# A Tutorial on the Structure of Distributed Optimization Algorithms

Bryan Van Scoy[1] Laurent Lessard[2]

## Abstract

We consider the distributed optimization problem for a multi-agent system. Here, multiple agents cooperatively optimize an objective by sharing information through a communication network and performing computations. In this tutorial, we provide an overview of the problem, describe the structure of its algorithms, and use simulations to illustrate some algorithmic properties based on this structure.

## 1 Introduction

Consider a group of agents that are connected together in a communication network, where each agent is capable of communicating with other agents using the network and performing local computations. For instance, each agent may be a computing node, robot, or mobile sensor.

As an illustrative example, consider the problem of large-scale machine learning. Here, each agent is a computing unit with access to a set of data, and the agents seek to cooperatively build a global model that fits all of the data [1, 2]. Let $n$ denote the number of agents, and let $f_i$ and $y_i$ denote the loss function and model parameters associated with agent $i \in \{1, \dots, n\}$. To construct a cohesive global model, we can minimize the total loss over all agents subject to the agents agreeing on the model. This can be formulated as the optimization problem

$$\text{minimize} \quad \sum_{i=1}^{n} f_i(y_i) \tag{1a}$$

$$\text{subject to} \quad y_1 = y_2 = \ldots = y_n. \tag{1b}$$

One approach to solve this problem is for all agents to send their data to a central server and have the server solve the problem to build the model. Some issues with this centralized approach are that i) the computations on the server scale with the number of agents, ii) the system is fragile in that failure of the central server causes the entire system to fail, and iii) data must be transmitted directly over the network and is therefore not private.

[1]Department of Electrical and Computer Engineering, Miami University, OH 45056, USA. Email bvanscoy@miamioh.edu

[2]Dept. of Mechanical and Industrial Eng., Northeastern University, MA 02115, USA. Email l.lessard@northeastern.edu

The work of L. Lessard was supported by the National Science Foundation under Grants No. 2136945 and 2139482.

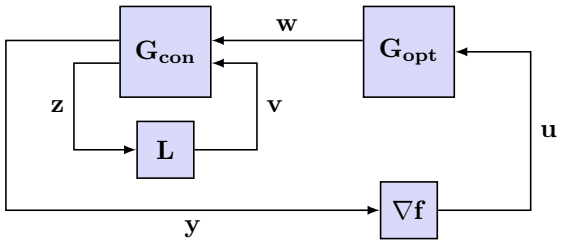



Figure 1: Decomposition of a distributed algorithm into an optimization method $\mathbf{G_{opt}}$ and second-order consensus estimator $\mathbf{G_{con}}$.

Instead, we seek a *distributed* solution to this problem in which each agent $i$ updates its model $y_i$ using its own loss function $f_i$ and variables that are communicated with neighboring agents. Such algorithms can be made scalable to a large number of agents, robust to failures of individual agents [3], and private from other agents [4].

Beyond large-scale machine learning, the distributed optimization problem has many other applications, such as multi-agent formation control [5], distributed spectrum sensing [6], and distributed allocation of resources [7, 8].

In this tutorial, our main objectives are as follows:

1. Provide an overview of the distributed optimization problem for a multi-agent system.
2. Describe the structure of algorithms to solve this problem including how they decompose into optimization and consensus components as in Figure 1.
3. Use simulations to illustrate some algorithmic properties based on this structural decomposition.

We setup the basic structure of distributed optimization algorithms in Section 2 and show how they decompose into optimization and consensus components in Section 3. We then use simulations to illustrate convergence properties in terms of this decomposition in Section 4.

Throughout the paper, subscripts index the agent and superscripts denote the iteration; for instance, $y_i^k$ is the variable $y$ on agent $i$ at iteration $k$. For a linear time-invariant system $G$, we denote its transfer function as $\widehat{G}(z)$. We use bold symbols to denote quantities that are aggregated over all agents, such as

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad \text{and} \qquad \mathbf{G} = \begin{bmatrix} G_1 & & \\ & \ddots & \\ & & G_n \end{bmatrix}. \tag{2}$$

## 2 Problem setup

In this section, we describe distributed algorithms for (1) in which agents compute their local gradient and share information through a communication network.

### 2.1 Communication network

We describe the communication network among the agents as a weighted directed graph, such as in Figure 2. Each vertex in the graph corresponds to an agent in the network and is represented by a circle. Edges in the graph are represented by arrows and indicate the flow of information from one agent to another. The weight of an edge is the amount by which the information is weighted.

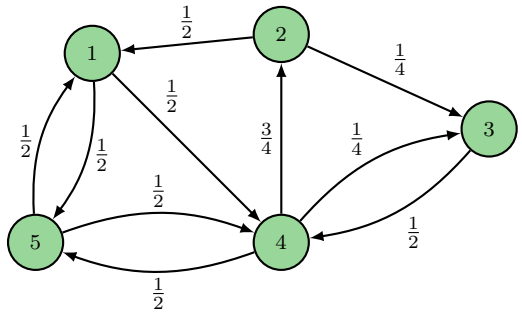



Figure 2: A weighted directed graph that represents the communication network among a group of agents.

Suppose that each agent $i$ has a scalar variable $z_i$. One way for an agent to fuse its information with that of its neighbors is to compute a weighted average of the difference between its variable and those of its neighbors. The weight $a_{ij}$ that agent $i$ places on information from agent $j$ is the weight of the edge from node $i$ to node $j$ in the graph. This is a linear operation over the concatenated vector $\mathbf{z} = (z_1, \ldots, z_n) \in \mathbb{R}^n$ that is represented by multiplication with the Laplacian matrix $L \in \mathbb{R}^{n \times n}$. In particular, the $i^{\text{th}}$ component of the product is

$$(L\mathbf{z})_i = \sum_{j=1}^{n} a_{ij} \, (z_i - z_j). \tag{3}$$

The weight $a_{ij}$ is nonzero only when agent $j$ is able to send information to agent $i$, so agent $i$ can compute this quantity using the communication network. For the graph in Figure 2, the Laplacian matrix is

$$L = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & 0 & -\frac{1}{2} \\ 0 & \frac{3}{4} & 0 & -\frac{3}{4} & 0 \\ 0 & -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} & 0 \\ -\frac{1}{2} & 0 & -\frac{1}{2} & \frac{3}{2} & -\frac{1}{2} \\ -\frac{1}{2} & 0 & 0 & -\frac{1}{2} & 1 \end{bmatrix}.$$

This graph is balanced in that, for each node, the sum of the weights of all incoming edges is equal to that of the outgoing edges [9]. In terms of the Laplacian matrix, this means that each row and column sums to one. Balanced graphs preserve averages since $\sum_{i=1}^{n} (L\mathbf{z})_i = \sum_{i=1}^{n} z_i$.

### 2.2 Distributed algorithms

We now describe the structure of distributed algorithms. Each agent $i$ maintains an estimate $y_i$ of the optimal solution to (1) and can evaluate its local gradient to obtain the quantity

$$u_i = \nabla f_i(y_i). \tag{4a}$$

Agent $i$ can also communicate some quantity $z_i$ with its local neighbors and fuse the information using the graph Laplacian to obtain

$$v_i = \sum_{j=1}^{n} a_{ij} \, (z_i - z_j). \tag{4b}$$

And finally, the algorithm must determine how to choose the point $y_i$ at which to evaluate the gradient and the point $z_i$ to communicate with neighboring agents in the network. We assume each agent uses the same algorithm and represent this operation as

$$\begin{bmatrix} y_i \\ z_i \end{bmatrix} = H \begin{bmatrix} u_i \\ v_i \end{bmatrix}. \tag{4c}$$

We focus on algorithms for which $H$ is causal and linear time-invariant (LTI), although some algorithms in the literature are nonlinear and/or time-varying [10]. Together, equations (4a)–(4c) represent the algorithm on agent $i$.

We can represent distributed algorithms compactly using the concatenated vectors $\mathbf{u}$, $\mathbf{v}$, $\mathbf{y}$, and $\mathbf{z}$ as in (2). In terms of these concatenated vectors, (4a) becomes

$$\mathbf{u} = \nabla \mathbf{f}(\mathbf{y}) \quad \text{where} \quad \nabla \mathbf{f} = \operatorname{diag}(\nabla f_1, \ldots, \nabla f_n). \tag{5a}$$

Likewise, using the Kronecker product $\otimes$, equation (4b) can be represented as

$$\mathbf{v} = \mathbf{L}\mathbf{z} \quad \text{where} \quad \mathbf{L} = L \otimes I_m \tag{5b}$$

with $m$ the dimension of the communicated variable $z_i$. And finally, (4c) becomes

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \mathbf{H} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \text{ where } \mathbf{H} = \begin{bmatrix} I_n \otimes H^{11} & I_n \otimes H^{12} \\ I_n \otimes H^{21} & I_n \otimes H^{22} \end{bmatrix}. \tag{5c}$$

These relationships are summarized by the block diagram in Figure 3, where the system $\mathbf{H}$ is in feedback with the gradient $\nabla \mathbf{f}$ and Laplacian $\mathbf{L}$.

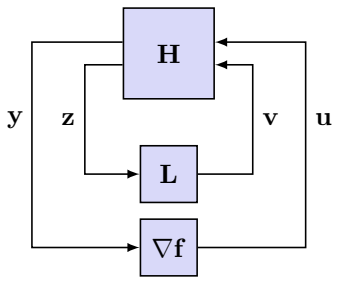



Figure 3: Structure of a general distributed algorithm.

## 3 Algorithm structure

The previous section describes the basic structure of a distributed algorithm as a system $H$ in feedback with the gradient of the objective functions and the Laplacian matrix. It does not, however, provide any insight into how properties of the algorithm depend on $H$, or even what choices of $H$ lead to sensible algorithms[1].

Intuitively, the optimization problem (1) is a combination of optimization (minimizing the sum of the functions) and consensus (having the agents agree on the solution). In this section, we first review algorithms for optimization and consensus separately, and then describe how any distributed algorithm of the form in Sec. 2.2 decomposes into these two components.

### 3.1 Consensus estimators

We now describe the problem of *consensus*. Suppose each agent $i$ observes a (potentially time-varying) signal $w_i^k$. A consensus estimator is an iterative procedure for each agent to estimate the average signal $w_{\text{avg}}^k = \frac{1}{n}\sum_{i=1}^n w_i^k$ by sharing information with its local neighbors [11].

The block diagram of one particular consensus estimator is shown in Figure 4.

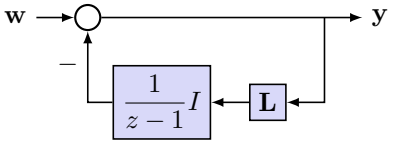



Figure 4: The proportional (P) estimator.

Let $x_i$ denote the state of the estimator on agent $i$. Then the proportional estimator is described by the recursion

$$y_i^k = w_i^k - x_i^k \tag{6a}$$

$$x_i^{k+1} = x_i^k + \sum_{j=1}^n a_{ij}\left(y_i^k - y_j^k\right) \tag{6b}$$

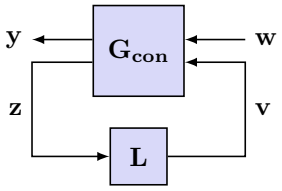
where the state is initialized such that $\sum_{i=1}^n x_i^0 = 0$.

In general, each agent $i$ has an input signal $w_i$ for which the agents seek to compute the average, an output signal $y_i$ that estimates the average of the inputs, a signal $z_i$ that the agent communicates with neighbors, and a signal $v_i$ that is the result of applying the Laplacian matrix to the communicated variables. The block diagram of a general consensus estimator with these components is shown in Figure 5.

For example, the transfer function of the P estimator is

$$\widehat{G}_{\text{con}}(z) = \begin{bmatrix} 1 & \frac{-1}{z-1} \\ 1 & \frac{-1}{z-1} \end{bmatrix}. \tag{7}$$

[1]For instance, a desireable property is for all fixed points of the algorithm to correspond to optimal solutions of (1).




$$\begin{bmatrix} y_i \\ z_i \end{bmatrix} = G_{\text{con}} \begin{bmatrix} w_i \\ v_i \end{bmatrix}$$

$$v_i = \sum_{j=1}^n a_{ij}\left(z_i - z_j\right)$$

Figure 5: General form of a consensus estimator.

When the input signal is constant, the output of the P estimator converges asymptotically to the average of the input signal, that is, it has zero steady-state error [11]. Such estimators are called *first-order* estimators. Likewise, a *second-order* estimator asymptotically tracks the average of signals whose deviations from their average are ramps. One way to construct a second-order estimator is by combining two first-order estimators in series.

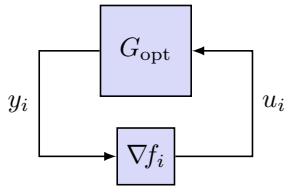
### 3.2 Optimization methods

Now consider a single agent $i$ that seeks to optimize its own objective function $f_i$ (as opposed to the sum of all the functions). To do so, the agent may use a gradient-based optimization method [12] that sequentially queries its gradient $\nabla f_i$.

A particular optimization method is the *gradient method*, which is described by the recursion

$$y_i^{k+1} = y_i^k - \alpha\,\nabla f_i(y_i^k) \tag{8}$$

where $\alpha > 0$ is the stepsize. In general, an optimization method applies a discrete-time dynamical system $G_{\text{opt}}$ to the signal of gradient values $u_i$ to choose the point $y_i$ at which to evaluate the next gradient. The block diagram of a general optimization method is shown in Figure 6.




$$y_i = G_{\text{opt}}\,u_i$$

$$u_i = \nabla f_i(y_i)$$

Figure 6: General form of an optimization method.

For example, the transfer function of the gradient method is

$$\widehat{G}_{\text{opt}}(z) = \frac{-\alpha}{z-1}.$$

For the optimization method to have a fixed point that satisfies the first-order optimality conditions (that is, the gradient is zero), the transfer function must have a pole at $z = 1$. The gradient method uses the minimal number of states to satisfy this requirement, but there are also *accelerated* methods that use additional states to achieve faster convergence [13].

### 3.3 Structural decomposition

Since optimization [12] and consensus [11] have been well-studied in the literature, we seek to represent distributed algorithms as a combination of these two components. It turns out that this is possible: we can combine a valid optimization method and second-order consensus estimator using Figure 1 to form a valid distributed algorithm [14, Theorem 2]. Conversely, any valid distributed algorithm decomposes in this way [14, Theorem 1].

While quite general, these results hold for algorithms in which $G_{\text{opt}}$, $G_{\text{con}}$, and $H$ are causal LTI systems that satisfy certain properties to be *valid*. The notion of a valid algorithm varies depending on the type, but roughly means that the algorithm behaves as desired in the simplest scenario. A valid optimization method, for instance, must converge to the optimal solution when applied to the quadratic function $y \mapsto \frac{\varepsilon}{2}\|y - y^\star\|^2$ for all $y^\star$ and all $\varepsilon > 0$ sufficiently small; see [14] for additional details.

## 4 Simulations

We now use simulations to illustrate some properties of distributed algorithms based on the decomposition in Figure 1. We first setup the problem and then describe each of the various properties.

Consider a set of $n = 5$ agents that are connected in a communication network as shown in Figure 2. Suppose the agents cooperate to solve a machine learning problem in which the loss function is quadratic in the model parameters. That is, the agents solve a distributed linear least squares problem. The objective function on agent $i$ is the quadratic

$$f_i(y) = \tfrac{1}{2} y^\mathsf{T} A_i y - b_i^\mathsf{T} y$$

parameterized by the symmetric matrix $A_i \in \mathbb{R}^{d\times d}$ and the vector $b_i \in \mathbb{R}^d$. The gradient is the linear function

$$\nabla f_i(y) = A_i y - b_i.$$

To generate the data, we sample the matrix $A_i$ such that its eigenvalues are evenly spaced in the interval $[\frac{1}{10}, 1]$, and we sample each element of the vector $b_i$ from a standard normal distribution.

The dimension of the model parameters is $d = 3$. Since the results depends on the problem data which is random, we simulate 1000 trials for each scenario.

### 4.1 Optimization and consensus errors

At each iteration, the error is a measure of the distance between the iterates of all the agents and the optimal solution to the distributed optimization problem (1). To define the error, we use the first-order optimality conditions which are as follows:

- Optimality: the sum of the gradients is zero

$$\sum_{i=1}^{n} \nabla f_i(y_i) = 0 \tag{9a}$$

- **Consensus:** the agents agree on the optimizer

$$y_1 = y_2 = \ldots = y_n \tag{9b}$$

We characterize the error of the iterates in terms of their distance from satisfying these conditions. This consists of two components: the size of the average gradient (the optimization error) and the amount of disagreement (the consensus error). For iterates $y_1^k, \ldots, y_n^k$, we define the optimization and consensus errors as follows:

$$e_{\text{opt}}^k = \left\| \sum_{i=1}^{n} \nabla f(y_i^k) \right\| \quad \text{and} \quad e_{\text{con}}^k = \sum_{i=1}^{n} \left\| y_i^k - \frac{1}{n}\sum_{j=1}^{n} y_j^k \right\|.$$

We take the total error as the maximum of the optimization and consensus errors, $e^k = \max\{e_{\text{opt}}^k, e_{\text{con}}^k\}$, which is zero if and only if the first-order optimality conditions are satisfied.

Figure 7 shows the optimization and consensus errors for each trial (thin) as well as the mean (thick) as a function of the number of iterations. Here, we use the distributed optimization algorithm in Figure 1 in which $G_{\text{con}}$ is two P estimators connected in series and $G_{\text{opt}}$ is the gradient method with stepsize $\alpha = 0.25$.

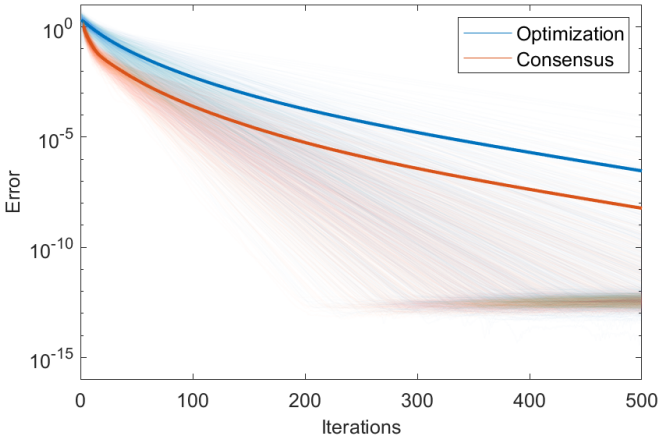



Figure 7: Optimization and consensus errors.

### 4.2 Factored form

Suppose that we iterate the algorithm for a long time. From the first-order optimality condition (9a), the average gradient must be zero at the optimal solution. The gradient of each individual agent, however, is nonzero in general[2]. Recall that the optimization method must have a pole at $z = 1$. The nonzero constant gradient resonates with this pole which causes $w_i$ to grow as a ramp. As this signal grows without bound, the error also grow over time as shown in Figure 8 (blue).

We can fix this issue, however, if the consensus estimator *factors* into two first-order estimators as shown in Fig. 9.

[2] If the gradient of each agent were zero at the optimal solution, then there would be no need to cooperate since each agent could solve the global problem by minimizing its local function!

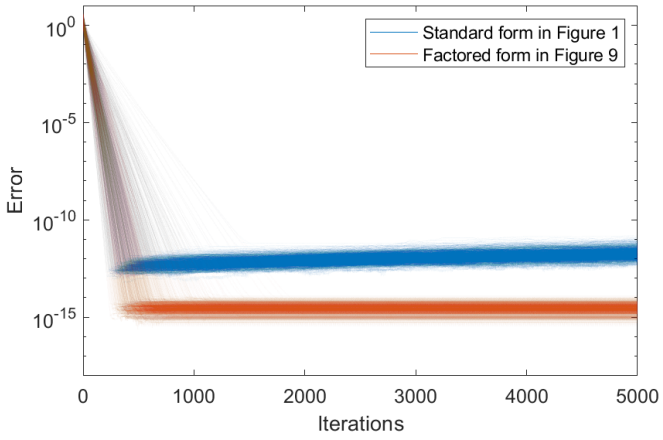



Figure 8: The factored form is numerically stable while the general form is not.

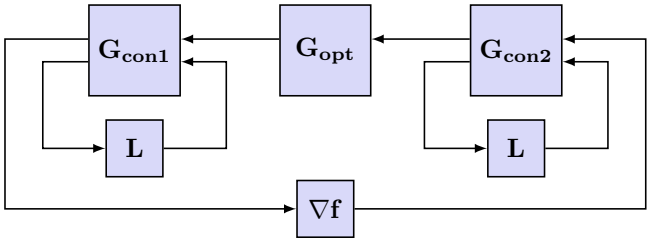



Figure 9: Factored form, where the second-order consensus estimator factors into two first-order estimators.

In this case, the nonzero gradient on each agent is first averaged by the estimator before being applied to the optimization method. Since the average gradient *is* zero, the input to the optimization method is now zero at the optimizer, so the signals no longer grow over time. This is illustrated in Figure 8 (red), where the error remains at the numerical precision of the computer over time. In this case, both consensus estimators are the P estimator.

### 4.3 Accelerated convergence

The decomposition in Figure 1 provides an intuitive procedure to accelerate the convergence of the algorithm. To accelerate the convergence, we can replace the consensus estimator with the accelerated version in Figure 10 that uses additional dynamics to (potentially) accelerate the rate of convergence.

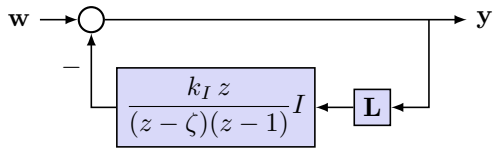



Figure 10: Accelerated consensus estimator.

Similarly, we can replace the gradient method with an accelerated optimization method. Many common first-order methods have the transfer function

$$\widehat{G}_{\text{opt}}(z) = \frac{-\alpha\,(z+\gamma\,(z-1))}{(z-\beta)(z-1)}.$$

As our intuition suggests, using the accelerated consensus estimator and optimization method improves the convergence rate as shown in Figure 11 (red), where we use parameters $(\alpha, \beta, \eta) = (0.1, 0.8, 0)$ and $(\zeta, k_I) = (0.1, 1.1)$.

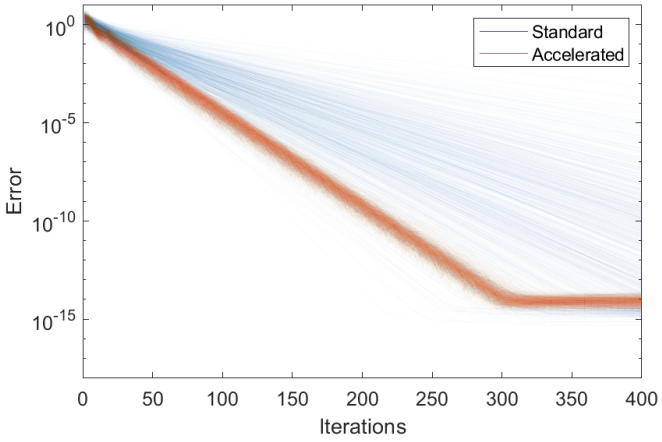



Figure 11: Combining an accelerated consensus estimator and accelerated optimization method may lead to faster convergence.

### 4.4 Robustness

Now suppose agent 1 leaves the network; the agent may have malfunctioned, ran out of power, or been hijacked by an adversary. The modified graph is shown in Fig. 12, where agent 1 and all of its connected edges are opaque to symbolize that it no longer affects the computation[3].

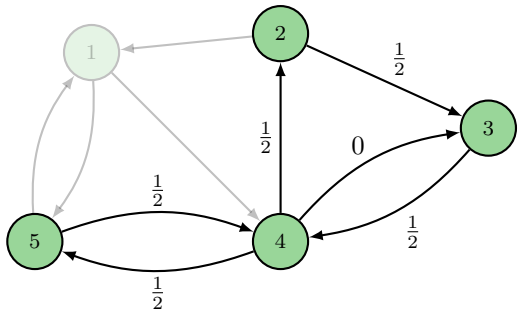



Figure 12: The modified graph after agent 1 leaves the communication network.

The proportional consensus estimator requires specific initialization in that the average state must be zero. This average is invariant in that it does not change over time.

[3]To maintain a balanced graph, the other agents update their weights so that the sum of the incoming weights is equal to that of the outgoing weights. While agent 4 is still capable of sending information to agent 3, the corresponding weight is zero indicating that the information is unused.

From (6a), we observe that the average state $x_i$ must be zero for the average output $y_i$ to equal the average of $w_i$. While we initially set the average state to be zero, it is in general nonzero once the network changes. This results in a systemic error as shown in Figure 13 (blue).

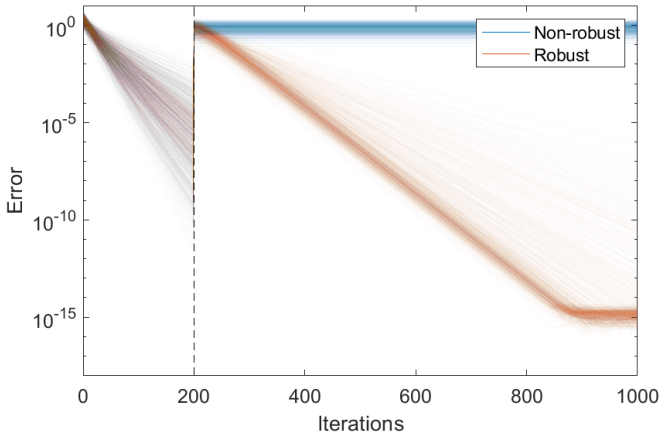



Figure 13: Agent 1 leaves the communication network at iteration $k = 200$. Using a robust consensus estimator enables the algorithm to recover from such changes.

An algorithm is *robust* to changes in the communication network if it does not require a specific initialization, in which case it eventually recovers from such changes. To obtain an algorithm that is robust, we can simply replace the consensus estimator $G_{\text{con2}}$ in Fig. 8 with an estimator that is robust. One such estimator is the PI estimator whose block diagram is shown in Figure 14.

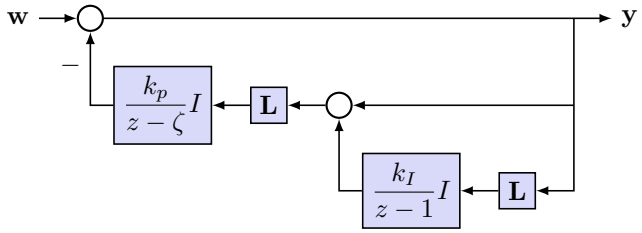



Figure 14: The proportional–integral (PI) estimator, which is robust to changes in the network.

Using this estimator, the error has a transient after the change in the communication network but then converges to zero as shown in Figure 13 (red). Here, we use the parameters $(k_p, k_I, \zeta) = (1, 0.5, 0.95)$.

## 5 Conclusion

In this tutorial, we studied distributed optimization for a multi-agent system. We described the structure of algorithms in terms of optimization methods and consensus estimators, and we illustrated some of the properties through simulations on a machine learning problem.

We focused on algorithms that consist of a linear time-invariant system in feedback with the gradient and the Laplacian; however, other types of algorithms may be needed depending on the class of objective functions to be optimized. We also focused on certain properties that depend on the algorithm structure, but we did not describe how to systematically analyze the convergence rate to the optimal solution; such analysis can be done using the techniques in [15–17].

## References


[1] P. A. Forero, A. Cano, and G. B. Giannakis, "Consensus-based distributed support vector machines," *Journal of Machine Learning Research*, vol. 11, pp. 1663–1707, 2010.

[2] B. Johansson, "On distributed optimization in networked systems," Ph.D. dissertation, KTH, 2008.

[3] I. L. Donato Ridgley, R. A. Freeman, and K. M. Lynch, "Self-healing first-order distributed optimization," in *$60^{th}$ IEEE Conference on Decision and Control*, 2021, pp. 3850–3856.

[4] I. L. D. Ridgley, R. A. Freeman, and K. M. Lynch, "Private and hot-pluggable distributed averaging," *IEEE Control Systems Letters*, vol. 4, no. 4, pp. 988–993, 2020.

[5] K.-K. Oh, M.-C. Park, and H.-S. Ahn, "A survey of multi-agent formation control," *Automatica*, vol. 53, pp. 424–440, 2015.

[6] J. A. Bazerque and G. B. Giannakis, "Distributed spectrum sensing for cognitive radio networks by exploiting sparsity," *IEEE Trans. Sig. Process.*, vol. 58, no. 3, pp. 1847–1862, 2009.

[7] D. A. Schmidt, C. Shi, R. A. Berry, M. L. Honig, and W. Utschick, "Distributed resource allocation schemes," *IEEE Signal Processing Magazine*, vol. 26, no. 5, pp. 53–63, 2009.

[8] S. S. Ram, V. V. Veeravalli, and A. Nedić, "Distributed non-autonomous power control through distributed convex optimization," in *IEEE INFOCOM*, 2009, pp. 3001–3005.

[9] R. A. Freeman, T. R. Nelson, and K. M. Lynch, "A complete characterization of a class of robust linear average consensus protocols," in *Amer. Contr. Conf.*, 2010, pp. 3198–3203.

[10] G. Qu and N. Li, "Accelerated distributed nesterov gradient descent," *IEEE Transactions on Automatic Control*, vol. 65, no. 6, pp. 2566–2581, 2020.

[11] S. S. Kia, B. Van Scoy, J. Cortés, R. A. Freeman, K. M. Lynch, and S. Martínez, "Tutorial on dynamic average consensus: The problem, its applications, and the algorithms," *IEEE Contr. Syst. Mag.*, vol. 39, no. 3, pp. 40–72, 2019.

[12] L. Lessard, B. Recht, and A. Packard, "Analysis and design of optimization algorithms via integral quadratic constraints," *SIAM J. Optim.*, vol. 26, no. 1, pp. 57–95, 2016.

[13] B. Van Scoy, R. A. Freeman, and K. M. Lynch, "The fastest known globally convergent first-order method for minimizing strongly convex functions," *IEEE Contr. Syst. Lett.*, vol. 2, no. 1, pp. 49–54, 2018.

[14] B. Van Scoy and L. Lessard, "A universal decomposition for distributed optimization algorithms," *IEEE Control Systems Letters*, vol. 6, pp. 3044–3049, 2022.

[15] S. Han, "Systematic design of decentralized algorithms for consensus optimization," *IEEE Contr. Syst. Lett.*, vol. 3, no. 4, pp. 966–971, 2019.

[16] A. Sundararajan, B. Van Scoy, and L. Lessard, "Analysis and design of first-order distributed optimization algorithms over time-varying graphs," *IEEE Trans. Contr. Netw. Syst.*, vol. 7, no. 4, pp. 1597–1608, 2020.

[17] A. Sundararajan, B. Hu, and L. Lessard, "Robust convergence analysis of distributed optimization algorithms," in *Allerton Conf. Commun. Contr. Comput.*, 2017, pp. 1206–1212.