Team 2 Group HW 1

Question 1

In [ ]:

```
from collections import Counter
```

In [ ]:

```
def num_q_cliques(adj, q):
    """
    Computes the number of q-cliques of a given graph (defined through its adjacency matrix).
    Adapted from https://www.geeksforgeeks.org/find-the-number-of-cliques-in-a-graph/
    """
    cliques = []
    visited = set()

    def dfs(node, clique):
        visited.add(node)
        clique.add(node)
        for neighbor, connected in enumerate(graph[node]):
            if connected and neighbor not in visited:
                dfs(neighbor, clique)

    for node in range(len(graph)):
        if node not in visited:
            clique = set()
            dfs(node, clique)
            if len(clique) > 1:
                cliques.append(clique)

    print(cliques)
    return Counter(map(len, cliques))[q]
```

In [ ]:

```python
from itertools import combinations, product

def num_q_cliques(adj, q):
    """
    Computes the number of q-cliques of a given graph (defined through its adjacency matrix).
    """
    n = len(adj)
    node_combinations = combinations(range(n),q)
    cycles = [all(adj[x][y] for x, y in product(l,l) if x != y) for l in node_combinations]
    num_cycles = sum(cycles)
    return num_cycles
```

In [ ]:

```python
# Example usage (Testing)
"""
graph = {
    'A': ['B', 'C', 'D'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'D'],
    'D': ['A', 'B', 'C'],
    'E': ['F'],
    'F': ['E']
}
"""
graph = [
    [0, 1, 1, 0, 0, 0],
    [1, 0, 1, 1, 0, 0],
    [1, 1, 0, 1, 0, 0],
    [0, 1, 1, 0, 0, 0],
    [1, 0, 0, 0, 0, 1],
    [1, 1, 0, 0, 1, 0]]
num_q_cliques(graph, 2)
```

Out[ ]:

```
6
```

Question 2

In [ ]:

```python
# f = open("..\kn57Nodes1to57_adj20.txt", "r")
f = open("/Users/yuchenhongshu/UMD_Courses/MATH420/HW/amsc420-project-2/kn57Nodes1to57_adj20.txt", "r")
lines = f.readlines()
num_nodes, num_edges = map(int, lines[0].split())
adj_matrix = [list(map(int, line[:-1].split())) for line in lines[1:]]
```

In [ ]:

```python
print(f"""Number of nodes: {num_nodes}
Number of edges: {num_edges}
```

```
Number of 3-cliques (Triangles): {num_q_cliques(adj_matrix, 3)}
Number of 4-cliques (Complete four-vertex subgraphs): {num_q_cliques(adj_matrix, 4)}""")
```

```
Number of nodes: 57
Number of edges: 804
Number of 3-cliques (Triangles): 7075
Number of 4-cliques (Complete four-vertex subgraphs): 43711
```

Question 3

In [ ]:
```
p_MLE = (2 * num_edges) / (num_nodes * (num_nodes - 1))

print(f"Maximum Likelihood Estimate (MLE) for p under Erdos-Renyi model: {p_MLE}")
```

```
Maximum Likelihood Estimate (MLE) for p under Erdos-Renyi model: 0.5037593984962406
```

Question 4

In [ ]:
```
from math import comb

expected_q_cliques = lambda q: comb(num_nodes, q) * (p_MLE ** (q * (q - 1) / 2))

print(f"""Expected number of 3-cliques: {expected_q_cliques(3)}
Expected number of 4-cliques: {expected_q_cliques(4)}""")
```

```
Expected number of 3-cliques: 3740.6218553903564
Expected number of 4-cliques: 6455.755303411616
```

In [ ]:
```
# import numpy as np
# from numpy.linalg import matrix_power

# np.trace(matrix_power(np.array(adj_matrix),4))/24
```

Question 5

In [ ]:
```
m = num_edges
n = num_nodes
t = num_q_cliques(adj_matrix, 3)

c_1 = 4 * m / (n * (n - 1))
c_2 = 24 * t / (n * (n - 1) * (n - 2))
```

```python
a_MM = (1/2) * (c_1 + (2 * c_2 - c_1 ** 3) ** (1/3))
b_MM = (1/2) * (c_1 - (2 * c_2 - c_1 ** 3) ** (1/3))

print(f"""Expected value for a for the SSBM with 2-communities SSBM(n,2,a,b) by Method of Moments: {a_MM},
Expected value for b for the SSBM with 2-communities SSBM(n,2,a,b) by Method of Moments: {b_MM}""")
```

```
Expected value for a for the SSBM with 2-communities SSBM(n,2,a,b) by Method of Moments: 0.9885790067394369,
Expected value for b for the SSBM with 2-communities SSBM(n,2,a,b) by Method of Moments: 0.01893979025304432
```

Question 6

In [ ]:

```python
E_n1 = n / 2
E_n1_square = (n ** 2 + n) / 4
E_n1_cube = (n ** 2) * (n + 3) / 8
E_n1_quad = n * (n + 1) * (n ** 2 + 5 * n - 2) / 16

expected_4_cliques = (
    (a_MM ** 6) / 24 *
    (
        2 * E_n1_quad -
        4 * n * E_n1_cube +
        (6 * (n ** 2) - 18 * n + 22) * E_n1_square +
        (-4 * (n ** 3) + 18 * (n ** 2) - 22 * n) * E_n1 +
        (n ** 4) - 6 * (n ** 3) + 11 * (n ** 2) - 6 * n
    ) +
    (a_MM ** 3) * (b_MM ** 3) / 6 *
    (
        -2 * E_n1_quad +
        4 * n * E_n1_cube +
        (-3 * (n ** 2) + 3 * n - 4) * E_n1_square +
        ((n ** 3) - 3 * (n ** 2) + 4 * n) * E_n1
    ) +
    (a_MM ** 2) * (b_MM ** 4) / 4 *
    (
        E_n1_quad -
        2 * n * E_n1_cube +
        ((n ** 2) + n - 1) * E_n1_square +
        (-(n ** 2) + n) * E_n1
    )
)

print(f"Expected number of 4-cliques: {expected_4_cliques}")
```

```
Expected number of 4-cliques: 46089.160392458965
```