Monitoring Least Squares Models of Distributed Streams

Moshe Gabel
Technion – Israel Institute of Technology
Haifa 32000 Israel
mgabel@cs.technion.ac.il

Daniel Keren
Haifa University
Haifa 31905 Israel
dkeren@cs.haifa.ac.il

Assaf Schuster
Technion – Israel Institute of Technology
Haifa 32000 Israel
assaf@cs.technion.ac.il

ABSTRACT

Least squares regression is widely used to understand and predict data behavior in many fields. As data evolves, regression models must be recomputed, and indeed much work has focused on quick, efficient and accurate computation of linear regression models. In distributed streaming settings, however, periodically recomputing the global model is wasteful: communicating new observations or model updates is required even when the model is, in practice, unchanged. This is prohibitive in many settings, such as in wireless sensor networks, or when the number of nodes is very large. The alternative, monitoring prediction accuracy, is not always sufficient: in some settings, for example, we are interested in the model's coefficients, rather than its predictions.

We propose the first monitoring algorithm for multivariate regression models of distributed data streams that guarantees a bounded model error. It maintains an accurate estimate using a fraction of the communication by recomputing only when the precomputed model is sufficiently far from the (hypothetical) current global model. When the global model is stable, no communication is needed.

Experiments on real and synthetic datasets show that our approach reduces communication by up to two orders of magnitude while providing an accurate estimate of the current global model in all nodes.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*least squares methods*; G.3 [**Probability and Statistics**]: correlation and regression analysis; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*

General Terms

Algorithms

Keywords

Regression; distributed streams; least squares; data mining

KDD '15, August 11–14, 2015, Sydney, NSW, Australia.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2783258.2783349 .

1. INTRODUCTION

Least squares regression is commonly used for prediction of new values from past values (i.e., forecasting), for analysis of existing phenomena through discovered coefficients (e.g., in econometrics [9] and social studies [30]), and as building blocks in other algorithms (e.g., in sparse coding [37, 1]).

Data behavior evolves, however, and changes can render a previously-computed model invalid. In such settings regression models must be updated to incorporate new observations, or to be periodically recomputed. This problem is exacerbated in distributed settings: when observations are distributed over many nodes, we are also faced with the additional cost of communicating updates.

The question then becomes not just *how* to efficiently compute the model, but *when.* Recomputing the model after each new observation seems is wasteful, as models tend to change slowly. Recomputing periodically still involves needless work if the model changes infrequently, yet may introduce unacceptable errors between scheduled updates.

Hence, there are two complementary approaches to distributed linear regression. The first is efficient distributed *computation* of the model. Indeed, much work has been devoted to this approach [23, 21, 33].

We focus on the second approach – *monitoring* the quality of a given model, and recomputing it only as needed (using any of the computational approaches). The monitoring approach looks at incoming data and triggers an alert if the previously-learned model is too dissimilar to the *hypothetical* global model that would have been built given the current data. This problem is difficult in distributed settings, since the existing model and the (hypothetical) current model are both *global* models – composed from the union of data at all nodes. Thus a distributed monitoring algorithm must deal with communication efficiency, in addition to the problem of monitoring a model without actually relearning it.

The monitoring approach has received little attention, possibly because the least squares solutions involve matrix inversion, which is difficult to analyze (Section 2.1). The few existing techniques monitor the model's prediction error or R^2 fit [4, 3], or a univariate model where the problem reduces to monitoring a ratio [8]. This is not always sufficient: in some settings we are interested in the model's coefficients (e.g., analysis), and in others (e.g., interpolation) we lack the ground truth to measure prediction error. Moreover, monitoring model error is a more general approach: prediction error and fit can be inferred from model error but not vice versa.

Our Contribution

We describe DILSQ: a novel communication-efficient monitoring algorithm for multivariate least squares models of distributed, dynamic data streams. To our knowledge, this is the first algorithm that monitors the multivariate regression model itself, rather than its prediction or fit. Given a previously-computed *global* model, we derive *local* constraints on the local data at each node. A node only communicates if its constraint is broken. These constraints guarantee that if no node communicates, the global *hypothetical* model is sufficiently close to the precomputed model.

DILSQ easily generalizes to more complex least squares variants such as GLS and RLS [9], and is independent of how the model was computed. Evaluation on two real datasets shows it reduces communication by up to two orders of magnitude. Complete elimination of all communication is also possible when the model is fixed.

2. RELATED WORK

Due to the ubiquity of linear regression, a great deal of research was dedicated to solving for the regression model not only in a centralized setting, but over distributed systems as well; for a comprehensive survey, see [33]. Typically, the distributed nodes compose a graph, each holding a portion of the data, and the goal is to solve for the regression model of the aggregated data. It is well-known that the accurate solution involves calculating a matrix-vector pair from the data (denote it A, c), and then calculating $A^{-1}c$. Since the global matrix-vector pair can be expressed as the sum of local pairs at the nodes, a path is defined over the graph, and the global pair is obtained by traversing this path; a *Hamiltonian path* is desirable, in order to reduce the time required to traverse the graph [20]. Spanning trees have also been applied to this end [28]. Eventually, the local estimates at the nodes converge, via message passing with neighbors, to a global consensus [24]. In [38] it was suggested that diffusion strategies outperform consensus-seeking methods.

Variants include taking advantage of the global matrix's sparseness in order to reduce traffic [7], and gradient-based methods run either sequentially or with some degree of parallelism [23, 21, 33, 40]. Such techniques were also applied in online distributed learning, where the sought classifier can sometimes be expressed as the solution of a linear regression problem [41].

While efficient solutions were developed for *computing* the linear regression model over distributed nodes, there are, to the best of our knowledge, only very few papers dealing with *monitoring* it – that is, imposing *local* conditions which imply that the *global* solution did not change by more than a pre-defined amount since the last time it was computed (Section 4). In [36], a heuristic is applied, and the nodes do not broadcast if the newly arriving data conforms with the current model up to some tolerance. In [4] distributed monitoring was applied to monitor the prediction error (Section 3.1) and quadratic fit error R^2 [3], but not the error in the model itself.

In [8], a one-dimensional regression problem is addressed – monitoring the ratio of two aggregated variables. Here we address the general, high-dimensional problem.

2.1 Distributed Monitoring

The last decade witnessed a sharp increase in work on imposing local conditions for monitoring the value of a function defined over distributed nodes. While the general problem is NP-complete [15], considerable progress has been made for real-life problems. Most work dealt with the simpler cases of linear functions [14, 13], as well as monotonic functions [25]. Some papers addressed non-linear problems, e.g., monitoring the value of a single-variable polynomial [34], and analysis of eigenvalue perturbation [10]. [11] describes a gossip-based protocol for monitoring several aggregates (some non-linear), which eventually converges to the monitored value, but cannot guarantee user-specified error bounds. In [35] a geometric approach for monitoring arbitrary functions over distributed streams was proposed, and later extended and generalized [16, 18]. However, nearly all work on geometric monitoring addressed functions which are either polynomials (typically quadratic), or defined by compositions of polynomial with simple functions such as medians and quotients. To the best of our knowledge, the problem addressed in this paper – monitoring the linear regression *model* (as opposed to its *fit error*) – was never addressed over a distributed setting. Note that the monitored function contains the highly complicated operation of matrix inversion, which is not linear or convex, and which, when written explicitly, becomes intractable even for relatively low dimensions (e.g., the analytic expression for the inverse of a 20×20 matrix involves polynomials with 20! monomials). Therefore, a straightforward application of previous work on geometric monitoring is impossible.

3. PROBLEM DEFINITION

Let $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ be a set of n observation pairs of $m < n$ independent variables and one dependent variable, where x_i are column vectors in $\mathbb{R}^m$, and y_i are the corresponding response scalars. We seek a linear transformation $\beta \in \mathbb{R}^m$, $\beta = (\beta_1, \ldots, \beta_m)^T$, that minimizes the sum of squared errors between y_i to the mapping of x_i. In other words, we seek a model β that minimizes $\|X\beta - y\|^2$, where X is the $n \times m$ matrix of row vectors $X \triangleq (x_1^T, \ldots, x_n^T)^T$, and y is the column vector composed of response scalars $y \triangleq (y_1, \ldots, y_n)^T$.

The optimal solution to this convex problem, known as *ordinary least squares* (OLS), is given by [9]

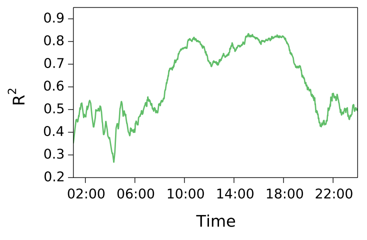
$$\beta = \left(X^T X\right)^{-1} X^T y \quad . \tag{1}$$

3.1 Monitoring OLS of Distributed Streams

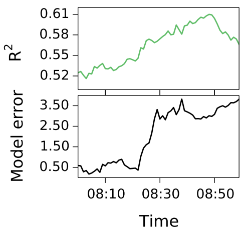
Assume that the observations $\{(x_i, y_i)\}$ are distributed across k nodes, and that these observations are dynamic – they change over time, as nodes receive new observations that replace older ones. As data evolves, it is possible that the previously computed model no longer matches the current true model. We wish to maintain an accurate estimation β_0 of the current global OLS model, β. The question is then when to update the model.

The simplest way is to update β every time a new observation arrives at the nodes, using a straightforward or incremental procedure. Though this gives the most accurate model, it is also wasteful. It requires communicating the update every time, and potentially disseminating the updated model to all nodes. It is especially wasteful when the current global model is similar to the old one.

Another simple solution the periodic algorithm: sending updates once every T times [39, 4] guarantees a reduction in



(a) Model β fit



(b) β_0 fit and error

Figure 1: (a) Model fit for the traffic dataset from Section 5.2, and (b) comparison with model error.

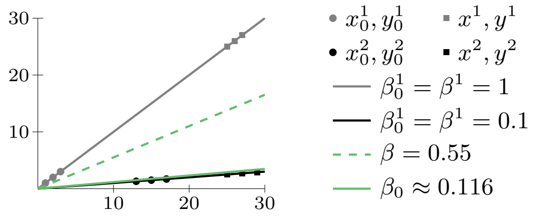


Figure 2: Monitoring distributed OLS models is difficult. Current local models β^1, β^2 are identical to the precomputed models β_0^1, β_0^2 but the combined global model is very different, $\beta - \beta_0 = 0.44$.

communication. The problem is that a fixed update schedule must balance communication and error *a priori*. For large T the estimate error may be unbounded for a long interval, yet if model changes are infrequent we waste communication.

Recent approaches monitor the prediction error $|y - X\beta_0|$, where X, y are the current observations [36, 4], the model's R^2 fit [3], or prediction error between divergent local models and the hypothetical global model [12].

Monitoring prediction error is not always sufficient, however. First, prediction is not the only application of regression. In some settings [9, 30] we are interested in model coefficients, rather than prediction performance. Yet prediction error may be small even when the difference between models is large. Consider the following example in $m = 3$ dimensions, with the precomputed model $\beta_0 = (1, 2, 3)^T$, the current model $\beta = (1, 1, 1)^T$, and with the observation $x = (-0.95, 2.05, -0.95)^T, y = \beta^T x = 0.15$. In this case the prediction error is small, $|x^T\beta_0 - y| = 0.15$, yet the models are very different: $\|\beta_0 - \beta\| = 2.236$.

Monitoring model fit is also tricky. Figure 1a shows the R^2 fit of the true model β in an interpolation problem (described in Section 5.2). The fit of the true model varies widely, and it is not clear where to set the R^2 monitoring threshold. Figure 1b shows an example where both the model error $\|\beta - \beta_0\|$ and the fit of the monitored model β_0 are increasing.

Thus, we aim to monitor the model estimation error itself. Let β_0 be the existing model, previously computed at some point in the past (the synchronization time), and let β be the hypothetical OLS model from current observations[1]. Given an error threshold ϵ, our goal is to raise an alert if

$$\|\beta_0 - \beta\| > \epsilon \quad ,$$

while minimizing communication. Note that monitoring model error is a more general approach: limiting model error allows us to bound prediction error $|x^T\beta_0 - x^T\beta|$ through Cauchy-Schwarz but not vice versa. Indeed, Sayed and Lopez [21] estimate the expected model error and use it to get expected prediction error.

4. MONITORING DISTRIBUTED LEAST SQUARES WITH CONVEX SUBSETS

Monitoring distributed OLS models is difficult because the global model cannot be inferred from the local model at each node. Even when all current local models β^j are similar to the precomputed local models β_0^j, the current global model β may be very different from the precomputed model β_0. Consider the example in Figure 2 with $k = 2$ nodes and $m = 1$. The global model deviation is very large, $\beta - \beta_0 = 0.44$, even though local models are identical: $\beta^1 = \beta_0^1$ and $\beta^2 = \beta_0^2$.

[1]β is hypothetical since we don't actually compute it.

To overcome this difficulty, we turn to *geometric monitoring*. Geometric monitoring [15, 16] is a communication-efficient approach that monitors whether a function of distributed data streams crosses a threshold. The key idea is to impose constraints on local data at the nodes, rather than on the function of the global aggregate. Given a function of the average of all local data and the threshold, we compute a convex *safe zone* for each node. As we show below, convexity plays a key role in the correctness of this scheme. As long as local data stay inside the safe zones, we guarantee that the function of the global average does not cross a threshold. Nodes communicate only when local data drifts outside the safe zone, which we call a *safe zone violation*. Once that happens, violations can be resolved, for example by gathering data from all nodes and recomputing β_0 and the safe zones.

To summarize, we want to impose conditions on the local data at each node so that as long as they hold, $\|\beta - \beta_0\| \le \epsilon$. The conditions should be "lenient" as possible – we wish to minimize the number of violations.

4.1 Notation

Define $A \triangleq \sum_{i=1}^n x_i x_i^T = X^T X$ and $c \triangleq \sum_{i=1}^n x_i y_i = X^T y$, and rewrite Eq. (1) as $\beta = A^{-1}c$. The global matrix A can be written as the sum of local matrices $A = \sum_{j=1}^k A^j$, where A^j is constructed from the local observations at node j. Similarly, $c = \sum_{j=1}^k c^j$ where c^j is constructed from the local observations at node j. Therefore, we can rewrite Eq. (1) as a function of the sums of A^j, c^j:

$$\beta = \left(\sum_j A^j\right)^{-1} \left(\sum_j c^j\right) = A^{-1}c \tag{2}$$

In our notation we use $\{A^j, c^j\}_k$ instead of the original observations $\{x_i, y_i\}_n$. Let $A_0 = \sum_{j=1}^k A_0^j$ and $c_0 = \sum_{j=1}^k c_0^j$ be the global sums of local values at nodes during the last sync time (when β_0 was computed), and $A = \sum_{j=1}^k A^j, c = \sum_{j=1}^k c^j$ be the current values. We define the local *drifts* as the deviation of local data from its initial values during sync: $\Delta^j = A^j - A_0^j$ and $\delta^j = c^j - c_0^j$.

We can now express global β and β_0 as a function of the averages of A^j, c^j and A_0^j, c_0^j. This will allow us to bound model changes inside a convex subset. Recall $\beta = A^{-1}c$. Similarly, $\beta_0 = A_0^{-1}c_0$. Values averaged over nodes (rather than summed) shall be denoted with $\hat{\cdot}$. Hence initial values

$$\hat{A}_0 = \frac{1}{k}\sum_{j=1}^{k} A_0^j \ , \ \hat{c}_0 = \frac{1}{k}\sum_{j=1}^{k} c_0^j \ , \ \hat{\beta}_0 = \hat{A}_0^{-1}\hat{c}_0 \ ,$$

and current values

$$\hat{A} = \frac{1}{k}\sum_{j=1}^{k} A^j \ , \ \hat{c} = \frac{1}{k}\sum_{j=1}^{k} c^j \ , \ \hat{\beta} = \hat{A}^{-1}\hat{c} \quad .$$

Note $(\frac{1}{k}A)^{-1} = kA^{-1}$ thus $\hat{\beta} = \hat{A}^{-1}\hat{c} = A^{-1}c = \beta$ and likewise $\hat{\beta}_0 = \beta_0$. In other words, we can compute the OLS model from the averages of local A^j, c^j rather than the sums:

$$\beta = \left(\frac{1}{k}\sum_j A^j\right)^{-1}\left(\frac{1}{k}\sum_j c^j\right) = \hat{A}^{-1}\hat{c} \qquad (3)$$

4.2 Convex Safe Zones

We propose to solve the monitoring problem by means of “good” convex subsets, called *safe zones*, of the data space. Each node monitors its own drift: as long as current values at local nodes (A^j, c^j) are sufficiently similar to their values at sync time (A_0^j, c_0^j), β_0 is guaranteed to be close to β.

Formally, we define a convex subset $\mathcal{C}$ in the space of matrix-vector pairs, such that $(0_{m\times m}, 0_m) \in \mathcal{C}$ and

$$(\Delta,\delta) \in \mathcal{C} \implies \|(\hat{A}_0+\Delta)^{-1}(\hat{c}_0+\delta) - \hat{A}_0^{-1}\hat{c}_0\| \le \epsilon \quad , \quad (4)$$

for any drift (Δ, δ), where $0_{m\times m}$ and 0_m are the $m \times m$ zero matrix and length m zero vector. Ideally, $\mathcal{C}$ should be “big”: as local data slowly drifts over time, it is desirable that drifts remain in $\mathcal{C}$ (otherwise communication is needed). Convexity plays a key role in our paradigm: if all drifts are in $\mathcal{C}$, then their average is also in $\mathcal{C}$.

Given such a subset $\mathcal{C}$, the basic monitoring paradigm is simple. As long as $(A^j - A_0^j, c^j - c_0^j) \in \mathcal{C}$, node j can remain silent. If all nodes are silent, then $\|\hat{\beta}_0 - \hat{\beta}\| = \|\beta_0 - \beta\| \le \epsilon$. If a violation of the local condition does occur at any node j, some form of violation recovery must take place, for example recomputing the global model and restarting monitoring.

We now prove the correctness of the paradigm.

LEMMA 1. *Let $\mathcal{C}$ be a convex subset that satisfies Eq.* (4)*. If $(\Delta^j, \delta^j) \in \mathcal{C}$ for all j, then $\|\beta - \beta_0\| \le \epsilon$.*

PROOF. Express $\hat{A}, \hat{c}$ using the average of local deviations:

$$\begin{aligned}(\hat{A},\hat{c}) &= \frac{1}{k}\sum_j (A^j, c^j) \\ &= (\hat{A}_0, \hat{c}_0) + \frac{1}{k}\sum_j (A^j - A_0^j, c^j - c_0^j) \\ &= (\hat{A}_0, \hat{c}_0) + \frac{1}{k}\sum_j (\Delta^j, \delta^j) \qquad (5)\end{aligned}$$

And from $\mathcal{C}$’s convexity,

$$\forall j \ (\Delta^j, \delta^j) \in \mathcal{C} \implies \frac{1}{k}\sum_j (\Delta^j, \delta^j) \in \mathcal{C} \qquad (6)$$

Denote $(\hat{\Delta}, \hat{\delta}) = \frac{1}{k}\sum_j (\Delta^j, \delta^j)$ and rewrite Eq. (5) and (6):

$$\begin{aligned}(\hat{A},\hat{c}) &= (\hat{A}_0, \hat{c}_0) + (\hat{\Delta}, \hat{\delta}) \\ \forall j \ (\Delta^j, \delta^j) \in \mathcal{C} &\implies (\hat{\Delta}, \hat{\delta}) \in \mathcal{C}\end{aligned}$$

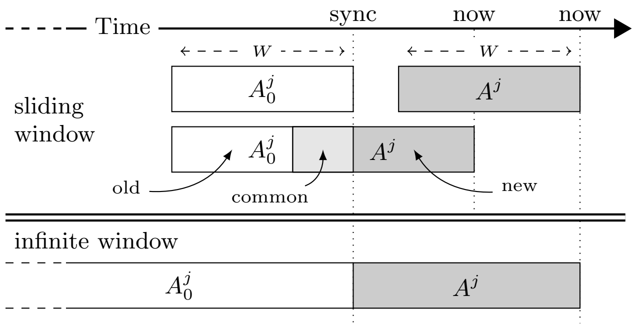


Figure 3: Sliding and infinite window models. When A^j overlaps A_0^j, $\Delta^j = A^j - A_0^j = \sum_{\mathbf{new}} x_i x_i^T - \sum_{\mathbf{old}} x_i x_i^T$.

Substitute in Eq. (4) to finally obtain:

$$\begin{aligned}\forall j \ (\Delta^j, \delta^j) \in \mathcal{C} \quad \implies \quad & \|(\hat{A}_0 + \hat{\Delta})^{-1}(\hat{c}_0 + \hat{\delta}) - \hat{A}_0^{-1}\hat{c}_0\| = \\ & \|\hat{\beta} - \hat{\beta}_0\| = \|\beta - \beta_0\| \le \epsilon\end{aligned}$$

which completes the proof. □

4.3 Infinite and Sliding Window

We differentiate between two different variations for computing the global model: sliding window and infinite window. In the *sliding window* model, β is computed from the last W samples seen at each node, and similarly β_0 is computed from the last W samples before sync. Conversely, in the *infinite window* model β is computed over all observations seen thus far, while β_0 is computed from all observations seen until last sync. Figure 3 illustrates these two models. Though the sliding window is clearly more practical, the infinite window model may be useful in some settings and so we discuss both.

Sliding Window. In the sliding window model each node computes A^j from the W samples seen at node j, while A_0^j (and hence $\hat{A}_0$) is built from the last W samples before sync. Computing Δ^j and δ^j, however, requires substracting observations that left the sliding window. If A_0^j, c_0^j and A^j, c^j do not overlap (Figure 3, top), then clearly $\Delta^j = A^j - A_0^j$ and $\delta^j = c^j - c_0^j$. It is also possible, however, that the current window overlaps the window used to build β_0. Figure 3 (middle) illustrates this case: Δ^j, δ^j become the sum of new samples from A^j, c^j minus the sum of old (non-overlapping) samples from A_0^j, c_0^j.

The convex constraint $\mathcal{C}$ on (Δ, δ) for this model is:

$$\epsilon\|\hat{A}_0^{-1}\Delta\| + \|\hat{A}_0^{-1}\delta\| + \|\hat{A}_0^{-1}\Delta\beta_0\| \le \epsilon \quad , \qquad (7)$$

where $\|A\|$ is the L_2 *operator norm* of the matrix A. The derivation of the convex constraint $\mathcal{C}$ is quite technical, and the details are available in Appendix A.

Alg. 1 shows the resulting monitoring algorithm each node runs. Note monitoring does not require any matrix inversions. Each node applies the local constraint from Eq. (7) to its own data. When a violation occurs at any node, it is reported to a *coordinator* node. The coordinator (Alg. 2) polls all nodes for their local data, computes a updated global model β_0 and distributes it to all nodes, along with updated $\hat{A}_0^{-1}$ used in the constraint. Monitoring then resumes. This is the simplest violation resolution protocol. We briefly discuss

Algorithm 1 Node j update with new observation x, y.

```
 1: $(A^j\ ,\ c^j) \leftarrow (A^j + x^T x\ ,\ c^j + x^T y)$
 2: Insert new $x, y$ to head of sliding window.
 3: Retrieve old $x_w, y_w$ exiting end of sliding window.
 4: $(A^j\ ,\ c^j) \leftarrow (A^j - x_w^T x_w\ ,\ c^j - x_w^T y_w)$
 5: $(\Delta^j\ ,\ \delta^j) \leftarrow (A^j - A_0^j\ ,\ c^j - c_0^j)$
 6: if $\epsilon\|\hat{A}_0^{-1}\Delta^j\| + \|\hat{A}_0^{-1}\delta^j\| + \|\hat{A}_0^{-1}\Delta^j\beta_0\| \le \epsilon$ then
 7:     Report violation to coordinator.
 8:     Receive new $\beta_0$, $\hat{A}_0^{-1}$ from coordinator.
 9:     $(A_0^j\ ,\ c_0^j) \leftarrow (A^j\ ,\ c^j)$
10: end if
```

Algorithm 2 Coordinator violation resolution algorithm.

```
1: Poll all nodes for $A^j, c^j$.
2: Compute updated $\hat{A}_0^{-1}, \beta_0$ from $A^j, c^j$ and distribute.
```

more sophisticated schemes [14, 15, 5] in Section 6. Similarly, the coordinator can use any algorithm to compute $\hat{A}_0^{-1}, \beta_0$.

Infinite Window. In this model the local drifts of each node i are $\Delta^j = A^j - A_0^j$ and $\delta^j = c^j - c_0$ as before, but A^j and c^j are computed from all observations ever seen at the node. We can use the same convex constraint from Section 4.3, but in this case Δ^j grows indefinitely, and so the condition $\|\hat{A}_0^{-1}\Delta\| < 1$ is not easy to satisfy, and may cause frequent synchronizations. Instead, we start from Eq. (9) and develop a more lenient constraint for this model. The resulting algorithm will be similar to Alg. 1, but without lines 2–4 and with the updated constraint in line 6. The coordinator algorithm is the same.

The convex constraint for the infinite window case is

$$\|\hat{A}_0^{-1}\delta\| + \|\hat{A}_0^{-1}\hat{c}_0\| \le \epsilon \quad . \tag{8}$$

Appendix B details its derivation.

Note that δ accumulates more samples as time passes, while $\hat{A}_0$ remains fixed. As δ's "weight" (number of samples) grows beyond $\hat{A}_0$'s, the constraint no longer holds and sycrhonization is needed. One way to avoid this is to replace $\|\hat{A}_0^{-1}\delta\|$ in Eq. 8 with $\|\Delta^{-1}\delta\|$, which is correct (using the same line of arguments in the Appendix). Alternatively, note that after each sync the samples from all δ^j's are added to the new $\hat{A}_0$, so its "weight" is roughly doubled. Thus $\hat{A}_0$'s weight grows exponentially, and synchronizations become increasingly rare.

4.4 Norm Constraint and the Sliding Window

The sliding window model constraint Eq. (7) requires $\|\hat{A}_0^{-1}\Delta^j\| < 1$ (embodied as $\epsilon\|\hat{A}_0^{-1}\Delta^j\| + \cdots \le \epsilon$). This requirement depends only the independent variables X^j, and does not depend in any way on the dependent variable y^j. It is quite possible that β is close to β_0, yet the norm constraint is violated, incurring extra communication. Fortunately, for many reasonable data distributions, if window size W is linear in the number of independent variables m then the norm constraint is satisfied almost surely. The details are in Appendix C.

The analysis assumes that consecutive observations in the data stream are independent. Consider, however, the case where some variables come from an over-sampled sensor, or measure slowly changing phenomena. In such cases $\|\Delta\|$ grows faster, linear in the number of identical observations, and will overwhelm A_0^{-1} faster. This can result in more frequent violations of the constraint, hence more communication. Such cases can be mitigated by increasing the window size W, subsampling (since data changes slowly anyway), or by the use of *generalized least squares* [9] with an appropriate scaling matrix for the time series process (Section 4.5).

4.5 Regularization and Variants

Our scheme generalizes very well to more sophisticated least squares variants [9]. We show two examples.

In *regularized least squares* the minimized function includes a regularization term to mitigate the effects of outliers and avoid overfitting. A commonly used form is *Tikhonov regularization*, also known as *ridge regression*, which finds β that minimizes $\|X\beta - y\|^2 + \|R^T R\beta\|^2$, where R is a suitable regularization matrix R. For $R = 0$ the problem reduces to ordinary least squares, and for $R = \lambda I$ it reduces to L_2 regularization. The optimal solution to this problem is

$$\beta = \left(X^T X + R^T R\right)^{-1} X^T y \quad .$$

This solution is quite similar to Eq. (1) and indeed we can monitor it using the same technique: compute $B_0^j = A_0^j + \frac{1}{k}R^T R$ and the resulting $B_0, \hat{B}_0$, and use them in Lemma 1 instead of $A_0^j, A_0, \hat{A}_0$.

Similarly, *generalized least squares* handles correlated measurements and errors by minimizing the Mahalanobis distance $(Y - X\beta)^T S^{-1}(Y - X\beta)$, where S is the covariance matrix of the residuals (errors). Again, GLS reduces to OLS if $S = I$. As before, we can monitor the optimal solution

$$\beta = \left(X^T S^{-1} X\right)^{-1} X^T \tilde{y}\ , \text{ where } \tilde{y} \triangleq S^{-1} y$$

by monitoring $B = \sum_i S^{-1} x_i x_i^T$ and $d = \sum_i x_i \tilde{y}_i$. GLS is particularly useful in time series analysis, where S is the process' structured covariance (or autocorrelation) matrix [32].

5. EVALUATION

We evaluated performance of our monitoring algorithm, DILSQ, for Distributed Least SQuare monitor, using simulations with two synthetic and two real-world distributed datasets. For each dataset, we run through the data, simulate the nodes (Alg 1) and the coordinator (Alg 2), count messages, and keep track of the resulting true models β and the current monitored models β_0. Our simulations use discrete time (rounds), and we use the OLS variant of our algorithm with sliding window (Section 4.3), except for the gas sensor dataset which uses the GLS variant (Section 4.5).

Our baseline is the *naive algorithm*, where each node sends every new measurement to a centralized location each round. We compare DILSQ to the T-periodic algorithm, denoted PER(T), a simple sampling algorithm that sends updates every T rounds. Though PER cannot guarantee maximum error, it can achieve arbitrarily low communication.

Our main performance metric is communication, measured in *normalized messages* – the average messages sent per round by each node [3]. Note that communication of the naive algorithm is always 1. When calculating and reporting results, we skip the first (incomplete) window (or the first epoch for the drift dataset described below).

DILSQ is designed to communicate as little as possible while always maintaining maximum model error below ϵ. It

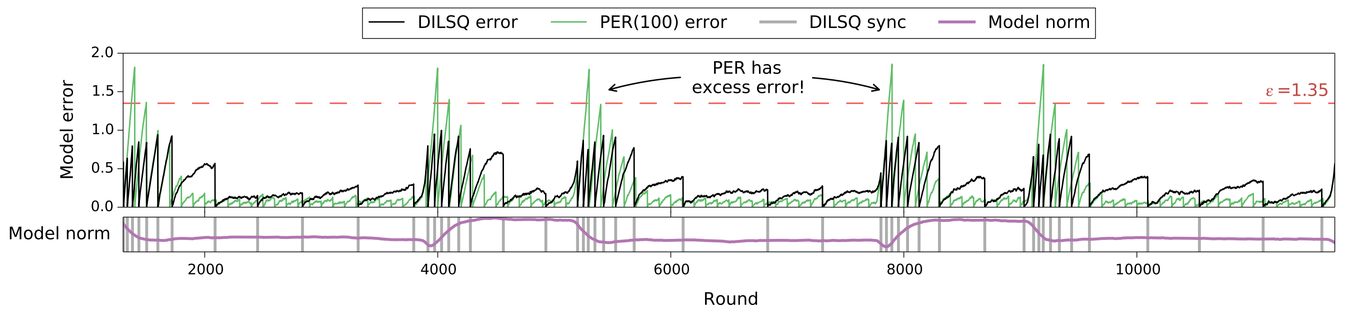


Figure 4: DILSQ model error (black) and syncs (bottom vertical lines) per round, compared to PER(100) error (green), for $k = 10$ simulated nodes with $m = 10$ dimensions, and threshold $\epsilon = 1.35$. Both algorithms reduce communication to 1%, but DILSQ only syncs when β changes (bottom purple line shows $\|\beta\|$). PER(100) syncs every 100 rounds, but is unable to maintain error below the threshold (dashed horizontal line).

guarantees maximum model error below the user-selected threshold ϵ, but PER does not. Hence, when comparing the two, we find *a posteriori* the maximum period T (hence minimum communication) for which the maximum error of PER(T) is equal or below that of DILSQ. Note this gives PER an unrealistic advantage. First, in a realistic setting we cannot know *a priori* the optimal period T. Second, model changes in realistic settings are not necessarily stationary: the rate of model change may evolve, which DILSQ will handle gracefully while PER cannot.

5.1 Synthetic Datasets

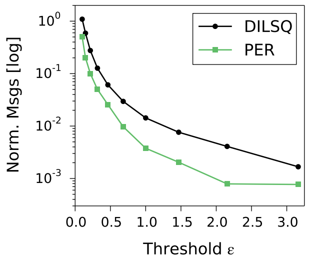
We use two types of synthetic dataset. In the *fixed* dataset, the true model $\beta_{\text{true}} \in \mathbb{R}^m$ is fixed, with elements drawn i.i.d from $N[0, 1]$[2]. We generate R rounds with k nodes, each receiving at each round a new data vector x of size m and scalar y. x is drawn i.i.d from $N(0, 1)$, and $y = x^T\beta_{\text{true}} + n$ where $n \sim N(0, \sigma^2)$ is Gaussian white noise of strength σ. In the *drift* dataset the coefficients of β_{true} change rapidly during 25% of one *epoch*, and are fixed during the rest of the epoch. We generate observations for E epochs using the same procedure. For each experiment we generate new data.

Default parameter values are $k = 10$ nodes, $m = 10$ dimensions, noise magnitude $\sigma = 10$ (to generate interesting results given the large window), window size $W = 1300$ and maximum error threshold $\epsilon = 0.5$, which is quite strict[3]. We generate $R = 16900$ rounds for the fixed dataset, or $E = 5$ epochs of 3900 rounds each for drift dataset.

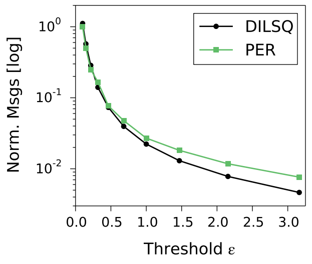
Figure 4 shows the behavior of the monitoring algorithm over such a simulation on the drift dataset with $\epsilon = 1.35$ and 3 epochs. For this configuration, DILSQ achieves communication of 0.01 messages per node per round, and the model error is always below the threshold. Conversely, the equivalent PER(100) algorithm is unable to maintain the error below the threshold, which would require a higher update frequency. When model changes in β are large and frequent DILSQ performs more syncrhonizations, resulting in updated $\beta_0 = \beta$ that decreases the error. When β is stable (it is never truly constant due to noise), syncrhonizations are much rarer. The periodic algorithm, on the other hand, syncrhonizes every 100 iterations even during the periods where β changes very little.

[2]therefore $\|\beta\|^2 \sim \chi^2_m$

[3]Given that elements of both β_0 and β are i.i.d $N(0, \sigma)$, then $\frac{\|\beta-\beta_0\|}{\sqrt{2}\sigma} \sim \chi_m$. The probability that a random $e = \beta - \beta_0$ will overwhelm ϵ is $P = 1 - \text{CDF}_{\chi_m}(\frac{\epsilon}{\sqrt{2}\sigma}) > 1 - 10^{-8}$.



(a) Fixed dataset



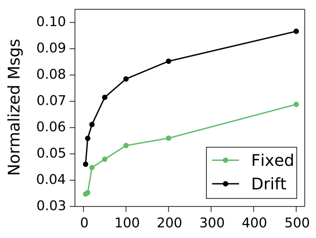
(b) Drift dataset

Figure 5: Communication for DILSQ (black) and periodic algorithm tuned to achieve same max error (green) at different threshold values. DILSQ communication on fixed model drops to zero for more permissive ϵ (not shown on logarithmic scale).

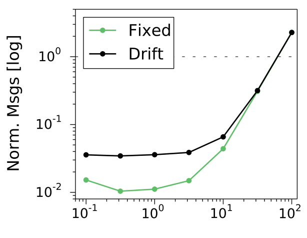
5.1.1 *Effect of Threshold*

Figure 5 shows the communication required for different threshold levels for the DILSQ algorithm, and the minimal communication required to match DILSQ using the PER algorithm with optimal period, as discussed above. For the fixed model dataset (Figure 5a) neither algorithm needs to sync very often to provide an accurate estimate. Had there been no noise, a single initial synchronization would have been sufficient, regardless of threshold. Note that for more permissive threshold values (or smaller noise magnitude σ) DILSQ achieved *zero* communication (beyond initial sync) for the fixed dataset (not shown in this log-scale figure).

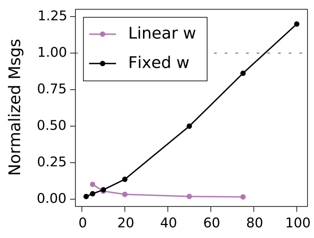
Performance on the drift dataset (Figure 5b) is more interesting. When ϵ is very strict, both algorithms perform roughly the same, with normalized messages of 0.25–0.75. As ϵ grows DILSQ develops an increasing advantage over PER with optimal period. The optimal period must be low enough to match the quickly changing model, and is wasteful on the intervals where β is quiescent. For our dataset, β_{true} is constant during roughly 75% of each epoch. For datasets with larger quiescent periods (or smaller window), the advantage of DILSQ will be even larger.



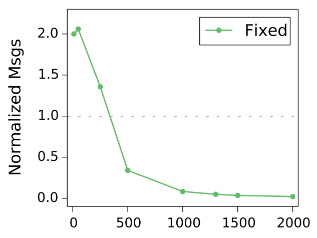
(a) Nodes k



(b) Noise σ [log]



(c) Dimension m



(d) Window size W

Figure 6: Communication vs. different parameters for the fixed (green) and drift (black) datasets. (a) shows DILSQ is scalable: communication increases slowly with number of nodes. (b) shows communication is fairly constant when noise is small ($\sigma < 10$). Comm. is zero for fixed model at low noise (not shown). (c) shows the required window size W is linear in m: communication does not increase when W is suitably sized (purple). (d) shows performance with fixed dataset. If $W < 144m$ data periodically saturates the norms in Eq. (7).

5.1.2 *Scalability*

Figure 6 explores how performance of DILSQ scales with different parameters.

Figure 6a shows communication for different values of the number of nodes k. We observe communication increases slowly, remaining below 10% even with 500 nodes.

Figure 6b shows normalized messages obtained at different noise magnitudes. Below a certain level of noise, communication is fairly constant, reflecting the choice of threshold ϵ. At lower values of noise (not shown), DILSQ requires no communication for the fixed model dataset, beyond the first window of W observations.

Figure 6c compares communication with the number of independent variables m on the drift dataset, confirming our analysis in Section 4.4. When window size W is fixed, communication grows linearly with dimension m. However, if W grows linearly with m, we see that communication remains very low (and in fact decreases a little). In both cases we keep epoch length to be $3W$ to maintain the same rate of change of β across the window.

Similarly, Figure 6d shows what happens when the window size is too small compared to the value predicted in Section 4.4. It depicts communication obtained on the fixed dataset, as a function of window size W. As window size decreases below $144m$ (see Appendix C), constraint violations are more frequent as data periodically overwhelms the norms in Eq. (7). As we will see below, in practical settings a much lower W can be used, since data values have finite ranges, change slowly, and model changes are more frequent.

5.2 Traffic Monitoring

Consider the following interpolation problem: given periodical traffic measurements (average velocity every minute) from a small number of sensors embedded along a long road, we wish to infer the current average velocity at every point along the road. We aim to solve this problem using polynomial regression. Note that in this case we have no good way to measure the true error of our model, since we do not have sensors in other locations. Moreover, as Figure 1 (derived from the same data below) shows, monitoring model fit (R^2) is also problematic (Section 3.1). Instead, we rely on the fact that we can limit the model error $\|\beta - \beta_0\|$.

We used two weeks' worth of velocity data collected during November 2014 from $k = 6$ sensors located along the Grenoble south ring in France [27]. Reported measurements of each sensor are aggregated once per minute, and when measurements are unavailable average velocity was assumed to be unchanged. The road is composed of several sections, and we model it as the interval $[1, 18]$, where the sensors are located at $l \in \{1, 3, 7, 11, 14, 18\}$. The data from every sensor at location l is always $x = [1, l, l^2, l^3, l^4]$, and y is the velocity measured by the sensor. Given model β built from measurements from the last hour ($W = 60$), the interpolated velocity at location $i \in [1, 18]$ is $[1, i, i^2, i^3, i^4]\beta$.

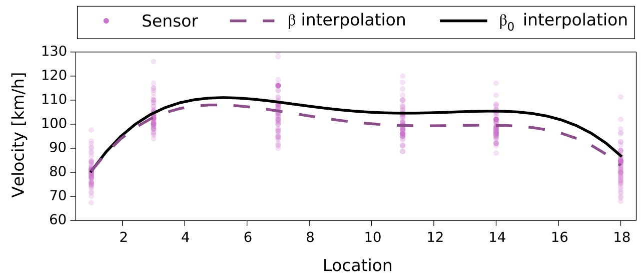
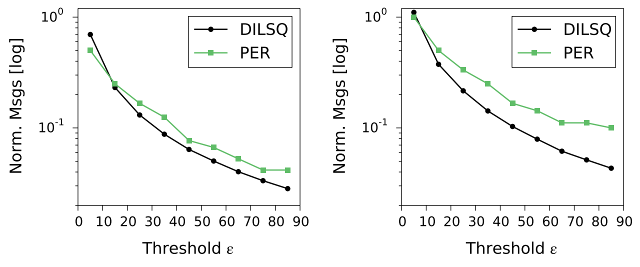


Figure 7: Velocity measurements from 8am–9am (pink dots), and interpolated velocity at 9am: true model (dashed) and DILSQ approximation (black).

Figure 7 shows the result of one such a prediction for 9am on Nov 1 2014, produced using $\epsilon = 25$. The pink dots represent average velocity measurements of each sensor between 8am to 9am. The dashed purple line is velocity interpolated using the exact least squares model β, while the black line is interpolated using DILSQ approximation β_0. Observe the resulting interpolation is fairly accurate, with errors below 10km/h across of range of interpolated positions.

Figure 8 explores the communication of DILSQ and matching PER with various levels of ϵ, for window sizes 60 and 30. DILSQ is superior to PER across all ranges except the unrealistically strict $\epsilon = 5$ (average $\|\beta\|$ is roughly 100). For one hour window, DILSQ obtains 0.12 normalized messages for $\epsilon = 25$ used in Figure 7, and can reduce communication to 0.03 for $\epsilon = 85$. For a much smaller window size of half an hour, DILSQ requires more communication but still achieves considerable communication reduction: it requires 20% communication for $\epsilon = 25$ and can use as little as 5% for $\epsilon = 85$. Finally, we observe that the communication gap between DILSQ and PER increases considerably with smaller window size, as β changes more quickly and is more sensitive to noise.



(a) Window size $W = 60$ (b) Window size $W = 30$

Figure 8: Communication for DILSQ (black) and periodic algorithm (green) on the traffic dataset at different ϵ values.

5.3 GLS on Gas Sensor Time Series

Data in this experiment consists of measurements collected by an array of 16 chemical sensors recorded at a sampling rate of 25Hz for 5 minutes, resulting in 7500 data points for each sensor. This dataset is described in [42], and is publicly available [19]. The original goal in [42] is to identify certain gas classes given high-level frequency features. Since the original target variable is nominal and fixed throughout the run in each experiment, we defined a different regression problem. We divided the 16 sensors to $k = 4$ "nodes", where in each node three sensors serve as the data x while the remaining sensor serves as the response y. We also added a constant variable 1 to x, to allow intercept in the model, hence $m = 4$. The regression task is therefore to predict the value of the 4th sensor in each node using the first three.

Note that in this setting measurement errors cannot be assumed to be independent, so an OLS models is ill-suited here. Instead, we assume errors are an AR(1) process and monitor the generalized least squares model [9]. We used an AR(1) parameter value $\phi = 0.95$ for the autocorrelation matrix [32]. Average $\|\beta\|$ is 0.3, so we use $\epsilon = 0.1$, resulting in 0.17 normalized messages for DILSQ. We note that using an OLS model with the same ϵ resulted in 1.15 normalized messages – the OLS model had to be updated very frequently as it was unstable.

We repeated the experiment for various ϵ values in the range [0.01,1] (figure omitted for lack of space). For $\epsilon < 0.1$ DILSQ is clearly superior: PER must communicate every round ($T = 1$) in order to match DILSQ, which achieves communication between 0.2 and 1 (for $\epsilon = 0.01$). When ϵ is more permissive, however, PER is superior and can obtain the same maximum error with less communication: with an extremely permissive $\epsilon = 1$, DILSQ requires 0.04 normalized messages while PER requires 0.015 for the same maximum error (though, of course, optimal T must be known *a priori* to achieve this performance).

6. CONCLUSIONS

DILSQ is the first communication-efficient monitoring algorithm for least-squares regression models that limits the error in model coefficients. By monitoring the deviation of the existing model from the true model, our approach is able to avoid costly communication and model recomputations, while guaranteeing bounded model error. Each round, each node checks a simple local constraint on its own local data; if it is satisfied, communication is avoided. If not, violation is resolved by collecting data from all nodes and computing a new global model. Evaluation on real-world datasets shows a communication reduction of up to two orders of magnitude. Simulations on synthetic datasets show our algorithm scales well with the number of nodes.

We emphasize that correctness of the local constraint is independent of network topology and the algorithm used to compute the model β_0. Hence it is straightforward to adapt our method to other settings. First, the role of the coordinator can easily be replaced with convergecasting [4, 39], yielding a peer-to-peer monitor. Alternatively, our distributed monitoring approach can easily be combined with an efficient distributed computation technique, enjoying the best of both worlds: the current model can be computed during sync using any of several existing algorithms, be they exact, iterative, or distributed [7, 21]. Similarly, our method is compatible with recent communication reduction techniques from the field of distributed streams, such as reference point prediction [6], individualized constraints or slack [14, 5], and local violation resolution [15]. We leave such extensions for future work.

7. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union's Seventh Framework Programme FP7-ICT-2013-11 under grant agreement Nº 619491 and Nº 619435.

8. REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Trans. Sig. Proc.*, 2006.

[2] Z. D. Bai and Y. Q. Yin. Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. *Ann. Prob.*, 1993.

[3] K. Bhaduri, K. Das, and C. Giannella. Distributed monitoring of the R^2 statistic for linear regression. In *Proc. SDM*, 2011.

[4] K. Bhaduri and H. Kargupta. An efficient local algorithm for distributed multivariate regression in peer-to-peer networks. In *Proc. SDM*, 2008.

[5] M. Gabel, D. Keren, and A. Schuster. Communication-efficient distributed variance monitoring and outlier detection for multivariate time series. In *Proc. IPDPS*, 2014.

[6] N. Giatrakos, A. Deligiannakis, M. Garofalakis, I. Sharfman, and A. Schuster. Prediction-based geometric monitoring over distributed data streams. In *Proc. SIGMOD*. ACM, 2012.

[7] C. Guestrin, P. Bodík, R. Thibaux, M. A. Paskin, and S. Madden. Distributed regression: an efficient framework for modeling sensor network data. In *Proc. IPSN*, 2004.

[8] R. Gupta, K. Ramamritham, and M. K. Mohania. Ratio threshold queries over distributed data sources. *PVLDB*, 2013.

[9] F. Hayashi. *Econometrics*. Princeton University Press, 2000.

[10] L. Huang, X. Nguyen, M. N. Garofalakis, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, and N. Taft. Communication-efficient online detection of network-wide anomalies. In *INFOCOM*, 2007.

[11] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM TOCS*, 2005.

[12] M. Kamp, M. Boley, D. Keren, A. Schuster, and I. Sharfman. Communication-efficient distributed online prediction by dynamic model synchronization. In *Proc. ECML PKDD*, 2014.

[13] S. R. Kashyap, J. Ramamirtham, R. Rastogi, and P. Shukla. Efficient constraint monitoring using adaptive thresholds. In *ICDE*, 2008.

[14] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proc. SIGMOD*, 2006.

[15] D. Keren, G. Sagy, A. Abboud, D. Ben-David, A. Schuster, I. Sharfman, and A. Deligiannakis. Geometric monitoring of heterogeneous streams. *IEEE Trans. Knowl. Data Eng.*, 2014.

[16] D. Keren, I. Sharfman, A. Schuster, and A. Livne. Shape sensitive geometric monitoring. *IEEE Trans. Knowl. Data Eng.*, 2012.

[17] A. Knutson and T. Tao. Honeycombs and sums of Hermitian matrices. *Notices Amer. Math. Soc.*, 2001.

[18] A. Lazerson, I. Sharfman, D. Keren, A. Schuster, M. N. Garofalakis, and V. Samoladas. Monitoring distributed streams using convex decompositions. *PVLDB*, 2015.

[19] M. Lichman. UCI machine learning repository, 2013.

[20] W. Lin, J. Cao, and X. Liu. E^3: Towards energy-efficient distributed least squares estimation in sensor networks. In *IWQoS 2014*, 2014.

[21] C. G. Lopes and A. H. Sayed. Distributed adaptive incremental strategies: Formulation and performance analysis. In *Proc. ICASSP*, 2006.

[22] V. A. Marčenko and L. A. Pastur. Distribution of eigenvalues for some sets of random matrices. *Math. USSR Sb.*, 1967.

[23] G. Mateos, J. A. Bazerque, and G. B. Giannakis. Distributed sparse linear regression. *IEEE Trans. Sig. Proc.*, 2010.

[24] G. Mateos and G. B. Giannakis. Distributed recursive least-squares: Stability and performance analysis. *IEEE Trans. Sig. Proc.*, 2012.

[25] S. Michel, P. Triantafillou, and G. Weikum. KLEE: a framework for distributed top-k query algorithms. In *Proc. VLDB*, 2005.

[26] K. S. Miller. On the inverse of the sum of matrices. *Mathematics Magazine*, 1981.

[27] F. Morbidi, L. Leon Ojeda, C. Canudas De Wit, and I. Bellicot. A new robust approach for highway traffic density estimation. In *ECC14*, 2014.

[28] M. A. Paskin, C. Guestrin, and J. McFadden. A robust architecture for distributed inference in sensor networks. In *Proc. IPSN*, 2005.

[29] S. Roman. *Advanced Linear Algebra*, volume 135 of *Graduate Texts in Mathematics*. Springer, 1995.

[30] S. Ronen, B. Gonçalves, K. Z. Hu, A. Vespignani, S. Pinker, and C. A. Hidalgo. Links that speak: The global language network and its association with global fame. *PNAS*, 2014.

[31] M. Rudelson and R. Vershynin. Non-asymptotic theory of random matrices: extreme singular values. In *Proc. ICM*, 2010.

[32] A. Saudargienė. Structurization of the covariance matrix by process type and block-diagonal models in the classifier design. *Informatica*, 1999.

[33] A. H. Sayed. Adaptive networks. *Proc. IEEE*, 2014.

[34] S. Shah and K. Ramamritham. Handling non-linear polynomial queries over dynamic data. In *ICDE*, 2008.

[35] I. Sharfman, A. Schuster, and D. Keren. A geometric approach to monitoring threshold functions over distributed data streams. *ACM Trans. Database Syst.*, 2007.

[36] X. Song, C. Wang, J. Gao, and X. Hu. DLRDG: distributed linear regression-based hierarchical data gathering framework in wireless sensor network. *Neural Comput. Appl.*, 2013.

[37] J. A. Tropp and A. C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Inf. Theory*, 2007.

[38] S. Y. Tu and A. H. Sayed. Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks. *IEEE Trans. Sig. Proc.*, 2012.

[39] R. Wolff, K. Bhaduri, and H. Kargupta. A generic local algorithm for mining data streams in large distributed systems. *IEEE Trans. Knowl. Data Eng.*, 2009.

[40] L. T. Yang and R. P. Brent. Parallel MCGLS and ICGLS methods for least squares problems on distributed memory architectures. *J. Supercomput.*, 2004.

[41] Y. Zhang, D. M. Sow, D. S. Turaga, and M. van der Schaar. A fast online learning algorithm for distributed mining of BigData. *SIGMETRICS PER*, 2014.

[42] A. Ziyatdinov, J. Fonollosa, L. Fernández, A. Gutierrez-Gálvez, S. Marco, and A. Perera. Bioinspired early detection through gas flow modulation in chemo-sensory systems. *Sens. Actuators B Chem.*, 2015.

APPENDIX

A. SLIDING WINDOW CONSTRAINT

To find a convex subset $\mathcal{C}$ satisfying the condition of Eq. (4), we first review some notions and well-known results on norms of real matrices [29]. We use the L_2 norm throughout.

DEFINITION 1. *Let A be a matrix. Its* operator norm*, or* spectral norm*, hereafter just* norm*, is defined as*

$$\|A\| = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}$$

It follows that for a matrix A and vector x, $\|Ax\| \leq \|A\|\|x\|$.

Moreover, for any two matrices A, B: $\|A+B\| \leq \|A\|+\|B\|$ and $\|AB\| \leq \|A\|\|B\|$. If A is a symmetric matrix, $\|A\| = \max_i |\lambda_i|$ where λ_i are the eigenvalues of A. Additionally, if A, B are symmetric then $\|AB\| = \|BA\|$.

LEMMA 2. *For square A with $\|A\| < 1$, (I-A) is invertible, the inverse being the* Neumann series *[26]: $(I-A)^{-1} = I + A + A^2 + A^3 + \dots$.*

LEMMA 3. *If A is square and $\|A\| < 1$, then*

$$\|(I+A)^{-1}\| = \|I - A + A^2 - A^3 + \dots\| \leq \frac{1}{1-\|A\|} \quad .$$

PROOF. Apply Lemma 2 and the triangle inequality:

$$\begin{aligned} \|\,(I+A)^{-1}\,\| &= \|I - A + A^2 - A^3 + A^4 - \dots\| \\ &\leq \|I\| + \|A\| + \|A^2\| + \dots \leq \frac{1}{1-\|A\|} \quad , \end{aligned}$$

since it is the sum of a geometric series. □

We begin by subtracting and adding $(\hat{A}_0 + \Delta)^{-1}\hat{c}_0$ to the bounded expression in Eq. (4):

$$\begin{aligned} &\|(\hat{A}_0+\Delta)^{-1}(\hat{c}_0+\delta) - \hat{A}_0^{-1}\hat{c}_0\| = \\ &\qquad \|(\hat{A}_0+\Delta)^{-1}\delta + \left((\hat{A}_0+\Delta)^{-1} - \hat{A}_0^{-1}\right)\hat{c}_0\| \quad . \end{aligned}$$

Applying the triangle inequality, we obtain:

$$\underbrace{\|(\hat{A}_0+\Delta)^{-1}\delta\|}_{E_1} + \underbrace{\left\|\left((\hat{A}_0+\Delta)^{-1} - \hat{A}_0^{-1}\right)\hat{c}_0\right\|}_{E_2} \quad . \tag{9}$$

Next, note that

$$(\hat{A}_0+\Delta)^{-1} = \left(\hat{A}_0\left(I + \hat{A}_0^{-1}\Delta\right)\right)^{-1} = \left(I + \hat{A}_0^{-1}\Delta\right)^{-1}\hat{A}_0^{-1}$$

and, assuming $\|\hat{A}_0^{-1}\Delta\| < 1$, we apply Lemma 2 to obtain:

$$\begin{aligned} &(\hat{A}_0+\Delta)^{-1} \\ &\quad = \left(I - \hat{A}_0^{-1}\Delta + \hat{A}_0^{-1}\Delta\hat{A}_0^{-1}\Delta - \dots\right)\hat{A}_0^{-1} \qquad (10) \\ &\quad = \hat{A}_0^{-1} - \hat{A}_0^{-1}\Delta\hat{A}_0^{-1} + \hat{A}_0^{-1}\Delta\hat{A}_0^{-1}\Delta\hat{A}_0^{-1} - \dots \qquad (11) \end{aligned}$$

Note the assumption $\|\hat{A}_0^{-1}\Delta\| < 1$ is not trivial, and we discuss it in Section 4.4 and Appendix C.

We now apply Eq. (10) and Lemma 3 to E_1 in Eq. (9):

$$\begin{aligned} E_1 &= \|(\hat{A}_0 + \Delta)^{-1}\delta\| \\ &= \left\|\left(I - \hat{A}_0^{-1}\Delta + \hat{A}_0^{-1}\Delta\hat{A}_0^{-1}\Delta - \dots\right)\hat{A}_0^{-1}\delta\right\| \\ &\le \|I - \hat{A}_0^{-1}\Delta + \hat{A}_0^{-1}\Delta\hat{A}_0^{-1}\Delta - \dots\|\|\hat{A}_0^{-1}\delta\| \\ &\le \frac{\|\hat{A}_0^{-1}\delta\|}{1 - \|\hat{A}_0^{-1}\Delta\|} \end{aligned} \tag{12}$$

Similarly, we apply Eq. (11) to E_2:

$$\begin{aligned} E_2 &= \left\|\left((\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1}\right)\hat{c}_0\right\| \\ &= \left\|\left(\hat{A}_0^{-1} - \hat{A}_0^{-1}\Delta\hat{A}_0^{-1} + (\hat{A}_0^{-1}\Delta)^2\hat{A}_0^{-1} - \dots - \hat{A}_0^{-1}\right)\hat{c}_0\right\| \\ &= \left\|-\left(\hat{A}_0^{-1}\Delta + (\hat{A}_0^{-1}\Delta)^2 - \dots\right)\hat{A}_0^{-1}\hat{c}_0\right\| \\ &= \left\|\left(I + \hat{A}_0^{-1}\Delta - (\hat{A}_0^{-1}\Delta)^2 + \dots\right)\hat{A}_0^{-1}\Delta\beta_0\right\| \end{aligned} \tag{13}$$

Applying Lemma 3 to Eq. (13) we obtain:

$$\begin{aligned} E_2 &\le \left\|I + \hat{A}_0^{-1}\Delta - (\hat{A}_0^{-1}\Delta)^2 + \dots\right\| \|\hat{A}_0^{-1}\Delta\beta_0\| \\ &\le \frac{\|\hat{A}_0^{-1}\Delta\beta_0\|}{1 - \|\hat{A}_0^{-1}\Delta\|} \end{aligned} \tag{14}$$

Substituting Eq. (12) and (14) in Eq. (9) and rearranging, we arrive at the convex constraint $\mathcal{C}$ on (Δ, δ):

$$\epsilon\|\hat{A}_0^{-1}\Delta\| + \|\hat{A}_0^{-1}\delta\| + \|\hat{A}_0^{-1}\Delta\beta_0\| \le \epsilon \quad . \tag{15}$$

This convex constraint allows us to apply Lemma 1. Satisfying Eq. (15) guarantees Eq. (4) is also satisfied, since the bounded expression is larger. Moreover, this bound is a subset of $\|\hat{A}_0^{-1}\Delta\| < 1$, a necessary condition for correctness, meaning we don't have to check it explicitly.

B. INFINITE WINDOW CONSTRAINT

A matrix A is *positive definite*, denoted $A \succ 0$, if $x^T A x > 0$ for all non-zero vectors x. This implies a partial ordering of square matrices: we denote $A \succ B$ if $A - B \succ 0$. Note $A \succ B \succ 0 \implies \|A\| > \|B\|$. Moreover, $A \succ B \succ 0 \implies B^{-1} \succ A^{-1} \succ 0$. Finally, observe that $\|(A+B)^{-1}u\| \le \|A^{-1}u\|$, since $A + B \succ A$ and therefore $A^{-1} \succ (A+B)^{-1}$. Similarly, $\|\left((A+B)^{-1} - A^{-1}\right)u\| = \|\left(A^{-1} - (A+B)^{-1}\right)u\| \le \|A^{-1}u\|$.

We apply the above to Eq. (9). Note that by construction, $\Delta^j = \sum_{i\in\mathcal{S}_j} x_i x_i^T$, where $\mathcal{S}_j$ is the set samples seen by node j since the last sync time, is symmetric and positive definite. Similarly, $\hat{A}_0$ is symmetric positive definite by construction. Thus, $E_1 = \|(\hat{A}_0 + \Delta)^{-1}\delta\| \le \|\hat{A}_0^{-1}\delta\|$, and $E_2 = \left\|\left((\hat{A}_0 + \Delta)^{-1} - \hat{A}_0^{-1}\right)\hat{c}_0\right\| \le \|\hat{A}_0^{-1}\hat{c}_0\|$.

The final convex constraint for the infinite window case is therefore

$$\|\hat{A}_0^{-1}\delta\| + \|\hat{A}_0^{-1}\hat{c}_0\| \le \epsilon \quad . \tag{16}$$

C. WINDOW SIZE AND DIMENSIONS

We will show that sliding window W linear in m will avoid overwhelming $\|\hat{A}_0^{-1}\Delta^j\|$ in Eq. (7).

For any matrix A, denote its largest and smallest eigenvalues by $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$. Recall that $\hat{A}_0 = \frac{1}{k}A_0^j$ and that in the sliding window model[4], $\Delta^j = A^j - A_0^j$, and that all these matrices are symmetric by construction. Moreover, if A is symmetric then $\|A\| = \sqrt{\lambda_{\max}(A^T A)} = |\lambda_{\max}(A)|$. Finally, $\lambda_{\max}(A^{-1}) = \frac{1}{\lambda_{\min}(A)}$, and therefore

$$\|\hat{A}_0^{-1}\| = \left\|\frac{1}{k}A_0^{-1}\right\| = \frac{k}{|\lambda_{\min}(A_0)|} = \frac{k}{\lambda_{\min}(A_0)} \quad .$$

Applying the above to the norm constraint:

$$\begin{aligned} \|\hat{A}_0^{-1}\Delta^j\| \le \|\hat{A}_0^{-1}\|\|\Delta^j\| &= \frac{k}{\lambda_{\min}(A_0)}|\lambda_{\max}(A^j - A_0^j)| \\ &\le \frac{k}{\lambda_{\min}(A_0)}|\lambda_{\max}(A^j) - \lambda_{\min}(A_0^j)| \quad . \end{aligned} \tag{17}$$

The last step is obtained from [17]:

$$A, B \text{ symmetric} \implies \lambda_{\max}(A+B) \le \lambda_{\max}(A) + \lambda_{\max}(B)$$

and since $\lambda_{\max}(-B) = -\lambda_{\min}(B)$.

The bound in Eq. (17) depends on the distribution of the data. Assume the elements of X are drawn i.i.d from $N(0,1)$, then the *Marchenku-Pastur law* [22] limits the spectrum of the *Wishart matrix* $X^T X$.

LEMMA 4. *Let $X \in \mathbb{R}^{w\times m}$ drawn as above such that $\frac{m}{W}$ converges to $0 < b \le 1$ as W and m grow to infinity*[5]. *Let $M = \frac{1}{W}X^T X$, and denote its largest and smallest eignevalues by $\lambda_{max}(M)$, $\lambda_{min}(M)$. Then almost surely*

$$\lambda_{max}(M) \to \left(1 + \sqrt{b}\right)^2 \ , \ \lambda_{min}(M) \to \left(1 - \sqrt{b}\right)^2 \quad .$$

Bai and Yin [2] extended this result to *any* zero-mean distribution with unit variance and finite fourth moment [31]. These can be achieved using [5], for example.

Note $A_0 = \sum_1^k A_0^j = \tilde{X_0}^T \tilde{X_0}$, where $\tilde{X_0} \in \mathbb{R}^{kW\times m}$ is the concatenation of all local data matrices. Applying Lemma 4 to Eq. (17), we obtain

$$\begin{aligned} \frac{k|\lambda_{\max}(A^j) - \lambda_{\min}(A_0^j)|}{\lambda_{\min}(A_0)} &= \frac{kW\left|\lambda_{\max}(\frac{1}{W}A^j) - \lambda_{\min}(\frac{1}{W}A_0^j)\right|}{kW\lambda_{\min}(\frac{1}{kW}A_0)} \\ &= \frac{|\lambda_{\max}(\frac{1}{W}A^j) - \lambda_{\min}(\frac{1}{W}A_0^j)|}{\lambda_{\min}(\frac{1}{kW}A_0)} \quad , \end{aligned}$$

which converges almost surely to

$$f_k(b) \triangleq \frac{|(1+\sqrt{b})^2 - (1-\sqrt{b})^2|}{\left(1 - \sqrt{\frac{b}{k}}\right)^2} = \frac{4\sqrt{b}}{\left(1 - \sqrt{\frac{b}{k}}\right)^2} \quad . \tag{18}$$

In practice, Eq. (7) is the sum of 3 norms, so we require $\|\hat{A}_0^{-1}\Delta^j\| < \frac{1}{3}$. Solving $0 < f_k(b) < \frac{1}{3}$ for b with $k > 1$ yields

$$\tfrac{m}{W} \le g_k \triangleq 72k^2 + k - 24k^{\frac{3}{2}} - 4\sqrt{3}\sqrt{108k^4 + 15k^3 - 72k^{\frac{7}{2}} - k^{\frac{5}{2}}}.$$

For given $k > 1$, selecting $W \ge \frac{m}{g_k}$ guarantees $\|\hat{A}_0^{-1}\Delta^j\| < \frac{1}{3}$ almost surely. The constant $\frac{1}{g_k}$ grows slowly: for $k = 2$, the window size W must be at least $\frac{1}{g_2} \approx 111.06m$; for $k = 10$, $W \ge \frac{1}{g_{10}} \approx 129.02m$; and for $k = 100$, $W \ge \frac{1}{g_{100}} \approx 139.22m$. In fact, g_k converges: $lim_{k\to\infty} g_k = \frac{1}{144}$, so a window size of $W \ge 144m$ is sufficient for any k.

[4]We discuss the worst case, when A^j, A_0^j do not overlap. When they do, Δ^j's effective window size is less than W.

[5]Trivially, if $W = \frac{m}{b}$.