Sunil Thapa
W0779269

1. Reported Metrices

	ER	BA	Real
N	1000	1000	115
<k>	4.1420	3.9920	10.6609
<C>	0.0060	0.0244	0.4032

2. Plots

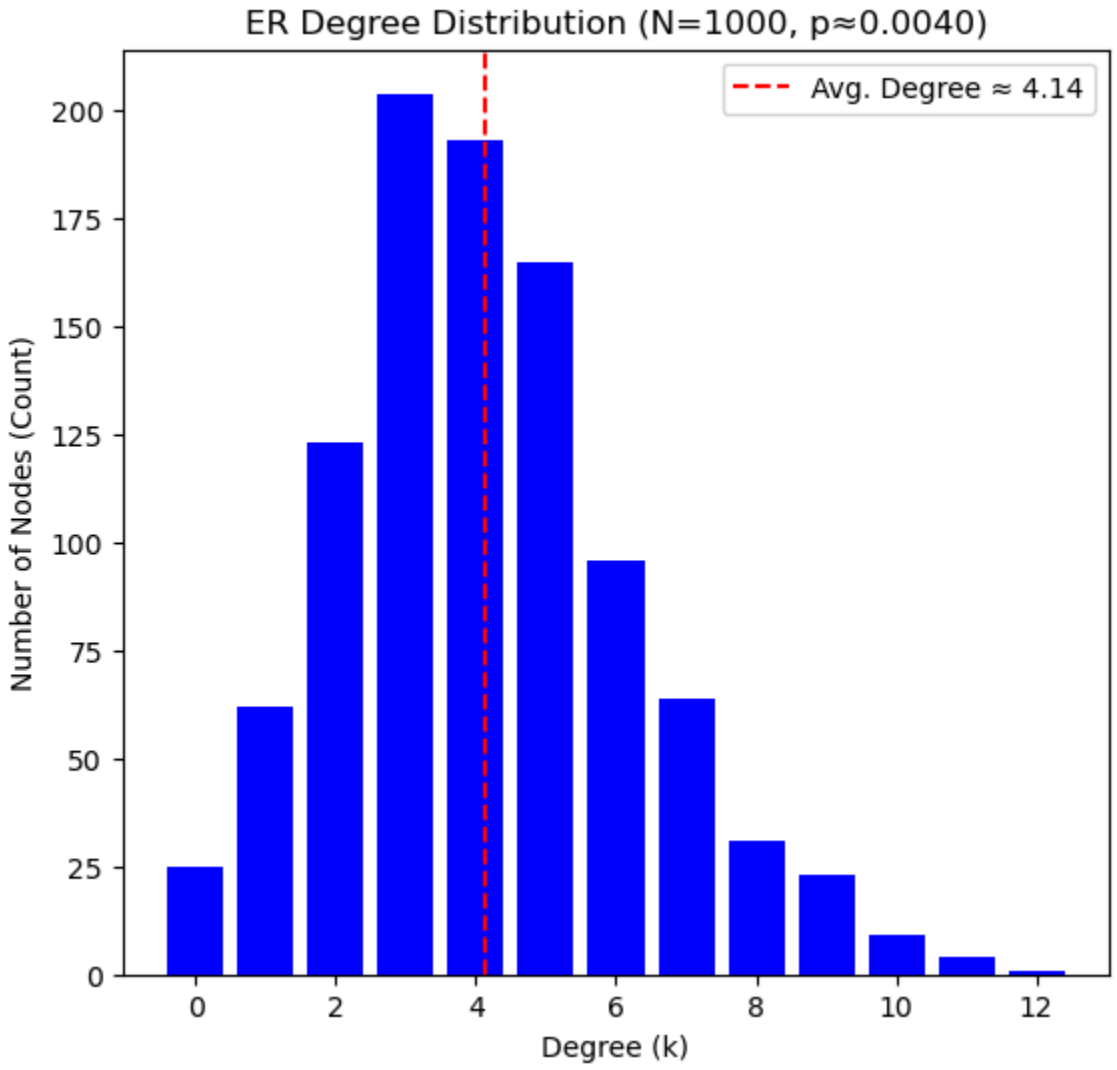


Fig: Bar plot for ER graph

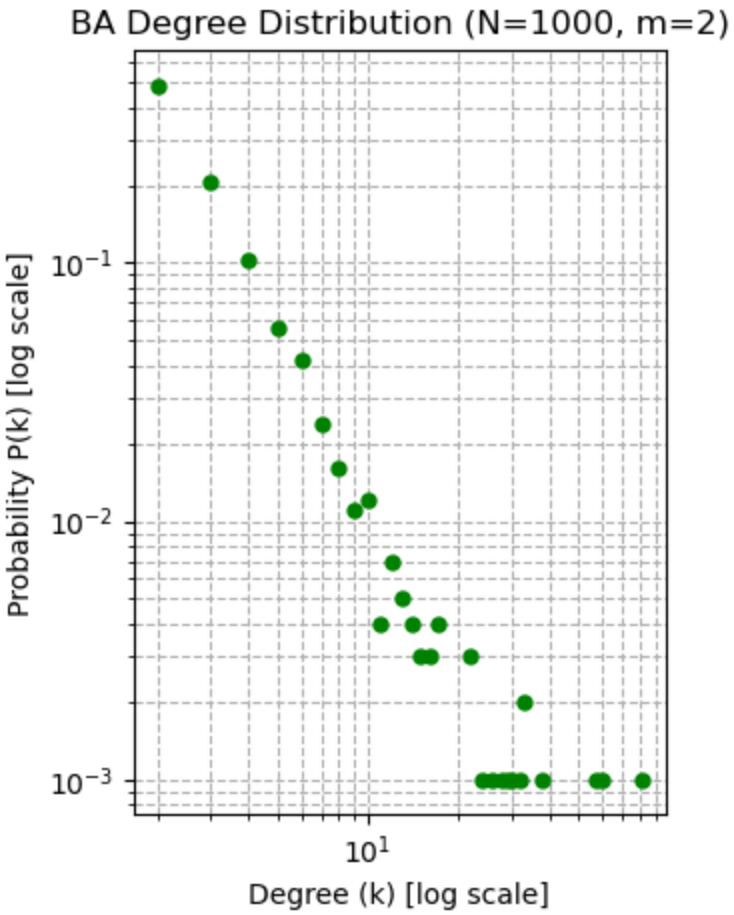


Fig: log-log plot for BA graph

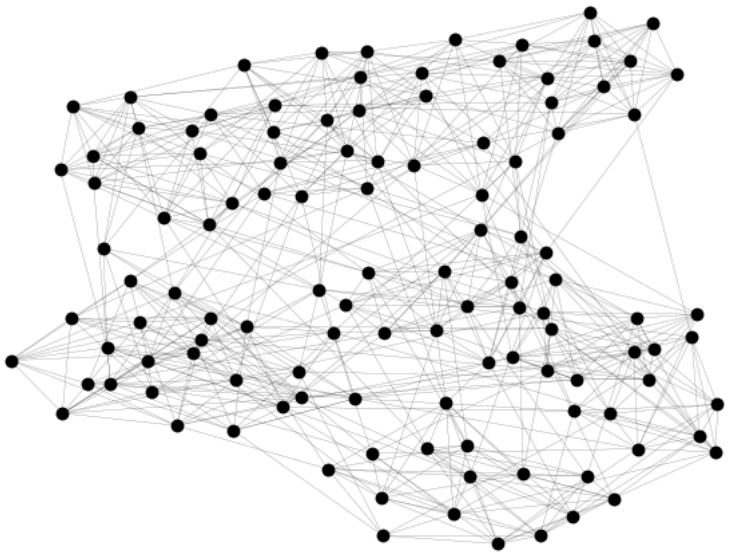


Fig: GML graph for Football Network

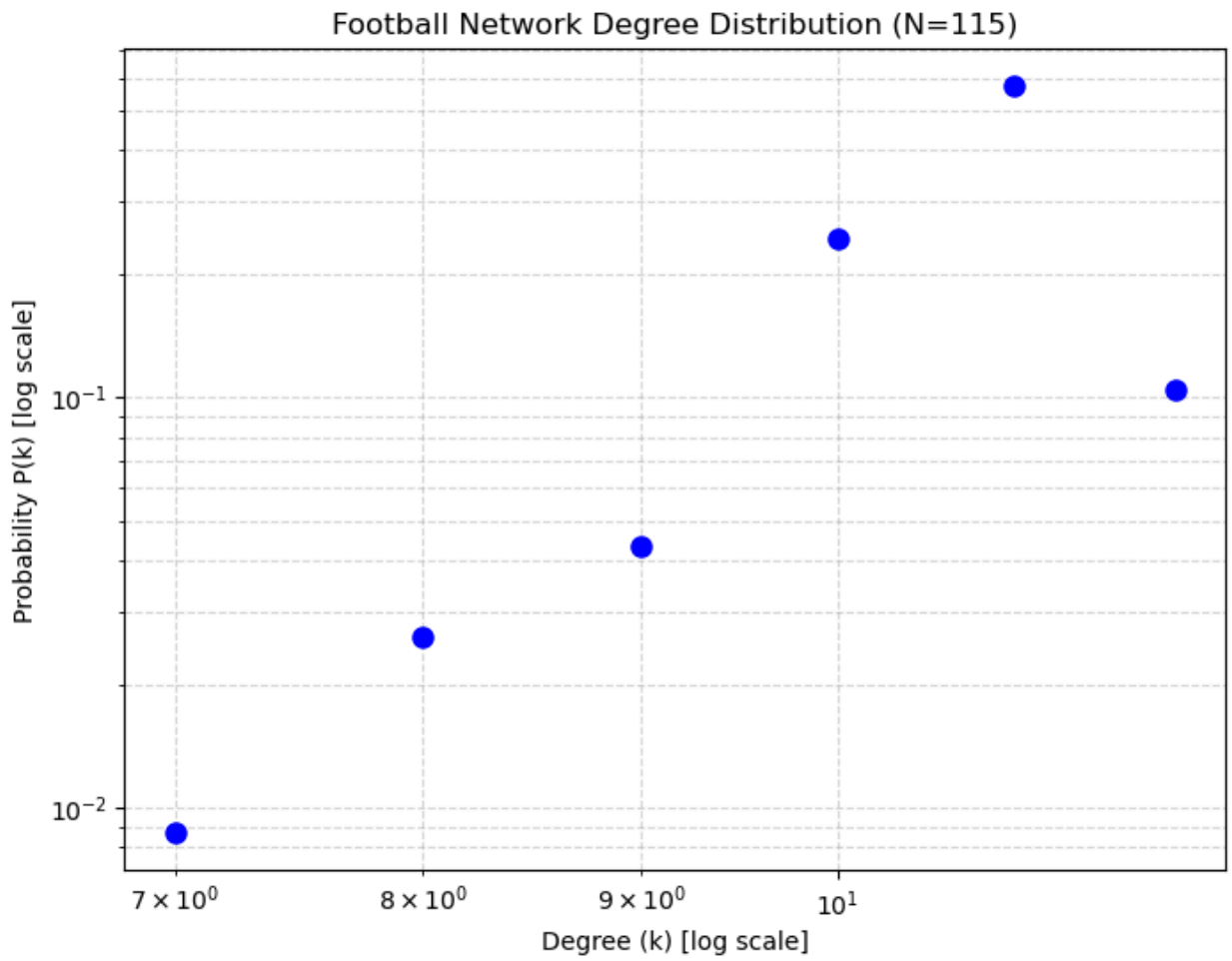


Fig: Degree distribution of the Football Network

3. Interpretation

The football network (<C> ≈ 0.403) has the highest clustering by a massive margin, while the Erdos-Renyi (ER) model (<C> ≈ 0.006) has the lowest clustering. The BA model's clustering (<C> ≈ 0.0244) is much higher than the random ER model's, but it's still tiny compared to the real network. This reveals a key weakness of the BA model.

- Why is ER lowest: Connections are purely random. The probability of two of a node's neighbours also being connected (forming a triangle) is minuscule in the BA model.
- Why Football is highest: This network represents a real-world social structure. It's built on communities (the conferences). Teams within the same conference play each other far more often, creating dense, tight-knit clusters. This "friends of my friends are also my friends" effect is called triadic closure, and it's very strong in real networks.
- Why BA is in the middle: The BA model's "preferrential attachment" indirectly creates some clustering. When a new node connects to a popular hub, it might also connect to another node that is also connected to that same hub, forming a triangle. However, the BA model has no explicit mechanism for triadic closure. It only models new nodes connecting in, not existing nodes forming new links (e.g., two teams in different

conferences deciding to play because they have a common rival). This is why its clustering coefficient is much lower than a real-world network.

The Barabási–Albert (BA) model is a much better approximation.

- The Football Network's log-log plot (like the BA plot) shows a "long tail," which is characteristic of a power-law distribution. This means there are many teams with a low number of games (low k) and a few "hub" teams that play a very high number of games (high k).
- The ER model's Poisson distribution is a terrible fit. It predicts that almost all teams would play the average number of games (≈ 11) and that hubs are statistically impossible. This is clearly not true for the real network.

The key mechanism responsible for this power-law shape is Preferential Attachment.

- Growth (the network adding nodes over time) is a necessary component, but it's not the whole story.
- Preferential Attachment is the "rich-get-richer" rule: new nodes (or in this case, new games being scheduled) are more likely to connect to nodes that are already well-connected. A well-known "hub" team (like a traditional powerhouse) is more likely to schedule games with other teams than a small, isolated team. This mechanism allows a few nodes to acquire a huge number of connections, creating the power-law tail. The ER model lacks this mechanism entirely, leading to its homogeneous, hub-less structure.

homework-2

November 5, 2025

[11]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import collections
```

[6]:
```
N = 1000
k = 4
```

0.0.1 Simulation of Erdos-Renyl Graph Model

[7]:
```
p = k/(N-1)
```

[9]:
```
G = nx.erdos_renyi_graph(N, p)
print(f"Simulation Parameters: N = {N}, p = {p:.5f}")
```

```
Simulation Parameters: N = 1000, p = 0.00400
```

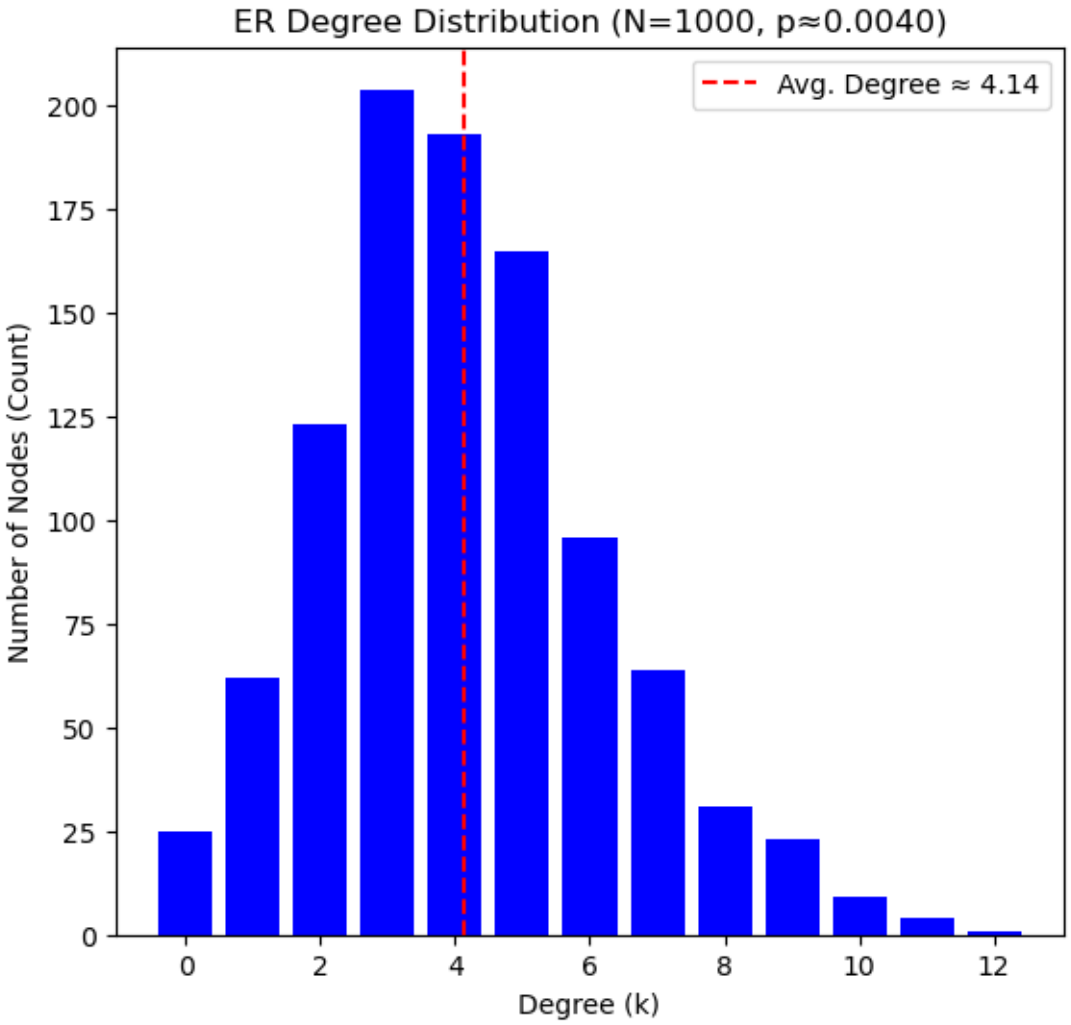
[10]:
```
degrees_er = [d for n,d in G.degree()]
avg_k_er = sum(degrees_er) /N
avg_c_er = nx.average_clustering(G)

print(f"Final Average Degree (<k>): {avg_k_er:.4f}")
print(f"Average Clustering Coefficient (C): {avg_c_er:.4f}")
```

```
Final Average Degree (<k>): 4.1420
Average Clustering Coefficient (C): 0.0060
```

[12]:
```
degree_counts_er = collections.Counter(sorted(degrees_er))
deg_er, cnt_er = zip(*degree_counts_er.items())

plt.figure(figsize=(14, 6))

# ER Plot
plt.subplot(1, 2, 1)
plt.bar(deg_er, cnt_er, width=0.80, color="b")
plt.title(f"ER Degree Distribution (N={N}, p {p:.4f})")
plt.xlabel("Degree (k)")
```

```
plt.ylabel("Number of Nodes (Count)")
plt.axvline(avg_k_er, color='red', linestyle='--', label=f'Avg. Degree ␣
↪{avg_k_er:.2f}')
plt.legend()
```

[12]: <matplotlib.legend.Legend at 0x105509ca0>



0.0.2 Barabasi-Albert (BA) Model

[13]:
```
m = 2
```

[14]:
```
BA_G = nx.barabasi_albert_graph(N, m)
print(f"Simulation Parameters: N={N}, m={m}")
```

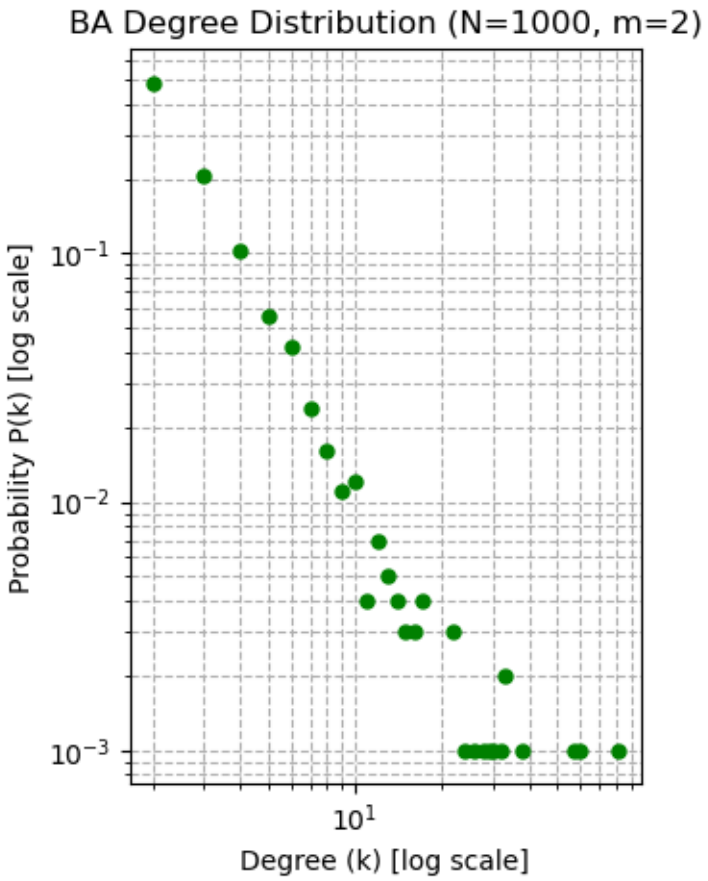
```
Simulation Parameters: N=1000, m=2
```

[16]:
```
degrees_ba = [d for n, d in BA_G.degree()]
avg_k_ba = sum(degrees_ba) / N
avg_c_ba = nx.average_clustering(BA_G)

print(f"Final Average Degree (<k>): {avg_k_ba:.4f}")
print(f"Average Clustering Coefficient (C): {avg_c_ba:.4f}")
```

```
Final Average Degree (<k>): 3.9920
Average Clustering Coefficient (C): 0.0244
```

[17]:
```
degree_counts_ba = collections.Counter(sorted(degrees_ba))
deg_ba, cnt_ba = zip(*degree_counts_ba.items())

pk_ba = [c / N for c in cnt_ba]
```

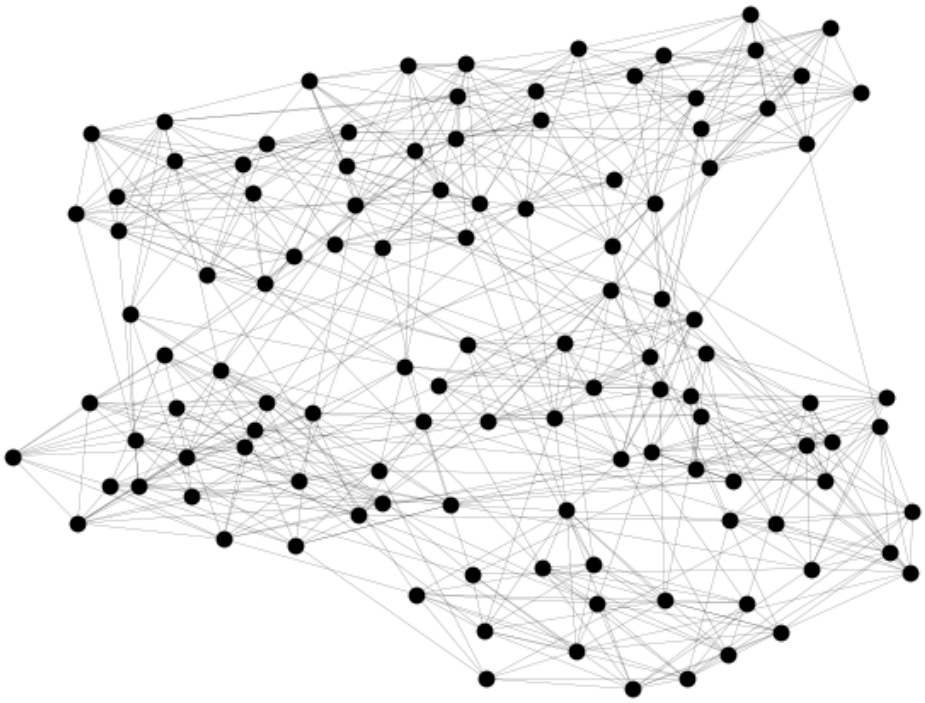
[18]:
```
plt.subplot(1, 2, 2)
plt.loglog(deg_ba, pk_ba, 'go', markersize=5)
plt.title(f"BA Degree Distribution (N={N}, m={m})")
plt.xlabel("Degree (k) [log scale]")
plt.ylabel("Probability P(k) [log scale]")
plt.grid(True, which="both", ls="--")

# Display plots
plt.tight_layout()
plt.show()
```



[19]:
```
import urllib.request
import io
import zipfile

url = "http://www-personal.umich.edu/~mejn/netdata/football.zip"

sock = urllib.request.urlopen(url)  # open URL
s = io.BytesIO(sock.read())  # read into BytesIO "file"
sock.close()

zf = zipfile.ZipFile(s)  # zipfile object
txt = zf.read("football.txt").decode()  # read info file
gml = zf.read("football.gml").decode()  # read gml data
# throw away bogus first line with # from mejn files
gml = gml.split("\n")[1:]
```

[20]:
```
G = nx.parse_gml(gml)
```

[23]:
```
options = {"node_color": "black", "node_size": 50, "linewidths": 0, "width": 0.
↪1}

pos = nx.spring_layout(G, seed=1969)  # Seed for reproducible layout
nx.draw(G, pos, **options)
plt.show()
```

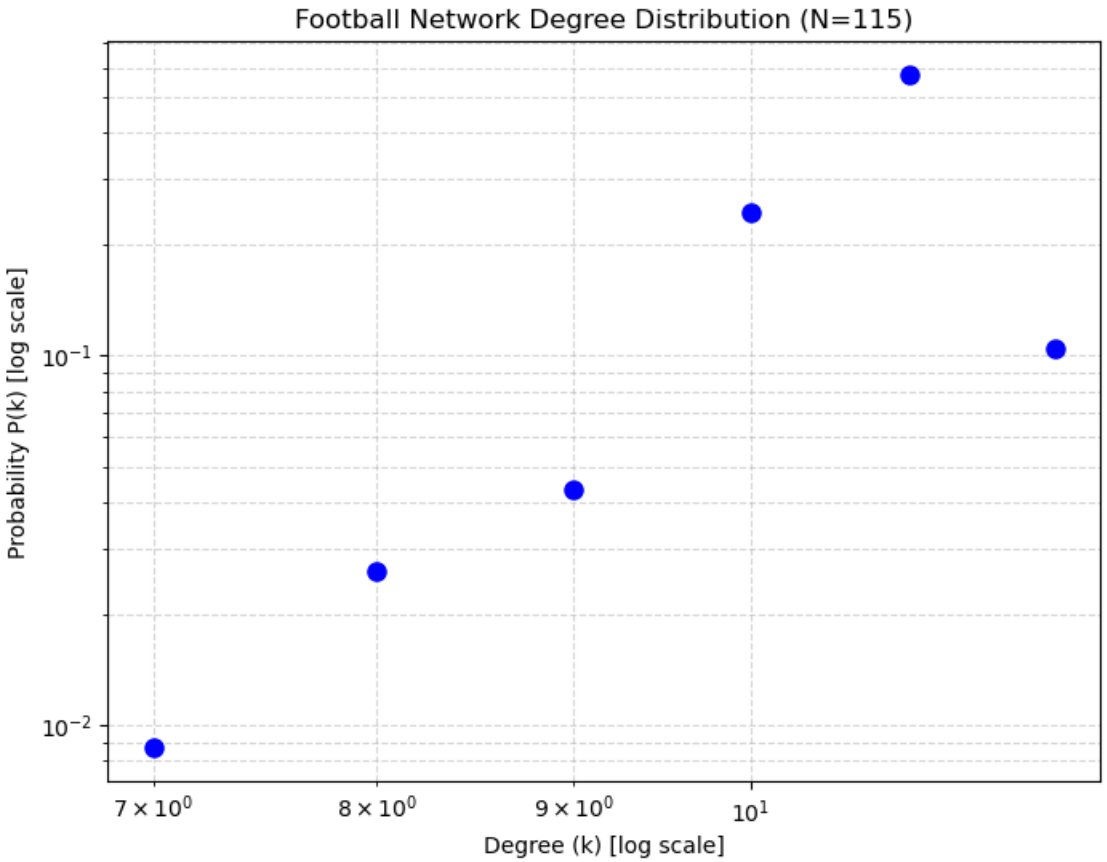


[25]:
```
N_real = G.number_of_nodes()
M_real = G.number_of_edges()

degrees_real = [d for n, d in G.degree()]
avg_k_real = sum(degrees_real) / N_real

# Calculate average clustering coefficient
avg_c_real = nx.average_clustering(G)

print(f"Network: College Football")
print(f"Nodes (N): {N_real}")
print(f"Edges (M): {M_real}")
print(f"Final Average Degree (<k>): {avg_k_real:.4f}")
```

```
print(f"Average Clustering Coefficient (C): {avg_c_real:.4f}")
```

```
Network: College Football
Nodes (N): 115
Edges (M): 613
Final Average Degree (<k>): 10.6609
Average Clustering Coefficient (C): 0.4032
```

[26]:
```
degree_counts_real = collections.Counter(sorted(degrees_real))
deg_real, cnt_real = zip(*degree_counts_real.items())

pk_real = [c / N_real for c in cnt_real]

plt.figure(figsize=(8, 6))
plt.loglog(deg_real, pk_real, 'bo', markersize=8)
plt.title(f"Football Network Degree Distribution (N={N_real})")
plt.xlabel("Degree (k) [log scale]")
plt.ylabel("Probability P(k) [log scale]")
plt.grid(True, which="both", ls="--", alpha=0.5)
plt.show()
```



[ ]: