ELEC 573 HW 3

Nicholas Glaze
nlcg2

2.

a) $\int_{0}^{\infty} \frac{\alpha x_0^{\alpha}}{x^{\alpha+1}}\,dx = \alpha x_0^{\alpha}\int_{x_0}^{\infty} x^{-\alpha-1}\,dx = \alpha x_0^{\alpha}\left(\frac{1}{-\alpha}x^{-\alpha}\right)\Big|_{x_0}^{\infty}$

$= \frac{\alpha x_0^{\alpha}}{-\alpha}\left(\frac{1}{\infty^{\alpha}} - \frac{1}{x_0^{\alpha}}\right)$

$= \frac{\alpha x_0^{\alpha}}{-\alpha}\left(-\frac{1}{x_0^{\alpha}}\right) = -x_0^{\alpha}\left(-\frac{1}{x_0^{\alpha}}\right)$

$= \boxed{1}$ => valid PDF

$F(X \geq x) = \int_{x}^{\infty} \frac{\alpha x_0^{\alpha}}{a^{\alpha+1}}\,da$

$= \alpha x_0^{\alpha}\left(\frac{1}{-\alpha}a^{-\alpha}\right)\Big|_{x}^{\infty}$

$= \frac{\alpha x_0^{\alpha}}{-\alpha}\left(\frac{1}{\infty} - \frac{1}{x^{\alpha}}\right) = -x_0^{\alpha}\left(-\frac{1}{x^{\alpha}}\right) = \left(\frac{x_0}{x}\right)^{\alpha}$

$\boxed{F(X \geq x) = \left(\frac{x_0}{x}\right)^{\alpha}}$

c) $l_n(\alpha) = \log\left(\prod_{i=1}^{n} \frac{\alpha x_0^{\alpha}}{x_i^{\alpha+1}}\right) = \sum_{i=1}^{n} \log\left(\frac{\alpha x_0^{\alpha}}{x_i^{\alpha+1}}\right) = \sum_{i=1}^{n}\left(\log(\alpha x_0^{\alpha}) - \log(x_i^{\alpha+1})\right)$

$\boxed{l_n(\alpha) = \sum_{i=1}^{n}\left(\log\alpha + \alpha\log x_0 - (\alpha+1)\log x_i\right)}$

$\frac{dl_n(\alpha)}{d\alpha} = \sum_{i=1}^{n}\left(\frac{1}{\alpha} + \log x_0 - \log x_i\right)$

$0 = \frac{n}{\alpha} + n\log x_0 - \sum_{i=1}^{n}\log x_i$

$\frac{n}{\alpha} = \sum_{i=1}^{n}\log x_i - n\log x_0 \Rightarrow \boxed{\hat{\alpha} = \frac{n}{\sum_{i=1}^{n}\log x_i - n\log x_0}}$

3.

a) $cl(G) = \frac{3 \cdot \#\text{triangles}}{\#\text{connected triples}}$, $d_i = 2k\ \forall i$

- # triangles per node, clockwise: choose 2 of the node's clockwise nbrs within k hops

avoid double-counting

$= \binom{k}{2} = \frac{k!}{2!(k-2)!} = \frac{k(k-1)}{2}$

- # connected triplets per node: $\binom{2k}{2} = \frac{2k(2k-1)}{2}$

$\Rightarrow cl(G) = \frac{3nk(k-1)}{2nk(2k-1)} = \frac{3(k-1)}{2(2k-1)} = \boxed{\frac{3k-3}{4k-2}}$

b) • # triangles;

- still $\frac{nk(k-1)}{2}$ original

- new:

• can be made by connecting nodes between $k+1$ and $2k$ hops away; prob that they're connected is $\underbrace{\frac{1}{2}n\,2k\cdot p}_{|E|} \Big/ \underbrace{\frac{n(n-1)}{2}}_{\binom{n}{2}} \approx \frac{2kp}{n}$

$\to$ however, as $n\to\infty$ this term goes to 0, so it's negligible.

$\Rightarrow \#\Delta \approx \frac{nk(k-1)}{2}$

• # triples;

- still $\frac{2nk(2k-1)}{2}$ original

- each new edge creates triples w/ all of the original $2k$ nodes @ either of its ends ($=(2k)^2$ triples)

• n possible, w/ prob. p

$\Rightarrow \underline{(2k)^2np}$

- new edges also create pairs w/ other new edges:

- ℓ new edges $\to \binom{\ell}{2} = \frac{\ell(\ell-1)}{2}$ new triples.

- expected # of new edges to a node is $\frac{2\overbrace{|E|}^{\text{original } E}p}{n} = \frac{2\left(\frac{2nk}{2}\right)p}{n} = \underline{2kp}$

$\Rightarrow$ exp. new triples is $n\frac{2kp(2kp-1)}{2} \approx (2k)^2p^2 \cdot \frac{n}{2}$

$$\Rightarrow cl(G) = \frac{\frac{1}{2}nk(k-1)\cdot 3}{\frac{1}{2}2nk(2k-1)+(2k)^2np+\underbrace{\frac{1}{2}n(2k^2)p^2}_{\text{ignore.}}} = \frac{\frac{3}{2}nk(k-1)}{nk(2k-1)+(2k)^2np} = \frac{\frac{3}{2}nk(k-1)}{2nk^2-nk+4k^2np}$$

$$= \frac{3k(k-1)}{2(2k^2-k+4k^2p)} = \frac{3k-3}{2(2k-1+4kp)} = \boxed{\frac{3k-3}{4k-2+8kp}}$$

Scanned with CamScanner

4. $i > m$:

a) $k_i(i) = m$ (each node connects to m others @ birth; node i is born @ time i)

b) $\boxed{\frac{dk_i(t)}{dt} \sim \frac{m}{t}}$ (a ratio of $\frac{m}{t}$ nodes have new connections added from time $t \to t+1$)

$k_i(t) = \int \frac{m}{t} dt = m \log t + c \to k_i(i) = m = m \log i + c$

$\Rightarrow c = m - m \log i$

$\Rightarrow k_i(t) = m \log t + m - m \log i$

$= m + m \log\left(\frac{t}{i}\right)$ $\square$

c) $k_i(t) = m + m \log\left(\frac{t}{i}\right) \geq d$

$m \log\left(\frac{t}{i}\right) \geq d - m$

$\frac{t}{i} \geq e^{\frac{d-m}{m}}$

$i \leq \frac{t}{e^{\frac{d-m}{m}}} \to$ t nodes by time t, so $\boxed{F(d) = e^{\frac{(m-d)}{d}}}$

$p(d) = -\frac{dF(d)}{dd} = -\frac{d}{dd} e^{\frac{m-d}{d}} = -\frac{d}{dd} e^{\frac{m}{d}-1} = -(e^{\frac{m}{d}-1}) \cdot -md^{-2}$

$p(d) = \boxed{\frac{1}{md^2} \cdot e^{\frac{m-d}{d}}}$

Scanned with CamScanner

In [ ]:

```python
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from collections import Counter, defaultdict
```

1. ER phase transitions

a) ER graph component sizes

In [ ]:

```python
def analyze_er_graph_components(n, p_range, graphs_per_p=20):
    means, sds = [], []
    for p in p_range:
        print('Analyzing ER graphs for p={}'.format(p))
        comp_sizes = []
        for i in range(graphs_per_p):
            G = nx.generators.fast_gnp_random_graph(n, p, seed=i)
            comp_sizes.append(max([len(comp) for comp in nx.connected_components(G)]))
        means.append(np.mean(comp_sizes))
        sds.append(np.std(comp_sizes))

    return np.array(means), np.array(sds)
```

In [ ]:

```python
n = 5000
p_range = np.logspace(-5, -2, num=20)
means, sds = analyze_er_graph_components(n, p_range)
```

```
Analyzing ER graphs for p=1e-05
Analyzing ER graphs for p=1.438449888287663e-05
Analyzing ER graphs for p=2.06913808111479e-05
Analyzing ER graphs for p=2.9763514416313192e-05
Analyzing ER graphs for p=4.281332398719396e-05
Analyzing ER graphs for p=6.158482110660267e-05
Analyzing ER graphs for p=8.858667904100833e-05
Analyzing ER graphs for p=0.00012742749857031334
Analyzing ER graphs for p=0.00018329807108324357
Analyzing ER graphs for p=0.00026366508987303583
Analyzing ER graphs for p=0.000379269019073225
Analyzing ER graphs for p=0.0005455594781168515
```

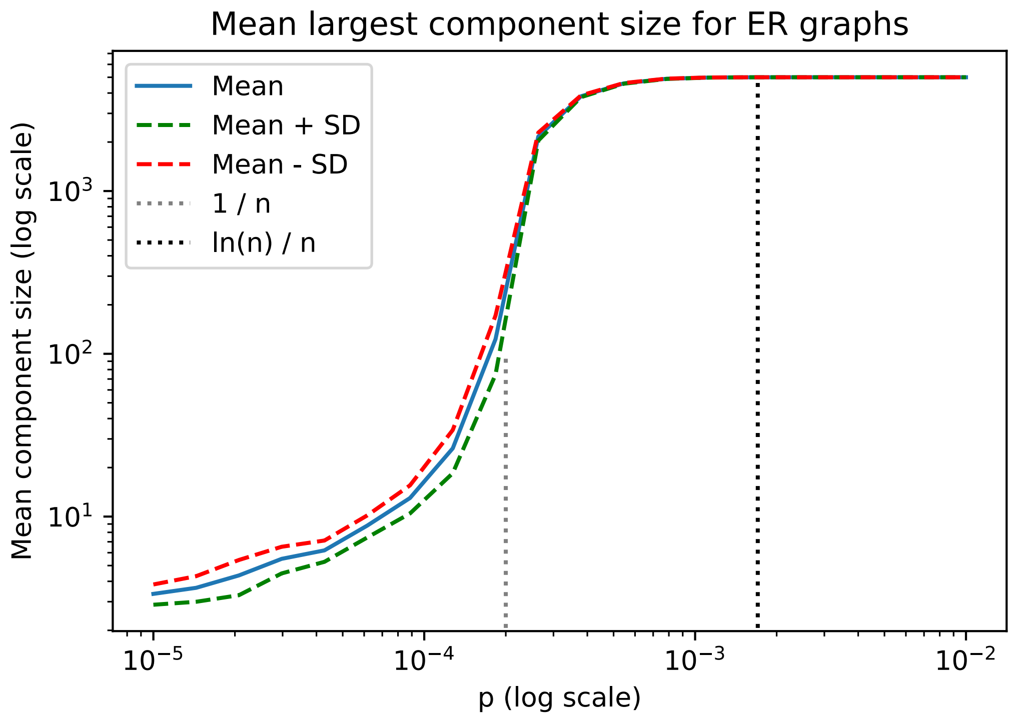
```
Analyzing ER graphs for p=0.0007847599703514606
Analyzing ER graphs for p=0.0011288378916846883
Analyzing ER graphs for p=0.001623776739188721
Analyzing ER graphs for p=0.002335721469090121
Analyzing ER graphs for p=0.003359818286283781
Analyzing ER graphs for p=0.004832930238571752
Analyzing ER graphs for p=0.0069519279617756054
Analyzing ER graphs for p=0.01
```

In [ ]:

```
def plot_with_sds(x, y, y_sds, n=5_000):
    plt.plot(x, y)
    plt.plot(x, y - y_sds, c='green', linestyle='dashed')
    plt.plot(x, y + y_sds, c='red', linestyle='dashed')

    # plot vertical lines at 1 / n and ln(n) / n
    plt.vlines(1 / n, 0, 100, linestyle='dotted', colors='gray')
    plt.vlines(np.log(n) / n, 0, 4900, linestyle='dotted', colors='black')

    plt.yscale('log')
    plt.xscale('log')
    plt.title('Mean largest component size for ER graphs')
    plt.xlabel('p (log scale)')
    plt.ylabel('Mean component size (log scale)')
    plt.legend(['Mean', 'Mean + SD', 'Mean - SD', '1 / n', 'ln(n) / n'])
```

In [ ]:

```
plot_with_sds(p_range, means, sds)
```



b) Interesting values of p

From class, we have that when p = lambda / n, lambda > 1 makes the largest component size converge to n, while lambda < 1 makes it converge to ln(n). Here, lambda = 1 represents a point at which the largest component size is transitioning rapidly, as it has the graph's peak slope. Also, I notice that as lambda shrinks less than 1, the size remains low (near ln(n) ~= 8), while as lambda grows above 1, the size approaches n.

Also, we have that when p = lambda * ln(n) / n, lambda < 1 makes the graph's probability of being connected converge to 0, while lambda > 1 makes it converge to 1. Here, to the left of ln(n) / n (i.e. lambda > 1), the mean component size has not yet converged to n, but to the right, the mean is equal to n and the standard deviation is 0, meaning that, as expected, all sampled graphs converge to being connected.

2. Fitting power-law exponents

b) Pareto sampling

```
In [ ]:
```

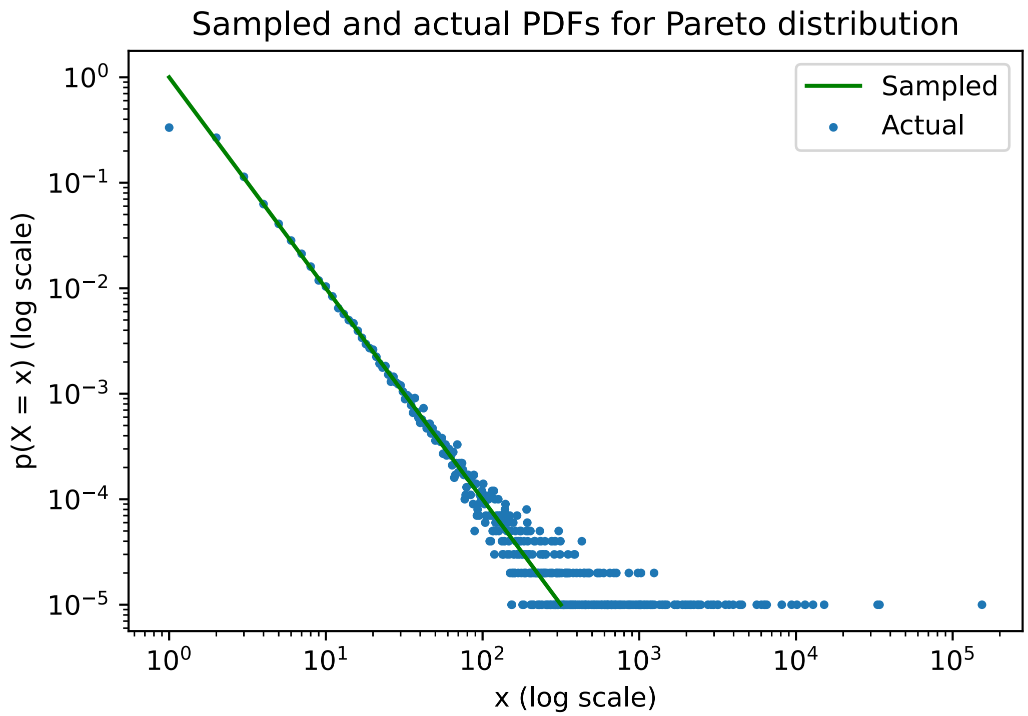
```python
def sample_pareto(alpha=1, x_0=1, size=100_000):
    # sample uniformly in [0, 1] and apply the pareto function to each sample
    pareto_func = np.vectorize(lambda u:  x_0 / (u**(1 / alpha)))
    u_samples = np.random.sample(size=size)
    return pareto_func(u_samples)
```

In [ ]:
```python
def compute_pdf(samples):
    # round samples to nearest integer
    sample_ints = np.rint(samples).astype(int)

    # count occurrences of each integer
    cts = Counter(sample_ints)
    x = np.array(sorted(cts.keys()))
    px_x = np.array([cts[i] for i in sorted(cts.keys())]) / len(sample_ints)

    return x, px_x
```

In [ ]:
```python
def compute_ccdf(x, px_x):
    # compute running total from right to left
    s = 0
    fbarx_x = np.zeros(x.shape)
    for i in range(len(x) - 1, -1, -1):
        s += px_x[i]
        fbarx_x[i] = s

    return x, fbarx_x
```

In [ ]:
```python
def plot_observations(x, y, alpha=1):
    plt.scatter(x, y, s=5)
    plt.xscale('log')
    plt.yscale('log')
    y_expected = 1 / x ** (alpha + 1)
    plt.plot(x[y_expected >= 1e-5], y_expected[y_expected >= 1e-5], c='green')
    plt.title('Sampled and actual PDFs for Pareto distribution')
    plt.legend(['Sampled', 'Actual'])
    plt.xlabel('x (log scale)')
    plt.ylabel('p(X = x) (log scale)')
```

In [ ]:
```python
plot_observations(x, px_x)
```



d) Estimating alpha

In [ ]:

```
def estimate_alpha_pdf_ls(samples):
    x, p_x = compute_pdf(samples)

    x_log = np.log(x)
    y_log = np.log(p_x)

    alpha_pred = -np.polyfit(x_log, y_log, deg=1)[0]
    return alpha_pred
```

In [ ]:

```
def estimate_alpha_ccdf_ls(samples):
    x, px_x = compute_pdf(samples)
    x, fbarx_x = compute_ccdf(x, px_x)

    x_log = np.log(x)
    y_log = np.log(fbarx_x)

    # CCDF exponent is (alpha - 1)
```

```
    alpha_pred = -np.polyfit(x_log, y_log, deg=1)[0] + 1

    return alpha_pred
```

In [ ]:

```
def estimate_alpha_mle(samples, x_0=1):
    n = len(samples)

    # from c), MLE is alpha = n / (sum_i(log(x_i)) - (n * log(x_0)))
    return n / (np.log(samples).sum() - (n * np.log(x_0)))
```

In [ ]:

```
def eval_alpha_estimate(estimate_alpha, name, samples=100):
    preds = []
    for _ in range(samples):
        samples = sample_pareto()
        preds.append(estimate_alpha(samples))
    m, sd = np.mean(preds), np.std(preds)
    print('{}:'.format(name))
    print('Mean: {}, SD: {}'.format(m, sd))
```

In [ ]:

```
eval_alpha_estimate(estimate_alpha_pdf_ls, 'Least-squares on PDF');
```

```
Least-squares on PDF:
Mean: 0.9587292458008777, SD: 0.048126100747554156
```

In [ ]:

```
eval_alpha_estimate(estimate_alpha_ccdf_ls, 'Least-squares on CCDF');
```

```
Least-squares on CCDF:
Mean: 1.9955482093481538, SD: 0.03403874599398693
```

In [ ]:

```
eval_alpha_estimate(estimate_alpha_mle, 'Log-likelihood maximization');
```

Least-squares regression on the CCDF gives a very poor result, while least-squares on the PDF and the log-likelihood MLE both give results somewhat close to the true value. The MLE almost exactly approximates true value, while the PDF least-squares is off by a bit more; also, the MLE has near-0 standard deviation, so since it is also the least biased, it is clearly the best estimate.

note: my numbers don't line up with Dr. Segarra's for i and ii, but he looked through my code during office hours and said that what I'm doing seems right, so we're not sure why my results are different. I plotted the PDF, CCDF, and fitted lines on a log-log graph below, if that helps at all.

In [ ]:
```
x, p_x = compute_pdf(sample_pareto())
x, fbarx_x = compute_ccdf(x, p_x)

x_log = np.log(x)

m, c = np.polyfit(x_log, np.log(p_x), deg=1)

plt.plot(x_log, [a * m + c for a in x_log])
plt.scatter(x_log, np.log(p_x), s=5)


m, c = np.polyfit(x_log, np.log(fbarx_x), deg=1)

plt.plot(x_log, [a * m + c for a in x_log])
plt.scatter(x_log, np.log(fbarx_x), s=5)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x7f18a828ee50>
```