Noise-Driven Excursions from Consensus: A Theoretical Review and Simulation for Simple Graphs with Common Edge Weights

Axel Haaker
Department of Mechanical Engineering
University of California at Santa Barbara
Adviser: Prof. Francesco Bullo

December 9, 2016

Abstract

In this report, a review of fundamental results from Markov theory is provided to motivate the discussion of a linear averaging process, the consensus iteration. The consensus iteration, ubiquitous in distributed control algorithms, is introduced and its behavior is analyzed using graph theory and several key theorems for ergodic Markov chains. In particular, necessary and sufficient conditions for convergence to consensus are provided. Noise is later added to the iteration, and it is shown that the convergence property is destroyed as the additive noise continually drives excursions from consensus. The size of the individual deviations from consensus tend to meander about zero, but converge in the mean-square sense. A theorem, from [Xiao et al., 2007], relating the limit of the mean-square deviation from consensus to the spectrum of a symmetric transition matrix for the noiseless iteration is reproduced here along with its proof, which is drawn out in detail. A generalization of this theorem due to [Jadbabaie and Olshevsky, 2015], is provided in the following section. The mean-square deviation, or disagreement, is computed for a set of simple graphs and common edge weights, together forming a set of reversible matrices. The MATLAB code used to run the simulations is included in the appendices, and may be of some use to others interested in Markov chains, graph theory, and consensus protocols.

Contents

Introduction

A popular problem in dynamical systems and control is that of developing an algorithm that allows a set of devices to synchronize themselves, i.e., reach consensus, with respect to some measured quantity. In application, limits imposed on system specifications such as power consumption, computational power, memory, etc. limit the amount of, and speed at which, any single device can receive, process, and transmit data. In effect, this restricts a device's awareness of the system as a whole, the sphere of influence of the device, the time it takes for information to diffuse throughout the system, and whether or not each device executes the same number of measurement updates in a given time period.

In that many of these problems have roots in consensus formation [Jadbabaie and Olshevsky, 2015], it is unsurprising that the consensus iteration plays a central role in many of them. In most early models, the iteration was assumed noiseless, and the analysis was greatly simplified. In reality, measurements are always subject to a degree of error, and so it is important to understand the behavior of the consensus iteration in the presence of additive noise.

It is assumed that the reader has a background in matrix analysis, linear systems, network analysis, and has had an introduction to probability and statistics at some point. With these assumptions, it will be relatively simple to understand the following introduction to Markov chains, the main result of [Jadbabaie and Olshevsky, 2015], DeGroot processes subject to noise, and the implementation of these items.

1 Elements of probability theory and statistics

The reader is assumed to have some familiarity with probability theory and statistics, so no attempt will be made to introduce or prove the following basic results and principles, taken from [Montgomery and Runger, 2007]. Rather, this section will simply serve as a corral for concepts employed throughout the rest of the paper. For further details, see [Montgomery and Runger, 2007], or [Grinstead and Snell, 2006].

1.1 Basic probability theorems

For a discrete sample space, the probability of an event E, denoted as $\mathrm{P}(E)$, equals the sum of the probabilities of the outcomes in E.

For two events A and B, the probability of the joint event $A \cup B$ is given by $\mathrm{P}(A \cup B) = \mathrm{P}(A) + \mathrm{P}(B) - \mathrm{P}(A \cap B)$. If the events are mutually exclusive, then $A \cup B = \emptyset$ and $\mathrm{P}(A \cap B) = 0$. This can be easily generalized to unions of greater than two events.

The conditional probability of an event B given an event A, denoted as $\mathrm{P}(B|A)$, is given by

$$\mathrm{P}(B|A) = \frac{\mathrm{P}(A \cap B)}{\mathrm{P}(A)},$$

for $\mathrm{P}(A) > 0$. Reversing the roles of A and B in this equation, and solving each for $\mathrm{P}(A \cap B)$ gives what is often referred to as the *multiplication rule for probabilities*:

$$\mathrm{P}(A \cap B) = \mathrm{P}(B|A)\mathrm{P}(A) = \mathrm{P}(A|B)\mathrm{P}(B).$$

1.2 Statistics

Definition 1.1. *A random variable is a real-valued function whose range is the sample space of a random experiment. If additionally, the range of the function is finite, or countably infinite, then it is said to be a discrete random variable [Montgomery and Runger, 2007].*

For a discrete random variable, X, the *probability distribution* of X is a description of the probability of each possible value of X.

Definition 1.2. *For a discrete random variable X with possible values x_1, x_2, ..., x_n, a probability mass function is a function such that*

(i) $f(x_i) \geq 0 \ \forall i$

(ii) $\sum_{i=1}^{n} f(x_i) = 1$

(iii) $f(x_i) = \mathrm{P}(X = x_i) \ \forall i$

Definition 1.3. *The mean or expected value of a discrete random variable X, denoted by μ or $\mathbb{E}(X)$, is*

$$\mu = \mathbb{E}(X) = \sum_x x f(x).$$

Theorem 1.1. *Let c be an arbitrary constant, and let X and Y be discrete random variables. The expected value function has the following properties:*

(i) (Expected value of a constant) $\mathbb{E}\, c = c$

(ii) (Monotonicity) If $X \leq Y$ almost surely, then $\mathbb{E}\, X \leq \mathbb{E}\, Y$

(iii) (Linearity) $\mathbb{E}(X + cY) = \mathbb{E}\, X + c\mathbb{E}\, Y$

Definition 1.4. *The variance of X, denoted by σ^2 or $\mathrm{Var}(X)$, is*

$$\sigma^2 = \mathrm{Var}(X) = \mathbb{E}(X - \mu)^2 = \sum_x (x - \mu)^2 f(x) = \sum_x x^2 f(x) - \mu^2.$$

The standard deviation of X is $\sigma = \sqrt{\sigma^2}$.

Theorem 1.2. *Let c be an arbitrary constant, and let X and Y be discrete random variables. The variance function has the following properties:*

(i) (Nonnegativity) $\mathrm{Var}\, X \geq 0$

(ii) (Variance of a constant) $\mathrm{Var}(c) = 0$

(iii) (Variance of a constant factor) $\mathrm{Var}\,(cX) = c^2 \mathrm{Var}\,X$

(iv) (Variance of a linear combination)

$$\mathrm{Var}\,(aX + bY) = a^2 \mathrm{Var}\,(X) + 2ab\mathrm{Cov}(X, Y) + b^2 \mathrm{Var}\,(Y)$$

(v) (Variance of sum of uncorrelated random variables)

$$\mathrm{Var}\,(X + Y) = \mathrm{Var}\,(X) + \mathrm{Var}\,(Y)$$

2 Markov chains

In order to understand how to bound the limiting disagreement in a discrete linear averaging algorithm subject to random noise, it is necessary to first look at how noise alone is propagated by such an algorithm. To accomplish this, it will be useful to review some useful concepts from probability theory, introduce Markov chains, and finally derive bounds for a simple case.

A major focus of classical probability theory, independent trials processes are useful for describing many common phenomena. Indeed, the discrete uniform distribution is useful for predicting the outcome of a fair die roll, the binomial distribution is useful for describing the outcome of successive binary outcome trials, the geometric distribution is good for describing the number of binary outcome trials necessary for a particular outcome to occur, etc. [Montgomery and Runger, 2007].

In many instances, it is reasonable to assume that a system or process is memoryless, i.e., that previous states have no effect on the outcome, but there are cases in which this strong assumption must be dropped. Indeed, consider R. A. Howard's classic example of a frog sitting on one of several lily pads in a pond. At any time, it may choose to move to an adjacent pad, or stay put, and each possibility has an associated probability. Clearly, in order to describe the process, one must include the current state when considering the next state.

In 1907, the Russian mathematician Andrei Andreyevich Markov began developing a tool for just such a probabilistic process. This type of process would later be named in his honor. Today, Markov chains are used in a wide variety of fields. Examples include modeling population processes in biology, generating sequences of random numbers from complex probability distributions, ranking search results for search engines, and quantifying the entropy of systems in information theory [Wikipedia, 2016].

2.1 Basic definitions

A Markov process describes the evolution of a system or process as a function of its previous states, in the presence of some randomness. In this way, Markov processes are similar in form to the equations of classical mechanics. That is, the state of a system at some point in the future is a function of the current state of the system. How the system came to be in its current state has no

bearing on the future state. Take as an example the trajectory of a thrown ball. Knowledge of the position and velocity of the ball at a single point in time gives the position and velocity at any other point in time. On the other hand, knowledge of just the position or velocity of the ball at some point, is not enough to determine the position or velocity at another point in time. From this example, it is clear that defining the state of the system as the position and velocity is an appropriate choice, whereas defining it as solely the position or velocity is not. It turns out that by carefully defining the state, nearly any random process can be described as a Markov process [Tsitsiklis, 2010].

A discrete-time, finite-state Markov chain is defined as a process that starts at some initial state s_i from a set of possible states $S = \{s_1, s_2, \ldots, s_r\}$ and transitions to other states with some probability denoted by p_{ij}, where s_j is the state at the next step. The fact that the next state of the system depends on the current state means that p_{ij} is a conditional probability, i.e., the probability of transitioning from s_i to s_j at step n is $p_{ij}^{(n)} = \mathrm{P}(s_n = s_j | s_{n-1} = s_i)$.

As the notation suggests, it is useful to present the set of transition probabilities in matrix form. This matrix is referred to as the *matrix of transition probabilities*, or simply as the *transition matrix*. The following theorem, taken directly from [Grinstead and Snell, 2006], illustrates an interesting property of the transition matrix, and gives a way the compute the *n-step transition probability*.

Theorem 2.1. *Let $\boldsymbol{P}$ be the transition matrix of a Markov chain. The ijth entry $p_{ij}^{(n)}$ of $\boldsymbol{P}^n$ gives the probability that the Markov chain, starting in state s_i, will be in state s_j after n steps.*

Proof. Consider the illustrative example in which there are $r = 2$ states, and transition matrix $\boldsymbol{P} = \left[\begin{smallmatrix} p_{11} & p_{22} \\ p_{21} & p_{22} \end{smallmatrix}\right]$. This system is illustrated in the *state diagram* provided in Figure 1.

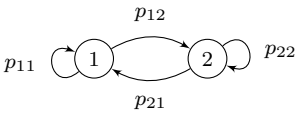


Figure 1: State diagram for a general 2-state Markov chain.

Without loss of generality, consider the transition from state s_1 to state s_2. The 1-step transition probability from s_1 to s_2 is p_{12}. The 2-step transition probability is the sum of the probabilities of all the ways to get from state s_1 to state s_2 in two steps. The probability of each route is computed using the multiplication rule for probabilities. With this,

$$p_{12}^{(2)} = p_{11}p_{12} + p_{12}p_{22} = \sum_{k=1}^{2} p_{1k}p_{k2} = [\boldsymbol{P}^2]_{12}$$

The 3-step transition probability is computed in the same fashion; the only

difference is the increased number of routes.

$$\begin{aligned} p_{12}^{(3)} &= p_{11}p_{11}p_{12} + p_{11}p_{12}p_{22} + p_{12}p_{22}p_{22} + p_{12}p_{21}p_{12} \\ &= p_{11}(p_{11}p_{12} + p_{12}p_{22}) + p_{12}(p_{21}p_{12} + p_{22}p_{22}) \\ &= \sum_{k=1}^{2} p_{1k} \left(\sum_{\ell=1}^{2} p_{k\ell}p_{\ell 2} \right) \\ &= \sum_{k=1}^{2} p_{1k}p_{k2}^{(2)} \\ &= [\boldsymbol{P}^3]_{12}. \end{aligned}$$

It is clear from these computations that if the process is continued for more steps, the formula will continue to hold. Moreover, additional states can be added, the only difference being an increased number of routes to consider. □

With a simple way to compute the n-step transition probability, it is natural to ask what the probability of being in each state after n-steps is, if the process starts in a known state. More generally, what can be said about the state after n steps if, rather than an initial state, an initial probability distribution is known. That is, what is the probability of it being at s_j after n steps, if initially it has some probability u_i of being at each state $s_i \in S$? The following theorem, taken directly from [Grinstead and Snell, 2006], answers this question.

Theorem 2.2. *Let $\boldsymbol{P}$ be the transition matrix of a Markov chain, and let $\boldsymbol{u}^\top = \boldsymbol{u}^\top(0)$ be the probability vector representing the starting distribution. Then the probability that the chain is in state s_i after n steps is the ith entry in the vector*

$$\boldsymbol{u}^\top(n) = \boldsymbol{u}^\top \boldsymbol{P}^n.$$

Proof. Let e_i represent the column vector of all zeros, and ith entry equal to one. Expand the product to obtain

$$\boldsymbol{u}^\top(n) = \boldsymbol{u}^\top \boldsymbol{P}^n = \begin{bmatrix} \boldsymbol{u}^\top \boldsymbol{P}^n \mathrm{e}_1 & \boldsymbol{u}^\top \boldsymbol{P}^n \mathrm{e}_2 & \cdots & \boldsymbol{u}^\top \boldsymbol{P}^n \mathrm{e}_r \end{bmatrix}.$$

The ith entry of $\boldsymbol{u}^\top(n)$ is

$$\boldsymbol{u}^\top \boldsymbol{P}^n \mathrm{e}_i = \sum_{k=1}^{r} \underbrace{u_k}_{\mathrm{P}(s_1 = s_k)} \quad \underbrace{\boldsymbol{P}_{ki}^n}_{\substack{n\text{-step} \\ \text{transition} \\ \text{probability} \\ \text{from } s_k \text{ to } s_i}} .$$

For initial state s_j, the probability that the chain will reach state s_i in n steps is the product of the probability of it starting in state s_k, u_k, and the n-step transition probability, $p_{ki}^{(n)}$ $(= \boldsymbol{P}_{ki}^n)$. Summing each probability gives the probability that the chain will reach state s_i in n steps from any starting state. Hence, the ith element of $\boldsymbol{u}^\top \boldsymbol{P}^n$ is the probability that the chain is in state s_i after n steps. □

Armed with this theorem, it is natural to wonder what the long-term behavior of the process is. In short, the answer to this question depends on the chain in question, and in particular on the types of states in the chain. A state in which the probability of departure is zero is known as an *absorbing* state. A state is said to be *recurrent* if, starting at that state, it will eventually return to that state. A state that is not absorbing is *transient*, and a set of transient states forms a *transient class*. A Markov chain with one or more absorbing states is called an *absorbing Markov chain*, whereas one with no absorbing states is referred to as an *ergodic*, or *irreducible*, *Markov chain* [Grinstead and Snell, 2006].

2.2 Absorbing Markov chains

Every state in a chain can be classified as either transient or absorbing. Since the two states have fundamentally different behavior in terms of what effect they have on future states of the chain, it makes sense to consider the chain as a collection of transient states that interact with a collection of absorbing states. The easiest way to analyze the network in this fashion is by rewriting the transition matrix $\boldsymbol{P}$ in *canonical form*. The definition of the canonical form, taken directly from [Grinstead and Snell, 2006], is provided below.

Definition 2.1. *Consider a general Markov chain with transition matrix $\boldsymbol{P}$, r absorbing states, and t transient states. Renumbering the states so that the transient states come before the absorbing states, the matrix $\boldsymbol{P}$ can be written in the following block form*

$$\boldsymbol{P} = \begin{bmatrix} \boldsymbol{Q} & \boldsymbol{R} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix},$$

where $\boldsymbol{Q} \in \mathbb{R}^{t \times t}$, $\boldsymbol{R} \in \mathbb{R}^{t \times r}$, and the identity matrix $\boldsymbol{I}$ is $r \times r$.

With the transition matrix in canonical form, it is easy to see that

$$\boldsymbol{P}^n = \begin{bmatrix} \boldsymbol{Q}^n & * \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix},$$

where $\boldsymbol{Q}^n$ is simply the n-step transition matrix within the set of transient states, $*$ is the n-step transition matrix from the set of transient states to the set of absorbing states, and $\boldsymbol{I}$ is the n-step transition matrix from the set of absorbing states to itself [Grinstead and Snell, 2006].

For any initial state in an absorbing chain, the chain will eventually reach an absorbing state, where it will remain for all further steps. Knowing this, it is natural to wonder how many transient states the chain will visit before being absorbed. As one would expect, it is generally not possible to predict precisely how many transient states will be visited, but it is possible to give the expected value of this quantity. The following definition, taken directly from [Grinstead and Snell, 2006] provides a means for such a prediction.

Theorem 2.3. *For an absorbing Markov chain $\boldsymbol{P}$, the matrix $\boldsymbol{N} = (\boldsymbol{I} - \boldsymbol{Q})^{-1}$ is called the* fundamental matrix for $\boldsymbol{P}$. *The entry n_{ij} of $\boldsymbol{N}$ gives the expected number of times that the process is in the transient state s_j if it is started in the transient state s_i.*

Proof. A proof is provided in [Grinstead and Snell, 2006]. □

In a similar vein, one might wonder how many steps must be taken before the chain is absorbed. The following theorem, taken directly from [Grinstead and Snell, 2006], answers this question.

Theorem 2.4. *Let a chain start in state s_i, and let t_i be the expected number of steps before the chain is absorbed. Let $\boldsymbol{t}$ be the column vector whose ith element is t_i. Then*

$$\boldsymbol{t} = \boldsymbol{N}\mathbb{1},$$

where $\mathbb{1}$ is the column vector with all unit entries.

Proof. From Theorem 2.3, n_{ij} is the expected number of times the chain is in state s_j if it started in s_i. A chain may only visit transient states before absorption, never after, so the number of times any transient state is visited before absorption is equal to the number of time steps to absorption. Mathematically,

$$t_i = \sum_{j=1}^{t} n_{ij} = \mathrm{e}_i^\top \boldsymbol{N}\mathbb{1}.$$

Hence, $\boldsymbol{t} = \boldsymbol{N}\mathbb{1}$. □

An important fact to note about this theorem is that it can actually be used to compute the expected number of steps between any two states. This time is commonly referred to as the *mean first passage time*, or *hitting time*, from the initial state to the terminus. To do this, one prescribes the initial state s_i, the terminal state s_j, and modifies the transition matrix $\boldsymbol{Q}$ so that all outgoing probabilities from s_j are zero, except p_{jj}, which is one. This modification makes s_j an absorbing state, and t_i the expected time to absorption for the modified chain, or mean first passage time for the original chain. An equivalent means of carrying out this computation is provided in Section 2.4.

2.3 Regular Markov chains

In an absorbing Markov chain with a single absorbing state, the chain will reach this state and remain there for all time. If there is more than one absorbing state, then the initial condition of the chain will have a role in determining the long-term state of the chain. In such a case, it is generally not possible to tell precisely where the state will be after a large number of steps, but a probabilistic description may be possible. That is, $\lim_{n\to\infty} \boldsymbol{u}^\top(n)$ may exist. A similar result holds for a certain type of ergodic chain, the *regular Markov chain*.

Definition 2.2. *A Markov chain with a primitive transition matrix $\boldsymbol{P}$ is called a* regular Markov chain *[Grinstead and Snell, 2006].*

Before continuing with the discussion of the convergence of the n-step transition matrix a brief review of primitive matrices and their properties is provided to ease the proof of the theorems.

First, the most common definition of a primitive matrix is given below.

Definition 2.3. *A matrix $\boldsymbol{P}$ is called* primitive *if there exists a positive integer k such that $\boldsymbol{P}^k > 0$ (each element of $\boldsymbol{P}^k$ is positive).*

This definition, while well-posed and concise, does not provide an efficient means for testing whether a given transition matrix is primitive. The following theorem addresses this deficiency by connecting the notion of primitivity to simple graph theoretical properties, thereby allowing one to recognize a primitive transition matrix by briefly inspecting the associated state diagram (see, e.g., Figure 1) for the chain.

Theorem 2.5. *A non-negative matrix $\boldsymbol{P}$ is primitive if and only if it is irreducible and aperiodic.*

Proof. $\boldsymbol{P}$ is aperiodic, so at least one state has a self-loop, say state s_ℓ. $\boldsymbol{P}$ is irreducible, so its associated graph, G, is strongly connected [Bullo, 2016]. Therefore, there exists a directed path between every pair of states in G. The number of directed paths of length k from state s_i to s_j is the (i, j) entry of $\boldsymbol{A}^k$, where $\boldsymbol{A}$ is the binary adjacency matrix associated to G [Bullo, 2016].

Now suppose there exists a directed path of length L starting at any state s_i, containing s_ℓ, and terminating at any state j. Then paths of length $L + 1$, $L+2, \ldots$ with terminals s_i and s_j can be constructed by repeatedly adding state s_ℓ to the original path. This implies $\exists m \in \mathbb{N}$ s.t. $\boldsymbol{A}^m > 0$. Thus, $\boldsymbol{A}$ is primitive. Consequently, $\boldsymbol{P}$ must also primitive. □

As an example, consider the state diagram in Figure 1. The graph is strongly connected ($\boldsymbol{P}$ is irreducible) if and only if $p_{12} \neq 0$ and $p_{21} \neq 0$. Additionally, for the graph to be aperiodic, there must be at least one self-loop, i.e., $p_{11} \neq 0$ or $p_{22} \neq 0$. Thus, $\boldsymbol{P}$ is primitive if and only if the off-diagonal elements, and at least one element on the main diagonal are positive.

Second, the Perron-Frobenius theorem, taken directly from [Bullo, 2016], provides useful constraints on the spectrum and eigenspace of a general primitive matrix.

Theorem 2.6. *Let $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ be primitive, with $n \geq 2$. Then*

(i) there exists a real simple eigenvalue $\lambda > 0$ such that $\lambda > |\mu|$ for all other eigenvalues μ, and

(ii) the right and left eigenvectors $\boldsymbol{v}$ and $\boldsymbol{w}^\top$ of λ are unique and positive, up to rescaling.

Proof. A proof is provided in [Bullo, 2016] (Chapter 2). □

Third, in the context of Markov chains, the transition matrix $\boldsymbol{P}$ is always row-stochastic. As stated in the following theorem, this property imposes yet another constraint on the of spectrum of $\boldsymbol{P}$.

Theorem 2.7. *For a row-stochastic matrix $\boldsymbol{P}$, 1 is an eigenvalue,* $\text{spec}(\boldsymbol{P})$ *is a subset of the unit disk, and $\rho(\boldsymbol{P}) = 1$.*

Proof. The proof of this theorem is a straight-forward application of the Gershgorin circle theorem. A proof complete with an illustrative diagram is provided in [Bullo, 2016]. □

Fourth, and finally, the eigenvalue and associated left-eigenvector mentioned in the Perron-Frobenius theorem come up so frequently that it is worthwhile to give them a special name just to simplify speech later on.

Definition 2.4. *Let $\boldsymbol{P}$ be a primitive row-stochastic matrix. The dominant eigenvalue of $\boldsymbol{P}$ is the unique maximum positive eigenvalue λ. The left-eigenvector associated with the dominant eigenvalue is referred to as the consensus eigenvector of $\boldsymbol{P}$, when normalized such that $\boldsymbol{w}^\top \mathbb{1} = 1$.*

Now, returning to the discussion of the asymptotic behavior of regular chains, consider the following theorem, taken directly from [Grinstead and Snell, 2006] and known as the fundamental limit theorem for regular Markov chains.

Theorem 2.8. *Let $\boldsymbol{P}$ be the transition matrix for a regular chain. Then,*

$$\lim_{n\to\infty} \boldsymbol{P}^n = \boldsymbol{W},$$

where $\boldsymbol{W}$ is the matrix with all rows equal to the consensus eigenvector $\boldsymbol{w}^\top$.

Proof. $\boldsymbol{P}$ is the transition matrix for a regular Markov chain, so it is primitive and stochastic. Theorems 2.6 and 2.7 assert that $\boldsymbol{P}$ has unit dominant eigenvalue λ, with associated right and left eigenvectors $\mathbb{1}$ and $\boldsymbol{w}^\top$, respectively, where $\boldsymbol{w}^\top$ is stochastic. Notice that $\boldsymbol{P}$ can be transformed into Jordan normal form by means of the similarity transformation

$$\boldsymbol{P} = \boldsymbol{T}\boldsymbol{J}\boldsymbol{T}^{-1} = \begin{bmatrix}\mathbb{1} & \tilde{\boldsymbol{V}}\end{bmatrix} \begin{bmatrix}\lambda & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{P}}\end{bmatrix} \begin{bmatrix}\boldsymbol{w} & \tilde{\boldsymbol{W}}^\top\end{bmatrix},$$

where $\tilde{\boldsymbol{P}}$ is the block of $\boldsymbol{J}$ associated with the eigenvalues of $\boldsymbol{P}$ in the interior of the unit disk, and $\tilde{\boldsymbol{V}}$ and $\tilde{\boldsymbol{W}}^T$ contain the right and left eigenvectors corresponding to these eigenvalues, respectively. The spectral radius of $\tilde{\boldsymbol{P}}$ is less than one, so $\tilde{\boldsymbol{P}}$ is convergent, i.e., $\lim_{n\to\infty} \tilde{\boldsymbol{P}}^n = \boldsymbol{0}$. With this, the desired result is obtained by computing the limit

$$\lim_{n\to\infty} \boldsymbol{P}^n = \lim_{n\to\infty} \begin{bmatrix}\mathbb{1} & \tilde{\boldsymbol{V}}\end{bmatrix} \begin{bmatrix}1 & \boldsymbol{0} \\ \boldsymbol{0} & \tilde{\boldsymbol{P}}\end{bmatrix}^n \begin{bmatrix}\boldsymbol{w} & \tilde{\boldsymbol{W}}^\top\end{bmatrix} = \mathbb{1}\boldsymbol{w}^\top = \boldsymbol{W}.$$

□

An interesting result of this theorem, is the fact that for a regular Markov chain the initial probability distribution has no effect on that in the long-term, as $n \to \infty$. This fact is stated in the following theorem, taken directly from [Grinstead and Snell, 2006].

Theorem 2.9. *Let $\boldsymbol{P}$ be the transition matrix for a regular chain and $\boldsymbol{v}^\top$ an arbitrary probability vector. Then*

$$\lim_{n\to\infty} \boldsymbol{v}^\top \boldsymbol{P}^n = \boldsymbol{w}^\top,$$

where $\boldsymbol{w}^\top$ is the unique fixed probability vector for $\boldsymbol{P}$.

Proof. A simple computation incorporating the claim in Theorem 2.8, and the fact that $\boldsymbol{v}^\top$ is a probability vector provides the desired result.

$$\begin{aligned}
\lim_{n\to\infty} \boldsymbol{v}^\top \boldsymbol{P}^n &= \boldsymbol{v}^\top \lim_{n\to\infty} \boldsymbol{P}^n = \boldsymbol{v}^\top \boldsymbol{W} \\
&= \boldsymbol{v}^\top \begin{bmatrix} w_1\mathbb{1} & w_2\mathbb{1} & \cdots & w_r\mathbb{1} \end{bmatrix} \\
&= \begin{bmatrix} w_1\boldsymbol{v}^\top\mathbb{1} & w_2\boldsymbol{v}^\top\mathbb{1} & \cdots & w_r\boldsymbol{v}^\top\mathbb{1} \end{bmatrix} \\
&= \boldsymbol{w}^\top.
\end{aligned}$$

□

An immediate consequence of this theorem is that a chain with distribution $\boldsymbol{w}^\top$ at the current time, will have distribution $\boldsymbol{w}^\top$ for all future times. For this reason, $\boldsymbol{w}^\top$ is known as the *fixed*, *stationary*, or *equilibrium* vector, or distribution. A similar independence result holds for the case of an ergodic Markov chain; the only difference is that the limiting distribution may not exist. This is the case when the chain is periodic.

The equilibrium distribution of a chain is the distribution the chain will tend to as the number of steps grows large, and is the answer to the earlier question about the long-term behavior of a chain. An equally interesting question concerns the behavior of the chain when the process is reversed. In the forward-case the one-step transition probability matrix $\boldsymbol{P}$ has elements p_{ij} denoting the probability of transitioning from state s_i to state s_j. In the reversed process, the transition matrix has elements p_{ji}, and is denoted by $\hat{\boldsymbol{P}}$. For a class of processes, called *reversible chains*, the stationary distribution for the forward-time process coincides with that of the reverse-time process. Formally,

Definition 2.5. *An ergodic chain is reversible if, and only if, for every pair of states s_i and s_j, $w_i p_{ij} = w_j p_{ji}$. Let $\boldsymbol{D}_{\boldsymbol{w}}$ denote diagonal matrix with entries $w_1, w_2, \ldots, w_r$, then the reversibility condition can be restated in terms of symmetry as*

$$\boldsymbol{D}_{\boldsymbol{w}}\boldsymbol{P} = (\boldsymbol{D}_{\boldsymbol{w}}\boldsymbol{P})^\top.$$

From this definition, it is easy to see that an ergodic process with a symmetric one-step transition matrix, $\boldsymbol{P}$, is reversible.

2.4 Mean first passage time

Mentioned at the end of Section 2.2, the concept of absorption time can be applied to compute the mean first passage time (MFPT) for an ergodic chain. While this is true, and is conceptually useful, it is also impractical. A better method for carrying out this computation is presented in this section, but before delving into it consider the definition of the MFPT, from [Grinstead and Snell, 2006].

Definition 2.6. *If an ergodic Markov chain is started in state s_i, the expected number of steps to reach state s_j for the first time is called the mean first passage, or hitting, time from s_i to s_j. It is denoted by m_{ij}. By convention, $m_{ii} = 0$.*

Mathematically, the MFPT for an irreducible, r-state Markov chain satisfies a system of r linear equations.

Theorem 2.10. *The MFPT from s_i to s_j satisfies the recursion*

$$m_{ij} = p_{ij} + \sum_{k \neq j} p_{ik} m_{kj}.$$

Proof. The key observation is that there are two possible ways to transition from s_i to s_j, $i \neq j$: in a single step, or in more than one step. In the first case, the MFPT is simply the transition probability, p_{ij}. In the other case, the chain goes from s_i to s_j by first stepping to some intermediate state s_k. In this event, notice that $m_{ij} = m_{kj} + 1$. Putting it all together and simplifying gives

$$\begin{aligned} m_{ij} &= p_{ij} + \sum_{k \neq j} p_{ik}(m_{kj} + 1) \\ &= \sum_{k=1}^{r} p_{ij} + \sum_{k \neq j} p_{ik} m_{kj} \\ &= 1 + \sum_{k \neq j} p_{ik} m_{kj}. \end{aligned}$$

□

In order to compute the MFPT, it is necessary to solve the above system of linear equations. The system can be written in matrix form as

$$\boldsymbol{m_j} = \mathbb{1} + \boldsymbol{Q_j} \boldsymbol{m_j},$$

or simply

$$(\boldsymbol{I} - \boldsymbol{Q_j}) \boldsymbol{m_j} = \mathbb{1}, \tag{2.1}$$

where $\boldsymbol{m_j}$ is the column vector of hitting times from s_i to s_j, for all $i \neq j$, and $\boldsymbol{Q_j}$ is the submatrix of $\boldsymbol{P}$ obtained by deleting the jth row and column. Note that $\boldsymbol{Q_j}$ is not the same as the matrix $\boldsymbol{Q}$ in the canonical form introduced in Definition 2.1. From this, the MFPT vector is given by

$$\boldsymbol{m_j} = (\boldsymbol{I} - \boldsymbol{Q_j})^{-1} \mathbb{1}.$$

A naive approach to solving this system for $\boldsymbol{m_j}$ would be by computing the matrix inverse. In general, the problem of computing matrix inverses is ill-posed, and this fact is only exacerbated as the dimension of the matrix increases, so it is bad practice [Ascher and Petzold, 1998]. Another method is Gaussian elimination, but fill-in will destroy the sparisty of $\boldsymbol{Q_j}$. Alternatively, it may be tempting to apply an iterative method such as the Jacobi, Gauss-Seidel, successive over-relaxation, and conjugate gradient method. However, the iterations will fail to converge since the matrix $\boldsymbol{I} - \boldsymbol{Q}$ is not generally symmetric, or positive-definite.

The process of matrix reduction can be used to solve the system. Consider the following definition, taken directly from [Sheskin, 1995].

Definition 2.7. *If $\boldsymbol{B}$ is a square, stochastic matrix, partitioned as*

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{T} & \boldsymbol{U} \\ \boldsymbol{R} & \boldsymbol{S} \end{bmatrix},$$

in which $\boldsymbol{S}$ is square, then the reduced matrix, or stochastic complement, of $\boldsymbol{T}$ in $\boldsymbol{B}$ is defined as the matrix

$$\boldsymbol{T} + \boldsymbol{U}(\boldsymbol{I} - \boldsymbol{S})^{-1}\boldsymbol{R}.$$

Now, consider the augmented matrix

$$\boldsymbol{G} = \begin{bmatrix} \mathbb{0} & \boldsymbol{I} \\ \mathbb{1} & \boldsymbol{Q_j} \end{bmatrix}, \tag{2.2}$$

where $\mathbb{0}$ is the zero vector, $\boldsymbol{I}$ is the $(r-1)$-by-$(r-1)$ identity matrix, and $\boldsymbol{Q_j}$ is as defined in Equation 2.1. The stochastic complement of $\boldsymbol{z}$ in $\boldsymbol{G}$ is

$$\mathbb{0} + \boldsymbol{I}(\boldsymbol{I} - \boldsymbol{Q_j})^{-1}\mathbb{1},$$

which is the same as the MFPT vector, $\boldsymbol{m_j}$!

The process of matrix reduction can be used to solve for the MFPT vector, $\boldsymbol{m_j}$, one element at a time. This is known as a state-reduction process, and more specifically, the Sheskin state-reduction process. The following theorem, taken from [Sheskin, 1995], explicitly states the Sheskin procedure.

Theorem 2.11. *Let $N = r - 1$. $\boldsymbol{m_j}$, the vector of hitting times from s_i to s_j can be computed by applying matrix reduction to the augmented matrix, $\boldsymbol{G}$, N times. The algorithm is as follows.*

(1) Initialize $k = N$, and $\boldsymbol{Q} = \boldsymbol{Q_j}_{N+k}$

(2) Let $\boldsymbol{G}(N+k) = \boldsymbol{G}$, as in Equation 2.2

(3) Partition $\boldsymbol{G}(N+k)$ as

$$\boldsymbol{G}(N+k) = \begin{bmatrix} \boldsymbol{T}_{N+k} & \boldsymbol{U}_{N+k} \\ \boldsymbol{R}_{N+k} & \boldsymbol{Q}_{N+k} \end{bmatrix},$$

where $\boldsymbol{G}_{N+k} \in \mathbb{R}^{(N+k)\times(N+k-2)}$, $\boldsymbol{T}_{N+k} \in \mathbb{R}^{(N+k-1)\times k}$, $\boldsymbol{R}_{N+k} \in \mathbb{R}^{1\times k}$, $\boldsymbol{U}_{N+k} \in \mathbb{R}^{(N+k-1)\times 1}$, and $\boldsymbol{Q}_{N+k} \in \mathbb{R}$.

(4) Perform one step of matrix reduction to $\boldsymbol{G(N+k)}$ *to obtain* $\boldsymbol{G}(N+K-1)$.

$$\boldsymbol{G}(N+k-1) = \boldsymbol{T}_{N+k} + \boldsymbol{U}_{N+k}(\boldsymbol{I} - \boldsymbol{Q}_{N+k})^{-1}\boldsymbol{R}_{N+k},$$

where

$$(\boldsymbol{I} - \boldsymbol{Q}_{N+k})^{-1} = \frac{1}{1 - g(N+k)_{N+k,k+1}}.$$

(5) Decrement k *by 1. If* $k > 0$*, go to step 3. Otherwise, stop.*

The final reduced matrix, $\boldsymbol{G}(N)$*, is* $\boldsymbol{m_j}$*.*

Proof. The proof of the theorem can be found in [Sheskin, 1995]. □

First, notice that the computations of $\boldsymbol{m_j}$, $j = 1, 2, \ldots, r$, are uncoupled, so the computation can be parallelized. Next, as in many algorithms that reduce the size of a matrix on each iteration, it is beneficial to allocate the full size of that matrix on the first iteration, and then modify a submatrix of diminishing size. Such a strategy reduces the amount of time the program spends managing resources, and can be applied to the computation of $\boldsymbol{G}_{N+k}$, $\boldsymbol{T}_{N+k}$, $\boldsymbol{R}_{N+k}$, $\boldsymbol{U}_{N+k}$, and $\boldsymbol{Q}_{N+k}$. Finally, the cost of computing the matrix of mean first passage times using this algorithm is $N^4+2N^3+N^2$ floating point operations (addition, subtraction, multiplication, and division).

Another thing to notice is that this algorithm makes no attempt to limit the number of subtractions made in the computation. This leads to a loss of significant digits that is exacerbated as the dimension of the transition matrix is increased. The Grassman, Taksar and Heyman (GTH) algorithm provides another means of carrying out the computation. A more accurate method, however, is the recently proposed extended GTH (EGTH) algorithm [Hunter, 2016].

Lastly, the set of vectors $\boldsymbol{m_j}$ are rather unwieldy, a more convenient alternative is placing them into a matrix form with the same structure as the transition matrix $\boldsymbol{P}$. Additionally, it is important to be able to express the hitting times between pairs of states for different Markov chains, without confusion. The following definition, introduces notation from [Jadbabaie and Olshevsky, 2015] that solves these problems.

Definition 2.8. *For a Markov chain with* r*-states, and transition matrix* $\boldsymbol{P}$*, the* r*-by-*r *MFPT matrix whose* ij*th element is the MFPT from* s_i *to* s_j *is denoted by* $\boldsymbol{M_P}$*. The* ij*th element of the MFPT matrix is denoted by* $\boldsymbol{M_P}(s_i \to s_j)$*.*

3 Consensus iteration

The primary focus of this report is the behavior of a ubiquitous recursion, often referred to as the *consensus iteration*. As its name would suggest, the iteration models the process of consensus formation among communicating entities.

Fittingly, the iteration plays a critical role in the design of distributed coordination and synchronization algorithms, i.e., policies that allow networked entities to coordinate their processes in the absence of a centralized controller.

In many applications, the quantity on which consensus is desired is subject to some additive noise. Sources of noise can be attributed to measurement, transmission, and discretization error, as well as uncertainty. In the presence of noise, the signal is polluted, and consensus is generally not reached. Instead of reaching an equilibrium, the estimates meander about the consensus value in the presence of noise. As will be shown in Section 4, the size of these excursions from consensus can be quantified when the system satisfies certain conditions.

The following subsections cover the noiseless and noisy consensus iteration, starting with the definition of the iteration, and closing with a description of the long-term behavior of the processes.

3.1 Noiseless iteration

Consider the definition of the noiseless consensus iteration, from [Jadbabaie and Olshevsky, 2015].

Definition 3.1. *The discrete-time linear system,*

$$\boldsymbol{x}(t+1) = \boldsymbol{P}\boldsymbol{x}(t+1),$$

where $\boldsymbol{P}$ is primitive and row-stochastic is known as the consensus iteration.

The name of the iteration derives from the fact that the matrix $\boldsymbol{P}$, when applied to the vector of opinions at time t, yields the updated vector $\boldsymbol{x}(t+1)$ whose elements are convex combinations of the elements of $\boldsymbol{x}(t)$. This process of repeated averaging, generates a sequence of iterates whose elements tend toward a common limit, i.e., a *consensus*. The following definition restates this fact.

Definition 3.2. *When all n components of $\boldsymbol{x}(t)$ converge to the same limit as $t \to \infty$, it is said that a consensus has been reached.*

The following theorem gives a sufficient condition for consensus formation.

Theorem 3.1. *If $\boldsymbol{P}$ is row-stochastic and primitive, then*

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \boldsymbol{w}^\top \boldsymbol{x}(0)\mathbb{1},$$

where $\boldsymbol{w}^\top$ is the consensus eigenvector of $\boldsymbol{P}$.

Proof. $\boldsymbol{P}$ is row-stochastic and primitive, so by Definition 2.2 it is the one-step transition probability matrix for a regular Markov chain. By Theorem 2.8, $\lim_{t\to\infty} \boldsymbol{P}^t = \boldsymbol{W}$, where $\boldsymbol{W}$ is the matrix with all rows equal to the consensus eigenvector, $\boldsymbol{w}^\top$, of $\boldsymbol{P}$. Hence,

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \lim_{t\to\infty} \boldsymbol{P}^t\boldsymbol{x}(0) = \boldsymbol{W}\boldsymbol{x}(0) = \boldsymbol{w}^\top\boldsymbol{x}(0)\mathbb{1}.$$

□

To demonstrate this theorem, consider the following primitive transition matrix, and initial opinion vector:

$$\boldsymbol{P} = \begin{bmatrix} 0.09 & 0.38 & 0.31 & 0.23 \\ 0.90 & 0.10 & 0.00 & 0.00 \\ 0.48 & 0.00 & 0.52 & 0.00 \\ 0.72 & 0.00 & 0.00 & 0.28 \end{bmatrix}, \text{ and } \boldsymbol{x}(0) = \begin{bmatrix} 0.60 \\ 0.30 \\ 0.13 \\ 0.21 \end{bmatrix}.$$

The consensus eigenvector and limiting opinion are given by

$$\boldsymbol{x}(\infty) = \big(0.37, 0.37, 0.37, 0.37\big), \text{ and } \boldsymbol{w}^\top = \begin{bmatrix} 0.42 & 0.18 & 0.27 & 0.13 \end{bmatrix},$$

and the process is illustrated in Figure 2.

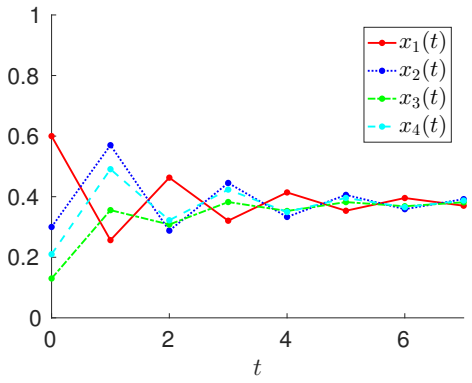


Figure 2: Noiseless consensus iteration on a network with star topology, center node 1, and weights sampled from the standard uniform distribution, along with the initial opinion vector.

Finally, in the case in which the transition matrix $\boldsymbol{P}$ is symmetric and the consensus is convergent, the system is said to achieve average consensus. In this configuration, the consensus value is the simple average of the initial opinion vector, so each agent is said to have had an equal weight on the final outcome of the process. For this reason, this configuration is referred to as the democracy configuration. The following definition explicitly restates this notion, which will be used repeatedly in Section 4.1.

Definition 3.3. *If $\boldsymbol{P}$ is symmetric, stochastic, and primitive, then*

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \frac{1}{n}\mathbb{1}\mathbb{1}^\top \boldsymbol{x}(0) = \text{average } (\boldsymbol{x}(0))\,\mathbb{1}$$

where $\boldsymbol{w}^\top = \frac{1}{n}\mathbb{1}^\top$ is the consensus eigenvector of $\boldsymbol{P}$. In this case, the system is said to have achieved average consensus [Bullo, 2016].

3.2 Noisy iteration

With a good understanding of the noiseless consensus iteration, complicate the model by including an additive noise term. The following is the definition of the consensus iteration with additive noise.

Definition 3.4. *The discrete-time stochastic process,*

$$\boldsymbol{x}(t+1) = \boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{v}(t),$$

where $\boldsymbol{P}$ is primitive and row-stochastic, and $\boldsymbol{v}(t)$ is a vector of independent, identically distributed (i.i.d.) random variables, with zero mean and unit variance is known as the noisy consensus iteration.

In the presence of additive noise, the consensus iteration becomes a stochastic process, whereas the noiseless version was deterministic. As such, the noisy version is less well behaved. In one sense the node values do converge to consensus, but not in a very meaningful sense. To see this consider the following two theorems from [Xiao et al., 2007].

Theorem 3.2. *$\mathbb{E}\,\boldsymbol{x}(t)$, the expected value of $\boldsymbol{x}(t)$, is propagated in the same way as $\boldsymbol{x}(t)$ is by the noiseless iteration. That is,*

$$\mathbb{E}\,\boldsymbol{x}(t+1) = \boldsymbol{P}\mathbb{E}\,\boldsymbol{x}(t).$$

If $\boldsymbol{P}$ is row-stochastic and primitive, with consensus eigenvector $\boldsymbol{w}^\top$, then the expectation has the consensus value

$$\lim_{t\to\infty} \mathbb{E}\,\boldsymbol{x}(t) = \boldsymbol{w}^\top \mathbb{E}\,\boldsymbol{x}(0)\mathbb{1}.$$

Proof. Start by writing an expression for the expected value of $\boldsymbol{x}(t+1)$, inserting the recursion for $\boldsymbol{x}(t)$, using the linearity of the expected value function, and the fact that the noise has zero mean. This gives

$$\mathbb{E}\,\boldsymbol{x}(t+1) = \mathbb{E}\,(\boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{v}(t)) = \boldsymbol{P}\mathbb{E}\,\boldsymbol{x}(t) = \boldsymbol{P}^t\mathbb{E}\,\boldsymbol{x}(0).$$

Finally, as shown in Theorem 3.1, the expected value of the node values converges to the consensus distribution

$$\lim_{t\to\infty} \mathbb{E}\,\boldsymbol{x}(t) = \lim_{t\to\infty} \boldsymbol{P}^t\mathbb{E}\,\boldsymbol{x}(0) = \boldsymbol{W}\mathbb{E}\,\boldsymbol{x}(0) = \boldsymbol{w}^\top\mathbb{E}\,(\boldsymbol{x}(0))\mathbb{1}.$$

□

Theorem 3.3. *The node values $\boldsymbol{x}(t)$ do not converge.*

Proof. This proof hinges on showing that the variance of an error measure increases with time. At consensus, the variance in opinions is zero, so consensus is not reached when the signal is corrupted by noise.

Consider the function $\bar{\boldsymbol{x}}(t) = \boldsymbol{w}^\top\boldsymbol{x}(t)\mathbb{1} = W\boldsymbol{x}(t)$, where $\boldsymbol{W}$ is the matrix with all rows equal to the consensus eigenvector, $\boldsymbol{w}^\top$, of $\boldsymbol{P}$. $\bar{\boldsymbol{x}}(t)$ can be viewed as

the consensus value of the noiseless iteration with initial condition $\boldsymbol{x}(t)$. Notice that $\bar{\boldsymbol{x}}(0) = \boldsymbol{w}^\top \boldsymbol{x}(0)\mathbb{1}$, is equal to the consensus value for the noiseless process. With this, the expected value of the difference $\boldsymbol{z}(t) \coloneqq \bar{\boldsymbol{x}}(t) - \bar{\boldsymbol{x}}(0)$ is a measure of the error in the consensus value of the noisy process, at time t. Using the linearity of the expected value, and Theorem 3.2, in the limit

$$\begin{aligned}
\lim_{t\to\infty} \mathbb{E}\,\boldsymbol{z}(t) &= \lim_{t\to\infty} \mathbb{E}\,(\boldsymbol{W}\boldsymbol{x}(t)) - \mathbb{E}\,(\boldsymbol{W}\boldsymbol{x}(0)) \\
&= \boldsymbol{W} \lim_{t\to\infty} \mathbb{E}\,\boldsymbol{x}(t) - \boldsymbol{W}\mathbb{E}\,\boldsymbol{x}(0) \\
&= \boldsymbol{W}(\boldsymbol{W}\mathbb{E}\,\boldsymbol{x}(0)) - \boldsymbol{W}\mathbb{E}\,\boldsymbol{x}(0) \\
&= (\boldsymbol{W}^2 - \boldsymbol{W})\mathbb{E}\,\boldsymbol{x}(0).
\end{aligned}$$

Furthermore,

$$\boldsymbol{W}^2 = \begin{bmatrix} \boldsymbol{w}^\top \\ \vdots \\ \boldsymbol{w}^\top \end{bmatrix} \begin{bmatrix} w_1\mathbb{1} & \cdots & w_n\mathbb{1} \end{bmatrix} = \begin{bmatrix} w_1\boldsymbol{w}^\top\mathbb{1} & \cdots & w_n\boldsymbol{w}^\top\mathbb{1} \end{bmatrix} = \boldsymbol{W},$$

so the error has zero mean in the limit, i.e.,

$$\lim_{t\to\infty} \mathbb{E}\,\boldsymbol{z}(t) = \boldsymbol{0}.$$

This makes sense considering the claim of Theorem 3.2. However, the variance of the error increases with time. Indeed,

$$\begin{aligned}
\operatorname{Var}\bar{\boldsymbol{x}}(t+1) &= \operatorname{Var}\,(\boldsymbol{W}\boldsymbol{x}(t)) \\
&= \boldsymbol{W}^2 \operatorname{Var}\boldsymbol{x}(t+1) \\
&= \boldsymbol{W}^2\,[\operatorname{Var}\,(\boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{v}(t))] \\
&= \boldsymbol{W}^2\left[\operatorname{Var}\,(\boldsymbol{P}^2\boldsymbol{x}(t-1) + \boldsymbol{v}(t-1)) + \operatorname{Var}\boldsymbol{v}(t)\right] \\
&= \boldsymbol{W}^2\left[\operatorname{Var}\,(\boldsymbol{P}^t\boldsymbol{x}(0)) + \sum_{k=1}^{t} \operatorname{Var}\boldsymbol{v}(k)\right].
\end{aligned}$$

This expression can be simplified by making the following observations: the elements of $\boldsymbol{v}(t)$ are *i.i.d.*, so $\operatorname{Var}\boldsymbol{v}(t) = \boldsymbol{I}$, for all $t > 0$; $\boldsymbol{W}^2 = \boldsymbol{W}$; and $\boldsymbol{W}\boldsymbol{P} = \boldsymbol{W}$. With this,

$$\begin{aligned}
\operatorname{Var}\bar{\boldsymbol{x}}(t+1) &= \boldsymbol{W}\boldsymbol{P}^{2t}\,[\operatorname{Var}\boldsymbol{x}(0) + t\boldsymbol{I}] \\
&= \boldsymbol{W}\,[\operatorname{Var}\boldsymbol{x}(0) + t\boldsymbol{I}]\,,
\end{aligned}$$

so the variance of the error grows linearly with time. Indeed,

$$\operatorname{Var}\,[\bar{\boldsymbol{x}}(t+1) - \bar{\boldsymbol{x}}(0)] = \boldsymbol{W}\,[\operatorname{Var}\boldsymbol{x}(0) + t\boldsymbol{I}] - \boldsymbol{W}^2\operatorname{Var}\boldsymbol{x}(0) = t\boldsymbol{I}.$$

□

To demonstrate these results, return to the example at the end of the previous section. In that example, the noiseless iteration converged to the consensus value in about 7 steps. If noise is introduced to the system, then the opinions of the sensors will fail to converge (c.f. Theorem 3.3), but their expected value will (c.f. Theorem 3.2). As is often the case, the noise term is much smaller than the full-scale value of the signal. Here, suppose the noise is 1% of the full-scale value of the signal. To achieve this for additive noise sampled from the standard normal distribution with zero mean and unit variance, sample opinions from the uniform distribution on $[0, 100]$. To retain similarity between examples, scale the old initial opinion vector, $\boldsymbol{x}(0)$, by 100, and reuse it. With this setup, one realization of the noisy iteration is plotted in Figure 3.

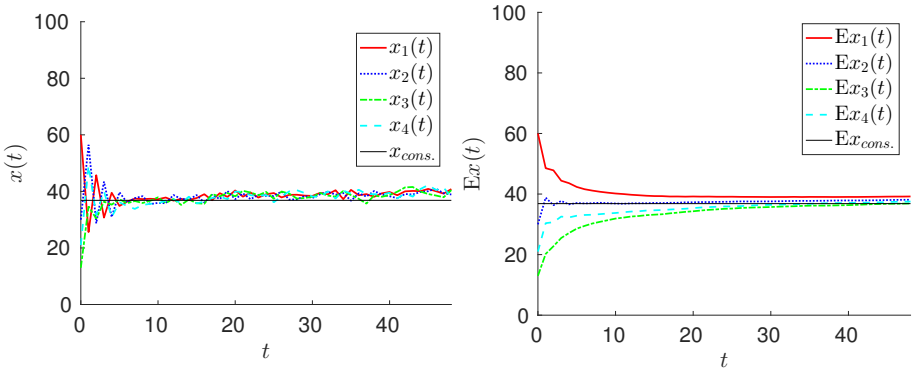


Figure 3: Noisy consensus iteration on a network with star topology, and center node 1. The weight matrix and initial opinion vector are the same as those used to generate the graphs in Figure 2, but the initial opinion is 100-times larger. The noise is sampled from the standard normal distribution ($\mu = 0$, $\sigma^2 = 1$).

4 Limiting disagreement

As demonstrated in the previous section, the consensus iteration fails to converge when corrupted by additive noise. The error measure $\bar{\boldsymbol{x}}(t)$, defined as the error in consensus values for the noiseless iteration with initial distributions $\boldsymbol{x}(t)$ and $\boldsymbol{x}(0)$, was shown to have zero mean, and linearly increasing variance. This result means that the noisy consensus iteration behaves much like the noiseless iteration, but meanders about the consensus value instead of converging.

In that error is present in every real-world application, it is important to understand the effect that additive noise has on the iteration. One way to go about this is by defining a new measure of the size of excursions from the consensus value one can expect when the expected value of the iterate has converged. The first section contains a powerful theorem from [Xiao et al., 2007] that addresses this problem for the case of a symmetric averaging matrix. Although it is not novel, the proof of the theorem provides a great deal of

insight, and it shows how useful the symmetry assumption really is. In the following section, the problem is generalized to the case in which the weight matrix is just reversible. Another powerful theorem, this time from [Jadbabaie and Olshevsky, 2015], provides a means of computing the limiting disagreement.

4.1 Symmetric transitions

In this section, a simple case of the more general Theorem 4.8 (given in Section 4.2) is derived for symmetric weight matrix $\boldsymbol{W}$. The derivation closely follows that given in [Xiao et al., 2007], but assumes non-unit variance for the i.i.d. noise term, goes somewhat further before resorting to symmetry enforcement, and goes into excruciating detail. The derivation is broken down into a number of smaller theorems, with the main result at the end of the section.

Consider the noisy consensus iteration (Definition 3.4) with symmetric, row-stochastic weight matrix $\boldsymbol{P}$. Let $v_i(t)$, $i = 1, \ldots, n$, $t = 0, 1, \ldots$, be independent random variables, identically distributed, with zero mean and variance σ^2. Let $\boldsymbol{w}^\top = \frac{1}{n}\mathbb{1}$ denote the consensus eigenvector of $\boldsymbol{P}$, and $\boldsymbol{W} = \mathbb{1}\boldsymbol{w}^\top$ denote the weighted averaging matrix. If the noiseless iteration converges to consensus, then $\|\boldsymbol{P} - \boldsymbol{W}\| < 1$.

In Section 3.2 the error measure $\bar{\boldsymbol{x}}(t)$ was used to show that the noisy consensus iteration is not convergent. This error measure is not well-suited for the task of quantifying the limiting disagreement of the iteration. A better measure is the vector of deviations of each agent from the consensus value, at time t, denoted by $\boldsymbol{z}(t)$. The *deviation vector* is defined by

$$\boldsymbol{z}(t) = \boldsymbol{x}(t) - \boldsymbol{W}\boldsymbol{x}(t) = (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{x}(t).$$

The *total mean-square deviation*, a measure of the total distance of the components of $\boldsymbol{x}(t)$ from consensus, is defined by

$$\delta(t) = \mathbb{E}\,\|\boldsymbol{z}(t)\|^2.$$

The final theorem of this section gives an explicit formula for the value of this error measure at steady state. The first step toward this result is proving the following theorem regarding the mean value of the deviation vector, at steady-state.

Theorem 4.1. *Let $\boldsymbol{P}$, $\boldsymbol{x}(t)$, and $\boldsymbol{v}(t)$ be as in Definition 3.4. Then*

$$lim_{t\to\infty}\mathbb{E}\,\boldsymbol{z}(t) = \boldsymbol{0}.$$

Proof. By definition, $\boldsymbol{z}(t) = (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{x}(t)$, so

$$\begin{aligned}
\boldsymbol{z}(t+1) &= (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{x}(t+1) \\
&= (\boldsymbol{I} - \boldsymbol{W})(\boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{v}(t)) \\
&= \boldsymbol{P}\boldsymbol{x}(t) - \boldsymbol{W}\boldsymbol{P}\boldsymbol{x}(t) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t) \\
&= \boldsymbol{P}\boldsymbol{x}(t) - \boldsymbol{W}\boldsymbol{x}(t) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t) \\
&= (\boldsymbol{P} - \boldsymbol{W})\boldsymbol{x}(t) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t) \\
&= (\boldsymbol{P} - \boldsymbol{W})(\boldsymbol{z}(t) + \boldsymbol{W}\boldsymbol{x}(t)) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t) \\
&= (\boldsymbol{P} - \boldsymbol{W})\boldsymbol{z}(t) + (\boldsymbol{P}\boldsymbol{W} - \boldsymbol{W}^2)\boldsymbol{x}(t) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t) \\
&= (\boldsymbol{P} - \boldsymbol{W})\boldsymbol{z}(t) + (\boldsymbol{I} - \boldsymbol{W})\boldsymbol{v}(t)
\end{aligned}$$

The expectation of $\boldsymbol{z}(t)$ is then given by

$$\begin{aligned}
\mathbb{E}\,\boldsymbol{z}(t) &= (\boldsymbol{P} - \boldsymbol{W})\mathbb{E}\,\boldsymbol{z}(t) + (I - \boldsymbol{W})\mathbb{E}\,\boldsymbol{v}(t) \\
&= (\boldsymbol{P} - \boldsymbol{W})(I - \boldsymbol{W})\mathbb{E}\,\boldsymbol{x}(t) \\
&= (\boldsymbol{P} - \boldsymbol{P}\boldsymbol{W} - \boldsymbol{W} + \boldsymbol{W}^2)\boldsymbol{P}^{t-1}\mathbb{E}\,\boldsymbol{x}(0) \\
&= (\boldsymbol{P} - \boldsymbol{W})\boldsymbol{P}^{t-1}\mathbb{E}\,\boldsymbol{x}(0) \\
&= (\boldsymbol{P}^t - \boldsymbol{W})\mathbb{E}\,\boldsymbol{x}(0) \\
&= (\boldsymbol{P} - \boldsymbol{W})^t\mathbb{E}\,\boldsymbol{x}(0).
\end{aligned}$$

But $\|\boldsymbol{P} - \boldsymbol{W}\| < 1$, so $\lim_{t\to\infty} \mathbb{E}\,\boldsymbol{z}(t) = 0$. □

This result is reassuring, and unsurprising, given the similar finding in the proof of Theorem 3.3 regarding the limiting expected value of the error, $\bar{\boldsymbol{x}}(t)$. As in that proof, the next item to consider is the variance of the error measure. The following theorem relates the total mean-square deviation to the variance of the deviation vector.

Theorem 4.2. *Let $\boldsymbol{\Sigma}(t) = \mathbb{E}\,\boldsymbol{z}(t)\boldsymbol{z}^\top(t)$. This is the second moment matrix of the deviation vector, $\boldsymbol{z}(t)$. The second central moment of a probability density function is the variance of the distribution. The total mean-square deviation can be expressed in terms of $\boldsymbol{\Sigma}(t)$ as $\delta(t) = \operatorname{Tr}\boldsymbol{\Sigma}(t)$.*

Proof. By definition, the total mean-square deviation, $\delta(t)$, is given by

$$\delta(t) = \mathbb{E}\,\|\boldsymbol{z}(t)\|^2 = \mathbb{E}\,\boldsymbol{z}(t)^\top\boldsymbol{z}(t) = \mathbb{E}\,\sum_{i=1}^{n} z_i^2(t).$$

Consider the outer-product $\boldsymbol{Z} = \boldsymbol{z}\boldsymbol{z}^\top$. The ijth element of $\boldsymbol{Z}$, denoted by z_{ij}, is given by $z_{ij} = z_i z_j$. With this, the trace is clearly $\operatorname{Tr}\boldsymbol{Z} = \sum_{i=1}^n z_i z_i = \sum_{i=1}^n z_i^2$. The desired result is obtained by combining this observation with the fact that both the trace and expected value functions are linear.

$$\delta(t) = \mathbb{E}\operatorname{Tr}\boldsymbol{z}(t)\boldsymbol{z}(t)^\top = \operatorname{Tr}\mathbb{E}\,\boldsymbol{z}(t)\boldsymbol{z}(t)^\top = \operatorname{Tr}\boldsymbol{\Sigma}(t).$$

□

With this simple characterization of the total mean-square deviation, $\delta(t)$, it is clear that the dynamics of $\delta(t)$ follow immediately from an understanding of the dynamics of the deviation second moment matrix, $\boldsymbol{\Sigma}(t)$. The following theorem gives the difference equation and initial condition characterizing these dynamics.

Theorem 4.3. *The deviation second moment matrix, $\boldsymbol{\Sigma}(t)$, satisfies the difference equation*

$$\boldsymbol{\Sigma}(t+1) = (\boldsymbol{P}-\boldsymbol{W})\boldsymbol{\Sigma}(t)(\boldsymbol{P}-\boldsymbol{W})^\top + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{\Sigma}_{\boldsymbol{v}}((\boldsymbol{I}-\boldsymbol{W}))^\top,$$

with initial condition

$$\boldsymbol{\Sigma}(0) = (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{x}(0)\boldsymbol{x}(0)^\top(\boldsymbol{I}-\boldsymbol{W})^\top,$$

where $\boldsymbol{\Sigma}_{\boldsymbol{v}}$ is the covariance matrix of $\boldsymbol{v}(t)$.

Proof. To save space, in the following manipulations, assume each time-varying variable is taken at time t, unless otherwise indicated. Now, consider the outer-product $\boldsymbol{Z}(t+1) = \boldsymbol{z}(t+1)\boldsymbol{z}(t+1)^\top$. Using the expression for $\boldsymbol{z}(t)$ found in the proof of Theorem 4.1, and expanding each factor gives

$$\begin{aligned}\boldsymbol{Z}(t+1) &= [(\boldsymbol{P}-\boldsymbol{W})\boldsymbol{x} + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{v}]\,[(\boldsymbol{P}-\boldsymbol{W})\boldsymbol{x} + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{v}]^\top \\ &= (\boldsymbol{P}-\boldsymbol{W})\boldsymbol{Z}(\boldsymbol{P}-\boldsymbol{W})^\top + (\boldsymbol{P}-\boldsymbol{W})\boldsymbol{z}\boldsymbol{v}^\top(\boldsymbol{I}-\boldsymbol{W})^\top \\ &\quad + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{v}\boldsymbol{z}^\top(\boldsymbol{P}-\boldsymbol{W})^\top + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{v}\boldsymbol{v}^\top(\boldsymbol{I}-\boldsymbol{W})^\top.\end{aligned}$$

The noise components $v_i(t)$, $i = 1, \ldots, n$, $t = 0, 1, \ldots$, are independent random variables with zero mean, so taking the expectation of the above equation yields the following first-order inhomogeneous difference equation.

$$\boldsymbol{\Sigma}(t+1) = (\boldsymbol{P}-\boldsymbol{W})\boldsymbol{\Sigma}(t)(\boldsymbol{P}-\boldsymbol{W})^\top + (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{\Sigma}_{\boldsymbol{v}}(\boldsymbol{I}-\boldsymbol{W})^\top.$$

The initial condition is obtained from the definition of the deviation second moment matrix.

$$\begin{aligned}\boldsymbol{\Sigma}(0) &= \mathbb{E}\,\boldsymbol{z}(0)\boldsymbol{z}(0)^\top = \mathbb{E}\left[(\boldsymbol{I}-\boldsymbol{W})\boldsymbol{x}(0)\boldsymbol{x}(0)^\top(\boldsymbol{I}-\boldsymbol{W})^\top\right] \\ &= (\boldsymbol{I}-\boldsymbol{W})\boldsymbol{x}(0)\boldsymbol{x}(0)^\top(\boldsymbol{I}-\boldsymbol{W})^\top.\end{aligned}$$

□

The above theorem is interesting, but to be useful, it needs to be considered in the light of the following theorem summarizing some basic results of Lyapunov stability theory from [Hespanha, 2009].

Theorem 4.4. *Let $\boldsymbol{A}, \boldsymbol{P}, \boldsymbol{Q} \in \mathbb{R}^{n\times n}$, with $\boldsymbol{P}, \boldsymbol{Q}$ symmetric. Let $\boldsymbol{A} \succ \boldsymbol{0}$ denote that $\boldsymbol{A}$ is positive definite. Then the following two statements are equivalent:*

(i) For every symmetric $\boldsymbol{Q} \succ \mathbf{0}$, there exists a unique $\boldsymbol{P} \succ \mathbf{0}$ satisfying the discrete-time Lyapunov equation

$$\boldsymbol{A}^\top \boldsymbol{P} \boldsymbol{A} - \boldsymbol{P} - \boldsymbol{Q} = \mathbf{0}.$$

Moreover, $\boldsymbol{P} \succ \mathbf{0}$ is symmetric, and is given by

$$\boldsymbol{P} = \sum_{k=0}^{\infty} \boldsymbol{A}^k \boldsymbol{Q} (\boldsymbol{A}^\top)^k.$$

(ii) The linear system discrete-time linear system

$$\boldsymbol{x}(t+1) = \boldsymbol{A}\boldsymbol{x}(t)$$

is globally asymptotically stable, i.e., $\boldsymbol{A}$ is Schur stable.

With this theorem, it can be shown that the difference equation from Theorem 4.3 has a unique stable equilibrium solution. In fact, this is shown in the following theorem.

Theorem 4.5. *The difference equation of Theorem 4.3 is stable. Further, if $\boldsymbol{P}$ is symmetric, then the steady-state deviation second moment matrix is given by*

$$\boldsymbol{\Sigma_{ss}} \coloneqq \lim_{t \to \infty} \boldsymbol{\Sigma}(t) = \sigma^2 \left((\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}^2)^{-1} - \boldsymbol{W} \right).$$

Proof. Suppose the difference equation is stable. Then there exists an equilibrium $\boldsymbol{\Sigma_{ss}} = \lim_{t \to \infty} \boldsymbol{\Sigma}(t)$. Insert this matrix into the difference equation and rearrange terms to see that it is of the form of the discrete-time Lyapunov equation from Theorem 4.5.

$$(\boldsymbol{P} - \boldsymbol{W}) \boldsymbol{\Sigma_{ss}} (\boldsymbol{P} - \boldsymbol{W})^\top - \boldsymbol{\Sigma_{ss}} + (\boldsymbol{I} - \boldsymbol{W}) \boldsymbol{\Sigma_v} (\boldsymbol{I} - \boldsymbol{W})^\top = \mathbf{0}.$$

By construction, $\boldsymbol{\Sigma}(t)$ is symmetric, so $\boldsymbol{\Sigma_{ss}}$ is also symmetric. The noise covariance matrix, given by $\boldsymbol{\Sigma})\boldsymbol{v} = \sigma^2 \boldsymbol{I}$, is symmetric, so the matrix $(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{\Sigma})\boldsymbol{v}(\boldsymbol{I} - \boldsymbol{W})^\top$ is symmetric as well. Moreover,

$$\begin{aligned} \boldsymbol{x}^\top (\boldsymbol{I} - \boldsymbol{W}) \boldsymbol{\Sigma_v} (\boldsymbol{I} - \boldsymbol{W})^\top \boldsymbol{x} &= \left[\sigma (\boldsymbol{I} - \boldsymbol{W})^\top \boldsymbol{x} \right]^\top \left[\sigma (\boldsymbol{I} - \boldsymbol{W})^\top \boldsymbol{x} \right] \\ &= \| \sigma (\boldsymbol{I} - \boldsymbol{W})^\top \boldsymbol{x} \|^2 \geq 0 \end{aligned}$$

for any nonzero $\boldsymbol{x} \in \mathbb{R}^n$, so it is also positive definite. Additionally, the noiseless consensus iteration is convergent, i.e., $\|\boldsymbol{P} - \boldsymbol{W}\| < 1$, so the matrix $(\boldsymbol{P} - \boldsymbol{W})^\top$ is convergent. Therefore, $\boldsymbol{\Sigma_{ss}}$ exists, is unique, and is given by

$$\begin{aligned} \boldsymbol{\Sigma_{ss}} &= \sum_{k=0}^{\infty} \left((\boldsymbol{P} - \boldsymbol{W})^\top \right)^k (\boldsymbol{I} - \boldsymbol{W}) \boldsymbol{\Sigma_v} (\boldsymbol{I} - \boldsymbol{W})^\top (\boldsymbol{P} - \boldsymbol{W}) \\ &= \sigma^2 \sum_{k=0}^{\infty} (\boldsymbol{P} - \boldsymbol{W})^k (\boldsymbol{I} - \boldsymbol{W})^2 (\boldsymbol{P} - \boldsymbol{W})^k \end{aligned}$$

if the symmetry of $\boldsymbol{P}$ is enforced. The following easily verified facts will be used to simplify this expression: $\boldsymbol{WP} = \boldsymbol{W}$, $\boldsymbol{PW} = \boldsymbol{W}$, $\boldsymbol{W}^2 = \boldsymbol{W}$, $(\boldsymbol{P}-\boldsymbol{W})\boldsymbol{W} = \boldsymbol{0}$, $(\boldsymbol{P}-\boldsymbol{W})^2 = \boldsymbol{P}^2 - \boldsymbol{W}$.

$$\begin{aligned}
\sigma^{-2}\boldsymbol{\Sigma_{ss}} &= \boldsymbol{I} - \boldsymbol{W} + \sum_{k=1}^{\infty}(\boldsymbol{P}-\boldsymbol{W})^k(\boldsymbol{I}-\boldsymbol{W})(\boldsymbol{P}-\boldsymbol{W})^k \\
&= \boldsymbol{I} - \boldsymbol{W} + \sum_{k=1}^{\infty}(\boldsymbol{P}-\boldsymbol{W})^{2k} - (\boldsymbol{P}-\boldsymbol{W})^k W(\boldsymbol{P}-\boldsymbol{W})^k \\
&= \boldsymbol{I} - \boldsymbol{W} + \sum_{k=0}^{\infty}(\boldsymbol{P}-\boldsymbol{W})^{2k} - \boldsymbol{I} \\
&= -\boldsymbol{W} + \sum_{k=0}^{\infty}\left((\boldsymbol{P}-\boldsymbol{W})^2\right)^k.
\end{aligned}$$

Next, recall that the Neumann series $\sum_{k=0}^{\infty}\boldsymbol{A}^k$ is convergent, with sum $(\boldsymbol{I}-\boldsymbol{A})^{-1}$, if $\|\boldsymbol{A}\| < 1$. With this, the steady-state deviation second moment matrix can be written as

$$\boldsymbol{\Sigma_{ss}} = \sigma^2\left(-\boldsymbol{W} + (\boldsymbol{I} - (\boldsymbol{P}-\boldsymbol{W})^2)^{-1}\right) = \sigma^2\left((\boldsymbol{I}+\boldsymbol{W}-\boldsymbol{P}^2)^{-1} - \boldsymbol{W}\right).$$

□

With this description of the deviation second moment matrix, it is relatively simple to relate it back to the total mean-square deviation, $\delta(t)$, as is shown below.

Theorem 4.6. *The steady-state mean-square deviation is given by*

$$\begin{aligned}
\delta_{ss} &= \sigma^2\left(\mathrm{Tr}\,(\boldsymbol{I}+\boldsymbol{J}-\boldsymbol{P}^2)^{-1} - 1\right) \\
&= \sigma^2\left(\frac{1}{2}\mathrm{Tr}\,(\boldsymbol{I}+\boldsymbol{J}-\boldsymbol{P})^{-1} + \frac{1}{2}\mathrm{Tr}\,(\boldsymbol{I}-\boldsymbol{J}+\boldsymbol{P})^{-1} - 1\right).
\end{aligned}$$

Proof. By Theorem 4.2, $\delta(t) = \mathrm{Tr}\,\boldsymbol{\Sigma}(t)$, so $\delta_{ss} = \mathrm{Tr}\,\boldsymbol{\Sigma_{ss}}$. In terms of the weight and averaging matrices, this is

$$\begin{aligned}
\delta_{ss} = \mathrm{Tr}\,\boldsymbol{\Sigma_{ss}} &= \sigma^2\mathrm{Tr}\,\left((\boldsymbol{I}+\boldsymbol{W}-\boldsymbol{P}^2)^{-1} - \boldsymbol{W}\right) \\
&= \sigma^2\left(\mathrm{Tr}\,(\boldsymbol{I}+\boldsymbol{W}-\boldsymbol{P}^2)^{-1} - \mathrm{Tr}\,(\boldsymbol{W})\right) \\
&= \sigma^2\left(\mathrm{Tr}\,(\boldsymbol{I}+\boldsymbol{W}-\boldsymbol{P}^2)^{-1} - \mathrm{Tr}\,\left(\frac{1}{n}\mathbb{1}\mathbb{1}^\top\right)\right) \\
&= \sigma^2\left(\mathrm{Tr}\,(\boldsymbol{I}+\boldsymbol{W}-\boldsymbol{P}^2)^{-1} - 1\right).
\end{aligned}$$

For the second equality, use the factorization for $\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}^2$ given and verified below.

$$\begin{aligned}\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}^2 &= (\boldsymbol{I} - \boldsymbol{W} + \boldsymbol{P})(\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}) \\ &= \boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}^2 + \cancel{\boldsymbol{P} - \boldsymbol{P}} + \cancel{\boldsymbol{W}\boldsymbol{P} - \boldsymbol{W}} + \cancel{\boldsymbol{P}\boldsymbol{W} - \boldsymbol{W}^2}\end{aligned}$$

The inverse of this product can be split using the following identity from matrix analysis.

$$\boldsymbol{A}^{-1} \coloneqq [(\boldsymbol{I} - \boldsymbol{B})(\boldsymbol{I} + \boldsymbol{B})]^{-1} = \frac{1}{2}(\boldsymbol{I} - \boldsymbol{B})^{-1} + \frac{1}{2}(\boldsymbol{I} + \boldsymbol{B})^{-1}$$

To see that this is true, use the fact that the factors of $\boldsymbol{A}$ commute, i.e., $\boldsymbol{A} = (\boldsymbol{I} - \boldsymbol{W})(\boldsymbol{I} + \boldsymbol{W}) = (\boldsymbol{I} + \boldsymbol{W})(\boldsymbol{I} - \boldsymbol{W}) = \boldsymbol{I} - \boldsymbol{B}^2$, and compute the product $\boldsymbol{A}^{-1}\boldsymbol{A}$.

$$\begin{aligned}\boldsymbol{A}^{-1}\boldsymbol{A} &= \frac{1}{2}(\boldsymbol{I} - \boldsymbol{B})^{-1}(\boldsymbol{I} - \boldsymbol{B})(\boldsymbol{I} + \boldsymbol{B}) + \frac{1}{2}(\boldsymbol{I} + \boldsymbol{B})^{-1}(\boldsymbol{I} + \boldsymbol{B})(\boldsymbol{I} - \boldsymbol{B}) \\ &= \frac{1}{2}(\boldsymbol{I} + \boldsymbol{B}) + \frac{1}{2}(\boldsymbol{I} - \boldsymbol{B}) = \boldsymbol{I}.\end{aligned}$$

Together, these facts give

$$\begin{aligned}\delta_{ss} &= \sigma^2 \left(\operatorname{Tr} (\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}^2)^{-1} - 1 \right) \\ &= \sigma^2 \left(\frac{1}{2} \operatorname{Tr} (\boldsymbol{I} + \boldsymbol{J} - \boldsymbol{P})^{-1} + \frac{1}{2} \operatorname{Tr} (\boldsymbol{I} - \boldsymbol{J} + \boldsymbol{P})^{-1} - 1 \right). \qquad (4.1)\end{aligned}$$

□

Finally, consider the main result of the section, the closed-form description of the mean-square deviation in terms of the eigenvalues of the weight matrix $\boldsymbol{P}$.

Theorem 4.7. *The steady-state disagreement for the noisy consensus iteration with symmetric weight matrix, $\boldsymbol{P}$, satisfying the condition $\|\boldsymbol{P} - \boldsymbol{W}\| < 1$, where $\boldsymbol{W} = \frac{1}{n}\mathbb{1}\mathbb{1}^\top$, is given by*

$$\delta_{ss} = \sigma^2 \sum_{i=2}^{n} \frac{1}{1 - \lambda_i(\boldsymbol{P})^2}.$$

Proof. Denote by $\lambda_1, \lambda_2, \ldots, \lambda_n$ the eigenvalues of $\boldsymbol{P}$, in descending order of magnitude, and by $v_1, v_2, \ldots, v_n$ the associated eigenvectors. Recall that the trace of a matrix is the sum of its eigenvalues, and the eigenvalues of a matrix are the reciprocals of those of its inverse. Applying these facts to the claim of Theorem 4.6 yields

$$\begin{aligned}\sigma^{-2}\delta_{ss} &= \frac{1}{2}\sum_{i=1}^{n} \lambda_i \left((\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P})^{-1} \right) + \frac{1}{2}\sum_{i=1}^{n} \lambda_i \left((\boldsymbol{I} - \boldsymbol{W} + \boldsymbol{P})^{-1} \right) - 1 \\ &= \frac{1}{2}\sum_{i=1}^{n} \frac{1}{\lambda_i(\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P})} + \frac{1}{2}\sum_{i=1}^{n} \frac{1}{\lambda_i(\boldsymbol{I} - \boldsymbol{W} + \boldsymbol{P})} - 1\end{aligned}$$

Next, notice that both $\boldsymbol{I} + \boldsymbol{W} - \boldsymbol{P}$ and $\boldsymbol{I} - \boldsymbol{W} + \boldsymbol{P}$ are row-stochastic, so they share the eigenvalue 1. Indeed,

$$(\boldsymbol{I} \pm \boldsymbol{W} \mp \boldsymbol{P})\mathbb{1} = \mathbb{1} \pm \boldsymbol{W}\mathbb{1} \mp \boldsymbol{P}\mathbb{1} = \mathbb{1} \pm \mathbb{1} \mp \mathbb{1} = \mathbb{1}.$$

Recall that since $\boldsymbol{P}$ is symmetric it has n mutually orthogonal eigenvectors, i.e., ${v_i}^\top v_j = 0$ for all $i \neq j$. Therefore,

$$(\boldsymbol{I} \pm \boldsymbol{W} \mp \boldsymbol{P})v_i = v_i \pm \boldsymbol{W}v_i \mp \boldsymbol{P}v_i = v_i \pm \mathbb{1}{v_1}^\top v_i \mp \lambda_i v_i = (1 \mp \lambda_i)v_i,$$

so the spectra are given by

$$\operatorname{spec}(\boldsymbol{I} \pm \boldsymbol{W} \mp \boldsymbol{P}) = \{1\} \cup \{1 \mp \lambda_i(\boldsymbol{P})\}_{i=2}^n .$$

With this, δ_{ss} can be rewritten as

$$\delta_{ss} = \sigma^2 \left[\frac{1}{2} + \frac{1}{2} \sum_{i=2}^{n} \frac{1}{1 - \lambda_i(\boldsymbol{P})} + \frac{1}{2} + \frac{1}{2} \sum_{i=2}^{n} \frac{1}{1 + \lambda_i(\boldsymbol{P})} - 1 \right].$$

A simpler form for δ_{ss} is obtained by using the identity $1/(1+y) + 1/(1-y) = 2/(1-y^2)$, for $y \neq 1$. Indeed, this last step yields the desired result,

$$\delta_{ss} = \sigma^2 \sum_{i=2}^{n} \frac{1}{1 - \lambda_i(\boldsymbol{P})^2}.$$

□

4.2 Reversible transitions

After seeing the derivation in the previous section, it is clear that the symmetry assumption greatly simplifies the analysis. Here, this constraint is relaxed, and an exact expression for the limiting disagreement in the more general case of reversible weight matrices is presented. The content of this section is almost entirely from [Jadbabaie and Olshevsky, 2015].

Consider again the noisy consensus iteration of Section 3.2. Let $\boldsymbol{P}$ denote the primitive, row-stochastic weight matrix, with consensus eigenvector $\boldsymbol{w}^\top$. Let $\boldsymbol{D_w} \coloneqq \operatorname{diag}(w_1, w_2, \ldots, w_n)$ be the matrix with $\boldsymbol{w}^\top$ on the main diagonal. Let $\boldsymbol{\Sigma_v}$ denote the covariance matrix of the noise vector $\boldsymbol{v}(t)$. As in Theorem 3.3, let $\bar{\boldsymbol{x}}$ denote the weighted average of the opinion vector $\boldsymbol{x}$, and $\boldsymbol{z} \coloneqq \boldsymbol{x} - \bar{\boldsymbol{x}}$ denote the difference between the current state and the consensus value of the noiseless iteration.

As in Section 4.1, the total deviation is measured by the mean-square deviation. As is, this formulation weights the deviation at each node equally. An alternative measure is the mean-square deviation with weights assigned according to the "influence" of each node.To clarify, nodes whose initial opinion make up a greater portion of the consensus opinion, are said to have a greater influence over the discourse.

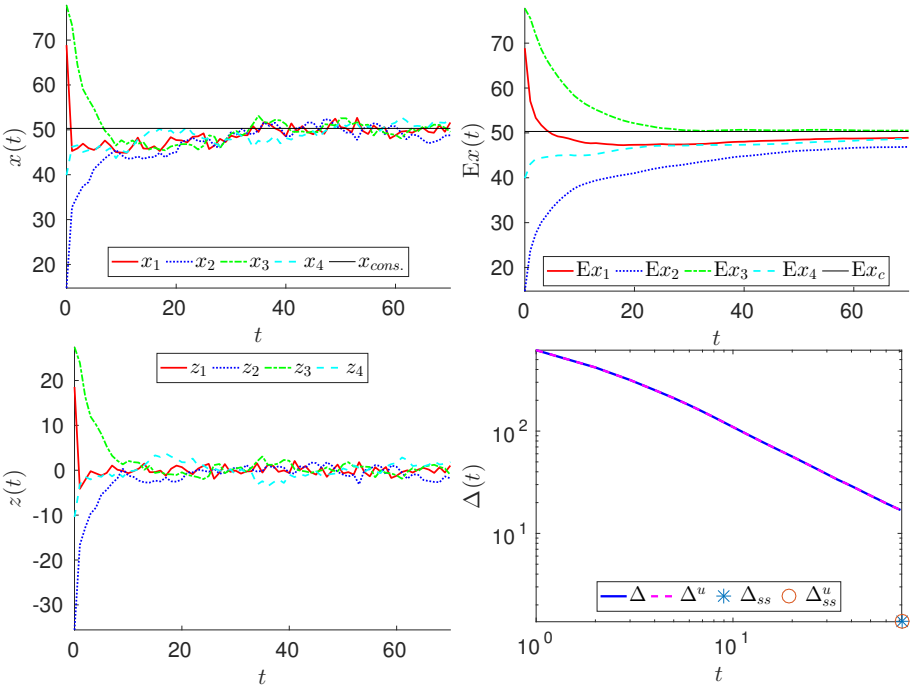


Figure 4: The noisy consensus iteration on a star graph with symmetric random edge weights. The measurements $\boldsymbol{x}(t)$ quickly approach the consensus value within the first 20 steps, while the size of the deviations $\boldsymbol{z}(t)$ relative to the variance of the additive noise $\boldsymbol{v}(t)$ is large. When the deviations and noise are of the same order of magnitude, the convergence to zeros of the deviations is disrupted. The expected-value $\mathbb{E}\,\boldsymbol{x}(t)$ of each set of measurements starts at the initial measurement values. As the iteration proceeds, the the set of expectations are slowly drawn together, and the limiting disagreement of the sensors reaches a limiting value $\boldsymbol{\Delta}$. In that the matrix $\boldsymbol{P}$ is symmetric, the limiting disagreements Δ_{ss} and Δ_{ss}^{uni} coincide.

Conceptually simple, the uniformly weighted deviation gives the extent to which signals drift, independent of the stationary distribution. No exact expression for the uniformly-weighted mean-square deviation for the noisy consensus iteration with reversible transition matrix is known of at this time [Jadbabaie and Olshevsky, 2015].

The uniformly weighted deviation does not account the fact that more influential nodes have greater potential to propagate error than others do. To accurately capture the effects of varying degrees of contribution, it makes sense to use the weighted mean-square deviation, with weights determined by the stationary distribution associated with the transition matrix, from Section 4.1. The main contribution of [Jadbabaie and Olshevsky, 2015] is an exact expression for this deviation, for the case of the reversible transition matrix.

These total deviation measures are denoted by $\Delta(t)$ and $\Delta^{\text{uni}}(t)$, respectively, are defined as follows.

$$\Delta(t) \coloneqq \sum_{i=1}^{n} w_i \mathbb{E}\, \boldsymbol{z}_i^2(t), \tag{4.2}$$

$$\Delta^{\text{uni}}(t) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\, \boldsymbol{z}_i^2(t). \tag{4.3}$$

With this, the *limiting disagreement*, is defined by

$$\Delta_{ss} \coloneqq \limsup_{t\to\infty} \Delta(t),$$
$$\Delta_{ss}^{\text{uni}} \coloneqq \limsup_{t\to\infty} \Delta^{\text{uni}}(t).$$

What follows is the main theoretical contribution of [Jadbabaie and Olshevsky, 2015], a characterization of the weighted mean-square deviation of signals for the noisy consensus iteration, in terms of the combinatorial properties of the underlying Markov chain.

Theorem 4.8. *If the noisy consensus iteration (Definition 3.4) with transition matrix $\boldsymbol{P}$ is primitive and reversible, then the limiting disagreement for the process is given by*

$$\Delta_{ss} = \boldsymbol{w}^\top \boldsymbol{M}_{\boldsymbol{P}^2} \boldsymbol{D}_{\boldsymbol{w}} \boldsymbol{\Sigma}_{\boldsymbol{v}} \boldsymbol{D}_{\boldsymbol{w}} \mathbb{1} - \text{Tr}\left(\boldsymbol{M}_{\boldsymbol{P}^2} \boldsymbol{D}_{\boldsymbol{w}} \boldsymbol{\Sigma}_{\boldsymbol{v}} \boldsymbol{D}_{\boldsymbol{w}}\right).$$

If, in addition, the noises at different nodes are uncorrelated, then

$$\Delta_{ss}(\boldsymbol{P}, \boldsymbol{\Sigma}_{\boldsymbol{v}}) = \sum_{i=1}^{n} \sum_{j=1}^{n} \sigma_i^2 w_i^2 w_j \boldsymbol{M}_{\boldsymbol{P}^2}(j \to i).$$

Furthermore, if the noises all have the same variance, then

$$\Delta_{ss}(\boldsymbol{P}, \sigma^2 \boldsymbol{I}) = \sigma^2 \sum_{i=1}^{n} \sum_{j=1}^{n} w_i^2 w_j \boldsymbol{M}_{\boldsymbol{P}^2}(j \to i).$$

Proof. The proof for this theorem is provided in [Jadbabaie and Olshevsky, 2015], and is extremely similar to that for the symmetric case (Theorem 4.7). □

Comparing the definitions of total disagreement (Equations 4.2, 4.3), to the definition of mean-square deviation in Theorem 4.2, it is clear that the steady-state mean-square deviation, in the symmetric case, is related to the uniformly-weighted limiting-disagreement by

$$\Delta^{\text{uni}}(t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}\, z_i^2(t) = \frac{1}{n} \text{Tr}\, \boldsymbol{\Sigma}(t) = \frac{1}{n} \delta(t) \tag{4.4}$$

Similarly, the weighted disagreement can be computed using

$$\Delta(t) = \sum_{i=1}^{n} \boldsymbol{w}_i^\top \mathbb{E}\, z_i^2(t) = \text{Tr}\left(\boldsymbol{D}_{\boldsymbol{w}} \boldsymbol{\Sigma}(t)\right) \tag{4.5}$$

5 Interpretation: Opinion dynamics

Longtime a focus of social psychologists, opinion dynamics is concerned with describing and modeling the way in which ideas, beliefs, and opinions of individuals change as they interact with others in a social network. Until recently, engineers had little use for such work, but as networked systems have become more widespread, parallels between the seemingly disparate fields have become increasingly apparent. Today, networked systems have become so pervasive in everyday life as to warrant the descriptor "the internet of things", and the push to improve the understanding and function of such systems has prompted the exchange of ideas between previously unbridged disciplines.

Formally introduced in 1974, the DeGroot model of opinion dynamics describes the effect of interpersonal influence on the opinions of the members of a group [DeGroot, 1974]. The sentiments an individual has toward another determine the influence one accords to the other [Jia et al., 2015]. A pair of close friends would likely accord a high degree of influence to each other, whereas two strangers would accord nothing to one other. The particular areas of expertise an individual displays to others also affects their degree of influence on those others when an issue connected to these areas is presented to the group [DeGroot, 1974], and even when an unconnected issue is presented. Likewise, the degree to which one is closed to outside influence can be interpreted as the weight one gives their own feelings on a matter [Jia et al., 2015]. Taken together, the set of interpersonal influences forms an influence network, in which each pair of adjacent individuals grants a weight to the eachother's opinions [Jia et al., 2015]. When an issue is presented to the group, the final opinion of each individual is determined by the influence network topology, and the set of interpersonal influences [Jia et al., 2015].

5.1 The standard DeGroot model

Consider a group of r individuals, tasked with collaborating to estimate the unknown value of some m-dimensional parameter $\boldsymbol{\theta}$. Initially, each individual i has some opinion about each dimension of the unknown parameter, and stores these opinions in a row vector $\boldsymbol{u_i}(0)^\top$. The group of individuals comprises a social network, and each individual has a set of neighbors with which they communicate, influence, and are influenced by.

Through this network, group members are made aware of the opinions of their neighbors, and broadcast their opinions to their neighbors. This exchange of information causes individuals to update their initial opinions based on those of others, according to their perception of others. Literature from empirical social psychology supports the notion that an individual forms new opinions as a convex combination of their current opinion and those of their neighbors [Jia et al., 2015]. The weights of these convex combinations are generated by a variety of conscious, and unconscious, cognitive mechanisms . In that each member of the group may have different areas of expertise, and varying degrees of information about the unknown parameter, it follows that, in this process of

opinion formation, certain individuals may exert a greater level of influence over their peers than will others. Mathematically, the degree of influence individual i accords j is denoted by p_{ij}, where $\sum_i p_{ij} = 1$. In summary, the opinion of individual i after the first update is

$$\boldsymbol{u_i}(1)^\top = \boldsymbol{P}\boldsymbol{u_i}(0)^\top,$$

where $\boldsymbol{P}$ is the r-by-r matrix with elements p_{ij}. Thus, if $\boldsymbol{U}(0)$ is the matrix with rows $\boldsymbol{u_1}(0)^\top$, $\boldsymbol{u_2}(0)^\top$, ..., $\boldsymbol{u_r}(0)^\top$, then the opinions of each individual are given by the rows of the matrix

$$\boldsymbol{U}(1) = \boldsymbol{P}\boldsymbol{U}(0),$$

The columns of this matrix contain the opinion of each individual on a fixed dimension of the unknown parameter $\boldsymbol{\theta}$, and are denoted by $\boldsymbol{x_i}$, for $i = 1, 2, \ldots, r$.

After the initial update, the group will be in the position is was before the update. That is, each member will display their opinion, and be subjected to the opinions of their peers. It is assumed that their perception of their peers will not change from its initial value, i.e., the influence network encoded by the weight matrix $\boldsymbol{P}$ is unchanged from one step to another. Hence, if iterated, the opinion matrix for the group at step n is given by

$$\boldsymbol{U}(n) = \boldsymbol{P}\boldsymbol{U}(n-1) = \ldots = \boldsymbol{P}^n\boldsymbol{U}(0).$$

It is assumed that individuals have no access to outside information regarding the unknown parameter $\boldsymbol{\theta}$. Hence, the update process is driven by the discrepancy members observe between their own opinions and those of their peers, and so it will continue until the opinions of every individual and their peers are in agreement, or until the process is terminated. In the case of agreement, the mismatch between opinions has been eliminated, and the opinion no longer changes when the process is iterated further. In such a case, the process is said to have reached an equilibrium state. If in addition, the opinion of every individual matches at equilibrium, the group is said to have reached a *consensus*. The following definition, from [DeGroot, 1974], gives a mathematical description of consensus.

Definition 5.1. *When all r components of $\boldsymbol{U}(n)$ converge to the same limit as $n \to \infty$, a consensus is said to have been reached.*

What this definition is saying is that at consensus, each individual shares the same opinion concerning each dimension k of the unknown parameter $\boldsymbol{\theta}$. Thus, at consensus, $\boldsymbol{x_i}$ is a constant vector, for $i = 1, \ldots, m$.

To characterize the consensus distribution, it is necessary to consider the matrix of weights, $\boldsymbol{P}$. First, notice that $\boldsymbol{P}$ is row-stochastic and encodes the interpersonal influence network of the group. With this, $\boldsymbol{P}$ can be interpreted as the one-step transition probability matrix of a Markov chain with r states and transition probabilities p_{ij} corresponding to the probability of individual i

subscribing to the opinion of individual j on any, and all, dimensions of the unknown parameter $\boldsymbol{\theta}$ when they update their opinion. Consequently, the Markov chain machinery erected in Section 2 can be used to derive conditions on the convergence of the process. The first such condition is presented in the theorem below, which was taken from [DeGroot, 1974].

Theorem 5.1. *If all the recurrent states of a Markov chain communicate with each other and are aperiodic, then a consensus is reached.*

Proof. Let $\boldsymbol{P}$ denote the one-step transition probability matrix for the Markov chain. If all recurrent states in the chain communicate with each other, then a directed path exists between each pair of recurrent states. Therefore, the chain cannot contain any absorbing states, which, while recurrent, are inescapable. Containing no absorbing states, the chain must contain only transient states, so it is ergodic. Moreover, the existence of a directed path between each pair of states implies the existence of a positive integer k, such that the ijth entry of $\boldsymbol{P}^k$ is positive. Thus, $\boldsymbol{P}$ is primitive, by Definition 2.3. Therefore, by Theorem 2.8, $\lim_{n\to\infty} \boldsymbol{P}^n = \boldsymbol{W}$, where all rows of $\boldsymbol{W}$ are equal to the vector $\boldsymbol{w}^\top$. Thus,

$$\begin{aligned}\lim_{n\to\infty} \boldsymbol{U}(n) &= \left(\lim_{n\to\infty} \boldsymbol{P}^n\right) \boldsymbol{U}(0) \\ &= \boldsymbol{W}\boldsymbol{U}(0) \\ &= \begin{bmatrix} \boldsymbol{w}^\top \boldsymbol{x_1}(0)\mathbb{1} & \boldsymbol{w}^\top \boldsymbol{x_2}(0)\mathbb{1} & \cdots & \boldsymbol{w}^\top \boldsymbol{x_m}(0)\mathbb{1} \end{bmatrix}.\end{aligned}$$

This limiting matrix has constant columns, so by Definition 5.1 a consensus is reached. □

As can be seen above, the DeGroot process for parameters of dimension is equivalent to m decoupled, or parallel, processes, each having the same influence matrix $\boldsymbol{P}$ and acting on a separate dimension of the parameter estimate. Because of this, the standard definition of the DeGroot model deals with the evolution of a point-estimate of an unknown parameter, i.e., the $m = 1$ case. The definition, from [Jia et al., 2015], is stated below.

Definition 5.2 (Standard DeGroot model)**.** *Consider a group of r individuals, each having an opinion on an unknown parameter $\boldsymbol{\theta} \in \mathbb{R}$. Let the opinions at time t be stored in the vector $\boldsymbol{x}(t)$. For a strongly connected, aperiodic, influence network represented by a row-stochastic matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, the opinions are updated via the process*

$$\boldsymbol{x}(t+1) = \boldsymbol{P}\boldsymbol{x}(t), \quad t = 0, 1, 2, \ldots,$$

with iterates tending toward the consensus distribution, given by

$$\lim_{t\to\infty} \boldsymbol{x}(t) = \boldsymbol{w}^\top \boldsymbol{x}(0)\mathbb{1},$$

where $\boldsymbol{w}^\top$ is the consensus eigenvalue of $\boldsymbol{P}$.

Written in this form, the DeGroot model of opinion dynamics is clearly identical to the noiseless consensus iteration introduced in Section 3. This means all the results from that section are applicable to this model, and conversely, that the behavior of the consensus iteration can be interpreted as a process of opinion formation on an influence network. This lastly explains some of the language used in describing the consensus iteration in preceding sections.

5.2 The noisy DeGroot model

The DeGroot model of opinion dynamics is elegantly simple; as a consequence, it fails to capture several critical phenomena. Most notably, groups of individuals do not, in general, come to consensus. This persistence of disagreement, while in some sense context dependent, can be achieved by introducing an additive noise term to the standard DeGroot model. Unsurprisingly, the resulting model, the noisy DeGroot model, is identical to the noisy consensus iteration. The model is defined below.

Definition 5.3 (Standard DeGroot model)**.** *Consider a group of r individuals, each having an opinion on an unknown parameter $\boldsymbol{\theta} \in \mathbb{R}$. Let the opinions at time t be stored in the vector $\boldsymbol{x}(t)$. For a strongly connected, aperiodic, influence network represented by a row-stochastic matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$, the opinions are updated via the process*

$$\boldsymbol{x}(t+1) = \boldsymbol{P}\boldsymbol{x}(t) + \boldsymbol{v}(t), \quad t = 0, 1, 2, \ldots,$$

where $\boldsymbol{v}(t)$ is a vector of i.i.d. random variables, with zero mean and unit variance.

In that the model is identical to the noisy consensus iteration, the results from Section 3.2 can be applied to it, and can be viewed through the lens of an opinion formation process.

6 Sample influence networks

Much has been said in previous sections about the matrix $\boldsymbol{P}$, but only one example of such a matrix has been given, and that was all the way back in the toy example of Section 3.1. In the context of Markov chains, $\boldsymbol{P}$ is referred to as a (1-step probability) transition matrix; in the context of the various flavors of the consensus iteration, it is referred to as a weight matrix; and so too in the context of opinion dynamics, but with weights encoding the set of interpersonal influences within a social network.

In each of these domains, the process converges to a stationary distribution, stationary vector, or consensus, provided that $\boldsymbol{P}$ is primitive, and row-stochastic (and that iterates are free from the pollution of additive noise!). Further, in order to compute the limiting disagreement of the process, $\boldsymbol{P}$ must also be reversible (a weakening of the symmetry property). Theorem 2.5 asserts that the first of these requirements has a graph theoretical interpretation. The last two

requirements, provided with a suitable network topology, can be met through clever choices of edge weights. Examples of such networks, and edge weights are provided in the following sections.

6.1 Graphs

The focus of this section is a set of simple, graphs whose associated adjacency matrices are primitive. That is, graphs that are both strongly connected and aperiodic. The first of these properties is achieved easily by considering only connected undirected graphs, rather than the more general case of connected digraphs, which generally harbor absorbing states. Aperiodicity can be achieved by adding a self-loop to a strongly connected graph.

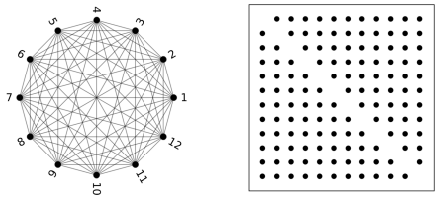
6.1.1 The complete graph

A complete graph is one in which each pair of vertices is connected. The complete graph with 12 nodes, and no self-loops, is illustrated in Figure 5, along with the zero pattern of its associated adjacency matrix. For a fixed number of vertices, the complete graph has the maximal number of edges, and the greatest degree of connectivity. The complete graph is an example of a *regular graph*, that is, a graph in which each pair of vertices have the same number of neighbors.

Complete graphs occur frequently in small social networks, in which each member is aware of the others, and has their own perception, disposition, and unconscious reactions to.

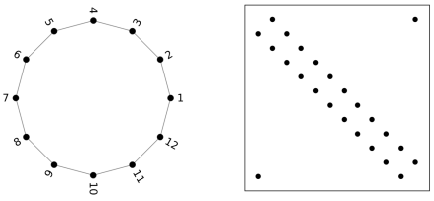
6.1.2 The circle graph

The circle graph is comprised of a set of vertices connected in a line, with the head connected to the tail, forming a ring. The circle graph with 12 vertices is given in Figure 6, along with the zero pattern of its adjacency matrix. The circle graph is an example of a regular graph.



(a) (b)

Figure 5: (a) The complete graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.



(a) (b)

Figure 6: (a) The circle graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

6.1.3 The line graph

The line graph is identical to the circle graph with any single edge deleted. The line graph and its adjacency matrix are provided in Figure 7. For a given number of vertices, the line graph has the minimum number of edges of any connected network.

6.1.4 The star graph

The star graph is the graph in which each pair of nodes connects to a single central node, and no other. The star graph and its adjacency matrix are provided in Figure 8.

The star graph corresponds to the autocratic configuration in the Friedkin-Johnsen model of social power evolution. The social power of an individual is measured by the degree of influence that individual has on the consensus opinion, i.e., the weight of their initial opinion, or their entry in the consensus eigenvector, in the final consensus opinion of the group. A wealth of social psychological data suggests that an individual's self-appraisal is largely due the opinions, perceptions, and dispositions that others display toward that individual. This mechanism of social power evolution is referred to as *reflected appraisal*. Taken together, the self-weights, which correspond to self-appraisals, are updated with the social powers, which correspond to the entries in the consensus eigenvector, after each DeGroot process. On a star topology, this model suggests an accumulation of social power for the central nodes, or the emergence of an autocrat [Jia et al., 2015].

Star graphs also occur frequently in engineering applications, in which a single master node communicates with and coordinates the actions of a set of unconnected slave nodes. A simple example is a simple Wi-Fi network, in which a wireless router sends and receives data from a set of internet-connected devices.

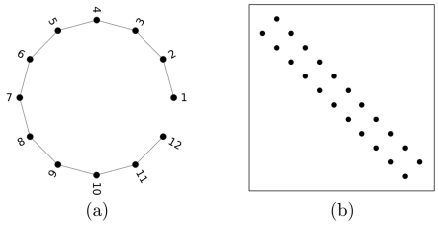


Figure 7: (a) The line graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

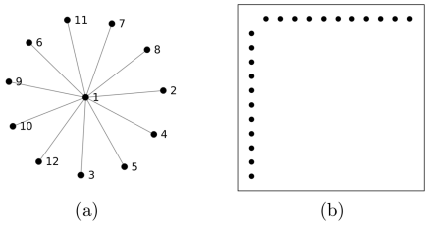


Figure 8: (a) The star graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

6.1.5 The two-star graph

The two-star graph is defined as the graph composed of a pair of star subgraphs, of equal size, whose centers are joined by an edge [Jadbabaie and Olshevsky, 2015]. By this definition, the two-star graph always has an even number of vertices. The two-star graph and its adjacency matrix are provided in Figure 7.

The two-star graph occurs in the context of organizations in which two agents lead their own respective groups, and communicate with one another.

6.1.6 The lollipop graph

The lollipop graph is another graph composed of two simpler subgraphs; in this case, a $n/2$-node line graph joined to a $n/2$-node star. By this definition, the lollipop graph always has an even number of vertices. The lollipop graph and its adjacency matrix are provided in Figure 10.

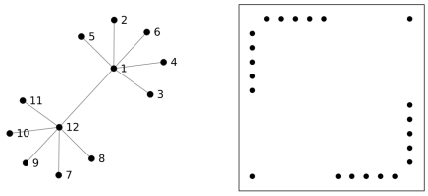


Figure 9: (a) The two-star graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

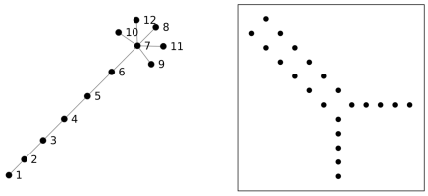


Figure 10: (a) The lollipop graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

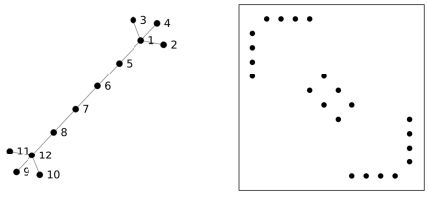
6.1.7 The starry line graph

The starry-line graph, introduced by Jadbabaie and Olshevsky [2015], is a dumbbell shaped graph, constructed by joining two $n/3$-node stars to either end of a $n/3$-node line. By this definition, the vertex count for the lollipop graph is always a multiple of three. The graph and its adjacency matrix are provided in Figure 11.

6.1.8 Two-dimensional grid

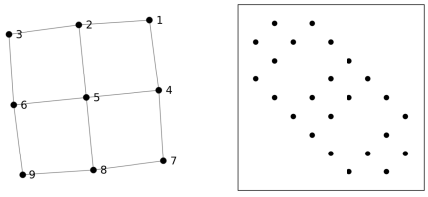
The two-dimensional grid graph with n nodes is essentially $\sqrt{n}$ separate line graphs stacked in the plane, with edges joining each vertex to its opposite number in the adjacent lines. More precisely, the vertex set is given by $\mathcal{V} = \{(i, j) : i = 1, \ldots, \sqrt{n},\ j = 1, \ldots, n\}$, where i denotes the row of a vertex, and j denotes its linear index. On the graph, an edge connects a pair of nodes (i_1, j_1) and (i_2, j_2) if and only if they are separated by a distance of one step on the grid.

That is, they are connected if they are one step apart in the same horizontal row, or if they are a step apart in adjacent rows. This is equivalent to the condition $|i_1 - i_2| + |j_1 - j_2| = 1$ [Jadbabaie and Olshevsky, 2015]. The complete graph with 9 nodes, and no self-loops, is illustrated in Figure 12, along with the zero pattern of its associated adjacency matrix.



(a) (b)

Figure 11: (a) The starry line graph with 12 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.



(a) (b)

Figure 12: (a) The two-dimensional grid graph with 9 vertices and no self-loops, and (b) the zero pattern of its associated adjacency matrix, in which blank entries represent zeros.

6.2 Edge weights

The focus of this section is on a collection of simple edge-weight rules that produce row-stochastic, reversible matrices. It is most natural to speak of these rules in terms of weighted adjacency matrices for a consensus protocol such as the DeGroot linear averaging model.

6.2.1 Equal-neighbor

The first rule to be considered here is also the simplest. In terms of a DeGroot process, each individual i in a social network accords influence to their neighbors, and to themselves. For the equal-neighbor rule, i allots themselves the same amount of influence as each of their neighbors. Explicitly stated, the weights for the equal-neighbor rule are assigned according to

$$p_{ij} = \begin{cases} 1/d_i, & \text{if } (i,j) \in \mathcal{E} \\ 0, & \text{otherwise,} \end{cases}$$

where d_i is the ith element of the degree vector, and $\mathcal{E}$ is the edge set of the graph. Thus, for the DeGroot process, the ith element of $\boldsymbol{x}(t)$ is updated with the average value of i's and their neighbors' opinions at time t [Bullo, 2016].

The equal-neighbor rule produces a symmetric transition matrix for regular undirected graphs. In general, it produces a reversible transition matrix. To see this, first notice that for a transition matrix P with equal-neighbor weights, the consensus eigenvector is given by $\boldsymbol{w}^\top = \frac{1}{2|E|}\begin{bmatrix} d_1 & d_2 & \dots & d_n \end{bmatrix}$, where $|E|$ is the

edge count in the associated graph $\mathcal{G}$ [Bullo, 2016]. Indeed, let $\boldsymbol{P_{01}}$ denote the binary adjacency matrix associated to $\mathcal{G}$, and observe that

$$\begin{aligned}\boldsymbol{w}^\top \boldsymbol{P} &= \frac{1}{2|E|}[d_1, \dots, d_n]\boldsymbol{P} = \frac{1}{2|E|}[d_1, \dots, d_n]\text{diag}(d_1^{-1}, \dots, d_n^{-1})\boldsymbol{P}_{01} \\ &= \frac{1}{2|E|}\mathbb{1}^T\boldsymbol{P}_{01} = \frac{1}{2|E|}[d_1, \dots, d_n] = \boldsymbol{w}^\top.\end{aligned}$$

Regarding the stochasticity of $\boldsymbol{w}^\top$, $\mathbb{1}^T w = \frac{1}{2|E|}\mathbb{1}^T[d_1, \dots, d_n]^T = \frac{1}{2|E|}\sum_{k=1}^{n} d_k = \frac{1}{2|E|}2|E| = 1$, as required by Definition 2.4.

Finally, $\boldsymbol{P}$ is reversible since it satisfies the symmetry condition of Theorem 2.5. Indeed,

$$\begin{aligned}\boldsymbol{D_w P} &= \frac{1}{2|E|}\text{diag}(d_1, \dots, d_n)\boldsymbol{P} \\ &= \frac{1}{2|E|}\text{diag}(d_1, \dots, d_n)[\text{diag}(d_1, \dots, d_n)]^{-1}\boldsymbol{P_{01}} \\ &= \frac{1}{2|E|}\boldsymbol{P_{01}} = \left(\frac{1}{2|E|}\boldsymbol{P_{01}}\right)^\top = (\boldsymbol{D_w P})^\top,\end{aligned}$$

where $\boldsymbol{P_{01}}$ is symmetric since $\mathcal{G}$ is undirected.

For the complete graph, with a full set of self-loops, the equal-neighbor weight matrix is the simple average matrix $\frac{1}{n}\mathbb{1}\mathbb{1}^\top$, so the noiseless process converges to consensus after a single iteration.

6.2.2 Metropolis-Hastings

The next set of weights, more commonly used for sampling from complex probability distributions, and for numerical integration, can also be used in consensus protocols. The weights are assigned to an adjacency matrix with a full set of self-loops according to the rule

$$p_{ij} = \begin{cases} 1/\left(\max\{d_i, d_j\} + 1\right) & \text{if } (i,j) \in \mathcal{E} \\ 1 - \sum_{j \in \mathcal{N}_i} 1/\left(\max\{d_i, d_j\} + 1\right) & i = j \\ 0, & \text{otherwise,} \end{cases}$$

where $\mathcal{N}_i$ is the set of nodes neighboring i [Xiao et al., 2007].

Applied to the adjacency matrix of an undirected graph the Metropolis-Hastings weights produce a symmetric transition matrix.

6.2.3 Jadbabaie & Olshevsky eqs. 33, 34

The next set of weights is a variation on the equal-neighbor weights. No particular name is given for them in [Jadbabaie and Olshevsky, 2015], so in this paper they will be referred to as the 3334 weights. For a given social network and the 3334 weights, each individual grants equal portions of influence to each

of its neighbors, and the other half of its total influence to themselves. That is, in forming its opinion, an agent will weight their own opinion as heavily as the collection of opinions of its neighbors. The weight matrix is constructed in two steps: first the equal-neighbor weights $\tilde{p}_{ij}$ are formed for a connected undirected graph without self-loops, i.e.,

$$\tilde{p}_{ij} = \begin{cases} 1/d_i & (i,j) \in \mathcal{E}, \\ 0 & \text{otherwise;} \end{cases} \tag{6.1}$$

then self-loops are added, and row-stochasticity is enforced, i.e.,

$$\boldsymbol{P} = \frac{1}{2}\boldsymbol{I} + \frac{1}{2}\tilde{\boldsymbol{P}}. \tag{6.2}$$

It is not difficult to see that since the equal-neighbor weights are reversible, so are these.

6.2.4 Random

The final set of weights are sampled from the standard uniform distribution on $[0, 1]$. The general random weight matrix is formed by populating the nonzero elements of the binary adjacency matrix with samples from the uniform distribution, and then normalizing the rows. A bistochastic matrix with random weights can be obtained by repeatedly normalizing the rows, and columns of the general random matrix [Cappellini et al., 2009].

A simple way to obtain a random reversible matrix is to simply create a random symmetric matrix. However, the problem with enforcing symmetry on a row-stochastic (resp. column-stochastic) matrix is that it breaks the stochasticity property of the matrix. It turns out that this can be countered by simply adding a step to the process for generating a random bistochastic matrix, detailed in [Cappellini et al., 2009]. In the revised algorithm, the rows of the matrix are normalized, then the columns, and finally symmetry is enforced by averaging the matrix with its transpose.

7 Simulation

In this section, the theory accumulated above is used to conduct a simple simulation, as well as interpret the results. For each graph type, and each choice of reversible edge weights given in Section 6, the limiting disagreement is computed. This computation is repeated for graphs with 4 to 64 nodes, with slight variations for graphs requiring special node counts (e.g., the two-dimensional grid, and starry line graphs). For the case of random edge weights, the computation is repeated for 10,000 different random matrices for each graph and for each node count. For the other weights, the transition matrix is fixed for each graph type, so just one computation per node count is necessary.

7.1 Execution

Several important ingredients are required in order to compute the limiting disagreement of the noisy consensus iteration for a transition matrix $\boldsymbol{P}$. Specifically, the computation requires the transition matrix itself, the associated consensus eigenvector and MFPT matrix, and specification of the variance of the additive noise. In this simulation, the additive noise is i.i.d., with unit variance and zero mean, so the variance matrix is simply the identity matrix.

For this experiment, the update matrix was generated by modifying the nonzero elements of a binary transition matrix, generated by a separate function. Specifically, the set of functions `adj_<type>.m`, where `type` specifies the graph type, generates these adjacency matrices. As input, these functions require the vertex count n, and instructions on whether or not to include a full set of self-loops in the graph. To facilitate later automation, these network generation functions can be called by way of the driving function `make_adj.m`, which accepts the same input arguments, in addition to the `type` of graph desired.

The update matrix, $\boldsymbol{P}$, is generated by the functions `weight_<rule>.m`, where `rule` specifies the rule to be used in assigning edge weights. As input, these functions take the adjacency matrix $\boldsymbol{A}$. As with the adjacency matrix generation functions, the update matrix generation functions can be called by way of the driver function `adj2update.m`. The driver accepts the adjacency matrix as input along with specification of the stochasticity of the output, and the symmetry of the output. These additional arguments are used to tell teh function whether or not to call the additional functions `make_bistochastic.m`, and `make_symmetric.m`, which change random weight matrices into bistochastic and symmetric matrices, respectively.

The next ingredient for the calculation is the consensus eigenvector of $\boldsymbol{P}$; it is computed by the function `consensus_eigenvector.m`. The function uses the built-in `eigs.m` function of MATLAB to compute the eigenvector of $\boldsymbol{P}$ associated with the eigenvalue 1, and issues a warning if the iteration fails to converge.

The final piece of the computation is the MFPT matrix $\boldsymbol{M}$. This can be computed using either the Sheskin algorithm provided in Section 2.4, or the more accurate EGTH function provided in the appendix of [Hunter, 2016].

Finally, with these elements, the function `limiting_disagreement_<rev>.m` is able to compute the limiting disagreement for the noisy consensus iteration specified. In the event that only the update matrix, and variance of the noise are provided, or any combination of the other arguments is missing, `limiting_disagreement_rev.m` will call their respective functions to obtain them.

In this experiment, the most costly computation was that of obtaining an MFPT matrix. For this reason, the set of 10,000 random matrices for each graph and node count (200,000 total) were computed separately along with their consensus eigenvectors, and MFPT matrices, and saved in a series of `.mat` files. This computation took about 9 hours to complete on a 2008 MacBook with a Core2Duo processor and 2GB of RAM. The functions mentioned above

are provided in the appendix.

7.2 Commentary

The plots of limiting disagreement for each type of graph as a function of the number of vertices are provided in Figures 13, and 16. In the former, the data is provided on logarithmic scales to capture the wide spread of the data for certain weight-graph combinations.

The complete graph has the lowest disagreement, for all weights, of any graph. The disagreement is also bounded, independent of the number of agents, as remarked upon in [Jadbabaie and Olshevsky, 2015]. On all of the other graphs considered here, the limiting disagreement grows with the size of the graph.

The circle and line graphs are nearly identical, the line graph has one less edge than does the circle. The limiting disagreement between the two is quite similar as a result, but it is somewhat higher in all cases for the line graph. Taken together with the nice performance of the complete graph and the fact that the two-dimensional grid has the second lowest disagreement, this seems to indicate that a higher degree of connectedness in a graph corresponds to a reduced level of asymptotic disagreement. Interestingly, the limiting disagreement increases linearly with node count for all choices of weights for the circle, and line graphs.

The star graph is an interesting case, in that there is a vast disparity in the limiting disagreement between the different choices of weights - the equal-neighbor, and 3334 weights perform substantially better than the others. A similar trend is evident for the two-star graph, but it is much less pronounced.

For the lollipop graph, the union of a star and a line, the nice behavior of the equal-neighbor and 3334 weights on the star are each overpowered by their poor performance on the line graph. The starry line graph has similar behavior. The limiting disagreement increases linearly with node count for the lollipop and starry line graphs with random and Metropolis-Hastings weights.

In Figures 15 and 16, the limiting disagreement as a function of node count is plotted for each choice of weights. Interestingly, the star graph, one of the worst-performing graphs for Metropolis-Hastings and random weights, is the best performing graph for 3334 weights. A similar phenomenon is observed for the two-star graph, but it is less pronounced.

Finally, as can be seen in Figure 17, the standard deviation of the disagreement decreases with increasing node count for the complete graph, and two-dimensional grid. For all other graphs the disagreement and its deviation tend to increase.

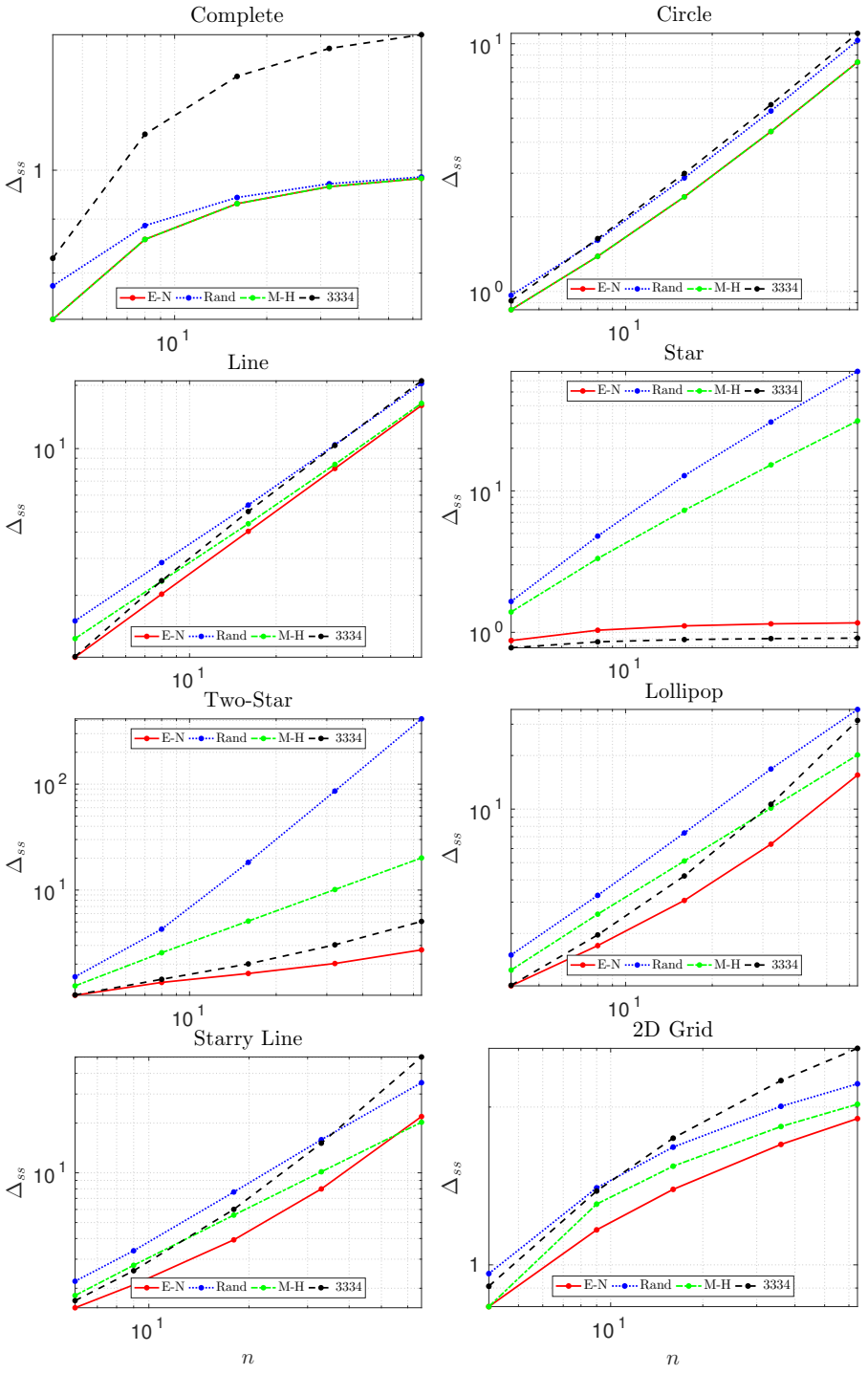


Figure 13: Limiting disagreement as a function of node count for various graphs, on logarithmic scales.

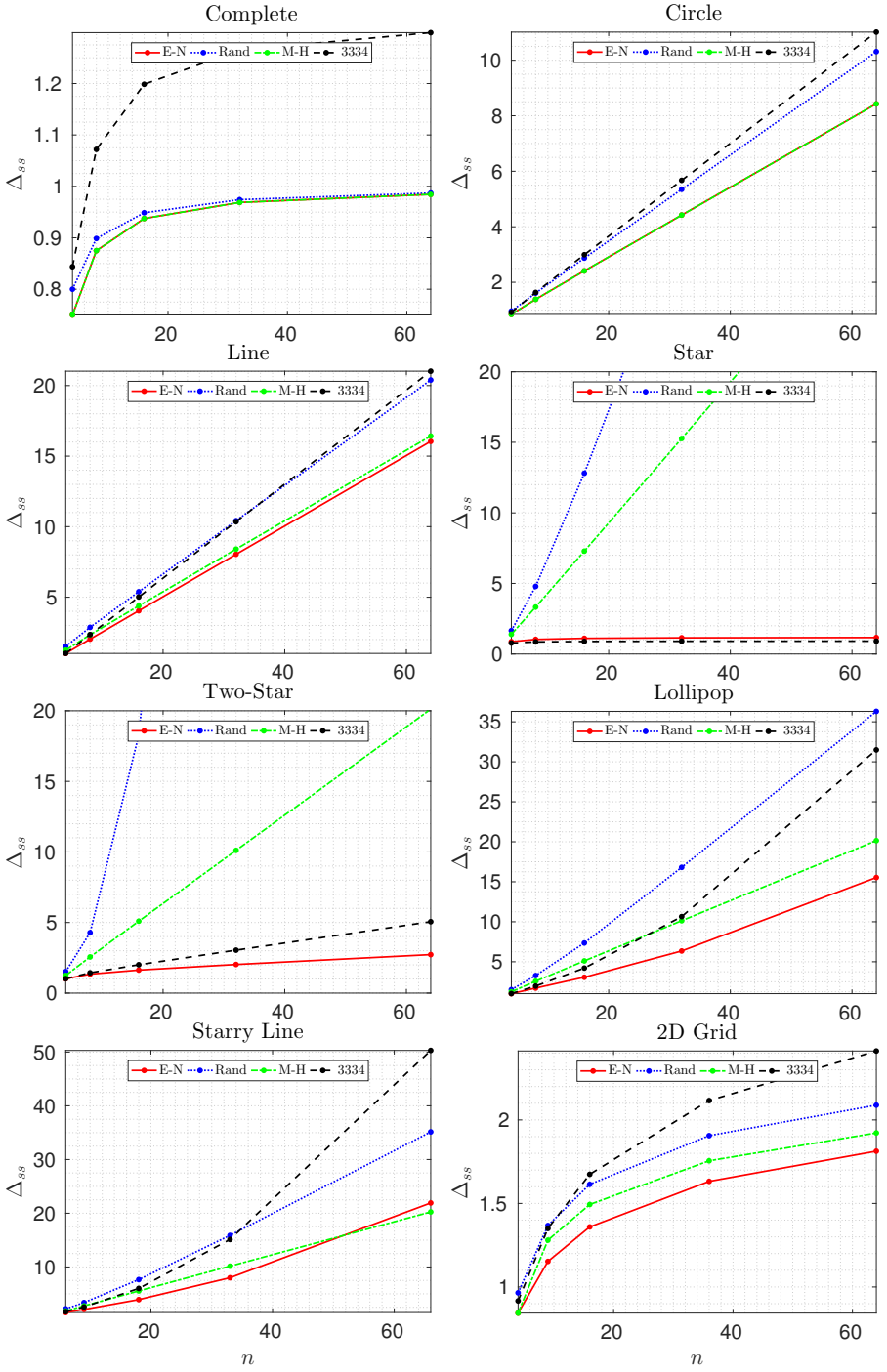


Figure 14: Limiting disagreement as a function of node count for various graphs, on linear scales.

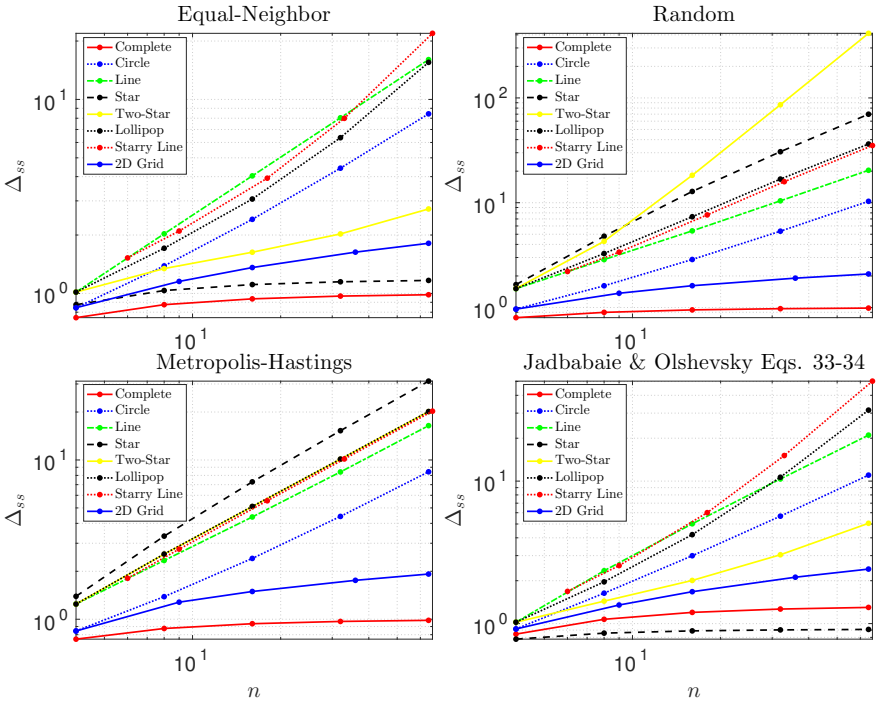


Figure 15: Limiting disagreement as a function of weight rule for various graphs, on logarithmic scales.

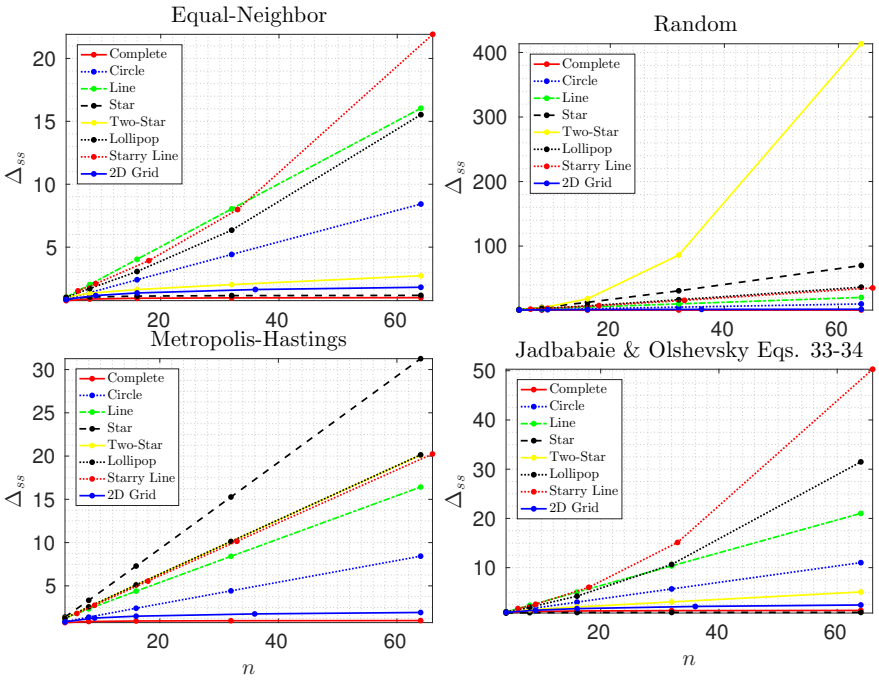


Figure 16: Limiting disagreement as a function of node count for various graphs, on linear scales.

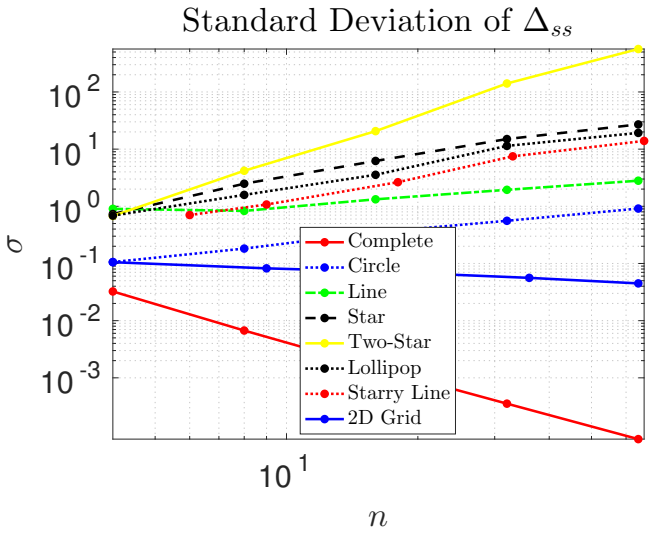


Figure 17: Standard deviation of the limiting disagreement for random graphs as a function of node count, on logarithmic scales.

References

U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. 1998. ISBN 0898714125.

F. Bullo. *Lectures on Network Systems*. Version 0.85, May 2016. URL `http://motion.me.ucsb.edu/book-lns`. With contributions by J. Cortés, F. Dörfler, and S. Martínez.

V. Cappellini, H. J. Sommers, W. Bruzda, and K. Zyczkowski. Random bistochastic matrices. *Journal of Physics A: Mathematical and Theoretical*, 42 (36):365209, 2009.

M. H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

C. M. Grinstead and J. L. Snell. *Introduction to Probability, 2nd edition*. American Mathematical Society, 2006. ISBN 0821894145.

J. P. Hespanha. *Linear Systems Theory*. Princeton University Press, 2009. ISBN 0691140219.

J. J. Hunter. Accurate calculations of stationary distributions and mean first passage times in markov renewal processes and markov chains. *Special Matrices*, 4(1):151–175, 2016.

A. Jadbabaie and A. Olshevsky. On performance of consensus protocols subject to noise: role of hitting times and network structure. *arXiv preprint arXiv:1508.00036*, 2015.

P. Jia, A. MirTabatabaei, N. E. Friedkin, and F. Bullo. Opinion dynamics and the evolution of social power in influence networks. *SIAM Review*, 57(3): 367–397, 2015.

D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers, 4th edition*. John Wiley & Sons, 2007. ISBN 0471745898.

T. J. Sheskin. Computing mean first passage times for a Markov chain. *International Journal of Mathematical Education in Science and Technology*, 26 (5):729–735, 1995.

J. Tsitsiklis. Markov chains i-iii. Lecture Notes for 6.041/6.431, Massachusetts Institute of Technology, 2010.

Wikipedia. Markov chain applications — Wikipedia, the free encyclopedia, 2016. URL `https://en.wikipedia.org/wiki/Markov_chain#Applications`. [Online; accessed 09-December-2016].

L. Xiao, S. Boyd, and S.-J. Kim. Distributed average consensus with least-mean-square deviation. *Journal of Parallel and Distributed Computing*, 67 (1):33–46, 2007.

A Network Generation Functions

A.1 Making an adjacency matrix

This is the driver for the adjacency matrix generating functions in the following sections.

```
function A = make_adj(type, n, varargin)
% This function generates an n-by-n sparse adjacency matrix. Valid
% arguments for type are as follows: 'Complete', 'Circle', 'Line', 'Star',
% 'Two-Star', 'Lollipop', 'Starry Line', '2D Grid'.
% Input:
%         type       - adjacency matrix type
%         n          - number of nodes in graph realized by adj matrix
%         c          - center node, (star graph only). Default: c=1
%         self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%         A          - n-by-n sparse binary adjacency matrix
% Syntax:
%         A = make_adj(type, n)
%         A = make_adj(type, n, self_loops)
%         A = make_adj('Star', n)
%         A = make_adj('Star', n, c)
%         A = make_adj('Star', n, c, self_loops)

switch nargin
    case 1
        error('Not enough input arguments!')
    case 2
        self_loops = 'no self-loops';
        c = 1;
    case 3
        if strcmp(type, 'Star')
            if isnumeric(varargin{1})
                c = varargin{1};
                self_loops = 'no self-loops';
            elseif ischar(varargin{1})
                c = 1;
                self_loops = varargin{1};
            else
                error('Unrecognized input argument for ''Star'' graph.')
            end
        end
        self_loops = varargin{1};
    case 4
        if strcmp(type, 'Star')
            c = varargin{1};
            self_loops = varargin{2};
        end
end

switch type
    case 'Complete'
        A = adj_complete(n, self_loops);
    case 'Circle'
        A = adj_circle(n, self_loops);
    case 'Line'
```

```
51          A = adj_line(n, self_loops);
52      case 'Star'
53          A = adj_star(n, c, self_loops);
54      case 'Two-Star'
55          A = adj_twostar(n, self_loops);
56      case 'Lollipop'
57          A = adj_lollipop(n, self_loops);
58      case 'Starry Line'
59          A = adj_starryline(n, self_loops);
60      case '2D Grid'
61          A = adj_twoDgrid(n, self_loops);
62      otherwise
63          error('Unrecognized input argument: adjacency matrix type.')
64  end

66  return
```

A.1.1 The complete graph

```
1   function A = adj_complete(n,varargin)
2   % This function makes an adjacency matrix A for a complete graph with
3   % n nodes.
4   % Input:
5   %         n           - number of nodes in graph
6   %         self_loops - 'self-loops', or 'no self-loops' (default)
7   % Output:
8   %         A           - adjacency matrix
9   % Syntax:
10  %         A = adj_complete(n)
11  %         A = adj_complete(n,'self-loops')
12  %         A = adj_complete(n,'self-loops')

14  switch nargin
15      case 1
16          A = ones(n)-speye(n);
17      case 2
18          if strcmp(varargin{1}, 'self-loops')
19              A = ones(n);
20          elseif strcmp(varargin{1},'no self-loops')
21              A = ones(n)-speye(n);
22          end
23  end

25  return
```

A.1.2 The circle graph

```
1   function A = adj_circle(n,varargin)
2   % This function makes a sparse adjacency matrix A for a circle graph with n
3   % nodes.
4   % Input:
5   %         n           - number of nodes in graph
6   %         self_loops - 'self-loops', or 'no self-loops' (default)
```

```
% Output:
%         A           - adjacency matrix
% Syntax:
%         A = adj_circle(n)
%         A = adj_circle(n,'self-loops')
%         A = adj_circle(n,'no self-loops')

switch nargin
    case 1
        A = spdiags(ones(n, 4),[-(n-1), -1, 1, n-1], n, n);
    case 2
        if strcmp(varargin{1}, 'self-loops')
            A = adj_circle(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_circle(n);
        end
end

return
```

A.1.3 The line graph

```
function A = adj_line(n,varargin)
% This function makes a sparse adjacency matrix A for a line graph with n
% nodes.
% Input:
%         n           - number of nodes in graph
%         self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%         A           - adjacency matrix
% Syntax:
%         A = adj_line(n)
%         A = adj_line(n,'self-loops')
%         A = adj_line(n,'self-loops')

switch nargin
    case 1
        A = spdiags(ones(n,2), [-1,1], n, n);
    case 2
        if strcmp(varargin{1},'self-loops')
            A = adj_line(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_line(n);
        end
end

return
```

A.1.4 The star graph

```
function A = adj_star(n, varargin)
% This function makes a sparse adjacency matrix A for a star graph with n
% nodes and center node c.
```

```
4  % Input:
5  %        n          - number of nodes in graph
6  %        c          - center node
7  %        self_loops - 'self-loops', or 'no self-loops' (default)
8  % Output:
9  %        A          - adjacency matrix
10 % Syntax:
11 %        A = adj_star(n)
12 %        A = adj_star(n, c)
13 %        A = adj_star(n, c, 'self-loops')
14 %        A = adj_star(n, c, 'no self-loops')

16 % To Do: find out which plan is faster for large matrices.

18 % plan 1: subtract center node from set with setdiff()
19 % A = sparse([setdiff(1:n, c), c*ones(1, n-1)],...
20 %             [c*ones(1, n-1), setdiff(1:n, c)],1 ,n ,n , 2*(n-1));

22 switch nargin
23     case 1
24         A = adj_star(n,1,'no self-loops');
25     case 2
26         if isnumeric(varargin{1})
27             c = varargin{1};
28             self_loops = 'no self-loops';
29         elseif ischar(varargin{1})
30             c = 1;
31             self_loops = varargin{1};
32         else
33             error('Unrecognized input argument for ''Star'' graph.')
34         end
35         A = adj_star(n, c, self_loops);
36     case 3
37         c = varargin{1};
38         self_loops = varargin{2};
39         if c == 1
40             A = sparse([2:n, ones(1, n-1)], ...
41                 [ones(1, n-1), 2:n], 1, n, n, 2*(n-1));
42         elseif c == n
43             A = sparse([1:n-1, c*ones(1, n-1)], ...
44                 [c*ones(1, n-1), 1:n-1], 1, n, n, 2*(n-1));
45         else
46             A = sparse([1:c-1, c+1:n, c*ones(1, n-1)], ...
47                 [c*ones(1, n-1), 1:c-1, c+1:n], 1, n, n, 2*(n-1));
48         end
49 end

51 if strcmp(self_loops, 'self-loops')
52     A = A + speye(n);
53 end

55 return
```

A.1.5 The two-star graph

```
function A = adj_twostar(n,varargin)
% This function makes a sparse adjacency matrix A for a two-star graph with
% n nodes, center nodes 1 and n, and stars of equal size connected at their
% centers.
% adjacency matrix.
% Input:
%        n           - number of nodes in graph. Must be even.
%        self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%        A           - adjacency matrix for two-star graph
% Syntax:
%        A = adj_twostar(n)
%        A = adj_twostar(n,'self-loops')

if mod(n,2) == 1
    error('Dimension of input matrix must be even!!!')
end

switch nargin
    case 1
        A = sparse([ones(1, n/2), n/2+1:n-1], [2:n/2, n*ones(1,n/2)],...
                   1, n, n, 2*n-1);
        A = A + A.';
    case 2
        if strcmp(varargin{1}, 'self-loops')
            A = adj_twostar(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_twostar(n);
        end
end

return
```

A.1.6 The lollipop graph

```
function A = adj_lollipop(n,varargin)
% This function makes a sparse adjacency matrix A for a lollipop graph with
% n nodes, i.e., a graph consisting of a n/2-node line graph, and a
% n/2-node complete graph, connected at nodes n/2 and n/2 +1.
% Input:
%        n           - number of nodes in graph. Must be even.
%        self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%        A           - adjacency matrix for lollipop graph
% Syntax:
%        A = adj_lollipop(n)
%        A = adj_lollipop(n, 'self-loops')
%        A = adj_lollipop(n, 'no self-loops')

if mod(n,2) == 1
    error('Dimension of input matrix must be even!!!')
end

switch nargin
    case 1
        % plan 1: put submatrices on diagonal, connect nodes n/2 and (n/2 +1)
```

```
        A = blkdiag(adj_line(n/2), adj_star(n/2, 1));
        A(n/2, n/2+1) = 1;
        A(n/2+1, n/2) = 1;
    case 2
        if strcmp(varargin{1},'self-loops')
            A = adj_lollipop(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_lollipop(n);
        end
end

return
```

A.1.7 The starry line graph

```
function A = adj_starryline(n,varargin)
% This function makes a sparse adjacency matrix A for a starry line graph
% with n nodes. The graph is made up of a line n/3-node line graph, and two
% n/3-node star graphs, with an edge connecting the center of each star to
% a either end of the line. The result is a dumbbell-shaped graph.
% Input:
%        n           - number of nodes in graph. Must be divisible by 3
%        self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%        A           - adjacency matrix for lollipop graph
% Syntax:
%        A = adj_starryline(n)
%        A = adj_starryline(n,'self-loops')
%        A = adj_starryline(n,'self-loops')

if mod(n,3) ~= 0
    error('Dimension of input matrix must be a multiple of 3!!!')
end

switch nargin
    case 1
        % plan 1: put submatrices on diagonal, add connections individually
        A = blkdiag(adj_star(n/3, 1), adj_line(n/3), adj_star(n/3, n/3));
        A(1, n/3+1) = 1;
        A(n/3+1, 1) = 1;
        A(2*n/3, n) = 1;
        A(n, 2*n/3) = 1;
    case 2
        if strcmp(varargin{1},'self-loops')
            A = adj_starryline(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_starryline(n);
        end
end

return
```

A.1.8 The two-dimensional grid graph

```
function A = adj_twoDgrid(n,varargin)
% This function makes a sparse adjacency matrix A for a two-dimensional
% grid graph with n nodes, where n is a perfect square.
% Input:
%        n          - number of nodes in graph. Must be a perfect square
%        self_loops - 'self-loops', or 'no self-loops' (default)
% Output:
%        A          - adjacency matrix for two-dimensional grid graph
% Syntax:
%        A = adj_twoDgrid(n)
%        A = adj_twoDgrid(n,'self-loops')
%        A = adj_twoDgrid(n, 'no self-loops')

if mod(sqrt(n), 1) ~= 0
    error('Dimension of input matrix must be a perfect square!!!')
end

switch nargin
    case 1
        node_list = [reshape(meshgrid(1:sqrt(n),1:sqrt(n)),[n,1]), ...
            repmat((1:sqrt(n))',sqrt(n),1)];
        e = 0;
        n_edges = 2*sqrt(n)*(sqrt(n)-1);
        edge_list = zeros(n_edges,2);
        for n1 = 1:length(node_list)
            for n2 = n1:length(node_list)
                if abs(node_list(n1,1)-node_list(n2,1))...
                        + abs(node_list(n1,2)-node_list(n2,2))  == 1
                    e = e+1;
                    edge_list(e,:) = [n1, n2];
                end
            end
        end
        I = ind2sub([n,n],edge_list);
        A = sparse(I(:,1), I(:,2), 1, n, n, n_edges);
        A = A + A.';
    case 2
        if strcmp(varargin{1},'self-loops')
            A = adj_twoDgrid(n) + speye(n);
        elseif strcmp(varargin{1},'no self-loops')
            A = adj_twoDgrid(n);
        end
end
return
```

B Transition matrix operations

B.1 Conversion to update matrix

This is the driver for the update matrix generation functions in the following sections.

```
function P = adj2update(A, rule, varargin)
% This function takes an adjacency matrix input (not necessarily binary)
```

```
3  % and outputs a row-stochastic adjacency matrix, with weights assigned
4  % according to the user-input rule. The following weighting rules are
5  % accepted:
6  %  rule             | in each row, each nonzero element _____
7  % -------------------+-------------------------------------------------
8  % 'equal neighbor'  | has the same weight.
9  % 'random'          | is sampled from the uniform distribution on (0,1)
10 % 'M-H'             | has the Metropolis-Hastings weight
11 % '3334'            | has weights from eqs. 33,34 of [AJ-AO:15], see
12 %                   | weight_3334.m for more information.
13 % 'in-degree'       | has weight proportional to in-degree
14 % 'out-degree       | has weight proportional to out-degree
15 % 'centrality'      | has weight proportional to some notion of centrality
16 %                   | (betweenness, closeness, eigenvalue, etc...)
17 %
18 % Input:
19 %       A             - an n-by-n adjacency matrix
20 %       rule          - weighting rule chosen from the table above)
21 %       stochasticity - 'row' (default) or 'bi'
22 %       symmetry      - if asymmetric enter 'asym' (default), else 'sym'.
23 % Output:
24 %       P             - the resulting update matrix
25 % Syntax:
26 %       P = adj2update(A, rule)
27 %       P = adj2update(A, rule, stochasticity)
28 %       P = adj2update(A, rule, stochasticity, symmetry)

30 switch nargin
31     case 1
32         error('Not enough input arguments!')
33     case 2
34         stochasticity = 'row';
35         symmetry = 'asym';
36     case 3
37         stochasticity = varargin{1};
38         symmetry = 'asym';
39     case 4
40         stochasticity = varargin{1};
41         symmetry = varargin{2};
42 end

44 is_reg = is_regular(A);
45 switch rule
46     case 'equal neighbor'
47         if strcmp(symmetry,'sym') && ~is_reg
48             error(['Can''t make non-regular update matrix'...
49                    ' symmetric with equal neighbor weights.']);
50         end
51         if strcmp(stochasticity,'bi') && ~is_reg
52             error(['Can''t make non-regular update matrix'...
53                    ' bistochastic with equal neighbor weights.']);
54         end
55         P = weight_equal_neighbor(A);
56     case 'random'
57         P = weight_random(A);
58         if strcmp(symmetry,'sym')
59             P = make_symmetric(P);              % use default TOL & NMAX
```

```
60          elseif strcmp(stochasticity,'bi') && ~strcmp(symmetry,'sym')
61              P = make_bistochastic(P);           % use default TOL & NMAX
62          end
63      case 'M-H'
64          P = weight_metropolis_hastings(A);
65      case '3334'
66          P = weight_3334(A);
67          if strcmp(stochasticity,'bi') && ~is_reg
68              error(['Can''t make non-regular update matrix'...
69                     ' bistochastic with 33,34 weights.']);
70          end
71      case 'in-degree'
72          error('Selected weight rule under construction.')
73      case 'out-degree'
74          error('Selected weight rule under construction.')
75      case 'centrality'
76          error('Selected weight rule under construction.')
77  end

79  return
```

B.1.1 Equal neighbor weights

```
1   function P = weight_equal_neighbor(A)
2   % This function takes the binary adjacency matrix A for a graph G and makes
3   % a weighted adjacency matrix using the equal-neighbor rule. That is, for a
4   % given node, the self-weight and weights accorded to its out-neighbors are
5   % equal. As an example, Sam gives as much credit to the thoughts of each of
6   % his friends as he does to himself. If A is symmetric, or equivalently, if
7   % G is undirected, then P is symmetric and doubly stochastic.
8   % Input:
9   %       A - n-by-n binary adjacency matrix w/ or w/o self-loops
10  % Output:
11  %       P - n-by-n weighted adjacency matrix with self-loops
12  % Syntax:
13  %       P = weight_equal_neighbor(A)

15  n = length(A);
16  if ~has_selfloops(A)
17      A = A + speye(n);                                   % add self-loops
18  end
19  [i,j] = ind2sub([n,n], find(A));            % find nonzero elements
20  P = sparse(i,j,1,n,n,nnz(A));               % replace with ones
21  P = spdiags(1./sum(P,2), 0, n, n)*P;

23  return
```

B.1.2 Random weights

```
1   function P = weight_random(A)
2   % This function takes the binary adjacency matrix A for a graph G and makes
3   % it row-stochastic, with elements sampled from the uniform distribution on
4   % (0,1). If A is symmetric, or equivalently, if G is undirected, then P is
```

```
% symmetric and doubly stochastic.
% Input:
%       A - n-by-n binary adjacency matrix w/ or w/o self-loops
% Output:
%       P - n-by-n weighted adjacency matrix with self-loops
% Syntax:
%       P = weight_random(A)

n = length(A);
if ~has_selfloops(A)                                % add self-loops
    A = A + speye(n);                               % at every node
end
[i,j] = ind2sub([n,n], find(A));                    % find nonzero elements
P = sparse(i,j,rand(length(i),1),n,n,nnz(A));       % replace with rand #s
P = spdiags(1./sum(P,2), 0, n, n)*P;                % make row-stochastic

return
```

B.1.3 Metropolis-Hastings weights

```
function P = weight_metropolis_hastings(A)
% This function takes the binary adjacency matrix A for a graph G and makes
% a weighted adjacency matrix using the Metropolis-Hastings rule.
% Input:
%       A - n-by-n binary adjacency matrix w/ self-loops at every node
% Output:
%       P - n-by-n weighted adjacency matrix
% Syntax:
%       P = weight_metropolis_hastings(A)

n = length(A);

if nnz(diag(A)) < n
    error('Input adjacency matrix must have self-loops at every vertex.')
end

d = sum(A,2);
P = A-diag(diag(A));
for i = 1:n
    for j = 1:n
        if j ~= i && A(i,j)
            P(i,j) = 1/( max([d(i),d(j)]) );
        end
    end
end
P = P + diag(1-sum(P,2));

return
```

B.1.4 Jadbabaie & Olshevsky eqs. 33, 34

```
function P = weight_3334(A)
% This function takes the binary adjacency matrix A for a graph G and makes
```

```
% a weighted adjacency matrix using equations 33 and 34 in the paper:
%   A. Jadbabaie and A. Olshevsky. On performance of consensus protocols
%   subject to noise: role of hitting times and network structure. arXiv
%   preprint arXiv:1508.00036, 2015.
% Input:
%       A - n-by-n binary adjacency matrix
% Output:
%       P - n-by-n row-stochastic, reversible transition matrix
% Syntax:
%       P = weight_3334(A)

n = length(A);

if any(diag(A))                                     % Remove self-loops
    A = A-spdiags(diag(A),0,n,n);
end

P = 0.5*spdiags(1./sum(A,2),0,n,n)*A + 0.5*speye(n);    % Do Eqs. 33 & 34

return
```

B.1.5 Enforcing bistochasticity

```
function [B, steps] = make_bistochastic(P, varargin)
% This function implements Sinkhorn's 1964 iteration for computing a
% bistochastic matrix.
% For more: see
%   V. Cappellini, H. J. Sommers, W. Bruzda, and K. Zyczkowski. Random
%   bistochastic matrices. Journal of Physics A: Mathematical and
%   Theoretical, 42(36):365209, 2009.
% Input:
%       P     - update matrix to be made bistochastic
%       TOL   - tolerance on row & column sums, default: TOL=1e-15/n
%       NMAX  - maximum number of iterations, default:5e2
% Output:
%       B     - bistochastic update matrix
%       isbi  - Boolean, is B bistochastic?
%       steps - number of iterations
% Syntax:
%                   B = make_bistochastic(P)
%                   B = make_bistochastic(P, TOL, NMAX)
%       [B, steps] = make_bistochastic(P)
%       [B, steps] = make_bistochastic(P, TOL, NMAX)

switch nargin
    case 1
        TOL = 1e-15;
        NMAX = 5e2;
    case 3
        TOL = varargin{1};
        NMAX = varargin{2};
end

B = P;
steps = 0;
err = realmax;
```

```
34 while err > TOL && steps < NMAX
35     steps = steps+1;
36     B = diag(1./sum(B,2))*B;                    % make row-stochastic
37     B = B*diag(1./sum(B,1));                    % make column-stochastic
38     err = full(max([sum(B,1) sum(B,2).'])) -1;
39 end

41 if err > TOL || steps >= NMAX
42     warning('Iteration failed to converge!')
43 end

45 return


48 %      rowsum = sum(B,2), colsum = sum(B,2), steps_remaining = NMAX-steps
```

B.1.6 Enforcing symmetry

```
1  function [S] = make_symmetric(P,varargin)
2  % Given a weighted update matrix P, as from adj2update(), this function
3  % makes the weight matrix symmetric. Note that this can only be done for
4  % update matrices whose binary adjacency matrices are symmetric. This
5  % function uses Sinkhorn's 1964 iteration for computing a
6  % bistochastic matrix, but with the added step of enforcing symmetry at the
7  % end of each iteration.
8  % For more about making a bistochastic matrix see:
9  %   V. Cappellini, H. J. Sommers, W. Bruzda, and K. Zyczkowski. Random
10 %   bistochastic matrices. Journal of Physics A: Mathematical and
11 %   Theoretical, 42(36):365209, 2009.
12 % Input:
13 %        P    - weighted update matrix
14 %        TOL  - tolerance for make_bistochastic(), e.g. 1e-5, default is
15 %               make_bistochastic()'s default (1e-15).
16 %        NMAX - maximum number of iterations for make_bistochastic(),
17 %               default is make_bistochastic()'s default (5e2).
18 % Output:
19 %        S        - symmetric weight matrix
20 % Syntax:
21 %        S = make_symmetric(P)
22 %        S = make_symmetric(P,TOL,NMAX)

24 switch nargin
25     case 1
26         TOL = 1e-15;
27         NMAX = 5e2;
28     case 3
29         TOL = varargin{1};
30         NMAX = varargin{2};
31 end

33 S = P;
34 steps = 0;
35 err = realmax;
36 while err > TOL && steps < NMAX
37     steps = steps+1;
38     S = diag(1./sum(S,2))*S;                    % make row-stochastic
```

```
39      S = S*diag(1./sum(S,1));                    % make column-stochastic
40      S = 0.5*(S+S.');                            % make symmetric
41      err = full(max([sum(S,1) sum(S,2).'])) -1;
42  end

44  if err > TOL || steps >= NMAX
45      warning('Iteration failed to converge!')
46  end

48  return
```

B.2 Hitting times

```
1   function M = hitting_time(P)
2   % This function computes the matrix of hitting times, M, for an ergodic
3   % Markov chain, given input probability transition matrix P. The
4   % computation is carried out using the method of matrix reduction
5   % demonstrated in the following paper:
6   %   T. J. Sheskin. Computing mean first passage times for a markov chain.
7   %   International Journal of Mathematical Education in Science and
8   %   Technology, 26(5):729 735, 1995.
9   % Input:
10  %       P - r-by-r, irreducible, stochastic matrix
11  % Output:
12  %       M - r-by-r matrix of mean first passage times
13  % Syntax:
14  %       M = hitting_time(P)

16  r = length(P);
17  N = r-1;
18  M = zeros(r,r);

20  for j = 1:r
21      if j > 1 && j < r
22          Q = P([1:j-1,j+1:r],[1:j-1,j+1:r]);
23      elseif j == 1;
24          Q = P(2:r,2:r);
25      elseif j == r
26          Q = P(1:r-1,1:r-1);
27      end
28      G = [zeros(N,1), eye(N,N); ones(N,1), Q];

30      k = N;
31      while k > 0
32          T(1:N+k-1,1:k) = G(1:N+k-1,1:k);          % plan 1,2,3
33          R(1,1:k) = G(N+k,1:k);
34          U(1:N+k-1,1) = G(1:N+k-1,k+1);
35          Q = G(N+k,k+1);
36          G(1:N+k-1,1:N+k-2-1) = T(1:N+k-1,1:k) + U(1:N+k-1,1)*R(1,1:k)/(1-Q);
37          k = k-1;
38      end

40      if j == 1
41          M(2:r,j) = G(1:N,1);
42      elseif j == r
```

```
43          M(1:N,j) = G(1:N,1);
44      else
45          M([1:j-1,j+1:r],j) = G(1:N,1);
46      end
47  end
48  return
```

C Consensus protocols

C.1 Consensus iteration

```
1   function [X, varargout] = consensus_iteration(P,x0,varargin)
2   % This function implements the noiseless consensus iteration with update
3   % matrix W, initial opinion vector x0, and iterating on x until each of its
4   % elements have difference no larger than TOL times the initial maximum
5   % difference, or until the number of iterations reaches NMAX. If the
6   % consensus flag is true, then the iteration process will be bypassed, and
7   % the function will return the consensus opinion of the group, if it
8   % exists. In this case, the output consensus-reached flag on the will be
9   % set to true.
10  % Input:
11  %       W    - row-stochastic, n-by-n update matrix
12  %       x0   - n-by-1 vector of initial opinions
13  %       TOL  - exit iteration when difference between each pair of
14  %              successive iterates is less than or equal to TOL times the
15  %              initial difference. By default, TOL = 1e-3.
16  %       NMAX - exit iteration after number of iterations reaches NMAX. By
17  %              default, NMAX = 1e2.
18  %       BI   - Bypass iteration? If so, BI = 1, elsewise BI = 0. By
19  %              default, BI = 0.
20  % Output:
21  %       X      - matrix containing each iterate of the opinion x
22  %       CR     - consensus reached? If so, CR = 1, elsewise CR = 0.
23  %       steps  - number of steps taken
24  %       w      - the left-eigenvector of P with eigenvalue 1, normalized to
25  %                have unit sum.
26  % Syntax:
27  %                    X = consensus_iteration(P,x0)
28  %                    X = consensus_iteration(P,x0,TOL,NMAX)
29  %                    X = consensus_iteration(P,x0,TOL,NMAX,BI)
30  %                [X CR] = consensus_iteration(P,x0)
31  %                [X CR] = consensus_iteration(P,x0,TOL,NMAX)
32  %                [X CR] = consensus_iteration(P,x0,TOL,NMAX,BI)
33  %          [X CR steps] = consensus_iteration(P,x0)
34  %          [X CR steps] = consensus_iteration(P,x0,TOL,NMAX)
35  %          [X CR steps] = consensus_iteration(P,x0,TOL,NMAX,BI)
36  %       [X CR steps w] = consensus_iteration(P,x0)
37  %       [X CR steps w] = consensus_iteration(P,x0,TOL,NMAX)
38  %       [X CR steps w] = consensus_iteration(P,x0,TOL,NMAX,BI)

40  switch nargin
41      case 2                  % Given: P, x0                    Set: TOL, NMAX, BI
42          TOL = 1e-3;
43          NMAX = 1e2;
```

```
44          BI = 0;
45      case 4                  % Given P, x0, TOL, NMAX        Set: BI
46          TOL = varargin{1};
47          NMAX = varargin{2};
48          BI = 0;
49      case 5                  % Given P, x0, TOL, NMAX, BI    Set: nothing
50          TOL = varargin{1};
51          NMAX = varargin{2};
52          BI = varargin{3};
53      otherwise
54          error('Unsupported number of input arguments.')
55  end

57  n = length(P);

59  if nargout == 4 || BI == 1
60      [V,D] = eigs(P.');                          % Eigenvals. & left eigenvecs.
61      [DR,DC] = find(D==max(max(abs(D))));        % Locate dominant eigenval.
62      if D(DR,DC) >= 1-1e3*eps && D(DR,DC) <= 1+1e3*eps
63          CR = 1;                                 % Consensus reached if
64      else                                        % dominant eigenvalue = +1
65          CR =0;
66      end
67      w = V(:,DC);                        % "dominant" eigenvec.
68      w = w/(ones(1,n)*w);                % normalized to have unit sum
69      X = w.'*x0*ones(n,1);               % consensus value
70      steps = 0;                          % iterated 0 times. Could use 2nd
71                                          % largest eigenval. to estimate number
72                                          % of steps to consensus.
73  end

75  env0 = max(x0)-min(x0);
76  if BI == 0                          % Don't bypass iteration
77      X = zeros(n,NMAX+1);
78      steps = 1;
79      X(:,steps) = x0;
80      CR = 0;
81      while steps <= NMAX && ~CR
82          steps = steps + 1;
83          X(:,steps) = P*X(:,steps-1);
84          env = max(X(:,steps))-min(X(:,steps));
85          if env > TOL*env0
86              CR = 0;
87          else
88              CR = 1;
89          end
90      end
91      X = X(:,1:steps);
92      steps = steps-1;
93  end

95  if nargout >= 2
96      varargout{1} = CR;
97  end
98  if nargout >= 3
99      varargout{2} = steps;
100 end
```

```
101 if nargout >= 4
102     varargout{3} = w;
103 end

105 return
```

C.2 Noisy consensus iteration

```
1  function [X, varargout] = noisy_consensus_iteration(P,x0,avg,Var,varargin)
2  % This function implements the noisy consensus iteration with update matrix
3  % P, initial opinion vector x0, and iterating on x. The iteration is
4  % terminated when the expected value of the iterate, Ex, reaches its
5  % consensus value, i.e., when each pair of entries of Ex have difference no
6  % larger than TOL times the initial maximum difference, or until the number
7  % of iterations reaches NMAX. If the consensus flag is true, then the
8  % iteration process will be bypassed, and the function will return the
9  % consensus expected opinion of the group, if it exists. In this case, the
10 % output consensus-reached flag on the will be set to true.
11 % Input:
12 %       P    - row-stochastic, n-by-n update matrix
13 %       x0   - n-by-1 vector of initial opinions
14 %       avg - mean of additive noise vector, v
15 %       var  - variance of additive noise vector, v
16 %       TOL  - exit iteration when difference between each pair of
17 %              successive iterates Ex is less than or equal to TOL times
18 %              the initial difference. By default, TOL = 1e-3.
19 %       NMAX - exit iteration after number of iterations reaches NMAX. By
20 %              default, NMAX = 1e2.
21 %       BI   - Bypass iteration? If so, BI = 1, elsewise BI = 0. By
22 %              default, BI = 0.
23 % Output:
24 %       X     - matrix containing each iterate of the opinion x
25 %       Ex    - vector containing each iterate, Ex, of the mean opinion x
26 %       CR    - consensus Ex reached? If so, CR = 1, elsewise CR = 0.
27 %       steps - number of steps taken
28 %       w     - the left-eigenvector of P with eigenvalue 1, normalized to
29 %               have unit sum.
30 % Syntax:
31 %                          X = consensus_iteration(P,x0,mean,var)
32 %                          X = consensus_iteration(P,x0,mean,var,TOL,NMAX)
33 %                          X = consensus_iteration(P,x0,mean,var,TOL,NMAX,BI)
34 %                     [X Ex] = consensus_iteration(P,x0,mean,var)
35 %                     [X Ex] = consensus_iteration(P,x0,mean,var,TOL,NMAX)
36 %                     [X Ex] = consensus_iteration(P,x0,mean,var,TOL,NMAX,BI)
37 %                  [X Ex CR] = consensus_iteration(P,x0,mean,var)
38 %                  [X Ex CR] = consensus_iteration(P,x0,mean,var,TOL,NMAX)
39 %                  [X Ex CR] = consensus_iteration(P,x0,mean,var,TOL,NMAX,BI)
40 %            [X Ex CR steps] = consensus_iteration(P,x0,mean,var)
41 %            [X Ex CR steps] = consensus_iteration(P,x0,mean,var,TOL,NMAX)
42 %            [X Ex CR steps] = consensus_iteration(P,x0,mean,var,TOL,NMAX,BI)
43 %         [X Ex CR steps w] = consensus_iteration(P,x0,mean,var)
44 %         [X Ex CR steps w] = consensus_iteration(P,x0,mean,var,TOL,NMAX)
45 %         [X Ex CR steps w] = consensus_iteration(P,x0,mean,var,TOL,NMAX,BI)

47 switch nargin
```

```
48      case 4                  % Given: P, x0, v                  Set: TOL,NMAX,BI
49          TOL = 1e-3;
50          NMAX = 5e5;
51          BI = 0;
52      case 6                  % Given P, x0, v, TOL, NMAX        Set: BI
53          TOL = varargin{1};
54          NMAX = varargin{2};
55          BI = 0;
56      case 7                  % Given P, x0, v, TOL, NMAX, BI    Set: nothing
57          TOL = varargin{1};
58          NMAX = varargin{2};
59          BI = varargin{3};
60      otherwise
61          error('Unsupported number of input arguments.')
62  end

64  n = length(P);

66  if nargout == 5 || BI == 1
67      [w, lambda] = eigs(P.',1);                          % Compute consensus
68      w = w/sum(w);                                       % eigenvector
69      if lambda >= 1-1e3*eps && lambda <= 1+1e3*eps
70          CR = 1;
71      else
72          CR = 0;
73      end
74      Ex = w.'*mean(x0)*ones(n,1);        % consensus value
75      steps = 0;                          % iterated 0 times. Could use 2nd
76                                          % largest eigenval. to estimate number
77                                          % of steps to consensus.
78      X = [];
79  end

81  env0 = max(x0)-min(x0);
82  if BI == 0                              % Don't bypass iteration
83      X = zeros(n,NMAX+1);
84      Ex = zeros(n,NMAX+1);
85      steps = 1;
86      X(:,steps) = x0;
87      Ex(:,steps) = x0;
88      CR = 0;
89      while steps <= NMAX && ~CR
90          steps = steps + 1;
91          v = avg + sqrt(Var)*randn(n,1);
92          X(:,steps) = P*X(:,steps-1)+v;
93          Ex(:,steps) = steps/(steps+1)*Ex(:,steps-1) + 1/(steps+1)*X(:,steps);
94          env = max(Ex(:,steps))-min(Ex(:,steps));

96          if env/steps > TOL*env0 %|| abs(mean(Ex(:,steps)-x0)) > TOL  %Ex(steps) > TOL
97              CR = 0;
98          else
99  %               mean(Ex(:,steps)-x0)
100             CR = 1;
101         end
102     end
103     X = X(:,1:steps);
```

```
105     % Slightly better way of computing Ex
106     Ex = cumsum(X,2)*spdiags(1./(1:steps).',0,steps,steps);

108 %     Ex = Ex(:,1:steps);
109     steps = steps-1;
110 end

112 if nargout >= 2
113     varargout{1} = Ex;
114 end
115 if nargout >= 3
116     varargout{2} = CR;
117 end
118 if nargout >= 4
119     varargout{3} = steps;
120 end
121 if nargout >= 5
122     varargout{4} = w;
123 end

125 return
```

D Limiting Disagreement

D.1 Disagreement

```
1  function [D,Du] = disagreement(x,w)
2  % This function computes the total mean-square deviation of opinions from
3  % the consensus value of the noiseless consensus iteration with initial
4  % condition x(t). The deviations are weighted by the stationary
5  % distribution of the update matrix P, or uniformly, given the matrix x
6  % whose columns are the iterated opinion vectors x(t) from
7  % noisy_consensus_iteration.m, and the stationary distribution of P.
8  % Input:
9  %        x   - n-by-ntimes matrix with columns of iterated opinions x(1),
10 %              ...,x(ntimes). x is generated by noisy_consensus_iteration().
11 %        w   - n-by-1 stochastic eigenvector of P, associated w/ the
12 %              dominant eigenvalue 1.
13 % Output:
14 %        D     - ntimes-by-1 vector of weighted mean-square deviations
15 %        Du    - ntimes-by-1 vector of uniformly weighted mean-square
16 %                deviations
17 % Syntax:
18 %        [D,Du] = disagreement(x,w)

20 [n, ntimes] = size(x);
21 w = reshape(w,1,n);
22 Z = (speye(n)-repmat(w,n,1))*x;          % set of deviation vectors
23 Sigma = zeros(n,n,ntimes);               % deviation second moment matrix
24 Du = zeros(ntimes,1);                    % uniformly weighted deviation
25 D = zeros(ntimes,1);                     % w-weighted deviation

27 Sigma(:,:,1) = Z(:,1)*Z(:,1).';              % initial deviations
28 Du(1) = sum(diag(Sigma(:,:,1)),1)/n;         %
```

```
29 D(1) = w*diag(Sigma(:,:,1));                    %

31 zzt = zeros(n,n,ntimes);                        % 3D array with pages
32 for t = 1:ntimes                                % z(t)*z(t).',
33     zzt(:,:,t) = Z(:,t)*Z(:,t).';               % i.e., squared deviation
34 end

36 % Expected value of squared-deviation at each time
37 Sigma = cumsum(zzt,3).*repmat(reshape(1./(1:ntimes),[1 1 ntimes]),[n n]);

39 for t = 2:ntimes
40 %     Sigma(:,:,t) = (t-1)/t*Sigma(:,:,t-1) + 1/t*Z(:,t)*Z(:,t).';
41     Du(t) = full(sum(diag(Sigma(:,:,t))))/n;        % total mean-square
42     D(t) = full(w*diag(Sigma(:,:,t)));              % deviations
43 end

45 return
```

D.2 Limiting Disagreement

D.2.1 Symmetric transitions

```
1  function [Du_ss,w] = limiting_disagreement_sym(P,Var,varargin)
2  % This function computes the uniformly weighted limiting disagreement of
3  % the noisy consensus iteration with symmetric, primitive, stochastic
4  % weight matrix P, additive noise term v, with zero mean & variance Var,
5  % and consensus eigenvector w of P.
6  % For more, see:
7  %   L. Xiao, S. Boyd, and S.-J. Kim. Distributed average consensus with
8  %   least- mean-square deviation. Journal of Parallel and Distributed
9  %   Computing, 67(1):33  46 , 2007.
10 % Input:
11 %        P   - n-by-n symmetric, primitive, stochastic weight matrix
12 %        Var - the scalar variance of the noise term in the noisy consensus
13 %              iteration.
14 %        w   - n-by-1 stochastic eigenvector of P, associated w/ the
15 %              dominant eigenvalue 1.
16 % Output:
17 %        Du_ss - steady-state uniformly weighted mean-square deviation
18 %        w     - consensus eigenvector
19 % Syntax:
20 %             Du_ss = limiting_disagreement_rev(P,Var)
21 %             Du_ss = limiting_disagreement_rev(P,Var,w)
22 %        [Du_ss w] = limiting_disagreement_rev(P,Var)

24 n = size(P,1);

26 switch nargin
27     case 2                                          % need to compute w
28         [w,~,failed_to_converge] = eigs(P.',1);
29         if failed_to_converge
30             warning(['Consensus eigenvalue iteration failed to '...
31                 'converge. Setting Du_ss = NaN.']);
32             Du_ss = NaN;
33             return
```

```
        end
        w = w.'/sum(w);                                    % make stochastic
    case 3                                                 % w is given
        w = reshape(varargin{1},1,n);
end

[~,lambda,failed_to_converge] = eigs(P,n-1,'SM');    % n-1 smallest eigs.
if failed_to_converge
    warning(['Consensus eigenvalue iteration failed to '...
             'converge. Setting D_ss = NaN.']);
    Du_ss = NaN;
    return
end
Du_ss = sum(1./(1-diag(lambda).^2),1)*Var/n;

% This method is O(n^3). An O(n^2) method is possible, as mentioned in
% Section II.C of the above article.
return
```

D.2.2 Reversible transitions

```
function [D_ss,w,M] = limiting_disagreement_rev(P,Var,varargin)
% This function computes the limiting disagreement (with weights from the
% stationary distribution) of the noisy consensus iteration with
% reversible, primitive, row-stochastic weight matrix P, additive noise
% term v, with zero mean & variance Var, and consensus eigenvector w of P.
% Input:
%        P   - n-by-n reversible, primitive, row-stochastic weight matrix
%        Var - the variance of the noise term in the noisy consensus
%              iteration. May be a scalar (in the case of i.i.d. noise), a
%              vector (in the case of uncorrelated noise), or a matrix (in
%              the case of correlated noise).
%        w   - n-by-1 stochastic eigenvector of P, associated w/ the
%              dominant eigenvalue 1.
%        M   - n-by-n mean first passage time matrix
%        Mm  - 'Sheskin', or 'EGTH' algorithm for computing MFPT matrix.
%              default: 'Sheskin'.
% Output:
%        D_ss  - steady-state weighted mean-square deviation
%        w     - consensus eigenvector
% Syntax:
%                    D_ss = limiting_disagreement_rev(P,Var)
%                    D_ss = limiting_disagreement_rev(P,Var,w)
%                    D_ss = limiting_disagreement_rev(P,Var,w,M)
%                    D_ss = limiting_disagreement_rev(P,Var,w,M,Mm)
%                  [D_ss,w] = limiting_disagreement_rev(P,Var)
%                  [D_ss,w] = limiting_disagreement_rev(P,Var,w)
%                  [D_ss,w] = limiting_disagreement_rev(P,Var,w,M)
%                  [D_ss,w] = limiting_disagreement_rev(P,Var,w,M,Mm)
%                [D_ss,w,M] = limiting_disagreement_rev(P,Var)
%                [D_ss,w,M] = limiting_disagreement_rev(P,Var,w)
%                [D_ss,w,M] = limiting_disagreement_rev(P,Var,w,M)
%                [D_ss,w,M] = limiting_disagreement_rev(P,Var,w,M,Mm)

n = size(P,1);
```

```
w = []; M = []; Mm = [];
for i = 1:length(varargin)
    if ischar(varargin{i})
        Mm = varargin{i};
    elseif isvector(varargin{i})
        w = reshape(varargin{i},1,n);
    elseif ismatrix(varargin{i})
        M = varargin{i};
    end
end

if isempty(M)
    if isempty(Mm) || strcmp(Mm,'Sheskin')
        M = hitting_time(P);
    elseif strcmp(Mm,'EGTH')
        M = hitting_time_EGTH(P^2);
    else
        error('Unrecognized MFPT method chosen.')
    end
end

if isempty(w)
    [w,failed_to_converge] = consensus_eigenvector(P);
    if failed_to_converge
        warning(['Consensus eigenvalue iteration failed to '...
            'converge. Setting D_ss = NaN.']);
        D_ss = NaN;
        return
    end
end

[Sr,Sc] = size(Var);
if isequal([Sr,Sc],[1,1])                    % noise terms are i.i.d.
    D_ss = Var*sum(w*bsxfun(@times,M,w.^2),2);
elseif isequal([Sr,Sc],[n,1])                % noise terms are uncorrelated
    Var = spdiags(Var,0,n,n);
    Dw = spdiags(w.',0,n,n);
    A = M*Dw*Var*Dw;
    TrA = 0;
    D_ss = sum(w*A,2)-TrA;
elseif isequal([Sr,Sc],[n,n])                % noise terms are correlated
    Dw = spdiags(w.',0,n,n);
    A = M*Dw*Var*Dw;
    TrA = trace(A);
    D_ss = sum(w*A,2)-TrA;
end

return
```