A Hessian inversion-free exact second order method for distributed consensus optimization

Dušan Jakovetić, Member, IEEE, Nataša Krejić, Nataša Krklec Jerinkić

Abstract—We consider a standard distributed consensus optimization problem where a set of agents connected over an undirected network minimize the sum of their individual (local) strongly convex costs. Alternating Direction Method of Multipliers (ADMM) and Proximal Method of Multipliers (PMM) have been proved to be effective frameworks for design of exact distributed second order methods (involving calculation of local cost Hessians). However, existing methods involve explicit calculation of local Hessian inverses at each iteration that may be very costly when the dimension of the optimization variable is large. In this paper, we develop a novel method, termed INDO (Inexact Newton method for Distributed Optimization), that alleviates the need for Hessian inverse calculation. INDO follows the PMM framework but, unlike existing work, approximates the Newton direction through a generic fixed point method (e.g., Jacobi Overrelaxation) that does not involve Hessian inverses. We prove exact global linear convergence of INDO and provide analytical studies on how the degree of inexactness in the Newton direction calculation affects the overall method’s convergence factor. Numerical experiments on several real data sets demonstrate that INDO’s speed is on par (or better) as state of the art methods iteration-wise, hence having a comparable communication cost. At the same time, for sufficiently large optimization problem dimensions \( n \) (even at \( n \) on the order of couple of hundreds), INDO achieves savings in computational cost by at least an order of magnitude.

Index Terms—Inexact Newton, proximal method of multipliers, distributed optimization, exact convergence, strongly convex problems.

I. INTRODUCTION

We consider problems of the form

\[
\min_{y \in \mathbb{R}^n} \sum_{i=1}^{N} f_i(y). \tag{1}
\]

Here, \( f_i : \mathbb{R}^n \rightarrow \mathbb{R}, i = 1, ..., N \), is a strongly convex local cost function assigned to a node within a network of distributed agents able to perform local operations and communicate with their neighbours. Formulation (1) finds a distributed solution of systems of linear equations \( \mathbf{Ax} = \mathbf{b} \); instead, only diagonal elements of the Hessians are inverted. Another appealing feature of the approach is compatibility with the theory of inexact Newton methods in centralized optimization, \([9]\). The main advantage of the inexact Newton methods is that they avoid oversolving of the Newtonian system of linear equations; as such, they have been employed in related problems, for example in minimizing finite sums in machine learning applications \([4]\). To be more specific, we consider the constrained reformulation of (1) in the augmented space (like in, e.g., \([21]\)) and build up on the distributed second order approximation of the Augmented Lagrangian.

The authors are with the University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics, Trg Dositeja Obradovica 4, Novi Sad, Serbia. Authors’ emails: dusan.jakovetic@dzi.uns.ac.rs; natasuk@uns.ac.rs; natasa.krklec@dmi.uns.ac.rs. This work is supported by the Ministry of Education, Science and Technological Development, Republic of Serbia. It is also supported in part by the Bilateral Cooperation Serbia – Croatia, project “Optimization Method Application in Biomedicine,” and by the European Union’s Horizon 2020 Research and Innovation program under grant agreement No 871518. The paper reflects only the view of the authors and the Commission is not responsible for any use that may be made of the information it contains.
in [20], [21] might be expensive, and thus an iterative solver for the system of linear equations is a natural choice. The convergence theory developed here relies on the theory for the proximal methods of multipliers, as the proposed method fits this general framework, following, e.g., [21]. However, the error analysis carried out here – reflecting the inexact solution of the Newtonian system of linear equations and how it affects the overall proximal multipliers convergence – is very different for the proposed method.

To summarize, the main contributions of this paper are the definition and convergence theory of the novel exact second order method termed INDO (Inexact Newton method for Distributed Optimization)\footnote{To avoid confusion, we clarify that the wording “inexact Newton” here refers to approximately solving the Newtonian system of linear equations that defines the Newton direction; on the other hand, the wording “exact method” here signifies that INDO converges to the exact solution of (1).}

Linear convergence to the exact solution is shown for strongly convex problems under a set of standard assumptions. The rate of convergence is also analysed, assuming that the linear solver is of fixed-point type, in particular we focused on the application of the Jacobi Over-relaxation method to derive an estimate of the convergence factor. The convergence factor is compared with the factor corresponding to ESOM method, [21], as ESOM operates within the same framework – proximal method of multipliers with approximate second order direction. Furthermore, we analyse computational and communication costs of these two methods and show that the computational costs of INDO can be significantly smaller than the costs of ESOM while the communication costs per iteration are the same. These claims are illustrated by numerical results for quadratic and logistic functions.

This paper is organized as follows. In Section 2 we introduce the problem model, the primal-dual framework based on the proximal method of multipliers and state the assumptions on the problem. The INDO method is introduced and theoretically analysed in Section 3. Details on distributed implementation as well as the analysis of the convergence factor are presented in Section 4, while Section 5 is devoted to practical implementation of INDO with the analysis of computational costs and comparison with ESOM [21] method through standard examples from machine learning. Some conclusions are drawn in Section 6. Finally, lengthy supporting proofs are delegated to the Appendix.

II. PRELIMINARIES

The notation used is the paper is the following. We use upper case blackboard bold letters to denote matrices in $\mathbb{R}^{N \times N}$, e.g., $A, B, \ldots$. For $A \in \mathbb{R}^{nN \times nN}$, we use its block elements representation $A = [A_{ij}]$, $A_{ij} \in \mathbb{R}^{n \times n}$. We denote scalar elements of $A$ by $a_{ij} \in \mathbb{R}$. Similarly, upper diagonal matrices $A, B, \ldots$ are used for stand-alone matrices in $\mathbb{R}^{n \times n}$. The vectors in $\mathbb{R}^{nN}$ are denoted by bold lowercase letters, e.g., $x \in \mathbb{R}^{nN}$; their component blocks are $x_i \in \mathbb{R}^n$; similarly, we use normal lowercase letters, e.g., $y$, for stand-alone vectors in $\mathbb{R}^n$. The notation $\| \cdot \|_2$ stands for the 2-norm of its vector or matrix argument; if the subscript is omitted, $\| \cdot \|$ also designates the 2-norm, if not stated otherwise. We denote by $\text{diag}(A)$ the diagonal matrix with diagonal elements equal to those of matrix $A$, and by $\sigma(A)$ the spectral radius of $A$.

The network of connected computational nodes (agents) is represented by a graph $G = (V, E)$, where $V$ is the set of nodes $\{1, \ldots, N\}$ and $E$ is the set of undirected edges $(i,j)$. Denote by $O_i$ the set of neighbors of node $i$ and let $\hat{O}_i = O_i \cup \{i\}$. The communication network is accompanied by a communication (weight) matrix $W$ with the following properties.

\begin{itemize}
  \item $A_1$: The matrix $W \in \mathbb{R}^{N \times N}$ is symmetric, doubly stochastic and
  \[ w_{ij} > 0 \text{ if } j \in \hat{O}_i, \quad w_{ij} = 0 \text{ if } j \notin \hat{O}_i \]
\end{itemize}

Let us assume that each of the $N$ nodes has its local cost function $f_i$ and has access to computing its first and second derivatives. Under the assumption $A_1$, the problem (1) has the equivalent form
\begin{equation}
\min_{x \in \mathbb{R}^{nN}} f(x) := \sum_{i=1}^N f_i(x_i) \quad \text{s. t.} \quad (\mathbb{I} - W)^{1/2} x = 0, \tag{2}
\end{equation}
where $x = (x_1; \ldots; x_N) \in \mathbb{R}^{nN}$, $W = W \otimes I \in \mathbb{R}^{nN \times nN}$ and $I \in \mathbb{R}^{nN \times nN}$ is the identity matrix.

A 2: The functions $f_i$’s are twice continuously differentiable and the eigenvalues of the local Hessians are bounded by positive constants $0 < \alpha \leq M < \infty$, i.e.,
\begin{equation}
\alpha I \preceq \nabla^2 f_i(y) \preceq MI, \quad \text{for all } y \in \mathbb{R}^n \text{ and } i = 1, \ldots, N. \tag{3}
\end{equation}

The above assumption implies that the function $f(x)$ is also strongly convex with constant $m$ and its gradient $\nabla f$ is Lipschitz continuous with constant $M$. Given that we consider a method of Newton-type, the assumption on Lipschitz continuous Hessian is also needed, as usual in the theory of Newton methods and in [21].

A 3: The Hessian of the objective function $\nabla^2 f(x)$ is Lipschitz continuous, i.e., there exists $L > 0$ such that
\begin{equation}
\| \nabla^2 f(x) - \nabla^2 f(z) \| \leq L \| x - z \|, \quad x, z \in \mathbb{R}^{nN}. \tag{4}
\end{equation}

The following Lemma from [22] will be used in the convergence analysis.

\begin{lemma}
Assume that $A_2A_3$ hold. Then for all $x, z \in \mathbb{R}^{nN}$
\begin{equation}
\| \nabla f(x) - \nabla f(z) + \nabla^2 f(x)(z - x) \| \\
\leq \min \{2M, \frac{L}{2} \| x - z \| \} \| x - z \|. \tag{5}
\end{equation}
\end{lemma}

Now, the Augmented Lagrangian function of (2) is defined as
\begin{equation}
\mathcal{L}(x, v) = f(x) + v^T (\mathbb{I} - W)^{1/2} x + \frac{\alpha}{2} x^T (\mathbb{I} - W) x, \tag{6}
\end{equation}
where $\alpha > 0$ is a constant and $v$ is dual variable. For a strictly positive proximal coefficient $\varepsilon$, the proximal method of multipliers is defined by the primal update
\begin{equation}
x^{k+1} = \arg \min \mathcal{L}(x, v^k) + \frac{\varepsilon}{2} \| x - x^k \|^2, \tag{7}
\end{equation}
while the dual update with the stepsize $\alpha$ is
\begin{equation}
\mathbf{v}^{k+1} = \mathbf{v}^{k} + \alpha (\mathbb{I} - \mathbb{W})^{1/2} \mathbf{x}^{k+1}.
\end{equation}

The primal step is not implementable in the distributed environment due to the augmented term $\frac{1}{2} \mathbf{x}^T (\mathbb{I} - \mathbb{W}) \mathbf{x}$ and therefore an approximation of $\mathcal{L}(\mathbf{x}, \mathbf{v})$ is needed. We consider the second order Taylor approximation with respect to $\mathbf{x}$ at the point $(\mathbf{x}^k, \mathbf{v}^k)$,

\begin{equation}
\mathcal{L}(\mathbf{x}, \mathbf{v}^k) \approx \mathcal{L}(\mathbf{x}^k, \mathbf{v}^k) + \mathbf{g}^k (\mathbf{x} - \mathbf{x}^k) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^k)^T \mathbb{H}^k (\mathbf{x} - \mathbf{x}^k),
\end{equation}

with
\begin{equation}
\mathbb{H}^k = \nabla^2 f(\mathbf{x}^k) + \alpha (\mathbb{I} - \mathbb{W}) + \epsilon \mathbb{I}.
\end{equation}

and
\begin{equation}
\mathbf{g}^k := \nabla_x \mathcal{L}(\mathbf{x}^k, \mathbf{v}^k) = \nabla f(\mathbf{x}^k) + (\mathbb{I} - \mathbb{W})^{1/2} \mathbf{v}^k + \alpha (\mathbb{I} - \mathbb{W}) \mathbf{x}^k.
\end{equation}

Using the second order Taylor approximation, one can define the Newton step $\mathbf{d}_N^k$ as
\begin{equation}
\mathbb{H}^k \mathbf{d}_N^k = -\mathbf{g}^k,
\end{equation}

and form the primal update as $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{d}_N^k$. Notice that the matrix $\mathbb{H}^k$ has the sparsity structure of the graph; hence, one can apply an iterative method of the fixed point type to solve (7). The details of such procedure can be found in [14]. However, solving (7) exactly might be too costly and an approximate solution to (7) is in fact sufficient for the convergence. Thus the step we use is defined by

\begin{equation}
\mathbb{H}^k \mathbf{d}^k = -\mathbf{g}^k + \mathbf{r}^k,
\end{equation}

for some residual $\mathbf{r}^k$ which obeys the classical Inexact Newton forcing condition [9]

\begin{equation}
\|\mathbf{r}^k\| \leq \eta_k \|\mathbf{g}^k\|.
\end{equation}

for a forcing term $\eta_k \geq 0$.

The dual update is given by (4) but the matrix $\alpha (\mathbb{I} - \mathbb{W})^{1/2}$ is not neighbor sparse. Thus we apply the same variable transformation as in [21], defining a sequence of variables $\mathbf{q}^k$ as

\begin{equation}
\mathbf{q}^k = (\mathbb{I} - \mathbb{W})^{1/2} \mathbf{v}^k.
\end{equation}

Now, multiplying the dual update (4) by $(\mathbb{I} - \mathbb{W})^{1/2}$ from the left and using the definition of $\mathbf{q}^k$ we obtain
\begin{equation}
\mathbf{q}^{k+1} = \mathbf{q}^k + \alpha (\mathbb{I} - \mathbb{W}) \mathbf{x}^{k+1},
\end{equation}

which is computable in the distributed manner. Notice that with this change of variables we get
\begin{equation}
\mathbf{g}^k = \nabla f(\mathbf{x}^k) + \mathbf{q}^k + \alpha (\mathbb{I} - \mathbb{W}) \mathbf{x}^k.
\end{equation}

\section{Distributed Inexact Newton Algorithm}

With the notation introduced in the previous section we are now in a position to state the general INDO method. Further details regarding the inexact Newton step computation (step S1 in the INDO method below), are postponed to Section 4. Therein, we discuss different possibilities for fulfilling the Inexact Newton condition (10) in a distributed environment. Assume that the forcing sequence $\{\eta_k\}_{k=0}^{\infty}$ is nonnegative.

**INDO method**

**Input:** $\mathbf{x}^0 \in \mathbb{R}^{nN}$, $\mathbf{q}^0 = 0\), $k = 0, \{\eta_k\}_{k=0}^{\infty}, \alpha > 0, \epsilon > 0$

S1 Find a step $\mathbf{d}^k$ such that
\begin{equation}
\mathbb{H}^k \mathbf{d}^k = -\mathbf{g}^k + \mathbf{r}^k, \|\mathbf{r}^k\| \leq \eta_k \|\mathbf{g}^k\|.
\end{equation}

S2 Compute the primal update
\begin{equation}
\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{d}^k.
\end{equation}

S3 Compute the dual update
\begin{equation}
\mathbf{q}^{k+1} = \mathbf{q}^k + \alpha (\mathbb{I} - \mathbb{W}) \mathbf{x}^{k+1},
\end{equation}

set $k = k + 1$ and return to S1.

As already stated, one can easily see that the algorithm fits the general proximal multipliers framework [21], with the primal step defined in S1-S2 and the dual update in S3. The key novelty is the direction computation in S1 where we generate a suitable inexact direction such that the residual is small enough with respect to the gradient norm, (10). This important property actually allows for flexible approach in the linear solver and avoids oversolving. That is, the proposed approach allows using a relatively large tolerance while the gradient is large, and implies stringent tolerance condition as the value of gradient decreases. The details of S1 implementation are presented in Section 4, where we also give INDO algorithm from the perspective of each node in the network.

Due to the convexity and the fact that we only have equality constraints, $\mathbf{x}^*$ is a solution of problem (2) if and only if there exists $\mathbf{v}^*$ such that
\begin{equation}
\nabla f(\mathbf{x}^*) + (\mathbb{I} - \mathbb{W})^{1/2} \mathbf{v}^* = 0,
\end{equation}

\begin{equation}
(\mathbb{I} - \mathbb{W})^{1/2} \mathbf{v}^* = 0.
\end{equation}

Strong convexity implies that $(\mathbf{x}^*, \mathbf{v}^*)$ is unique.

The convergence analysis we develop below relies on the reasoning from [21] in general but the error analysis, with respect to the proximal method of multipliers (see Section 2) is fundamentally different. We present the proof through a sequence of technical Lemmas, some of which correspond to parts of proofs in [21] that are rather similar. Proofs of the Lemmas similar to [21] are provided in Appendix for completeness.

**Lemma 3.1:** Let $\{\Pi, N\}$ hold. A sequence generated by Algorithm INDO satisfies
\begin{equation}
\mathbf{v}^{k+1} - \mathbf{v}^k - \alpha (\mathbb{I} - \mathbb{W})^{1/2} (\mathbf{x}^{k+1} - \mathbf{x}^*) = 0.
\end{equation}

**Proof.** Given the rule for dual update in S3 and the definition of $\mathbf{q}^k$ in (8), we have
\begin{equation}
\mathbf{v}^{k+1} - \mathbf{v}^k = \alpha (\mathbb{I} - \mathbb{W})^{1/2} \mathbf{x}^{k+1}.
\end{equation}
Adding the zero term from (12) multiplied by $\alpha$, i.e., $\alpha(I - \mathcal{W})^{1/2}x^k$, implies the statement.

**Lemma 3.2:** Let assumptions $A[1,3]$ hold and $\{x^k, q^k\}$ be a sequence generated by Algorithm INDO. Then
\[
\nabla f(x^{k+1}) - \nabla f(x^k) + \varepsilon d^k + (I - \mathcal{W})^{1/2}(\nabla x^{k+1} - \nabla x^k) + e^k = 0,
\]
where
\[
e^k = \nabla^2 f(x^k)d^k + \nabla f(x^k) - \nabla f(x^{k+1}) - r^k, \quad r^k = \|d^k\|g^k + g^k.
\]

**Proof.** By the definition of step $d^k$ in S2, (8) and (11) we have
\[
\begin{align*}
0 & = \|d^k\| + g^k - r^k + \varepsilon d^k \\
& = (\nabla^2 f(x^k) + \alpha(I - \mathcal{W}) + \varepsilon I)d^k \\
& + \alpha(I - \mathcal{W})d^k - r^k + \varepsilon d^k \\
& + \alpha(I - \mathcal{W})x^k - r^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) \\
& = \nabla f(x^{k+1}) - \nabla f(x^k) + \varepsilon d^k \\
& + (I - \mathcal{W})^{1/2}v^{k+1} - v^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) + \varepsilon d^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) + \varepsilon d^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) + \varepsilon d^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) + \varepsilon d^k \\
& - (\nabla f(x^k) - \nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) + \varepsilon d^k.
\end{align*}
\]

\]

**Lemma 3.3:** Let assumptions $A[1,3]$ hold and $\{x^k, q^k\}$ be the sequence generated by INDO. Then
\[
\|e^k\| \leq \min\{2M, \frac{L}{2}\|d^k\|\|d^k\| + \eta_k(2\alpha + M)\|x^k - x^*\| + \sqrt{2}\eta_k\|v^k - v^*\|.
\]

**Proof.** By Lemma 2.1, Lemma 3.2 and the condition in S2 of the algorithm we have
\[
\begin{align*}
\|e^k\| & \leq \|\nabla^2 f(x^k)d^k + \nabla f(x^k) - \nabla f(x^{k+1})\| + \|r^k\| \\
& \leq \min\{2M, \frac{L}{2}\|d^k\|\|d^k\| + \eta_k\|g^k\|.
\end{align*}
\]

Furthermore, the definition of $g^k$ in (11) and optimality conditions (11), (12) imply
\[
\begin{align*}
\|g^k\| & = \|\nabla f(x^k) + (I - \mathcal{W})^{1/2}v^k + \alpha(I - \mathcal{W})x^k \\
& - (\nabla f(x^*) + (I - \mathcal{W})^{1/2}v^*) - \alpha(I - \mathcal{W})x^k\| \\
& \leq \|\nabla f(x^k) - \nabla f(x^*)\| + \|(I - \mathcal{W})^{1/2}(v^k - v^*)\| \\
& + \|\alpha(I - \mathcal{W})(x^k - x^*)\| \\
& \leq M\|x^k - x^*\| + \sqrt{2}\|v^k - v^*\| \\
& + 2\alpha\|x^k - x^*\| \\
& = (M + 2\alpha)\|x^k - x^*\| + \sqrt{2}\|v^k - v^*\|.
\end{align*}
\]
Placing the last inequality into (13) we get the statement.

In the convergence analysis below we will use the following inequality, (21). Let $a, b$ be two arbitrary vectors of the same dimension and $\xi > 0$ be an arbitrary real number. Then
\[
-2a^Tb \leq \frac{1}{\xi}||a||^2 + \xi||b||^2.
\]

The technical lemma below is proved in (21), although it is not stated separately, see the proof of Theorem 2, inequality (93) in (21).

**Lemma 3.4:** Assume that $A[1,3]$ hold and $\{x^k, q^k\}$ be a sequence generated by Algorithm INDO. Then
\[
\begin{align*}
\|v^{k+1} - v^*\|^2 & \leq \frac{\beta\varepsilon^2}{(\beta - 1)\lambda_2}\|d^k\|^2 \\
& + \frac{\phi\varepsilon}{\lambda_2}\|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 + \frac{\beta\phi}{(\phi - 1)\lambda_2}\|e^k\|^2,
\end{align*}
\]
where $\lambda_2$ is the smallest nonzero eigenvalue of $I - \mathcal{W}$, and $\beta, \phi > 1$ are arbitrary constants.

Let us define the sequence of concatenated dual and primal errors as well as the Lyapunov function using the matrix $G$ below,
\[
\begin{pmatrix}
\v & x \\
\mathcal{G} & \begin{pmatrix}\|\|0 & \v G \\
\mathcal{0} & \alpha\v G
\end{pmatrix}
\end{pmatrix}
\]

Analogously define the concatenated $u^*$ and consider the sequence $\|u^k - u^*\|^2 = \|v^k - v^*\|^2 + \alpha\v\|x^k - x^*\|^2$. We will prove that that the sequence $\|u^k - u^*\|^2$ converges to zero linearly and hence the sequence of primal errors $\|x^k - x^*\|$ converges to zero linearly as well.

The following Lemma is proved in the Appendix.

**Lemma 3.5:** Assume that $A[1,3]$ hold and let $\{x^k, q^k\}$ be a sequence generated by Algorithm INDO. Then
\[
\begin{align*}
\|u^{k+1} - u^\ast\|^2 & \leq \frac{2\alpha}{m + M}\|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \\
& - \frac{\alpha\v}{M + M}\|x^{k+1} - x^\ast\|^2 \frac{\beta\v}{m + M}\|x^{k+1} - x^\ast\|^2 + \alpha\v\|x^k - x^\ast\|^2 \\
& - \alpha\v\|\v^k\|^2 + \alpha\v\|e^k\|^2.
\end{align*}
\]

The main convergence result is stated in the following Theorem.

**Theorem 3.1:** Assume that $A[1,3]$ hold and let $\{x^k, q^k\}$ be a sequence generated by Algorithm INDO. Let $\beta, \phi > 1$ be arbitrary constants, $\lambda_2$ the smallest positive eigenvalue of $(I - \mathcal{W})$ and $\zeta \in ((m + M)/(2mM), \v/8M^2))$. Then there exists $\bar{\eta} > 0$ such that for $\eta_k \leq \bar{\eta}$, the sequence of Lyapunov functions $\|u^k - u^\ast\|^2$ satisfies
\[
\|u^{k+1} - u^\ast\|^2 \leq \frac{1 + \delta}{1 + \delta}\|u^k - u^\ast\|^2,
\]
where $\delta \leq \min\{\delta_a, \delta_b\}$ with
\[
\begin{align*}
\delta_a & = \frac{2mM}{(m + M)e} + \frac{1}{\v} \\
\delta_b & = \frac{(\alpha\v - 8M^2\alpha\v)(\phi - 1)(\beta - 1)\lambda_2}{\beta\v^2(\phi - 1) + 8M^2\beta(\beta - 1)\phi^2} \frac{2\alpha\lambda_2}{(m + M)\phi}.
\end{align*}
\]
and $\delta < \delta$.

**Proof.** To prove the statement we have to find $\delta$ such that
\[
\delta\|u^{k+1} - u^\ast\|^2 \leq \|u^k - u^\ast\|^2 - \|u^{k+1} - u^\ast\|^2.
\]

(16)
Using the estimate for \(\|v^{k+1} - v^*\|\) given in Lemma 3.4 and (15) in Lemma 3.5, the inequality (16) holds if
\[
\delta \alpha \varepsilon \|x^{k+1} - x^*\|^2 + \frac{\delta \beta \varepsilon^2}{(\beta - 1)\lambda_2} \|d^k\|^2 \\
+ \frac{\delta \phi \beta}{\lambda_2} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \\
+ \frac{\beta \phi}{\lambda_2} \|e^k\|^2 - \delta \alpha \varepsilon \|x^k - x^*\|^2 \leq (2m + M) \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 - \delta \|v^k - v^*\|^2 \\
+ \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 - \alpha \varepsilon \|e^k\|^2.
\]

By Lemma 3.3 we have
\[\|e^k\|^2 \leq 8M^2 \|d^k\|^2 + 4\eta_k^2 (M + 2\alpha)^2 \|x^k - x^*\|^2 + 8\eta_k^2 \|v^k - v^*\|^2.
\]
Substituting this bound for \(\|e^k\|^2\) and the corresponding lower bound for \(-\|e^k\|^2\) at both sides of (17) we get the inequality
\[
0 \leq \|x^{k+1} - x^*\|^2 \left(\frac{2 \alpha}{m + M} - \frac{\delta \phi \beta}{\lambda_2}\right) \\
+ \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \left(\alpha \varepsilon - \frac{\delta \beta \varepsilon^2}{(\beta - 1)\lambda_2} - 2M^2 \left(\frac{\delta \phi \beta}{\lambda_2} + \alpha \varepsilon\right)\right) \\
+ \|d^k\|^2 \left(\alpha \varepsilon - \frac{\delta \beta \varepsilon^2}{(\beta - 1)\lambda_2} - \frac{\delta \phi \beta}{\lambda_2} + \alpha \varepsilon\right) \\
+ \alpha \delta \left(\tilde{\delta} - 8\eta_k^2 \left(\alpha \varepsilon + \frac{\delta \phi \beta}{(\beta - 1)\lambda_2}\right)\right).
\]

The last inequality holds for \(\delta\) specified in the statement and \(\delta < \delta\) if
\[
\eta_k^2 \leq \tilde{\eta}^2 \\
\leq \min\{\frac{\delta \alpha \varepsilon}{4(M + 2\alpha)^2(\alpha \varepsilon + \frac{\delta \beta \varepsilon^2}{(\beta - 1)\lambda_2})}, \frac{\delta}{8(\alpha \varepsilon + \frac{\delta \beta \varepsilon^2}{(\beta - 1)\lambda_2})}\}
\]
\[\square\]

### IV. Distributed Computation of Inexact Newton Step

In this section we focus on the Step S1 in (10) of the algorithm INDO, for a fixed iteration \(k\). We drop the iteration counter in the Hessian matrix \(H^k\) in (6) and remaining relevant quantities to simplify notation. That is, further on denote \(H^k = H\), i.e.,
\[
H = \nabla f(x^k) + \alpha (I - W) + \varepsilon I,
\]
where \(\varepsilon > 0\) is the proximal parameter and \(g = g^k = \nabla f(x^k) + q^k + \alpha (I - W) x^k\). By assumption A2 the Hessian of the objective function \(f\) is positive definite with \(m \succ \nabla^2 f(x^k)\), the matrix \(\alpha (I - W)\) is positive semidefinite and since \(\varepsilon > 0\) we conclude that there holds
\[
(m + \varepsilon) I \preceq H.
\]
Notice that we can even make \(H\) strictly diagonally dominant by taking \(\varepsilon\) large enough. Furthermore, \(H\) has the same sparsity structure of the network and hence one can easily apply an iterative solver of the fixed point type in S1 of Algorithm INDO.

Let us look more closely at the step calculation in S1. For simplicity of exposition here we concentrate on the Jacobi Overrelaxation (JOR) method although other options for linear solver are possible. For each \(k\) we need to solve the system \(Hd = -g\) approximately i.e. we are looking for \(d^k\) such that (10) holds. Thus we will consider the linear system
\[
Hd = -g,
\]
with \(H\) satisfying (19). With the splitting \(H = D - G\), where \(D\) is the diagonal part of \(H\), the Jacobi Overrelaxation matrix \(T_\gamma\) is defined as
\[
T_\gamma = \gamma D^{-1}G + (1 - \gamma)I
\]
and the JOR iterative method is defined as
\[
d^{k+1} = T_\gamma d^k - \gamma D^{-1}g,
\]
with \(\gamma\) being the relaxation parameter, for arbitrary \(d^0\). For \(\gamma = 1\) we get the Jacobi iterative method with the iterative matrix \(T_1 = D^{-1}G\).

The matrix \(T_\gamma\) has the same sparsity structure as \(W\) and all other matrices we considered so far, including the global Hessian. More precisely we can specify block-rows of \(T_\gamma\) that are used in the JOR method by each node as follows. Notice that, for matrix \(DD\), we have its diagonal blocks \(D_{ii} \in \mathbb{R}^{n \times n}, i = 1, \ldots, N\), given by
\[
[D_{ii}] = \varepsilon + \alpha (1 - w_{ii}) + \left[\nabla^2 f_i(x^k)\right]_{jj}, j = 1, \ldots, n,
\]
where \([\nabla^2 f_i(x^k)]_{jj}\) denotes the \(j\)-th diagonal element of the local Hessian \(\nabla^2 f_i(x^k)\). Similarly, with matrix \(G\), consider its blocks \(G_{ij} \in \mathbb{R}^{n \times n}, i, j = 1, \ldots, N\). Then for \(i \neq j\) we have that \(G_{ij}\) is diagonal with elements \(\alpha w_{ij}\) on the diagonal, and for \(i = j\) we have
\[
G_{ii} = diag(\nabla^2 f_i(x^k)) - \nabla^2 f_i(x^k),
\]
with \(\text{diag}(\nabla^2 f_i(x^k)) \in \mathbb{R}^{n \times n}\) being the diagonal of local Hessian \(\nabla^2 f_i(x^k)\), for all \(i = 1, \ldots, N\). Thus the matrix \(T_\gamma\) has block elements \([T_\gamma]_{ii} = D_{ii}^{-1}G_{ii}, i, j = 1, \ldots, N\). Let \([T_\gamma]_{ii} \in \mathbb{R}^{n \times n}\) be the block row of \(T_\gamma\), \(i = 1, \ldots, N\)
\[
[T_\gamma]_{ii} = [D_{ii}^{-1}G_{ii}, \ldots, D_{ii}^{-1}G_{ii}, \ldots, D_{ii}^{-1}G_{ii}],
\]
Similarly, we can define the partition of the gradient vector \(g\) with the \(th\) component being
\[
g^k = \nabla f_i(x^k) + q^k + \alpha [(1 - w_{ii}) x_i^k - \sum_{j \in \ell_i} w_{ij} x_j^k], i = 1, \ldots, N.
\]

Notice that each node \(i\) can compute \([T_\gamma]_{ii}\) and \(g^k\).

Let us now state the \(k + 1\)-th iteration of INDO algorithm in the node-wise manner.

**Algorithm INDO-nodewise**

**Given**: \(x^k, q^k, \alpha > 0, \varepsilon > 0, \ell_k \geq 1, \ell \in \mathbb{N}\).

**S1** Computing the direction \(d^k\)

**S1.1** Each node computes \([T_\gamma]_{ii}\) and \(g^k\) by (23) - (26) and chooses \(d^k \in \mathbb{R}^n\).
S1.2 Each node computes $d_i^k$ as follows.

For $\ell = 0, \ldots, \ell_k - 1$
- Each node sends $d_i^\ell$ to all its neighbors and receives $d_j^\ell, j \in O_i$.
- Each node computes
  \[d_i^{\ell + 1} = [T_{ij}]d_i^{\ell} - \gamma D^{-1}_{ii}y_i^k\] (27)
  with $d_i^0 = (d_i^1, \ldots, d_i^N)$.
Endfor.
Set $d_i^k = d_i^{\ell_k}$.

S2 Primal update

S2.1 Each node updates the primal variable
  \[x_i^{k+1} = x_i^k + d_i^k,\]
  S2.2 Each node sends $x_i^{k+1}$ to all its neighbours and receives $x_j^{k+1}, j \in O_i$.
S3 Each node computes the dual update
  \[q_i^{k+1} = q_i^k + \alpha [(1 - w_{ii})x_i^{k+1} - \sum_{j \in O_i} w_{ij}x_j^{k+1}],\]
and sets $k = k + 1$.

Notice that Step 1.2 requires only the neighboring elements of $d^\ell$ as $G_{ij} = 0$ if $w_{ij} = 0, i \neq j$, and thus the step is well defined and (27) is equivalent to (22). The algorithm above specifies a choice for Step S1 in the general INDO method in (10), but it is not straightforward to understand the connection. Namely, in Step 1.2 of the node-wise algorithm we state that each node should perform $\ell_k$ JOR iterations, while the main algorithm (see (10)) requires the step $d^k$ such that the inexact forcing condition (10) holds with some $\eta_k > 0$ holds. In the sequel we first analyse the convergence conditions of the JOR method for solving (20) and then we show that one can in fact determine $\ell_k$ such that (10) holds after $\ell_k$ iterations in Step 1.2 of the above algorithm.

The JOR method is convergent for
  \[\gamma \in (0, 2/\sigma(D^{-1}\mathbb{I}))\] (28)
for symmetric positive definite matrices, with $\sigma(D^{-1}\mathbb{I})$ being the spectral radius of $D^{-1}\mathbb{I}$, [11]. In the statement below we estimate the upper bound for the spectral radius of matrix $D^{-1}\mathbb{I}$, using the block-wise Euclidean norm as in [2], [14]. For $T \in \mathbb{R}^{N \times N}$, $T = [T_{ij}], T_{ij} \in \mathbb{R}^{n \times n}$ we define
  \[\|T\|_b = \max_{1 \leq i \leq N} \sum_{j=1}^N \|T_{ij}\|_2.\] (29)
Clearly, $\|\cdot\|_b$ is a norm.

**Proposition 4.1:** Let Assumptions 1-3 hold and $\alpha > 0$. Then the Jacobi Overrelaxation method (20) is convergent for
  \[\gamma \in \left(0, 2\frac{m + \alpha(1 - w_d) + \varepsilon}{M + \varepsilon + \alpha(1 - w_m) + \alpha(1 - w_d)}\right),\]
where $w_d = \max_{1 \leq i \leq N} w_{ii}, w_m = \min_{1 \leq i \leq N} w_{ii}$.

**Proof.** Given the fact that for positive $\varepsilon$ and $\alpha$ we have that $\mathbb{I}$ is positive definite, the convergence interval is determined by (28). On the other hand we have $\sigma(D^{-1}\mathbb{I}) \leq \|D^{-1}\mathbb{I}\|_b$, so we have to estimate $\|D^{-1}\mathbb{I}\|_b$. By definition of $D^{-1}\mathbb{I}$ and the norm we have
  \[\|D^{-1}\mathbb{I}\|_b = \max_{1 \leq i \leq N} \sum_{j=1}^N \|D^{-1}\mathbb{I}_{ij}\|_2\]
  \[= \max_{1 \leq i \leq N} \left\|I - D^{-1}_{ii}G_{ii}\right\|_2 + \sum_{j \neq i} \|D^{-1}_{ii}G_{ij}\|_2\].
For each $i = 1, \ldots, N$ we have that $D_{ii}$ is given by (23). Furthermore, the diagonal elements of local Hessian $\nabla^2 f_i(x_i^k)$ are in the interval $[m, M]$ by $A^k$. Therefore
  \[\|D_{ii}\|_2 \geq m + \alpha(1 - w_{ii}) + \varepsilon \geq m + \alpha(1 - w_d) + \varepsilon.\]
Next, for $i \neq j$, we have \[\|G_{ij}\|_2 = \alpha w_{ij}\] and $\sum_{j \neq i} w_{ij} = 1 - w_{ii} \leq 1 - w_m$ by the properties of $W$ stated in $A^k$. On the other hand, for $i = j$, we have by (24)
  \[\|G_{ii}\|_2 \leq \|\nabla^2 f_i - \text{diag}(\nabla^2 f_i)\|_2.\]
It can be shown that $\|\nabla^2 f_i(x_k^k) - \text{diag}(\nabla^2 f_i(x_k^k))\|_2 \leq M - m$ and thus
  \[\|G_{ii}\|_2 \leq M - m.\]
Combining the bounds for $D_{ii}$ and $G_{ii}$ we get
  \[\|I - D_{ii}^{-1}G_{ii}\|_2 \leq 1 + \|D_{ii}^{-1}G_{ii}\|_2\]
  \[\leq 1 + \frac{M - m}{m + \alpha(1 - w_d) + \varepsilon}\]
  \[= \frac{M + \alpha(1 - w_d) + \varepsilon}{m + \alpha(1 - w_d) + \varepsilon}.\]
Finally, using the bound above and the bounds for $\|D_{ii}^{-1}\|_2$ and $\|G_{ij}\|_2$, we get
  \[\|D^{-1}\mathbb{I}\|_b \leq \frac{\alpha(1 - w_m) + \varepsilon}{m + \alpha(1 - w_d) + \varepsilon}\]
  \[= \frac{M + \varepsilon + \alpha(1 - w_m) + \alpha(1 - w_d)}{m + \alpha(1 - w_d) + \varepsilon},\]
and the statement follows. □

Notice that the bound obtained above does not depend on $k$, and hence we can claim that the method in Step 1.2 of the node-wise algorithm will converge to the true solution $d^*$ for an arbitrary $d^0 \in \mathbb{R}^{nN}$ if we define the JOR matrix with the relaxation parameter $\gamma$ as stated in Proposition 4.1.

The remaining question is the number of inner iterations in Step 1.2 $\ell_k$ that ensures that the forcing condition (10) is satisfied. In the statement below we will prove that such $\ell_k$ exists considering a special choice of $d_i^0$ for simplicity of exposition. Similar result can be proved for other choices of $d_i^0$.

**Proposition 4.2:** Let $A^k$ hold and assume that $\gamma$ is chosen from the interval specified in Proposition 4.1. For given $\eta_k > 0$ denote by $d^k$ a step obtained through Step 1.2 of the node-wise algorithm INDO with $d_i^0 = 0, i = 1, \ldots, N$ and $\ell_k$ inner iterations [27]. Then $\|Hd^k + g\| \leq \eta_k \|g\|$ after at most
  \[\ell_k \geq \left\lfloor \frac{\ln(\eta_k/\varepsilon)}{\ln(\sigma(T_{ij}))} \right\rfloor,\]
where iterations, where $\lceil a \rceil$ denotes the smallest integer greater than or equal to $a$ and
\[
c := \frac{M + 2 + \varepsilon}{m + \varepsilon} \sqrt{\frac{\varepsilon + M + \alpha(1 - w_m)}{\varepsilon + m + \alpha(1 - w_d)}}.
\]

**Proof.** Denote by $d^*$ the exact solution of $Hd = -g$ and consider the sequence \{d$^\ell$\} generated at Step 1.2 with $d^0 = 0$. As $d^{\ell + 1} = \mathcal{T}_\gamma d^\ell - \gamma D^{-1}g$, $\ell = 0, 1, \ldots$ and $d^* = \mathcal{T}_\gamma d^* - \gamma D^{-1}g$ we have
\[
d^\ell - d^* = \mathcal{T}_\gamma (d^\ell - d^*).
\]
Notice that $\mathcal{T}_\gamma$ is not symmetric, but it has the same set of eigenvalues as the following symmetric matrix
\[
\mathcal{T}_\gamma' = D^{-1/2} (\gamma G + (1 - \gamma) D) D^{-1/2} := D^{-1/2} C_\gamma D^{-1/2},
\]
where $\mathcal{T}_\gamma = D^{-1} C_\gamma$ and both $D$ and $C_\gamma$ are symmetric (see Remark 4.2 of [1] for instance). Therefore, we conclude
\[
\sigma(\mathcal{T}_\gamma) = \sigma(\mathcal{T}_\gamma') = \|\mathcal{T}_\gamma\|
\]
and according to the choice of $\gamma$ and Proposition 4.1 we have $\sigma(\mathcal{T}_\gamma) < 1$. Moreover, it can be shown that
\[
\mathcal{T}_\gamma^\ell = D^{-1/2} (\mathcal{T}_\gamma')^\ell D^{1/2}
\]
and thus we obtain
\[
\|d^\ell - d^*\| \leq \|D^{-1/2}\|\|\mathcal{T}_\gamma^\ell\|\|\mathcal{T}_\gamma^{1/2}\|\|d^0 - d^*\| = 0
\]
(30)
\[
\|I - W\| \leq 2. Finally, we have
\[
\|Hd^\ell + g\| \leq \|Hd^\ell - Hd^*\| \leq \|H\|\|d^\ell - d^*\| \leq cD \frac{M + 2 + \varepsilon}{m + \varepsilon} \sigma(\mathcal{T}_\gamma)\|g\|
\]
and we get
\[
\|Hd^{\ell_k} + g\| \leq \eta_k \|g\|
\]
for $\ell_k$ given in the statement. □

The above statement implies that we can run INDO with the fixed number of inner iterations for all $k$, by taking $\ell_k = \ell(\bar{\eta})$ (see Theorem 3.1), to avoid the overhead needed to define suitable iteration-leving quantities $\eta_k$ and $\ell_k$. In fact, the numerical results we present in the next Section show that INDO behaves remarkably well if we take a very modest number of inner iterations at each outer iteration; namely, even $\ell_k = 1$ with the so-called warm start gives satisfactory results. In practice, this also alleviates the need to calculate the complicated expression for $\ell(\bar{\eta})$ as per Proposition 4.2 that depends on certain global quantities such as $\sigma(\mathcal{T}_\gamma)$.

Another possibility would be to define the forcing condition in the block infinity norm, i.e.,
\[
\|Hd^k + g\|_\infty \leq \eta_k \|g\|_\infty.
\]
Then each node can compute the value of local residual $\|r^\ell_i\|_\infty$ at each $\ell$; subsequently, at each $\ell$, the nodes can run an iterative primitive for computing maximum (e.g., [42]) of their local residuals until the overall forcing condition in the block infinity norm is satisfied.

A couple of words on the forcing sequence \{nk\} are due here. Given that we are in the framework of the proximal method of multipliers, the most we can achieve is the overall linear convergence as the linear convergence is achieved even if the Newtonian linear systems are solved exactly i.e. for $\eta_k = 0$, see [21]. On the other hand we know from the classical optimization theory that $\eta_k$ greatly influences the local convergence rate of Inexact Newton methods. In fact for $\eta_k \to 0$ we have superlinear local convergence while $\eta_k = \mathcal{O}(\|g^k\|)$ recovers the quadratic convergence of the Newton method. In fact, one can show that the bound for $\bar{\eta}$ can be made arbitrarily close to 1 and still have a convergent method if a weighted norm is used, see [37]. But given the overall linear convergence at most in the assumed framework, it seems most efficient to work with a constant $\eta_k$ during the whole process as such procedure maintains the linear rate of convergence while minimizing the effort needed for tuning $\eta_k$ and $\ell_k$. The key property – avoiding oversolving in the approximate Newton step calculation – is achieved with a constant $\eta_k$ (and hence constant $\ell_k$). The comparison with ESOM, both in terms of (inner iterations) convergence factor and numerically, presented in the next section strongly supports the approach we advocate here – approximate second order direction with the error proportional to the gradient value while incurring a small computational cost.

**V. PERFORMANCE ANALYSIS: NUMERICAL RESULTS AND FURTHER ANALYTICAL INSIGHTS**

In this Section we analyze computational and communication costs of INDO and test its performance on a set of standard test examples. INDO method is a second-order method within Proximal Method of Multipliers that differs from the ESOM method, [21] in computing the primal step. So, we will first compare the theoretical convergence rates of these two methods. These findings are presented in Subsection 5.1, while the numerical examples and cost comparison are presented in Subsection 5.2.

**A. Inner iterations’ convergence rate of INDO and ESOM**

We provide here a more detailed comparison of INDO with the ESOM method proposed in [21] with respect to inner
iterations’ convergence. The two algorithms have identical dual variable updates, given by (8). Hence we focus on the primal variable updates. At each outer iteration \( k \), both INDO and ESOM approximately solve the system of linear equations (20). While INDO solves (20) via (21)–(22), the splitting matrix \( \sigma \) linearly with the convergence factor determined by the spectral radius \( \| \mathcal{H} \|_{\text{op}} \). Here, for ESOM, we have that \( \sigma \) is in general difficult to explicitly evaluate. Assuming for simplicity that all elements on the diagonal of \( \mathcal{E} \) are taken as \( 1 + \max \{ \deg(i), \deg(j) \} \), where \( \deg(i) \) stands for the degree of node \( i \), for directly connected nodes \( i \) and \( j \), and the diagonal weights are \( w_{i,i} = 1 - \sum_{j \neq i} w_{i,j} \). The matrix \( \mathcal{W} \) generated in this way satisfies (40).

The recommended tuning parameter choice \( \alpha = M = \varepsilon \) still requires a beforehand global knowledge of system parameters, specifically the constant \( M \). Assuming that each node knows \( M \)-a Lipschitz constant of its own function \( f_i \)'s gradient, \( M = \max_{i=1,...,N} M_i \) can be obtained by running beforehand a distributed algorithm for computing maximum of scalar values held by the nodes, e.g., [42]. This can be done with a low overhead, wherein the number of required inter-neighbor (scalar-transmission) communication rounds is on the order of the network diameter. INDO as implemented in Section V also requires the beforehand knowledge of \( m \) and \( w_i \); these quantities can be computed beforehand analogously to \( M \).

B. Numerical results

We compare INDO with ESOM proposed in (21) on quadratic cost functions (simulated data) and on logistic regression problems (real data). The network we consider has \( N = 30 \) nodes and is formed as follows, [16]. The points are sampled randomly and uniformly from \([0,1]\times[0,1] \). The edges between points exist if their Euclidean distance is smaller than \( r = \sqrt{\log(N)/N} \). The resulting graph instance considered is connected. The weight coefficients in the communication matrix \( \mathcal{W} \) are taken as \( w_{i,j} = 1/(1 + \max \{ \deg(i), \deg(j) \}) \), where \( \deg(i) \) stands for the degree of node \( i \), for directly connected nodes \( i \) and \( j \), and the diagonal weights are \( w_{i,i} = 1 - \sum_{j \neq i} w_{i,j} \). The matrix \( \mathcal{W} \) generated in this way satisfies (40).

Let us now describe the test examples. Quadratic local cost functions are of the form

\[
    f_i(y) = \frac{1}{2}(y - b_i)^T B_{ii} (y - b_i)
\]

and the data is simulated as in [15], i.e., vectors \( b_i \) are drawn from the Uniform distribution on \([1,31] \), independently from each other. Matrices \( B_{ii} \) are of the form \( B_{ii} = P_i S_i P_i \), where \( S_i \) are diagonal matrices with Uniform distribution on \([1,101] \) and \( P_i \) are matrices of orthonormal eigenvectors of \( \frac{1}{2}(C_i + C_i^T) \) where \( C_i \) have components drawn independently from...
the standard Normal distribution. Given that in this case we can compute the exact minimizer \( y^* \), the error is measured as
\[
E(x^k) := \frac{1}{N} \sum_{i=1}^{N} \frac{\|s_i^k - y^*\|}{\|y^*\|},
\]
(41)

Both Algorithms require the Hessian lower and upper bound \( M, m \) and we calculate them as \( M = \max_i M_i \), where \( M_i \) is the largest eigenvalue of \( B_{ii} \) and \( m = \min_i m_i \), where \( m_i > 0 \) is the smallest eigenvalue of \( B_{ii} \). The dimension of the problem is set to \( n = 100 \).

The two methods are also compared on binary classification problems and the following data sets: Mushrooms [32] (\( n = 112 \), total sample size \( T = 8124 \), LSVT Voice Rehabilitation [29] (\( n = 309 \), total sample size \( T = 126 \)) and Parkinson’s Disease Classification [25] (\( n = 754 \), total sample size \( T = 756 \)). For each of the problems, the data is divided across 30 nodes of the graph described above. The logistic regression with the quadratic regularization is used and thus the local objective functions are of the form
\[
f_i(y) = \frac{1}{|J_i|} \sum_{j \in J_i} \log(1 + e^{-\zeta_j p_j^T y}) + \frac{m}{2} \|y\|^2,
\]
where \( J_i \) collects the indices of the data points assigned to node \( i \), \( p_j \in \mathbb{R}^n \) is the corresponding vector of attributes and \( \zeta_j \in \{-1, 1\} \) represents the label. The data is scaled in a such way that \( M = 1 + m \) with \( m = 10^{-4} \). Since the solution is unknown in general, the error is measured as the average value of the original objective function across the nodes’ estimates
\[
V(x^k) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} f_j(x^k_i).
\]
(42)

We fix the free parameters of INDO method to \( \alpha = \varepsilon = M \). The reasoning behind this choice is explained in Section 5.1. We use this choice in all the tested examples, although it may not be the optimal. As discussed in the previous section, step S1 of INDO method can be implemented in different ways depending on the network characteristics, dimension of the problem, etc. We use a practical version of INDO in the tests by taking a fixed number \( \ell \) of inner iterations of JOR method (e.g., \( \ell = 1 \)) with
\[
\gamma = 2(1 + \varepsilon + \alpha(1 - w_{ii})/(M + 2\alpha + \varepsilon))
\]
denoted by INDO-\( \ell \) in the sequel. Initial \( d^0 \) is obtained by solving
\[
\|d^0 + g^0\|_\infty \leq \|g^0\|_\infty.
\]
In all the subsequent iterations we use the so called warm start, i.e., we set \( d^0 = d^{k-1} \) at Step 1.2 in the node-wise INDO representation. Given that we implement the fixed number of inner iterations \( \ell \) the forcing term \( \eta_k \) is not explicitly imposed.

Denote by ESOM-\( \ell \)-\( \alpha, \varepsilon \) the ESOM method with \( \ell \) inner iterations and the corresponding free parameters \( \alpha \) and \( \varepsilon \). Notice that ESOM-\( \ell \) requires the same amount of communications as INDO-\( \ell \). Therefore, we plot iterations versus the corresponding error measures of both methods to compare the performance with respect to the communication costs.

Regarding the computational cost, the main advantage of INDO method with respect to ESOM is the following: ESOM requires inverting full \( n \times n \) matrices at each node, while INDO only inverts the diagonal ones. The difference is more evident in non-quadrican cases where the corresponding matrices need to be inverted in every outer iteration of ESOM method. This is even more significant in problems with relatively large \( n \).

Let us now estimate the computational costs of the two algorithms more precisely. Both ESOM and INDO are second order methods, so the cost of calculating the derivatives are the same. Moreover, the outer iterations updates are identical. The main difference lies in performing the inner iterations.

In order to estimate the costs, let us observe the formula for INDO [27], i.e.,
\[
d_{i+1}^{\ell} = \gamma D_{ii}^{-1} \left( G_{ii} d_{i}^{\ell} + \alpha \sum_{j \in O_i} w_{ij} d_{j}^{\ell} - g_{i}^{k} \right) + (1 - \gamma) d_{i}^{\ell},
\]
and ESOM
\[
d_{i+1}^{\ell} = \left[ D_{E}^{-1} \right]_{ii} \left( \alpha(1 - w_{ii}) d_{i}^{\ell} + \alpha \sum_{j \in O_i} w_{ij} d_{j}^{\ell} - g_{i}^{k} \right),
\]
(43)

Both methods perform one consensus step (\( \sum_{j \in O_i} w_{ij} d_{j}^{\ell} \)) per inner iteration per node. Both methods perform one matrix-vector product per inner iteration per node as well: INDO calculates \( G_{ii} d_{i}^{\ell} \) and ESOM calculates product of \( D_{E}^{-1} \) with the corresponding vector. The difference lies in the following. INDO inverts the diagonal matrix \( D_{ii} \) and multiplies it with the corresponding vector in each inner iteration, i.e., at each inner iteration we have component-wise division of two vectors. Thus, the cost of this operations can be estimated to \( nl \) scalar products in \( \mathbb{R}^n \) (SPs) per outer iteration per node for INDO-\( \ell \) algorithm. On the other hand, ESOM calculates the inverse of possibly dense symmetric positive definite matrix \( D_{E}^{-1} \) which can be estimated to \( n^2/2 \) SPs. For logistic regression problems, we estimate the common computational costs (see [17] for more details) of both algorithms to \( |J_i|(2 + n/2) + N + n\ell + N\ell/n \) SPs per node per (outer) iteration: \( |J_i|(2 + n/2) \) SPs is the cost of calculating the gradient and the Hessian of local cost function, \( N = nN/n \) SPs comes from consensus step in calculating \( g^k \), matrix-vector products take \( nl \) SPs and the consensus with respect to \( d_{j} \) vectors takes \( tN/n \) SPs. Thus, the overall computational cost per node per iteration in logistic regression case is estimated to \( |J_i|(2 + n/2) + N + n\ell + N\ell/n + n\ell \) for INDO-\( \ell \) and to \( |J_i|(2 + n/2) + N + n\ell + N\ell/n + n^2/6 \) for ESOM-\( \ell \). Clearly, these costs differ by the order \( O(n) \). For the quadratic costs we do not have costs of calculating the local Hessian in every iteration while the cost of calculating the local gradient is \( n \) SPs. Other common costs are the same as in logistic regression case. INDO still needs \( nl \) SPs in every iteration, while ESOM only inverts the Hessian in the initial phase with the cost of \( n^2/6 \) SPs.

Figure 1 presents results on simulated quadratic costs for different number of inner iterations, i.e. with respect to the communication costs. We test different combinations of \( \alpha \) and \( \varepsilon \) for ESOM method, while INDO is tested with fixed parameters as explained above. The best tested ESOM algorithm is ESOM-\( \ell \)-M-M. Other variants of ESOM start better, but they fail to converge to the same solution vicinity of the exact solution as the INDO method.
Fig. 1. INDO (solid line) versus ESOM methods (dotted lines): error
\( \text{Error}(\ell) \) with respect to iterations/communication cost (the two top figures) and computational cost (the two bottom figures) for the number of inner iterations \( \ell = 1 \) (first and third figure from top) and \( \ell = 2 \) (second and fourth figure from top); Simulated quadratic costs with \( n = 100 \) and \( N = 30 \).

Fig. 2. INDO (solid line) versus ESOM methods (dotted lines): error
\( \text{Error}(\ell) \) with respect to iterations/communication cost (the two top figures) and computational cost (the two bottom figures) for the number of inner iterations \( \ell = 1 \) (first and third figure from top) and \( \ell = 2 \) (second and fourth figure from top); Mushrooms dataset.
Fig. 3. INDO (solid line) versus ESOM methods (dotted lines): error (42) with respect to iterations/communication cost (the two top figures) and computational cost (the two bottom figures) for the number of inner iterations $\ell = 1$ (first and third figure from top) and $\ell = 2$ (second and fourth figure from top). LSVT Voice Rehabilitation dataset.

Fig. 4. INDO (solid line) versus ESOM methods (dotted lines): error (42) with respect to iterations/communication cost (the two top figures) and computational cost (the two bottom figures) for the number of inner iterations $\ell = 1$ (first and third figure from top) and $\ell = 2$ (second and fourth figure from top). Parkinson’s Disease Classification dataset.
Figures 2-4 present analogous tests on logistic regression problems and the data sets Mushrooms, LSVT Voice Rehabilitation and Parkinson’s Disease Classification. As already mentioned, for these problems $M = 1.0001$ so we omit the case $\alpha = 0.01, \varepsilon = 1$ for ESOM method in these tests. All the parameters for INDO algorithm are the same as in quadratic case. The results show that INDO algorithm is better or comparable with the ESOM algorithm with respect to iterations and communication costs. The difference seems to be bigger in the case of $\ell = 1$ than for the larger number of inner iterations $\ell = 2$. On the other hand, INDO outperforms ESOM method with respect to computational cost. Notice that for the Mushrooms dataset, $n$ is relatively small with respect to $|J_i|$ and thus the common cost is the dominant cost, so the computational cost plots are very similar (in the sense of relative comparison INDO versus ESOM) to the ones that represent iterations and communication costs. For the remaining logistic regression problems, $n$ is comparable to the total sample size $T$ and thus the INDO’s advantage in computational cost savings is more evident.

VI. CONCLUSIONS

In this paper, we proposed INDO, an exact distributed second order method for strongly convex distributed consensus optimization. INDO is design based on the framework of Proximal Method of Multipliers (PMM) and maintains a primal and a dual variable per iteration. Unlike exact distributed second order methods like ESOM [21] and DQM [22], INDO does not involve explicit Hessian inverse calculation when calculating the primal variable update. Instead, INDO calculates the primal variable update via a few inner iterations of a fixed point method (e.g., Jacobi overrelaxation) to approximately solve the system of linear equations that underlie the Newton direction evaluation; this update process alleviates the need for Newton inverse calculation. We prove that INDO achieves a global linear convergence to the exact solution of the problem of interest. Then, we provide analysis that reveals how the degree of inexactness in solving the Newton direction systems of linear equations affects the overall convergence rate. Furthermore, we provide intuitive bounds on the convergence factor of the involved linear system solver; these bounds shed light on the role of different method’s parameters and provide guidelines on how these parameters should be set. Numerical experiments on several real data sets demonstrates that INDO achieves a comparable speed iteration-wise and communication cost-wise as ESOM, while at the same time reducing computational cost by at least an order of magnitude when the dimension of the optimization variable is on the order of couple of hundreds or larger.

REFERENCES

[1] Bai, Z., Silverstein, J. W., Spectral Analysis of Large Dimensional Random Matrices, Springer, 2010.
[2] Bajović, D., Jakovetić, D., Krejić, N., Krklec Jerinkić, N., Newton-like Method with Diagonal Correction for Distributed Optimization, SIAM Journal on Optimization, 27, 2 (2017), 1171-1203.
[3] Baingana, B., Giannakis, G., B., Joint Community and Anomaly Tracking in Dynamic Networks, IEEE Transactions on Signal Processing, 64(8), (2016), pp. 2013-2025.
[4] Bellavia, S., Krejić, N., Krklec Jerinkić, N., Subsampled Inexact Newton Methods for minimizing large sums of convex functions,IMA J. Numer. Anal. 40,4 (2020), 2309-2341.
[5] Berahas, A. S., Bollapragada, R., Keskar, N. S., Wei, E., Balancing Communication and Computation in Distributed Optimization, IEEE Transactions on Automatic Control, 64(8), (2019), pp. 3141-3155.
[6] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., Distributed optimization and statistical learning via the alternating direction method of multipliers, Foundations and Trends in Machine Learning, 3(1), (2011) pp. 1-122.
[7] Cattivelli, F., Sayed, A. H., Diffusion LMS strategies for distributed estimation,IEEE Transactions on Signal Processing, 58(3), (2010) pp. 1035–1048.
[8] Chen, M., Mokhtari, A., Ribeiro, A., Decentralized Quasi-Newton Methods, IEEE Trans. Signal Process, vol. 65, no. 10, pp. 2613–2628, 2017.
[9] Dembo, R.S.,., Eisenstadt, S.C., Steihaug, T., Inexact Newton Methods, SIAM Journal on Numerical Analysis, 19,2, (1982), 400-408.
[10] Eisen, M., Mokhtari, A., Ribeiro, A., A Primal-Dual Quasi-Newton Method for Exact Consensus Optimization, IEEE Transactions on Signal Processing, Vol. 67, No. 23, Dec. 2019., pp. 5983–5997.
[11] Greenbaum, A., Iterative Methods for Solving Linear Systems, SIAM, 1997.
[12] Jakovetić, D., A Unification and Generalization of Exact Distributed First Order Methods, IEEE Transactions on Signal and Information Processing over Networks, 5(1), (2019), pp. 31-46.
[13] Jakovetić, D., Bajović, D., Xavier, J., Moura, J.M.F., Primal-Dual Methods for Large-Scale and Distributed Convex Optimization and Data Analytics, Proceedings of the IEEE, 108,11 (2020), 1923-1938.
[14] Jakovetić, D., Krejić, N., Krklec Jerinkić, N., Malaspina, G., Micheletti, A., Distributed Fixed Point Method for Solving Systems of Linear Algebraic Equations, Automatica (2021), Vol. 134.
[15] Jakovetić, D., Xavier, J., Moura, J. M. F., Fast distributed gradient methods, IEEE Transactions on Automatic Control, 59(5), (2014) pp. 1131–1146.
[16] Jakovetić, D., Krejić, N., Krklec Jerinkić, N., Exact spectral-like gradient method for distributed optimization, Computational Optimization and Applications, 74, (2019), pp. 703–728.
[17] Jakovetić, D., Krejić, N., Krklec Jerinkić, N., EFIX: Exact Fixed Point Methods for Distributed Optimization, arxiv preprint, arXiv:2012.05466v1 (2020).
[18] Lee, J. M., Song, J., Jung, S., Lee, J., A rate adaptive convolutional coding method for multicarrier DS/CDMA systems, MILCOM 2000 Proceedings 21st Century Military Communications. Architectures and Technologies for Information Superiority (Cat. No.00CH37155), Los Angeles, CA, (2000), pp. 932-936.
[19] Mansoori, F., Wei, E., A Fast Distributed Asynchronous Newton-Based Optimization Algorithm, IEEE Transactions on Automatic Control, Vol. 65, No. 7, July 2020, pp. 2769–2784.
[20] Mokhtari, A., Ling, Q., Ribeiro, A., Network Newton Distributed Optimization Methods, IEEE Transactions on Signal Processing, 65,1 (2017), 146-161.
[21] Mokhtari, A., Shi, W., Ling, Q., Ribeiro, A., A Decentralized Second Order Method with Exact Linear Convergence Rate for Consensus Optimization, IEEE Transactions on Signal and Information Processing over Networks, 2(4), (2016), pp. 507-522.
[22] Mokhtari, A., Shi, W., Ling, Q., Ribeiro, A., DQM: Decentralized quadratically approximated alternating direction method of multipliers, IEEE Transactions on Signal Processing, vol. 64, no. 19, pp. 5158-5173, Oct. 2016.
[23] Mota, J., Xavier, J., Aguiar, P., Pischel, M., Distributed optimization with local domains: Applications in MPC and network flows, IEEE Transactions on Automatic Control, 60(7), (2015), pp. 1-122.
[24] Qu, G., Li, N., Harnessing smoothness to accelerate distributed optimization,IEEE Transactions on Automatic Control, 64(8), (2019), pp. 3141-3155.
[25] Sakar, C., Serbes, Gorkem, Gunduz, Aysegul, Nizam, Hatice, Sakar, Betul. (2018). Parkinson’s Disease Classification. UCI Machine Learning Repository.
[26] Scutari, G., Sun, Y., Parallel and Distributed Successive Convex Approximation Methods for Big-Data Optimization, Multi-agent Optimization, Lecture Notes in Mathematics, pp 141-308, Springer, 2018.
[27] Shi, W., Ling, Q., Wu, G., Yin, W., EXTRA: an Exact First-Order Algorithm for Decentralized Consensus Optimization, SIAM Journal on Optimization, 2(25), (2015), pp. 944-966.
Now, Lemma 3.1 implies $\alpha(x^{k+1} - x^*)^T(I - W)^{1/2} = (v^{k+1} - v^k)^T$, and therefore
\[
\frac{2\alpha m M}{m + M} \|x^{k+1} - x^*\|^2 + \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \\
\leq -2\alpha (x^{k+1} - x^*)^T (I - W)^{1/2} (v^{k+1} - v^*) - 2\alpha \varepsilon (x^{k+1} - x^*)^T e^k.
\]

Notice further that
\[
2(v^{k+1} - v^k)^T (v^{k+1} - v^*) = \|v^{k+1} - v^k\|^2 + \|v^{k+1} - v^*\|^2
\]
and
\[
2(x^{k+1} - x^*)^T d^k = \|d^k\|^2 + \|x^{k+1} - x^*\|^2 - \|x^k - x^*\|^2.
\]

Going back to (47) we get
\[
\frac{2\alpha m M}{m + M} \|x^{k+1} - x^*\|^2 + \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \\
\leq -\|v^{k+1} - v^k\|^2 - \|v^{k+1} - v^*\|^2 + \|v^k - v^*\|^2
\]
\[
- \alpha \varepsilon \|d^k\|^2 - \alpha \varepsilon \|x^{k+1} - x^*\|^2
\]
\[
+ \alpha \varepsilon \|x^k - x^*\|^2 - 2\alpha (x^{k+1} - x^*)^T e^k.
\]

Lemma 3.1 implies that $\|v^{k+1} - v^k\|^2 = \|x^{k+1} - x^*\|^2 \alpha^2 (I - W)$ and by definition of Lyapunov function we have $\|u^{k+1} - u^*\|^2 = \|v^k - v^*\|^2 - \|v^{k+1} - v^*\|^2 + \alpha \varepsilon \|x^k - x^*\|^2 - \alpha \varepsilon \|x^{k+1} - x^*\|^2$. Thus, inequality (50) reduces to
\[
\frac{2\alpha m M}{m + M} \|x^{k+1} - x^*\|^2
\]
\[
+ \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2 \\
\leq \|u^{k+1} - u^*\|^2 - \|u^k - u^*\|^2 - 2\alpha \varepsilon \|x^{k+1} - x^*\|^2 + 2\alpha \varepsilon \|x^k - x^*\|^2
\]
\[
- 2\alpha (x^{k+1} - x^*)^T e^k.
\]

Regrouping the terms in the above inequality yields
\[
\|u^{k+1} - u^*\|^2 - \|u^k - u^*\|^2 \\
\leq - \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2
\]
\[
- \|v^{k+1} - v^*\|^2 - \|v^{k+1} - v^*\|^2 - \|v^k - v^*\|^2
\]
\[
- 2\alpha \varepsilon \|d^k\|^2 + 2\alpha \varepsilon \|x^{k+1} - x^*\|^2 - \alpha \varepsilon \|x^k - x^*\|^2 - \alpha \varepsilon \|x^{k+1} - x^*\|^2.
\]

Using the bound $2(x^{k+1} - x^*)^T e^k \geq -1/\zeta \|x^{k+1} - x^*\|^2 - \zeta \|e^k\|^2$, which holds for any $\zeta > 0$, we get
\[
\|u^{k+1} - u^*\|^2 - \|u^k - u^*\|^2 \\
\leq - \frac{2\alpha}{m + M} \|\nabla f(x^{k+1}) - \nabla f(x^*)\|^2
\]
\[
- \|v^{k+1} - v^*\|^2 - \|v^{k+1} - v^*\|^2
\]
\[
- \|v^k - v^*\|^2 + 2\alpha \varepsilon \|x^{k+1} - x^*\|^2 - \alpha \varepsilon \|x^k - x^*\|^2 - \alpha \varepsilon \|x^{k+1} - x^*\|^2.
\]