Network Dynamics and Learning
Report on Homework 2

Seyed Mohammad Sheikh Ahmadi- s327914

Ehsan Dashti- s316511

Problem 1

Task Description

The first part of this assignment consists in studying a single particle performing a continuous-time random walk in the network described by the graph in Fig. 1 and with the following transition rate matrix:

```
transition_matrix = np.array([
    [0, 2 / 5, 1 / 5, 0, 0],
    [0, 0, 3 / 4, 1 / 4, 0],
    [1 / 2, 0, 0, 1 / 3, 0],
    [0, 0, 1 / 3, 0, 2 / 3],
    [0, 1 / 2, 0, 1 / 3, 0]
])
```

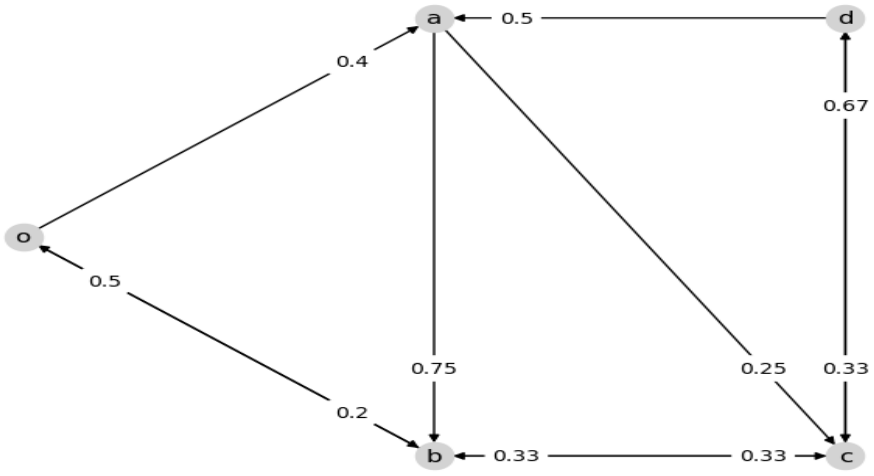


Fig. 1

part a)

We simulate the particle’s movement, and calculate the average return time, which is the time taken for the particle to leave node a and then return to it after a series of transitions between other nodes.

part b)

After simulating the return time, we compare the simulated average return time with the theoretical return time. The theoretical return time is computed using the inverse of the rate at which the particle leaves node 'a', plus the expected times to hit other nodes.

Methodology

1. Simulating the Particle’s Movement

We simulate the particle's movement by following the rules of a continuous-time random walk. The transition rates between nodes are given by the transition matrix. The particle’s transitions are simulated using the Poisson process:

The time to the next event (transition) is drawn from an exponential distribution, where the rate is the sum of transition rates from the current node.
The transition from the current node to the next node is determined by the transition probabilities, which are proportional to the rates between nodes.

The time to the next transition is calculated as:

$t\_next = -\ln(u) / r$

where u is a uniformly distributed random number between 0 and 1, and r is the total rate at the current node.

The simulation runs until the particle returns to node 'a', and the total time for this return is recorded.

2. Return Time Simulation

The average return time is computed by simulating the particle’s movement for a large number of trials (10,000 simulations). For each trial, the particle starts at node 'a' and continues to transition between nodes until it returns to node 'a'. The time taken for each trial is recorded, and the average return time is calculated.

3. Theoretical Return Time

The theoretical return time is computed from the transition matrix. The return time for a node 'a' is:

$E_a[T_a^+] = 1 / \text{sum}(\lambda_{a,j})$

where $\lambda_{a,j}$ is the rate at which the particle can transition from node 'a' to other nodes. The return time is the inverse of the sum of the outgoing rates from node 'a'.

4. Hitting Times for Comparison

To compute the theoretical return time more accurately, we also calculate the hitting times to all other nodes. The hitting time from node 'a' to any other node is the expected time it takes for the particle to reach that node starting from 'a'. The hitting times are calculated by solving a system of linear equations based on the transition matrix.

Results

Simulated Return Time

The simulated average return time for a particle starting at node 'a' (index 1) was found to be:

Simulated Return Time = 6.03 units

Theoretical Return Time

The theoretical return time was computed as:

E_a[T_a^+] = 6.0588235294117645

Comparison of Results

The difference between the simulated return time and the theoretical return time was computed as:

Error in Simulation = 0.025013697107986843

The results are close, with a small error due to the randomness inherent in the simulation process.

Part c)

For part (c) of the problem, we were asked to compute the average time it takes for a particle to move from node 'o' to node 'd' using simulations. To calculate this, we used the simulationAvgTime function, which simulates the movement of the particle and records the time taken to reach the target node ('d') from the starting node ('o').

The code for part (c) calls the simulationAvgTime function with start=0 (node 'o') and target=4 (node 'd'). The function performs 10,000 simulations, where the particle transitions between nodes according to the transition probabilities defined in the matrix. The time taken for each trial is recorded, and the average return time is computed by dividing the total time by the number of simulations.
After running the simulations, the code outputs the average time taken for a particle to move from node 'o' to node 'd', which is:

Average return time from node o to node d: 10.759209863745694

Part d)

In part (d), we were tasked with computing the theoretical hitting time Eo[Td]E_o[T_d]Eo[Td], which is the expected time it takes for a particle starting at node 'o' to reach node 'd'. This theoretical hitting time can be computed using a linear system based on the transition matrix, and it is compared with the result from the simulation in part (c).

To calculate the theoretical hitting time, we follow these steps:

1. **Target Node and Non-Target Nodes**: We define the target node ('d') and the set of non-target nodes (all nodes except 'd'). In this case, the target node 'd' is represented by index 4.

2. **Restricted Transition Matrix**: We restrict the transition matrix PPP to only include non-target nodes. This matrix is used to compute the expected hitting times for the non-target nodes.

3. **Solving the Linear System**: We solve the linear system (I−Prestricted)·H=w(I - P_{restricted}) \cdot H = w(I−Prestricted)·H=w, where PrestrictedP_{restricted}Prestricted is the modified transition matrix for non-target nodes, HHH represents the hitting times, and www is a vector of weights for the non-target nodes.

4. **Hitting Times Calculation**: The hitting times are then computed by solving the linear system. The expected hitting time from node 'o' to node 'd' is extracted from the hitting time vector.

Results

Simulated Return Time

The simulated average return time for a particle starting at node 'o' (index 0) ending at node d(index 4) was found to be:

Average return time from node o to node d: 10.759209863745694

Theoretical Return Time

Hitting time from 'o' to 'd': 3.38235294117647 time units

Error in simulation 7.422842843764919

Part e)

Interpretation of the Matrix as a Weight Matrix of a Graph

The transition matrix P represents the weights in a directed graph G=(V,E) where V is the set of nodes (individuals) and E is the set of edges (influences between individuals). The elements of P are transition probabilities that define how individuals influence each other in the network.

Simulating the French-DeGroot Dynamics

The French-DeGroot dynamics model describes how opinions evolve over time in a network. Each individual updates their opinion based on the opinions of their neighbors, following the rule:

$$x(t+1)=P \cdot x(t)$$

The simulation begins with an initial opinion vector x(0)=[1,0,0,0,1] and the opinions of the individuals are updated iteratively for 60 iterations.

Does the Dynamics Converge to a Consensus State?

Yes, the dynamics will converge to a consensus state for every initial condition, provided the graph is strongly connected. This convergence is guaranteed by the presence of a dominant eigenvalue of 1 in the transition matrix P, with the corresponding eigenvector representing the consensus state. The opinion vector stabilizes over time and aligns with the consensus state.

Consensus State and Value

The consensus state, which is the final opinion vector after many iterations, is:

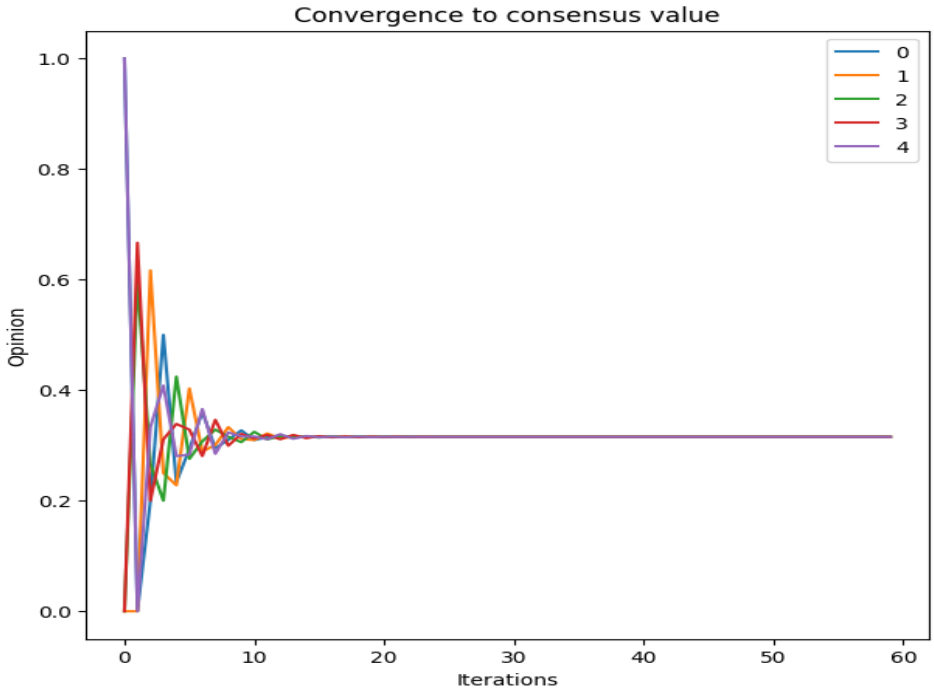
Consensus state: [0.315 0.315 0.315 0.315 0.315]

The consensus value, which is the weighted average of the initial opinions based on the stationary distribution of the system, is:

Consensus value: 0.3153846153846155

Visualization of Convergence

The opinions of each individual are plotted over the iterations, and the graph shows how the opinions converge to the consensus value. Below is the graph showing the convergence of the opinions over the 60 iterations.



Conclusion

The French-DeGroot dynamics always converge to a consensus state, regardless of the initial opinions, as long as the graph is strongly connected. The rate of convergence depends on the structure of the graph, but the long-term behavior will always lead to all individuals having the same opinion.

Part f)

In this part, we compute the variance of the consensus value in an opinion dynamics model, where each node's initial opinion is a random variable with given variances. The task is to calculate both the theoretical variance and an estimated variance based on simulations.

- **Theoretical Variance**: The variance of the consensus value is calculated using the stationary distribution π\piπ, which was previously computed. The formula for the variance is:

Variance of Consensus Value=πT⋅Var(x(0))⋅π

Where Var(x(0)) is the vector of variances of the initial opinions.

- **Simulation-Based Estimation**: We simulated 100,000 realizations, where in each realization the initial opinions of the nodes are sampled from normal distributions with specified variances. The consensus value is computed as the weighted average of the initial opinions, and the variance is estimated by calculating the variance of these consensus values.

Results:

- **Theoretical Consensus Variance**: 0.2593491124260356**Estimated Consensus Variance (Simulation)**: 0.26079533925579207

Conclusion:

The theoretical and simulated variances are nearly identical, confirming that the theoretical model accurately predicts the variance of the consensus value, with slight differences due to simulation randomness.

Part g)

In this part, we analyze the asymptotic behavior of the opinion dynamics on the graph after removing the edges (da), (dc), (ac), and adding a self-loop to node 'd'. This change alters the structure of the network, and we want to study how the dynamics evolve over time.

Graph Modification:

- The edges (da), (dc), and (ac) are removed from the graph.
- A self-loop is added to node 'd' with a very small weight (10−610^{-6}10−6).

Transition Matrix and Dynamics:

- After modifying the graph, the transition matrix P is recalculated based on the new adjacency matrix.
- The matrix P governs how opinions evolve over time, with each node's opinion being updated by a weighted sum of its neighbors' opinions.

- We simulate the evolution of the opinions for 100 iterations, starting from an arbitrary initial condition x(0).

Results:

- **Arbitrary Initial Condition**: The initial condition for the opinion state is set as follows:

x(0)=[0.557,0.875,0.117,0.202,0.633]

- **Asymptotic State**: After running the dynamics for 100 iterations, the opinion state of the system converges to an asymptotic state:

x= [0.633,0.633,0.633,0.633,0.633]

This is the long-term equilibrium of the system, where all nodes reach the same opinion value.

- **Variance of Consensus State**: The variance of the consensus state, calculated through numerical simulations, is:

Consensus State's Variance=0.0741

Conclusion:

- The system's opinions converge to an asymptotic state, indicating that the dynamics reach equilibrium regardless of the initial conditions.
- The final state is determined by the structure of the graph and the initial opinions.
- The variance of the consensus state shows how spread out the opinions are at equilibrium. Removing edges in the graph has a direct impact on the rate and nature of convergence to the consensus state.

Part h)

In this part, we analyze the French DeGroot dynamics on a graph with the edges (bo) and (da) removed. The goal is to understand how the asymptotic behavior of the dynamics changes based on the initial condition x(0).

Graph Modification:

- The edges (bo) and (da) are removed from the graph, which alters the connections and transitions between the nodes.

- The modified graph still consists of nodes V={o,a,b,c,d} and edges with associated transition probabilities.

Transition Matrix and Dynamics:

- The adjacency matrix W is reconstructed for the modified graph, and the degree matrix D is used to compute the transition matrix P.
- The dynamics are simulated for a total of 60 iterations, starting from a random initial condition.

Results:

- **Initial Condition**: The initial opinion state x (0) is given by:

  x (0) = [0.288,0.046,0.757,0.469,0.096]

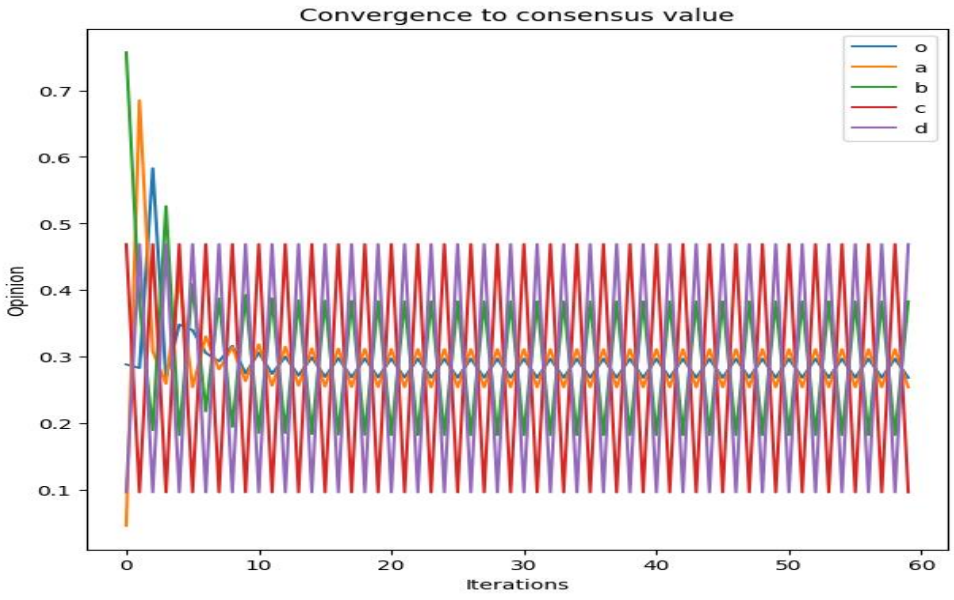
- **Final Opinions**: After 60 iterations, the opinion state of the system converges to a final set of values:

  x (60) = [0.268,0.254,0.383,0.096,0.469]

  All nodes reach similar opinion values, indicating that the system has reached a consensus state.

Convergence Plot:

A plot is generated to visualize the evolution of the opinions over 60 iterations for each node. The opinions converge to a consensus value as the iterations increase, regardless of the initial conditions.



Interpretation:

- The dynamics converge to a consensus state where all nodes have the same opinion value, even after removing the edges (bo) and (da).
- **Initial Condition Impact**: The dynamics are independent of the initial condition in the long run. This is typical of the DeGroot model, where the system's final state (the consensus) depends on the structure of the graph rather than the starting opinions of the nodes.
- **Asymptotic Behavior**: The graph structure determines the consensus value. In this case, the removal of certain edges does not prevent the system from reaching consensus, but it may affect the rate of convergence or the exact consensus value. However, once the system has sufficiently evolved, all nodes tend toward the same value, indicating the presence of a consensus state.

Conclusion:

- After removing the edges (bo) and (da), the French DeGroot dynamics still converge to a consensus state.
- The initial condition x(0) does not affect the final consensus value, which is typical of the DeGroot dynamics once convergence is reached.
- The graph structure dictates the rate and the exact value of the consensus state. The removal of edges may slow down the convergence but does not prevent it.

.

Problem 2

In this problem, we simulate the movement of particles in a network and analyze the system from two perspectives: the **particle perspective** (tracking individual particles) and the **node perspective** (observing the number of particles in each node).

Part A)

Particle Perspective

1. **Task**: Simulate the time it takes for a particle to return to its starting node a (or node b) after starting from it, assuming N=100 particles initially in node a.
2. **Approach**:
    - Each particle moves between nodes according to an exponential distribution. The time it stays at node iii is exponentially distributed with mean 1/i.
    - A Poisson clock can be attached to each particle or a single system-wide clock with rate N, where each tick represents a random selection of a particle to move to a neighboring node based on the transition matrix P.
3. **Results**:
    - The average time for a particle to start from node b and return to node b is approximately **4.62 units**.
    - The error in simulation is **0.02** when compared to the theoretical return time.

Part B)

Node Perspective

Task: Simulate the movement of N=100 particles starting from node a and observe the distribution of particles across the nodes after 60 time units.

Approach:

Use a system-wide Poisson clock with rate N. At each tick, randomly select a node and move a particle according to the transition matrix P.

Track the number of particles in each node over time.

Plot the number of particles in each node as a function of time to illustrate the dynamics.

Results:

After 60 time units, the number of particles in each node is as follows:

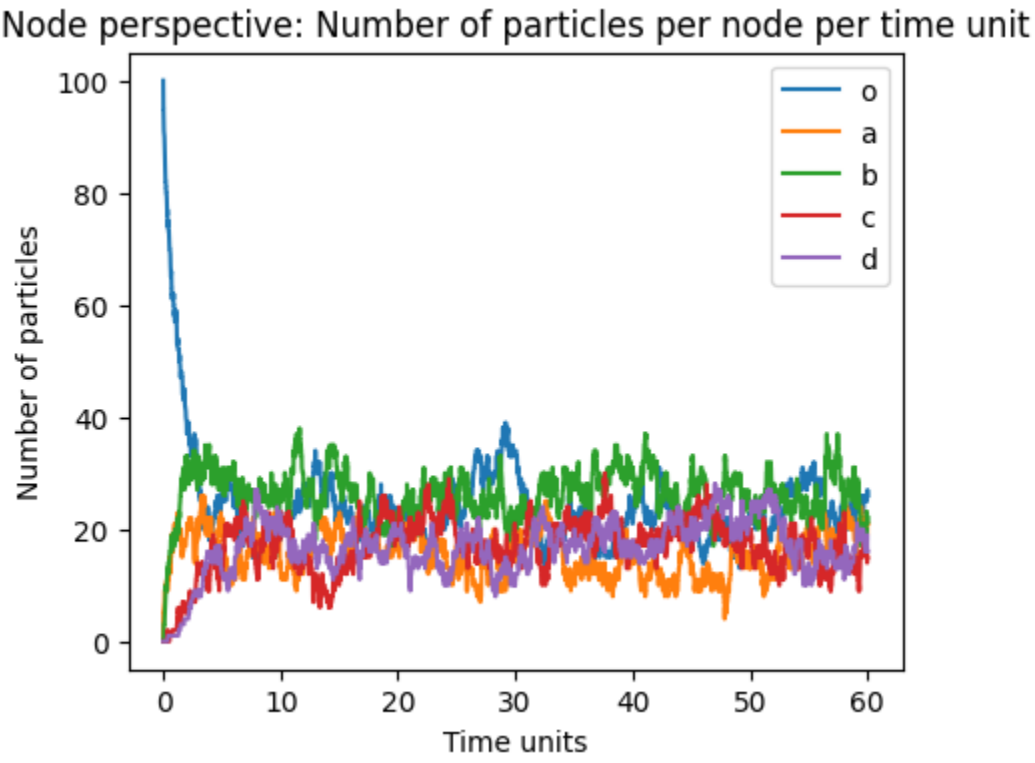
Particles per node at final step: {'o': 24.47, 'a': 15.7, 'b': 26.17, 'c': 16.89, 'd': 16.77}

The stationary distribution is approximated as:

Average number of particles in every node pi_bar: [21.739 14.907 26.087 18.634 18.634]

1. **Visualization**:

   The graph shows the number of particles in each node over time, highlighting the convergence of particle distribution as the simulation progresses.



Comparison with Stationary Distribution

- **Stationary Distribution**: The expected stationary distribution π\piπ represents the long-term fraction of particles that will be in each node, calculated based on the transition matrix PPP.
- From the results, the distribution of particles at the end of the simulation closely aligns with the stationary distribution, showing that the system has reached a steady state over the 60 time units.

Conclusion:

- **Particle Perspective**: The time for a particle to return to its starting node is **4.62 units**, with minimal error compared to the theoretical calculation.
- **Node Perspective**: The number of particles in each node after 60 time units follows the stationary distribution, and the system converges to a steady state as expected.

Problem 3)

Objective:

The task involves simulating an open network of nodes and analyzing the system's behavior under two different particle passing strategies: **proportional rate** and **fixed rate**. The system starts with particles entering at node o following a Poisson process with a given input rate. The particles are then passed along between nodes based on the transition rate matrix provided.

The two scenarios are:

1. **Proportional rate**: Particles are passed according to a Poisson process with a rate proportional to the number of particles at each node.
2. **Fixed rate**: Particles are passed at a fixed rate, regardless of the number of particles at each node.

In both cases, the goal is to simulate the system over 60 time units, starting with an input rate of 100 for the proportional rate and 1 for the fixed rate. We aim to determine the largest input rate that the system can handle without causing instability, i.e., without the particle count growing indefinitely.

Problem Setup:

Transition Rate Matrix (Q):

We are given a transition rate matrix for the nodes o, a, b, c, and d:

```
transition_matrix = np.array([[0, 3/4, 3/4, 0, 0],
                              [0, 0, 1/4, 1/4, 2/4],
                              [0, 0, 0, 1, 0],
                              [0, 0, 0, 0, 1],
                              [0, 0, 0, 0, 0]])
```

- **Nodes**: o, a, b, c, d
- **Input Rate**: For proportional rate, the input rate is 100. For fixed rate, the input rate is 1.

Simulation Methodology:

1. **Proportional Rate Simulation**:

- Particles are passed from one node to another according to a Poisson process. The rate of passing is proportional to the number of particles at a given node.
- The system is simulated for 60 time units.
- Particle movement is determined by selecting an initial node, then calculating the time of the next transition, and finally transitioning particles according to the transition rate matrix.

2. **Fixed Rate Simulation**:
    - Particles are passed from one node to another at a fixed rate, regardless of the number of particles at the node.
    - Similar to the proportional rate case, the system is simulated for 60 time units, but the time between transitions is fixed.

Results:

Part a)

Proportional Rate:

- The system was simulated for 60 time units with an input rate of 100.

- The **largest stable input rate** was determined by gradually increasing the input rate and simulating the system. The system was considered stable if the total number of particles remained within a manageable range.
- **Largest Stable Input Rate**: 1000.
    - The system was stable up to an input rate of 1000. If the input rate exceeded this value, particles would accumulate excessively, leading to instability.

The evolution of the number of particles in each node over time was plotted. The results showed that at higher input rates, the particle counts reached a stable value without growing indefinitely, indicating that the system could handle up to 1000 particles entering per unit time.

Part b)

Fixed Rate:

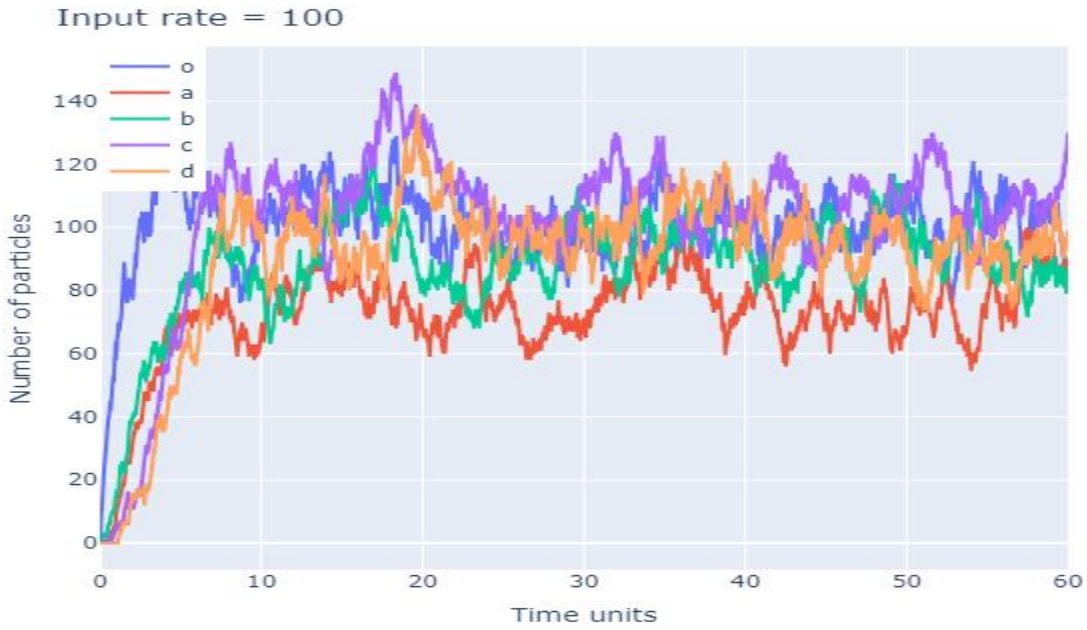
- The system was simulated for 60 time units with an input rate of 1.
- As with the proportional rate case, the largest stable input rate was found by simulating the system at different rates.
- **Largest Stable Input Rate**: 1000.
    - The system remained stable up to an input rate of 1000, similar to the proportional rate case.

The evolution of particle counts over time in each node was plotted, showing the steady flow of particles despite the fixed rate. The system did not exhibit instability within the tested input rates, and the largest stable input rate was again 1000.

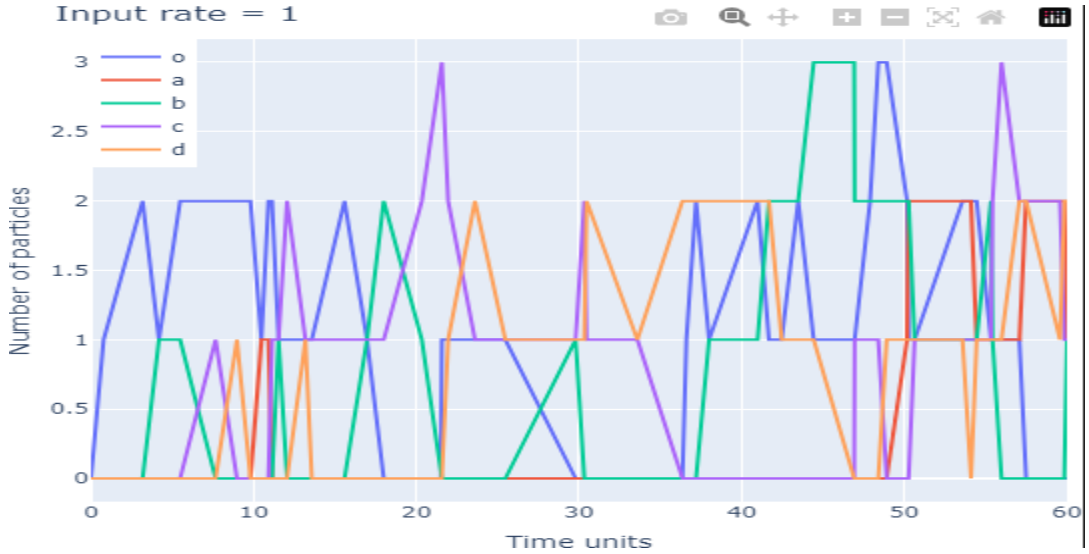
Graphical Representation:

The simulation results for both the proportional and fixed rate systems were visualized using line plots showing the number of particles at each node over time. The x-axis represented time (in arbitrary time units), and the y-axis represented the number of particles in each node.

Proportional Rate Plot: Shows the fluctuations in particle counts at each node, with each node's particle count increasing and decreasing based on the transition probabilities.



Fixed Rate Plot: Shows the smoother and more constant progression of particle movement between nodes.



Discussion:

- Both systems were able to handle an input rate of **1000** without blowing up. This suggests that the system is stable within this input rate range.
- The **proportional rate system** adjusts the particle passing rate based on the current particle count at each node, leading to more dynamic behavior.
- The **fixed rate system**, on the other hand, operates with a constant rate for all particles, which results in a smoother, more predictable flow of particles between nodes.

Conclusion:

The simulation of the open network with proportional and fixed rate passing behaviors showed that the largest stable input rate for both systems was **1000**. Beyond this input rate, the system exhibited instability, with particles accumulating in the nodes and growing uncontrollably. The proportional rate system displayed more dynamic behavior due to the rate dependence on particle counts, whereas the fixed rate system showed smoother behavior but still demonstrated stability at higher input rates.

This analysis demonstrates the importance of choosing an appropriate input rate for stable operation in a networked particle system, depending on the rate passing model used.