Statistical study of the eigenvalues of random graphs through Random Matrix Theory

Vannsh Jagtiani

Abstract

Randomness is inherent in nature. From bacterial population growth to noise in electrical circuits, stochastic processes highlight the randomness which is evident in the inherent nature of physical phenomena. In this work we perform a numerical experiment by studying the statistics of eigenvalues of adjacency matrices of randomly connected graphs. We motivate graph theory and highlight the method of generating random graphs used for this study. We then look at the eigenvalues of the adjacency matrices to look at their eigen- spectrum. We further statistically observe the spacing distribution and find that the spacing exhibits Gaussian Orthogonal Ensemble (GOE) distribution.

Introduction

Random matrix theory is a field of mathematics which has been used to understand complex systems. The idea came by observation of the energies of heavy nuclei which exhibit complex many body interactions. This leads to energy levels not following a specific mathematical function. Wigner postulated, if the underlying processes are so complex, that they almost appear random, can they show some statistical relations? This led to exploration of random matrices and their eigenvalue distribution.

In this work, we work with the hypothesis that the eigenvalues of the adjacency matrix of a randomly connected graph demonstrates features from random matrix theory due to inherent randomness in the connectedness of the graph. We do this by using python. We create 1000 randomly connected graphs, each with 1000 nodes, giving us 1000000 eigenvalues, making it a large number for statistical significance. We then arrange these eigenvalues in the ascending order and calculate the spacings by subtracting adjacent values. These obtained values are then plotted as a histogram and compared with the GOE distribution.

GOE distribution is one of the distributions used to study many body interactions. This is visible in many nuclear reactions, molecular interactions, and chaotic quantum systems. If we are able to draw a relation between the said distribution and randomly connected graphs, which can be easily modeled using modern tools like networkx in python, we can suggest a possible connection between the two. This can help us model complex nuclear reactions or similar many body phenomena using randomly connected graphs. This makes this work interesting and worth exploring in the broader picture of applications of GOE and random matrix theory.

Overview to graph theory and random graphs

Many real-world problems can be described by means of a diagram consisting of a set of vertices together with lines joining certain pairs of these vertices. The points could represent

various entities, all the way from cities connected by major flight lines to houses in a colony joined by a set of streets. Such diagrams are known as graphs (West, D. B. 2001). A graph is essentially made of a set of “vertices” or “nodes” which are joined together by “edges”. Such problems often can be thought of many players interacting in some manner to a certain dynamic.

The history of graph theory may be specifically traced back to as early as 1735 when Swiss mathematician Leonard Euler solved the Königsberg bridge problem. The Königsberg bridge problem was concerned with the possibility of finding a path over every one of seven bridges that span a forked river flowing past an island but without crossing any bridge more than once. Euler argued that no such path existed. His proof essentially proved the first theorem in graph theory. Since the period, other discoveries in graph theory soon followed such as A.F. Mobius’ Bipartite Graphs in 1840, the concept of trees by Gustav Kirchhoff in 1845, Thomas Gutherie’s famous four-color problem in 1852 and William R. Hamilton’s Hamiltonian Graphs in 1856. In the year 1941, Ramsey worked on colorations which led to the identification of another branch of Graph Theory called extremely Graph Theory. The study of asymptotic graph connectivity gave rise to random graph theory.

Graph theory has significant real-life applications in subjects like Chemistry, Biology, Physics, Computer Sciences, and other fields of Operational Research (Mondal, B., & De, K. 2017). The specific fields that extensively use graphs are Biochemistry, Electrical Engineering through communication networks and coding theory, Computer Science through algorithms and computations, Operation Research through scheduling and detailing.

The specific study of Random Graphs in their own right began in the paper originally published under the name of Erdós and Rényi (P. Erdós and A. Rényi, Budapest). A random graph as the name suggests, a graph generated at random with a number of edges and vertices. Therefore, the probability space of the different number of random graphs that can be generated is huge and multiplies with a greater number of edges and vertices, as any vertex can be connected to an edge without prearrangement. The number of edges which connect to a vertex is known as the degree of that particular vertex.

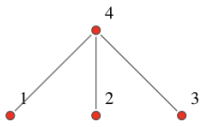
In early days any literature explaining and conjoining the theories of random graphs was often scattered and ambiguous. In the first paper published in 1960 by Erdós and Rényi, not only was this subject matter made concise and clear but random graphs were also given specific types and models like Uniform Random Graph Model where a graph G(n,m) chosen uniformly from a set of graphs with a given set of vertices [n] and and m edges or Binomial Random Graph Model, where A graph G(n,p) with a given set of vertices [n] joined by an edge independently from the subject matter with probability p. In our work, we limit the scope to graphs generated as uniform random graphs.

Before we move to the adjacency matrix, we must realize that each of these graphs can be expressed in the form of a matrix. The matrix, an arrangement of rows and columns, forms a two-dimensional square array which can represent the connectedness of the graph, known as adjacency matrix. The best way to describe the word “matrix” in mathematically accurate terms would be a set of numbers arranged in rows and columns to form an array. Matrices are an

intricate yet concise way to store data in a form that is easily accessible and understandable. Matrices find a huge number of applications in operational sciences, computer science, engineering, physics, economics, and graphics.

The form of a matrix is denoted with [aij] where i is the number of rows in a matrix and j is the number columns. The order of a matrix refers to the number of rows of the matrix multiplied by the number of columns of the matrix (i*j). Let's take a simple example:
[1 0 0]
This matrix is called a simple row matrix. As visible there is only a single row (i=1) while there are three columns (j=3) in this particular matrix. Hence, the order of the matrix (i*j) becomes 1x3.

An adjacency matrix, or simply known as a connection matrix, is a matrix pertaining to the edges connecting the vertices of a graph. It is a representation of the relation between different vertices connected to each other by edges. Each cell of an adjacency matrix represents if a vertex is connected to an adjacent vertex by an edge or not, and hence the value of that particular cell can be [0] for no connection or [1] for an edge connecting them. Since we assume that if a vertex *i* is connected to a vertex *j*, vertex *j* is also connected to vertex *i*. This makes the adjacency matrix symmetric and hence the matrix is its own transpose. For instance, this is a simple random graph (Kang, M., & Petrasek, Z. 2014) with four vertices labeled 1, 2, 3 and 4. In this specific graph, the vertex 1 has zero connections with itself, zero connections with vertex 2, zero connections with vertex 3, but one edge connecting it to vertex 4. Hence, its row with adjacency calculation becomes [0 0 0 1]. As the process is completed with every single vertex in the graph, an adjacency matrix with a set number of rows and columns is generated, in binary with all cells being occupied with either 0 or 1.



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1: Example of a graph and its Adjacency matrix. Each vertex, if connected to another vertex, gives a 1 weight to the corresponding element. Since we assume bijection, i.e. if a vertex is connected to another vertex, that implies the reverse is also true, this makes the adjacency matrix symmetric.

Random Matrix Theory

Many real-life comparisons between random matrix theory can be applied in fields of Nuclear Physics, Electrical Engineering, Data Analysis, Computational Neuroscience and Quantum Mechanics. The Wigner distribution is often seen in data analytics of all such fields such as in the modeling of the nuclei of heavy atoms, quantum optics, matrix multiplication etc. Through the material ahead, we would like to present a similar distribution through random matrices of uniform randomly generated graphs.

Random matrix theory is the study of matrices with random entries. When the eigenvalues of an ensemble of such matrices are studied, they show a specific kind of distribution and properties. The spacing in these eigenvalues when plotted, shows primarily three kinds of relations, Gaussian unitary ensemble (GUE), Gaussian orthogonal ensemble (GOE), and Gaussian symplectic ensemble (GUE). In this work, we limit our discussion to the GOE type distribution.

A matrix A_N is called Gaussian orthogonal ensemble (GOE) if

- A_N is symmetric;
- $A_N(i,i) \sim \mathcal{N}(0,2)$ for i = 1, ..., N and $A_N(i,j) \sim \mathcal{N}(0,1)$ for 1 ≤ i ≤ j ≤N;
- $(A_N(i,j))$ independent for 1 ≤ i ≤ j ≤N.

The spacing ratios are given by:

$$P(r,\beta) = C_\beta \frac{(r+r^2)^\beta}{(1+r+r^2)^{1+3\beta/2}}$$

Where β = 1 for GOE ensemble and C_β is π/2. The probability distribution looks like the following.

$$p_1(s) = \frac{\pi}{2} s e^{-\frac{\pi s^2}{4}}$$

The above equation gives the distribution function of GOE which can be compared with the eigenvalue distribution for checking the validity of the GOE for our set of eigenvalues.

Methodology

We begin by generating 1000 random Erdós Rényi graphs with 1000 vertices. This is achieved by using the networkx package in python. We then look at the adjacency matrix of each of these matrices. The spectrum or the eigenvalues of these adjacency matrices are calculated. Only the real values of these eigenvalues are considered. We then calculate the spacings of these eigenvalues by sorting these eigenvalues in ascending order and calculating the next neighbor's difference. The histogram is plotted and compared with the GOE distribution.

Random Graph Generation:

The first step for this data analysis is the generation of random graphs using a programme written in Python. It involves a pre-set random generation algorithm that utilises a set of parameters stating the number of vertices to be 1000. The programme then randomly generates different edges connecting each vertex and thus, creates a random graph. It makes sure that no combination of edges and vertices is repeated more than once.

The algorithm involves the following steps -
1. Importing networkx as nx, a free Python library released under the BSD-new license which allows the generation of multiple edges connected to multiple vertices in a graph.
2. Setting the pre-set number of vertices of each random graph to 1000 as ‘n’.
3. Setting the graph to be generated by nx as a random erdos_renyi_graph under the variable ‘g’.
4. Printing the vertices and edges of the graph to create a random graph.
5. Looping the same process until the desired number of 1000 random graphs with 1000 vertices each have been generated.

Adjacency Matrix Generation:
The next step in the process is the generation of adjacency matrices for each random graph, using an algorithm written in Python.

The algorithm involves the following steps -
1. Importing networkx as nx.
2. Importing pyplot as plt from matplotlib. Pyplot is a group of command functions from Matplotlib, a command library in Python used to generate static and dynamic figures.
3. Creating a function for the generation of adjacency matrices under which -
If the two vertices of the random graph are connected by an edge, the adjacency matrix will have the digit ‘1’ printed in that position.
Else, the adjacency matrix will have the digit ‘0’ printed in that position to symbolize no connecting edge between the vertices.
4. Adding the set of vertices and edges of the already generated random graphs under two different variables for each, which will allow the programme to generate the adjacency matrix for each graph.
5. Looping the same process until the desired adjacency matrices for all 1000 random graphs have been generated.

Eigen Value Generation:
The third step of this statistical analysis is the generation of eigenvalues from the random graphs. Each random graph is 1000 vertices and thus, will have 1000 eigenvalues. Such eigenvalues will be generated for all the random graphs and then arranged in ascending order for their statistical study and histogram generation. Another Python programme will be run for the process.

The algorithm involves the following steps -
1. Importing numpy library as np. Numpy is a Python for adding support for the generation of multi-layered arrays and matrices.
2. Creating a 2-D array in Numpy to generate the respective eigenvalues.

3. Taking values of the adjacency matrices and vertices of the graph under the Python programme in specific variable names.
4. Printing the Eigenvalues by running a Numpy command.

Combination Example:
Let's take a simple example of a generated random graph with 100 vertices, and each edge given with the notation (x,y) where x and y are the two vertices that the edge is connecting.

The next step was to find the adjacency matrix of this random graph, which was once again given by a set of binary zeros and ones, each representing the connections of a vertex with another vertex by a particular edge.

Following this adjacency matrix is the step to find the eigenvalues of this particular matrix. As the random graph generated has 100 vertices, the adjacency matrix of this graph will also have 100 eigenvalues.

Results

When a graph of the desired Eigenvalue distribution is finally produced and generated, it forms a GOE distribution (Figure 3), concurrent with the distribution seen in a wigner surmise (Figure 2). Hence, random matrices generated from randomly generated graphs also show a wigner distribution of their Eigenvalues.

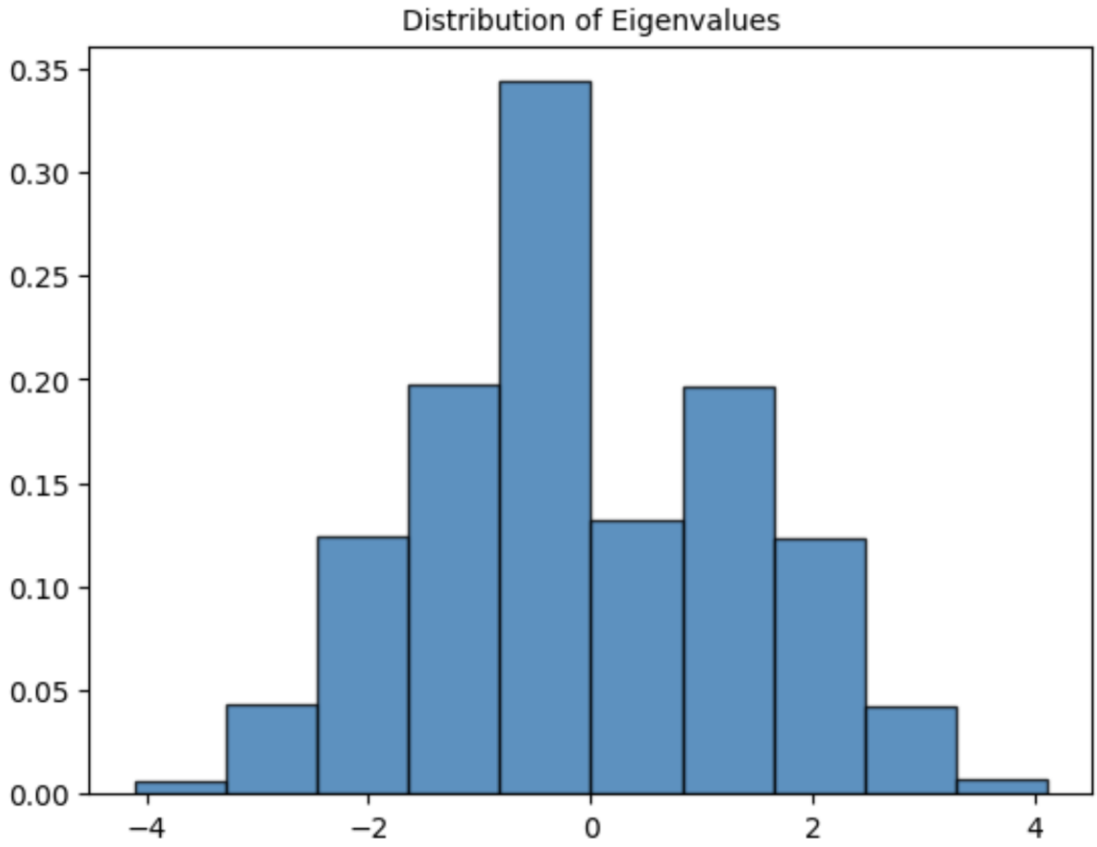


Figure 2: The plot of distribution of eigenvalues of adjacency matrices of randomly connected graphs. 1000000 eigenvalues generated from 1000 randomly connected graphs with 1000 nodes each is used.

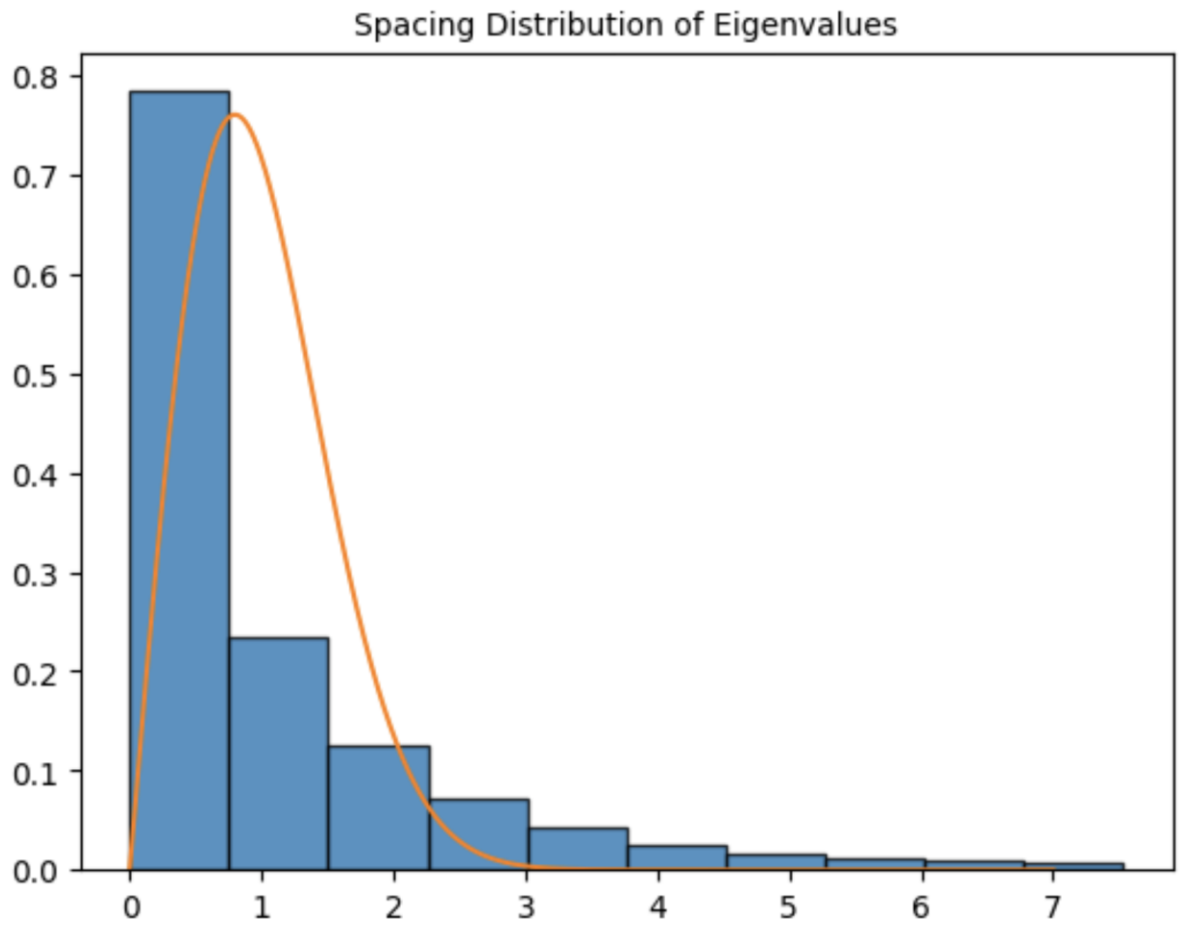


Figure 2: The yellow line represents the GOE distribution function. It can be seen to agree with the spacing distribution with the histogram.

Conclusion

In this work, we explored the relation between random matrices and randomly connected graphs. We studied the eigen-spectrum of the adjacency matrices and found that it agrees with the GOE distribution. This relation opens up a great deal of avenues of studying many body interactions using randomly connected graphs.

References

1. Bondy, J. A., & Murty, U. S. R. (1976). *Graph theory with applications* (Vol. 290). London: Macmillan.

2. West, D. B. (2001). *Introduction to graph theory* (Vol. 2). Upper Saddle River: Prentice hall.

3. Mondal, B., & De, K. (2017). An overview of applications of graph theory in the real field. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *2*(5), 751-759.

4. Bollobás, B., & Bollobás, B. (1998). *Random graphs* (pp. 215-252). Springer New York.

5. Kang, M., & Petrasek, Z. (2014). Random graphs: Theory and applications from nature to society to the brain. *Internationale Mathematische Nachrichten*, *227*, 1-24.

6. Singh, H., & Sharma, R. (2012). Role of adjacency matrix & adjacency list in graph theory. *International Journal of Computers & Technology*, *3*(1), 179-183.

7. Chung, F. R. (1995). Eigenvalues of graphs. In *Proceedings of the International Congress of Mathematicians: August 3–11, 1994 Zürich, Switzerland* (pp. 1333-1342). Birkhäuser Basel.

8. Doob, M. (2004). Eigenvalues of graphs. *Topics in algebraic graph theory*, 30-57.

9. *Adjacency Matrix -- from Wolfram MathWorld*. (n.d.). Adjacency Matrix -- From Wolfram MathWorld. https://mathworld.wolfram.com/

10. *Matrix | Definition, Types, & Facts*. (n.d.). Encyclopedia Britannica. https://www.britannica.com/science/matrix-mathematics

11. *Introduction to Adjacency Matrix For Graphs*. (2023, January 13). Codedamn News. https://codedamn.com/news/programming/introduction-to-adjacency-matrix-for-graphs

12. Milo, R., Kashtan, N., Itzkovitz, S., J. Newman, M. E., & Alon, U. (2003, December 1). *On the uniform generation of random graphs with prescribed degree sequences*. arXiv.org. https://arxiv.org/abs/cond-mat/0312028v2

13. Tao, T. (2023). *Topics in random matrix theory* (Vol. 132). American Mathematical Society.

14. Guhr, T., Müller–Groeling, A., & Weidenmüller, H. A. (1998). Random-matrix theories in quantum physics: common concepts. *Physics Reports*, *299*(4-6), 189-425.

15. Bohigas, O. (1991). *Random matrix theories and chaotic dynamics* (No. IPNO-TH--90-84). Paris-11 Univ.

16. Edelman, A., & Rao, N. R. (2005). Random matrix theory. *Acta numerica*, *14*, 233-297.

17. Sugiyama, M., Ghisu, M. E., Llinares-López, F., & Borgwardt, K. (2018). graph kernels: R and Python packages for graph comparison. *Bioinformatics*, *34*(3), 530-532.

18. Chung, J., Pedigo, B. D., Bridgeford, E. W., Varjavand, B. K., Helm, H. S., & Vogelstein, J. T. (2019). Graspy: Graph statistics in python. *Journal of Machine Learning Research*, *20*(158), 1-7.

Appendix 1: Code:

RANDOM GRAPH GENERATION CODE:

```
from networkx.generators.random_graphs import erdos_renyi_graph

n = 100
p = 0.5
g = erdos_renyi_graph(n,p)

print(g.nodes)
print(g.edges)
```

ADJACENCY MATRIX CODE:

```
import networkx as nx
from matplotlib import pyplot as plt

def createAdjacencyMatrix(vertices,edges):
  noofvertices=len(vertices)
  adjM=[]
  while(len(adjM)<noofvertices):
    temp=[]
    for i in range(noofvertices):
      temp.append(0)
    adjM.append(temp)
  for edge in edges:
    i=edge[0]
    j=edge[1]
    if i>=noofvertices or j>=noofvertices or i<0 or j<0:
      print(f"Not a Proper Input in Edge {i},{j}")
    else:
      adjM[i][j]=1
      adjM[j][i]=1
  G=nx.Graph()
  G.add_edges_from(edges)
  nx.draw_networkx(G)
  plt.show()
  print(adjM)
  return adjM

vertices= x

edges= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
```

```
51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]

# create numpy 2d-array
m = np.array()

print("Printing the Original square array:\n",m)

# finding eigenvalues and eigenvectors
w, v = np.linalg.eig(m)

# printing eigen values
print("Printing the Eigen values of the given square array:\n",w)

# printing eigen vectors
print("Printing Right eigenvectors of the given square array:\n",v
```

Code 2: Calculating the graphs for 1000 nodes and 1000 instances

```
import numpy as np
import pandas as pd
import csv
import random
from networkx.generators.random_graphs import erdos_renyi_graph
import networkx as nx
from matplotlib import pyplot as plt
from itertools import combinations
from itertools import groupby

def gnp_random_connected_graph(n, p):
    """
    Generates a random undirected graph, similarly to an Erdős-Rényi
    graph, but enforcing that the resulting graph is conneted
    """
    edges = combinations(range(n), 2)
    G = nx.Graph()
    G.add_nodes_from(range(n))
    if p <= 0:
        return G
    if p >= 1:
        return nx.complete_graph(n, create_using=G)
    for _, node_edges in groupby(edges, key=lambda x: x[0]):
        node_edges = list(node_edges)
        random_edge = random.choice(node_edges)
        G.add_edge(*random_edge)
```

```
        for e in node_edges:
            if random.random() < p:
                G.add_edge(*e)
    return G

ev=[]
nodes=100
seed = random.randint(1,10)
probability = 0.001
for i in range(1000):
    G = gnp_random_connected_graph(nodes,probability)
    ev=ev+np.real(nx.adjacency_spectrum(G, weight='weight')).tolist()

s=np.linspace(0,7,1000)
bins = np.linspace(0, 7, 10)
plt.title('Spacing Distribution of Eigenvalues',fontsize=10)
plt.hist(sp2/avg, density= True, alpha=0.8, histtype='bar', ec='black')
#plt.hist(x, bins, alpha=0.5, histtype='bar', ec='black')
plt.xticks(fontsize = 10)
plt.yticks(fontsize = 10)
plt.plot(s,(np.pi/2)*s*np.exp((-np.pi/4)*s*s))
plt.show()
```