**HW1 – Generating and identifying random/scale free networks**

**Submission guidelines:**

You are required to submit your solution as a zip file:

* Python file with your functions implemented, the file’s name will be ID.py where ID is the student’s ID.
* A PDF file with answers to open questions (these are marked in a blue font).
* The zip file’s name will be ID.zip where ID is the student’s ID. For example, the student Moshe Moshe with an ID of 1234567, will submit a zip file “1234567.zip”. Within his zip file, his implemented solution “1234567.py” and a pdf file.
* You are required to implement a (static) function called ‘get\_name’ that returns your full name in English.
* You are required to implement a (static) function called ‘get\_id’ that returns your ID number.

Objectives:

* Generating random networks
* Analyze generated random networks
* Estimate networks’ parameters (random or scale free)
* Distinguish between random networks and scale free networks

As mentioned in HW0 - before you start please have the following settings on your PC:

* Python 3.7 (or newer)
* Anaconda environment
* networkX package installed

1. **Generating and analysing random networks**
2. Use the networkX package and implement a function (called “random\_netwroks\_generator”) that generates random networks, according to the G(n,p) model we learned. The function parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘n’ | int (>0) | Int. number of nodes in each network |  |
| ‘p’ | float (>0 and <=1) | probability for pair of nodes to be connected |  |
| ‘num\_netwroks’ | int (>0) | Number of networks to generate | 1 |
| ‘directed’ | bool (True/False) | Whether the network is directed or not | False |
| ‘seed’ | int | A seed for the random generation of the the function call | your ID # |

The function should return **a list** (of size ‘num\_netwroks’) with networkX objects. Each element in the list is a network.

1. Implement a function (called “network\_stats”) that calculates basic statistics for a given network.

The function gets as input a single networkX object and returns **a dictionary** with some statistics about the network. The returned dictionary should hold the following key/value pairs:

|  |  |  |
| --- | --- | --- |
| **Key** | **Value type** | **Explanation** |
| ‘degrees\_avg’ | float | Average degrees distribution. |
| ‘degrees\_std’ | float | Standard deviation degrees distribution. |
| ‘degrees\_min’ | float | Degrees distribution minimum value. |
| ‘degrees\_max’ | float | Degrees distribution maximum value. |
| ‘spl’ | float | Average shortest path length between pairs of users. |
| ‘diameter’ | float | The network’s diameter. |

1. Implement a function (called “networks\_avg\_stats”) that calculates basic statistics for a given **list** of networks.

The function gets as input **a list** of networkX objects (same as the output of the “random\_netwroks\_generator” function). It returns **a dictionary** with some statistics about the list of networks. The returned dictionary should hold the same key/value pairs listed in section (ii) above, where values are averaged over the list of networks. As an example, the ‘diameter’ value in the returned dictionary should hold the **average** diameter of the networks in the input list.

1. Use sections (i) to generate the 4 types of undirected random networks listed in the table below. Then, use sections (ii) and (iii) to calculate the basic statistics for each type.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | ‘num\_netwroks’ parameter | ‘n’ parameter | ‘p’ parameter |
| a | 20 | 100 | 0.1 |
| b | 20 | 100 | 0.6 |
| c | 10 | 1000 | 0.1 |
| d | 10 | 1000 | 0.6 |

1. Analyse the statistics you got for the different networks’ type. Does it make sense? How do the parameters affect the statistics?
2. **Random networks - hypothesis testing**

Load the [pickle file](https://moodle2.bgu.ac.il/moodle/pluginfile.php/2561627/mod_folder/content/0/rand_nets.p?forcedownload=1) (‘rand\_nets.p’) that contains a python list (of size 10). Each item in the list is a network object. For this part of the HW, we assume that each network was generated by the random network model (i.e., G(N,p)). In addition, we assume that the ‘p’ parameter is one out of the following 4 options: [0.01, 0.1, 0.3, 0.6].

* 1. Load the pickle file, and make sure you can access each network in the extracted list.
  2. Implement a function (called “rand\_net\_hypothesis\_testing”) that performs hypothesis testing for a random network’s ‘p’ parameter. The hypothesis test is as follows:

The function parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘network’ | a networkX object | A random network to test the hypothesis against |  |
| ‘theoretical\_p’ | float (>0 and <1) | The H0 ‘p’ value assumption |  |
| ‘alpha’ | float (>0 and <1) | The significance level of the test | 0.05 |

The function should return a tuple of size two. The first element is the p-value of the test. The second element is a string of ‘accept’ or ‘reject’ (‘accept’ means that was accepted, ‘reject’ means was rejected).

* 1. Implement a function (called “most\_probable\_p”) that finds the most probable ‘p’ parameter of a given network (out of the 4 options above). The function’s single input parameter is ‘graph’ (a netwrokX object). The function’s output is a single float value which is one out of the following: -1, 0.01, 0.1, 0.3, 0.6 (-1 in case none of the options fitted well).

HINT: What does the degree distribution of a random network should look like? Look for a relevant hypothesis testing of this distribution parameter in [scipy.stats](https://docs.scipy.org/doc/scipy/reference/stats.html).

* 1. Choose one network (out of the 10) for which you have clearly found the optimal ‘p’ parameter for (e.g., your ‘most\_probable\_p’ returned one out of the [0.01, 0.1, 0.3, 0.6] values.

For this network, run the ‘rand\_net\_hypothesis\_testing’ again. This time, change the ‘theoretical\_p’ parameter a bit (10% up or down). What happens to the results? What does it mean? Repeat this step, but now with a much bigger/smaller network - do the results change? What does it mean?

A Bonus Section (NOT mandatory)

Another optional way to validate a distribution is through QQ-PLOT. If you are not familiar with it you can read about it [here](https://en.wikipedia.org/wiki/Q%E2%80%93Q_plot).

* 1. Use the QQ-Plot technique to validate that the network ‘behaves’ like a random network.

HINT: You can use the normal distribution as the distribution to compare to. The justification for this approximation is that for large enough sample size (n>30), the normal distribution is a well approximation of the binomial distribution.

1. **Find an optimal 𝛾 parameter to a scale-free network**

Load the [pickle file](https://moodle2.bgu.ac.il/moodle/pluginfile.php/2561627/mod_folder/content/0/scalefree_nets.p?forcedownload=1) (‘scalefree\_nets.p’) that contains a python list (of size 10). Each item in the list contains a network. For this part of the HW, we assume that each network behaves like a scale-free network.

* 1. Load the pickle file, and make sure you can access each network in the extracted list.

In the rest of this assignment, you should use the ‘powerlaw’ python package. First install it, and then take the time to read the [documentation](https://arxiv.org/pdf/1305.0215.pdf) and understand how to work with the package.

HINT: Do not get confused with the names of the parameters specified in the ‘powerlaw’ package. In the package, the ‘alpha’ notation is used for the the 𝛾 parameter.

* 1. implement a function called ‘find\_opt\_gamma’ that uses the ‘powerlow’ package to find an optimal 𝛾 parameter for a given network.

The function parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘network’ | a networkX object | A random network |  |
| ‘treat\_as\_social\_network’ | bool (True/False) | Whether to treat the specified network as a social network\* | True |

\* The reason we have this flag is because social networks have special characteristics (e.g., degree distribution is discrete and not continuous).

The function should return the optimal 𝛾 parameter found (a float).

* 1. Run the ‘find\_opt\_gamma’ over the ten networks you explored in section (i) to find an optimal 𝛾 parameter for each.
  2. Run the ‘networks\_stats’ from question 1 over a single scale-free network. Compare the statistics you get now with the ones you got in section 1. Does it make sense? Hint: it is best to compare networks with approximately the same number of nodes and edges.

1. **Distinguish between random networks and scale free networks**

Extract the [zipped file](https://drive.google.com/file/d/1dRD8_OBx0lf2AC0D2b5KPw7GTxCmJVpC/view?usp=sharing) and load the pickle file (‘mixed\_nets.p’) that contains a python list (of size 20). Each item in the list contains a network.

* 1. Load the pickle file, and make sure you can access each network in the extracted list.
  2. implement a function called ‘netwrok\_classifier’ that decides whether a given network is a random network or a scale-free network (the most probable out of these two).

The function parameters are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter name** | **Parameter type** | **Explanation** | **Default value** |
| ‘network’ | a networkX object | A network to classify |  |

The function should return an int value (1, 2). 1 in case it classified it as a random network, 2 in case it classified it as a scale-free network.

* 1. Run the ‘netwrok\_classifier’ over the 20 networks you explored in section (i) to classify whether each is a random network or a scale-free network.

**Notes:**

Please make sure you install the required packages using the latest version.

If you use any package that is not mentioned in the list below, please include a “requirements.txt” file, stating which packages you used for your solution.

List of packages:

* Pandas
* NetworkX
* Numpy
* Scipy
* tqdm
* Matplotlib
* Powerlaw
* Seaborn
* statsmodels