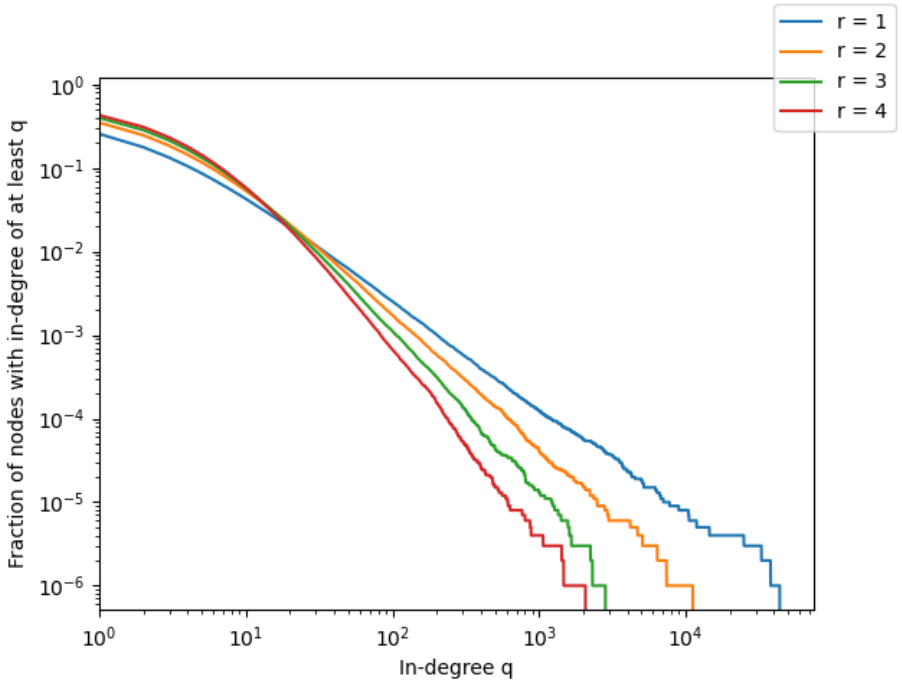
Network Analysis and Modeling
CSCI 5352, Fall 2020
Prof. Dan Larremore
Homework 11 and 12
Student: Alex Book; Collaborators: None

1. (100 pts total) Consider Price's model of a citation network, applied to publications in a single field.

   (a) (35 pts) Implement the simulation algorithm described in Chapter 13.1 of *Networks* [or 14.1 in the first edition].

   - For choices of $c = 3$ and $n = 10^6$, make a single figure showing the four complementary cumulative distribution functions $\Pr(K \geq k_{\text{in}})$ (the ccdf) for network in-degree k_{in}, one for each choice of $r = \{1, 2, 3, 4\}$.



   - Briefly discuss the impact of the uniform attachment mechanism on the distribution's shape and comment about the fraction of vertices with $k_{\text{in}} = 0$.
     A higher r-value causes the curve to become steeper, which essentially means that the "wealth gap" in citations becomes smaller. As expected, higher r-values cause the playing field to become noticeably more even. Additionally, the fraction of vertices with at least 1 citation is larger at higher r-values, which means that the fraction of vertices with $k_{in} = 0$ becomes smaller as r-values increase.

(b) (30 pts) Reasonable values of the model parameters for real citation networks are $c = 12$ and $r = 5$.

- For these choices, use your numerical simulation to calculate (i) the average number of citations to a paper (in-degree) in the first 10% of published papers (vertices) and (ii) the average number for a paper in the last 10%.
  The average number of citations to a paper in the first and last 10% of published papers are 81.32777 and .186872, respectively.

- Briefly discuss the implications of your results with respect to the "first-mover advantage," and the corresponding bias in citation counts for the first papers published in a field.
  The "first-movers" have a clear advantage in this simulation, and typically in real life as well, as they make themselves (and in this case, their papers) available to others as soon as possible, and thus reap the benefits. The latecomers surely make themselves available at some point, but since a greater wealth of connections allows individuals to be chosen/cited more often it is significantly harder for latecomers to catch up to their first-mover counterparts and to succeed. This supports the saying of "the rich get richer."

Hint: For a *good* estimate, average your answer over many repetitions of the simulation.

(c) (15 pts) Visit the *Index of Complex Networks* at `icon.colorado.edu`. Under the ICON entry for "arXiv citation networks (1993-2003)," obtain both the network and dates files for the hep-ph citation network.

- For (i) the first 10% and (ii) the last 10% of papers with submission dates, compute their average in-degree. Discuss any steps you took to convert the input data into a form on which you could perform these calculations.
  The average number of citations to a paper in the first and last 10% of published papers are 19.376 and 2.002, respectively. In order to convert the input data into a form fit for these calculations (and then actually do the calculations), I did the following:
  i. Read in the paper ID and date of publication from the appropriate file, removing the "11" from any paper ID that started with it (per the comment at the top of the file).
  ii. Create a dictionary from this file, structured as key = paper ID, value = [publication date, out-degree, in-degree].
  iii. Read in the edge list file, and if an edge spans between two papers that are in our dictionary of known nodes, add 1 to the out-degree and in-degree of the appropriate papers.
  iv. Sort the papers in order of publication date, and remove any papers with both an in- and out-degree of zero (as these papers were never used in the edge list file, so shouldn't be considered as part of the calculations).
  v. Find the mean in-degree of both the first and last 10% of papers, and print them.

- Briefly discuss how well, and why, these empirical values agree or disagree with your model estimates from question (1b).
  These values have the same trend as the model estimates from question (1b), as the "first-movers" still have a higher degree on average. However, it seems the the model from (1b) was more extreme than reality. Some possible reasons for this are discussed below in (1d).

Hint: You will need to "clean" these data a little in order to get good results. The two files contain non-identical lists of ids; there are 30,558 that occur in both files. To do the analysis, it will be useful to construct a list of pairs (i, t_i) of node ids and the dates they were created, in increasing order of t_i. Then think about how to transform the input citation network into a form by which to calculate the desired values.

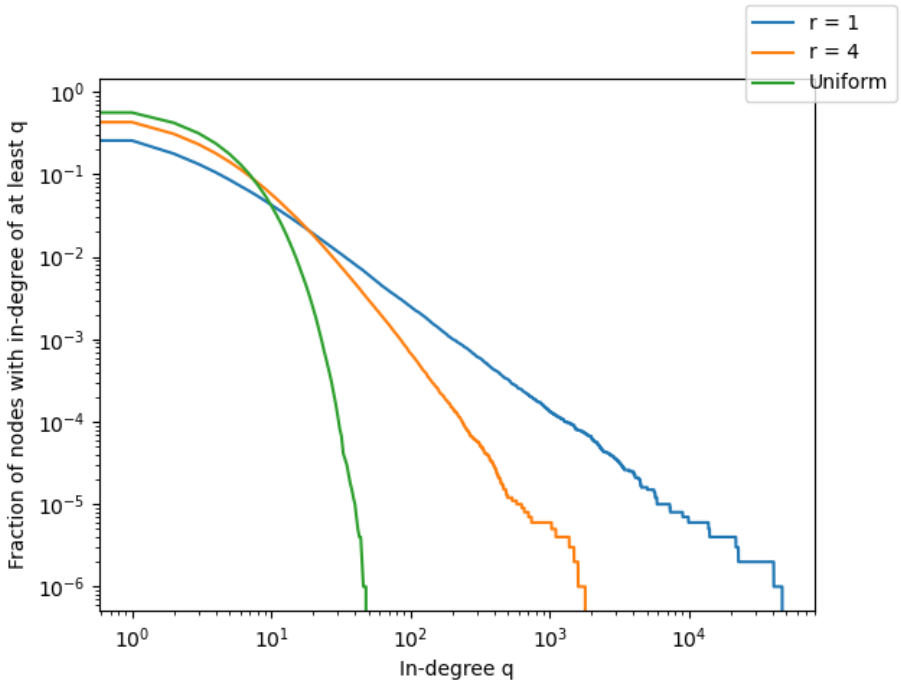
(d) (20 pts) Recall that Price's model is a dramatically simplified view of how nodes in a citation network accumulate new connections. Describe at least three ways that the "preferential attachment" mechanism is unrealistic in this context, and for each, suggest a way that you could analyze a real citation network to demonstrate the difference between what the model predicts and what the real world shows.

i. Papers won't necessarily reference other papers with probability exactly proportional to their degree or uniformly at random, as there will be other factors involved (topic, level of detail, quality of paper, trust of the authors, etc.). Depending on what factor is being taken into account, each node could have some qualifiers that cause matches to be more likely to cite each other (i.e. if both papers are talking about hockey fights, they are more likely to cite each other than one paper on hockey fights and another on corn growth).

ii. Authors may, in all honesty, exhibit some degree of laziness in looking for papers to cite, which changes how likely certain papers are to be cited and/or how many papers may be cited.

iii. Not every paper cites the same amount of other papers, as it is in Price's model. Having certain papers cite far more or less than the average amount could make a significant difference between reality and the model.

(e) (20 pts extra credit) Now consider a variation of Price's model in which we remove the preferential attachment part. That is, each time a new vertex joins the network, each of its c edges attaches to an existing vertex with equal probability. Using the same parameter choices as in question (1a), produce a figure showing the ccdfs for both this model and Price's model, for $r = \{1, 4\}$. Briefly discuss the differences in terms of how citations (edges) are distributed across papers (vertices).



As expected, edges are are more evenly distributed across papers when the preferential attachment mechanism is removed. There is still some level of "wealth gap", as earlier-added papers have more chances to be cited by other papers, but the chances of being cited no longer being proportional to degree evens out the playing field quite a bit.

Code for problems 1a, 1b, and 1e:

```
import numpy as np
import matplotlib
from matplotlib import pyplot as plt
import seaborn as sns

def ccdf_part_a(n, c, a):
    edges = np.zeros((n*c-6, 2)).astype(int) # structure is [out node, in node]
    in_degrees = np.zeros(n).astype(int)
    num_edges = 0

    for i in range(n):
        print(i)
        new_edges = []
        while len(new_edges) < min(c, i):
            r = np.random.uniform()
            if r < c / (c + a):
                target_i = np.random.randint(num_edges+1)
                target = edges[target_i][1]
            else:
                target = np.random.randint(i)
            if target not in new_edges:
                new_edges.append(target)
                edges[num_edges] = [i, target]
                in_degrees[target] += 1
                num_edges += 1

    min_degree = np.amin(in_degrees)
    max_degree = np.amax(in_degrees)
    bins = np.arange(0, max_degree+2, 1)
    hist, _ = np.histogram(in_degrees, bins)
    return bins[:-1], 1 - np.cumsum(hist)/np.sum(hist)

def ccdf_part_b(n, c, a):
    edges = np.zeros((n*c-6, 2)).astype(int) # structure is [out node, in node]
    in_degrees = np.zeros(n).astype(int)
    num_edges = 0

    for i in range(n):
        print(i)
        new_edges = []
        while len(new_edges) < min(c, i):
            r = np.random.uniform()
            if r < c / (c + a):
```

```
                target_i = np.random.randint(num_edges+1)
                target = edges[target_i][1]
            else:
                target = np.random.randint(i)
            if target not in new_edges:
                new_edges.append(target)
                edges[num_edges] = [i, target]
                in_degrees[target] += 1
                num_edges += 1
    return in_degrees

def ccdf_part_e(n, c):
    edges = np.zeros((n*c-6, 2)).astype(int) # structure is [out node, in node]
    in_degrees = np.zeros(n).astype(int)
    num_edges = 0

    for i in range(n):
        print(i)
        new_edges = []
        while len(new_edges) < min(c, i):
            target = np.random.randint(i)
            if (target not in new_edges) and (target != i):
                new_edges.append(target)
                edges[num_edges] = [i, target]
                in_degrees[target] += 1
                num_edges += 1

    min_degree = np.amin(in_degrees)
    max_degree = np.amax(in_degrees)
    bins = np.arange(0, max_degree+2, 1)
    hist, _ = np.histogram(in_degrees, bins)
    return bins[:-1], 1 - np.cumsum(hist)/np.sum(hist)

if __name__ == '__main__':
    ############################
    # PART A
    x1, y1 = ccdf_part_a(10**6, 3, 1)
    np.savetxt('data1x', x1, newline=" ")
    np.savetxt('data1y', y1, newline=" ")

    x2, y2 = ccdf_part_a(10**6, 3, 2)
    np.savetxt('data2x', x2, newline=" ")
    np.savetxt('data2y', y2, newline=" ")

    x3, y3 = ccdf_part_a(10**6, 3, 3)
```

```
np.savetxt('data3x', x3, newline=" ")
np.savetxt('data3y', y3, newline=" ")

x4, y4 = ccdf_part_a(10**6, 3, 4)
np.savetxt('data4x', x4, newline=" ")
np.savetxt('data4y', y4, newline=" ")

x1 = np.loadtxt('data1x')
y1 = np.loadtxt('data1y')

x2 = np.loadtxt('data2x')
y2 = np.loadtxt('data2y')

x3 = np.loadtxt('data3x')
y3 = np.loadtxt('data3y')

x4 = np.loadtxt('data4x')
y4 = np.loadtxt('data4y')

fig, ax = plt.subplots(1, 1)
ax.plot(x1, y1)
ax.plot(x2, y2)
ax.plot(x3, y3)
ax.plot(x4, y4)
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_xlabel('In-degree q')
ax.set_ylabel('Fraction of nodes with in-degree of at least q')
ax.set_xlim(10**0)
fig.legend(['r = 1', 'r = 2', 'r = 3', 'r = 4'])
plt.show()

############################
# PART B
first10percent = []
last10percent = []
for i in range(100):
    print(i)
    in_degrees_b = ccdf_part_b(10**6, 12, 5)
    avg_first_10percent = np.mean(in_degrees_b[:10**5])
    avg_last_10percent = np.mean(in_degrees_b[(10**6) - (10**5):])
    first10percent.append(avg_first_10percent)
    last10percent.append(avg_last_10percent)

avg_first_10percent = np.mean(first10percent)
```

```
avg_last_10percent = np.mean(last10percent)
print(avg_first_10percent, avg_last_10percent)

############################
# PART E
x0_e, y0_e = ccdf_part_e(10**6, 3)
np.savetxt('data0x_e', x0_e, newline=" ")
np.savetxt('data0y_e', y0_e, newline=" ")

x1_e, y1_e = ccdf_part_a(10**6, 3, 1)
np.savetxt('data1x_e', x1_e, newline=" ")
np.savetxt('data1y_e', y1_e, newline=" ")

x4_e, y4_e = ccdf_part_a(10**6, 3, 4)
np.savetxt('data4x_e', x4_e, newline=" ")
np.savetxt('data4y_e', y4_e, newline=" ")

x1_e = np.loadtxt('data1x_e')
y1_e = np.loadtxt('data1y_e')

x4_e = np.loadtxt('data4x_e')
y4_e = np.loadtxt('data4y_e')

x0_e = np.loadtxt('data0x_e')
y0_e = np.loadtxt('data0y_e')

fig, ax = plt.subplots(1, 1)
ax.plot(x1_e, y1_e)
ax.plot(x4_e, y4_e)
ax.plot(x0_e, y0_e)
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_xlabel('In-degree q')
ax.set_ylabel('Fraction of nodes with in-degree of at least q')
fig.legend(['r = 1', 'r = 4', 'Uniform'])
plt.show()
```

Code for problem 1c:

```
import numpy as np
import pandas as pd
from pprint import pprint

def nodes_to_dict(filename):
    file = open(filename, 'r')

    node_dict = {}

    for line in file:
        arr = line.split()
        if '#' not in arr:
            node, date = arr
            # cleaning cross-listed papers
            if node[:2] == '11':
                node = node[2:]
            node = int(node)
            if node not in node_dict.keys():
                node_dict[node] = [date, 0, 0] # [date, out-degree, in-degree]
    file.close()
    return node_dict

def edge_list_to_array(filename, node_dict):
    file = open(filename, 'r')

    for line in file:
        arr = line.split()
        if '#' not in arr:
            out_node, in_node = arr
            out_node, in_node = int(out_node), int(in_node)
            if (out_node in node_dict.keys()) and (in_node in node_dict.keys()):
                node_dict[out_node][1] += 1
                node_dict[in_node][2] += 1

    return node_dict

if __name__ == '__main__':
    node_dict = nodes_to_dict('partc_dates')
    node_dict = edge_list_to_array('partc_edges', node_dict)

    sorted_nodes = [(k,v) for (k, v) in sorted(node_dict.items(), key=lambda x: x[1])]

    for tuple in sorted_nodes:
```

```
    # if both out-degree and in-degree are zero, remove the node
    if tuple[1][1] == 0 and tuple[1][2] == 0:
        sorted_nodes.remove(tuple)

degrees = 0
count = 0
for tuple in sorted_nodes[:len(sorted_nodes)//10]:
    degrees += tuple[1][2]
    count += 1
first10_avg = degrees/count

degrees = 0
count = 0
for tuple in sorted_nodes[len(sorted_nodes)-len(sorted_nodes)//10:]:
    degrees += tuple[1][2]
    count += 1
last10_avg = degrees/count

print(first10_avg, last10_avg)
```