MM409/MM509 – Lab. 2
February 7, 2019 [FA]

By the end of this Lab, you will

- be familiar with more basic Matlab operations;
- have used the `help` command a few times;
- know how to use `for` and `if` stetements;
- have built a few more functions on your own and written a script that contains the starting steps of your assignment.

Before you start: download the folder 'DATASET' from MyPlace and save it in your current folder.

Review of some conventions and useful tips

- Use lower case letters for vectors and scalars;
- use UPPER case letters for matrices;
- blank spaces are not allowed when naming variables/scripts/functions. Use '_' instead;
- **always** comment your code and use insightful names for your variables;
- to clear the command window (without affecting the workspace) use `clc`;
- to clear the workspace (remove ALL your variables) use `clear`;
- **ALWAYS end a line with a ';'** ;

Getting started 2

- `ans` is a temporary variable that MATLAB uses to store outputs when you don't assign names;
- `=` is used to *assign* values;
- `>,>=,<,<=,==` are logical operators;
- `&&` is the logical "and";
- `||` is the logical "or";
- `~` is the logical "not";
- `i=1:n` is short for "`i` from `1` to `n`";
- `i=1:k:n` is short for "`i` from `1` to `n` with step `k`";
- `i=n:-1:1` is short for "`i` from `n` to `1`";

Play around a bit with the logical operators. Write statements that include numbers, vectors or matrices or combination of these and see how MATLAB responds. What happens, for example, when you ask if a certain vector is ≥ 0?

Another thing to note about `~` is that it can be used to tell MATLAB to avoid saving an output that we do not need. For example, suppose that you are using the function `max` to find the *location* of the

largest element in a vector `v` – you do not care about the actual value, you just want to know where it is. You can see from the help of this function that

```
>> [val,idx] = max(v);
```

stores in `val` the maximum value in `v` and in `idx` the index such that `v(idx) = val`. However, you do not need to know the actual value of the maximum, therefore you can use `~` to avoid saving `val` as:

```
>> [~,idx] = max(v);
```

This is very different from `i = max(v)`, since by calling the function in this way, MATLAB will return `i = val`. Try and see the difference by working on small vector, such as a pseudo-random vector of length five

```
>> v = rand(5,1);
```

`for` statements

A `for` statement is used to make MATLAB perform the same type of task for a fixed number of times. This could involve, for example, updating a certain matrix/vector/number a fixed number of times, or perform the same operation on all the columns of a matrix.

What do the following `for` loops do? Comment the code.

1)

```
s = 0;
for i = 1:10
  s = (s + i)^ 2;
end
```

2) Suppose you have a matrix `A` in your workspace:

```
n = size(A,1);
for i = 1:n
  A(:,i) = i*A(:,i);
end
```

`if` statements

An `if` statement is used to make MATLAB perform a certain sequence of instructions *only* if a given condition is satisfied. Generally, you will describe your condition using a mathematical statement (something like: `i>1000`, where `i` is a variable that you have computed somewhere before the `if`). If the condition is true, then MATLAB will perform the tasks described inside the `if`; otherwise it will move on to either check another condition or leave the `if` statement. In the simplest form, an `if` statement looks like:

`if` *condition 1*
  set of tasks to perform if condition 1 is satisfied;
`end`

We have seen an instance of this `if` statements when we built the function `degree_adj.m` during lab1. Two other examples: what do the following functions do? Comment the code:

1)

```
function [s1,s2] = mystery_function(n)
  s1 = 0;
  s2 = 0;
    for i = 1:n
        if mod(n,2) == 0
            s1 = s1 + i;
        else
            s2 = s2 + i;
        end
    end
end
```

2) `function p = another_mystery_function(n)`

```
   p = 100;
      if n >= p
          p = 42;
      end
end
```

`if` statements are very versatile: if you have more than one possible option when it comes to the operation you need to perform and each of these different options require a different condition to be satisfied, then you can use an `if` statement as follows:

```
if condition 1
      set of tasks to perform if condition 1 is satisfied;
   else if condition 2
     set of tasks to perform if condition 2 is satisfied;
           ⋮
          else       % if none of the above conditions is satisfied
             set of tasks to perform, if any;
          end
           ⋮
   end
end
```

Exercise: Write a function that takes a number n as input and returns another number which equals

- 0, if the input is larger than 10 but smaller than 15
- 12, if the input is smaller than 10, or
- n, otherwise.

Tasks To practice `for` and `if` statements, you will code a function that build the incidence matrix and compare it with another function that I have coded. The goal is to show that `for` loops are inefficient in Matlab and are thus best avoided.

- Download from Myplace the function `adj2inc_slow.m`. This is an example of how **not** to code in Matlab. I have used two nested for loops to browse the adjacency matrix and look for nonzeros, and a counter to access a new row in B every time I find a nonzero in A – so that I can record the corresponding edge in an empty line in B. You need to code something that is less stupid than what I have done.
- Write a function `adj2inc_better.m` that takes in input the adjacency matrix of a simple graph and returns its incidence matrix. You will need to use **one** `for` loop (can you do without?) The functions that you may want to use are: `isequal`, `triu` (to overwrite your matrix `A`, when appropriate), and `find` or `adj2list`.

Once this is done:

- In your command window: load a file from the DATASET folder, say `minnesota.mat`. This represents the road network of the State of Minnesota.

  ```
  >> load 'DATASET/minnesota.mat'
  ```

A matrix should have appeared in your workspace. We now want to time how long it takes for the two different functions to build B. To time something, you need to use the `tic/toc` commands as follows:

```
>> tic; B1 = adj2inc_slow(A); toc
```

`tic` starts a stopwatch, `toc` stops it. Matlab then returns a message like this:

```
Elapsed time is 7.430731 seconds.
```

Note that your computer might take longer (it depends on the computer itself, as well as on how many other things your computer is doing – Facebook, etc. slow it down considerably).

Now time the second function, the one you have coded:

```
>> tic; B2 = adj2inc_better(A); toc
```

which on my PC returns: `Elapsed time is 0.009658 seconds.`

As you can see, my code is highly inefficient compared to what you should have gotten, and the difference is not small at all! Take home message: the fewer `for` loops, the better.

Scripts

In MATLAB scripts are just empty pages: they are used to record a sequence of instructions that you might need to repeat for different datasets, but that are not as general and broadly applicable as functions. Functions are the equivalent of a calculator, while scripts are the equivalent of a piece of paper where you note the calculations that you want to perform.

The variables that you call in scripts are *global*. This means that the names that you use inside a script for your variables will affect your workspace, potentially overwriting things! So be careful. If you have a matrix called, say, `M` in your workspace and you think you might need it in the future, do not use the name `M` for other things inside your script. In the same way, if you compute something within a script, you can re-use it later on, as it will be stored in the workspace.

Most of the scripts in Network Science start in the same way:

- load/write the data;
- compute the number of nodes in your network; [Hint: `size`]
- check symmetry of your adjacency matrix; [Hint: `isequal`]
- compute the number of edges. [Hint: `nnz`]

Your task is now to write a script where you use what you have learned and coded so far. It is extremely important that you comment your script properly. It is likely that you will be coming back to this for your project, so you need to be able to go back to it and understand it fully at a first glance.

Your script should perform the following:

1. clear the workspace and the command window;
2. load one file (one of the matrices in the DATASET - start with zachary or sawmill, since they are small graphs!). For example:

   ```
   >> load 'DATASET/zachary1.mat'
   ```

The dataset 'USAir97list.txt' is in a different format. If you try and load it directly from the command window you'll see that it has dimension 4252×3. Instead of being given the adjacency matrix, you are given the adjacency list. Therefore, before going any further you will need to convert your data. In the command window:

```
>> clear all
>> load('DATASET/USAir97list');
>> A = .....  % do whatever operation you need to do to convert an adj list to an adj matrix
>> save('DATASET/USAir97adj') % save your adjacency matrix with the name USAir97adj.mat
```

3. compute the number of nodes [Hint: `size`.]
4. compute the number of (undirected) edges [Hint: `nnz` – this changes depending on whether A is symmetric or not.]
5. display the sparsity pattern of the matrix i.e., a plot where ones in the matrix are represented by blue dots, while zeros are left blank. To do this, you need open a new figure by calling `figure` and at the following line use the command `spy(A)`
6. compute the degree(s) of each node [Hint: this should sound familiar..]
7. compute the mean degree(s) [Hint: `mean`]
8. check whether the adjacency matrix is symmetric. If that is the case, then build the associated incidence matrix. Check that $B^T B = D - A$, where D is the diagonal matrix with the degrees of the nodes. [Hint: use `isequal` and not `==`. Use what you computed at step 6 and the built-in function `diag` to form D.]

Datasets are from `https://sparse.tamu.edu/`, where you can also find many other matrices – not just adjacency matrices!

Solutions will be updated on Myplace by the end of next week.