat a matrix as a vector. For example, `b(100)` refers to the 100th element of `b` (regardless of the dimensionality of `b`). The order of indexing is row-first. For example, suppose `a = [1 2; 3 4]`. Then, `a(2)` is equal to 3.
- Indexing can also be used to selectively modify the contents of a matrix. This is done by using indexing on the left-hand side of an assignment. For example, `b(3,:) = [1 2 3]` changes the contents of the third row of `b`.
- A special indexing shortcut is `b(:)` which returns all of the elements of `b` as a column vector.
- It is possible to assign a scalar to multiple elements. This is because MATLAB automatically repeats the scalar for you. For example, `b(3,:) = 7` is a valid command that sets all elements in the third row of `b` to 7.

7. Useful matrix-manipulation operations

`size` - return the dimensionality of a matrix
```
size([1 2 3])                          → [1 3]
```
`length` - return the number of elements in a vector
```
length([1 2 3])                        → 3
```
`reshape` - change the dimensionality of a matrix without altering its contents
```
reshape([1 2 3 4],[2 2])               → [1 3; 2 4]
```
`permute` - change the order of dimensions of a matrix
```
permute([1 2 3 4],[2 1])               → [1; 2; 3; 4]
```
`'` - swap the dimensions of a 2D matrix
```
[1 2 3 4]'                             → [1; 2; 3; 4]
```
`repmat` - replicate the contents of a matrix, forming a larger matrix
```
repmat([1 2],[1 3])                    → [1 2 1 2 1 2]
```
`bsxfun` - automatically apply `repmat` as required by certain function calls
```
bsxfun(@plus,[1 2 3],[5 5 5; 6 6 6])   → [6 7 8; 7 8 9]
```
`find` - convert logical indices into integer indices
```
find([2 5 6 2]==2)                     → [1 4]
```

8. Basic mathematical operations

`+ - * / ^` - add, subtract, multiply, divide, exponentiate
```
1+1                                    → 2
2*[4 6]                                → [8 12]
2^3                                    → 8
```
`sqrt` - square root
```
sqrt([4 25])                           → [2 5]
```
`exp` - exponential
```
exp(1)                                 → 2.718
```
`log`, **`log10`** - natural logarithm, base-10 logarithm
```
log10(100)                             → 2
```
`abs` - absolute value
```
abs(-3)                                → 3
```
`.+ .- .* ./ .^` - apply operations element-wise
```
[2 4 6].*[3 1 2]                       → [6 4 12]
```
`round`, **`floor`**, **`ceil`** - convert decimals to integers
```
floor(5.1)                             → 5
```
`min`, **`max`** - find the minimum or maximum
```
min([10 2 5])                          → 2
```
`sum` - add all elements
```
sum([1 2 3])                           → 6
```

9. Other operations

< > <= >= == ~= - various logical comparisons

```
5 > 4                                  → 1
[10 0 1] == 10                         → [1 0 0]
```

: - construct a series of equally spaced numbers

```
1:.5:3                                 → [1 1.5 2 2.5 3]
5:-1:1                                 → [5 4 3 2 1]
5:8                                    → [5 6 7 8]
```

mean, **median**, **std**, **var**, **prctile** - statistical operations

```
median([1 2 3])                        → 2
prctile([1 2 3],50)                    → 2
```

sort, **union** - list operations

```
sort([3 1 2 2])                        → [1 2 2 3]
union([1 2],[4 2 2 1])                 → [1 2 4]
```

rand, **randn**, **randperm** - randomization operations

```
randn(1,5)                             → [1.368 0.656 1.931 1.826 -1.206]
randperm(5)                            → [2 4 5 3 1]
```

fprintf, **sprintf** - text-printing operations

```
fprintf('%.5f',pi)                     → '3.14159' is written to the command window
fprintf('The answer is %d.',10)        → 'The answer is 10.' is written to the command window
```

isempty, **isnan**, **isfinite** - data-checking operations

```
isempty([])                            → 1
isnan([1 NaN 10])                      → [0 1 0]
isfinite([1 2 Inf NaN])                → [1 1 0 0]
```

fopen, **fclose** - file-manipulation operations

```
fid = fopen('test.txt','w');
fprintf(fid,'%d %d %d',1,2,3);
fclose(fid);                           → '1 2 3' is written to a file called 'test.txt'
```