EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

Blind distributed estimation algorithms for adaptive networks

Muhammad O Bin Saeed, Azzedine Zerguine* and Salam A Zummo

Abstract

Until recently, a lot of work has been done to develop algorithms that utilize the distributed structure of an ad hoc wireless sensor network to estimate a certain parameter of interest. However, all these algorithms assume that the input regressor data is available to the sensors, but this data is not always available to the sensors. In such cases, blind estimation of the required parameter is needed. This work formulates two newly developed blind block-recursive algorithms based on singular value decomposition (SVD) and Cholesky factorization-based techniques. These adaptive algorithms are then used for blind estimation in a wireless sensor network using diffusion of data among cooperative sensors. Simulation results show that the performance greatly improves over the case where no cooperation among sensors is involved.

Keywords: Blind estimation; Diffusion; Adaptive networks

1 Introduction

This work studies the problem of blind distributed estimation over an ad-hoc wireless sensor network (WSN). WSNs have recently generated a great deal of renewed interest in distributed computing. New research avenues have opened up in the fields of estimation and tracking of parameters of interest, in applications requiring a robust, scalable and low-cost solution. To appreciate such applications, consider a set of N sensor nodes spread over a geographic area as shown in Figure 1. Sensor measurements are taken at each node at every time instant. The objective of the sensor is to estimate a certain unknown parameter of interest using these sensed measurements.

Several algorithms have been devised in the literature for distributed estimation [1-5]. The work in [1] introduces a distributed estimation approach using the recursive least squares algorithm. Other algorithms involving the least-mean-square (LMS) approach have also been suggested [2-5].

However, all these algorithms assume that the input regressor data, $\mathbf{u}_{k,i}$, is available at the sensors. If this information is not available, then the said problem becomes a blind estimation problem. Blind algorithms have been a topic of interest ever since Sato devised a blind algorithm [6] in the context of equalization [7]. Since then, several algorithms have been derived for blind estimation [8-15]. The work in [8] summarizes the second-order statistics-based approaches for blind identification. These include multichannel as well as single-channel blind estimation methods such as the works in [9] and [10]. The work in [11] is one of the most cited blind estimation techniques for a single-input-single-output (SISO) model. However, unlike in [11], it is shown in [12] that the technique of [11] can be improved upon using only two blocks of data. A key idea in [12] is used in [13] to devise an algorithm that does indeed show improvement over the algorithm of [11]. However, the computational complexity of this new algorithm (in [13]) is very demanding. A generalized algorithm is devised in [14], improving upon both algorithms developed in [12,13]. In [15], a Cholesky factorization-based least squares solution is suggested that simplifies the work of [11,13,14]. Although the performance of the algorithm developed in [15] is not as good as that of the previous algorithms, it nevertheless provides an excellent trade-off between performance level and computational complexity. However, in systems where less complexity is required and performance can be compromised to some extent, this algorithm would provide a good substitute to the algorithms developed in [12,13].

*Correspondence: azzedine@kfupm.edu.sa
Electrical Engineering Department, King Fahd University of Petroleum & Minerals, Dhahran 31261, Saudi Arabia

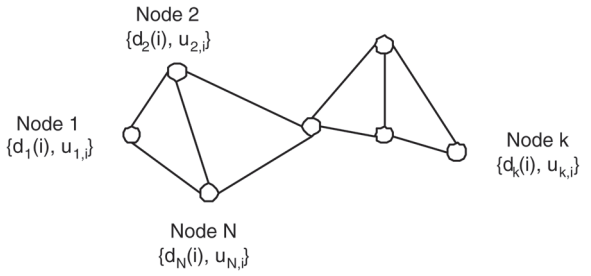

Figure 1 Adaptive network of *N* nodes.

As mentioned above, for the case where the input regressor data is not available to the WSN environment used, then blind estimation techniques become mandatory. In this case, since blind estimation techniques have not yet been developed for this field, blind block-recursive least squares algorithms would have to be devised, inspired from the works in [11] and [15], and then implemented in a distributed WSN environment using the diffusion approach suggested in [1].

The following notation has been used here. Boldface letters are used for vectors/matrices and normal font for scalar quantities. Matrices are defined by capital letters and small letters are used for vectors. The notation $(.)^T$ stands for transposition for vectors and matrices and expectation operation is denoted by $E[.]$. Any other mathematical operators used in this paper will be defined as and when introduced in the paper.

The paper is divided as follows: Section 2 defines the problem statement. Section 3 gives a brief overview of the blind estimation algorithms taken into consideration in this work. Section 4 proposes the newly developed recursive forms of the two algorithms, as well as their diffusion counterparts, to be used in wireless sensor networks. Section 5 studies the computational complexity of all the algorithms. The simulation results are discussed in detail in section 6. Finally, the paper is concluded in section 7.

2 Problem statement

Consider a network of K sensor nodes deployed over a geographical area, to estimate an $(M \times 1)$ unknown parameter vector $\mathbf{w^o}$ as shown in Figure 1. Each node k has access to a time realization of a scalar measurement $d_k(i)$ that is given by

$$d_k(i) = \mathbf{u}_{k,i}\mathbf{w}^o + \nu_k(i), \qquad 1 \le k \le K \tag{1}$$

where $\mathbf{u}_{k,i}$ is a $(1 \times M)$ input regressor vector, ν_k is a spatially uncorrelated zero-mean additive white Gaussian noise with variance $\sigma^2_{\nu_k}$ and i denotes the time index. The input data is assumed to be Gaussian. The aim of this work is to estimate the unknown vector $\mathbf{w^o}$ using the sensed data $d_k(i)$ without knowledge of the input regressor vector. The estimate of the unknown vector can be denoted by an $(M \times 1)$ vector $\mathbf{w}_{k,i}$. Assuming that each node k cooperates only with its neighbors and k has access to updates $\mathbf{w}_{l,i}$, from its $\mathcal{K}_k$ neighboring nodes at every time instant i, where $\{l \in \mathcal{K}_k, l \neq k\}$, in addition to its own estimate, $\mathbf{w}_{k,i}$. The adapt-then-combine (ATC) diffusion scheme [16] first updates the local estimate at each node using the adaptive algorithm and then fuses together the estimates from the $\mathcal{K}_k$ neighboring nodes. This scheme will be used in this work for the development of our distributed algorithm. Note that, even though this work is designed for a fixed topology, it can be extended to a dynamic one.

3 Blind estimation algorithm

In this work, the input regressor data, $\mathbf{u}_k(i)$ is assumed to be not available to the sensors and the unknown vector $\mathbf{w}^o$ is estimated using only the sensed values, $d_k(i)$. Since the data considered here is Gaussian, a method using second-order statistics only is sufficient for such an estimation problem as it will capture all the required data statistics. Even for the case of non-Gaussian data, such an approach would provide a suboptimal yet accurate enough estimate. The work in [11] uses the second-order statistics in an intelligent manner to create a null space with respect to the unknown vector $\mathbf{w}^o$. At the receiver end, this null space is then exploited to estimate the unknown vector. The authors in [15] further simplify the algorithm of [11] by proposing a new algorithm that reduces complexity but at a cost of performance degradation. These two algorithms are taken into consideration in this work as one provides excellent results whereas the other provides a computationally tractable solution.

3.1 Singular value decomposition-based blind algorithm

The work in [11] uses redundant filterbank precoding to construct data blocks that have trailing zeros. These data blocks are then collected at the receiver and used for blind channel identification. In this work, however, there is no precoding required. The trailing zeros will still be used though, for estimation purposes. Let the unknown parameter vector be of size $(M \times 1)$. Suppose the input vector is a $(P \times 1)$ vector with $(P - M)$ trailing zeros

$$\mathbf{s}_k(i) = \left\{s_{k,0}(i), s_{k,1}(i), \ldots, s_{k,M-1}(i), 0, \ldots, 0\right\}^T, \tag{2}$$

where P and M are related through $P = 2M - 1$. The unknown parameter vector can be written in the form of a convolution matrix given by

$$\mathbf{W} = \begin{bmatrix} w(0) & \cdots & w(M-1) & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & w(0) & \cdots & w(M-1) \end{bmatrix}^T, \tag{3}$$

where $\mathbf{w^o} = [\mathbf{w}(\mathbf{0}), \mathbf{w}(\mathbf{1}), \ldots, \mathbf{w}(\mathbf{M-1})]$ is the unknown parameter vector. The output data block can now be written as

$$\mathbf{d}_k(i) = \mathbf{W}\mathbf{s}_k(i) + \mathbf{v}_k(i), \tag{4}$$

where $\mathbf{d}_k(i)$ is the $((2M-1)\times 1)$ output data block and $\mathbf{v}_k$ is the added noise. The output blocks $\{\mathbf{d}_k(i)\}$ are collected together to form the following matrix

$$\mathbf{D}_{k,N} = (\mathbf{d}_k(0), \mathbf{d}_k(1), \ldots, \mathbf{d}_k(N-1)), \tag{5}$$

where N is greater than the minimum number of data blocks required for the input blocks to have a full rank. The singular value decomposition (SVD) of the autocorrelation of $\mathbf{D}_{k,N}$ gives a set of null eigenvalues. Thus, the eigendecomposition

$$\mathbf{D}_{k,N}\mathbf{D}_{k,N}^T = \begin{pmatrix} \bar{\mathbf{U}}_k & \tilde{\mathbf{U}}_k \end{pmatrix} \begin{pmatrix} \Sigma_{M\times M} & \mathbf{0}_{M\times(M-1)} \\ \mathbf{0}_{(M-1)\times M} & \mathbf{0}_{(M-1)\times(M-1)} \end{pmatrix} \times \begin{pmatrix} \bar{\mathbf{U}}_k^T \\ \tilde{\mathbf{U}}_k^T \end{pmatrix}, \tag{6}$$

where $\bar{\mathbf{U}}_k$ is the $(P\times M)$ matrix of eigenvectors and $\tilde{\mathbf{U}}_k$ is a $(P\times(M-1))$ matrix whose columns form the null space for $\mathbf{D}_{k,N}$. This implies

$$\tilde{\mathbf{U}}_k^T\mathbf{W} = \mathbf{0}, \tag{7}$$

which can also be written as

$$\tilde{\mathbf{u}}_{k,m}^T\mathbf{W} = \mathbf{0}^T, \tag{8}$$

where $m = 1, \ldots, M-1$ and $\tilde{\mathbf{u}}_{k,m}$ is simply the m_{th} vector of $\tilde{\mathbf{U}}_k$. This equation denotes convolution since, as pointed earlier, $\mathbf{W}$ is essentially a convolution matrix. Since convolution is a commutative operation, Equation 8 can also be written as

$$\mathbf{w}^T\mathcal{U}_k := \mathbf{w}^T\left(\mathcal{U}_{k,1}\ldots\mathcal{U}_{k,M-1}\right) = \mathbf{0}^T, \tag{9}$$

where $\mathbf{w}$ is a component vector of the convolution matrix $\mathbf{W}$ and $\mathcal{U}_{k,m}$ is an $(M\times M)$ Hankel matrix given by

$$\mathcal{U}_{k,m} = \begin{bmatrix} \tilde{\mathbf{u}}_{k,m}(0) & \tilde{\mathbf{u}}_{k,m}(1) & \cdots & \tilde{\mathbf{u}}_{k,m}(P-M) \\ \tilde{\mathbf{u}}_{k,m}(1) & \tilde{\mathbf{u}}_{k,m}(2) & \cdots & \tilde{\mathbf{u}}_{k,m}(P-M+1) \\ \vdots & \vdots & \vdots & \vdots \\ \tilde{\mathbf{u}}_{k,m}(M-1) & \tilde{\mathbf{u}}_{k,m}(M) & \cdots & \tilde{\mathbf{u}}_{k,m}(P-1) \end{bmatrix}. \tag{10}$$

The final parameter estimate is given by the unique solution (up to a constant factor) of Equation 9. It is important to note here that due to the presence of noise, the final estimate is not accurate.

3.2 Cholesky factorization-based blind algorithm

The work in [15] describes a method that replaces the SVD operation with the Cholesky factorization operation to blindly estimate the channel. Again, the received block data matrix can be written as (5). Taking the autocorrelation of $\mathbf{D_{k,N}}$ and assuming the input data regressors to be white Gaussian with variance $\sigma^2_{\mathbf{s},k}$, we get

$$\mathbf{R}_{\mathbf{d},k} = \mathrm{E}\left[\mathbf{D}_N\mathbf{D}_N^T\right] = \sigma^2_{\mathbf{s},k}\mathbf{W}\mathbf{W}^T + \sigma^2_{\mathbf{v},k}\mathbf{I}. \tag{11}$$

Now if the second-order statistics of both the input regressor data as well as the additive noise are known, then the correlation matrix for the unknown vector can be written as

$$\mathbf{R}_{\mathbf{w}} = \mathbf{W}\mathbf{W}^T = \left(\mathbf{R}_{\mathbf{d},k} - \sigma^2_{\mathbf{v},k}\mathbf{I}\right)/\sigma^2_{\mathbf{s},k}. \tag{12}$$

However, this information, particularly the information about the input regressor data, is not always known and cannot be easily estimated either. Therefore, the correlation matrix of the unknown parameter vector has to be approximated by the correlation matrix of the received/sensed data. Now the algorithm in [15] uses the Cholesky factor of this correlation matrix to provide a least squares estimate of the unknown parameter vector.

The method given in [15] is summarized here. Since the correlation matrix is not available at the receiver, an approximate matrix is calculated using K blocks of data. So the correlation matrix is given by

$$\hat{\mathbf{R}}_{\mathbf{d},k} = \frac{1}{K}\sum_{i=1}^{K}\mathbf{d}_k(i)\mathbf{d}_k^T(i). \tag{13}$$

As the second-order statistics of the noise are not known, the noise variance is estimated and then subtracted from the data correlation matrix. Thus, we have

$$\hat{\mathbf{R}}_{\mathbf{w},k} = \hat{\mathbf{R}}_{\mathbf{d}_k} - \hat{\sigma}^2_{\mathbf{v},k}\mathbf{I}_K = \frac{1}{K}\sum_{i=1}^{K}\mathbf{d}_k(i)\mathbf{d}_k^T(i) - \hat{\sigma}^2_{\mathbf{v},k}\mathbf{I}_K. \tag{14}$$

Taking the Cholesky factor of this matrix gives us the upper triangular matrix $\hat{\mathbf{G}}_k$

$$\hat{\mathbf{G}}_k = \text{chol}\left\{\hat{\mathbf{R}}_{\mathbf{w},k}\right\}. \tag{15}$$

Next, we use the *vec* operator to get an $(M^2 \times 1)$ vector $\hat{\mathbf{g}}_k$

$$\hat{\mathbf{g}}_k = \text{vec}\left\{\hat{\mathbf{G}}_k\right\}. \tag{16}$$

It is given in [15] that the vectors $\mathbf{g}$ and $\mathbf{w}^o$ are related through

$$\mathbf{g} = \mathbf{Q}\mathbf{w}^o, \tag{17}$$

where $\mathbf{Q}$ is an $(M^2 \times M)$ selection matrix given by $\mathbf{Q} = \left[\mathbf{J}_1^T \mathbf{J}_2^T \ldots \mathbf{J}_M^T\right]^T$, and the $(M \times M)$ matrices $\mathbf{J}_q$ are defined as

$$\mathbf{J}_c = \begin{cases} 1, & \text{if } a+b = c-1 \\ 0, & \text{otherwise,} \end{cases} \tag{18}$$

where $\{a, b, c\} = 0, \ldots, M-1$. So the least squares estimate of the unknown parameter vector is given by [15]

$$\hat{\mathbf{w}}_k = \left(\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T\hat{\mathbf{g}}_k. \tag{19}$$

The work in [15] also gives a method for estimating the noise variance that is on the whole adequate except it may not provide correct estimates of the noise variance at low SNRs. As a result, subtracting the estimated variance from the autocorrelation matrix may not yield a positive-definite matrix. In such cases, the use of Cholesky factorization may not be justified. However, neglecting the noise variance estimate altogether may lead to a poor estimate of the parameter vector. Despite this shortcoming, the main advantage of this method remains its very low computational complexity. Whereas the method of [11] requires the singular value decomposition of the autocorrelation matrix followed by the building of Hankel matrices using the null eigenvectors and then finding a unique solution to an over-determined set of linear equation, this method [15] simply evaluates the Cholesky factor (upper triangular matrix) of the autocorrelation matrix and then uses it to directly find the required estimate. Computational complexity is, thus, greatly reduced but at the cost of a performance degradation.

Both of the above-mentioned methods require several blocks of data to be stored before estimation can be performed. Although the least squares approximation gives a good estimate, a sensor network requires an algorithm that can be deployed in a distributed manner, which is possible only with recursive algorithms. Therefore, the first step would be to make both algorithms in [11] and [15] recursive in order to utilize them in a WSN setup.

4 Proposed recursive blind estimation algorithms

In the ensuing, the previously mentioned blind estimation algorithms are made recursive and applied over a wireless sensor network.

4.1 Blind block recursive SVD algorithm

Here, we show how the algorithm from [11] can be made into a blind block-recursive algorithm. Since the algorithm requires a complete block of data at each processing instant, we therefore base our iterative process on data blocks as well. So instead of the matrix $\mathbf{D}_k$, we have the block data vector $\mathbf{d}_k$. The autocorrelation matrix for the first data block is defined in as

$$\hat{\mathbf{R}}_{\mathbf{d},k}(1) = \mathbf{d}_k(1)\mathbf{d}_k^T(1). \tag{20}$$

The matrix is expanded for two blocks in the original algorithm as

$$\begin{aligned} \hat{\mathbf{R}}_{\mathbf{d},k}(2) &= \mathbf{D}_{k,2}\mathbf{D}_{k,2}^T \\ &= \begin{bmatrix} \mathbf{d}_k(1) & \mathbf{d}_k(2) \end{bmatrix}\begin{bmatrix} \mathbf{d}_k^T(1) \\ \mathbf{d}_k^T(2) \end{bmatrix} \\ &= \mathbf{d}_k(1)\mathbf{d}_k^T(1) + \mathbf{d}_k(2)\mathbf{d}_k^T(2) \\ &= \hat{\mathbf{R}}_{\mathbf{d},k}(1) + \mathbf{d}_k(2)\mathbf{d}_k^T(2). \end{aligned} \tag{21}$$

Thus, a generalization of (21) can be written as

$$\hat{\mathbf{R}}_{\mathbf{d},k}(i) = \hat{\mathbf{R}}_{\mathbf{d},k}(i-1) + \mathbf{d}_k(i)\mathbf{d}_k^T(i). \tag{22}$$

The first few iterations may not give a good estimate and the error may even seem to be increasing as the matrix will be rank deficient at this early stage. However, as more data blocks are processed, the rank becomes gradually full and the estimate then begins to gradually improve. The next step is to get the eigendecomposition of the autocorrelation matrix. Applying the SVD on $\mathbf{R}_{\mathbf{d},k}$ yields the eigenvector matrix $\mathbf{U}_k$, from which we get the $(M-1 \times M)$ matrix $\tilde{\mathbf{U}}_k$ that forms the null space of the autocorrelation matrix. From $\tilde{\mathbf{U}}_k$, we then form the M Hankel matrices of size $(M \times M+1)$ each, which are then concatenated to give the matrix $\mathcal{U}_k(i)$ from which the estimate $\tilde{\mathbf{w}}_k(i)$ is finally derived. This sequential derivation process is depicted below in (23):

$$\text{SVD}\left\{\mathbf{R}_{\mathbf{d},k}(i)\right\} \Rightarrow \mathbf{U}_k(i) \Rightarrow \tilde{\mathbf{U}}_k(i) \Rightarrow \mathcal{U}_k(i) \Rightarrow \tilde{\mathbf{w}}_k(i). \tag{23}$$

The update for the estimate of the unknown parameter vector is then given by

$$\hat{\mathbf{w}}_k(i) = \lambda_k\hat{\mathbf{w}}_k(i-1) + (1-\lambda_k)\,\tilde{\mathbf{w}}_k(i), \tag{24}$$

It can be seen from (23) that the recursive algorithm does not become computationally less complex. However, it does require lesser memory compared to the original algorithm of [11] and the result improves with an increase

in the number of data blocks processed. The performance almost matches that of the algorithm of [11].

Algorithm 1 describes the steps of the blind block recursive SVD (RS) algorithm. The forgetting factor is fixed in this case. If the forgetting factor value were to be changed to $\lambda_k(i) = 1 - \frac{1}{i}$, the algorithm would then become the variable forgetting factor RS (VFFRS) algorithm. However, simulations show that the VFFRS algorithm converges more slowly than its fixed forgetting factor counterpart. The simulation results show that if the forgetting factor is small for the fixed forgetting factor case, the algorithm converges faster even though it gives a higher error floor at steady-state. While for the VFFRS algorithm, the forgetting factor increases with time resulting in slow convergence even though the steady-state error is lower compared to the fixed forgetting factor case.

Algorithm 1 Summary of blind block recursive SVD algorithm

Step 1. Form auto-correlation matrix for iteration i from equation (22).
Step 2. Get $\mathbf{U}_k(i)$ from SVD of $\hat{\mathbf{R}}_{\mathbf{d},k}(i)$.
Step 3. Form $\tilde{\mathbf{U}}_k(i)$ from the null eigenvectors of $\mathbf{U}_k(i)$.
Step 4. Form Hankel matrices of size $(L \times M-1)$ from individual vectors of $\tilde{\mathbf{U}}_k(i)$.
Step 5. Form $\mathcal{U}_k(i)$ by concatenating the Hankel matrices.
Step 6. The null eigenvector from the SVD of $\mathcal{U}_k(i)$ is the estimate $\tilde{\mathbf{w}}_k(i)$.
Step 7. Use $\tilde{\mathbf{w}}_k(i)$ in equation (24) to get the update $\hat{\mathbf{w}}_k(i)$.

4.2 Blind block recursive Cholesky algorithm

In this section, we show how the algorithm of [15] can be converted into a blind block recursive solution.

Equation 14 can be rewritten as

$$\begin{aligned}\hat{\mathbf{R}}_{\mathbf{w},k}(i) &= \frac{1}{i}\sum_{n=1}^{i}\mathbf{d}_k(n)\mathbf{d}_k^T(n) - \hat{\sigma}^2_{\mathbf{v},k}\mathbf{I}_K \qquad (25)\\ &= \frac{1}{i}\mathbf{d}_k(i)\mathbf{d}_k^T(i) + \frac{1}{i}\sum_{n=1}^{i-1}\mathbf{d}_k(n)\mathbf{d}_k^T(n) - \hat{\sigma}^2_{\mathbf{v},k}\mathbf{I}_K\\ &= \frac{1}{i}\left(\mathbf{d}_k(i)\mathbf{d}_k^T(i) - \hat{\sigma}^2_{\mathbf{v},k}\mathbf{I}_K\right) + \frac{i-1}{i}\hat{\mathbf{R}}_{\mathbf{w},k}(i-1).\end{aligned}$$

Similarly, we have

$$\hat{\mathbf{G}}_k(i) = chol\left\{\hat{\mathbf{R}}_{\mathbf{w},k}(i)\right\}, \qquad (26)$$

$$\hat{\mathbf{g}}_k(i) = vec\left\{\hat{\mathbf{G}}_k(i)\right\}. \qquad (27)$$

Letting $\mathbf{Q}_A = \left(\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T$, we have

$$\bar{\mathbf{w}}_k(i) = \mathbf{Q}_A\hat{\mathbf{g}}_k(i). \qquad (28)$$

We further apply a smoothing step to get the final estimate:

$$\hat{\mathbf{w}}_k(i) = \lambda_k(i)\hat{\mathbf{w}}_k(i-1) + (1-\lambda_k(i))\,\bar{\mathbf{w}}_k(i), \qquad (29)$$

where $\lambda_k(i) = 1 - \frac{1}{i}$ is a variable forgetting factor.

Letting $\lambda_k(i) = 1 - \frac{1}{i}$, the blind block recursive Cholesky algorithm is summarised in Algorithm 2. This table defines the Blind Block Recursive Cholesky algorithm with variable forgetting factor (VFFRC). If the forgetting factor is fixed then the algorithm can simply be called Blind Block Recursive Cholesky (RC) algorithm. Simulation results show that the VFFRC algorithm converges to the least squares solution obtained through the algorithm given in [15]. The RC algorithm can also achieve the same result if the value of the forgetting factor is extremely close to 1. However, the convergence speed of the RC algorithm is slower than that of the VFFRC algorithm even though it requires lesser memory and is slightly less computationally complex. There are two issues with the recursive algorithm. Firstly, the Cholesky factorization cannot be applied until at least M blocks of data have been received as the data correlation matrix needs to be first positive definite before the Cholesky method can be correctly applied. The second issue involves the variance of the additive noise. In [15], it is shown that if the noise variance can be estimated, the estimate of the unknown vector will improve. However, using the noise variance in the recursive algorithm can make the resulting matrix have zero or negative eigenvalues before a sufficient number of data blocks were processed, thus making the use of Cholesky factorization unjustifiable. However, neglecting the noise variance altogether will lead to a performance degradation of this algorithm even though it will be computationally less complex than the SVD approach. One approach is to estimate the noise variance after a certain number of blocks have been received and then use that value for the remainder of the iterations.

Algorithm 2 Summary of blind block recursive Cholesky (RC) algorithm

Step 1. Let forgetting factor be defined as $\lambda_k(i) = 1 - \frac{1}{i}$.
Step 2. Form auto-correlation matrix for iteration i using $\lambda_k(i)$ in equation (25) to get

$$\hat{\mathbf{R}}_{\mathbf{w},k}(i) = (1-\lambda_k(i))\left(\mathbf{d}_k(i)\mathbf{d}_k^T(i) - \hat{\sigma}^2_{\mathbf{v},k}\mathrm{I}_K\right) + \lambda_k(i)\,\hat{\mathbf{R}}_{\mathbf{w},k}(i-1)$$

Step 3. Get $\hat{\mathbf{G}}(i)$ as the Cholesky factor of $\hat{\mathbf{R}}_{\mathbf{w},k}(i)$.
Step 4. Apply the *vec* operator to get $\hat{\mathbf{g}}_k(i)$.
Step 5. Use $\lambda_k(i)$ in equation (29) to get the final update

$$\hat{\mathbf{w}}_k(i) = \mathbf{Q}_A\left(\hat{\mathbf{g}}_k(i) - \lambda_i\hat{\mathbf{g}}_k(i-1)\right) + \lambda_k(i)\hat{\mathbf{w}}_k(i-1).$$

4.3 Diffusion blind block recursive algorithms

In a wireless sensor network, a distributed algorithm is required, through which nodes can interact with each other and improve their individual estimates as well as the overall performance of the network. In such a scenario, a recursive algorithm is required. This is one major reason for requiring a recursive blind algorithm. Each node can individually update its estimate and then collaborate with the neighboring nodes to improve that estimate. A comparison of different distributed schemes has shown that the *Adapt-Then-Combine* (ATC) diffusion strategy provides the best performance [16]. Therefore, we also implement our distributed algorithms using the ATC scheme.

For the diffusion-based RS algorithm, all nodes evaluate their own autocorrelation matrix updates and then perform the SVD operation. This is followed by a preliminary estimate of the unknown vector. The preliminary results are then combined with those of the neighboring nodes. As a result of this cooperation, the result of the network improves, as will be shown in the simulations. Similarly, for the diffusion RC algorithm, each node performs the recursion and reaches a preliminary estimate of the unknown vector which is then combined with those of the neighboring nodes. These are summarized in Algorithms 3 and 4, where the subscript k denotes the node number, N_k is the set of neighbors of node k, $\hat{\mathbf{h}}_k$ is the intermediate estimate for node k and c_{lk} is the weight connecting node k to its neighboring node $l \in \mathcal{N}_k$, where $\mathcal{N}_k$ includes node k, and $\sum c_{lk} = 1$.

Algorithm 3 Summary of diffusion blind block recursive SVD algorithm

Step 1. Form auto-correlation matrix for iteration i from equation (22) for each node k.

$$\hat{\mathbf{R}}_{\mathbf{d},k}(i) = \mathbf{d}_{k,i}\mathbf{d}_{k,i}^T + \hat{\mathbf{R}}_{\mathbf{d},k}(i-1)$$

Step 2. Get $\mathbf{U}_k(i)$ from SVD of $\hat{\mathbf{R}}_{\mathbf{d},k}(i)$.
Step 3. Form $\tilde{\mathbf{U}}_k(i)$ from the null eigenvectors of $\mathbf{U}_k(i)$.
Step 4. Form Hankel matrices of size $(L \times M-1)$ from individual vectors of $\tilde{\mathbf{U}}_k(i)$.
Step 5. Form $\mathcal{U}_k(i)$ by concatenating the Hankel matrices.
Step 6. The null eigenvector from the SVD of $\mathcal{U}_k(i)$ is the estimate $\tilde{\mathbf{w}}_{k,i}$.
Step 7. Use $\tilde{\mathbf{w}}_{k,i}$ in equation (24) to get the intermediate update $\hat{\mathbf{h}}_{k,i}$.

$$\hat{\mathbf{h}}_{k,i} = \lambda\hat{\mathbf{w}}_{k,i-1} + (1-\lambda)\,\tilde{\mathbf{w}}_{k,i}$$

Step 8. Combine estimates from neighbors of node k to get $\hat{\mathbf{w}}_{k,i}$.

$$\hat{\mathbf{w}}_{k,i} = \sum_{l\varepsilon N_k} c_{lk}\hat{\mathbf{h}}_{l,i}$$

Algorithm 4 Summary of diffusion blind block recursive Cholesky algorithm

Step 1. Let forgetting factor be defined as $\lambda_{k,i} = 1 - \frac{1}{i}$.
Step 2. Form auto-correlation matrix for iteration k from

$$\hat{\mathbf{R}}_{\mathbf{w},k}(i) = (1-\lambda_{k,i})\left(\mathbf{d}_{k,i}\mathbf{d}_{k,i}^T - \hat{\sigma}_{v,k}^2\mathrm{I}_K\right) + \lambda_{k,i}\hat{\mathbf{R}}_{\mathbf{w},k}(i-1)$$

Step 3. Get $\hat{\mathbf{G}}_k(i)$ as the Cholesky factor of $\hat{\mathbf{R}}_{\mathbf{w},k}(i)$.
Step 4. Apply the *vec* operator to get $\hat{\mathbf{g}}_{k,i}$.
Step 5. The intermediate update is then given as

$$\hat{\mathbf{h}}_{k,i} = \mathbf{Q}_A\left(\hat{\mathbf{g}}_{k,i} - \lambda_{k,i}\hat{\mathbf{g}}_{k,i-1}\right) + \lambda_{k,i}\hat{\mathbf{w}}_{k,i-1}.$$

Step 6. The final update is the weighted sum of the estimates of all neighbors of node k

$$\hat{\mathbf{w}}_{k,i} = \sum_{l\varepsilon N_k} c_{lk}\hat{\mathbf{h}}_{l,i}$$

5 Complexity of the recursive algorithms

In order to fully understand the variation in performance of these two algorithms, it is necessary to look at their computational complexity as it will allow us to estimate the loss in performance that would result from a reduction in computational load. We first analyze the complexity of the original algorithms and then deal with that of their recursive versions.

5.1 Blind SVD algorithm

The length of the unknown parameter vector is M and the data block size is K. Since a total number of N data blocks are required for the estimation of the unknown parameter vector, where $N \geq K$, the resulting data matrix will therefore be of size $K \times N$. The data correlation matrix will thus be of size $K \times K$ and this function will require $K^2(2N-1)$ calculations (including both multiplications and additions) for its computation. The next step is singular value decomposition (SVD), done using the QR decomposition algorithm. This algorithm requires a total of $\left[\frac{4}{3}K^3 + \frac{3}{2}K^2 + \frac{19}{6}K - 6\right]$ calculations. Then the null eigenvectors are separated and each eigenvector is used to form a Hankel matrix with all the Hankel matrices then stacked together to form a matrix of size $M \times (K-M)(M-1)$. The unique null vector of this new matrix gives the estimate of the unknown vector. To find this eigenvector requires another $\left[\left(2K+\frac{7}{3}\right)M^3 - 2M^4 + (1-4K)\frac{M^2}{2} + \frac{19}{6}M - 6\right]$ calculations. So the overall computational load required for the algorithm can be given as

$$T_{C,\mathrm{SVD}} = \frac{4}{3}K^3 + \left(2N+\frac{1}{2}\right)K^2 + \frac{19}{6}(K+M) + \left(2K+\frac{7}{3}\right)M^3 - 2M^4 + (1-4K)\frac{M^2}{2} - 12. \quad (30)$$

5.2 Blind Cholesky algorithm

Like the SVD algorithm, here also the unknown vector length is M and the data block size is K. For computational purposes and for the blind SVD algorithm, the total number of data blocks is taken as N. The correlation process is the same except for the final averaging step which results in an extra division so the total number of calculations becomes $K^2(2N-1)+1$. The next step is to estimate the noise variance, which requires an SVD decomposition and therefore an extra number of calculations given by $\left[\frac{4}{3}K^3+\frac{3}{2}K^2+\frac{19}{6}K-6\right]$. After the SVD decomposition, only one further division is required to estimate the noise variance. The noise variance is then subtracted from the diagonal of the correlation matrix, resulting in another K calculations. After that, the Cholesky factorization is performed, which requires $\left[\frac{1}{3}\left(M^3+3M^2+M\right)\right]$ calculations. Finally, the last step is to get the estimate of the unknown vector through the pseudo-inverse of Cholesky-factorized data correlation matrix and this step requires further $\left[M\left(2M^2-1\right)\right]$ calculations. Thus, the total number of calculations required is given as

$$T_{C,\mathrm{Chol}}=\frac{4}{3}K^3+\left(2N+\frac{1}{2}\right)K^2+\frac{19}{6}K-4+\frac{1}{3}\left(7M^3+3M^2-M\right). \tag{31}$$

5.3 Blind block recursive SVD algorithm

When the blind SVD algorithm is made recursive, we notice that this will involve only a slight change in the overall algorithm but will really halve the total computational load. Since the correlation matrix is only being updated at each iteration, the number of calculations required for the first step are now only $2K^2$ instead of $K^2(2N-1)$. However, an extra $(M+2)$ calculations are required for the final step. The overall number of calculations is thus given as

$$T_{C,\mathrm{SRLS}}=\frac{4}{3}K^3+\frac{7}{2}K^2+\frac{19}{6}K+\left(2K+\frac{7}{3}\right)M^3-2M^4+(1-4K)\frac{M^2}{2}+\frac{25}{6}M-10=O\left(K^3\right). \tag{32}$$

Table 1 Computations for original least squares algorithms under different settings

$M=4$	$N=10$ $K=8$	$N=10$ $K=10$	$N=20$ $K=8$	$N=20$ $K=10$	$N=20$ $K=20$
SVD	2,434	4,021	3,714	6,021	28,496
Chol	2,180	3,575	3,460	5,575	27,090

Table 2 Computations for recursive algorithms under different settings

$M=4$	$K=8$	$K=10$	$K=20$
RS	1,352	2,327	13,702
RC	1,100	1,883	12,298
RCNV	300	372	972

5.4 Blind block recursive Cholesky algorithm

Similarly, the number of calculations for the first step for this algorithm is reduced to $(2K^2+2)$ from the $(K^2(2N-1)+1)$ calculations required by its non-recursive counterpart. The final step includes an extra (K^2+M+2) calculations. Thus, the total number of calculations is now given as

$$T_{C,\mathrm{CRLS}}=\frac{4}{3}K^3+\frac{7}{2}K^2+\frac{19}{6}K+\frac{1}{3}\left(7M^3+3M^2+2M\right)=O\left(K^3\right). \tag{33}$$

However, it should be noted here that the estimation of the noise variance need not be repeated at each iteration. After a few iterations, the number of which can be fixed *a priori*, the noise variance can be estimated once only and then this same estimated value can be used in the

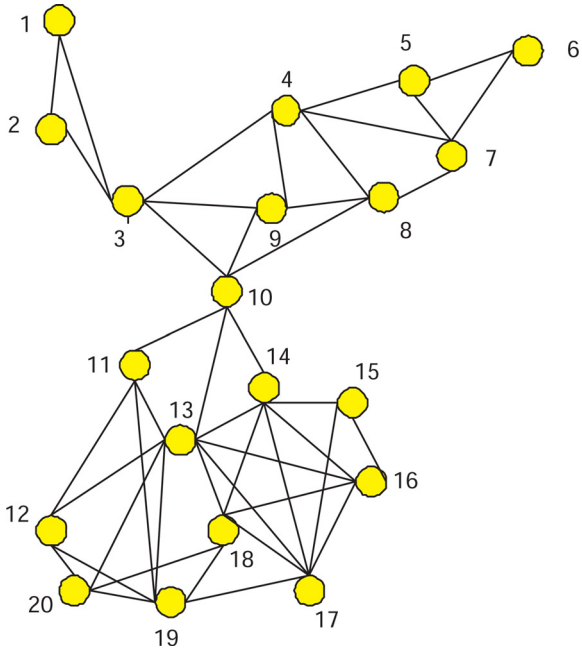

Figure 2 Network of 20 nodes.

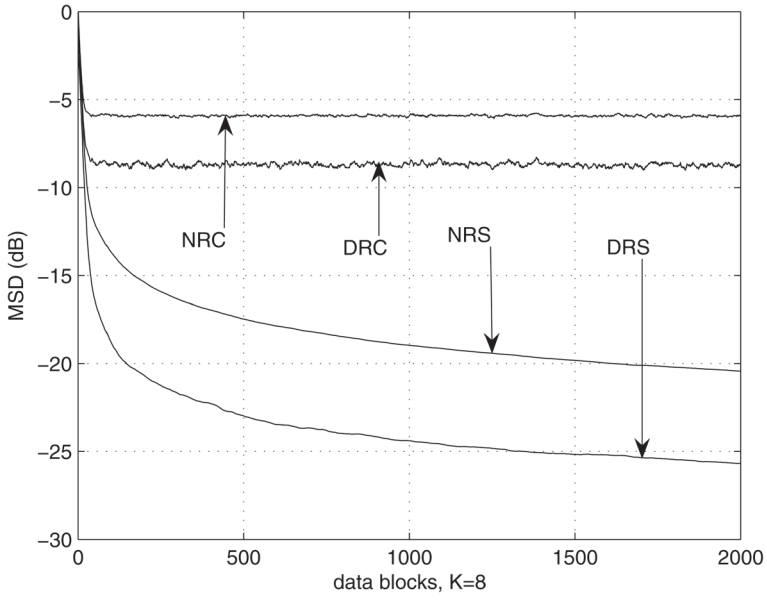

Figure 3 **MSD at SNR = 10 dB and λ = 0.9.**

remaining iterations. The number of calculations, thus, reduces to

$$T_{C,\mathrm{CRLS}} = 2K^2 + \frac{1}{3}\left(7M^3 + 3M^2 + 2M\right) + 4 = O\left(K^2\right). \tag{34}$$

5.5 Comparison of all algorithms

Here, we compare all of the algorithms discussed in the previous sections, using specific scenarios where the value for M is fixed to 4, that of K is varied for all algorithms and the value of N is varied between 10 and 20 for the least squares algorithms. The number of calculations for the two recursive algorithms discussed before are shown for one iteration only. Recall that in the second algorithm, i.e. the blind block recursive Cholesky algorithm, the noise variance is calculated only once, after a pre-selected number of iterations have occurred, and then kept constant for the remaining iterations. Tables 1 and 2 below summarize the results.

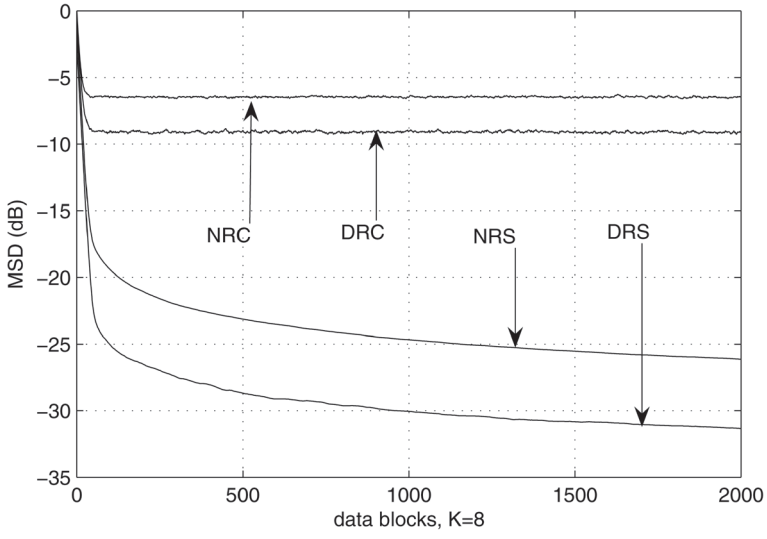

Figure 4 **MSD at SNR = 20 dB and λ = 0.9.**

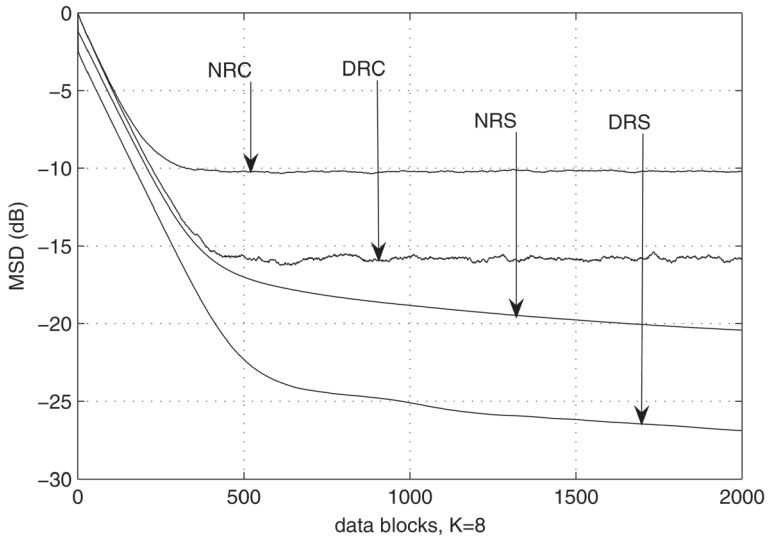

Figure 5 MSD at SNR = 10 dB and $\lambda = 0.99$.

Table 1 lists the number of computations for the original algorithms, showing that the Cholesky-based method requires fewer computations than the SVD-based method and so the trade-off between performance and complexity is justified. If the number of blocks is small, then the Cholesky-based method may even perform better than the SVD-based method as shown in [15]. Here it is assumed that the exact length of the unknown vector is known. Generally, an upper bound of this value is known and that value is used instead of the exact value, resulting in an increase in computations. This assumption is made for both algorithms here to make their comparative study fair.

Table 2 lists the computations-per-iteration for the recursive versions of these two algorithms. RS and RC give the number of computations for the recursive SVD algorithm and the recursive Cholesky algorithm respectively. RCNV lists the number of computations when the noise variance is estimated only once in the recursive Cholesky algorithm. This shows how the complexity of the algorithm can be reduced by an order of magnitude by adopting an extra implicit assumption regarding the

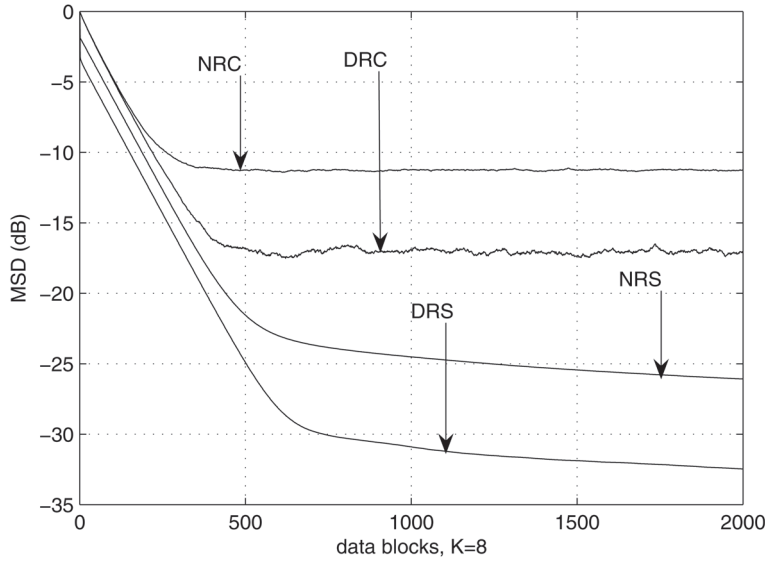

Figure 6 MSD at SNR = 20 dB and $\lambda = 0.99$.

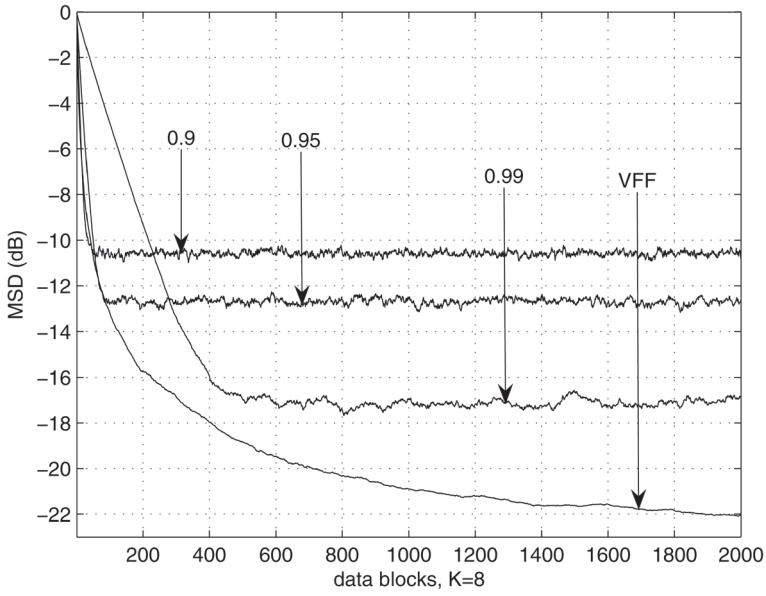

Figure 7 MSD at SNR = 20 dB for RC with different forgetting factors.

wide-sense stationarity of the noise and hence the constancy of its variance from one iteration to the next. Although the performance does suffer slightly, the gain in the reduction of computational complexity more than compensates for this loss.

It is important to note here that even though the SVD and Cholesky factorization operations are being run at every iteration, there is a significant gain achieved in the calculation of the autocorrelation function. While each batch processing algorithm would require a total of P^2N^2 multiplications, where $(P \times N)$ is the size of the data block matrix, the recursive algorithms only require P^2N multiplications. Thus, there is a reduction in the number of multiplications by a factor of N, which becomes significant when the number of blocks N is large.

6 Simulations and results

Here we compare results for the newly developed recursive algorithms. Results are shown for a network of 20 nodes, shown in Figure 2. The forgetting factor is both

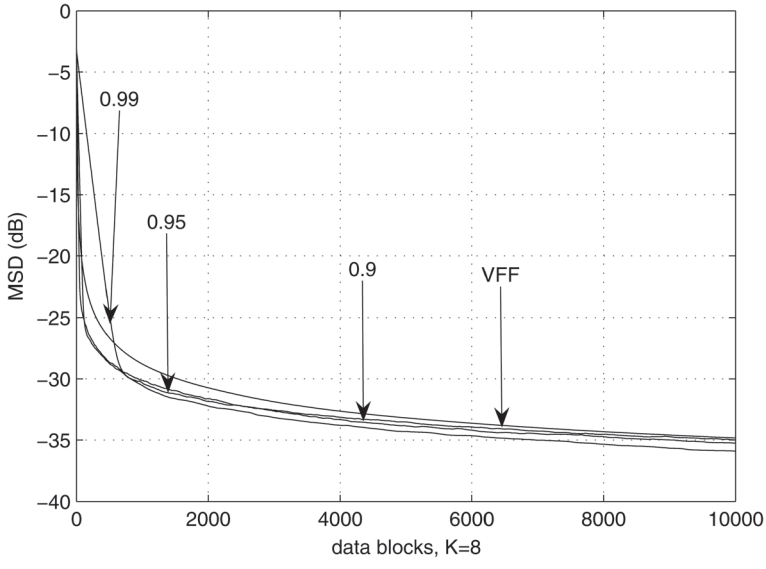

Figure 8 MSD at SNR = 20 dB for RS with different forgetting factors.

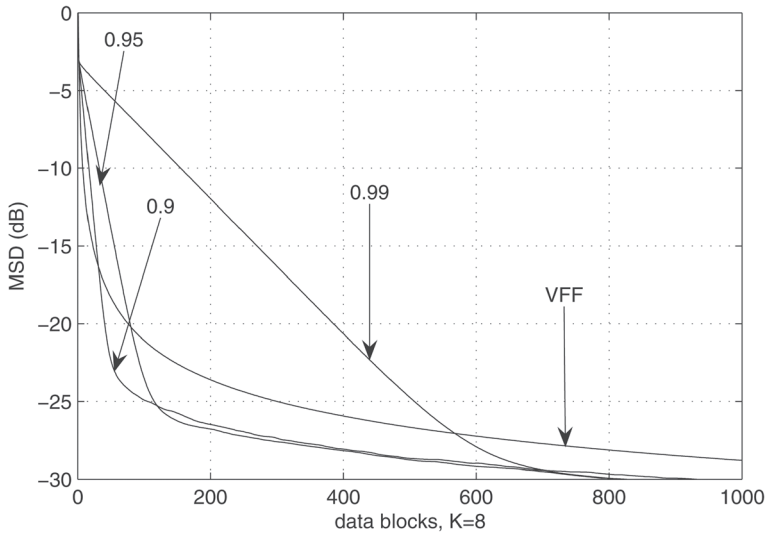

Figure 9 MSD at SNR = 20 dB for RS with different forgetting factors (Transient behavior).

varied as well as kept fixed in order to study its impact on the performance of each algorithm. The two algorithms are compared with each other and in different scenarios. The forgetting factor, data block size and network size are changed one at a time while all other variables are kept constant in order to closely monitor the impact of each of these three varying parameters on the performance of each algorithm.

6.1 Performance of the SVD and Cholesky algorithms

Initially, the two algorithms are used to identify an unknown parameter vector of length $M = 4$ in an environment with the two signal-to-noise ratios (SNR) of 10 and 20 dB. The two forgetting factors used are fixed at $\lambda = \{0.9, 0.99\}$. The block size is taken as $K = 8$. The results for both algorithms are shown in Figures 3, 4, 5 and 6, for both diffusion (DRC, DRS) and no cooperation (NRC, NRS) cases. As can be seen from these figures, the Cholesky algorithm does not perform well with the smaller forgetting factor. However, the performance improves appreciably with an increase in the forgetting factor but its speed of convergence decreases significantly as well. However, the one main positive attribute of the Cholesky algorithm remains to be its low computational complexity. For the SVD algorithm, the performance improves slightly with an increase in forgetting factor but at a significant loss of

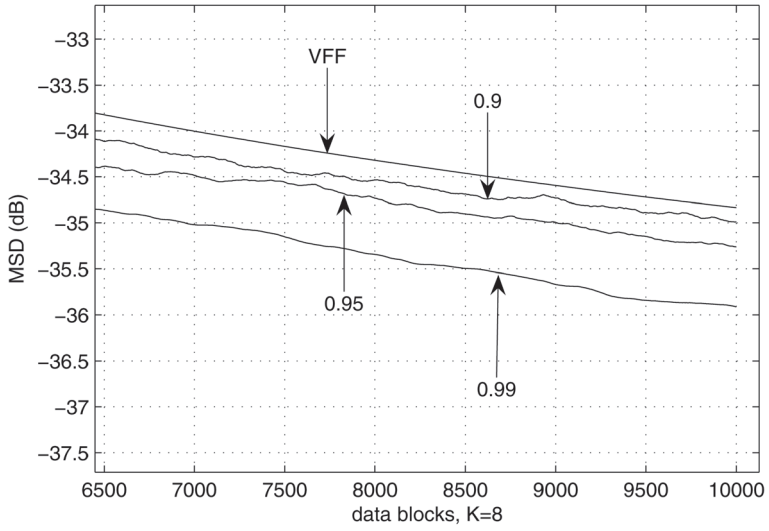

Figure 10 MSD at SNR = 20 dB for RS with different forgetting factors (Near steady-state behavior).

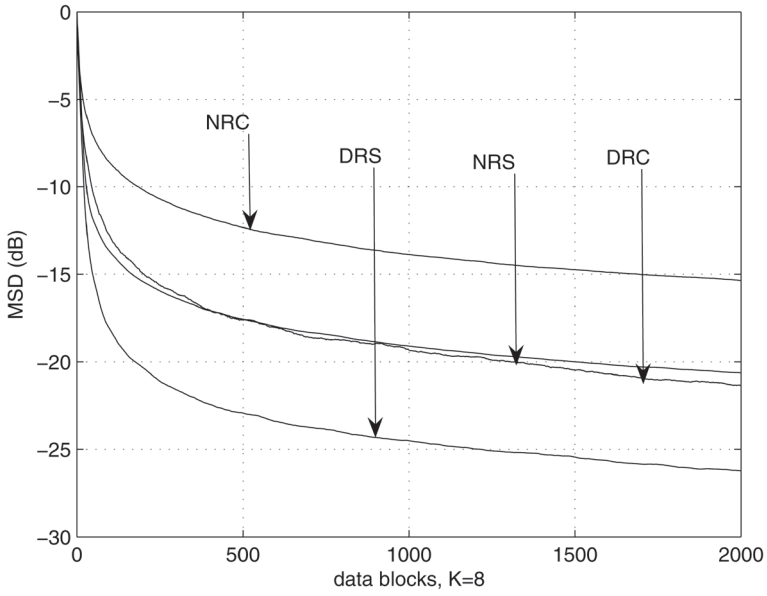

Figure 11 MSD at SNR = 10 dB under best performance conditions.

convergence speed. These remarks have prompted us to further analyze the impact of the forgetting factor on the performance on these two algorithms, as discussed next.

6.2 Further simulation-based analysis of the effect of forgetting factor

Next, the performance of each algorithm is separately studied for different values of the forgetting factor. For the fixed forgetting factor case, the values taken are $\lambda = \{0.9, 0.95, 0.99\}$ and the results are compared with those of the variable forgetting factor case. The SNR is chosen as 20 dB and the network size is taken to be 20 nodes. Figure 7 shows the results for the Cholesky factorization-based RC algorithm. It is seen that the performance improves as the forgetting factor is increased but the convergence slows down. The algorithm performs best when the forgetting factor is variable. The results for the SVD-based RS algorithm are shown in Figures 8, 9 and 10. Figure 8 shows the results for all three fixed forgetting factors as well as those for its variable one. However, as

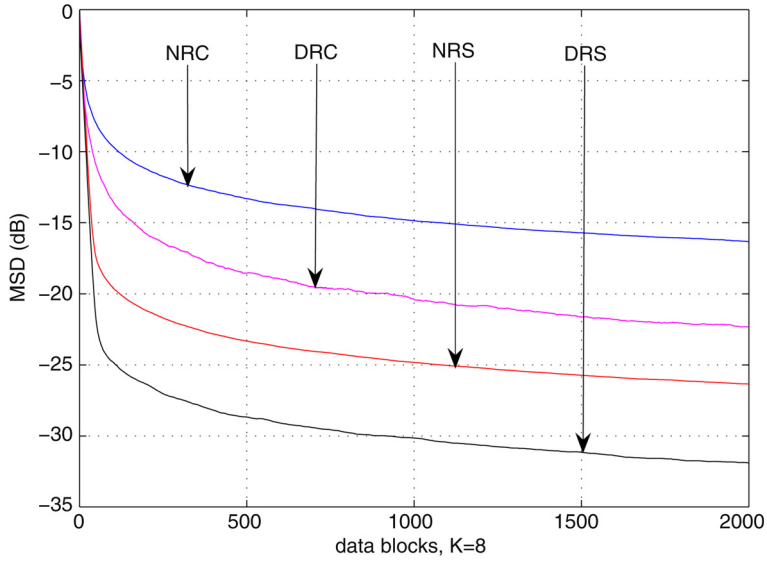

Figure 12 MSD at SNR = 20 dB under best performance conditions.

Table 3 Performance comparison with the batch processing algorithms (all results are in dB)

SNR	CHOL [15]	NRC	DRC	SVD [11]	NRS	DRS
10 dB	−17.79	−15.36	−21.34	−25.69	−20.63	−26.23
20 dB	−18.36	−16.34	−22.35	−31.38	−26.35	−31.90

there is not much difference in performance between the four cases, Figure 8 is then zoomed in to see more clearly the algorithm's transient and near-steady-state behavior. Figure 9 shows the result of this zooming effect. The speed of convergence is fastest for $\lambda = 0.9$ and slowest for $\lambda = 0.99$. For the variable forgetting factor (VFF) case, the speed is fast initially but then slows down with time. Figure 10 shows the behavior of the algorithm near steady-state. It is evident that the fixed case of $\lambda = 0.99$ would yield the lowest steady-state error whereas the VFF case would take the longest to reach the steady-state. Although the steady-state performance of the variable forgetting factor may be as good as for the case of $\lambda = 0.99$ or even better, its speed of convergence is too slow.

6.3 Performance of the two algorithms using an optimal forgetting factor

From the results of Figures 7, 8, 9 and 10, it can easily be inferred that the Cholesky factorization-based approach yields the best results when the forgetting factor is varied whereas the SVD-based algorithm performs best if the forgetting factor is fixed. In order to have a fair performance comparison, the two algorithms need to be compared under conditions in which they both perform best. Figures 11 and 12 give the best performance results of the two algorithms, respectively. As can be seen from these two figures, at an SNR of 10 dB, the Cholesky-based DRC algorithm performs slightly better than the SVD-based RS algorithm without diffusion, whereas both SVD-based algorithms outperform the Cholesky-based algorithms at an SNR of 20 dB. However, the RC algorithm remains computationally less complex than the RS one. A final choice of either of these two algorithms will have to be based on a trade-off between their complexity and performance.

It would be only fair to compare the performance of these algorithms with the original batch processing algorithms from [11] and [15]. At an SNR of 10 dB, the MSE value for the SVD-based algorithm of [11] is −25.69 dB while that of the Cholesky-based algorithm from [15] is −17.79 dB. The corresponding numbers at an SNR of 20 are −31.38 and −18.36 dB, respectively. Comparing these results with the figures, we see that both recursive algorithms perform similar to their batch-processing counterparts. Furthermore, the diffusion algorithms perform better than the batch processing algorithms. The comparison results are tabulated in Table 3.

6.4 Effect of block size

Since it has been stated in [11] and [15] that the block size can affect the performance of the algorithm, the

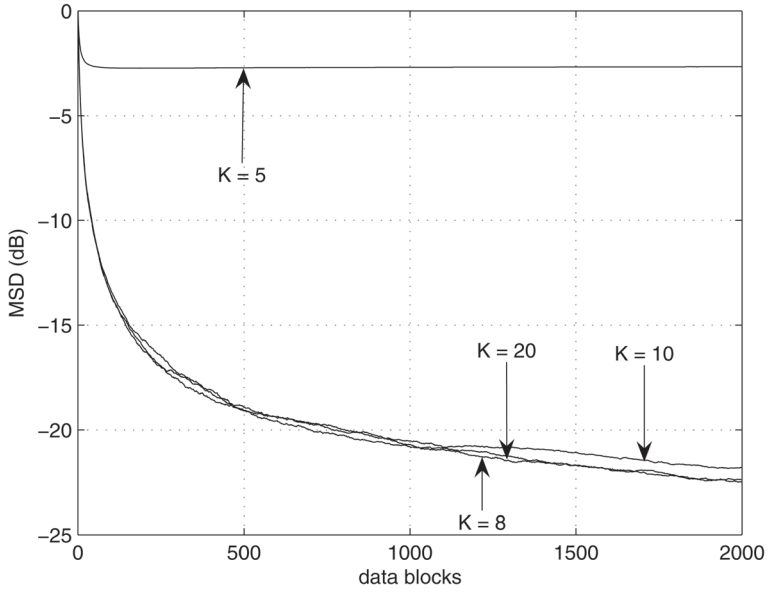

Figure 13 MSD at SNR = 20 dB for varying K for RC.

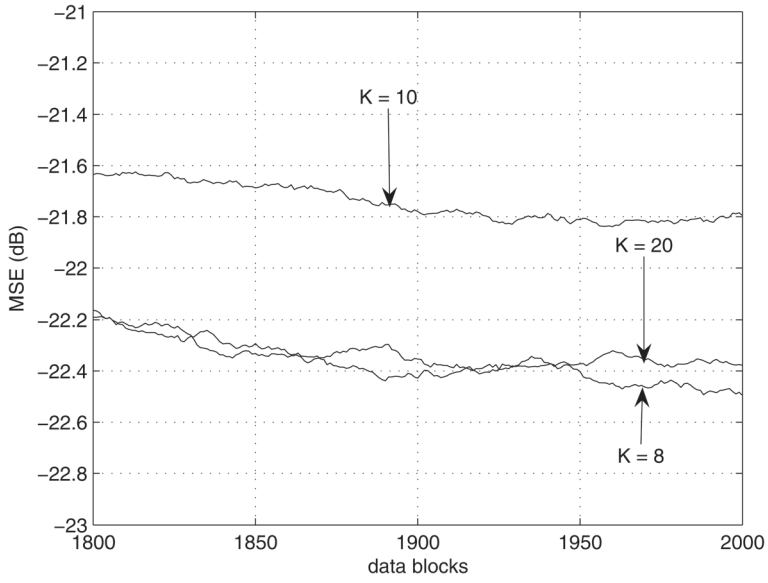

Figure 14 MSD at SNR = 20 dB for varying K for RC (last 200 runs).

performance of our algorithms is also tested here for various block sizes. The block size is varied as $K = \{5, 8, 10, 15, 20\}$ and the SNR is set to 20 dB. These settings are applied to both algorithms separately. Here it is important to note that as the size of the data block increases, the total amount of data required for the same number of blocks also increases. Figures 13 and 14 show the results for the RC algorithm and clearly demonstrate that the algorithm fails badly for $K = 5$. However, for the remaining block sizes, the algorithm's performance remains almost unaffected by the block size changes. The convergence speeds are nearly the same (see Figure 13) and the performance at steady-state is similar as well for the remaining block sizes, with only a slight difference (see Figure 14). From Figure 14 it can be inferred that the best result, in every respect, is achieved when the block size is just large enough to achieve a full rank input data matrix ($K = 8$ in this case), as expected. Thus, it is essential to estimate a tight upper bound for the size of the unknown vector in order to achieve good performance. Figures 15

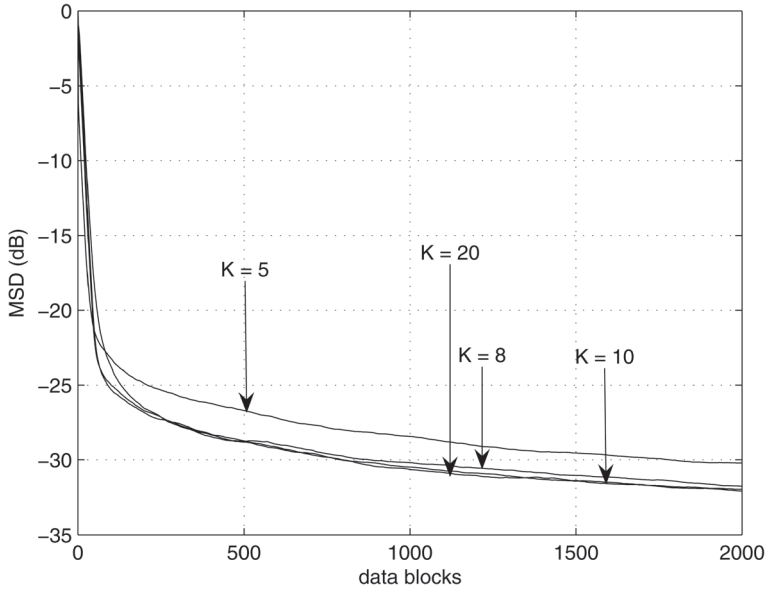

Figure 15 MSD at SNR = 20 dB for varying K for RS.

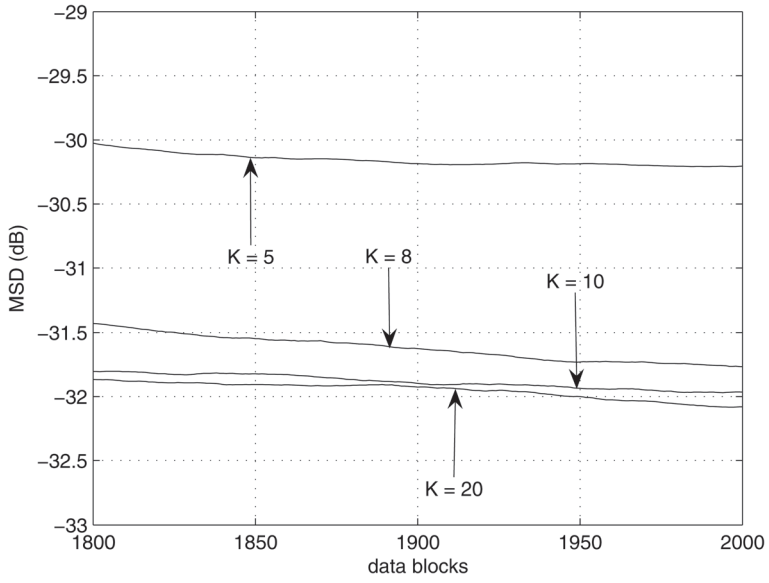

Figure 16 MSD at SNR = 20 dB for varying K for RS (last 200 runs).

and 16 show the results for the RS algorithm. Here, the performance improves gradually with an increase in block size. However, the speed of convergence is slow for a large block size (see Figure 15) even though a larger block size gives better performance at steady-state (see Figure 16). Again it can be inferred that it is best to keep the block size reasonably small in order to achieve a good trade off between performance and speed of convergence, especially when taking into account the fact that a larger block size would mean sensing more data for the same number of blocks.

6.5 Effect of network size

Here the effect of the size of the network on the performance of the algorithms is discussed. For this purpose, the size of the network is varied over the range $N = \{10 - 50\}$ while the forgetting factor is kept fixed at $\lambda = 0.9$ for the

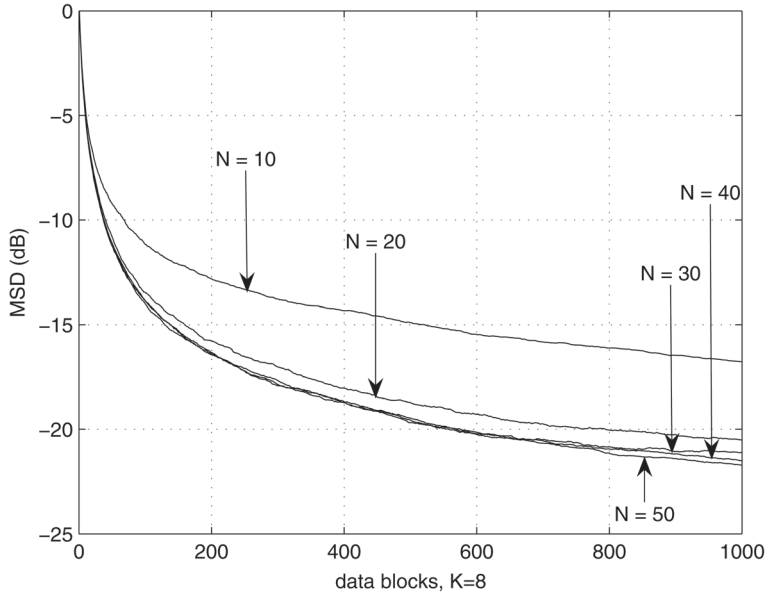

Figure 17 MSD at SNR = 20 dB for varying network sizes for RC.

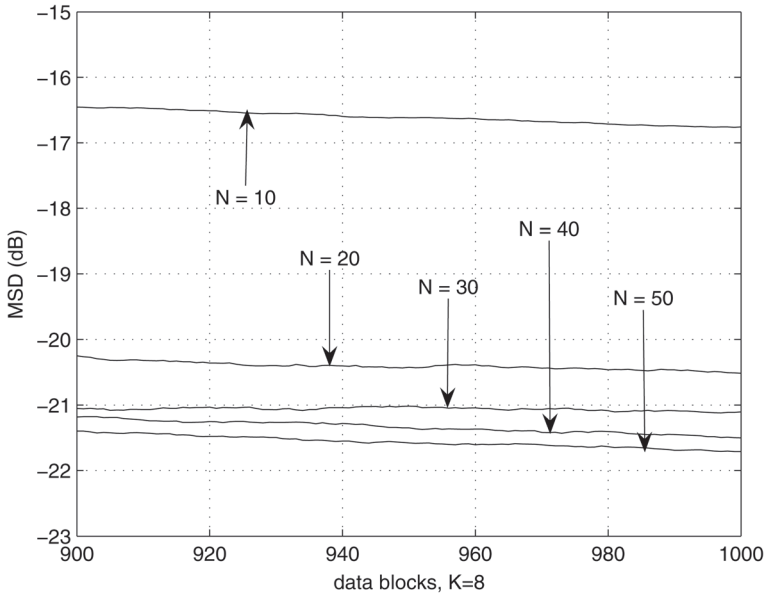

Figure 18 MSD at SNR = 20 dB for varying network sizes for RC (last 100 runs).

RS algorithm and made variable for the RC algorithm. The block size is taken as $K = 8$. This performance comparison is also carried out for both algorithms separately. The number of neighbors for each node is increased gradually as the size of the network is increased. Figures 17 and 18 show results for the RC algorithm. The performance is poor for $N = 10$ but improves as N increases. The initial speed of convergence is similar for various network sizes as can be seen in Figure 17 but, near steady-state, the networks with large sizes show a slight improvement in performance, as shown in Figure 18. Figures 19 and 20 show the results for the RS algorithm. Here the trend is slightly different. It can be seen that the initial speed of convergence improves with an increase in N (see Figure 19) but the improvement in performance is slightly smaller near steady-state (see Figure 20). Also, the difference in performance is smaller for larger networks, which is as expected.

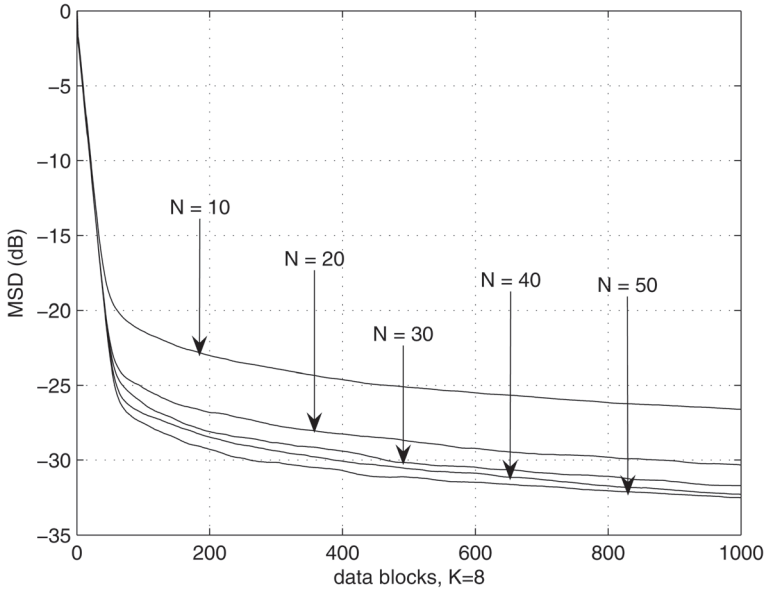

Figure 19 MSD at SNR = 20 dB for varying network sizes for RS.

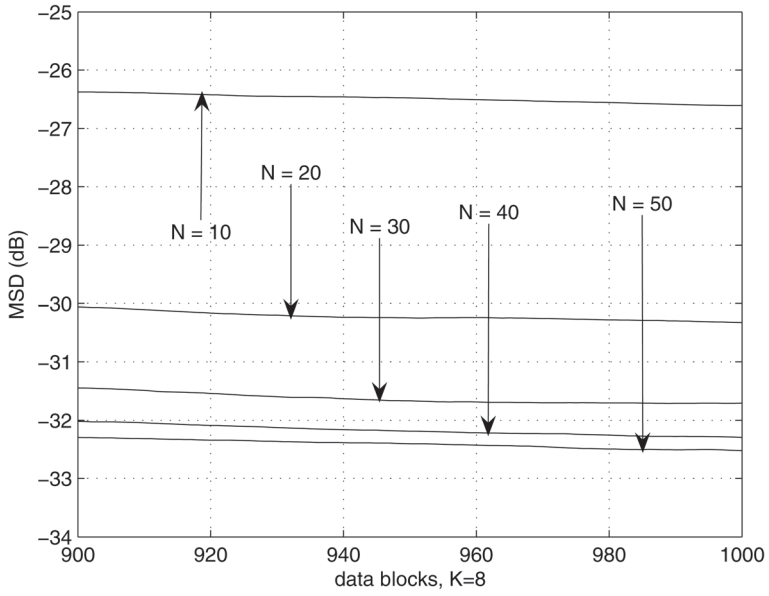

Figure 20 MSD at SNR = 20 dB for varying network sizes for RS (last 100 runs).

6.6 Effect of node malfunction

Finally, it is shown how the performance can be affected if one or more nodes malfunction. Two different network sizes are chosen in two different SNR scenarios to show how performance gets affected by node malfunction or failure. First, a network of 20 nodes is used and five nodes are switched off. Here, switching off a node means that it stops to participate in any further estimation process. The functioning nodes then re-calibrate the weights for the remaining neighbors while the weights for the failed nodes are set to zero. The nodes with the maximum number of neighbors are switched off to see how seriously the network performance might be affected. Results are shown for both SNRs, 10 and 20 dB, in Figures 21 and 22 respectively. The network size is then increased to 50 nodes with about a quarter of the nodes (13) switched off. The corresponding results are shown in Figures 23 and 24. The RC algorithm's performance is worst affected at SNR = 10

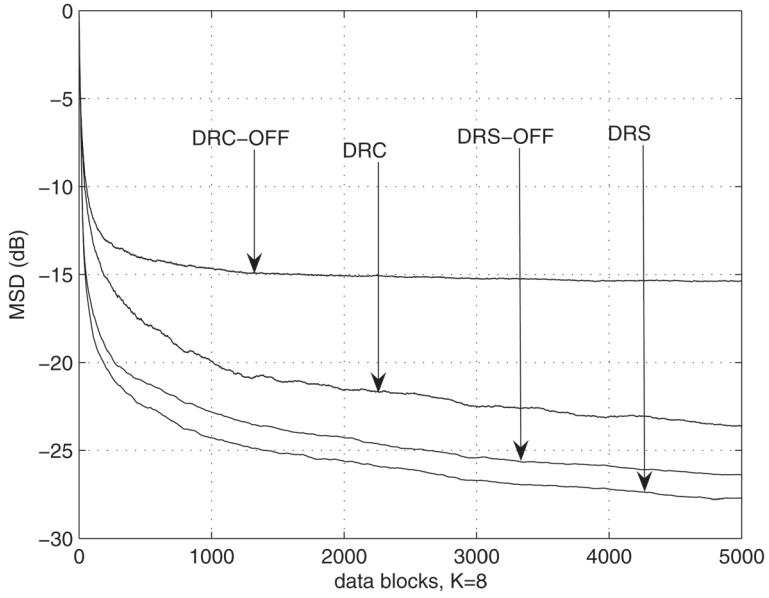

Figure 21 MSD at SNR = 10 dB and $N = 20$ nodes when five most connected nodes are switched off.

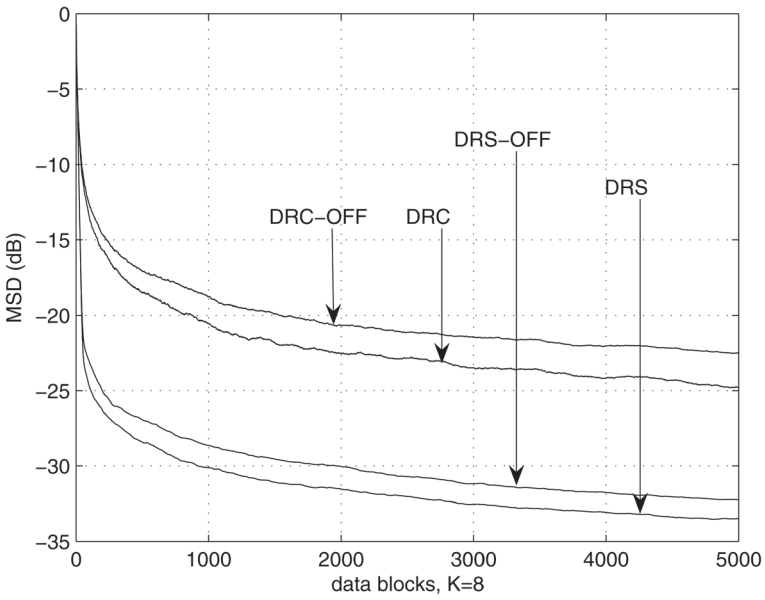

Figure 22 MSD at SNR = 20 dB and $N = 20$ nodes when five most connected nodes are switched off.

dB but remains almost unaffected at SNR = 20 dB, with the small difference in performance getting even smaller when the network size is increased. Under similar test conditions as for the RC algorithm, the degradation of the SVD-based algorithm's performance was found to be similar to that of the RC's in all test cases. This clearly shows that the SVD-based algorithm is also strongly dependent on the connectivity of the nodes. As expected, the overall performance improves with an increase in network size. The effect of switched-off nodes on the performance of both algorithms, however, is similar when the ratio of switched-off nodes with maximum numbers of neighbor nodes to the total number nodes is the same.

7 Conclusion

This work develops blind block recursive least squares algorithms based on Cholesky factorization and singular value decomposition (SVD). The algorithms are then

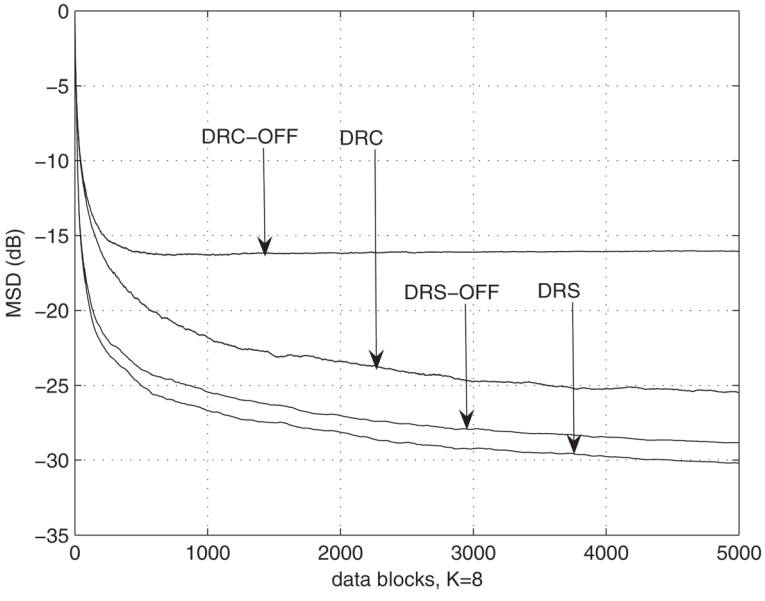

Figure 23 MSD at SNR = 10 dB and $N = 50$ nodes when 13 most connected nodes are switched off.

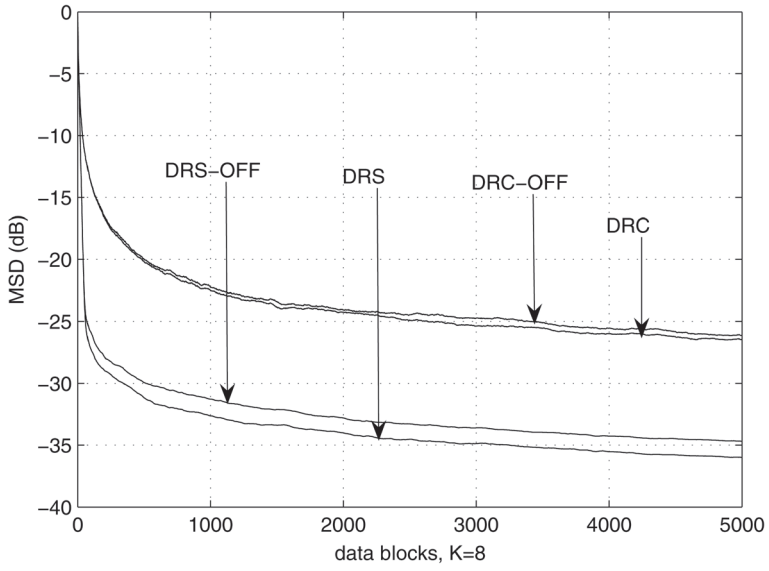

Figure 24 MSD at SNR = 20 dB and $N = 50$ nodes when 13 most connected nodes are switched off.

used to estimate an unknown vector of interest in a wireless sensor network using cooperation between neighboring sensor nodes. Incorporating the algorithms in the sensor networks creates new diffusion-based algorithms, which are shown to perform much better than their non-diffusion-based counterparts. The new algorithms have been tested using both a variable as well as a fixed forgetting factor. The two developed algorithms are named diffusion blind block recursive Cholesky (DRC) and diffusion blind block recursive SVD (DRS) algorithms. Extensive simulation work comparing the two algorithms under different scenarios revealed that the DRS algorithm performs much better than the DRC algorithm but at the cost of a higher computational complexity. Also, of the two algorithms, the DRC algorithm performs better when the forgetting factor is variable whereas the DRS algorithm gives better results with a fixed forgetting factor. In the case of DRS, the value of the forgetting factor does not effect the overall performance a great deal except for a slight variation in convergence speed and steady-state performance. It was also seen that the size of the data block has an effect on the performance of the two algorithms. The speed of convergence slows down with an increasing block size which means an increasing amount of data to be processed. A block size increase, however, does not necessarily improve performance. It was found that, in general, a small block size gives a better performance. Therefore, it is essential to estimate a very low upper bound to the size of the unknown vector so that the data block size to be used is not unnecessarily large. Next, it was noticed that an increase in the network size improves performance but the improvement gradually decreases with an increasing network size. Moreover, it was shown that switching off some nodes with the largest neighborhoods can slightly degrade the performance of the algorithm. Finally at low SNRs, the Cholesky-based algorithm suffers from a severe degradation, whereas the SVD-based one only experiences a slight degradation.

Competing interests

The authors declare that they have no competing interests.

Acknowledgments

The authors acknowledge the support provided by the Deanship of Scientific Research (DSR) at KFUPM under Research Grants RG1216, RG1112, SB101024, and FT100012.

Received: 16 July 2014 Accepted: 19 July 2014
Published: 1 September 2014

References

1. AH Sayed, CG Lopes, Distributed recursive least-squares strategies over adaptive networks, in *Proceedings of the 40th Asilomar Conference on Signals, Systems, Computers* (Monterey, CA, 2006), pp. 233–237
2. CG Lopes, AH Sayed, Incremental adaptive strategies over distributed networks. IEEE Trans. Signal Process. **55**, 4064–4077 (2007)
3. CG Lopes, AH Sayed, Diffusion least-mean squares over adaptive networks: formulation and performance analysis. IEEE Trans. Signal Process. **56**(7), 3122–3136 (2008)
4. ID Schizas, G Mateos, GB Giannakis, Distributed LMS for consensus-based in-network adaptive processing. IEEE Trans. Signal Process. **57**(6), 2365–2382 (2009)
5. MO Bin Saeed, A Zerguine, SA Zummo, A variable step-size strategy for distributed estimation over distributed networks. Eur. J. Adv. Signal Process. **2013**, 135 (2013)
6. Y Sato, A method of self-recovering equalization for multilevel amplitude-modulation. IEEE Trans. Commun. **COM-23**(6), 679–682 (1975)
7. J Proakis, *Digital Communications*, 4th edn. (McGraw-Hill, New York, 2000)
8. L Tong, S Perreau, Multichannel blind identification: from subspace to maximum likelihood methods. Proc. IEEE **86**(10), 1951–1968 (1998)
9. G Xu, H Liu, L Tong, T Kailath, A least-squares approach to blind channel identification. IEEE Trans. Signal Process. **43**(12), 2982–2993 (1995)

10. K Abed-Meraim, W Qiu, Y Hua, Blind system identification. IEEE Trans. Signal Process. **45**(3), 770–773 (1997)
11. A Scaglione, GB Giannakis, S Barbarossa, Redundant filterbank precoders and equalizers part II: blind channel estimation, synchronization, and direct equalization. IEEE Tran. Signal Proc. **47**(7), 2007–2022 (1999)
12. JH Manton, WD Neumann, Totally blind channel identification by exploiting guard intervals. Syst. Control Lett. **48**(2), 113–119 (2003)
13. DH Pham, JH Manton, A subspace algorithm for guard interval based channel identification and source recovery requiring just two received blocks, in *Proceedings of the IEEE ICASSP '03*, vol. 4 (Hong Kong, 2003), pp. 317–320
14. B Su, PP Vaidyanathan, A generalized algorithm for blind channel identification with linear redundant precoders. Eur. J. Adv. Signal Proc. **2007**, 1–13 (2007). Article ID 25672
15. J Choi, C-C Lim, A cholesky factorization based approach for blind FIR channel identification. IEEE Tran. Signal Proc. **56**(4), 1730–1735 (2008)
16. F Cattivelli, AH Sayed, Diffusion LMS strategies for distributed estimation. IEEE Trans. Signal Process. **58**(3), 1035–1048 (2010)

doi:10.1186/1687-6180-2014-136
Cite this article as: Bin Saeed *et al.*: **Blind distributed estimation algorithms for adaptive networks.** *EURASIP Journal on Advances in Signal Processing* 2014 **2014**:136.