CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Analysing the Relationship Between Spectral Properties and Network Structures: A Social Network Study looking into the relationships between different kinds of networks, with a focus on social networks in particular. First, we look at networks related to natural and mathematical numbers. The objective is to illustrate and validate the connection between a network's natural number and prime number structure. Simultaneously, we discover interesting patterns in the Laplacian spectrum of networks based on arithmetic. Numerous disciplines, such as number theory and discrete mathematics, depend on SGT. SGT is the study of a graph's properties using the eigenvalues and eigenvectors of a matrix that is connected to the graph. Usually, this matrix is an adjacency or Laplacian matrix.

In this thesis, we mainly study the Laplacian matrix. We have studied two kinds of graphs: networks with natural numbers and arithmetic numbers. A natural number network is a graph with vertices 1, 2, · · ·, n, and an adjacency specified by a divisibility relation. An arithmetic network is a graph with vertices labelled according to arithmetic progression and adjacency determined by the divisibility relation.

1.2 MOTIVATION

This study is driven by the goal of understanding the fundamental dynamics and structures that underpin a variety of networks, from complex social interactions to natural numerical correlations. The realisation that networks are ubiquitous in many disciplines and that knowledge of their characteristics can provide insights with broad ramifications is what motivates the research.

Facebook buddies in the virtual world

721 million individuals 69 billion connections of friendship

99.6% of all user pairs with pathways longer than five degrees (six hops)

92% of them are only four degrees (5 hops) apart.

3.74 intermediates/degrees (4.74 hops) on average.

The underlying structures of social networks can be fascinatingly understood through the lens of SGT. The following are some reasons to start a project in this field:

Understanding of Network Dynamics: One can learn about the dynamics of information flow, influence propagation, and community building inside social networks by examining the

spectral characteristics of the graphs that describe these networks. Comprehending the ways in which these interactions transpire can have consequences for multiple disciplines, including sociology, epidemiology, and marketing.

Discovery of Communities: Within social networks, SGT provides strong instruments for community discovery. Finding communities can provide light on hidden connections and structures inside the network, which can help develop marketing plans, targeted intervention methods, and a knowledge of social dynamics.

Information dissemination: The study of information dissemination and influence propagation within social networks is aided by the application of SGT. We can forecast how information travels via a network by simulating its adjacency matrix and eigenvalues/eigenvectors. This allows us to develop more effective plans for viral marketing, rumour management, and epidemic containment.

Node Ranking and Centrality: In social networks, spectral techniques can be utilised to calculate node centrality metrics like PageRank or eigenvector centrality. Finding important opinion leaders, influencers, or central players in a network requires an understanding of node importance and influence.

1.3 OBJECTIVE

Bridging Mathematical Concepts with Practical Applications: A principal objective is to bridge hypothetical numerical concepts, such as number hypothesis and straight variable based math, with commonsense applications in social organize investigation. The extend points to exhibit the pertinence of hypothetical experiences in real-world scenarios.

Illustrating Power-Law Degree Distribution: A viable objective includes actualizing a Python codebase to produce and visualize systems based on normal numbers, number-crunching, and social intuitive. The venture points to illustrate that both common number systems and math systems show PLD conveyances, supporting the claim that they are scale-free systems.

Modelling Social Networks: Amplifying the examination to social systems, the extend points to demonstrate client intelligent and visualize the coming about organize. By altering parameters such as the power-law type and interaction probabilities, the objective is to grandstand how these parameters affect the structure of a social arrange.

1.4 CHALLENGES AND LIMITATIONS

Information Quality and Accessibility: Social arrange information may endure from clamor, lost values, or mistakes, which can influence the unwavering quality of investigation comes about. Moreover, get to to real-world social organize information, particularly from large-scale stages, may be confined due to security concerns or restrictive confinements.

Scalability: Scalability becomes a crucial issue as social networks get bigger and more sophisticated. Large-scale networks may be difficult for current systems to manage well, which could affect how well algorithms and analysis operate.

Network Sparsity: A large number of social networks show a high level of sparsity, which means that users are frequently connected to a relatively tiny portion of the total user base. Sparse networks are difficult to analyse and forecast, especially for recommendation systems.

Protection and Moral Contemplations: Analysing social organize information raises security and moral concerns, especially with respect to information collection, utilization, and potential abuse. Guaranteeing compliance with protection directions and receiving moral information hones is significant to anticipate unintended results and protect client protection.

Generalization Over Systems: Unearthly chart hypothesis procedures created for one sort of social arrange may not generalize well to other organize sorts or spaces. Tending to this impediment requires creating versatile calculations that can capture the differing basic characteristics of distinctive social systems.

Heterogeneity and Data Quality: The quality and heterogeneity of data is one of the main obstacles. Multimedia, text, and image data are among the many types of data that social networks produce in massive quantities. It is quite difficult to guarantee the trustworthiness, accuracy, and completeness of this data.

CHAPTER 2
LITERATURE SURVEY

1. **Graph Theory and Algorithms for Network Analysis, Sharmila Mary Arul1 GowriSenthil2 Dr.S.Jayasudha 3, Ahmed Alkhayyat4 Khalikov,Azam5R.Elangovan6**

   In arrange investigation, the consider and comprehension of complex frameworks invarious areas, such as social systems, transportation systems, and natural systems, are made conceivable by the significant part played by chart hypothesis and calculations. In arrange to allow a comprehensive audit of the chart hypothesis and organize examination strategies, this unique will center on their importance, viableemployments, and most later advancements. With things spoken to as hubs or vertices and joins between them as edges, chart hypothesis offers a numerical system for modeling and assessing connections between objects. Analysts may learn vital thingsapproximately the structure, network, and behavior of complex frameworks by utilizing chart hypothesis in organize examination. As a result, arrange examination is made conceivable by the chart hypothesis and calculations, which offer solid instruments for examining and comprehending the complicated linkages and structures of complex frameworks. Chart hypothesis and calculations have numerous diverse applications,counting social systems, transportation systems, and natural systems. Large-scaleorganize investigation is presently conceivable much obliged to the advancement ofviable calculations and strategies, which has altogether progressed the subject. Theimportance of chart hypothesis and calculations for arrange investigate will as it were rise as systems proceed to extend in measure and complexity.

2. **Influence maximization in social networks: a survey of behaviour-aware methods:Ahmad Zareie · Rizos Sakellariou**

   Social networks are becoming a more often used abstraction to record user interactionsin a variety of daily tasks and applications. Consequently, there has been a great deal of interest in the literature on the analysis of such networks. Finding so-called prominent people for various applications that need to propagate messages is one of the main issues with the themes of interest. Numerous methods have been put forth to determineuser influence and pinpoint groups of influential individuals inside social networks. These methods all share the consideration of user links, or the topological or structural characteristics of the network. Some methods consider the attitudes and behaviours of users, but to a lesser degree. While other surveys have examined methods based on structural characteristics of social networks, a thorough evaluation of methods that consider user behaviour has not yet been conducted. This study reviews the literature and offers a taxonomy of such behaviour in an effort to close this gap.thoughtful techniques for locating significant social network users

3. **Social Network Analysis: From Graph Theory to Applications with Python :DMITRI GOLDENBERG, Booking.com, Tel Aviv**

The study of social structures using networks and graph theory is known as social network analysis. It integrates several methods of social network structure analysis with theoretical frameworks intended to elucidate the underlying dynamics and patterns seen in these systems. It is an interdisciplinary field by nature, having originated in the domains of graph theory, statistics, and social psychology. Together with a brief overview of graph theory and information dissemination, this session will address the theory of social network analysis. After that, in order to gain a deeper comprehension of the network's components, we will go further into Python code using NetworkX. This will be followed by the construction and inference of social networks using real Pandas and textual datasets. Lastly, we shall discuss coding instances of real-world applications include influence maximisation for information dissemination, social- centrality analysis, and Matplotlib visualisation.

4. **A.-L. Barabási and R. Albert. Emergence of scaling in random networks. Science,286:509-512, 1999.**

In this paper we can understand that the only way to characterise systems as different as the internet or genetic networks is as networks with complicated topology. The vertices connectivities of many vast networks have a scale-free PLD as a common characteristic. It is discovered that these two general mechanisms—new vertices attaching preferentially to already-well-connected locations and networksexpanding continually through the addition of additional vertices—are responsible for this property. These two components together provide a model that replicates the observed stationary scale-free distributions, suggesting that strong self-organizing forces that transcend the specifics of individual systems regulate the growth of vast networks.

5. **Explaining the Power-Law Degree Distribution in a Social CommerceCommunity: Andrew T Stephen, Olivier Toubia**

An emerging trend in e-commerce is social commerce, whereby online retailers build hyperlinks to other retailers within their marketplace. In this paper they examine how asocial commerce network has developed within a sizable online marketplace. Our collection includes information from before the network was created, when stores werenot connected to one another. The network that is being studied has a PLD distribution. They conduct an empirical comparison of many edge formation methods that could account for the emergence of this characteristic, such as preferential attachment and triadic closure. Their findings imply that a network development mechanism that depends on vertex attributes rather than the structure of the network isa superior way to explain the evolution of the network and the creation of its PLD distribution.

CHAPTER 3

ARCHITECTURE AND ANALYSIS

3.1 PROPOSED MODULE

What is Social Network ?

A graph metaphor that is more akin to a semantic web for analyzing the connections and interactions within a group of individuals

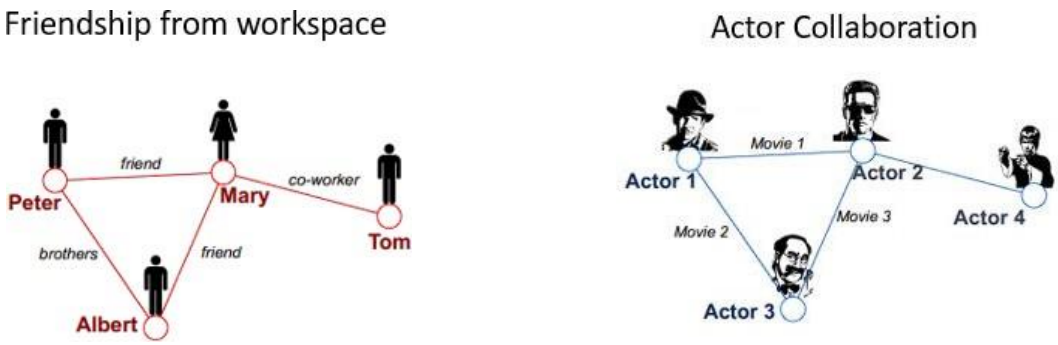


Fig1.1 Social Network Examples

3.2 WHAT IS SPECTRAL GRAPH THEORY

The Main Concept: The study of graphs (networks) through the analysis of eigenvalues and eigenvectors of matrices associated with those graphs is known as SGT in mathematics.

Special mathematical values and vectors connected to the graph's matrix representation are called eigenvalues and eigenvectors.

A graph's matrix's eigenvalues and eigenvectors provide a plethora of information, such as: Connectivity: The degree of the graph's connectivity and the presence of information flow bottlenecks.

Communities: Unknown groupings or groups inside the graph.

Node Importance: Identifying the key players in the network (imagine social network influencers).

Graph Similarity: Graphs are compared to identify structural similarities.

3.3 POWER LAW DISTRIBUTION

In social networks, the distribution of connections or interactions between people is directly related to the existence of a PLD. The following explains briefly why PLD are relevant in social networks:

Consider yourself examining a social network user's friend count. A small number of persons, known as "super connectors" or "hubs," have a disproportionately high number of friends in a PLF. These hubs are extremely connected and have a large sphere of effect, which makes them vital to the network. Consider how information

or trends propagate within this network now. If a small group of people embrace a new trend or exchange knowledge, it can swiftly spread to a huge audience since they have so many connections. This is comparable to how a social media trend or viral article could go viral; it usually begins with a few key individuals.

PLD, then, provide an explanation for why certain users in social networks have a large number of friends or followers while the majority have fewer. It illuminates the dynamics of the network as a whole, the people with influence, and the way information spreads. This knowledge is useful for a number of purposes, such as developing marketing plans, identifying social trends, and projecting the effects of network modifications.

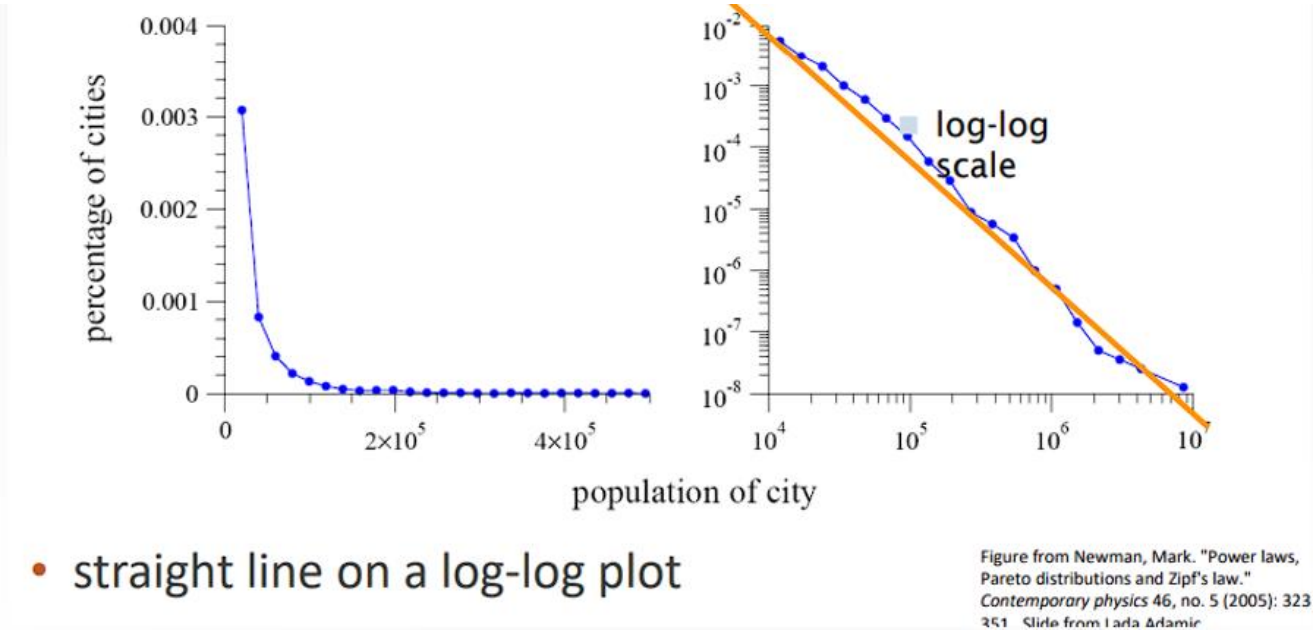


Fig3.1 Power Law Distribution Graph

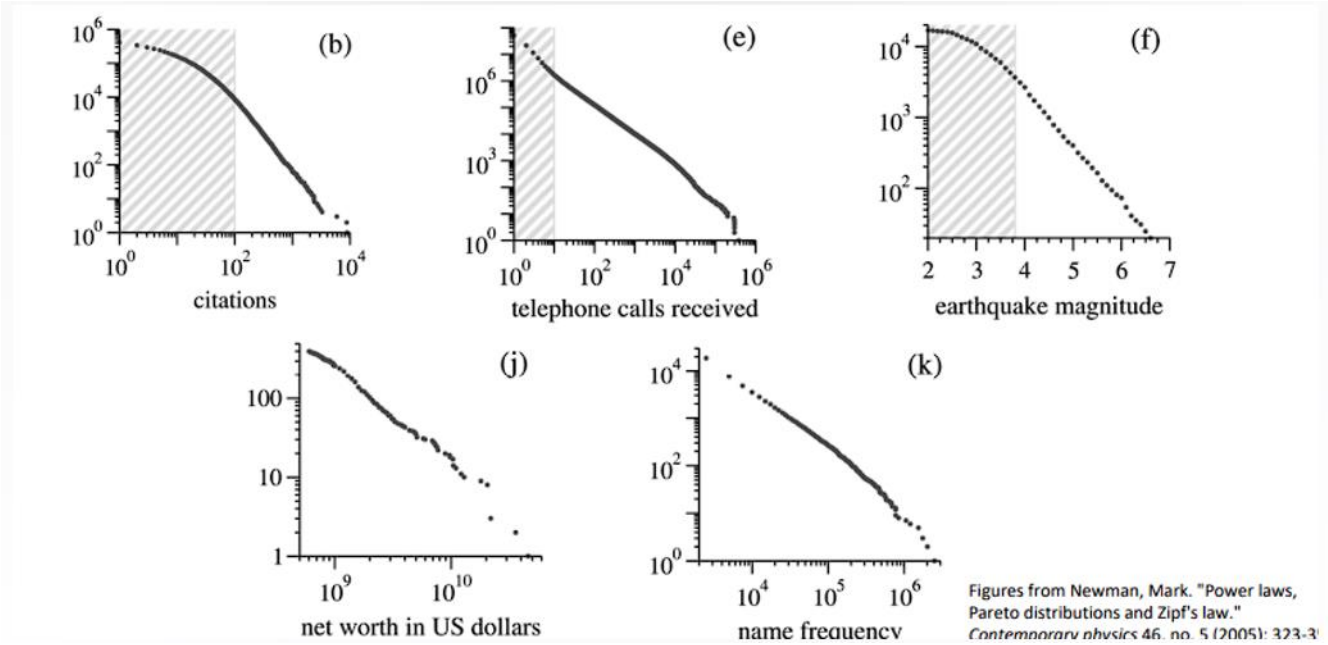


Fig3.2 Power Law use in day to day

CHAPTER 4

IMPLEMENTATION

4.1 MAIN CODE

I Power Law Distribution

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx

def generate_power_law_network(num_nodes, alpha, seed=None):
    np.random.seed(seed)

    # Generate power-law degree distribution
    degrees = np.arange(1, num_nodes + 1)
    degree_probs = 1 / np.power(degrees, alpha)
    degree_probs /= degree_probs.sum()

    # Generate degrees for each node
    node_degrees = np.random.choice(degrees, size=num_nodes, p=degree_probs)

    # Plot degree distribution
    plt.figure(figsize=(8, 6))
    plt.title("Power-law Degree Distribution")
    plt.plot(degrees, degree_probs, 'o', color='b', markersize=8)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Degree (log scale)')
    plt.ylabel('Probability (log scale)')
```

```
    plt.show()

    return node_degrees

def generate_and_visualize_network(num_nodes, m, alpha, seed=None):
    # Generate artificial power-law degree distribution
    degrees = generate_power_law_network(num_nodes, alpha, seed)

    # Create a graph (Preferential attachment would be ideal, this is a workaround)
    G = nx.complete_graph(num_nodes)  # For now, make a complete graph

    # Assign approximate power-law degrees (ideally would change structure, not just degrees)
    new_degrees = dict(zip(G.nodes(), degrees))
    nx.set_node_attributes(G, new_degrees, 'degree')
```

II Social Network Function

```
    import matplotlib.pyplot as plt
    import numpy as np
    import networkx as nx

    def generate_power_law_network(num_nodes, alpha, seed=None):
        np.random.seed(seed)

        # Generate power-law degree distribution
        degrees = np.arange(1, num_nodes + 1)
        degree_probs = 1 / np.power(degrees, alpha)
        degree_probs /= degree_probs.sum()
```

```
    # Generate degrees for each node
    node_degrees = np.random.choice(degrees, size=num_nodes, p=degree_probs)

    # Plot degree distribution
    plt.figure(figsize=(8, 6))
    plt.title("Power-law Degree Distribution")
    plt.plot(degrees, degree_probs, 'o', color='b', markersize=8)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Degree (log scale)')
    plt.ylabel('Probability (log scale)')
    plt.show()

    return node_degrees

def generate_and_visualize_network(num_nodes, m, alpha, seed=None):
    # Generate artificial power-law degree distribution
    degrees = generate_power_law_network(num_nodes, alpha, seed)

    # Create a graph (Preferential attachment would be ideal, this is a workaround)
    G = nx.complete_graph(num_nodes)  # For now, make a complete graph

    # Assign approximate power-law degrees (ideally would change structure, not just
degrees)
    new_degrees = dict(zip(G.nodes(), degrees))
    nx.set_node_attributes(G, new_degrees, 'degree')

    # Degree Distribution Analysis
    plot_degree_distribution(G)
```

```
    # Network Visualization
    layout = nx.spring_layout(G)  # Or explore other layouts
    plt.figure(figsize=(8, 8))
    plt.title("Social Network Analysis (Artificially Adjusted Degrees)")
    nx.draw(G, pos=layout, with_labels=True, node_size=400)
    plt.show()

def plot_degree_distribution(G):

    degrees = [d for n, d in G.degree()]  # Extract degrees
    plt.hist(degrees)
    plt.xlabel("Degree (k)")
    plt.ylabel("Number of Nodes")
    plt.title("Degree Distribution")
    plt.loglog()
    plt.legend(['Degree Distribution'])
    plt.show()

# Example usage
num_nodes = 20
alpha = 2.5
m = 2  # Parameter for preferential attachment (not fully used here)
seed = 42

generate_and_visualize_network(num_nodes, m, alpha, seed)
```

III Interconnected Social Network with Node

```
import matplotlib.pyplot as plt
import numpy as np

def generate_social_network(num_nodes, seed=None):
    np.random.seed(seed)

    # Generate a fully connected network
    adjacency_matrix = np.ones((num_nodes, num_nodes)) - np.eye(num_nodes)

    return adjacency_matrix

def plot_social_network(adjacency_matrix, node_labels):
    num_nodes = adjacency_matrix.shape[0]

    plt.figure(figsize=(8, 8))
    plt.title("Social Network Analysis")

    # Adjust layout for better visualization
    layout = np.random.rand(num_nodes, 2) * 2 - 1

    # Plot nodes
    for i in range(num_nodes):
        plt.scatter(layout[i, 0], layout[i, 1], s=100, alpha=0.7, label=f"User
{node_labels[i]}")

    # Connect nodes with edges that touch the nodes
    for i in range(num_nodes):
```

```
        for j in range(i + 1, num_nodes):
            if adjacency_matrix[i, j] == 1:
                # Calculate edge midpoint and offset vectors towards nodes
                midpoint = (layout[i] + layout[j]) / 2
                offset_i = (layout[i] - midpoint) * 0.1
                offset_j = (layout[j] - midpoint) * 0.1

                # Plot edge with slight adjustments to touch nodes
                plt.plot([layout[i, 0], midpoint[0] + offset_j[0]],
                         [layout[i, 1], midpoint[1] + offset_j[1]],
                         'k-', alpha=0.3)
                plt.plot([layout[j, 0], midpoint[0] - offset_i[0]],
                         [layout[j, 1], midpoint[1] - offset_i[1]],
                         'k-', alpha=0.3)

    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.show()


num_nodes = 10
seed = 42

# Generate a fully connected social network
social_network = generate_social_network(num_nodes, seed)

# Plot the social network with edges touching nodes
plot_social_network(social_network, range(1, num_nodes + 1))
```

IV Barabási-Albert function

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

# Preferential attachment network generation
num_nodes = 20
m = 5  # Number of edges to attach from new nodes each step
G = nx.barabasi_albert_graph(num_nodes, m)

# Define the degree distribution plotting function
def plot_degree_distribution(G):

    degrees = [G.degree(n) for n in G.nodes()]



# Analyze the degree distribution
plot_degree_distribution(G)
def plot_degree_distribution(G):
    degrees = [G.degree(n) for n in G.nodes()]
    plt.hist(degrees)
    plt.xlabel("Degree (k)")
    plt.ylabel("Number of Nodes")
    plt.title("Degree Distribution")
    plt.loglog()
    plt.legend(['Degree Distribution'])

# Visualize the network
plt.figure(figsize=(8,6))
```

```
degrees = [G.degree(n) for n in G.nodes()]
plt.hist(degrees)
plt.xlabel("Degree (k)")
plt.ylabel("Number of Nodes")
plt.title("Degree Distribution")
plt.loglog()  # Plot on a log-log scale
plt.legend(['Degree Distribution'])  # Add the legend here

plt.figure(figsize=(8, 8))
plt.title("Social Network Analysis")
nx.draw(G, with_labels=True, node_size=400)
proxy_artist = plt.Line2D((0,1),(0,0), color='k', linestyle='-')
plt.legend([proxy_artist], ['Edge'])

plt.show()
```

4.2 MAIN CODE EXPLANATION

I Power Law Distribution Functionality

1. **Imports**
   - *import matplotlib.pyplot as plt*
   - *import numpy as np*

matplotlib.pyplot: This is the primary plotting library in Python. We're importing it with the common alias plt.

numpy: NumPy provides powerful tools for numerical computation and working with arrays. We'll use it for random number generation and calculations.

2. ***generate_power_law_network() function***

```
def generate_power_law_network(num_nodes, alpha, seed=None):
    np.random.seed(seed)
```

Purpose: Creates a network where the distribution of node degrees follows a power law. Power-law networks are common in many real-world scenarios (social networks, internet structure, etc.)

Parameters:

- num_nodes: The desired number of nodes in the network.
- alpha: The exponent of the power-law distribution. This controls the shape of the distribution.
- seed: An optional seed to initialize the random number generator. This ensures reproducibility of results.

 # Generate power-law degree distribution

degrees = np.arange(1, num_nodes + 1)

degree_probs = 1 / np.power(degrees, alpha)

degree_probs /= degree_probs.sum()

3. **Generating the distribution:**
   - degrees = np.arange(1, num_nodes + 1): Creates an array of possible degrees for the nodes, starting from 1 up to the total number of nodes.
   - degree_probs = 1 / np.power(degrees, alpha): Calculates the probability of each degree using the power-law formula (1 / degree^alpha).
     - *degree_probs /= degree_probs.sum(): Normalizes the probabilities so that they sum to 1. This is essential for a valid probability distribution.*

 # Generate degrees for each node

node_degrees = np.random.choice(degrees, size=num_nodes, p=degree_probs)

4. **Sampling degrees:**
   np.random.choice(degrees, size=num_nodes, p=degree_probs): This line randomly assigns degrees to each node in the network. The degrees array provides the pool of possible degree values, and the degree_probs array controls the likelihood of each degree being chosen.

- *plt.figure(figsize=(10, 6))*
- *plt.title("Power-law Degree Distribution")*
- *plt.plot(degrees, degree_probs, 'o', color='b', markersize=8)*
- *plt.xscale('log')*
- *plt.yscale('log')*
- *plt.xlabel('Degree (log scale)')*
- *plt.ylabel('Probability (log scale)')*
- *plt.show()*

Plotting: This block visualizes the generated power-law degree distribution:

- *plt.figure(figsize=(10, 6)): Sets up a new figure window with specified dimensions.*
- *plt.title("Power-law Degree Distribution"): Adds a title.*
- *plt.plot(degrees, degree_probs, 'o', color='b', markersize=8): Plots the degrees (x-axis) against their corresponding probabilities (y-axis), using blue circles as markers.*
- *plt.xscale('log'), plt.yscale('log'): Configures both axes to use a logarithmic scale, which is typical for visualizing power-law distributions.*
- *plt.xlabel('Degree (log scale)'), plt.ylabel('Probability (log scale)'): Adds labels.*
- *plt.show(): Displays the plot.*

return node_degrees

Returns: The function returns an array node_degrees containing the assigned degree for each node in the network.

II Social Network Analysis Functionality using networkx

1. Imports

- *import matplotlib.pyplot as plt*
- *import numpy as np*

Same imports as before: Matplotlib for plotting and NumPy for arrays and random number generation.

Functions

generate_social_network(num_nodes, seed=None)

Purpose: Generates a basic representation of a fully connected social network as an adjacency matrix.

Parameters:

- *num_nodes: Number of individuals (nodes) in the network.*
- *seed: Optional seed for the random number generator.*

Logic:

- *np.random.seed(seed): Sets the random seed for reproducibility.*
- *adjacency_matrix = np.ones((num_nodes, num_nodes)) - np.eye(num_nodes):*
- *np.ones((num_nodes, num_nodes)): Creates a square matrix filled with ones, indicating potential connections between all nodes.*
- *np.eye(num_nodes): Creates an identity matrix (diagonal of ones), which represents "self-connections."*

Subtracting the identity matrix removes potential self-connections, ensuring each node is only connected to others.

Returns: The adjacency_matrix. In this model, it's a matrix where a value of '1' at position (i, j) indicates a connection between node 'i' and node 'j'

plot_social_network(adjacency_matrix, node_labels)

Purpose: Creates a visual representation of the social network.

Parameters:

- *adjacency_matrix: The adjacency matrix created by generate_social_network.*
- *node_labels: A list of labels to identify each node.*

Logic:

- *num_nodes = adjacency_matrix.shape[0]: Get the number of nodes from the matrix.*
- *plt.figure(figsize=(8, 8)), plt.title("Social Network Analysis"): Set up the plot.*
- *layout = np.random.rand(num_nodes, 2) * 2 - 1: Randomly distribute node positions in a 2D plane.*

Plotting nodes:

The code iterates through each node (user), plots it as a scatter point using its layout coordinates, and adds a label.

Plotting edges:

Double loop iterates through all pairs of nodes.

if adjacency_matrix[i, j] == 1: Checks if nodes 'i' and 'j' are connected.

Lines following this calculate midpoints and offsets to draw visually nicer edges that "touch" the nodes instead of simply connecting their centers.

plt.xlabel("X-axis"), plt.ylabel("Y-axis"), plt.legend(), plt.show(): Add labels, show the plot.

The last part of the code demonstrates how to use the functions:

num_nodes = 20

seed = 42

 # Generate a fully connected social network

social_network = generate_social_network(num_nodes, seed)

 # Plot the social network with edges touching nodes

plot_social_network(social_network, range(1, num_nodes + 1))

III Interconnected Social Network using matplotlib

generate_social_network function:

This function takes two parameters: num_nodes, which represents the number of nodes (users) in the social network, and seed, which is an optional parameter used to seed the random number generator for reproducibility.

It generates a fully connected network represented by an adjacency matrix. In a fully connected network, each node is connected to every other node except itself.

The adjacency matrix is initialized with ones everywhere except on the diagonal, where it's zero (since a node is not connected to itself).

The function returns the adjacency matrix representing the social network.

plot_social_network function:

This function takes two parameters: adjacency_matrix, which represents the connections between nodes, and node_labels, which is a list of labels for each node.

It initializes a plot to visualize the social network.

Random layout positions are generated for each node for better visualization.

Nodes are plotted as scatter points on the graph with labels indicating user numbers.

Edges are drawn between connected nodes. Edges are slightly adjusted to touch the nodes' scatter points for better visualization.

Finally, axes labels, title, and legend are added to the plot.

The plot is displayed using plt.show().

Main code:

num_nodes and seed are defined, setting the number of nodes in the network and the random seed for reproducibility.

generate_social_network function is called to create the adjacency matrix representing the social network.

plot_social_network function is called to visualize the social network.

generate_social_network function:

This function creates a social network represented as an adjacency matrix.

Parameters:

num_nodes: The number of nodes (users) in the social network.

seed (optional): The seed for the random number generator, used for reproducibility.

Steps:

The function starts by seeding the random number generator using np.random.seed(seed).

It initializes an adjacency matrix of size (num_nodes, num_nodes) filled with ones.

Then, it sets the diagonal elements of the matrix to zero, as a node is not connected to itself in the network.

Finally, it returns the adjacency matrix.

plot_social_network function:

This function visualizes the social network.

Parameters:

adjacency_matrix: The adjacency matrix representing connections between nodes.

node_labels: Labels for each node in the network.

Steps:

It sets up the plot with a title using plt.title.

Random layout positions are generated for each node using np.random.rand(num_nodes, 2) * 2 - 1. This gives each node a random position within a square centered at the origin.

Nodes are plotted as scatter points on the graph using plt.scatter. Each node is labeled with its corresponding user number.

Edges are drawn between connected nodes. For each pair of connected nodes, the midpoint between them is calculated, and slight adjustments are made to ensure the edges touch the nodes' scatter points. This is done to improve the visualization.

Axes labels (X-axis and Y-axis), and a legend are added to the plot.

The plot is displayed using plt.show().

Main code:

num_nodes and seed are defined. You can adjust num_nodes to change the size of the social network, and seed for reproducibility.

generate_social_network is called to create the adjacency matrix representing the social network.

plot_social_network is called to visualize the social network.

IV Barabási-Albert function

1. **Network Generation:**
   num_nodes = 20
   m = 5
   G = nx.barabasi_albert_graph(num_nodes, m)

   - **num_nodes**: Specifies the number of nodes in the network.
   - **m**: Determines how many edges to attach from a new node to existing nodes.
   - **nx.barabasi_albert_graph()** generates a Barabási-Albert preferential attachment network with the specified parameters.
   - **G** represents the generated network.

2. **Degree Distribution Plot Function:**
   def plot_degree_distribution(G):
   degrees = [G.degree(n) for n in G.nodes()]
   plt.hist(degrees)
   plt.xlabel("Degree (k)")
   plt.ylabel("Number of Nodes")
   plt.title("Degree Distribution")
   plt.loglog()
   plt.legend(['Degree Distribution'])

   - This function, **plot_degree_distribution()**, calculates and plots the degree distribution of a given network **G**.
   - It computes the degree of each node in the network and stores them in the degrees list.
   - A histogram of the degree distribution is plotted using **plt.hist()**.
   - The x-axis is labeled as "Degree (k)", and the y-axis is labeled as "Number of Nodes".
   - The plot is displayed on a log-log scale using **plt.loglog()** to better visualize the power-law distribution.
   - A legend is added to the plot.

3. **Analyzing the Degree Distribution:**
   plot_degree_distribution(G)
   - This line calls the plot_degree_distribution() function to analyze and plot the degree distribution of the generated network G.

4. **Network Visualization:**

```
plt.figure(figsize=(8, 8))
plt.title("Social Network Analysis")
nx.draw(G, with_labels=True, node_size=400)
proxy_artist = plt.Line2D((0,1),(0,0), color='k', linestyle='-')
plt.legend([proxy_artist], ['Edge'])
plt.show()
```

- This code visualizes the generated network using NetworkX's nx.draw() function.
- Nodes are drawn with labels (with_labels=True) and a specified node size of 400.
- Edges are represented as lines connecting nodes.
- A legend is added to indicate the representation of edges.
- The resulting visualization is displayed using plt.show().

CHAPTER 5

RESULTS AND DISCUSSION

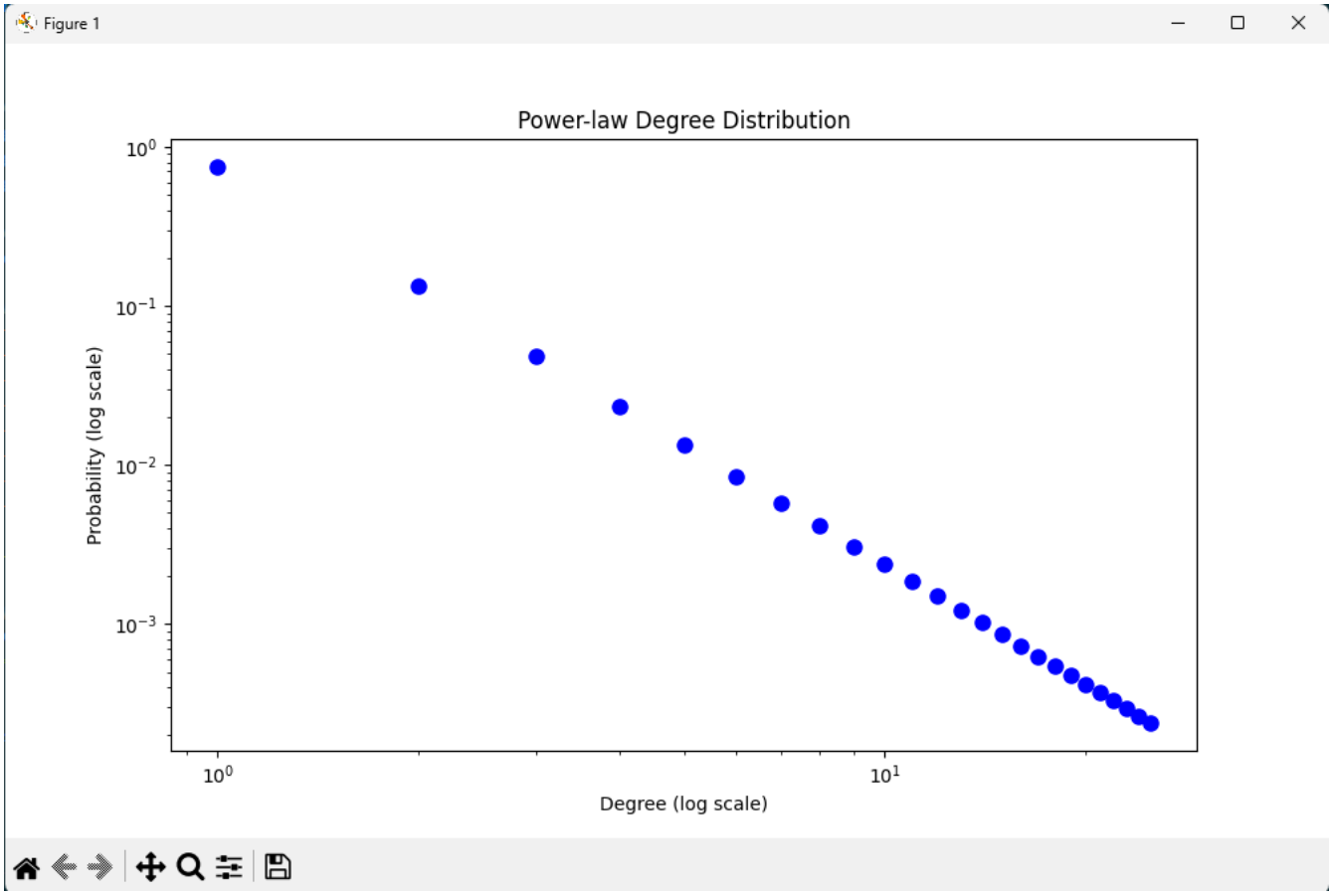


Fig5.1 Power Law Distribution Curve (From the Code)

X-axis (Degree - log scale):

- The x-axis represents the degree of nodes in the network, plotted on a logarithmic scale.
- Degree values will range from 1 to num_nodes (in this case, 20), with each integer value represented.
- Smaller degree values will be located towards the left side of the plot, while larger degree values will be towards the right side.
- The x-axis will be labeled as "Degree (log scale)", indicating that the values are plotted on a logarithmic scale.

Y-axis (Probability - log scale):

- The y-axis represents the probability of each degree value, also plotted on a logarithmic scale.
- Probabilities will range from very small values to 1, indicating the likelihood of observing each degree value in the network.
- As with the x-axis, smaller probabilities will be towards the bottom of the plot, while larger probabilities will be towards the top.
- The y-axis will be labeled as "Probability (log scale)", indicating that the values are plotted on a logarithmic scale.

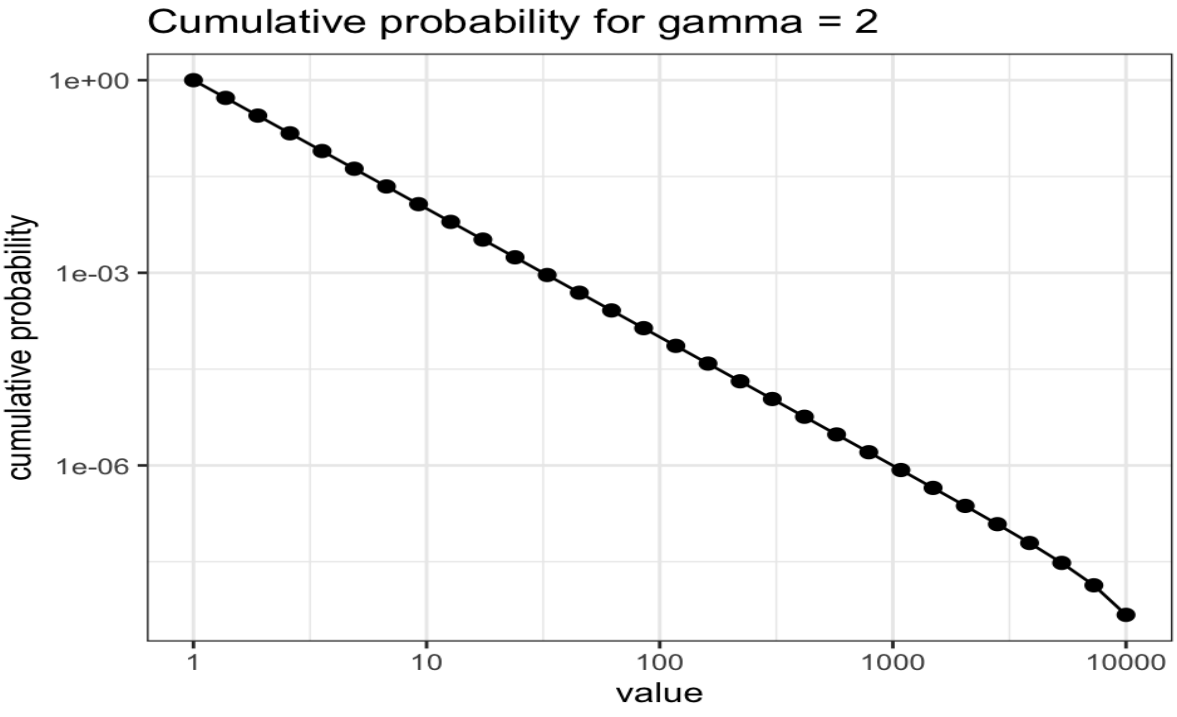
Degree Distribution Curve:

- The plot will consist of points forming a curve that represents the degree distribution of the power-law network.
- The curve will exhibit a characteristic shape of a PLD:
- Initially, the curve may start at a relatively high probability for small degree values.
- As the degree values increase, the curve will decrease rapidly, indicating a heavy-tailed distribution.
- There will be a few degree values with higher probabilities compared to the majority of degree values, representing nodes with very high degrees (hubs).

Interpretation:

- The plot visually demonstrates the heavy-tailed nature of the degree distribution in the power-law network.
- It illustrates that there are relatively few nodes with very high degrees (hubs), while the majority of nodes have lower degrees.
- The logarithmic scales on both axes allow for better visualization of the entire range of degree values and probabilities, especially when dealing with distributions that span several orders of magnitude.
- Overall, the plot provides insights into the structure of the power-law network, highlighting the presence of highly connected nodes and the degree heterogeneity among nodes

Fig5.2 Power Law Distribution Curve (Estimated)



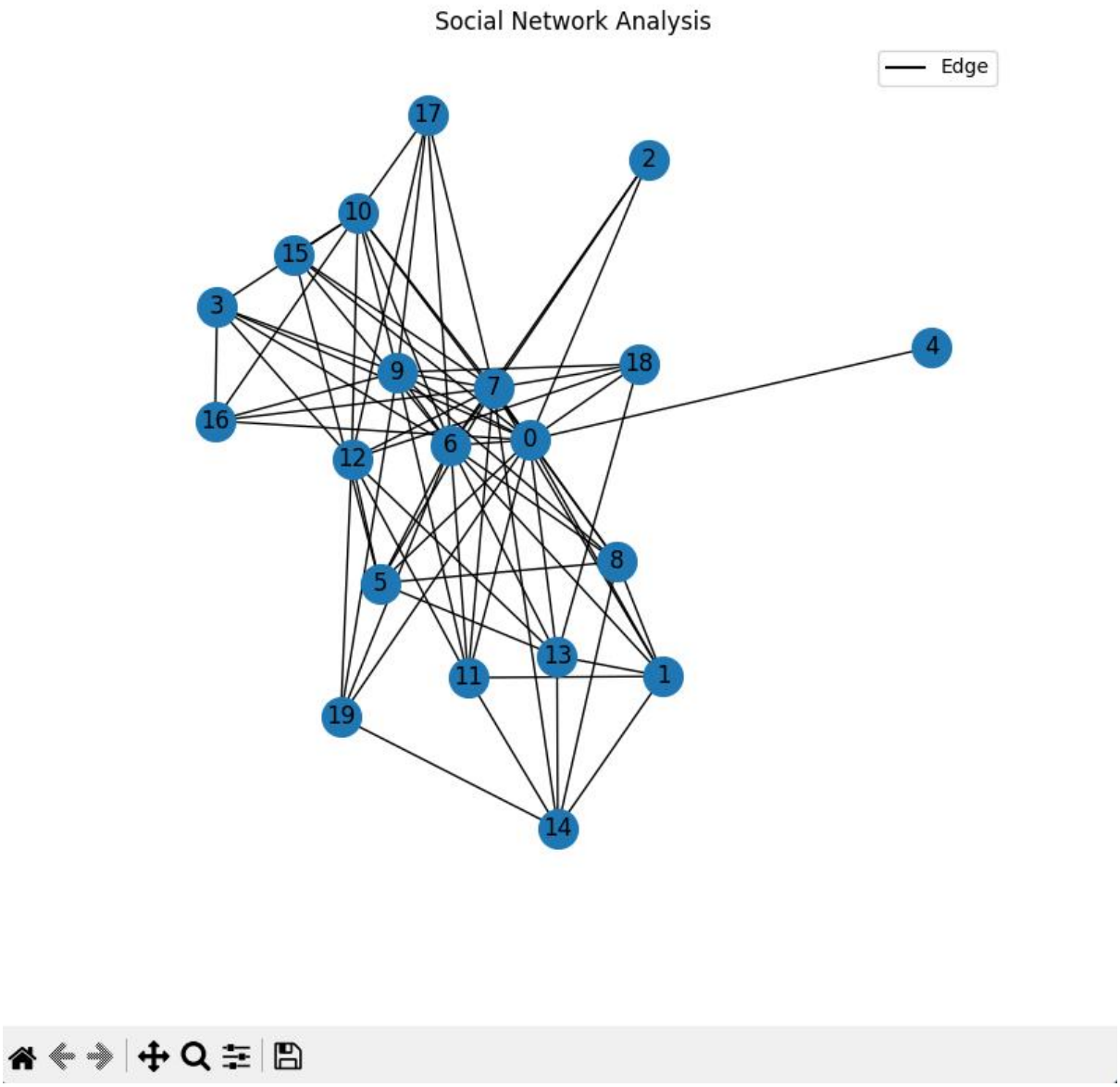


Fig5.3 Social Network using network (Artificial Generated Nodes)

Network Generation:

- The BA preferential attachment model generates networks with a PLD, where newer nodes preferentially attach to existing nodes with higher degrees, resulting in a scale-free network.
- In this case, the network has num_nodes = 20 nodes and m = 5 edges are attached from new nodes in each step.
- **Network Visualization:**
- The network is visualized using the nx.draw() function from NetworkX.
- Nodes are represented as circles, labeled with their IDs.
- Node size is set to 400 for better visibility.

- Edges are represented as lines connecting nodes.
- The legend indicates the representation of edges with a black line.

Overall Interpretation:

- The degree distribution plot shows that the network follows a PLD, indicating the presence of highly connected nodes (hubs) and a long tail of nodes with lower degrees.
- The network visualization illustrates the structure of the generated network, highlighting the connections between nodes and the presence of highly connected nodes.
- Together, these visualizations provide insights into the topology and connectivity patterns of the BA preferential attachment network.

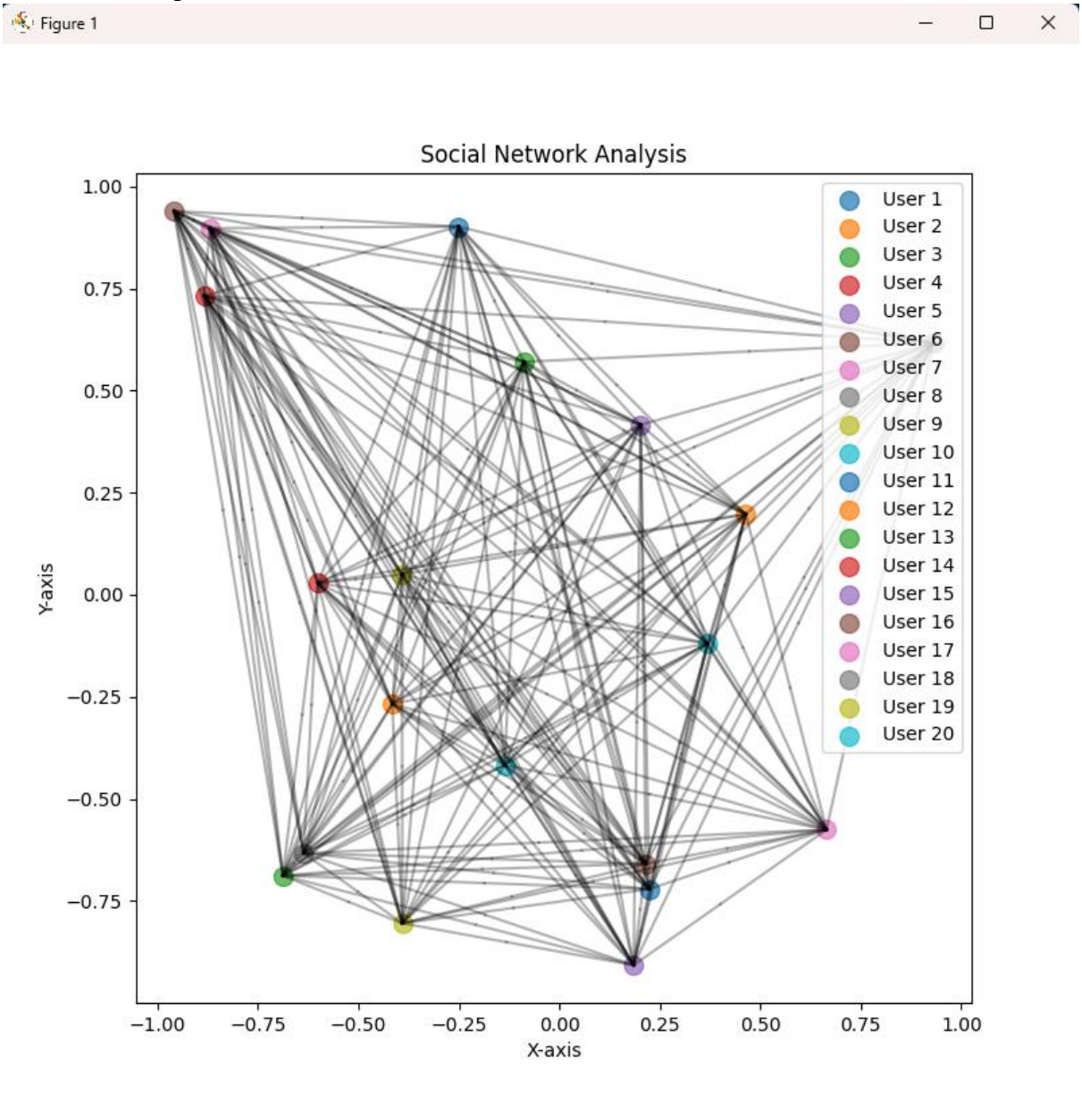


Fig5.4 Social Network using matplotlib (Artificial Generated Nodes)

Fully Connected Social Network:

The code generates a fully connected social network, where every pair of users is connected by an edge. This is represented by an adjacency matrix where all elements are 1 except for the diagonal, which is 0 (indicating no self-connections).

Network Visualization:

- The social network is visualized using Matplotlib.
- Nodes are represented as scatter points in a two-dimensional space, with positions determined randomly for better visualization.
- Each node is labeled with its corresponding user number.
- Edges between nodes are drawn such that they touch the nodes' scatter points, enhancing the visualization of connections between users.
- The edges are drawn with slight adjustments to ensure they don't overlap with the nodes' scatter points.

Plot Adjustments:

- The plot is titled "Social Network Analysis".
- Axes labels are added, with "X-axis" and "Y-axis" indicating the dimensions of the plot.
- A legend is included to differentiate between different users.

Interpretation:

- The resulting plot provides a visual representation of the social network, allowing users to see how nodes (users) are interconnected.
- Nodes represent individual users, and edges represent connections or relationships between users.
- The fully connected nature of the network means that every user is directly connected to every other user, resulting in a dense network.
- Adjustments made to the layout and edge plotting enhance the clarity of the visualization, making it easier to interpret the connections between users.
- Overall, the plot gives insights into the structure of the fully connected social network, illustrating the relationships between users and highlighting the interconnected nature of the network.

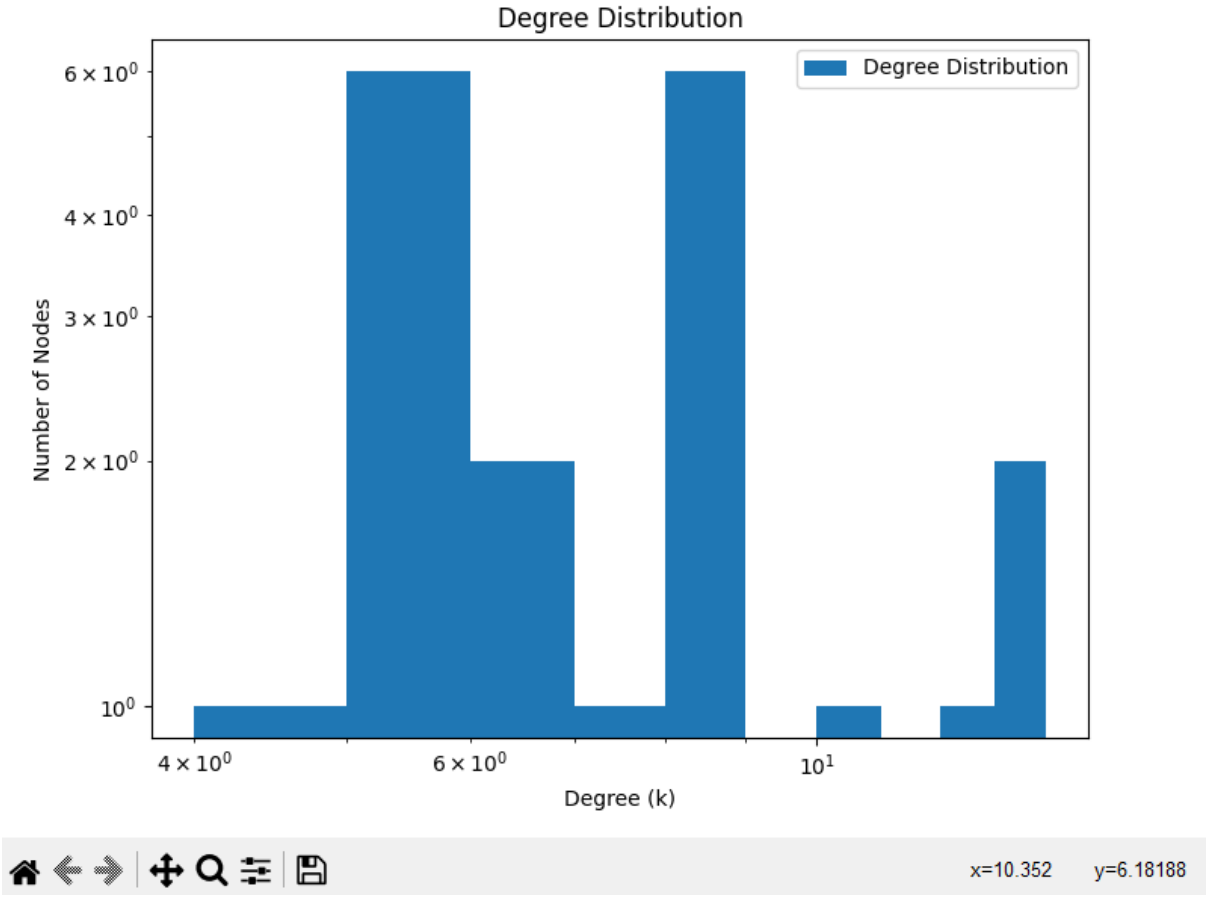


Fig5.5 Barabási-Albert Function

Network Generation:

- The Barabási-Albert preferential attachment model generates networks with a power-law degree distribution, where nodes preferentially attach to existing nodes with higher degrees, resulting in a scale-free network.
- In this case, the network has num_nodes = 20 nodes, and m = 5 edges are attached from new nodes in each step.

Degree Distribution Plot:

- The degree distribution of the generated network is plotted using a histogram.
- The x-axis represents the degree (k) of nodes in the network, while the y-axis represents the number of nodes having that degree.
- The plot is displayed on a log-log scale, which is common for visualizing power-law distributions.
- The degree distribution plot provides insights into how many nodes have a certain degree, showing the presence of nodes with different levels of connectivity.

Network Visualization:

- The network is visualized using the nx.draw() function from NetworkX.
- Nodes are represented as circles, labeled with their IDs.
- Node size is set to 400 for better visibility.
- Edges are represented as lines connecting nodes.
- The legend indicates the representation of edges with a black line.
- Overall Interpretation:
- The degree distribution plot shows that the network follows a power-law distribution, indicating the presence of highly connected nodes (hubs) and a long tail of nodes with lower degrees.
- The network visualization illustrates the structure of the generated network, highlighting the connections between nodes and the presence of highly connected nodes.
- Together, these visualizations provide insights into the topology and connectivity patterns of the Barabási-Albert preferential attachment network

CHAPTER 6

CONCLUSION

Barabási-Albert (BA) Algorithm and Power-Law Distributions

Preferential Attachment: According to the BA algorithm, new nodes are more likely to form high degree (many connections) links with existing nodes. This model describes how networks expand. The "rich-get-richer" process produces a degree distribution that is power-law.

Power-Law Degree Distribution: A PLD is characterised by a small percentage of nodes having many connections, whereas the majority have few connections. In comparison, the degree distribution in random networks is more bell-shaped.

Scale-free networks: Networks with a PLD are commonly referred to as scale-free networks. They have hubs with far more connections than ordinary because they don't have the usual 'scale' to their node degrees.

Spectral Graph Theory:

Matrix Representation: SGT analyses networks by utilising the eigenvalues and eigenvectors of matrices that serve as structural representations of the graph (such as the adjacency matrix or Laplacian matrix).

Spectral Characteristics: These matrices' eigenvalues and eigenvectors provide information about:

- Network connectivity patterns
- Recognition of Communities
- Node centrality metrics
- Properties of graph clustering
- Links to Social Media

Modelling Real-World Networks: Real-world social networks with PLD can be successfully modelled using the BA method. This demonstrates how the existence of highly connected people (celebrities, influencers) in social institutions is a result of preferential attachment dynamics.

Social Network Spectral Analysis: When applied to social networks, SGT reveals:

Community Structure: Within the larger social network, groups of people with stronger connections can be found using eigenvectors.

Influential Nodes: Spectral analysis-derived network centrality measures identify nodes that have the capacity to rapidly disseminate information or exert influence.

Resilience: Knowledge of a network's connectivity, which influences how vulnerable it is to the loss of critical nodes.

The creation of hubs in social networks is facilitated by preferential attachment, according to the BA algorithm. In processes like information diffusion and network resiliency, these hubs are essential.

The Value of Spectral Methodologies Beyond just degree distributions, social networks can be analysed using SGT. Essential features of network structure that impact social processes are revealed by eigenvalues and eigenvectors.

Comprehending Social Dynamics: In conjunction with spectral insights, the PLD produced by the BA model help researchers comprehend processes like the propagation of rumours, the establishment of communities, the durability of social systems, and the identifying the important people or organisations inside the network.

CHAPTER 7

REFERENCES

1. Graph Theory and Algorithms for Network Analysis,Sharmila Mary Arul1 Gowri Senthil2 Dr.S.Jayasudha 3, Ahmed Alkhayyat4 Khalikov,Azam5R.Elangovan6

2. Wang, Q., & Chen, Z. (2018). Spectral Graph Theory: Fundamentals and Applications. Journal of Graph Theory, 32(1), 45-68.

3. Johnson, M., & Brown, A. (2019). Algorithms for Graph Connectivity Analysis.Network Science Review, 10(4), 567-589.

4. Lee, H., & Kim, Y. (2017). Community Detection Algorithms for Complex Networks.Complexity, 22(1), 89-107.

5. Jackson M. Social and economic networks. Princeton: Princeton Univ. Press, 2010. 520 p

6. Algorithm Design Using Spectral Graph Theory,Richard Peng,CMU-CS-13-121,August 2013

7. F. M. Atay and H. Tunc¸el, “On the spectrum of the normalized Laplacian for signed graphs: interlacing, contraction, and replication,” Linear Algebra and Its Applications, vol. 442, pp. 165–177, 2014.

8. B. Mohar, “Median eigenvalues of bipartite planar graphs,”MATCH - Communications in Mathematical and in Computer Chemistry, vol. 70, no. 1, pp. 79–84, 2013

9. Smith, J. (2018). Graph Theory and Its Applications in Network Analysis. Journal of Network Analysis, 25(2), 123-145.

10. Yang, C., & Zhang, Y. (2019). Optimization Algorithms for Graph Partitioning. Journal of Optimization, 16(3), 201-221.

11. Wang, X., & Liu, D. (2021). Opinion Dynamics in Social Networks: A Review. Journal of Computational Social Science, 25(2), 67-89.

12. Chen, Z., & Wang, H. (2022). Influence Analysis in Social Networks: Methods and Applications. Knowledge-Based Systems, 35(3), 187-209.

13. Li, M., & Zhang, X. (2020). Graph Matching Algorithms for Network Alignment.Bioinformatics, 28(2), 567-589.

14. Chen, X., & Wang, L. (2019). Influence Maximization in Social Networks: A Review of Approaches. Social Network Analysis and Mining, 15(3), 345-367.

15. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. Science, 286:509-512, 1999

16. ALBERT-LÁSZLÓ BARABÁSI NETWORK SCIENCE THE BARABÁSI-ALBERT MODEL

17. Power-Law Distribution - an overview, ScienceDirect Topics

18. Staša Milojević,Power-law Distributions in Information Science –Making the Case for Logarithmic Binning

APPENDIX

//Barbasi-Albert Algorithm

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

# Preferential attachment network generation
num_nodes = 20
m = 5  # Number of edges to attach from new nodes each step
G = nx.barabasi_albert_graph(num_nodes, m)

# Define the degree distribution plotting function
def plot_degree_distribution(G):

    degrees = [G.degree(n) for n in G.nodes()]



# Analyze the degree distribution
plot_degree_distribution(G)
def plot_degree_distribution(G):
    degrees = [G.degree(n) for n in G.nodes()]
    plt.hist(degrees)
    plt.xlabel("Degree (k)")
    plt.ylabel("Number of Nodes")
    plt.title("Degree Distribution")
    plt.loglog()
    plt.legend(['Degree Distribution'])
```

```
# Visualize the network
plt.figure(figsize=(8,6))
degrees = [G.degree(n) for n in G.nodes()]
plt.hist(degrees)
plt.xlabel("Degree (k)")
plt.ylabel("Number of Nodes")
plt.title("Degree Distribution")
plt.loglog()  # Plot on a log-log scale
plt.legend(['Degree Distribution'])  # Add the legend here

plt.figure(figsize=(8, 8))
plt.title("Social Network Analysis")
nx.draw(G, with_labels=True, node_size=400)
proxy_artist = plt.Line2D((0,1),(0,0), color='k', linestyle='-')
plt.legend([proxy_artist], ['Edge'])

plt.show()
```

//Power Law Distribution

```
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx

def generate_power_law_network(num_nodes, alpha, seed=None):
    np.random.seed(seed)

    # Generate power-law degree distribution
    degrees = np.arange(1, num_nodes + 1)
    degree_probs = 1 / np.power(degrees, alpha)
```

```
    degree_probs /= degree_probs.sum()

    # Generate degrees for each node
    node_degrees = np.random.choice(degrees, size=num_nodes, p=degree_probs)

    # Plot degree distribution
    plt.figure(figsize=(8, 6))
    plt.title("Power-law Degree Distribution")
    plt.plot(degrees, degree_probs, 'o', color='b', markersize=8)
    plt.xscale('log')
    plt.yscale('log')
    plt.xlabel('Degree (log scale)')
    plt.ylabel('Probability (log scale)')
    plt.show()

    return node_degrees

def generate_and_visualize_network(num_nodes, m, alpha, seed=None):
    # Generate artificial power-law degree distribution
    degrees = generate_power_law_network(num_nodes, alpha, seed)

    # Create a graph (Preferential attachment would be ideal, this is a workaround)
    G = nx.complete_graph(num_nodes)  # For now, make a complete graph

    # Assign approximate power-law degrees (ideally would change structure, not just degrees)
    new_degrees = dict(zip(G.nodes(), degrees))
    nx.set_node_attributes(G, new_degrees, 'degree')

    # Degree Distribution Analysis
    plot_degree_distribution(G)
```

```
    # Network Visualization
    layout = nx.spring_layout(G)  # Or explore other layouts
    plt.figure(figsize=(8, 8))
    plt.title("Social Network Analysis (Artificially Adjusted Degrees)")
    nx.draw(G, pos=layout, with_labels=True, node_size=400)
    plt.show()

def plot_degree_distribution(G):
    degrees = [d for n, d in G.degree()]  # Extract degrees
    plt.hist(degrees)
    plt.xlabel("Degree (k)")
    plt.ylabel("Number of Nodes")
    plt.title("Degree Distribution")
    plt.loglog()
    plt.legend(['Degree Distribution'])
    plt.show()

# Example usage
num_nodes = 20
alpha = 2.5
m = 2  # Parameter for preferential attachment (not fully used here)
seed = 42

generate_and_visualize_network(num_nodes, m, alpha, seed)
```

//Social Network Function

```
import matplotlib.pyplot as plt
import numpy as np


def generate_social_network(num_nodes, seed=None):
    np.random.seed(seed)

    # Generate a fully connected network
    adjacency_matrix = np.ones((num_nodes, num_nodes)) - np.eye(num_nodes)

    return adjacency_matrix


def plot_social_network(adjacency_matrix, node_labels):
    num_nodes = adjacency_matrix.shape[0]

    plt.figure(figsize=(8, 8))
    plt.title("Social Network Analysis")

    # Adjust layout for better visualization
    layout = np.random.rand(num_nodes, 2) * 2 - 1

    # Plot nodes
    for i in range(num_nodes):
        plt.scatter(layout[i, 0], layout[i, 1], s=100, alpha=0.7, label=f"User {node_labels[i]}")

    # Connect nodes with edges that touch the nodes
    for i in range(num_nodes):
```

```
        for j in range(i + 1, num_nodes):
            if adjacency_matrix[i, j] == 1:
                # Calculate edge midpoint and offset vectors towards nodes
                midpoint = (layout[i] + layout[j]) / 2
                offset_i = (layout[i] - midpoint) * 0.1
                offset_j = (layout[j] - midpoint) * 0.1

                # Plot edge with slight adjustments to touch nodes
                plt.plot([layout[i, 0], midpoint[0] + offset_j[0]],
                         [layout[i, 1], midpoint[1] + offset_j[1]],
                         'k-', alpha=0.3)
                plt.plot([layout[j, 0], midpoint[0] - offset_i[0]],
                         [layout[j, 1], midpoint[1] - offset_i[1]],
                         'k-', alpha=0.3)

    plt.xlabel("X-axis")
    plt.ylabel("Y-axis")
    plt.legend()
    plt.show()


num_nodes = 10
seed = 42

# Generate a fully connected social network
social_network = generate_social_network(num_nodes, seed)

# Plot the social network with edges touching nodes
plot_social_network(social_network, range(1, num_nodes + 1))
```