20240404_Barabasi Network with attached Vectors

April 4, 2024

[2]:
```
import networkx as nx
import numpy as np
from scipy.stats import rv_discrete

def generate_power_law_degree_sequence(n, exponent, minimum_degree=1):
    """Generate a degree sequence with a power-law distribution."""
    # Adjust probabilities to generate a sequence with an even sum.
    while True:
        # Probability distribution for the degrees
        probabilities = np.array([((i + minimum_degree) ** (-exponent)) for i␣
 ↪in range(n)])
        probabilities /= probabilities.sum()

        # Create a random variable with the specified distribution
        distribution = rv_discrete(name='custm', values=(np.arange(n),␣
 ↪probabilities))

        # Generate the degree sequence
        degree_sequence = distribution.rvs(size=n) + minimum_degree

        # Check if the sum of the degree sequence is even
        if sum(degree_sequence) % 2 == 0:
            break
        # If the sum is odd, we adjust a random degree by 1 (making minimal␣
 ↪impact on the distribution)
        else:
            degree_sequence[np.random.randint(n)] += 1
            if sum(degree_sequence) % 2 == 0:
                break

    return degree_sequence

def create_network_with_degree_sequence(degree_sequence):
    """Create a network from a given degree sequence."""
    # Use the configuration model to generate a random graph with the degree␣
 ↪sequence
    G = nx.configuration_model(degree_sequence)
```

```
    # Remove parallel edges and self-loops
    G = nx.Graph(G)  # Converts to simple graph, removing parallel edges and self-loops
    G.remove_edges_from(nx.selfloop_edges(G))

    return G

# Parameters for network generation
n = 100  # Number of nodes
exponent = 2.1
minimum_degree = 2  # Ensuring a minimum degree to help avoid disconnected nodes

# Generate a degree sequence with the desired power-law exponent
degree_sequence = generate_power_law_degree_sequence(n, exponent, minimum_degree)

# Generate the network
network = create_network_with_degree_sequence(degree_sequence)

# Attach vectors to each node in the network
def attach_vectors(network, mean, std_dev):
    for node in network.nodes():
        network.nodes[node]['vector'] = np.random.normal(mean, std_dev, 300)

# For the network, use N(1000, 30) or any other distribution as required
attach_vectors(network, 1000, 30)

# This code adjusts the degree sequence to ensure an even sum before network generation,
# addressing the requirement for undirected graphs.
```

[3]:
```
import networkx as nx
import numpy as np
from scipy.sparse import csr_matrix

# Assuming 'network' is your graph with attached vectors
# Calculate Laplacian matrix of the graph
laplacian = nx.laplacian_matrix(network).toarray()

# Number of nodes and the vector length
n_nodes = network.number_of_nodes()
vector_length = 300  # Length of vectors attached to each node

# Initialize an empty 3D array
three_dim_array = np.zeros((n_nodes, n_nodes, vector_length))
```

```
# Populate the X and Y dimensions with the Laplacian matrix
for i in range(vector_length):
    three_dim_array[:, :, i] = laplacian

# Populate the Z dimension with the vectors
for idx, node in enumerate(network.nodes(data=True)):
    vector = node[1]['vector']
    for i in range(n_nodes):
        for j in range(n_nodes):
            if i == j:  # Populate the diagonal with the vector elements
                three_dim_array[i, j, :] = vector

# Now `three_dim_array` is your 3D array with the desired properties
```

[4]:
```
three_dim_array
```

[4]:
```
array([[[1014.14908369, 1037.83883747, 1027.13509084, …,
          969.05497902, 1042.16192464, 1008.83334782],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        …,
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ]],

       [[   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [1014.14908369, 1037.83883747, 1027.13509084, …,
          969.05497902, 1042.16192464, 1008.83334782],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        …,
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ]],

       [[   0.        ,    0.        ,    0.        , …,
            0.        ,    0.        ,    0.        ],
        [   0.        ,    0.        ,    0.        , …,
```

```
       0.        ,    0.        ,    0.        ],
 [1014.14908369, 1037.83883747, 1027.13509084, …,
   969.05497902, 1042.16192464, 1008.83334782],
 …,
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ]],

…,

[[   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 …,
 [1014.14908369, 1037.83883747, 1027.13509084, …,
   969.05497902, 1042.16192464, 1008.83334782],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ]],

[[   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 …,
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [1014.14908369, 1037.83883747, 1027.13509084, …,
   969.05497902, 1042.16192464, 1008.83334782],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ]],

[[   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
 [   0.        ,    0.        ,    0.        , …,
       0.        ,    0.        ,    0.        ],
```

```
…,
[   0.        ,    0.        ,    0.        , …,
    0.        ,    0.        ,    0.        ],
[   0.        ,    0.        ,    0.        , …,
    0.        ,    0.        ,    0.        ],
[1014.14908369, 1037.83883747, 1027.13509084, …,
  969.05497902, 1042.16192464, 1008.83334782]]])
```

[ ]: