CS/EE/CMS/IDS 144 **Gurus: Emily & Dharshini**

HW 3: Heavy-tails and Small Worlds

Assigned: 01/22/2026 Due: 01/29/2026 by 11:59 PM (Midnight) PT

Collaboration is allowed for all problems and at the top of your homework sheet, please list all the people with whom you discussed. Crediting help from other classmates will not take away any credit from you. The details of the collaboration policy for this course are available in the Resources tab on Piazza.

You must turn in your homework electronically via Gradescope. ***Be sure to submit your homework as a single file.***

Start early and come to office hours with your questions! We also encourage you to post your questions on Piazza, as well as answer the questions asked by others on Piazza.

1 Coding + Data Analysis [70 points]

1.1 Heavy vs. light [30 points]

So far you haven't had any "hands-on" experience with heavy-tailed distributions. To help you get a better feel for the difference between heavy and light tails in this problem you'll experiment with them a bit.

In particular, we'll contrast:

(i) a Normal distribution with $\mu = 1$, $\sigma^2 = 1$

(ii) a Weibull distribution with $\alpha = 0.3$

(iii) a Pareto distribution with $\alpha = 0.5$ (and scale parameter $x_L = \frac{1}{3}$)

For the distributions (i) and (ii), set $\mathbb{E}[X] = 1$ for consistency. Recall that the mean of the Weibull distribution is $\beta\Gamma(1+\frac{1}{\alpha})$. For this problem, you'll need to simulate i.i.d. random variables X_i, where $i = 1 \ldots 10000$ for each of the distributions listed above. In parts (a) and (b), we'll focus on sums of these random variables. We define the partial sums as $S_n := \sum_{i=1}^{n} X_i$, $n \geq 1$.

Note: You can use any language for this exercise, but we will provide some tips for Mathematica and Python. **We have provided a Jupyter Notebook template (Python) for you to use, but feel free to use your own language of choice or implement it from scratch! Please export both your code and your plots as a pdf in order to submit it to Gradescope.**

(a) **Law of Large Numbers:** Make two plots of S_n vs. n for each of the distributions – the first plot over $n \in \{1, 2, \ldots, 20\}$ and the second one over the full range of n. Interpret your plots, in light of the law of large numbers and turn in your code.

(b) **Central Limit Theorem:** The Central Limit Theorem tells us that deviations of S_n from its mean are of size $\sqrt{n}$. That is, $S_n \approx nE[X] + O(\sqrt{n})$.
Plot $\frac{S_n - nE[X]}{\sqrt{n}}$ vs. n for each of the distributions. Interpret your plots, in light of the central limit theorem, and turn in your code.

(c) **The 80-20 rule:** Vilfredo Pareto was motivated to define the Pareto distribution by this observation: 80% of the wealth in society is held by 20% of the population. This is an important distinguishing feature between heavy-tailed and light-tailed distributions. To observe this, suppose that your samples represent the incomes of 10000 individuals in a city. Since some of your samples for the Normal distribution might be negative, ignore the case of the Normal distribution for this part of the problem, since a negative income doesn't make much sense.

Compute the fraction $f(r)$ of the total income of the city held by the wealthiest $r\%$ of the population, for $r = 1, 2, \cdots, 20$.For each of the distributions, plot $f(r)$ vs. r. (preferably both functions on a single plot). Interpret your plot(s) and turn in your code.

(d) **Identifying heavy tails:** Recall that two typical (but risky) approaches to identify heavy tails in empirical data is to observe the "frequency plot", a plot of the empirical pdf, and the "rank plot", a plot of the empirical ccdf. Frequency and rank plots that look linear on log-log scales are often considered indicators of heavy-tailed distributed data.

For each of the distributions (i)–(iii), plot the frequencies and ranks of the 10000 samples on log-log scales, using separate plots for each distribution. Since we are using a log-log scale, filter out all negative and zero values before graphing. Then filter out outliers. Why do we want to remove these? For the frequency plots, remember to experiment with various bin sizes and to choose one such that the plots are useful. (Note that bins aren't needed for the rank plot.) Then, use linear regression to fit a line through the points on each plot. Display the best-fit lines on the plots as well as the R-squared values. What do your plots tell you about identifying heavy tails based on frequency and rank plots? Again, please turn in your code! I look forward to reading every line :)

Note: The frequency plot is similar to the histogram except that the frequency plot has dots at the frequency values while a histogram has bars at those values. The rank plot is similar to the complementary cumulative distribution function (CCDF) except that the rank plot has dots corresponding to each empirical observation and the dots are not connected.

Both Mathematica and Python have lots of built-in packages which make doing this problem easier. Some tips for each are below:

If you're using Mathematica:

- You can use the standard functions `NormalDistribution`, `WeibullDistribution`, `ParetoDistribution` to generate random samples. In particular, to generate a random number for a normal distribution $\mathcal{N}(\mu, \sigma)$, use the command `data = RandomVariate[NormalDistribution[`u`, `σ`], LengthOfData]`. The other functions are used in a similar manner.
- To find partial sums in a list, use the `Accumulate[data]` function.
- To sort the elements in a list in increasing order, use `Sort[[data], Greater]` function.
- To fit a linear model to the data, use `fit = LinearModelFit[]`, and you can obtain statistics via `fit["RSquared"]`.
- To plot, consider `Plot = ListLinePlot[data, AxesLabel->xLabel,ylabel, AxesOrigin->xStart, yStart]`. You can use `Show[plot1, plot2]` to graph.

- To index specific parts of a dataset, use `data[[start:end]]`.
- Other useful functions: `ListLogLogPlot, LogLogPlot, Table, Map`.

If you're using Python:

- Consider using numpy and scipy. You can use `numpy.random.normal`, `numpy.random.weibull`, `numpy.random.pareto`.
  See also, for example, `scipy.stats.pareto` or `scipy.stats.exponweib`.
- To find partial sums in a list, use `numpy.cumsum`.
- To sort the elements in a list in increasing order, use `np.sort`.
- To plot, consider `matplotlib`. Instead of using the numpy functions you can also use `matplotlib.mlab` in some cases.
- Other useful function: `np.linspace`,

Of course, this task can be accomplished using MATLAB or any of various computing packages. Feel free to ask questions of the TAs (or work with other students who are more familiar with Mathematica, Python, etc) if something gives you trouble. And, remember to post on Piazza!

1.2 The Devil is in the Details [40 points]

We talked in class about one way to generate graphs with heavy-tailed degree distributions – preferential attachment. In HW 2, we studied one of the most basic random graph models: the Erdős–Rényi model. For this problem, we'll use a related, but much more general model: the *configuration model*. While both preferential attachment and the configuration model generate graphs with heavy-tailed degree distributions, the graphs that result can be very different in other ways. In this problem we will contrast these models.

(a) **Preferential attachment model**: Consider a community initially with only 2 residents who share a link or edge. Residents move into this community one at a time for $t = 1, 2, \ldots T-2$, and the weather is so nice that no one leaves. Upon arrival, a new resident becomes acquainted with, or forms an edge with, one of the existing residents. The probability p_i that a newcomer connects to resident i is

$$p_i = \frac{k_i}{\sum_j k_j}$$

where k_i is the degree of resident i, and the sum is made over all pre-existing residents j. When a newcomer enters the community, they are more likely to become acquainted with one of the more visible residents who already have many connections rather than with a relative unknown with few connections.

Your task:

- Write a function that can generate such a network. Your function should accept as a parameter T, the number of total residents. Your function should return a representation of this undirected graph (e.g. a networkx graph) and a list of the degrees of the nodes in the network. Please document and submit your code.
- Plot the empirical distribution (i.e. a CCDF, as seen in problem 1.1) of node degrees for $T = 300$ on a log-log scale. What characteristics do you see: are they as you would expect? If it helps, plot and analyze a CDF and go from there.

(b) **Configuration model**: Next we consider a simulation of the configuration model using a stub-matching procedure. More specifically, the procedure goes as follows:

1. Define the desired degree distribution for your graph: an array $k = [k_1, \ldots, k_n]$ where k_i is the degree of node i and n is the desired number of nodes in the graph. (**Note that this differs from other models in that you actually *specify* the degree distribution beforehand**).
2. Now create a second array v that contains each index i of k exactly k_i times for each element in the degree sequence. Each of these i's represents a stub of the i^{th} node in the graph.
3. Next, randomly permute the elements of v.
4. Finally, create a random network by connecting adjacent pairs in v with an edge. In other words, connect up the nodes corresponding to the first and second elements, the third and fourth, and so on. Formally, v is of length $2m$ and the m undirected edges of the graph are (v_1, v_2), (v_3, v_4), ..., (v_{2m-1}, v_{2m}).

If this was confusing, don't worry! Figure 1 below provides a visual example of the algorithm that should clarify things. One important thing to note is that this algorithm can sometimes create improper (i.e., non-simple) networks with self-loops or multiple edges between two nodes – for the purposes of this exercise, this is acceptable.

Your task:

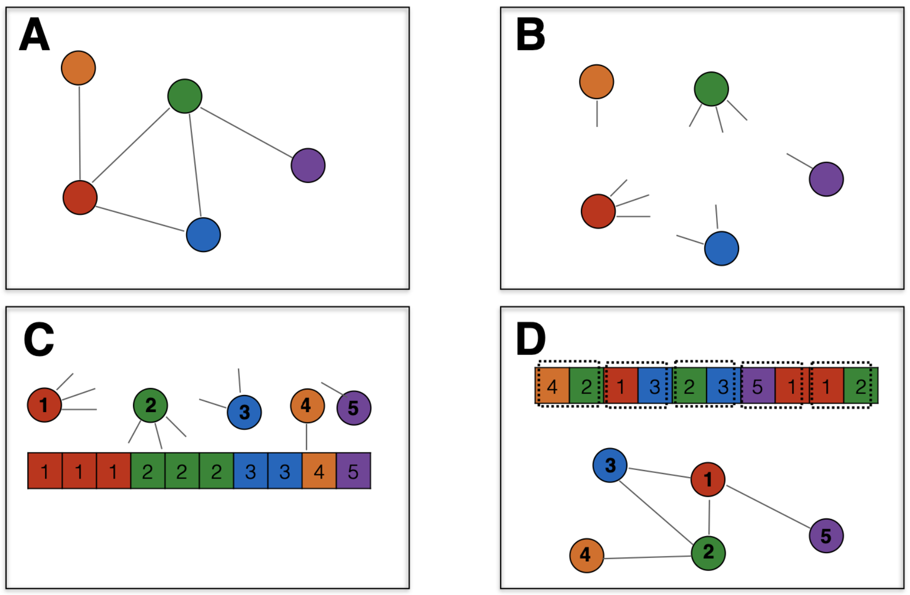

Figure 1: ***(A):*** *A toy example network where the nodes are labeled with different colors.* ***(B):*** *We break the toy network into "stubs", where each node of degree k has k half-edges coming out of it.* ***(C):*** *We label the nodes and make a vector v where each node's label is repeated in the vector k times (where k is that node's degree).* ***(D):*** *We take a random permutation of v and then connect adjacent pairs (dashed boxes) to create a random network.*

- Write a function that can generate a network using the configuration model. Your function should return a representation of the graph (e.g. a networkx graph). Please document and submit your code.
- Generate a random graph according to the configuration model that has the same degree distribution as the graph you generated in part (a).

(c) Finally, we will compare the networks generated in part (a) and part (b). Using your favorite graph visualization tool(s) (see Problem 2 on Homework 2), visualize each of the random graphs created by the two models, and present those visualizations here. What differences do you see in the graphs generated by the preferential attachment model compared to those generated by the configuration model? To get a better sense of the stochasticity, perhaps generate multiple instances of each random graph for the same parameters, and compare/contrast the visualizations you produce across these different instances.

2 Theory [30 points]

2.1 When monkeys type... [5 points]

We saw in class that one example of heavy-tailed distributions appearing in real-world situations is language – word frequencies. Here we'll see that this shouldn't be so surprising because even monkeys typing randomly end up with a heavy-tailed distribution on word frequencies!

Imagine a monkey randomly typing away on a keyboard[1]. The keyboard has n characters ($n > 1$ is a constant) and a space bar. The monkey hits the space bar with probability q and each of the n characters with equal probability $\frac{1-q}{n}$, with $\frac{1-q}{n} < q$. Each keystroke is independent.

(a) What is the probability that the monkey types a particular $c-$letter word (i.e., a specific c letters followed by a space)?

(b) Rank all the words the monkey can type in decreasing order of probability. Assume for simplicity that rank 1 is assigned to the 'empty' word of length 0.

Let's take $n = 10$ as an example. The empty word has rank 1. There are 10 possible 1-letter words, which we assign ranks 2-11 in an arbitrary order. Similarly, there are 10^2 possible 2-letter words, which we assign ranks 12-111, and so on. So any n-letter word will have some minimum and some maximum rank.

Let P_r be the probability of occurrence of a particular word with rank r. Show that

$$\lim_{r\to\infty} \frac{\log(P_r)}{\log(r)} = \log_n\left(\frac{1-q}{n}\right).$$

where $\log_n(\cdot)$ denotes logarithm to the base n.
Hint: What is the relationship between the rank r and the length of the word? Think about the above example.

(c) Interpret the result in part (b). Specifically, what can you say about the distribution of r (i.e. the distribution P_r)? Attempt to explain the difference between this distribution and the result of part (a).

2.2 Friendship Paradox, Part 2 [15 points]

We started our discussion of the "Friendship Paradox" in HW 1, where you showed that, in a graph with average degree μ and variance σ^2, the average degree of a neighbor is $\mu + \sigma^2/\mu$. That is, your average friend has more friends than you do. This is surprising! How dare they? But, so far, you've only shown a "weak" version of the paradox – one that holds in expectation. The goal in this problem is to show that the paradox can be strengthened. In particular, one can prove that the ratio of the degree of a random neighbor of a node to the degree of the node is *very likely* to be large.

Strengthening the paradox will however depend on the network model we consider. For this problem, we will use the *configuration model* you explored in Problem 1.2.

The configuration model is defined as follows. The model generates a random graph on n nodes with a desired, fixed degree sequence, i.e., with the number of nodes with degree d matching a specified f_d. To do this, nodes are created with "stub" edges according to the desired degree. Then, the two stubs are chosen uniformly at random and connected to form an edge. (Note that this can lead to self loops and multiple edges for any pair of nodes.) This is repeated until no stubs remain. See Figure 2 for an illustration.

[1]Relevant: https://xkcd.com/1530/

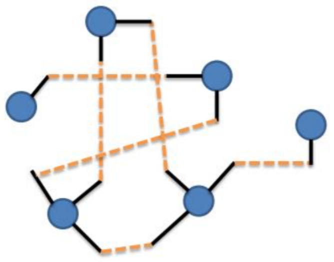

Figure 2: *An illustration of the configuration model for a degree sequence of 3, 3, 2, 2, 1, 1. Stubs are shown as solid lines and connections between them leading to edges in the graph are dashed.*

We will focus on configuration graphs with Pareto degree distributions, i.e., where $f_d = c \cdot d^{-\alpha}$ for some normalizing constant c and $\alpha \in (1, 2)$. You worked with the Pareto distribution in Problem 1.1, and here are some simple (and useful) facts to remember:

- Let δ be the highest degree in the graph. Since we consider Pareto degree distributions, we have that the highest degree satisfies $c\delta^{-\alpha} = 1$, and so $\delta = c^{1/\alpha}$.
- The number of nodes in the graph, n, is on the order of $\Theta(c)$.[2]
- The number of edges in the graph, $m = |E|$, is on the order of $\Theta(c^{2/\alpha})$.[3]

The goal of this problem is to prove the following, stronger version of the friendship paradox: For a configuration graph with a Pareto degree distribution having $\alpha \in (1, 2)$, consider a random node, v_1, and a random neighbor of it, v_2. The ratio between the degree of v_2 and the degree of v_1 is $\Omega(n^{1-(1/\alpha)-\epsilon})$ with constant probability.

Your task: To prove this result, we will take the following approach.

(i) Show that, the expected degree of a node chosen uniformly at random is $\Theta(c^{2/\alpha-1})$.

(ii) Use (i) to show that, with a constant probability, a node selected uniformly at random will have a "small" degree, specifically, a degree that is $O(c^{2/\alpha-1})$. *Hint: The Markov's inequality might be useful here.*

(iii) Show that the total number of stubs adjacent to nodes of "large" degree, i.e., degree $\geq c^{(1/\alpha)-\epsilon}$, is $\Theta(m)$. *Hint: Consider summing up the number of stubs for high-degree nodes.*

(iv) Given (iii), what is the probability that a uniformly chosen node has a neighbor with degree greater than $c^{(1/\alpha)-\epsilon}$?

(v) Combine (ii) and (iv) together to prove the result.

[2]To see this, note that $n = \sum_{d=1}^{\delta} cd^{-\alpha} = \Theta(c)$.

[3]To see this, note that $m = \frac{1}{2}\sum_{d=1}^{\delta} d \cdot cd^{-\alpha} = c\sum_{d=1}^{c^{1/\alpha}} d^{1-\alpha}$. Using the fact that $\sum_{d=1}^{t} d^{-\alpha} = \Theta(t^{1-\alpha})$ for the range of α we consider.

2.3 Exploiting the long tail [10 points]

Many companies have business models designed specifically to exploit the existence of heavy-tailed distributions. A nice pop-press book on the topic is "The long tail" by Chris Anderson.

Probably the most cited example of this is Amazon. Compared to a brick-and-mortar store, which can sell only a small number of popular items; Amazon can sell a huge variety of items, including ones that are not particularly popular. If the world was light-tailed, this advantage would be small, but since the world is heavy-tailed, these unpopular items still make up a large revenue source in the aggregate.

In this problem, we'll study a simple characterization of consumer behavior to show why Amazon's business model is so profitable. This model should be very reminiscent of the preferential attachment model we studied in class.

The model: For simplicity, consider a market where at each time, t, a consumer arrives and makes two purchases: one based on individualistic behavior and one based on social behavior. These purchases are made simultaneously, so the product just chosen as an individual purchase should not be considered as a social purchase at the same time.

We will define $m_i(t)$ to be the sales volume of product i at time t. The individual purchase is a product that no one has ever purchased before. For convenience, we assume that the product i is chosen at time $t = i$ for individual purchase, which can be seen as just a relabelling of the items. In other words, $m_t(t) = 1$.

The social purchase is a product that many consumers have purchased. The likelihood that the consumer chooses a given product at time t for his social purchase is proportional to the volume at the previous time step, $m_i(t-1)$, of the product, i.e., this purchase is a consequence of peer pressure. Assume that the first customer (at $t = 1$) does not make a social purchase.

Your task: You will analyze the model above to determine the distribution of $m_i(t)$. The following steps should guide you.

(a) What is the expected increase in market size ($m_i(t)$) for product i between time t and $t+1$? It helps to split this into two possible cases: whether $i = t$ or $i < t$, which relates them to whether they are eligible for purchases by individualistic or social behavior.

(b) Using mean value and continuous time approximation, as we did for the preferential attachment model in class, argue that the following differential equation describes the rate of change of volume of product i at time t:

$$\frac{dm_i(t)}{dt} = \frac{m_i(t)}{2t-1} \text{for } t > i.$$

(c) Solve the above differential equation for $m_i(t)$ using the boundary conditions to determine the constant(s). In this step, make the approximation $log(2t-1) \approx log(2t)$ for convenience of computation.

(d) What is the market share, $\frac{m_p(t)}{\Sigma_{j=1}^{t} m_j(t)}$, of the most popular product p? How does this relate to the success of Amazon?