# Practical 2.2 - Network Percolations

Jan Izquierdo Ramos, Jaume Jurado Sanchez

October 2024

**Name:** Jan Izquierdo, Jaume Jurado

## SYSTEMS AND NETWORK BIOLOGY - PRACTICAL 2 (PART 2)

## Network percolation

---

To submit your report, answer the questions below and save the *notebook* clicking on `File > Download as > iPython Notebook` in the menu at the top of the page. **Rename the notebook file** to `practical9_name1_name2.ipynb`, where `name1` and `name2` are the first surnames of the two team members (only one name if the report is sent individually). Finally, **submit the resulting file through the *Aul@-ESCI***.

*IMPORTANT REMINDER: Before the final submission, remember to* ***reset the kernel*** *and re-run the whole notebook again to check that it works.*

---

In this session we will study the percolation transition in random networks. To that end we will use the Python package `NetworkX`.

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
from scipy.stats import poisson
```

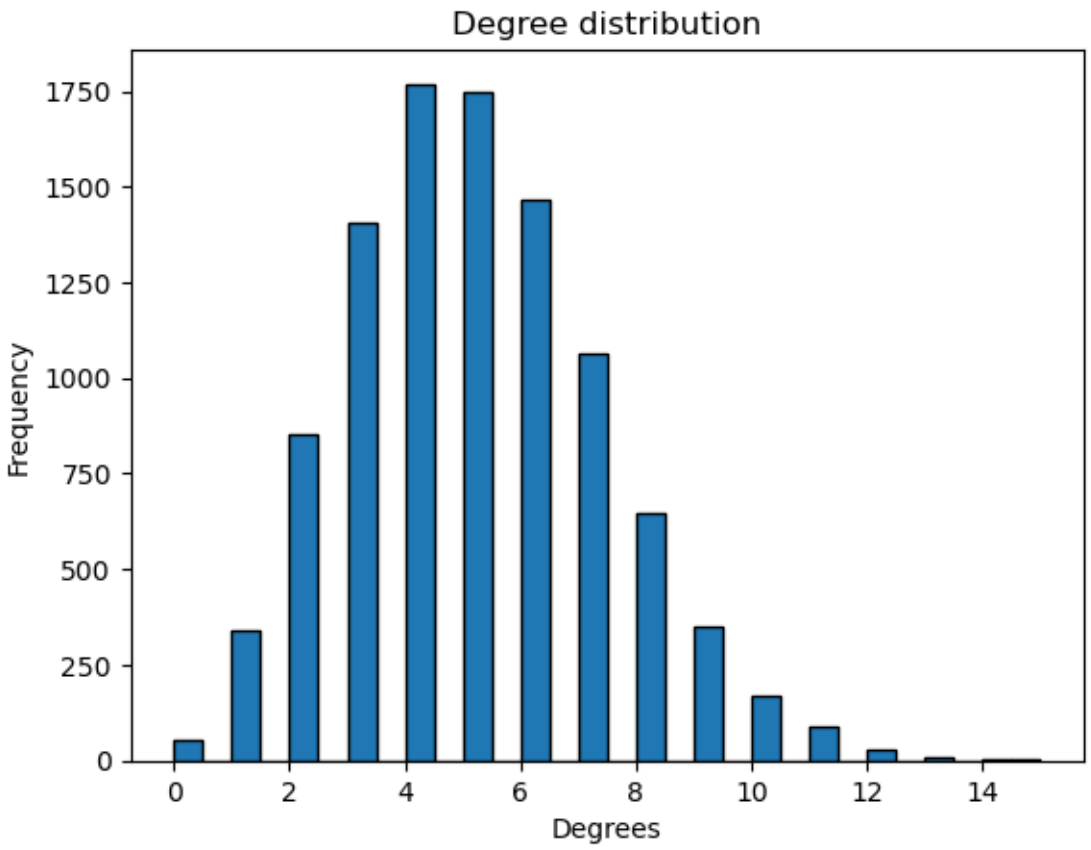
First, use the **NetworkX** function `erdos_renyi_graph` to generate an Erdös-Renyi random network with $n = 10000$ nodes and connection probability $p = 0.0005$. Plot the degree distribution.

[3]:
```
nw=nx.erdos_renyi_graph(n=10000, p=0.0005)
nw.number_of_nodes()
degrees=dict(nw.degree()) #node:degreee
np.mean(list(degrees.values()))
```

[3]: 4.9832

[4]:
```
plt.hist(x=degrees.values(), bins=30, edgecolor="Black")
plt.title("Degree distribution")
plt.xlabel("Degrees")
plt.ylabel("Frequency")
```

[4]:
```
Text(0, 0.5, 'Frequency')
```




---

The connections from the created graph have a resemblance to a poisson distribution, we will try to compare it to a real poisson distribution and see if the data fits.

---

Verify that the degree distribution obtained above matches a Poisson distribution with mean equal to the average degree of the network.

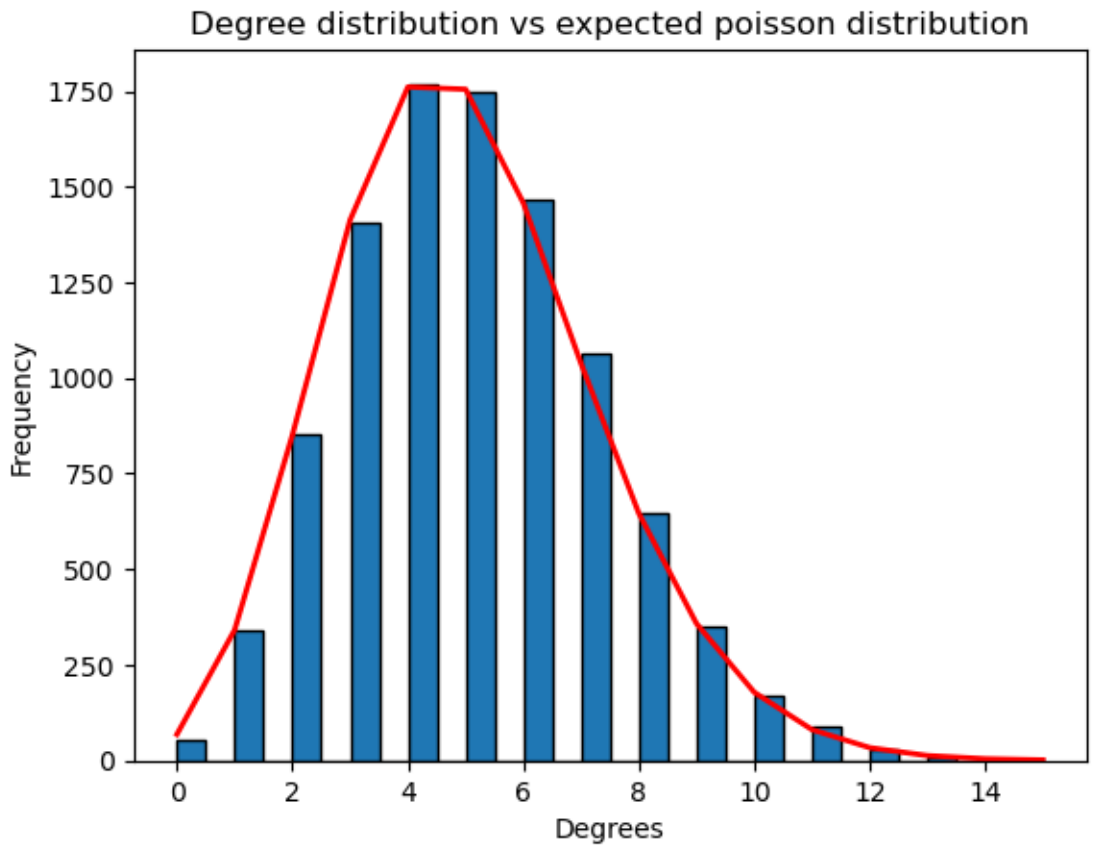
[5]:
```
#degrees.values()
avg_degree = np.mean(list(degrees.values()))

allDegrees = np.arange(0, max(degrees.values()) + 1)
```

```
poisson_samples=poisson.pmf(allDegrees, avg_degree) * len(degrees.values())␣
 ↪#Multiply by 10000 to scale the poisson dstribution data to our data

print(len(degrees.values()))
plt.hist(x=degrees.values(), bins=30, edgecolor="Black")
plt.plot(allDegrees, poisson_samples, linewidth=2, color="red")
plt.title("Degree distribution vs expected poisson distribution")
plt.xlabel("Degrees")
plt.ylabel("Frequency")
plt.show()
```

```
10000
```




We will now study the percolation transition discussed in class. To that end, generate an ensemble of Erdös-Renyi networks (e.g. 100 networks) with 1000 nodes each. Compute for each network the fraction of nodes that belong to the largest connected component, and calculate its average over the ensemble of networks with a given connection probability $p$. Repeat this calculation for a range of values of $p$ between 0 and $10/n$, with $n$ being the number of nodes of the network.

[6]:
```
np.random.seed(1)
def con_comp_nodes(n, p):
    nw=nx.erdos_renyi_graph(n, p)
    u_nw=nw.to_undirected()
    m_component=max(nx.connected_components(u_nw))
    m_nw=nw.subgraph(m_component).to_undirected().copy()
    nnodes=m_nw.number_of_nodes()
    return nnodes

n=1000
plist=np.linspace(0,10/n, 10)
crossPmean=[]
graph_crossPmean=[]

#p in a nested for loop (outer)use np.linspace(0, 10/n, k) to get list of values␣
 ↪to use as p (k is len of list you want to obtain)
for p in plist:
    print("P is:",p)
    nnode_list=[]
    graph_nnlist=[] #For the plotting of the transition curve later on
    for _ in range(100):
        nnode_list.append(con_comp_nodes(n, p))
        graph_nnlist.append(con_comp_nodes(n, p)/n) #Keep a separate list so␣
 ↪that we can still calculate the averages

    avg_nnode=np.mean(nnode_list)
    crossPmean.append(avg_nnode)
    graph_crossPmean.append(np.mean(graph_nnlist)) #The separate list allows to␣
 ↪print a coherent result for our values and still keep a list to calculate our␣
 ↪graph

print(crossPmean)


print("The average number of nodes in the largest connected component acorss␣
 ↪different p values is", np.mean(crossPmean))
```

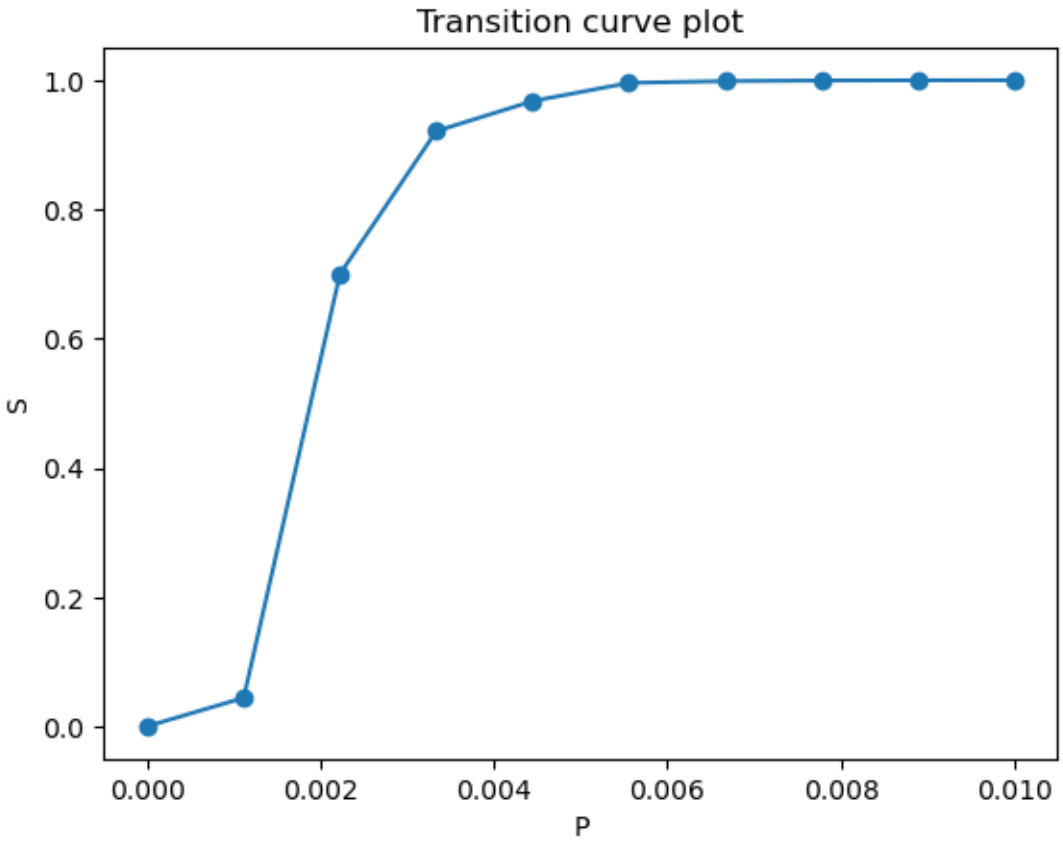
```
P is: 0.0
P is: 0.0011111111111111111
P is: 0.0022222222222222222
P is: 0.003333333333333333
P is: 0.0044444444444444444
P is: 0.005555555555555556
P is: 0.006666666666666666
P is: 0.0077777777777777776
P is: 0.008888888888888889
P is: 0.01
```

```
[1.0, 55.59, 695.44, 940.41, 987.49, 996.06, 998.91, 999.61, 999.89, 999.97]
The average number of nodes in the largest connected component acorss different
p values is 767.437
```

Finally, plot the transition curve.

[8]:
```
plt.scatter(plist, graph_crossPmean)
plt.plot(plist, graph_crossPmean)
plt.xlabel("P")
plt.ylabel("S")
plt.title("Transition curve plot")
```

[8]:
```
Text(0.5, 1.0, 'Transition curve plot')
```




---

When increasing the probability of connections (x axis) we can observe that the number of nodes in the graph does increase, altough a huge difference can be observed between p=0 and p=0.3, the curve stabilizes from there onwards at a much higher number of connections.

---

[ ]: