

# Generalized gradient optimization over lossy networks for partition-based estimation

M. Todescato, N. Bof, G. Cavraro, R. Carli, L. Schenato

October 31, 2017

## Abstract

We address the problem of distributed convex unconstrained optimization over networks characterized by asynchronous and possibly lossy communications. We analyze the case where the global cost function is the sum of locally coupled local strictly convex cost functions. As discussed in detail in a motivating example, this class of optimization objectives is, for example, typical in localization problems and in partition-based state estimation. Inspired by a generalized gradient descent strategy, namely the block Jacobi iteration, we propose a novel solution which is amenable for a distributed implementation and which, under a suitable condition on the step size, is provably locally resilient to communication failures. The theoretical analysis relies on the separation of time scales and Lyapunov theory. In addition, to show the flexibility of the proposed algorithm, we derive a resilient gradient descent iteration and a resilient generalized gradient for quadratic programming as two natural particularizations of our strategy. In this second case, global robustness is provided. Finally, the proposed algorithm is numerically tested on the IEEE 123 nodes distribution feeder in the context of partition-based smart grid robust state estimation in the presence of measurements outliers.

## 1 Introduction

The widespread of smart wireless electronic devices with the consequent creation of large-scale cyber-physical networked systems promises a new revolution in many fields. However, these novel engineering systems and the advent of the “Big-Data” era require the development of new computational paradigms, due to the increasing amount of devices and data to be consistently managed. For example, many problems can be cast as optimization problems. As so, in the last years there has been a growing attention to distributed optimization tools which have become so important for two different reasons: first, the advent of Big Data asks for *parallelisation* of the computational burden among many processing units since it is inconceivable to run optimization algorithms on one single (super)-computer. Second, many optimization problems are *sparse* by nature since correlation between data is local. Nevertheless, one of the major hurdle to effectively deal with distributed optimization using multiple processing units is to guarantee synchronous and reliable communication. Indeed, communication can be wireless and CPU execution times might not be known in advance as in the context of cloud-computing. For this reason, although distributed optimization has a long history in the parallel and distributed computation literature, see, e.g., [1], it has mainly focused on synchronous algorithms. However, to suitably fit with the upcoming large-scale system scenario, in the last years it has been reconsidered from a new peer-to-peer perspective. The first class of algorithms appearing in this new literature relies on primal sub-gradient or descent iterations, as in [2, 3, 4], which have the advantage to be easy to implement and suitable for asynchronous computation. In order to induce robustness in the computation and improve convergence speed, augmented lagrangian algorithms such as the Alternating Direction Methods of Multipliers (ADMM) have been recently proposed. A first distributed ADMM algorithm was proposed in [5, 6, 7], while a survey on this technique is [8]. However, for a distributed implementation, ADMM usually requires problems with very specific structures. In fact, most of the ADMM distributed algorithms are based on a consensus iteration [9]. Thus, a common drawback of this technique is that each node must store in its local memory a copy of the entire state vector. To avoid this problem, a recent partition-based and scalable approach applied to the ADMM algorithm is presented in [10], while to comply with asynchronous computation, suitable modification of the ADMM algorithm have been proposed in [11, 12]. Finally, distributed algorithms based on Newton methods have been proposed to speed-up the computation [13, 14].

<sup>1</sup>M. Todescato, N. Bof, R. Carli and L. Schenato are with the Department of Information Engineering, University of Padova, Italy, 35031. E-mail: [todescat,bofnicol,carlirug,schenato]@dei.unipd.it.

<sup>2</sup>G. Cavraro is with the Virginia Polytechnic Institute and State University, VA, USA. E-mail: cavraro@vt.edu.

In this paper, we address the problem of *distributed* convex unconstrained optimization over networks characterized by asynchronous and possibly lossy communications. We analyze the case where the global cost function is the sum of *locally coupled* local costs. More specifically, by “locally coupled” and “distributed” we mean the following

**Definition 1 (Local coupling and distributed algorithm)** Consider a set of  $N$  processing units,  $\{1, \dots, N\}$ , which are interconnected according to a certain communication network. To each unit  $i \in \{1, \dots, N\}$  a local cost  $J_i$  is assigned. We say that  $J_i$  is locally coupled according to the communication network, if  $J_i$  depends only on quantities which are related to unit  $i$  and to units directly connected to it. In this case, a distributed algorithm is a procedure running over the communication network and among the processing units, which only requires the exchange of local information among connected units. Differently said, with a slight abuse of nomenclature, a distributed algorithm is defined as a locally coupled procedure.  $\square$

Given Definition 1, this study is motivated mainly by two facts. The first is its practical engineering relevance. Indeed, as discussed in detail in a motivating example we provide in Section 3, the structure of the class of convex optimization problems we analyze, characterizes a large variety of applications such as multi-area electric grid state estimation [15, 7], localization in multi-robots formation [16] and sensors networks [17] and Network Utility Maximization [18]. The second is due to the class of gradient-based algorithms (e.g., [2, 3, 4]) we consider to solve our optimization problem. In particular, while it has the advantage to be easy to implement and suitable for asynchronous implementations, this class usually does not lead to “distributed” solutions as intended in Definition 1. Indeed, the derivatives of costs obtained as the sum of locally coupled costs are, usually, not locally coupled, yet they depend on information related to multi-hop processing units. Hence, the local functional dependence cannot be directly exploited. To overcome this issue, in some cases ([2, 3]) the algorithms require the local exchange of global information, hence all the processors eventually reach consensus to an optimal solution. In others, the algorithms require multiple communication rounds within the same algorithmic iteration ([4]). However, this solution implicitly asks for synchronicity. Hence, to deal with the case of lossy communications, a natural approach is to make the processing units store the last successfully received information from the neighboring units in order to leverage the vast existing body of literature on the so called *partially asynchronous iterative methods* [1]. However, as later described in Section 3, in this scenario, because of packet drops and communication failures, the same state variables happen to appear in multiple delayed version. Thus, it is not possible to write the evolution of the state variables as a *partially asynchronous iterative methods*. Finally, regarding the problem of computing non-locally coupled derivatives, another used approach is to exploit hyper-communication graphs which differs from the graph structure induced by the local coupling characterizing the cost function, thus artificially bypassing the limitations due to a “truly” distributed procedure. As a particular example of this fact, consider, for instance, the case where the processors communicate through a communication network with star topology. According to Definition 1, each peripheral node can communicate only with the central node, while the central processor can communicate with everyone else. Conversely, if a two-hop communication is exploited, then the communication network turns out to be described by an *all-to-all* topology.

In this regard, the main contribution of the paper is a truly distributed algorithm, based on a modified *generalized gradient descent* iteration which, under suitable assumptions on the step size, is provably convergent and which is resilient to the presence of packet losses in the communication channel. To the best of the authors’ knowledge, this is one of the first provably convergent algorithms in the presence of packet losses, since even if both ADMM algorithms and distributed sub-gradient methods (DSM) can handle asynchronous computations, they still require reliable communication and usually do not satisfy Definition 1. Interestingly, the proposed algorithm is also suitable for fully parallel computation, i.e., multiple agents can communicate and update their local variable simultaneously, and for broadcast communication, i.e., nodes do not need to enforce a bidirectional communication such as in gossip algorithms, and therefore is very attractive from a practical point of view. It is anticipated that we presented the proposed algorithm in two preliminary versions. In [19] for the specific case of quadratic programming, while in [17] for the specific application of sensors networks locations. The algorithm is inspired on and resembles the block Jacobi iteration appeared in [1, 20]. However, in [1], since the authors are majorly interested in parallel computation rather than implementing a distributed procedure suitable for today’s sensors networks, they implicitly assume to exploit an hyper-communication graph. Conversely, in [20] the particular local dependency considered in the cost function ensures that first and successive derivatives are locally coupled.

To show the flexibility of the proposed procedure, we derive a resilient gradient descent iteration and a resilient generalized gradient for quadratic programming as two natural particularizations of our strategy. In this second case, we are able to provide global robustness.

Finally, we numerically study our algorithm on the standard IEEE 123-nodes test feeder for robust state estimation in the presence of measurements outliers.

The rest of the paper is organized as follows: the rest of this section is devoted to the necessary notation and preliminaries. In Section 2 we formulate the problem. In Section 3 we provide a motivating example for the proposed set-up. In Section 4 we analyze the case of synchronous and ideal communications. In Section 5 we analyze the case of asynchronous and possibly unreliable communications. In Section 6 we test our algorithm. Finally, we present some concluding remarks in Section 7.

## 1.1 Mathematical Preliminaries

In this paper,  $\mathcal{G}(\mathcal{V}, \mathcal{E})$  denotes a directed graph where  $\mathcal{V} = \{1, \dots, N\}$  is the set of vertices and  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$  is the set of directed edges. More precisely the edge  $(i, j)$  is incident on node  $i$  and node  $j$  and is assumed to be directed away from  $i$  and directed toward  $j$ . The graph  $\mathcal{G}$  is said to be bidirected if  $(i, j) \in \mathcal{E}$  implies  $(j, i) \in \mathcal{E}$ . Given a directed graph  $\mathcal{G}(\mathcal{V}, \mathcal{E})$ , a directed path in  $\mathcal{G}$  consists of a sequence of vertices  $(i_1, i_2, \dots, i_r)$  such that  $(i_j, i_{j+1}) \in \mathcal{E}$  for every  $j \in \{1, \dots, r-1\}$ . The length of a path is the number of directed edges which it consists of. The directed graph  $\mathcal{G}$  is said to be *strongly connected* if for any pair of vertices  $(i, j)$  there exists a directed path connecting  $i$  to  $j$ . Given the directed graph  $\mathcal{G}$ , the set of neighbors of node  $i$ , denoted by  $\mathcal{N}_i$ , is given by  $\mathcal{N}_i = \{j \in \mathcal{V} \mid (i, j) \in \mathcal{E}\}$ . Moreover,  $\mathcal{N}_i^+ = \mathcal{N}_i \cup \{i\}$ . Let us denote the cardinality of  $\mathcal{N}_i^+$  by  $\mu_i$ , while the  $j$ -th neighbor of  $i$  by  $\mathcal{N}_i^+(j)$ . Given a directed graph  $\mathcal{G}(\mathcal{V}, \mathcal{E})$  with  $|\mathcal{E}| = M$ , let the *incidence matrix*  $\mathcal{A} \in \mathbb{R}^{M \times N}$  of  $\mathcal{G}$  be defined as  $\mathcal{A} = [a_{ei}]$ , where  $a_{ei} = 1, -1, 0$ , if edge  $e$  is incident on node  $i$  and directed away from it, is incident on node  $i$  and directed toward it, or is not incident on node  $i$ , respectively. Given a vector or a matrix, with  $(\cdot)^\top$  we denote its transpose, while with  $\Re(\cdot)$  and  $\Im(\cdot)$  its real and imaginary parts, respectively. Given a vector  $v$ , with  $\text{diag}(v)$  we denote the diagonal matrix whose diagonal elements are equal to the elements of  $v$ . Given a matrix  $V$ , with  $\text{diag}(V)$  we denote the vector obtained with the diagonal elements of  $V$ . Given a group of matrices  $V_1, \dots, V_n$ , with  $\text{blkdiag}(V_1, \dots, V_n)$  we denote the block diagonal matrix whose  $i$ -th block diagonal element is equal to  $V_i$ . Moreover, we denote with  $\mathcal{A}_d := \mathcal{A}^\top \mathcal{A}$  the *adjacency matrix* or *laplacian matrix* of  $\mathcal{G}$  which has the property that  $[\mathcal{A}_d]_{ij} \neq 0$  if and only if  $(i, j) \in \mathcal{E}$ . If we associate to each edge a weight different from one, then it is possible to define the *weighted laplacian matrix* as  $\mathcal{L} = \mathcal{A}^\top W \mathcal{A}$ , where  $W \in \mathbb{R}^{M \times M}$  represents the diagonal matrix containing in its  $i$ -th element the weight associated to the  $i$ -th edge. We will also consider *strictly convex functions*  $f(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ , i.e.,  $\forall x_1 \neq x_2$  and  $\eta \in (0, 1)$  then  $f(\eta x_1 + (1 - \eta)x_2) < \eta f(x_1) + (1 - \eta)f(x_2)$  and *radially unbounded*, i.e.  $\|x\| \rightarrow +\infty \Rightarrow f(x) \rightarrow \infty$ . Finally, with the symbols  $\mathbb{E}$  and  $\mathbb{P}$  we denote, respectively, the expectation operator and the probability of an event.

## 2 Problem Formulation

Consider a set of  $N$  agents  $\mathcal{V} = \{1, \dots, N\}$ , where each agent  $i \in \mathcal{V}$  is described by its state vector  $x_i \in \mathbb{R}^{n_i}$ . Assume the agents can communicate among themselves through a bidirected strongly connected *communication graph*  $\mathcal{G}(\mathcal{V}, \mathcal{E})$ . In this paper, we are interested in extending to the more general case of convex costs the algorithm first presented in [19] for the case of quadratic programming and in [17] for the specific application of sensors networks localization. In particular, we examine a particular class of separable strictly convex cost functions which exhibit local and possibly nonlinear dependence among the states of neighboring nodes. By defining the overall state vector as  $x = [x_1^\top, \dots, x_N^\top]^\top \in \mathbb{R}^n$  ( $n = \sum_i n_i$ ), we consider the following optimization problem

$$\min_x J(x) \equiv \min_{x_1, \dots, x_N} \sum_{i=1}^N J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}). \quad (1)$$

Observe that the local dependence coincides with the communication graph  $\mathcal{G}$ , i.e., each cost function  $J_i$  depends on information regarding only agent  $j \in \mathcal{N}_i^+$ .

We will consider the following assumption on the cost functions:

**Assumption 2 (Strict convexity and radial unboundedness)** *The function  $J(x)$  is assumed to be strictly convex and radially unbounded.*  $\square$

Observe that under the previous assumption the minimizer  $x^*$  of Problem (1) exists and is unique

$$x^* := \underset{x}{\operatorname{argmin}} J(x), \quad (2)$$

but the local costs function  $J_i$  do not need to be strictly convex and radially unbounded. Indeed in many estimation problems the local cost functions  $J_i$  are just strictly convex but not radially unbounded. The standard approach to solve the previous optimization problem is to resort to some centralized iterative algorithm acting on  $J$ , e.g., Newton-Raphson, which makes use of global knowledge of the network's states, costs and topology. On the contrary, by leveraging the particular local dependence characterizing each cost function  $J_i$ , we want to solve Problem (1) by developing a procedure which is *distributed*, i.e., exploiting only local exchange of information among neighbors, and *resilient*, i.e., resilient to communication limitations and non idealities.

We will also use the following simplified notation for local components of gradients and Hessians:

$$\nabla_i J_j = \frac{\partial J_j}{\partial x_i}, \quad \nabla_{i\ell}^2 J_j = \frac{\partial^2 J_j}{\partial x_i \partial x_\ell}.$$

**Remark 3 (On the class of separable cost functions)** *The class of functions considered can arise in diverse applications such as state estimation in smart electric grids [19] and sensor networks localization [17], just to mention some of them. In the particular case of quadratic cost, the optimization problem falls onto the standard linear least-squares framework. Nevertheless, as it will be shown in the simulation Section 6, the class is much more general and comprises penalty functions used, e.g., to perform robust statistics and general nonlinear least-squares optimization. Of particular interest is the more general framework of parallel computation in optimization. For both privacy and efficiency reasons, the computational burden can be split among several distributed machines. To each of them only information about  $J_i$  is assigned. Thanks to local exchange of information, the machines must distributively compute a solution of (1).*  $\square$

**Remark 4 (Partition-based modular communication architecture)**

*Note that the particular communication architecture considered, seamlessly describes the case of communications among single peer agents as well as among large areas consisting of a collection of peers. The only difference relies on the particular definition of the set  $\mathcal{V}$  and of the agents' state  $x_i$ . For instance, in the case of sensor localization, each sensor might represent an agent of  $\mathcal{V}$  while  $x_i$  might describe its absolute position in an inertial global reference frame. Conversely, in the case of smart grids state estimation, one agent might describe an entire electric feeder; then,  $x_i$  would be either voltages or currents at all the electric buses of the corresponding feeder.*  $\square$

### 3 Motivating example: State estimation in Smart Power Distribution Grids

In steady state the voltages and currents in a power distribution grid are regulated by the Kirchhoff's laws which can be written as follow:

$$Lv = i^c,$$

where  $L$  is the admittance matrix, and  $v$  and  $i^c$  are the vector collecting all the  $N$  voltages and currents of the nodes in the grid, respectively. The admittance matrix is a sparse matrix, in the sense that the current at a specific node  $i$ , namely  $i_i^c$ , depends only on its own voltage and the voltages of its physically connected neighbour nodes  $\mathcal{N}_i$ , i.e.

$$i_i^c = \sum_{j \in \mathcal{N}_i^+} L_{ij} v_j.$$

In future smart distribution grids, it is expected that each node  $i$  would be able to take noisy measurements of its voltage and current, i.e.

$$\begin{aligned} y_i^v &= v_i + w_i^v, \\ y_i^{i^c} &= i_i^c + w_i^{i^c} = \sum_{j \in \mathcal{N}_i^+} L_{ij} v_j + w_i^{i^c}, \end{aligned}$$

where  $w_i^v, w_i^{i^c}$  represent the measurement noise for the voltage and current measurements, respectively. It is also expected that these nodes are embedded with communication capabilities, such as power line communication (PLC), which allow them to communicate with their physically connected neighbours. As so the communication network and the physical network will coincide. The (centralized) state estimation problem is the process that, given all the measurements  $\{y_i^v, y_i^{i^c}\}_{i=1}^N$ , should return the best estimate of all the voltages and currents  $\{v_i, i_i^c\}_{i=1}^N$ . The standard approach is to cast this problem as a least-square estimation problem, where the unknown quantities to be estimated are the voltages  $v^*$ , since the currents can be estimated directly from the voltages via the Kirchhoff's law  $i^{c*} = Lv^*$ . In this work, we are interested in solving this problem in a distributed fashion via a partition-based communication architecture. For the sake of clarity, let us assume that the grid is divided into  $N$  partitions each corresponding to a node. To each partition, we associate the corresponding voltage, which we collect in the vector  $x_i \in \mathbb{R}^1$ . Let us also define with  $y_i = [y_i^v, y_i^{i^c}]^\top \in \mathbb{R}^2$  and  $w_i = [w_i^v, w_i^{i^c}]^\top \in \mathbb{R}^2$  the measurement vector and the measurement noise corresponding to the measurements of the voltage and current at node  $i$ . Let us also define the vectors  $x = [x_1, \dots, x_N]^\top \in \mathbb{R}^N$ ,  $y = [y_1^\top, \dots, y_N^\top]^\top \in \mathbb{R}^{2N}$ ,  $w = [w_1^\top, \dots, w_N^\top]^\top \in \mathbb{R}^{2N}$ . As so the measurement model can be written as:

$$y_i = \sum_{j=1}^N A_{ij} x_j + w_i = \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j + w_i \quad (A_{ij} = 0 \text{ if } j \notin \mathcal{N}_i^+),$$

where  $A_{ij}$  can be easily be obtained from the elements of the matrix  $L$ , or equivalently in vector form

$$y = Ax + w,$$

---

<sup>1</sup>In reality, the voltages and currents in steady state are phasors, i.e., should be represented as complex numbers. However, the discussion in this section can be extended w.l.o.g. also to the more realistic scenario, which is indeed considered in the Simulation section below.

where  $A = [A_1^\top, \dots, A_N^\top]^\top \in \mathbb{R}^{2N \times N}$  and  $A_i = [A_{i1}, \dots, A_{iN}] \in \mathbb{R}^{2 \times N}$ . If we define

$$J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) = \frac{1}{2} \|y_i - A_i x\|^2, \quad J(x) = \sum_{i=1}^N J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) = \frac{1}{2} \|y - Ax\|^2,$$

with

$$\begin{aligned} \nabla J(x) &= A^\top (Ax - y), \quad \nabla^2 J(x) = H = A^\top A, \\ H_{ij} &= \sum_{\ell=1}^N A_{\ell i}^\top A_{\ell j} = \sum_{\ell \in \mathcal{N}_i^+} A_{\ell i}^\top A_{\ell j} = \sum_{\ell \in (\mathcal{N}_i^+ \cap \mathcal{N}_j^+)} A_{\ell i}^\top A_{\ell j} \end{aligned}$$

the optimal (centralized) least squares solution<sup>2</sup> is given by:

$$x^* = \operatorname{argmin}_x J(x) = (A^\top A)^{-1} A^\top y.$$

A standard approach to asymptotically obtain the optimal solution is to employ an iterative algorithm based on the generalized gradient descent:

$$x^+ = x - \epsilon D^{-1} A^\top (Ax - y) = x - \epsilon D^{-1} \nabla J(x) = x - \epsilon D^{-1} (Hx - A^\top y),$$

where  $\epsilon$  is a suitable stepsize and  $D$  is a strictly positive definite matrix, i.e.  $D > 0$ . A typical way to solve the previous update in a distributed fashion is to pick a block-diagonal matrix  $D$ , i.e.  $D = \operatorname{blkdiag}(D_1, \dots, D_N)$ , so that the previous centralized update can be written as

$$\begin{aligned} x_i^+ &= x_i - \epsilon D_i^{-1} \left( \sum_{j=1}^N H_{ij} x_j - \sum_{j=1}^N A_{ji}^\top y_j \right) \\ &= x_i - \epsilon D_i^{-1} \left( \sum_{j=1}^N \sum_{\ell=1}^N A_{\ell i}^\top A_{\ell j} x_j - \sum_{j=1}^N A_{ji}^\top y_j \right) \\ &= x_i - \epsilon D_i^{-1} \left( \sum_{j \in \mathcal{N}_i^+, \forall \ell \in \mathcal{N}_i^+} A_{\ell i}^\top A_{\ell j} x_j - \sum_{j \in \mathcal{N}_i^+} A_{ji}^\top y_j \right), \end{aligned} \tag{3}$$

where we exploited the property that  $A_{ij} = 0$  if  $j \notin \mathcal{N}_i^+$ . While the second summation involves only measurements that belongs to the neighbours of node  $i$ , the first summation requires the node  $i$  to collect the state variables  $x_j$  that belongs to the neighbours of the neighbours. As so, this implementation is not really distributed, since two-hop communication is required. Although this is not impossible from a practical perspective, it requires substantial additional communication and synchronization efforts. An alternative approach which allows the implementation of a truly distributed algorithm is to create the additional local variable at each node  $i$ :

$$z_i = A_i x_i = \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j, \quad \forall i,$$

which can be collected in the vector  $z = [z_1^\top, \dots, z_N^\top]^\top$ , so that in matrix form the previous expression can be written as  $z = Ax$ . With this notation the generalized gradient descent can be written as:

$$\begin{aligned} z_i^+ &= \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j \\ x_i^+ &= x_i - \epsilon D_i^{-1} \left( \sum_{\ell=1}^N A_{\ell i}^\top \underbrace{\sum_{j=1}^N A_{\ell j} x_j}_{z_\ell} - \sum_{j=1}^N A_{ji}^\top y_j \right) \\ &= x_i - \epsilon D_i^{-1} \sum_{j \in \mathcal{N}_i^+} A_{ji}^\top (z_i^+ - y_i). \end{aligned}$$

---

<sup>2</sup>The formulation can be extended to the weighed least square solutions if noise with different variances  $R$  are included which would lead to the solution  $x^* = (A^\top R^{-1} A)^{-1} A^\top R^{-1} y$ , but for the sake of clarity in the notation of this section, it is omitted.



This alternative solution requires two communication rounds to compute  $x_i^+$ , since first it is necessary to send the  $x_i$  to compute  $z_i^+$ , and then to transmit  $z_i$  to the neighbours<sup>3</sup>. In practical scenarios, such as using PLC protocols, synchronization of transmissions and updates can be difficult. Moreover packet losses might occur, i.e. some messages from the neighbours might not be received. A naive solution to this problem, is to use local registers that keep in memory the latest message received from the neighbours, and then use these values whenever an update of the local variables  $x_i, z_i$  is needed. It can be shown that this is equivalent to a scenario where every node  $j \in \mathcal{N}_i$  use a delayed version of the local variables  $x_i, z_i$ . Since the variables  $z_i$  are function of the (possibly delayed) state variables  $x_i$ , the variables  $x_i$  are themselves functions of the delayed version of variables  $x_i$  of the network. More specifically, it can be shown that the previous update equations can be written as

$$z_i(t+1) = \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j(\tau'_{ij}(t)), \quad (4)$$

$$x_i(t+1) = x_i(t) - \epsilon D_i^{-1} \left( \sum_{\ell=1}^N A_{\ell i}^\top \sum_{j=1}^N A_{\ell j} x_j(\tau_{\ell j}(t)) - \sum_{j=1}^N A_{ji}^\top y_j \right). \quad (5)$$

where  $0 \leq \tau_{ij}(t), \tau'_{ij}(t) \leq k$  represent the delay of each variable which depends on the specific sequence of packet losses and variable updates, and explicitly included the time dependency of each variable. Note that in the last equation the variable  $x_j$  might appear with multiple instances with different delays into the update of the variable  $x_i$ , i.e. it is not possible to write the evolution of variables of the original generalized gradient descent algorithm given in Eqn. (3) as a *partially asynchronous iterative methods* (see chapter 7 of [1]), for which an extensive body of literature exists, since the cited framework would require the algorithm to be written as:

$$x_i(t+1) = x_i(t) - \epsilon D_i^{-1} \left( \sum_{j=1}^N H_{ij} x_j(\tau_{ij}(t)) - \sum_{j=1}^N A_{ji}^\top y_j \right). \quad (6)$$

Motivated by this observation, in this work we will propose an alternative mathematical machinery based on Lyapunov theory and the separation of time scale principle to prove convergence of the asynchronous algorithm (5) for a sufficiently small stepsize  $\epsilon$ . Note that this machinery can also be applied to more general convex problems. This is useful, for example, in the presence of outliers or sensor faults in order to develop more robust estimators than least squares. In fact, a common way to enforce robustness in the estimation is to replace the quadratic cost function defined above with the 1-norm of the residuals, that is

$$J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) = \|y_i - A_i x\|_1. \quad (7)$$

However, since (7) is not differentiable, it cannot be directly used with our algorithm. To deal with this issue, in the Simulation section 6, we will exploit the following modification of the 1-norm [21]

$$\|\cdot\|_{1,\nu} : \mathbb{R}^n \rightarrow \mathbb{R}, \quad x \mapsto \|x\|_{1,\nu} := \sum_{i=1}^N \sqrt{x_i^2 + \nu}, \quad (8)$$

where  $\nu > 0$  is such that the smaller the selected value of  $\nu$  is, the better the approximation of the 1-norm is. In particular, the approximation of each term in the summation of the cost function is quadratic when  $x_i$  belongs to a small neighborhood of 0. The next Sections will then provide a fully distributed generalized gradient descent algorithm which is resilient to lossy communication.

## 4 Synchronous update and reliable communication

In this section we analyze the case of synchronous and ideal, i.e., reliable, communications among neighbors, leaving the extension to the more realistic case of unreliable communication to Section 5.

Consider the optimization Problem (1). In the ideal communication case, one possible choice to iteratively solve Problem (1) is to exploit the so called *generalized gradient descent* iteration

$$x^+ = x - \epsilon D^{-1}(x) \nabla J(x), \quad x(0) = x_0, \quad (9)$$

---

<sup>3</sup>It is necessary to transmit the measurements  $y_i$  only at the initialization phase since they do not change during the course of the evolution of the algorithm.

where  $\nabla J(x) := \left[ \frac{\partial J(x)}{\partial x} \right]_x^\top$  is the gradient of  $J$  evaluated at the current value  $x$ ,  $D(x)$  is a generic positive definite matrix, possibly function of  $x$  itself, and  $\epsilon$  a suitable positive constant, referred to as *step size*. Observe that depending on the particular choice of  $D(x)$ , Eq. (9) describes various types of algorithms. Indeed, if  $D(x) = I$ , the standard gradient descent iteration is obtained; if  $D(x)$  is chosen to be diagonal with diagonal elements equal to those of the Hessian matrix, then a Jacobi descent iteration is retrieved; while, if  $D(x)$  is equal to the entire Hessian, then Eq. (9) returns the classical Newton-Raphson iteration. The algorithm we propose (and describe in Section 5) is inspired by the particular case of (9), referred to as *block Jacobi*, where we choose  $D(t)$  to be the *block diagonal* matrix such that

$$D(x) = \text{blkdiag}(D_1(x), \dots, D_N(x)), \quad D_i(x) := \nabla_{ii}^2 J(x), \quad i \in \mathcal{V}, \quad (10)$$

i.e., where each diagonal block coincides with the second order derivative of  $J$  w.r.t.  $x_i$ . Thanks to this choice for the matrix  $D$ , Eq. (9) can be split into partial state updates each of which equal to

$$x_i^+ = x_i - \epsilon D_i^{-1}(x) \nabla_i J(x), \quad i \in \mathcal{V}. \quad (11)$$

Now, it is convenient to explicitly take into account the separable structure of the cost function  $J$  in order to show that each gradient block  $\nabla_i J$  as well as each  $D_i$  block can be computed exploiting only local information coming from agent's *two-steps neighbors*, i.e., agents connected to agent  $i$  by a directed path of length two. Indeed, for the gradient we have that

$$\nabla_i J(x) = \sum_{j \in \mathcal{N}_i^+} \nabla_i J_j(\{x_k\}_{k \in \mathcal{N}_j^+}) = \nabla_i J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) + \sum_{j \in \mathcal{N}_i} \nabla_i J_j(x_j, \{x_k\}_{k \in \mathcal{N}_j}), \quad (12)$$

and it can be seen that the first term on the RHS of Eq. (12) depends only on information coming from  $j \in \mathcal{N}_i^+$ ; while, the second term depends on information coming from neighbors of node  $i$  and from the neighbors of its neighbors,  $k \in \mathcal{N}_j^+$ . A similar reasoning applies to  $D_i$  indeed,

$$\begin{aligned} D_i(x) &:= \sum_{j \in \mathcal{N}_i^+} \nabla_{ii}^2 J_j(\{x_k\}_{k \in \mathcal{N}_j^+}) \\ &= \nabla_{ii}^2 J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) + \sum_{j \in \mathcal{N}_i} \nabla_{jj}^2 J_j(x_j, \{x_k\}_{k \in \mathcal{N}_j}). \end{aligned} \quad (13)$$

Again, the first term in the RHS of Eq. (13) depends only on node  $i$  direct neighbors,  $j \in \mathcal{N}_i^+$ , while the second term requires information coming from the neighbors of its neighbors. In view of a distributed computation, we assume each agent  $i \in \mathcal{V}$ , once gathered the neighbors states  $\{x_j\}_{j \in \mathcal{N}_i}$ , can compute and store in its local memory, in addition to the state  $x_i$ , the following variables

$$\rho_i^{(j)}(x) := \nabla_j J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}), \quad \xi_i^{(j)}(x) := \nabla_{jj}^2 J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}), \quad (14)$$

which represent the partial components of the first and second derivatives of its local cost  $J_i$  evaluated at the current state value. Observe that, since in a distributed framework each agent is assumed to have information only regarding its local cost  $J_i$ , the  $\rho$ 's and  $\xi$ 's variables represents the quantities which agent  $i$  must compute and send to its neighbors in order to let them compute their corresponding gradient and hessian blocks. Likewise, agent  $i$  needs to receive similar variables from each one of its neighbors. Indeed, thanks to Eqs. (12)–(13), it holds that

$$\nabla_i J(x) = \sum_{j \in \mathcal{N}_i^+} \rho_j^{(i)}(x), \quad D_i(x) = \sum_{j \in \mathcal{N}_i^+} \xi_j^{(i)}(x). \quad (15)$$

As above stressed, each agent  $i \in \mathcal{V}$ , to iteratively compute (11), can perform its computations autonomously assuming it has at its disposal information coming from its two-steps neighbors. However, this presents two major drawbacks:

1. it clashes with a truly distributed setting which exploits the exchange of information only among one-step neighbors;
2. within successive iterations, to ensure consistency and thus convergence of the procedure to a minimizer of Problem (1), all the communications must be synchronous and reliable.

To workaround the first issue one possible solution would be, at each iteration, to perform two communication rounds among one-step neighbors as illustratively shown in Figure 1. The first round is used to exchange the state values among neighboring agents in order them to compute all the partial information terms according to Eqs. (14)–(15); while the second round is

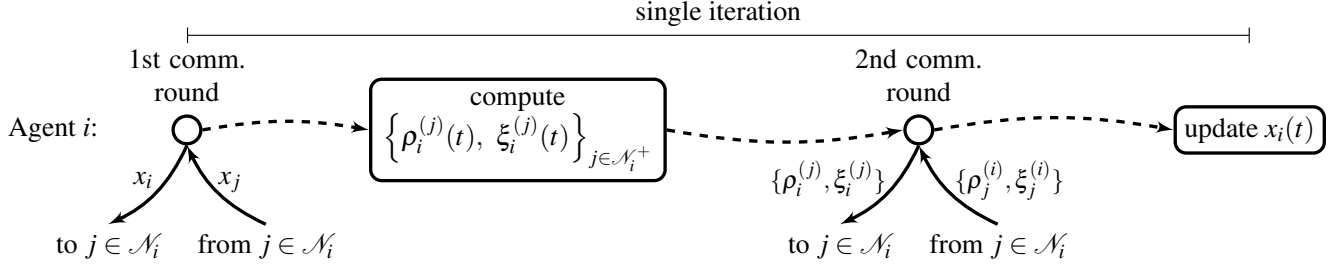


Figure 1: Communication scheme to perform one single block Jacobi iteration (11) in a distributed setting which assumes only information exchange among one-step neighbors.

used to communicate the computed variables in order to perform the state update as in Eq. (11). Regarding the second issue, it necessarily enforces the use of suitable synchronization algorithms as well as re-transmission protocols in case of packet failures. What above described has been compactly written in algorithmic form as reported in Algorithm 1 in which `flagtransmission` denotes a variable to control communication and update among the agents. Note that, even if these might provide possible answers, it is understood they do not represent satisfactory solutions for real-world applications. Conversely, in the next section we propose a truly distributed and resilient iterative procedure which, by naturally exploiting information coming from one-step neighbors and being resilient to packet losses and communication non idealities, is much more appealing from an engineering perspective.

---

**Algorithm 1** Distributed Block Jacobi algorithm (node  $i$ ).

---

**Require:**  $x_i^o, \epsilon$

- 1:  $x_i \leftarrow x_i^o$
- 2: **if** `flagtransmission` = 1 **then**
- 3:   **Broadcast:**  $x_i$
- 4:   **Receive:**  $x_j, \forall j \in \mathcal{N}_i$
- 5:    $\rho_i^{(j)} \leftarrow \nabla_j J_i(\{x_k\}_{k \in \mathcal{N}_j^+}), \forall j \in \mathcal{N}_i^+$
- 6:    $\xi_i^{(j)} \leftarrow \nabla_{jj}^2 J_i(\{x_k\}_{k \in \mathcal{N}_j^+}), \forall j \in \mathcal{N}_i^+$
- 7:   **Broadcast:**  $\rho_i^{(j)}, \xi_i^{(j)}, \forall j \in \mathcal{N}_i$
- 8:   **Receive:**  $\{\rho_j^{(i)}, \xi_j^{(i)}\}, \forall j \in \mathcal{N}_j$
- 9:    $x_i \leftarrow x_i - \epsilon (\sum_{j \in \mathcal{N}_i^+} \xi_j^{(i)})^{-1} (\sum_{j \in \mathcal{N}_i^+} \rho_j^{(i)})$
- 10: **end if**

---

## 5 Asynchronous updates and unreliable communication: the Resilient Block Jacobi (RBJ) algorithm

In this section we consider the more realistic case of asynchronous and unreliable communications where each agent might either receive asynchronous information coming from its neighbors, or not receive it. In particular, we present a modified iteration and analyze its corresponding iterative algorithm, which we refer to as *resilient block Jacobi*, which (i) exploits only information coming from one-step neighbors; (ii) requires only one communication round per algorithmic iteration; (iii) is based on an asynchronous communication protocol; (iv) is resilient to communication failures. First, we present our algorithm for the general case of separable convex costs. Later, we particularize the algorithm to suit two special cases and showing its flexibility.

Consider the standard block Jacobi iteration (11). As analyzed in Section 4, the procedure exhibits some fundamental criticisms which deeply compromise its distributed and asynchronous implementation and yet its robustness properties. Thus, to develop our algorithm, we need to suitably modify iteration (11). The modification we propose is apparently naive since the idea is to simply equip each agent with an additional amount of memory storage to keep track of the last received and available information corresponding to each neighbor. This additional memory is then used to perform Eq. (11). Indeed note that, if agent  $i$  does not receive some of the information coming from its neighbors, it does not have the necessary information to synchronously compute neither (14) nor (15) and thus it is not able to update its state according to (11).



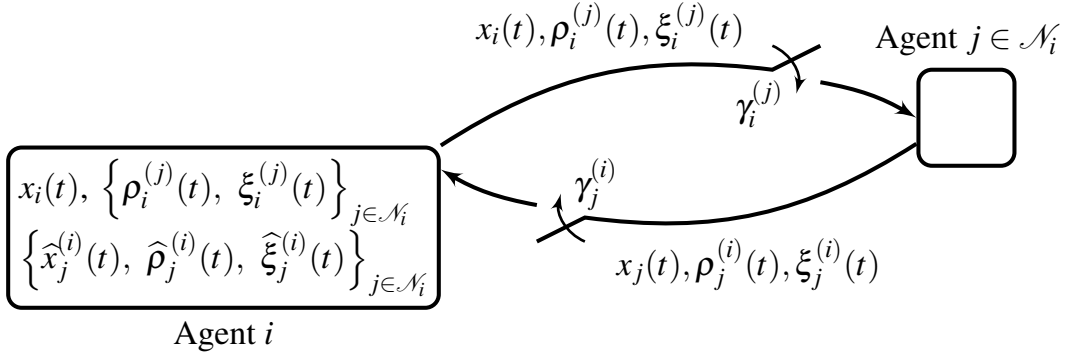


Figure 2: Memory storage and communication scheme between pairs of neighbors agents for the RBJ algorithm.

To model randomly occurring packet losses is convenient to introduce the indicator function

$$\gamma_j^{(i)}(t) = \begin{cases} 1 & \text{if } i \text{ received the information sent by } j \text{ at iteration } t \\ 0 & \text{otherwise.} \end{cases}$$

with the assumption that  $\gamma_i^{(i)}(t) = 1$ , since node  $i$  has always access to its local variables. Then, as suggested above, the main idea is to equip each agent  $i$  with auxiliary variables  $\{\hat{x}_j^{(i)}(t), \hat{\rho}_j^{(i)}(t), \hat{\xi}_j^{(i)}(t)\}_{j \in \mathcal{N}_i}$ , used to keep track of the last available information received from each neighbors. Specifically, the dynamic for the  $j$ -th set of additional variables is given by

$$\{\hat{x}_j^{(i)}(t), \hat{\rho}_j^{(i)}(t), \hat{\xi}_j^{(i)}(t)\} = \begin{cases} \{x_j(t), \rho_j^{(i)}(t), \xi_j^{(i)}(t)\}, & \text{if } \gamma_j^{(i)}(t) = 1; \\ \{\hat{x}_j^{(i)}(t-1), \hat{\rho}_j^{(i)}(t-1), \hat{\xi}_j^{(i)}(t-1)\}, & \text{if } \gamma_j^{(i)}(t) = 0. \end{cases} \quad (16)$$

Thanks to this additional memory at every algorithmic iteration, each agent can perform its local update which, inspired on Eq. (11), becomes equal to

$$x_i(t+1) = x_i(t) - \epsilon \left( \sum_{j \in \mathcal{N}_i^+} \hat{\xi}_j^{(i)}(t) \right)^{-1} \left( \sum_{j \in \mathcal{N}_i^+} \hat{\rho}_j^{(i)}(t) \right). \quad (17)$$

Observe that the differences between Eqs. (11) and (17) are mainly two:

1. the variables in agent  $i$ 's memory, used to store the first and second partial derivatives of  $J_i$  w.r.t.  $j \in \mathcal{N}_i$ , are necessarily computed as

$$\rho_i^{(j)}(t) = \nabla_j J_i(x_i(t), \{\hat{x}_k^{(i)}(t)\}_{k \in \mathcal{N}_i}), \quad \xi_i^{(j)}(t) = \nabla_{jj}^2 J_i(x_i(t), \{\hat{x}_k^{(i)}(t)\}_{k \in \mathcal{N}_i}), \quad (18)$$

that is, they are evaluated at the last stored states' values; likewise, the values of the additional variables  $\{\hat{\rho}_j^{(i)}, \hat{\xi}_j^{(i)}\}_{j \in \mathcal{N}_i}$  correspond to those last received from each neighbor and computed by each of them using the last available information on their neighbors' states;

2. conversely to the synchronous implementation of the algorithm, at each iteration only one communication round is performed. This means the agents send only one packet per iteration, consisting of the state and the partial derivatives. See Figure 2 for an illustrative representation.

Thanks to this simple modification the agents can perform their updates asynchronously and independently. Moreover, since only one communication round per iteration is required, both the communication burden and the number of possible communication failures are reduced. Nevertheless, it is worth stressing that, even if no packet losses occur, the classical block Jacobi and our resilient block Jacobi iteration does not exactly coincide. Indeed, in the resilient case, by sending only one packet per iteration, the state and the partial derivative information would be "delayed" one from each other of one iteration if compared with the synchronous implementation. The *resilient block Jacobi* algorithm (hereafter referred to as RBJ algorithm) for separable convex functions is formally described in Algorithm 2 where it is presented in an event-based update performed by a generic node  $i$ . The variables `flagtransmission`, `flagreception`, `flagupdate` are flag variables which determines which specific action a node is performing, namely transmission, reception or update. When each action is performed it cannot be interrupted, however the specific order or consecutive calls of an action do not impair the convergence of the proposed algorithm and therefore can be used independently of the specific communication protocol or CPU multitasking scheduling.

---

**Algorithm 2** Resilient Block Jacobi (RBJ) Algorithm (node  $i$ )

---

**Require:**  $x_i^o, \epsilon$ **Initialization** (atomic)

- 1:  $x_i \leftarrow x_i^o$
- 2:  $\hat{x}_j^{(i)} \leftarrow 0, \forall j \in \mathcal{N}_i$
- 3:  $\rho_j^{(j)} \leftarrow 0, \forall j \in \mathcal{N}_i$
- 4:  $\hat{\rho}_j^{(i)} \leftarrow 0, \forall j \in \mathcal{N}_i$
- 5:  $\xi_i^{(j)} \leftarrow I, \forall j \in \mathcal{N}_i$
- 6:  $\hat{\xi}_j^{(i)} \leftarrow I, \forall j \in \mathcal{N}_i$
- 7:  $\text{flag}_{\text{transmission}} \leftarrow 1$  (optional)

**Transmission** (atomic)

- 8: **if**  $\text{flag}_{\text{transmission}} = 1$  **then**
- 9:    $\text{transmitter\_node\_ID} \leftarrow i$
- 10:    $\rho_j^{(j)} \leftarrow \nabla_j J_i(x_i, \{\hat{x}_k^{(i)}\}_{k \in \mathcal{N}_i}), \forall j \in \mathcal{N}_i$
- 11:    $\xi_i^{(j)} \leftarrow \nabla_{jj}^2 J_i(x_i, \{\hat{x}_k^{(i)}\}_{k \in \mathcal{N}_i}), \forall j \in \mathcal{N}_i$
- 12:   **Broadcast:**  $\text{transmitter\_node\_ID}, x_i, \{\rho_i^{(j)}, \xi_i^{(j)}\}_{j \in \mathcal{N}_i}$
- 13:    $\text{flag}_{\text{transmission}} \leftarrow 0$
- 14:    $\text{flag}_{\text{reception}} \leftarrow 1$  (optional)

15: **end if****Reception** (atomic)

- 16: **if**  $\text{flag}_{\text{reception}} = 1$  **then**
- 17:    $j \leftarrow \text{transmitter\_node\_ID}$
- 18:    $\hat{x}_j^{(i)} \leftarrow x_j$
- 19:    $\hat{\rho}_j^{(i)} \leftarrow \rho_j^{(j)}$
- 20:    $\hat{\xi}_j^{(i)} \leftarrow \xi_j^{(j)}$
- 21:    $\text{flag}_{\text{reception}} \leftarrow 0$
- 22:    $\text{flag}_{\text{update}} \leftarrow 1$  (optional)

23: **end if****Estimate update** (atomic)

- 24: **if**  $\text{flag}_{\text{update}} = 1$  **then**
- 25:    $\hat{\rho}_i^{(i)} \leftarrow \nabla_i J_i(x_i, \{\hat{x}_k^{(i)}\}_{k \in \mathcal{N}_i})$
- 26:    $\hat{\xi}_i^{(i)} \leftarrow \nabla_{ii}^2 J_i(x_i, \{\hat{x}_k^{(i)}\}_{k \in \mathcal{N}_i})$
- 27:    $x_i \leftarrow x_i - \epsilon \left( \sum_{j \in \mathcal{N}_i^+} \hat{\xi}_j^{(i)} \right)^{-1} \left( \sum_{j \in \mathcal{N}_i^+} \hat{\rho}_j^{(i)} \right)$
- 28:    $\text{flag}_{\text{update}} \leftarrow 0$
- 29:    $\text{flag}_{\text{transmission}} \leftarrow 1$  (optional)

30: **end if**

---

**Remark 5 (Resilient gradient descent (RGD) Algorithm)** *If memory, communication and computational complexity are a concern, it is possible to modify the proposed algorithm mimicking the standard gradient descent algorithm. In this framework, the second order information is not needed and therefore the variables  $\xi_j^{(i)}, \hat{\xi}_j^{(i)}$  (lines 5, 6, 17, 21 in Algorithm 2) do not need to be computed and the update for the local variable  $x_i$  (line 22 in Algorithm 2) should be replaced with the following:*

$$x_i \leftarrow x_i - \epsilon \sum_{j \in \mathcal{N}_i^+} \hat{\rho}_j^{(i)}.$$

*Obviously, the price to pay for this choice is a likely decrease in convergence speed.*

**Remark 6 (Resilient Weighted Least Squares (RWLS) Algorithm)** *If the local cost functions are quadratic, i.e:*

$$J_i(x_i, \{x_j\}_{j \in \mathcal{N}_i}) = \frac{1}{2} \|y_i - A_i x\|_{W_i}^2 = \frac{1}{2} (y_i - \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j)^\top W_i (y_i - \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j),$$

*where  $W_i > 0$  are the local weights, then the problem to be solved becomes a Weighted Least Squares problem. For this special case, the gradient and the hessian components simplify to:*

$$\rho_i^{(j)}(x) := A_{ij}^\top W_i \left( \sum_{j \in \mathcal{N}_i^+} A_{ij} x_j - y_i \right), \quad \xi_i^{(j)}(x) := A_{ij}^\top W_i A_{ij}, \quad (19)$$

*therefore the RBJ Algorithm can be simplified by substituting lines 10 and 11 with the following updates:*

$$\rho_i^{(j)} \leftarrow A_{ij}^\top W_i (A_{ii} x_i + \sum_{j \in \mathcal{N}_i} A_{ij} \hat{x}_j^{(i)} - y_i), \quad \forall j \in \mathcal{N}_i, \quad (20)$$

$$\xi_i^{(j)} \leftarrow A_{ij}^\top W_i A_{ij}. \quad (21)$$

*It is clear from the previous expression, that the algorithm could be modified by having a preliminary phase when the  $\xi_i^{(j)}$  are transmitted reliably to the neighbours so that eventually  $\hat{\xi}_i^{(j)} = \xi_i^{(j)}$ , and then the algorithm could simply transmit the variables  $x_i, \rho_i^{(j)}$  and update the variables  $x_i, \hat{x}_j^{(i)}, \hat{\rho}_j^{(i)}$  which are the only variables that evolve over time, thus considerably reducing the communication complexity which corresponds with that of the RGD algorithm.*

## 5.1 Theoretical analysis of RBJ Algorithm

Before presenting the major theoretical result characterizing the convergence properties of the proposed RBJ algorithm, we introduce the following assumption on the nature of lossy communication we consider. It mainly states that each agent  $i \in \mathcal{V}$  receives information coming from each agent  $j \in \mathcal{N}_i$  at least once within any window of  $T$  iterations of the algorithm.

**Assumption 7 (Persistent communication)** *There exists a constant  $T$  such that, for all  $t \geq 0$ , for all  $i \in \mathcal{V}$  and for all  $j \in \mathcal{N}_i$ ,*

$$\mathbb{P} \left[ \{\gamma_j^{(i)}(t), \dots, \gamma_j^{(i)}(t+T)\} = \{0, \dots, 0\} \right] = 0.$$

□

**Theorem 8 (Local convergence of the RBJ algorithm)** *Let Assumptions 2 and 7 hold. Moreover assume that the cost functions  $J_i$  are three-time differentiable and continuous. Consider Problem (1) and the RBJ algorithm. Let  $x^*$  be the minimizer of (1). There exists  $\bar{\epsilon} > 0$  and  $\delta > 0$ , such that, if  $0 < \epsilon < \bar{\epsilon}$  and  $\|x(0) - x^*\| < \delta$ , then the trajectory  $x(t)$ , generated by the RBJ algorithm, converges exponentially fast to  $x^*$ , i.e.,*

$$\|x(t) - x^*\| \leq C \rho^t$$

*for some constants  $C > 0$  and  $0 < \rho < 1$ .*

□

The proof of Theorem 8 can be found in Appendix A, and basically relies on separation of time scales principle between the dynamics of the states  $x_i$ 's and those of the auxiliary variables  $\hat{x}_j^{(i)}$ 's,  $\rho_j^{(i)}$ 's,  $\hat{\rho}_j^{(i)}$ 's,  $\xi_j^{(i)}$ 's and  $\hat{\xi}_j^{(i)}$ 's. Loosely speaking, the result builds on the idea that if the step-size  $\epsilon$  is small enough, the variation of the true states  $x_i$ 's is sufficiently slow and, despite the lossy communication, the values of the auxiliary variables stored in memory equal the true values.

**Remark 9 (Local convergence of the RGD algorithm)** *The same argument used in the previous theorem can be applied to the Robust Gradient Descent Algorithm presented in Remark 5 above under the weaker assumption that the cost functions  $J_i$  are two-time differentiable, thus providing the same local exponential convergence. Typically, the critical value  $\bar{\epsilon}$  for the RGD algorithm is smaller than that of the RBJ algorithm, and consequently also the rate of convergence is slower.*  $\square$

**Lemma 10 (Global convergence RWLS algorithm)** *Let Assumptions 2 and 7 hold. Consider Problem (1) with a quadratic cost function  $J(x)$  and the RWLS algorithm. There exists  $\bar{\epsilon}$  such that, if  $0 < \epsilon < \bar{\epsilon}$ , then, for any  $x(0) \in \mathbb{R}^n$ , the trajectory  $x(t)$ , generated by the RWLS algorithm, converges exponentially fast to the minimizer  $x^*$  of the corresponding problem, i.e.,*

$$\|x(t) - x^*\| \leq C\rho^t$$

for some constants  $C > 0$  and  $0 < \rho < 1$ .  $\square$

## 6 Simulations

In this section we present some simulative results obtained using the RBJ algorithm. The simulations involve the IEEE 123 nodes distribution grid benchmark (see [7]). The problem we address is the robust estimation of the voltage level at each node of the grid (except the PCC node which is assumed fixed and known) from voltage and current measurements in the presence of measurements outliers. We recall that voltages and currents in an AC power distribution grid are complex values. However, in view of the state estimation problem we consider, it is convenient to exploit an equivalent standard reformulation in rectangular coordinates. In particular, given the complex vectors of voltages and currents, denoted as  $\mathbf{v} \in \mathbb{C}^{122}$  and  $\mathbf{i}^c \in \mathbb{C}^{122}$  respectively, and the weighted Laplacian matrix  $\mathcal{L} \in \mathbb{C}^{122 \times 122}$  describing the electric grid, thanks to Kirchhoff's voltage and current laws, it holds that

$$\mathbf{i}^c = \mathcal{L}\mathbf{v}. \quad (22)$$

However, by rewriting voltage and currents in rectangular coordinates as

$$\mathbf{v} := [\Re(\mathbf{v})^\top \Im(\mathbf{v})^\top]^\top \in \mathbb{R}^{244}, \quad \mathbf{i}^c := [\Re(\mathbf{i}^c)^\top \Im(\mathbf{i}^c)^\top]^\top \in \mathbb{R}^{244}.$$

and, similarly, by splitting  $\mathcal{L}$  into its real and imaginary parts as

$$L = \begin{bmatrix} \Re(\mathcal{L}) & -\Im(\mathcal{L}) \\ \Im(\mathcal{L}) & \Re(\mathcal{L}) \end{bmatrix},$$

Eq. (22) is equivalent to

$$\mathbf{i}^c = L\mathbf{v}.$$

Thus, by assuming to collect both current and voltage measurements directly in rectangular coordinates<sup>4</sup>, our measurement model reads as

$$\begin{bmatrix} y^v \\ y^{i^c} \end{bmatrix} = \begin{bmatrix} I \\ L \end{bmatrix} \mathbf{v} + \begin{bmatrix} w^v \\ w^{i^c} \end{bmatrix} + \begin{bmatrix} o^v \\ o^{i^c} \end{bmatrix}, \quad \begin{bmatrix} w^v \\ w^{i^c} \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma_v^2 \text{diag}(|v|) & \\ & \sigma_{i^c}^2 \text{diag}(|i^c|) \end{bmatrix} \right),$$

where  $I \in \mathbb{R}^{244}$  is the identity matrix,  $y^v, y^{i^c} \in \mathbb{R}^{244}$  are the measurements, which we collect in vector  $y \in \mathbb{R}^{488}$ ,  $w^v, w^{i^c} \in \mathbb{R}^{244}$  are the measurements' noise, and  $o^v, o^{i^c} \in \mathbb{R}^{244}$  are sparse vectors which contain possible measurement outliers. We choose<sup>5</sup>  $\sigma_v = 10^{-3}$ [p.u.] and  $\sigma_{i^c} = 10^{-1}$ [p.u.]. Finally, concerning the outliers, 10% of the measurements are corrupted, and the distribution of the outliers is uniform between 1/100 and 1/80 of the respective measurement for voltages and between 1/2 and 1 of the respective current measurement.

As suggested at the end of Section 3, to perform robust state estimation in the presence either of measurements faults or outliers, one interesting choice for the cost function is the modified 1-norm defined is Eq. (8) as

$$\|r\|_{1,\nu}$$

where  $r = y - Av$  are the measurements residuals with  $A = [I \ L^\top]^\top$ . To run the RBJ algorithm we need to identify some partition of the grid. To do so, the feeder is divided into  $N$  non overlapping areas, and a computing unit, which can collect

<sup>4</sup>According to future smart grids paradigm, it is assumed each node of the grid to be equipped with a smart measurement units, e.g., a Phasor Measurement Unit (PMU), which can return measurements of current and voltage. Usually, electric quantities are measured in polar coordinates. However, for the sake of simplicity, we assume to have at our disposal measurements directly in rectangular coordinates, stressing that, thanks to a suitable linearization, it is always possible to pass from polar to rectangular coordinates.

<sup>5</sup>The choice for the measurements error standard deviations is dictated by the fact that the de facto standard for modern PMUs requires at most a 0.1% error in the voltage measurements. This translates in a current error of more or less 10%.

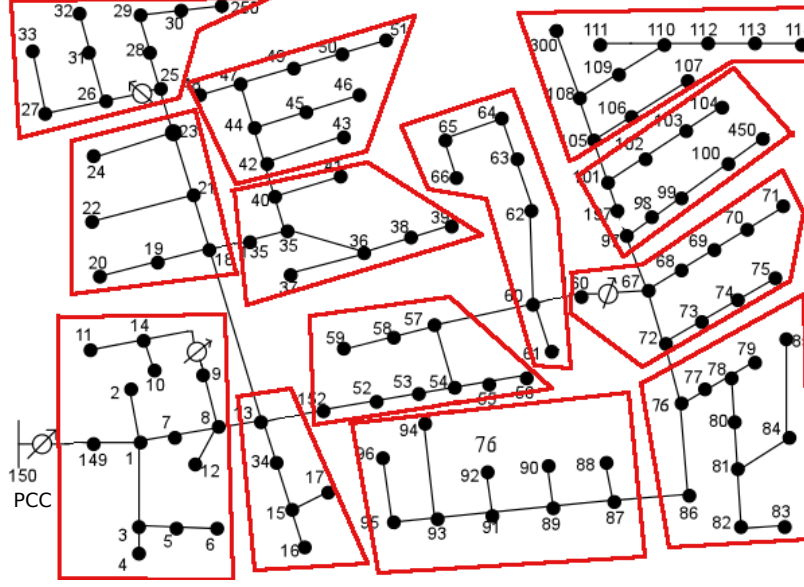


Figure 3: Division in  $N = 13$  areas of the IEEE 123 nodes distribution grid.

the measurements of the nodes belonging to the area and can run the algorithm, is associated to each area. An example of the division in areas is given in Figure 3. The communication graph  $\mathcal{G}$  can be obtained from the division in areas, and in particular, two units can communicate with each other if the two areas are physically connected (that is if there exists two nodes, each one belonging to one of the areas, which are connected by an electric wire). The vectors  $y$ ,  $x$  and  $r$  and matrix  $A$  are divided according to the partition of the nodes in areas. Given the cost function, the values of  $\rho_j^{(i)}$  and  $\xi_j^{(i)}$  are computed as:

$$\begin{aligned}\rho_j^{(i)} &= A_{ji}^\top \left( (\text{diag}(r_j(t)))^2 + \nu I \right)^{-1/2} r_j(t), \quad \forall j \in \mathcal{N}_i^+ \\ \xi_j^{(i)} &= \nu H_{ji}^\top \left( (\text{diag}(r_j(t)))^2 + \nu I \right)^{-3/2} H_{ji}, \quad \forall j \in \mathcal{N}_i^+.\end{aligned}$$

We tested the RBJ algorithm under two different scenarios to evaluate the influence of different parameters involved in the algorithm. All the results showed are obtained averaging over 100 Monte Carlo runs (MCR).

In the first considered scenario we study the influence of the number  $N$  of areas on the performance of the algorithm. Observe that, for the case  $N = 1$  the proposed RBJ algorithm resembles a Newton-Raphson iteration. Thus, in general and as shown in Figure 4, the fewer the number of areas, the faster the convergence rate. In the second scenario we analyze the influence of the step size  $\epsilon$  on the convergence rate. As can be seen from Figure 5, we can infer that the convergence rate improves for bigger values of  $\epsilon$ . Nevertheless, it is important to highlight the fact that the algorithm may diverge if the selected  $\epsilon$  is a too large. Finally, we have considered a third scenario, not reported here, in which the robustness of the algorithm is tested for increasing values of the packet loss probability. The results show that the algorithm is really robust to packet losses as can be seen both from Figures 4–5 where the algorithm has been tested considering a 30% packet loss probability. In particular, since the curves obtained are really close to each other, we decided not to show the results of the simulations. As last remark regarding the packet loss scenario, it is numerically observed that the higher the packet loss probability, the smaller the value of the step size to ensure convergence of the algorithm. Therefore, in the choice of the step size of the algorithm, which is still an open problem, the degree of reliability of the communication network must be considered.

## 7 Conclusions

Considering the emerging area of large-scale multi-agent systems, this paper addressed the always more timely problem of unconstrained robust distributed convex optimization in the presence of communication non idealities. In particular, we analyzed a particular class of locally coupled cost functions which arise in diverse interesting engineering problems such as multi-area state estimation of smart electric grids and multi-robot localization, just to mention two possible applications. We considered a particularly flexible partition-based communication architecture which seamlessly accounts for peer-to-peer and

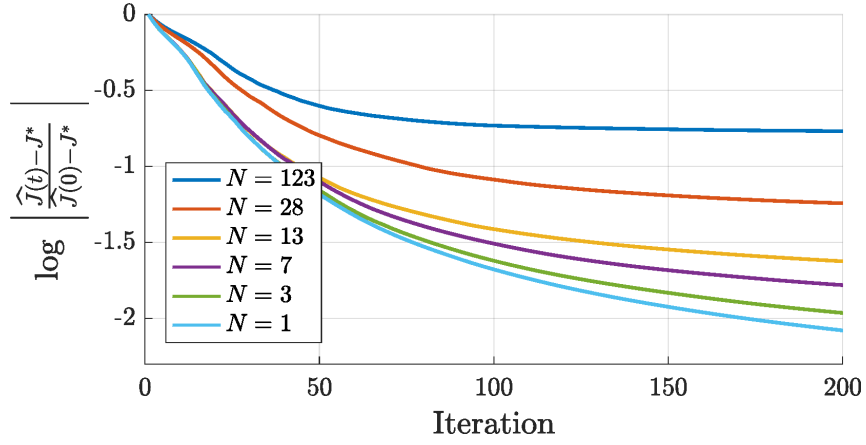


Figure 4: Normalized cost function as a function of the iteration, for different numbers of areas  $N$ . In all simulations there is a packet loss probability of 30% and  $\epsilon = 0.0004$ . The results are obtained averaging over 100 MCR.

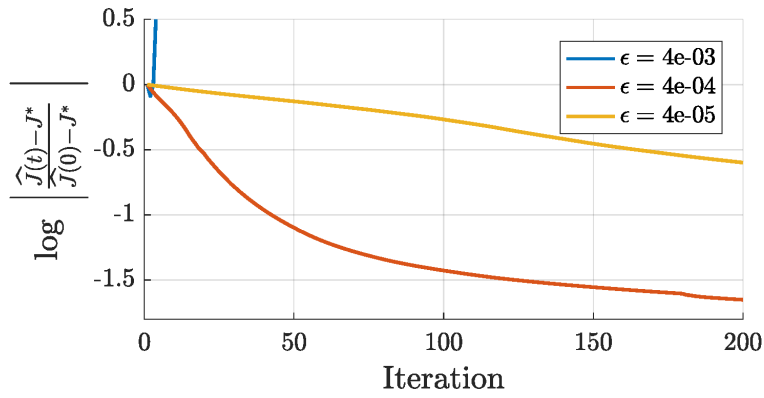


Figure 5: Normalized cost function as a function of the iteration, for different values of the parameter  $\epsilon$ . The number of areas used is  $N = 13$  and the results are obtained averaging over 100 MCR. In all simulations there is a packet loss probability of 30%.



wide-area communications. We proposed a generalized gradient algorithm based on the well-known Jacobi iteration. By leveraging Lyapunov theory and separation of time scale principle, we proved robustness of the algorithm to packet drops and communication failures. Finally, we extensively tested the proposed solution for robust state estimation in the presence of measurements outliers using as benchmark the standard IEEE 123 nodes distribution feeder.

## A Proof of Theorem 8

The proof of Theorem 8 relies on the time scale separation of the dynamic of the  $x_i$ 's and of the auxiliary variables  $\hat{x}_j^{(i)}$ 's,  $\hat{\rho}_j^{(i)}$ 's and  $\hat{\xi}_j^{(i)}$ 's, and fully exploits the following Lemma

**Lemma 11 (Time scale separation principle for discrete time dynamical systems)** *Consider the dynamical system*

$$\begin{bmatrix} x(t+1) \\ y(t+1) \end{bmatrix} = \begin{bmatrix} I & -\epsilon B \\ C(t) & F(t) \end{bmatrix} \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}. \quad (23)$$

Let the following assumptions hold

1. There exists a matrix  $G$  such that  $y = Gx$  satisfies the expression  $y = C(t)x + F(t)y$ ,  $\forall t, \forall x$

2. the system

$$z(t+1) = F(t)z(t) \quad (24)$$

is exponentially stable;

3. the system

$$\dot{x}(t) = -BGx(t) \quad (25)$$

is exponentially stable.

4. The matrices  $C(t)$  and  $F(t)$  are bounded, i.e. there exists  $m > 0$  such that  $\|C(t)\| < m$ ,  $\|F(t)\| < m$ ,  $\forall t \geq 0$ .

Then, there exists  $\bar{\epsilon}$ , with  $0 < \epsilon < \bar{\epsilon}$  such that the origin is an exponentially stable equilibrium for the system (23).  $\square$

**Proof 12 (Proof of Lemma 11)** Let us first consider the following change of variable:

$$z(t) = y(t) - Gx(t)$$

The dynamics of the system in the variables  $x, z$  can be written after some straightforward manipulations as follows:

$$\begin{bmatrix} x(t+1) \\ z(t+1) \end{bmatrix} = \left( \underbrace{\begin{bmatrix} I - \epsilon BG & 0 \\ 0 & F(t) \end{bmatrix}}_{\Sigma(t)} + \epsilon \underbrace{\begin{bmatrix} 0 & -BG \\ GBG & GB \end{bmatrix}}_{\Gamma} \right) \underbrace{\begin{bmatrix} x(t) \\ z(t) \end{bmatrix}}_{\mu(t)} \quad (26)$$

where we used Assumption 1. From Assumption 2, 3 and 3, using converse Lyapunov theorems [22], it follows that there exist positive definite matrices  $P_x > 0$  and  $P_z(t) > 0$  such that

$$-P_x BG - G^T B^T P_x \leq -aI, \quad F(t)^T P_z(t+1) F(t) - P_z(t) \leq -aI, \quad \forall t$$

where  $a$  is a positive scalar and  $P_z(t)$  is bounded, i.e.  $\|P_z(t)\| \leq m$ . We will use the following positive definite Lyapunov function to prove exponential stability of the whole system:

$$U(x, z, t) = x^T P_x x + z^T P_z(t) z = \begin{bmatrix} x^T & z^T \end{bmatrix} \underbrace{\begin{bmatrix} P_x & 0 \\ 0 & P_z(t) \end{bmatrix}}_{P(t)} \begin{bmatrix} x \\ z \end{bmatrix}$$

If we define time difference of the Lyapunov function as  $\Delta U(x, z, t) = U(x(t+1), z(t+1), t+1) - U(x(t), z(t), t)$  we get:

$$\begin{aligned} \Delta U(x, z, t) &= x^T (-\epsilon(P_x BG + G^T B^T P_x) + \epsilon^2 G^T B^T P_x BG) x + \\ &\quad + z^T (F(t)^T P_z(t+1) F(t) - P_z(t)) z + 2\epsilon \mu^T \Sigma^T(t) P(t+1) \Gamma \mu + \epsilon^2 \mu^T \Gamma^T P(t+1) \Gamma \mu \\ &\leq -\epsilon a \|x\|^2 - a \|z\|^2 + \underbrace{\epsilon^2 \|P_x^{\frac{1}{2}} BG\|^2}_{b} \|x\|^2 + 2\epsilon \mu^T \Sigma^T(t) P(t+1) \Gamma \mu + \epsilon^2 \|P^{\frac{1}{2}}(t+1) \Gamma\|^2 \|\mu\|^2 \end{aligned}$$

Note that the top left block of  $\Gamma$  is zero and that  $\Sigma(t)$  and  $P(t)$  are diagonal and bounded for all times. From this it follows that

$$\Sigma^T(t)P(t+1)\Gamma = \begin{bmatrix} 0 & \star \\ \star & \star \end{bmatrix} \implies 2\mu^T \Sigma^T(t)P(t+1)\Gamma \mu \leq c(2\|x\|\|z\| + \|z\|^2)$$

for some positive scalar  $c$ . Boundedness of  $P(t)$  also implies that

$$\|P^{\frac{1}{2}}(t+1)\Gamma\|^2 \|\mu\|^2 \leq d(\|x\|^2 + \|z\|^2)$$

for some positive scalar  $d$ . Putting all together we get

$$\Delta U(x, z, t) \leq \begin{bmatrix} \|x\| & \|z\| \end{bmatrix} \begin{bmatrix} -\epsilon a + b\epsilon^2 & \epsilon c \\ \epsilon c & -a + \epsilon c + \epsilon^2 d \end{bmatrix} \begin{bmatrix} \|x\| \\ \|z\| \end{bmatrix}$$

It follows immediately that there exists a critical  $\bar{\epsilon}$  such that for  $0 < \epsilon < \bar{\epsilon}$  the matrix in the above equation is strictly negative definite and therefore the system is exponentially stable.

We are now ready to state the formal proof of Theorem 8.

**Proof 13 (Proof of Theorem 8)** The proof relies on Lemma 11. In order to improve readability, this proof is broken into few steps. The first step is to write the evolution of the RBJ algorithm as the evolution of a dynamical system. The second step is to find its equilibrium point and to linearize it around this point. The third step is show that the linerized dynamical system satisfies the three assumptions listed in Lemma 11.

RBJ as a dynamical system:

First of all, note that thanks to Assumption 2 the second order derivatives and in particular all the variables  $\xi_j^{(i)}, \hat{\xi}_j^{(i)}$  are always well defined and invertible. Now, let the vectors  $\hat{e}_j^{(i)}$  be the vectorization of  $\hat{\xi}_j^{(i)}$ ,  $\hat{e}_j^{(i)} = \text{vec}(\hat{\xi}_j^{(i)})$ , and the un-vectorization operator  $\text{vec}^{-1}$  as the inverse of the vectorization operator, i.e.  $\text{vec}^{-1}(\hat{e}_j^{(i)}) = \hat{\xi}_j^{(i)}$ . Let  $\hat{x}_i, \hat{\rho}_i$  and  $\hat{e}_i$  be the vectors in which all the  $\hat{x}_j^{(i)}$ 's, the  $\hat{\rho}_j^{(i)}$ 's, and the  $\hat{e}_j^{(i)}$ 's are stacked, respectively, i.e.  $\hat{x}_i = (\hat{x}_{j_1}^{(i)} \cdots \hat{x}_{j_{N_i}}^{(i)})$  and similarly for  $\hat{\rho}_i$  and  $\hat{e}_i$ . Let  $x, \hat{x}, \hat{\rho}, \hat{e}$  be the vectors collecting all the  $x_i, \hat{x}_i$ 's,  $\hat{\rho}_i$ 's and  $\hat{e}_i$ 's, respectively, i.e.  $x = (x_1 \cdots x_N)$  and similarly for  $\hat{x}, \hat{\rho}$  and  $\hat{e}$ .

For every agent  $i$  and neighbours  $j \in \mathcal{N}_i$ , the dynamic of the local variables are given by the following equations:

$$x_i(t+1) = f_1^i(x(t), \hat{\rho}(t), \hat{e}(t)) \quad (27a)$$

$$\hat{x}_j^{(i)}(t+1) = f_2^{ij}(x(t), \hat{x}(t), t) \quad (27b)$$

$$\hat{\rho}_j^{(i)}(t+1) = f_3^{ij}(x(t), \hat{x}(t), \hat{\rho}(t), t) \quad (27c)$$

$$\hat{e}_j^{(i)}(t+1) = f_4^{ij}(x(t), \hat{x}(t), \hat{e}(t), t) \quad (27d)$$

where

$$f_1^i(x, \hat{\rho}, \hat{e}) = x_i - \epsilon \left( \underbrace{\sum_{j \in \mathcal{N}_i^+} \text{vec}^{-1}(\hat{e}_j^{(i)})}_{f_e^i(\hat{e})} \right)^{-1} \left( \underbrace{\sum_{j \in \mathcal{N}_i^+} \hat{\rho}_j^{(i)}}_{f_\rho^i(\hat{\rho})} \right) \quad (28a)$$

$$f_2^{ij}(x, \hat{x}, t) = \begin{cases} \hat{x}_j^{(i)} & \text{if } \gamma_j^{(i)}(t) = 0 \\ x_j & \text{if } \gamma_j^{(i)}(t) = 1 \end{cases} \quad (28b)$$

$$f_3^{ij}(x, \hat{x}, \hat{\rho}, t) = \begin{cases} \hat{\rho}_j^{(i)} & \text{if } \gamma_j^{(i)}(t) = 0 \\ \nabla_i J_j(x_j, \{\hat{x}_k^{(j)}\}_{k \in \mathcal{N}_j}) & \text{if } \gamma_j^{(i)}(t) = 1 \end{cases} \quad (28c)$$

$$f_4^{ij}(x, \hat{x}, \hat{e}, t) = \begin{cases} \hat{e}_j^{(i)} & \text{if } \gamma_j^{(i)}(t) = 0 \\ \text{vec} \left( \nabla_{ii}^2 J_j(x_j, \{\hat{x}_k^{(j)}\}_{k \in \mathcal{N}_j}) \right) & \text{if } \gamma_j^{(i)}(t) = 1 \end{cases} \quad (28d)$$

Note that the variables  $\rho_j^{(i)}$  and  $\xi_j^{(i)}$  do not appear in the dynamics since they are deterministic functions of the variables  $x$  and  $\hat{x}$ , and therefore can be omitted.

Equilibrium point and linearization:

Let  $x^*$  be the minimizer of the optimization problem and let us define

$$\begin{aligned} H_{hk} &= \nabla_{hk}^2 J(x^*) = \sum_{j=1}^N \underbrace{\nabla_{hk}^2 J_j(x_j^*, \{x_k^*\}_{k \in \mathcal{N}_j})}_{H_{hk}^j} = \sum_{j=1}^N H_{hk}^j \\ \hat{x}_j^{(i)*} &= x_j^* \\ \hat{\rho}_j^{(i)*} &= \nabla_i J_j(x_j^*, \{x_k^*\}_{k \in \mathcal{N}_j}) \\ \hat{e}_j^{(i)*} &= \text{vec}(\nabla_{ii}^2 J_j(x_j^*, \{x_k^*\}_{k \in \mathcal{N}_j})) = \text{vec}(H_{ii}^j) \end{aligned}$$

Notice that  $\sum_{j=1}^N \hat{\rho}_j^{(i)*} = \nabla_i J(x^*) = 0$ , since the gradient computed at the minimizer is zero. It is now simple to verify by direct inspection that  $(x^*, \hat{x}^*, \hat{\rho}^*, \hat{e}^*)$  is an equilibrium point for the dynamical system described by (27). Next, we will analyze the behaviour of system (27) in the neighborhood of the equilibrium point  $(x^*, \hat{x}^*, \hat{\rho}^*, \hat{e}^*)$ . Consider the change of variables

$$\begin{aligned} \psi &= x - x^* \\ \hat{\psi} &= \hat{x} - \hat{x}^* \\ \hat{\eta} &= \hat{\rho} - \hat{\rho}^* \\ \hat{\zeta} &= \hat{e} - \hat{e}^* \end{aligned} \quad (29)$$

If we linearize equations (27) around  $(x^*, \hat{x}^*, \hat{\rho}^*, \hat{e}^*)$ , we obtain

$$\psi_i(t+1) \simeq \psi_i(t) - \epsilon H_{ii}^{-1} \sum_{j \in \mathcal{N}_i^+} \hat{\eta}_j^{(i)} \quad (30)$$

$$\hat{\psi}_j^{(i)}(t+1) \simeq \begin{cases} \hat{\psi}_j^{(i)}(t) & \text{if } \gamma_j^{(i)}(t) = 0 \\ \psi_j(t) & \text{if } \gamma_j^{(i)}(t) = 1 \end{cases} \quad (31)$$

$$\hat{\eta}_j^{(i)}(t+1) \simeq \begin{cases} \hat{\eta}_j^{(i)}(t) & \text{if } \gamma_j^{(i)}(t) = 0 \\ H_{ij}^j \psi_j(t) + \sum_{k \in \mathcal{N}_j} H_{ik}^j \hat{\psi}_k^{(i)}(t) & \text{if } \gamma_j^{(i)}(t) = 1 \end{cases} \quad (32)$$

$$\hat{\zeta}_j^{(i)}(t+1) \simeq \begin{cases} \hat{\zeta}_j^{(i)}(t) & \text{if } \gamma_j^{(i)}(t) = 0 \\ K_{ij}^j \psi_j(t) + \sum_{k \in \mathcal{N}_j} K_{ik}^j \hat{\psi}_k^{(i)}(t) & \text{if } \gamma_j^{(i)}(t) = 1. \end{cases} \quad (33)$$

where in Eqn. (30) we used the fact that  $\left. \frac{\partial f_i^i}{\partial \hat{e}} \right|_{x^*, \hat{\rho}^*, \hat{e}^*} = -\epsilon \left. \frac{\partial (f_e^i)^{-1}}{\partial \hat{e}} f_\rho^i \right|_{x^*, \hat{\rho}^*, \hat{e}^*} = 0$  since  $f_\rho^i|_{x^*, \hat{\rho}^*, \hat{e}^*} = \nabla_i J(x^*) = 0$ , and the fact that  $f_e^i(\hat{e}^*) = H_{ii}$ . In Eqn.(32) we used the fact that  $H_{ik}^j = \nabla_{ik}^2 J_j(x_j^*, \{x_k^*\}_{k \in \mathcal{N}_j})$ . Finally, in Eqn. (33) the matrices  $K_{ik}^j$  depends on third order derivatives of  $J(x)$  whose values are unimportant for the analysis of the stability of the dynamics. By collecting all the variables together, we obtain the system

$$\begin{aligned} \begin{bmatrix} \psi(t+1) \\ \hat{\psi}(t+1) \\ \hat{\eta}(t+1) \\ \hat{\zeta}(t+1) \end{bmatrix} &= \begin{bmatrix} I & 0 & -\epsilon B & 0 \\ C_1(t) & F_1(t) & 0 & 0 \\ C_2(t) & F_2(t) & F_3(t) & 0 \\ C_3(t) & F_4(t) & 0 & F_5(t) \end{bmatrix} \begin{bmatrix} \psi(t) \\ \hat{\psi}(t) \\ \hat{\eta}(t) \\ \hat{\zeta}(t) \end{bmatrix} \\ \begin{bmatrix} \psi(t+1) \\ y(t+1) \end{bmatrix} &= \begin{bmatrix} I & -\epsilon B \\ C(t) & F(t) \end{bmatrix} \begin{bmatrix} \psi(t) \\ y(t) \end{bmatrix}. \end{aligned} \quad (34)$$

where  $y = (\hat{\psi}, \hat{\xi}, \hat{\zeta})$  collects the fast dynamic variables. Notice that  $F_1(t)$ ,  $F_3(t)$  and  $F_5(t)$  are diagonal matrices whose entries are either 1 or 0, depending on the communication between agent success, and, as a consequence,  $F(t)$  is a lower triangular matrix,  $\forall t$ .

Assumption 1 of Lemma 11:

We now start proving that the linearized dynamics above satisfies the three assumptions of Lemma 11 where  $\psi$  plays the role of  $x$  in the Lemma. It is simple to verify by direct inspection that for a fixed  $\psi$ , the following maps satisfy Assumption 1 of

Lemma 11:

$$\hat{\psi}_j^{(i)} = \psi_j \quad (35)$$

$$\hat{\eta}_j^{(i)} = H_{ij}^j \psi_j + \sum_{k \in \mathcal{N}_j} H_{ik}^j \psi_k \quad (36)$$

$$\hat{\zeta}_j^{(i)} = K_{ij}^j \psi_j + \sum_{k \in \mathcal{N}_j} K_{ik}^j \psi_k \quad (37)$$

in fact, this is equivalent of saying that there exists a matrix  $G$  such that  $y = G\psi$  satisfies the equality  $y = C(t)\psi + F(t)y$  for all  $\psi$  and  $t$ .

Assumption 2 of Lemma 11:

Let us now consider the fast dynamics of the system given by the following system:

$$z(t) = F(t-1) \cdots F(0)z(0) = \Omega(t)z(0)$$

Assumption 7, on the persistent communication among the agents, assures that

$$\begin{aligned} F_1(T-1) \cdots F_1(0) &= \Omega_1(T) = 0 \\ F_3(T-1) \cdots F_3(0) &= \Omega_3(T) = 0 \\ F_5(T-1) \cdots F_5(0) &= \Omega_5(T) = 0 \end{aligned}$$

in fact when  $\gamma_j^{(i)}(t) = 1$ , the corresponding rows in the matrices  $F_1(t), F_3(t), F_5(t)$  become zero, and this property will be inherited also by the product matrices  $\Omega_1(T), \Omega_3(T), \Omega_5(T)$  since all  $F_1(t), F_2(t), F_3(t)$  are diagonal. Since all  $\gamma_j^{(i)}(t)$  will be equal to one at least once within the window  $t \in [0, \dots, T-1]$ , then the matrices  $\Omega_1(T), \Omega_3(T), \Omega_5(T)$  must be all zero. Finally, since the matrix  $F(t)$  is lower triangular, we have that after a maximum of  $(2T+1)$  iterations the product matrix  $\Omega(2T+1)$  will be zero and thus  $z(2T+1) = 0$ . That is, the fast variable dynamic is exponentially stable, since it reaches the equilibrium in a finite number of iteration.

Assumption 3 of Lemma 11:

Finally, consider the slow dynamical system

$$\dot{\psi}(t) = -BG\psi(t). \quad (38)$$

which by direct substitution from the previous analysis can be locally written as:

$$\dot{\psi}_i(t) = -H_{ii}^{-1} \left( \sum_{j \in \mathcal{N}_i^+} \left( H_{ij}^j \psi_j + \sum_{k \in \mathcal{N}_i} H_{ik}^j \psi_k \right) \right) = -H_{ii}^{-1} H^i \psi$$

where  $H$  was defined above and corresponds to the Hessian of the global cost  $J$  computed at  $x^*$ , i.e.  $H = \nabla^2 J(x^*)$  and  $H^i$  is its  $i$ -th block-row, i.e.,  $H^i = [\nabla_{i1}^2 J(x^*) \cdots \nabla_{iN}^2 J(x^*)]$ . This implies that

$$BG = (\text{diag}(H))^{-1} H,$$

therefore, if we choose

$$V(\psi) = \frac{1}{2} \psi^\top H \psi,$$

as a Lyapunov function, it is straightforward to see that system (38) is asymptotically stable since  $\dot{V}(\psi(t)) = -\psi^\top(t) H (\text{diag}(H))^{-1} H \psi(t)$ ,  $0, x \neq 0$  being  $H > 0$  by assumption.

Assumption 4 of Lemma 11:

This comes from the observation that the time-variance of the state matrices depends on the specific sequence of packet losses that can occur. Since there are only a finite number of possible different sequences, the assumption is clearly satisfied.

Concluding, system (34) satisfies the hypothesis of Lemma 11, and thus there exists  $\bar{\epsilon}$ , with  $0 < \epsilon < \bar{\epsilon}$  such that, by using the resilient block Jacobi Algorithm 2,

$$\lim_{t \rightarrow \infty} x(t) = x^*.$$

locally exponentially fast.

## References

- [1] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.
- [2] A. Nedic and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *Automatic Control, IEEE Transactions on*, vol. 54, no. 1, pp. 48–61, 2009.
- [3] A. Nedic, A. Ozdaglar, and P. A. Parrilo, “Constrained consensus and optimization in multi-agent networks,” *Automatic Control, IEEE Transactions on*, vol. 55, no. 4, pp. 922–938, 2010.
- [4] D. E. Marelli and M. Fu, “Distributed weighted least-squares estimation with fast convergence for large-scale systems,” *Automatica*, vol. 51, pp. 27–39, 2015.
- [5] I. D. Schizas, A. Ribeiro, and G. B. Giannakis, “Consensus in ad hoc wsns with noisy linkspart i: Distributed estimation of deterministic signals,” *Signal Processing, IEEE Transactions on*, vol. 56, no. 1, pp. 350–364, 2008.
- [6] V. Kekatos and G. B. Giannakis, “Distributed robust power system state estimation,” *Power Systems, IEEE Transactions on*, vol. 28, no. 2, pp. 1617–1626, 2013.
- [7] S. Bolognani, R. Carli, and M. Todescato, “State estimation in power distribution networks with poorly synchronized measurements,” in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, Dec 2014, pp. 2579–2584.
- [8] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.
- [9] E. Wei and A. Ozdaglar, “On the  $O(1/k)$  convergence of asynchronous distributed alternating direction method of multipliers,” in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 2013, pp. 551–554.
- [10] T. Erseghe, “A distributed and scalable processing method based upon admm,” *Signal Processing Letters, IEEE*, vol. 19, no. 9, pp. 563–566, 2012.
- [11] F. Iutzeler, P. Bianchi, P. Ciblat, and W. Hachem, “Asynchronous distributed optimization using a randomized alternating direction method of multipliers,” in *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*. IEEE, 2013, pp. 3671–3676.
- [12] P. Bianchi, W. Hachem, and F. Iutzeler, “A stochastic coordinate descent primal-dual algorithm and applications to large-scale composite optimization,” *arXiv preprint arXiv:1407.0898*, 2014.
- [13] M. Zargham, A. Ribeiro, A. Ozdaglar, and A. Jadbabaie, “Accelerated dual descent for network flow optimization,” *Automatic Control, IEEE Transactions on*, vol. 59, no. 4, pp. 905–920, 2014.
- [14] F. Zanella, D. Varagnolo, A. Cenedese, G. Pillonetto, and L. Schenato, “Newton-raphson consensus for distributed convex optimization,” in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011, pp. 5917–5922.
- [15] A. J. Conejo, S. de la Torre, and M. Canas, “An optimization approach to multiarea state estimation,” *IEEE Transactions on Power Systems*, vol. 1, no. 22, pp. 213–221, 2007.
- [16] A. Carron, M. Todescato, R. Carli, and L. Schenato, “An asynchronous consensus-based algorithm for estimation from noisy relative measurements,” *IEEE Transactions on Control of Network Systems*, vol. 1, no. 3, pp. 283 – 295, 2014.
- [17] N. Bof, M. Todescato, R. Carli, and L. Schenato, “Robust distributed estimation for localization in lossy sensor networks,” in *6th IFAC Workshop on Distributed Estimation and control in Networked Systems (NecSys16)*. IFAC, 2016, pp. 250–255.
- [18] D. P. Palomar and M. Chiang, “A tutorial on decomposition methods for network utility maximization,” *Selected Areas in Communications, IEEE Journal on*, vol. 24, no. 8, pp. 1439–1451, 2006.
- [19] M. Todescato, G. Cavarero, R. Carli, and L. Schenato, “A robust block-jacobi algorithm for quadratic programming under lossy communications,” in *IFAC Workshop on Distributed Estimation and Control in Networked Systems (NecSys)*, 2015.

- [20] S.-Y. Bin and C.-H. Lin, "An implementable distributed state estimator and distributed bad data processing schemes for electric power systems," *IEEE transactions on power systems*, vol. 9, no. 3, pp. 1277–1284, 1994.
- [21] M. Arguez, C. Ramirez, and R. Sanchez, "An  $\ell_1$ -algorithm for underdetermined systems and applications," in *Fuzzy Information Processing Society (NAFIPS), 2011 Annual Meeting of the North American*. IEEE, 2011, pp. 1–6.
- [22] H. K. Khalil, *Nonlinear Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.