Homework #6

AA 597: Networked Dynamics Systems
Prof. Mehran Mesbahi
Due: Mar 1, 2024 11:59pm
Soowhan Yi

All the codes are available at the end of the documents or here. https://github.com/SoowhanYi94/ME597

P1. 3.8 Consider the uniformly delayed agreement dynamics over a weighted graph, specified as

$$\dot{x}_i(t) = \sum_{j \in N(i)} w_{ij}(x_j(t-\tau) - x_i(t-\tau))$$

for some $\tau > 0$ and $i = 1, \cdots n$. Show that this delayed protocol is stable if

$$\tau < \frac{\pi}{2\lambda_n(G)}$$

where $\lambda_n(G)$ is the largest eigenvalue of the corresponding weighted Laplacian. Conclude that, for the delayed agreement protocol, there is a tradeoff between faster convergence rate and tolerance to uniform delays on the information-exchange links.
As we have sum of inputs ($\sum_{i \in V(i)} \dot{u}_i(t) = 0$) equal to 0, we know that the average of agent values are invariant set of quantities, therefore reaching average-consensus. Now, the Laplacian transform of this agreement protocol can be written as

$$X(s) = (sI + L(s))^{-1}x(0)$$

where the $L(s)$ is Laplacian transform of weighted laplacian matrix. Since we know the guaranteed stability of the response, the transfer function is guaranteed to be stable. Also, we know that the transfer function involving time delay would have the time delay term $e^{-\tau s}$ in front. Therefore, the transfer function has to be stable, and the poles should be on the left hand side of the plane.

$$\begin{aligned} G(s) &= (sI + e^{-\tau s}L(s))^{-1} \\ Z(s) &= sI + e^{-\tau s}L(s) = 0 = sI + e^{-\tau s}Q(s)\Lambda Q(s)^{-1} = 0 \\ & Q(s)^{-1}sIQ(s) + e^{-\tau s}\Lambda = 0 \\ \therefore & sI + e^{-\tau s}\Lambda = 0 \end{aligned}$$

Using frequency domain analysis, we can substitute s with jw. Therefore,

$$s = -e^{-\tau s}\Lambda = -e^{-\tau jw}\Lambda = -(\cos(-\tau w) + j\sin(-\tau w))\Lambda = -(\cos(\tau w) - j\sin(\tau w))\Lambda$$

. Since the real part of s should be negative, $\cos(-\tau w) = \cos(\tau w) \geq 0$. Also $\Lambda = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n)$, where $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Therefore, substituting jw into $sI + e^{-\tau s}\Lambda = 0$, yields

$$\begin{aligned} jw + e^{-\tau jw}\lambda_n = 0 \\ -jw + e^{\tau jw}\lambda_n = 0 \end{aligned}$$

. Multiplying those two equations yields,

$$\begin{aligned} &(jw + e^{-\tau jw}\lambda_n)(-jw + e^{\tau jw}\lambda_n) = w^2 - jwe^{-\tau jw}\lambda_n + jwe^{\tau jw}\lambda_n + \lambda_n^2 \\ &= w^2 + \lambda_n^2 - jw\lambda_n(e^{-\tau jw} - e^{\tau jw}) = w^2 + \lambda_n^2 - jw\lambda_n(-2jsin(\tau w)) = (w - \lambda_n)^2 + 2w\lambda_n - 2w\lambda_n(sin(\tau w)) \\ &\therefore (w - \lambda_n)^2 + 2w\lambda_n(1 - sin(\tau w)) = 0 \end{aligned}$$

. Sine we know $w \geq 0, \tau \geq 0$ and $\lambda_n \geq 0$, $w = \lambda_n$ and $1 = sin(w\tau)$, and therefore, the smallest τ that satisfies above equations is

$$\tau = \frac{\pi}{2\lambda_n}$$

. Therefore if $\tau \leq \frac{\pi}{2\lambda_n}$, the delayed agreement protocol is stable.
Also according to the Gersgorian theorum, $\lambda_n \leq 2d_{max}(G)$, where $d_{max}(G)$ is the maximum out degree in a node of the graph. Therefore $\tau \leq \frac{\pi}{4d_{max}(G)} \leq \frac{\pi}{2\lambda_n}$. So we can infer that the bigger the time delay, we are allowed to have less maximum out degree in a node of the graph, and causing a slower convergence rate. Therefore there is a tradeoff between faster convergence rate and the tolerance to uniform time delay.

P2. 3.9 A matrix M is called essentially non-negative if there exists a sufficiently large μ such that $M + \mu I$ is non-negative, that is, all its entries are non-negative. Show that e^{tM} for an essentially non-negative matrix M is non-negative when $t \geq 0$.
If M is essentially non-negative, then there exists a sufficiently large μ that $M + \mu I$ is non-negative. Let $N = M + \mu I \geq 0$. Then

$$e^{tM} = e^{t(N-\mu I)} = e^{tN}e^{-\mu tI}$$

Since $t \geq 0$, $e^{tN} \geq 0$, and $e^{-\mu t} \geq 0$, $e^{tN}e^{-\mu tI} = e^{tM} \geq 0$

P3. 3.16 Consider a network of n processors, where each processor has been given an initial computational load to process. However, before the actual processing occurs, the processors go through an initialization phase, where they exchange certain fractions of their loads with their neighbors in the network. Specifically, during this phase, processor i adopts the load-update protocol

$$p_i(k+1) = p_i(k) - \sum_{j \in N(i)} \omega_{ij}(p_i(k) - p_j(k))$$

for $k = 0, 1, \cdots$, that is, it sends a fraction ω_{ij} of its load imbalance with its neighbors to each of them. What is the necessary and sufficient condition on the weights ω_{ij} in above equation such that this initialization phase converges to a balanced load for all processors when the network is (1) a path graph, (2) cycle graph (3) a star graph?
Lemma 8.1 (Necessary Condition)
Let the weight matrix be $W = diag(\omega_{i,j})$ for all pairs of $(i, j) \in E$. Then it is necessary to have connected graph, and to have the absolute value of the largest eigenvalue of weighted laplacian matrix less than 2 for the networks to converge to balanced load for all processors.

$$p_i(k+1) = p_i(k) - \sum_{j \in N(i)} \omega_{ij}(p_i(k) - p_j(k))$$
$$p(k+1) = p(k) - L_\omega(G)p(k) = (I - L_\omega)(G)$$

Let $M_\omega(G) = (I - L_\omega(G))$, then

$$p(k+1) = M_\omega(G)p(k) = M_\omega(G)^k p(0)$$

where $\Lambda = diag(\lambda_1, \lambda_2, \cdots, \lambda_n)$ and $\lambda_i \forall i \in n$ are eigenvalues of $M_\omega(G)$. So, for eigenvector x,

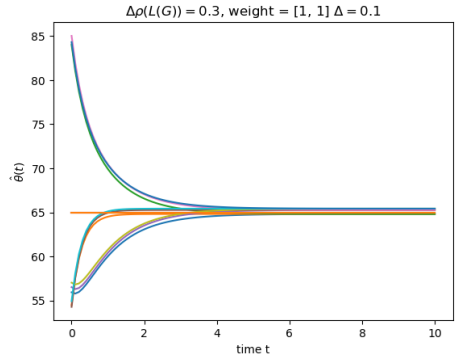
$$M_\omega(G)x = (I - L_\omega(G))x = \lambda(M_\omega(G))x$$
$$(I - L_\omega(G) - \lambda(M_\omega(G))I)x = 0, \text{where } x \neq 0$$
$$I - L_\omega(G) - \lambda(M_\omega(G))I = 0$$
$$Q^{-1}(I - L_\omega(G) - \lambda(M_\omega(G))I)Q = 0 \text{ where Q is a eigenvector matrix}[q_1, q_2, \cdots, q_n]$$
$$\therefore \lambda(M_\omega(G)) = I - \lambda(L_\omega(G)), \Lambda(M_\omega(G)) = 1 - \Lambda(L_\omega(G))$$
$$\lim_{k\to\infty} p(k+1) = \lim_{k\to\infty} \Lambda^k p(0) = \lim_{k\to\infty} (1 - \Lambda(L_\omega(G)))^k p(0)$$

Therefore $|1 - \Lambda(L_\omega(G))|$ has to be less than 1 in order for the network to converge, and the $\Lambda(L_\omega(G)) < 2$ has to satisfy. Finally the absolute value of largest eigenvalue of the weighted laplacian matrix of the graph has to be less than 2, then it would converge to the balanced load of all processors. Delta in original equation is omitted because it is 1 in this problem.
Corollary 8.2 (Sufficient Condition)
If the weights are determined as the inverse of maximum of sum of weights adjacent to the node associated with the edge, it is guaranteed to have maximum eigenvalue of weighted laplacian matrix less than 2.

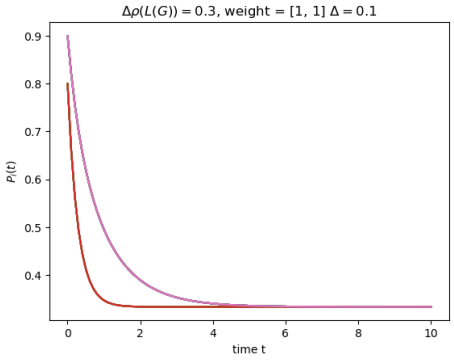
$$W_{jj} = (max(d_\omega(u), d_\omega(v)))^{-1}$$

where edge uv is the jth edge, and W_{jj} is the weight associated witht the edge.
So, in (1) a path graph, those weights would be larger going into the center of the path graph. (2) cycle graph and (3) a star graph would have the same weights in all of the edges because their nodes' that forms the edge degrees are same.

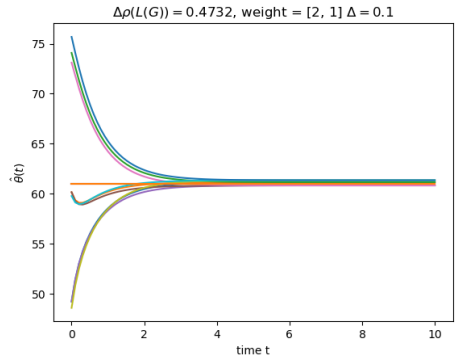
P4. 8.1 Let $H_i, i = 1, 2, 3$, be the rows of the 3×3 identity matrix in the observation scheme $z_i = H_i\ x + v_i$ for a three-node sensor network, observing state $x \in R^3$. It is assumed that the nodes form a path graph and that v_i is a zero-mean, unit variance, Gaussian noise. Choose the weighting matrix W (8.8) and the step size Δ in (8.20) - (8.21), conforming to the condition (8.14). Experiment with the selection of the weights for a given value of Δ and their effect on the convergence properties of the distributed least square estimation (8.20) - (8.21).



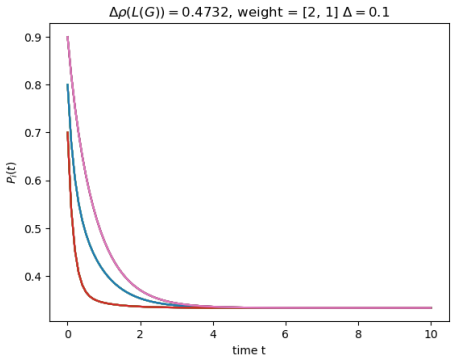
(a) $\Delta\rho(L(G)) = 0.3, weight = [1, 1], \Delta = 0.1$



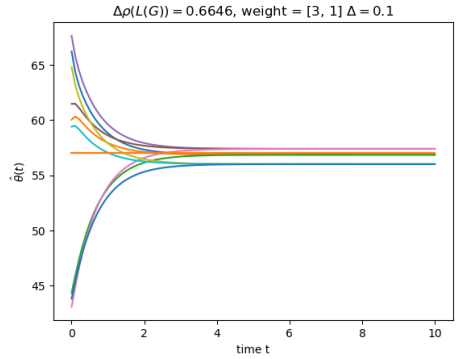
(b) $\Delta\rho(L(G)) = 0.3, weight = [1, 1], \Delta = 0.1$



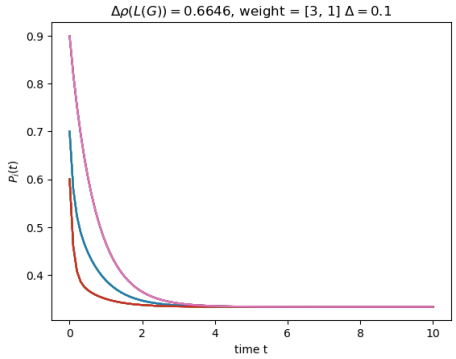
(c) $\Delta\rho(L(G)) = 0.4732, weight = [2, 1], \Delta = 0.1$



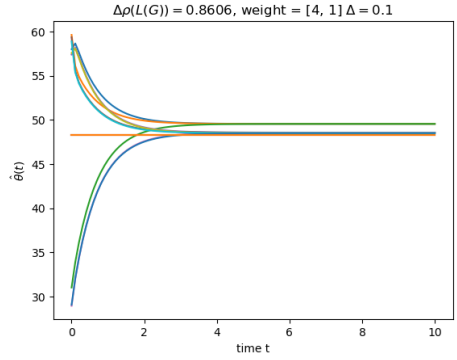
(d) $\Delta\rho(L(G)) = 0.4732, weight = [2, 1], \Delta = 0.1$



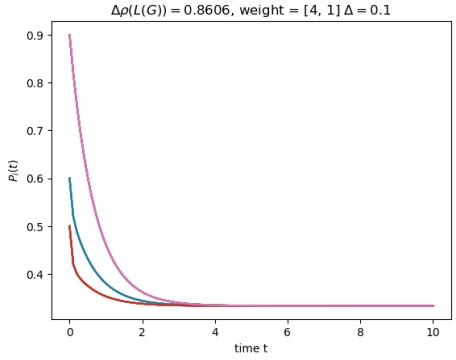
(a) $\Delta\rho(L(G)) = 0.6646, weight = [3, 1], \Delta = 0.1$



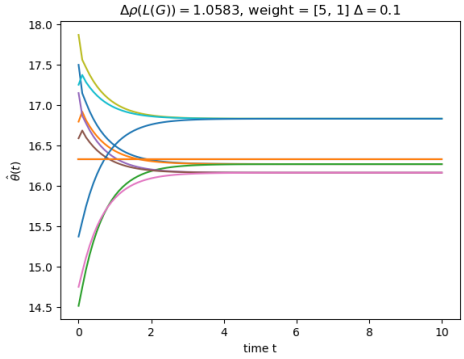
(b) $\Delta\rho(L(G)) = 0.6646, weight = [3, 1], \Delta = 0.1$



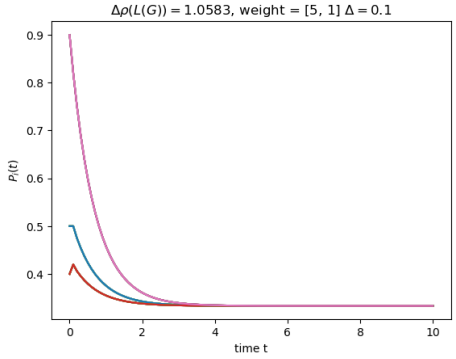
(c) $\Delta\rho(L(G)) = 0.8606, weight = [4, 1], \Delta = 0.1$



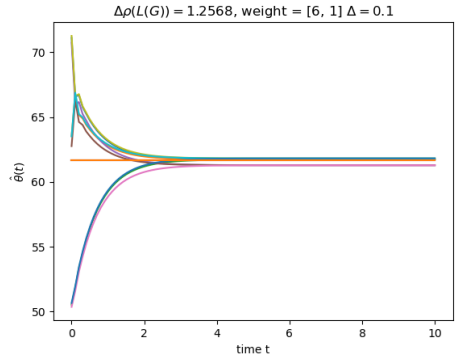
(d) $\Delta\rho(L(G)) = 0.8606, weight = [4, 1], \Delta = 0.1$



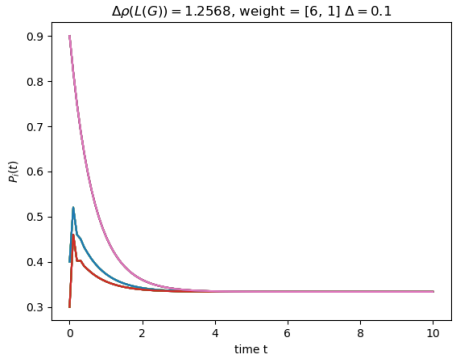
(e) $\Delta\rho(L(G)) = 1.0583, weight = [5, 1], \Delta = 0.1$



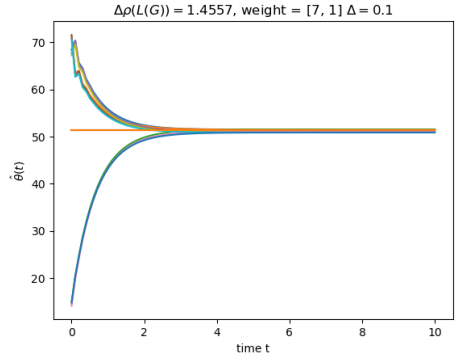
(f) $\Delta\rho(L(G)) = 1.0583, weight = [5, 1], \Delta = 0.1$



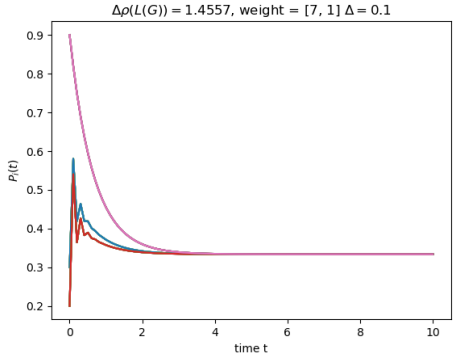
(a) $\Delta\rho(L(G)) = 1.2568, weight = [6, 1], \Delta = 0.1$



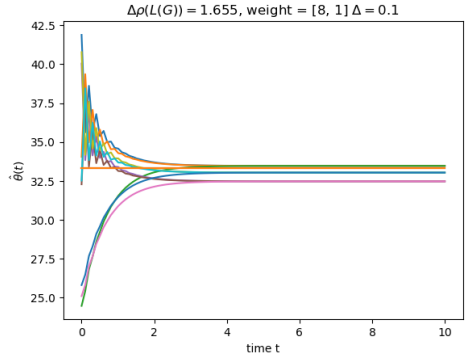
(b) $\Delta\rho(L(G)) = 1.2568, weight = [6, 1], \Delta = 0.1$



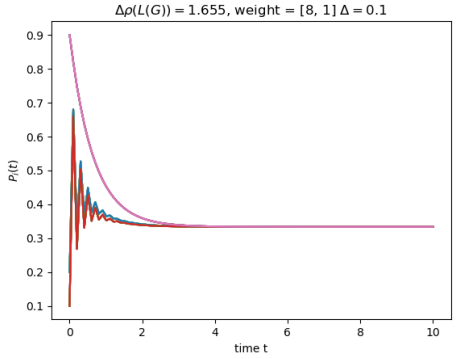
(c) $\Delta\rho(L(G)) = 1.4557, weight = [7, 1], \Delta = 0.1$



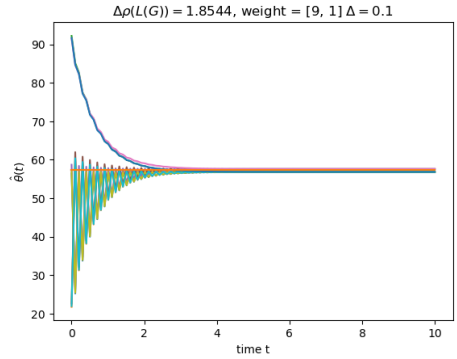
(d) $\Delta\rho(L(G)) = 1.4557, weight = [7, 1], \Delta = 0.1$



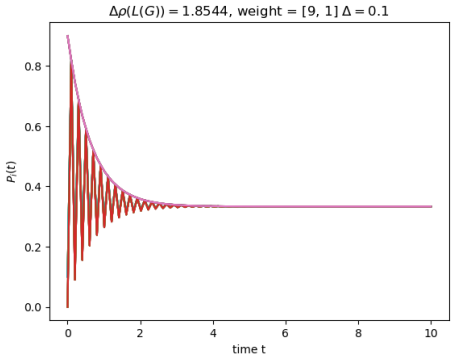
(e) $\Delta\rho(L(G)) = 1.655, weight = [8, 1], \Delta = 0.1$



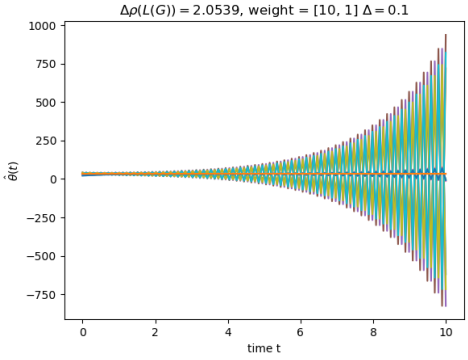
(f) $\Delta\rho(L(G)) = 1.655, weight = [8, 1], \Delta = 0.1$



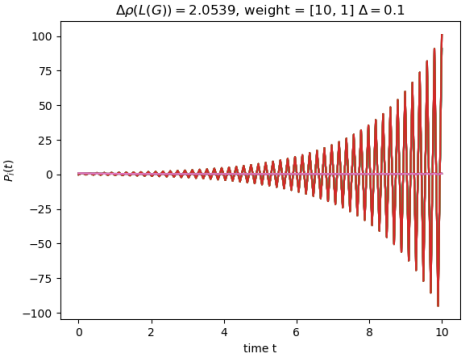
(a) $\Delta\rho(L(G)) = 1.8544, weight = [9, 1], \Delta = 0.1$



(b) $\Delta\rho(L(G)) = 1.8544, weight = [9, 1], \Delta = 0.1$



(c) $\Delta\rho(L(G)) = 2.053, weight = [10, 1], \Delta = 0.1$



(d) $\Delta\rho(L(G)) = 2.053, weight = [10, 1], \Delta = 0.1$

As we can see from those experiments, if there is significant disproportion in weights, then $\Delta\rho(L(G))$ starts increasing and their values starts to diverge as it gets bigger than 2.

P5. 10.8 If the network is connected, then the followers will end up (asymptotically) at

$$x_f = -A_f^{-1} B_f x_l$$

given the static leader positions x_l . Show that each component of x_f above is in fact given by a convex combination of the components of x_l .
By definition, a convex combination of the components of x_l is $\sum_{i \in l} \alpha_i x_{l_i}$ where $\sum_{i \in l} \alpha_i = 1$ and $\alpha_i \in R_+$. Since we know that the connected network has positive definite A_f, it is invertible and inverted matrix would also be positive definite. Now we just need to show if all the components in B_f matrix is negative or 0, and which are. As we have $D = [D_f; D_l]$, $B_f = D_f D_l^T$, therefore each entries of B_f will be negative if and only if the follower corresponding to that entry is connected to some leaders. I think the each entries might be the negative of number of connected leaders to that corresponding follower. If that specific follower is not connected to the leaders, then the entries would be 0. Therefore, for connected graphs, at least one of the follower has connection with the leader, and we can conclude that each entries of B_f is either negative or 0. Finally, by definition, each component of x_f is convex combination of the components of x_l, because A_f are positive definite, and each entries of B_f is either negative or 0.

P6. 10.9 Consider the linear-quadratic optimal control problem

$$\min_u \int_0^\infty (u(t)^T R u(t) + x(t)^T Q x(t)) dt$$

where the matrices Q and R are, respectively, positive semidefinite and positive definite, and

$$\dot{x}(t) = -A_f x(t) - B_f u(t)$$

corresponds to a controllable network.
Now, just because the followers execute a decentralized control strategy it does not follow that the leaders' optimal control strategy will be decentralized. Determine if this is the case for this infinite horizon optimal control problem.
For linear quadratic optimal problem, the feedback control law is given by

$$u = -R^{-1} B^T P_+ x$$

where this P_+ is the positive definite solution of the algebraic Riccati equation(ARE).

$$A^T P + PA - PBR^{-1} B^T P + Q = 0$$
$$\dot{x}(t) = (A - BR^{-1} B^T P_+) x(t)$$

Therefore we need to use the whole or the global information of x in order to get the control law for the network. So the leaders can not have decentralized optimal control strategy because it requires global information of all the nodes in the network.

Code P4

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

def main():
    num = 3
    graph = nx.path_graph(num)
    delta = .1
    for j in range(10):
```

```
        weight = {(0,1): 1+j, (1,2): 1}
        iteration = 100 #number of iteration to simulate limit
        time_lim = iteration *delta
        nx.set_edge_attributes(graph, weight, 'weight')
        max_eigen_val = np.max(nx.laplacian_spectrum(graph, weight='weight'))
        L_w_G = nx.laplacian_matrix(graph, weight='weight').toarray()
        del_max_eig = round(delta*max_eigen_val,4)
        M_G = np.eye(num) - delta*L_w_G
        true_value = np.random.randint(0, 100, size = num)
        avg_true_value = np.average(true_value)
        H =  np.eye(num)
        estimates = []
        P_mat = []
        t = np.linspace(0, time_lim, iteration)
        for i in range(num):
            rand_noise = np.random.normal(0,1, size = num)
            z = H@true_value + rand_noise
            P = np.eye(9)
            for k in range(iteration):
                z = M_G @z
                P = np.kron(M_G,np.eye(3)) @P
                estimates.append(z)
                P_mat.append(P)
            plt.figure(2*j)
            plt.plot(t, estimates)
            plt.plot(t, [avg_true_value]*iteration)
            plt.xlabel("time t")
            plt.ylabel(r"$\hat \theta (t)$")
            plt.title(r"$\Delta  \rho (L(G)) =" f"{del_max_eig}$" ", weight = " f"{list(weight.values())}
            estimates = []
            plt.figure(2*j+1)
            P_mat_plot = []
            for k in range(iteration):
                P_mat_k = P_mat[k]
                P_mat_plot.append(np.array(np.diag(P_mat_k)))
            plt.plot(t, P_mat_plot)
            plt.xlabel("time t")
            plt.ylabel(f"$P_i (t)$")
            plt.title(r"$\Delta  \rho (L(G)) =" f"{del_max_eig}$" ", weight = " f"{list(weight.values())}
            P_mat = []
        plt.show()

if __name__ == "__main__":
    main()
```