Network Dynamics and Learning
Homework II

Francesco Grandi
Student id: 306272
Politecnico di Torino
Turin, Italy
s306272@studenti.polito.it
Exchanged ideas with Alessandro Casella (student id:306081) and Elisa Feraud (student id: 295573)
In this report, some results can change at every run of the related code. For this reason, not all the results have been reported.

1. Problem 1

The first problem consists in simulating a continuous-time random walk of a particle in a network represented by the directed graph in Figure 1. This is achieved using a transition matrix Λ, describing the rate at which the Markov Chain passes through different states.

$$\Lambda = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 1/3 & 0 & 1/3 & 0 \end{pmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \tag{1}$$

To simulate a continuous random walk, a Poisson clock is assigned to every node. Therefore, given the current state i, the particle waits for ω_i-rate exponential time then jumps to a new state j chosen with probability P_{ij} defined in Equation (2).

$$\omega_i = \sum_j \Lambda_{ij}, \quad P_{ij} = \frac{\Lambda_{ij}}{\omega_i} \tag{2}$$

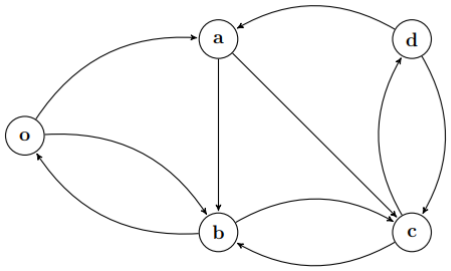


Figure 1: Closed network with transition matrix in (1)

1.1. Problem 1a

To find the average time it takes a particle that starts in node a to leave the node and return to it, for each step of a $100'000$-iterations cycle it is computed a random time elapsed from the last iteration according to the Poisson distribution defined as below:

$$t_{next} = -\frac{\ln(u)}{r} \tag{3}$$

where u is randomly chosen from a uniform distribution over $(0, 1)$.
Whenever the particle goes back to state a, the time elapsed is stored in a variable and a counter of how many times the particle passes through a is increased by one. By considering all the simulation times, the obtained average return time to a is $\hat{T}_a^+ \approx 6.830$.

1.2. Problem 1b

To compare the simulated return time with the theoretical one, Kac's formula is used:

$$\mathbb{E}_a[T_a^+] = \frac{1}{\omega_a \overline{\pi}_a} \tag{4}$$

where ω_a is defined in Equation (2) and $\overline{\pi}_a$ is the component of $\overline{\pi}$ matching node a. In general, $\overline{\pi}$ describes the fraction of time spent in the various nodes by a random walk on the graph as well as being the invariant distribution of the jump chain. It is computed as:

$$\overline{\pi} = \overline{P}' \overline{\pi}, \tag{5}$$

with

$$\overline{P}_{ii} = 1 - \sum_{i \neq j} \overline{P}_{ij}, \quad \overline{P}_{ij} = \frac{\Lambda_{ij}}{\max_i \omega_i} \; if \; i \neq j \tag{6}$$

As expected, the computed value $\mathbb{E}_a[T_a^+] = 6.750$ is close to $\hat{T}_a^+ \approx 6.830$.

1.3. Problem 1c

This task requires finding the expected hitting time of a particle from o to d on the same graph and transition matrix as before. $100'000$ iterations are again a good compromise between the accuracy of the result and the time needed to finish the cycle. Each time elapsed for the transitions is computed as in Equation (3) and added in the variable *total_time*. Instead, the hitting times are collected in the list *arrive_time*, obtained by the difference between the last value of *transition_times* and the first time when the particle passed in o once having touched d. In addition, the boolean variable *been_in_o* is helpful to start and stop the counting of the hitting times when the particle respectively passes in o and in d. By averaging the different simulation times, the value obtained is $\hat{T}_{o,d} \approx 8.725$.

1.4. Problem 1d

To compare the simulated return time with the theoretical one, the following linear system needs to be solved:

$$(\mathbb{I} - \hat{P})v = \frac{\mathbb{1}}{\hat{\omega}} \tag{7}$$

which has the solution:

$$v_o = \mathbb{E}_o[T_d] = \frac{1}{\omega_o} + \sum_{j \in \mathcal{R}} \hat{P}_{oj} \mathbb{E}_j[T_d] \tag{8}$$

where R is the set of nodes of the network except d, $\hat{P}$ and $\hat{\omega}$ are the corresponding matrix and vector without the entries related to state d. As expected, the obtained value $v_o = 8.786$ is fairly close to the estimate got from the simulations.

1.5. Problem 1e

This point requires interpreting the transition matrix Λ as a weight matrix and to simulate the French-DeGroot dynamics on the graph $\mathcal{G}$ with an arbitrary initial condition $x(0)$. Eventually, it is asked to verify that the initial opinion converges to a consensus. To obtain the result, a random initial opinion is computed. Afterward, a cycle of $100'000$ steps is initiated to simulate the opinion dynamics through the linear averaging dynamics equation:

$$x(t) = Px(t-1) \tag{9}$$

where P is the normalized adjacency matrix. At the end of the simulation, all the initial opinions converge to a consensus. This is justified theoretically as a strongly connected and aperiodic graph that converges to a consensus for $t \to \infty$:

$$\lim_{t \to +\infty} x(t) = \alpha \mathbb{1} \tag{10}$$

where

$$\alpha = \pi x(0) \tag{11}$$

and π is the invariant distribution of the normalized weight matrix P.

Indeed, $\mathcal{G}$ is strongly connected as each of its nodes is reachable from any other node. In addition, the cycles $a \to c \to d \to a$ and $b \to c \to b$ have respectively period equal to 3 and 2. Therefore, $\mathcal{G}$ is also aperiodic since the greatest common divisor of the lengths of its cycles equals one.

1.6. Problem 1f

Given an initial opinion distribution and the corresponding variance, it is asked to simulate the evolution of the opinion dynamics also from the point of view of the variance. This is achieved through a simulation of 1200 steps. By choosing a uniform distribution with range between $a = 0$ and $b = np.random.rand()$, the variance and the mean are expressed as follows:

$$\sigma^2 = \frac{1}{12}(a-b)^2, \quad \mu = \frac{1}{2}(a+b) \tag{12}$$

Each variance is computed on a final opinion distribution obtained through 500 steps-long simulated dynamics of different initial opinion distributions. To obtain the variance of the consensus value by simulation, it is enough to compute the mean of the squared errors. To compare the simulated result with the theoretical one, the variance of the asymptotic consensus value is computed as follows:

$$\sigma_c^2 = \sigma^2 \sum_{i \in G} \pi_i^2 \tag{13}$$

obtained by applying the variance over all the final opinions in the graph, whose value results are similar to the simulated one.

1.7. Problem 1g

At this point, it is asked to observe the behavior of the dynamics by removing the edges (d, a) and (d, c). Since the two edges are removed, node d becomes a sink node. Because of this, it is necessary to add a self-loop to d, to simulate the French-DeGroot dynamics. Adding a self-loop to d is necessary because to compute the diagonal matrix D, it is required that $w_i > 0$. In this way, the constraint is satisfied. The matrix Λ' becomes:

$$\Lambda' = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/3 & 0 & 2/3 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \tag{14}$$

By computing the French-DeGroot dynamics, it is clear that the opinion of each node converges to a consensus value. In particular, by computing $\pi = [0, 0, 0, 0, 1]$, it is possible to observe that only the initial opinion of d influences the consensus value. This is as expected since the consensus value depends on the initial opinions of the trapping components that correspond to node d in this situation. Furthermore, it is reasonable to expect convergence,

because the graph is composed of one aperiodic globally reachable component. Then, considering the initial state of the dynamics composed of random variables with variance σ^2, it is possible to compute the variance of the consensus value. Since the consensus value depends only on the initial opinion of node d, it is reasonable to expect that the variance of the consensus value is similar to the variance of the initial state. To compute the variance of the initial state, it is necessary to determine the distribution of the initial opinions. By choosing a uniform distribution with a range between $a = 0$ and $b = np.random.rand()$, the variance and the mean are expressed as in Equation (12).
The formula used to compute the theoretical variance of the consensus state is the one reported in Equation (13).
The value of the variance of the consensus state obtained by simulating is given by the mean of the squared errors of every simulation. Since the obtained value is similar to the theoretical one, the expectations are met.

1.8. Problem 1h

At this point, it is required to remove the edges (c, b) and (d, a) of the original graph. The matrix Λ' becomes:

$$\Lambda' = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 2/5 & 1/5 & 0 & 0 \\ 0 & 0 & 3/4 & 1/4 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 2/3 \\ 0 & 0 & 0 & 1/3 & 0 \end{pmatrix} \end{array} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \tag{15}$$

By computing the French-DeGroot dynamics as in the previous point, it is possible to observe that there is no convergence. This confirms the expectations: since the set of nodes c and d is globally reachable and periodic, it is reasonable to expect that the French-DeGroot dynamics do not converge. This can be also motivated by the fact that another requirement is not respected: the new graph obtained is not strongly connected anymore, since from node c or node d is not possible to reach the other nodes. As in the previous point, the variance of the consensus value is computed.

2. Problem 2

In this problem, the same graph of Problem 1 with the same rate matrix Λ is considered. In this case, the aim is to simulate many particles moving around in the network in continuous time. In particular, there will be considered two different points of view:

- *particle perspective*: it follows the particles;
- *node perspective*: the movements of the particles are observed from the point of view of the nodes

2.1. Problem 2a

In this point, 100 particles moving around the network in continuous time are considered. Assuming that they start

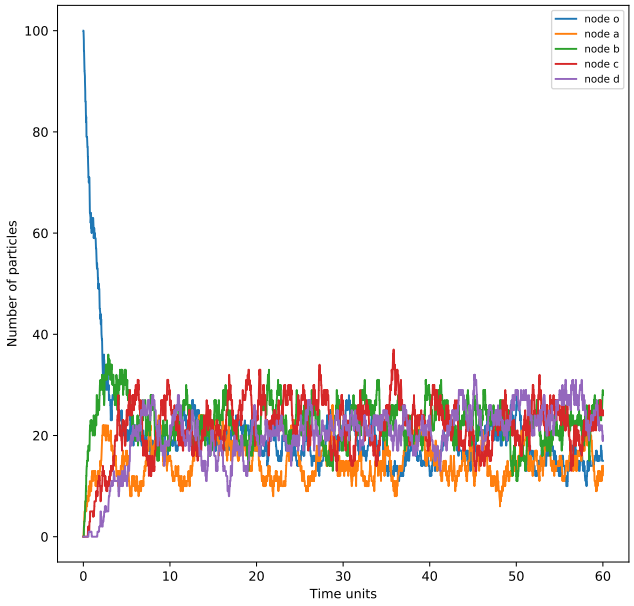


Figure 2: Number of particles in each node in relation to the time

in node a, it is required to compute the average time for a particle to return to node a. Therefore, it is considered the *particle perspective*. The method *getAverageTime(node1,simulations)* computes the average time for a particle to move from $node1$ (i.e. the starting node) and return to it, given a certain number of simulations. For each simulation, it computes the average time needed by 100 particles to return to $node1$. Then, after computing this for every simulation, the total time (the sum of the average time of each simulation) is divided by the number of simulations: in this way, the average time for a particle to return to $node1$ is computed. In this case, the starting node $node1$ is a, and the average time for a particle to return to node a is $\hat{T}_a^+ \approx 6.769$. It is interesting to notice that the result is similar to the theoretical return time obtained in Problem 1b.

2.2. Problem 2b

At this point, the *node perspective* is considered. It is required to calculate the average number of particles in the different nodes at the end of the simulation, assuming 100 particles start in node o and that the system is simulated for 60-time units. To do this, the method *getNumberParticlesPlot(node1,simulations)* is defined. Firstly, an array of particles is defined: the element at the position of the starting node ($node1$ that in this case is node o) contains 100 particles. Then, for each time unit, the array is updated to the position of each particle: for instance, if a particle

goes from o to a, the element in position 0 is updated by subtracting its value of one unit, while the value of the element in position 1 is incremented of 1 unit. Thus, the total number of particles (obtained by summing the array of particles for each simulation) is divided by the number of simulations. In this way, the vector containing the average number of particles in each node is:

$$avgParticles \approx [18.651,\ 14.819,\ 22.19,\ 22.036,\ 22.304]$$

During the iterations, the values of the first simulation are collected to plot the number of particles in each node during the simulation time (Figure 2). To compare the simulation result reported above with the stationary distribution of the continuous-time random walk followed by the single particles, it is necessary to divide each value of the vector reported above by the total average number of particles, obtaining:

$$\hat{\bar{\pi}} \approx [0.186,\ 0.148,\ 0.221,\ 0.220,\ 0.223]$$

Having a stationary distribution

$$\bar{\pi} \approx [0.185,\ 0.148,\ 0.222,\ 0.222,\ 0.222]$$

it is possible to notice that the two vectors are quite similar, and since the differences are very near zero, the two arrays contain similar values.

3. Problem 3

The objective of this problem is to study how different particles affect each other in a continuous time when moving around in the open network of Figure 3, according to the transition rate matrix Λ_{open}.

$$\Lambda_{open} = \begin{array}{c} \begin{matrix} o & a & b & c & d \end{matrix} \\ \begin{pmatrix} 0 & 3/4 & 3/8 & 0 & 0 \\ 0 & 0 & 1/4 & 1/4 & 2/4 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} o \\ a \\ b \\ c \\ d \end{matrix} \end{array} \tag{16}$$

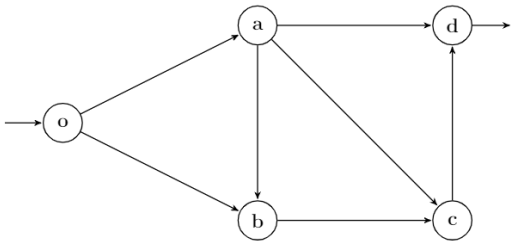


Figure 3: Open network with transition matrix in (16)

To simulate the entrance of particles in the system at node o according to a Poisson process with input rate $\lambda = 1$, a new node o' connected to o is added, such that the rate for its associated Poisson clock $\omega_{o'}$ equals the input rate: $\omega_{o'} = \lambda$. Based on the definition of Poisson clock's rate w reported in Equation (17), the Matrix (16) determines

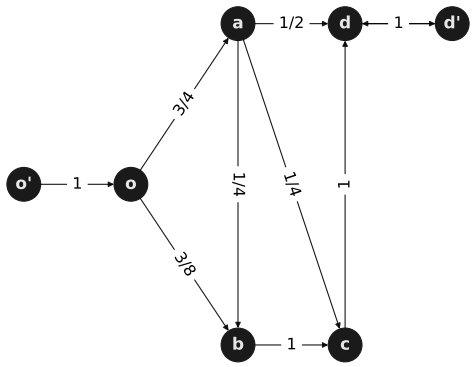


Figure 4: Network with transition matrix in (18)

$\omega_d = 0$, which means that once the process is in d, it remains in d forever, because there is not a node to which this node can send particles.

$$w = \Lambda \mathbb{1}, \quad D = diag(\omega), \quad P = D^{-1}\Lambda \tag{17}$$

To change this not-suitable representation where the matrix P in Equation (17) is not well defined, it is possible to either create a self-loop $\Lambda_{ii} > 0$ with $\omega_d = 2$ or equivalently add another node d' connected to node d to simulate a self-loop. Opting for the latter option, $w_d = w_{d'} = 1$ is imposed. All these changes bring about the matrix Λ used for the next simulations, which defines the network in Figure 4.

$$\Lambda = \begin{array}{c} \begin{matrix} o' & o & a & b & c & d & d' \end{matrix} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3/4 & 3/8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 1/4 & 2/4 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{matrix} o' \\ o \\ a \\ b \\ c \\ d \\ d' \end{matrix} \end{array} \tag{18}$$

Two different scenarios are simulated for 60-time units which differ by what rate the nodes will pass along particles:

- *proportional rate*: each node i passes along particles according to a Poisson process with rate $\omega_i N_i(t)$;
- *fixed rate*: each node i passes along particles with a fixed rate ω_i.

Note that the number of particles inside the source o' is always kept constant, as well as in node d'. In particular, at every time instant, the number of particles in o' equals the input rate: whenever this node spreads a particle into the system, its number of particles is not lowered, differently from what happens for the remaining nodes. Similarly, whenever the node d sends a particle to d', the number of particles in d' never updates.
To model this Continuous Time Markov Chain, it is chosen the approach which consists in equipping each node i with its Poisson clock with the rate ω_i. When the clock of a node i ticks, a particle in this node i jumps into a neighbor j with probability P_{ij} defined in Equation (2), that is an equivalent formulation of Equation (17).

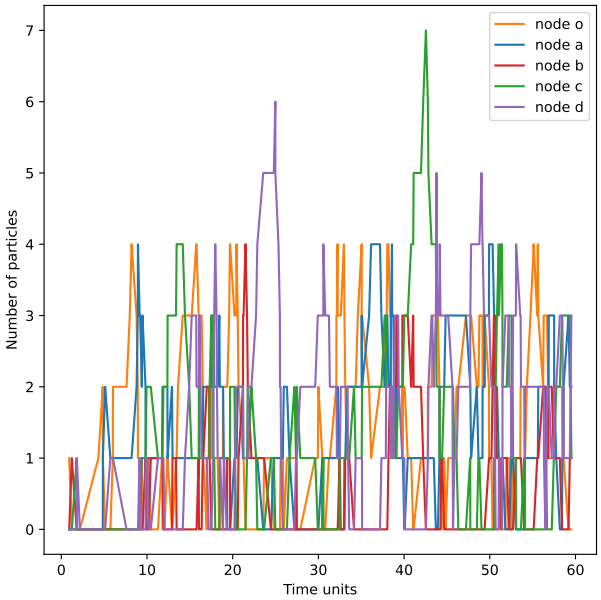


Figure 5: Evolution of the particles distribution: $\lambda = 1$, proportional rate

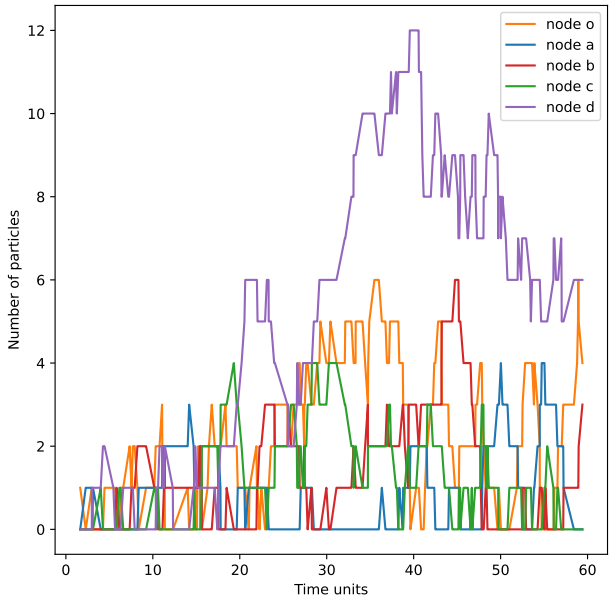


Figure 6: Evolution of the particles distribution: $\lambda = 1$, fixed rate

3.1. Problem 3a

The aim is to simulate the system with nodes having rate *proportional* to their number of particles and to analyze the evolution of the number of particles in each node over time. The random time that node i waits for its next transition, i.e. the waiting time for node i to send a particle, is defined as in Equation 3 with $r = \omega_i N_i(t)$ and is computed whenever a transition happens in node i. Each scheduled time for transitions of node i (i.e. the time instants when i is expected to send particles) is obtained by summing the time instant related to the last tick of node i's Poisson clock with the waiting time of i for such transition. Until the deadline of 60 temporal units, this computation is repeated in all the nodes to individuate the first next transition that is going to happen:

$$t_{min} = \min_{i \in \mathcal{V}} (t_{tick,i} + t_{next,i}) \tag{19}$$

The particle in node i moves to a node j randomly chosen according to the i-th row of the transition matrix P: the particle removed from node i is then added to the already existing particles (if there are) of node j.
When the input rate equals one ($\lambda = 1$), the distribution of particles among the nodes appears to randomly change during the simulation, so the particles do not get stuck in a specific node for a long time and their flow is continuous as shown in Figure 5.
The other requirement of this problem is to find the largest input rate that the system can handle without blowing up. The more the input rate is increased ($\lambda > 1$), the more the particles enter the network at a higher rate, which means that the probability for node o' of sending a particle to o increases with the increase of λ. The distributions of particles over time for different values of λ are shown in Figures (7, 8, 9, 10, 11, 12). As $\lambda \to \infty$, the number of particles never blows up, but it is consistent with the input rate λ. This can be explained by taking into account the Equation (3): with a higher input rate, more particles are spread into the network with the consequence that the waiting random times for nodes are lower than before. Then, more transitions are performed in the same amount of 60-time units and this proves the ability of the inner nodes to adapt to the rate of the source node.

3.2. Problem 3b

The tasks of this problem are the same as Problem 3a, but considering the system with nodes having *fixed* rate, independent from the number of particles inside.
The simulation is done with the same logic as before. The only difference is about the definition of the random time that node i waits for its next transition, due to its fixed rate, so now Equation (3) is used with $r = \omega_i$.
With $\lambda = 1$, the behavior of the system is quite comparable with the simulation made with a proportional rate, indeed the system manages to handle the particles, as reported in Figure 6. Particles have a bit more propensity for getting stuck in a node with respect to Figure 5 but the system is fast enough to overcome this situation and get a new equilibrium between the number of particles that come into and go out of

the network. A possible interpretation of this behavior lies in the fact that since now $t_{next,i}$ is not influenced by $N_i(t)$, then its value is bigger than in Exercise 3a: this results in bigger waiting times for transitions, so particles go through fewer movements.
Since for each node $t_{next,i}$ is a constant, it is expected that the more the input rate grows, the higher the number of particles inside the system. Therefore, the nodes do not have the capacity of adapting depending on the input rate, resulting in a way that the number of particles that enter the system in o is higher than the number of particles that exit the system from d. As shown in Figures (13, 14, 15, 16, 17, 18), an input rate equal to 2 is enough to make the system blow up. The more λ increases, the more the number of particles in node o differs by some orders of magnitude with respect to the other nodes.

Appendix

Figure 7: $\lambda = 10$

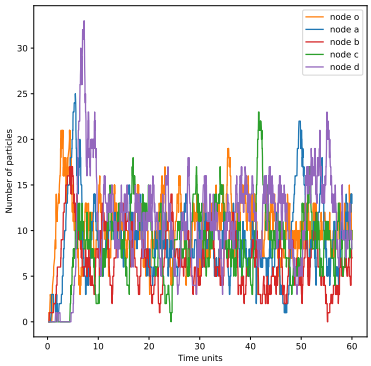


Figure 8: $\lambda = 50$

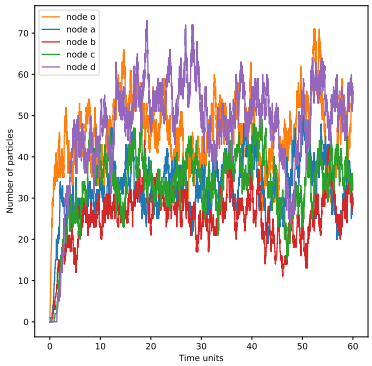


Figure 9: $\lambda = 100$

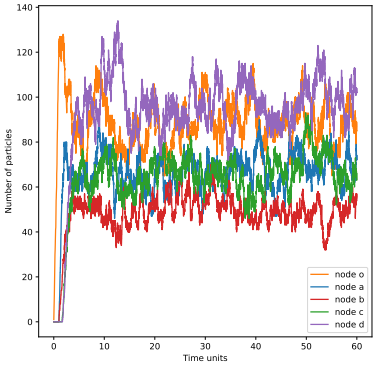


Figure 10: $\lambda = 500$

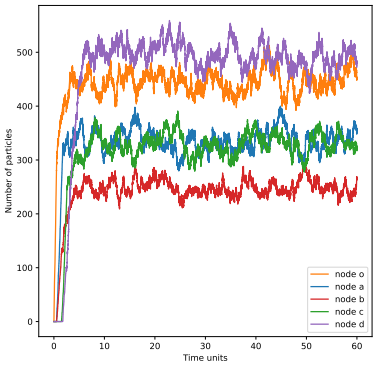


Figure 11: $\lambda = 1000$

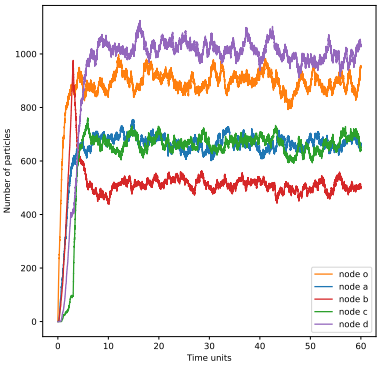
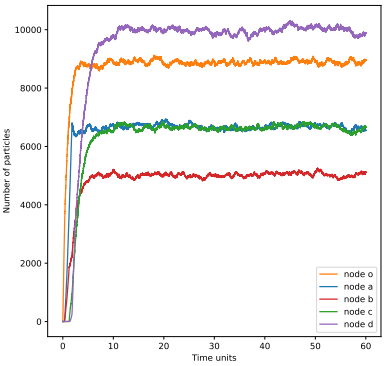


Figure 12: $\lambda = 10000$



Evolution of the particle distribution in the nodes for different values of the input rate $\lambda = (10, 50, 100, 500, 1000, 10000)$ with proportional rate.

Figure 13: $\lambda = 2$

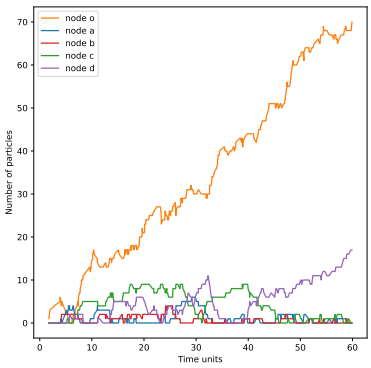


Figure 14: $\lambda = 3$

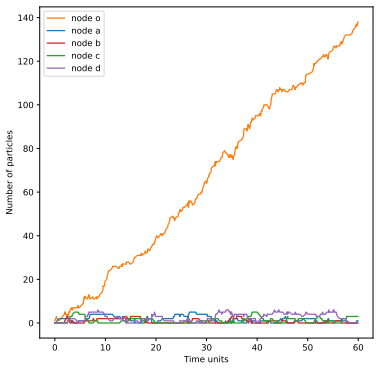


Figure 15: $\lambda = 4$

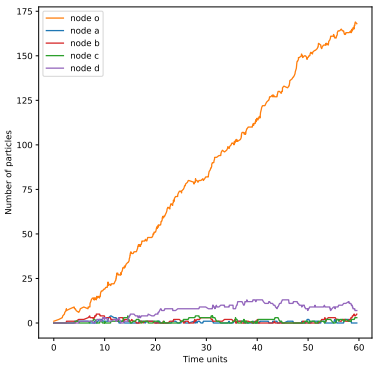


Figure 16: $\lambda = 5$

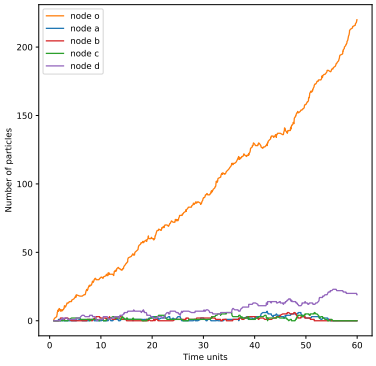


Figure 17: $\lambda = 25$

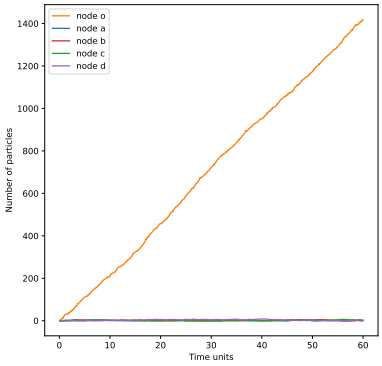
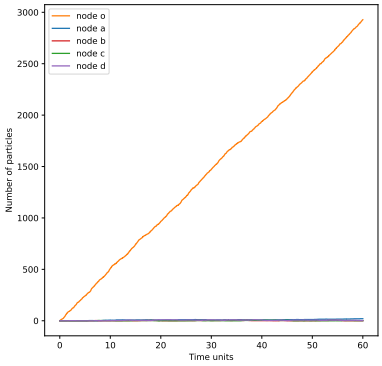


Figure 18: $\lambda = 50$



Evolution of the particle distribution in the nodes for different values of the input rate $\lambda = (2, 3, 4, 5, 25, 50)$ with a fixed rate.