Computers & Geosciences 90 (2016) 134–143

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

Stream Kriging: Incremental and recursive ordinary Kriging over spatiotemporal data streams

Xu Zhong *, Allison Kealy, Matt Duckham

Department of Infrastructure Engineering, University of Melbourne, Victoria 3010, Australia

ARTICLE INFO

Article history:
Received 18 June 2015
Received in revised form
29 January 2016
Accepted 2 March 2016
Available online 3 March 2016

Keywords:
Incremental ordinary Kriging
Recursive ordinary Kriging
Spatial interpolation
Spatiotemporal stream processing
Online algorithms

ABSTRACT

Ordinary Kriging is widely used for geospatial interpolation and estimation. Due to the $O(n^3)$ time complexity of solving the system of linear equations, ordinary Kriging for a large set of source points is computationally intensive. Conducting real-time Kriging interpolation over continuously varying spatiotemporal data streams can therefore be especially challenging. This paper develops and tests two new strategies for improving the performance of an ordinary Kriging interpolator adapted to a stream-processing environment. These strategies rely on the expectation that, over time, source data points will frequently refer to the same spatial locations (for example, where static sensor nodes are generating repeated observations of a dynamic field). First, an incremental strategy improves efficiency in cases where a relatively small proportion of previously processed spatial locations are absent from the source points at any given iteration. Second, a recursive strategy improves efficiency in cases where there is substantial set overlap between the sets of spatial locations of source points at the current and previous iterations. These two strategies are evaluated in terms of their computational efficiency in comparison to ordinary Kriging algorithm. The results show that these two strategies can reduce the time taken to perform the interpolation by up to 90%, and approach average-case time complexity of $O(n^2)$ when most but not all source points refer to the same locations over time. By combining the approaches developed in this paper with existing heuristic ordinary Kriging algorithms, the conclusions indicate how further efficiency gains could potentially be accrued. The work ultimately contributes to the development of online ordinary Kriging interpolation algorithms, capable of real-time spatial interpolation with large streaming data sets.

1. Introduction

Spatial interpolation is a common approach to estimating the unknown value of a random field at a given location from known values at a set of spatial source points. A wide variety of spatial interpolation algorithms have been developed, such as inverse distance weighting (IDW), nearest neighborhood, natural neighborhood, thin plate spline, and ordinary Kriging. Ordinary Kriging is recognized as the best linear unbiased estimator because it minimizes the variance of estimation error based on the statistical properties of the random field (Cressie, 1992). The accuracy of ordinary Kriging is highly dependent on the effectiveness of the stochastic model of the random field. When the spatial structure of the random field cannot be modeled effectively, ordinary Kriging may be less accurate than other interpolation algorithms (Falivene et al., 2010; Lu and Wong, 2008). But when an effective stochastic model can be determined from the variogram of the source points, ordinary Kriging is the most accurate interpolation algorithm. The accuracy gain is critical in some situations. For example, in cases where the interpolated surface is required as an input to another simulation model, best accuracy is highly desirable, due to the likely amplification of errors in the interpolated surface as a result of error propagation in the simulation model.

The accuracy gain of ordinary Kriging comes at low scalability. Ordinary Kriging consists of three steps: calculating variogram from source points, fitting a model to the variogram, and linear estimation. Efficient approaches have been proposed to conduct the first two steps (De Baar et al., 2013; Pebesma, 2004). The computational complexity of ordinary Kriging is dominated by the linear estimation step which requires $O(n^3)$ time complexity to solve a system of $n+1$ linear equations for n source points. Even in a conventional "offline" computing environment, where the entire data set is available to the algorithm, scaling ordinary Kriging to large spatial data sets presents significant computational challenges. However, emerging data sources, such as wireless sensor

* Corresponding author.
E-mail address: zhong.xu@unimelb.edu.au (X. Zhong).

http://dx.doi.org/10.1016/j.cageo.2016.03.004
0098-3004/

networks, are much more suited to *online* processing (Ali et al., 2005). Hence online spatial and temporal data stream processing algorithms have been investigated extensively for real-time location-based applications and continuous queries (Ali et al., 2006, 2007; Jin et al., 2006; Jung et al., 2014; Kamel et al., 2010; Nittel and Leung, 2004). In online processing, an algorithm cannot know in advance what data it will receive, and must instead deal with new data sequentially as it arrives (Albers, 2003). Sensor networks, consisting of networks of sensor nodes continuously sensing their environment, are typical streaming spatiotemporal data sources that require online processing. Today, such networks consist of hundreds of nodes. In the future, these networks are expected to scale to thousands or even millions of nodes. Even today's fastest approximate ordinary Kriging algorithms are not well-adapted to dealing with such volumes of dynamic data in real time. Thus, scalable and online ordinary Kriging algorithms are increasingly demanded.

Current efforts for improving the computational efficiency of ordinary Kriging primarily focus on approximate solutions to the system of linear equations. One widely-adopted approach involves reducing the rank of the linear system (the source-to-source covariance matrix and the target-to-source covariance vector). The simplest method is *local neighborhood Kriging*, where target points are estimated using only the closest source points (Hartman and Hössjer, 2008; Rivoirard and Romary, 2011). Underlying this method is the principle of the *screen effect*, where distant source points have near-zero weights for a target (Stein, 2002). The accuracy of the local neighborhood Kriging depends strongly on the selection of neighborhoods. Searching for neighborhoods from massive sets of target points can itself become computationally intensive. Hessami et al. (2001) proposed an efficient neighborhood selection scheme based on Delaunay triangulation. In addition, local neighborhood Kriging may result in discontinuous interpolated and estimation error surfaces. Rivoirard and Romary (2011) improved the continuity of the local neighborhood Kriging estimation surface by penalizing the outermost source points of the neighborhood with random noise. The variance of the noise increases infinitely when a data point is on the boundary of the neighborhood. Another more general and global lower rank Kriging approach is projecting the linear system to a lower dimensional space, where the linear system can be solved with lower computational load. This can be achieved by modeling the spatial covariance with q basis functions (Cressie and Johannesson, 2008) or a predictive process model with q selected knots (Banerjee et al., 2008). The problem of inverting the original covariance matrix can be reduced to inverting a matrix of rank q. The accuracy of these methods strongly depends on the choice of the covariance projection matrix. Determining an optimal projection matrix is the main challenge in these approaches, because it changes with the covariance of the spatial model and the spatial characteristics of the source points.

Another approximate approach to reducing computational load is to transform a dense covariance matrix to a sparse matrix. In this way, Hartman and Hössjer (2008) approximated a Gaussian field by a Gaussian Markov random field with sparse precision matrix. This approximation is accurate, but it is limited to Gaussian fields and not able to handle wide variants of covariance models. A more general approach is to use a positive definite but compactly supported function to taper the spatial covariance matrix to zero beyond a certain range (Furrer et al., 2006). The tapered sparse linear system can be solved more efficiently, using algorithms such as SYMMLQ (Paige and Saunders, 1975), than solving the original dense system, while preserving asymptotic optimality. Particularly for Gaussian covariance models, the discrete Gauss transform and nearest-neighbors search techniques can further improve the efficiency of SYMMLQ (Memarsadeghi et al., 2008). The tapered sparse covariance matrix can be rearranged to reduce bandwidth and solved with more efficient algorithms for narrow sparse systems (Sakata et al., 2004). Lower rank Kriging and covariance tapering Kriging methods can be combined to take into account both global and local spatial features (Sang and Huang, 2012). The combined Kriging system can be solved efficiently using incomplete Cholesky decomposition (Romary, 2013).

More efficient exact solutions to ordinary Kriging can be achieved using parallelization (Cheng, 2013; de Ravé et al., 2014; Guan et al., 2011; Hu and Shu, 2015; Pebesma, 2004; Pesquer et al., 2011; Umer et al., 2008). A task decomposition scheme has been developed for Kriging (Guan et al., 2011), where the Kriging system is solved with parallel QR-decomposition. After solving the weights, the linear estimations for all target points are calculated in parallel. Umer et al. (2008) proposed a distributed iterative Gaussian elimination algorithm for solving the Kriging system of linear equations in a sensor network. The linear map of Gaussian elimination is transmitted across the sensor network along a chain of nodes. This distributes across the sensor network the computational load and the energy consumption for solving the Kriging system. However, the overall computational load of these approaches remains the same as for a centralized algorithm.

All the approaches above do not consider the spatial and temporal autocorrelation in the source data streams, which has been used to aid spatial interpolation methods, such as IDW (Appice et al., 2015), shape-function based methods (Li et al., 2011), and Kriging based methods (Guccione et al., 2012; Kerwin and Prince, 1999a; Vargas-Guzmán and Yeh, 1999). Further, these approaches do not consider the temporal evolution of source data points, an inherent issue in streaming data. Two distinct aspects of dynamism are important to consider: *revision* (the dynamic arrival of new data about a field, i.e., “change in our knowledge”), and *update* (dynamic change to the field itself, i.e., “change in the world”). The *sequential simple Kriging* algorithm addresses the first problem. It avoids using the entire set of source points at once by processing subsets from a partition of the source points sequentially (Vargas-Guzmán and Yeh, 1999). In this algorithm, the interpolation results are revised every time a new subset of source points are incorporated into the Kriging system. Sequential simple Kriging results an exact solution, producing an identical interpolation to that using the whole data set simultaneously, once all subsets of the partition have been processed. Such an *incremental* approach can increase efficiency in processing revisions, as new data about a field arrives. However, incremental approaches are not well adapted to dealing with updates, where the field itself has changed.

By contrast, *recursive update Kriging* is well-adapted to process new data about changes to the underlying interpolated field (i.e., updates). In recursive update Kriging, the time series of spatial functions is treated as a space–time state model (Kerwin and Prince, 1999a). The model assumes that the trends of the mean and spatial covariance model of the time series of functions are static. Recursive update Kriging is able to estimate the values of the time series of functions at target spatial points from source points with observation errors. The algorithm extends ordinary Kriging to incorporate temporal correlation, which improves accuracy when observation error exists. When the observations at the source points are precise (no observation error), this approach reduces to individually batch-Kriging at each iteration. Hence, from the perspective of online Kriging, recursive Kriging offers no improvements in computational efficiency. A further limitation of this approach is that the total number and locations of source points must remain fixed for all times. Even though time-varying source points are considered in (Kerwin and Prince, 1999b), it is still required that the displacement must be small. In practice, it is expected that even in a sensor network with static sensor nodes,

nodes may often fail or be removed, or new nodes may be deployed. Thus, the locations of new source points may differ substantially from the preceding points.

Such incremental and recursive computation can dramatically improve algorithm efficiency by alleviating the need to recompute the entire solution at each iteration. Hence, it should be no surprise that incremental and recursive strategies are widely used in stream processing (Acharya and Lee, 2014; Mokbel et al., 2005; Ünal et al., 2006; Zhang et al., 2008, 2014). However, none of the solutions currently in the literature can claim to offer efficient and exact Kriging results in cases where both revisions (e.g., new source data about a field from sensors at previously unknown spatial locations) and updates (e.g., new source data from known locations about changes to a dynamic field) may occur. Our key contribution in this paper is to develop and compare an incremental algorithm and a recursive algorithm for exact and efficient ordinary Kriging over spatiotemporal data streams where both revisions and updates may occur. Under the data model described in Section 2.2.2, the algorithms provide exactly the same results with the original ordinary Kriging algorithm. Hence the efficiency gain comes at no loss in accuracy; and the performance of the algorithms in presence of sudden data changes, anomalies, noise data are the same with the original Kriging algorithm. Both algorithms are based on the expectation that most values of the covariance matrix are invariant at two consecutive executions of the ordinary Kriging interpolation. In short, the common feature of spatiotemporal data streams that many (though not all) source data points will tend to originate from the same spatial location is used, even as the interpolated field is expected to change over time. In this circumstance, our algorithms can calculate the inverse of current covariance matrix more efficiently using the inverse of historical covariance matrix. The effectiveness of the algorithms developed in this paper is evaluated and compared through experiments using a commercial, off-the-shelf stream processing platform (IBM InfoSphere), which demonstrates the practical efficiency gains of using the algorithms.

Following an introduction to the spatial interpolation problem for spatiotemporal streams, Section 3 describes the novel incremental and the recursive algorithms for calculating the inverse of the covariance matrix. Section 4 demonstrates and compares the efficiency of the algorithms experimentally. Section 5 discusses possible extensions of the algorithms and future work. Finally, Section 6 conclude this paper.

2. Problem description

The traditional ordinary Kriging estimator is briefly introduced in this section. The application of ordinary Kriging estimation within the context of spatiotemporal data streams is then sketched in Section 2.2.

2.1. Ordinary Kriging

Given the values $\mathbf{z} = \{z(p_1),\ z(p_2), \ldots, z(p_n)\}$ at a finite set of n source points $\mathbf{p} = \{p_1,\ p_2, \ldots, p_n\}$, a linear estimate of the values $\hat{\mathbf{z}} = \{\hat{z}(p_{t_1}),\ \hat{z}(p_{t_2}), \ldots, \hat{z}(p_{t_m})\}$ at m target points $\mathbf{p_t} = \{p_{t_1},\ p_{t_2}, \ldots, p_{t_m}\}$ is:

$$\hat{\mathbf{z}} = \mathbf{W}^{\top}\mathbf{z}, \tag{1}$$

where $\mathbf{W} \in \mathbb{R}^{n\times m}$ is the linear weights. Ordinary Kriging is the best linear unbiased estimator which ensures the zero mean and the minimum variance of the estimation error. Solving the optimization problem results in the following system of linear equations:

$$\begin{bmatrix} \mathbf{C_{pp}} & \mathbf{1}_n \\ \mathbf{1}_n^{\top} & 0 \end{bmatrix}\begin{bmatrix} \mathbf{W} \\ \boldsymbol{\mu}^{\top} \end{bmatrix} = \begin{bmatrix} \mathbf{C_{pp_t}} \\ \mathbf{1}_m^{\top} \end{bmatrix}, \tag{2}$$

where $\mathbf{C_{xy}} \in \mathbb{R}^{|\mathbf{x}|\times|\mathbf{y}|}$ stands for the covariance matrix between two given points sets $\mathbf{x}$ and $\mathbf{y}$, $|(\cdot)|$ stands for the cardinality of $(\cdot)$, $\mathbf{1}_n \in \mathbb{R}^n$ is a column vector of all ones, $^{\top}$ stands for the matrix transpose operation, $\boldsymbol{\mu} \in \mathbb{R}^m$ is the Lagrange multiplier.

Although the kernel matrix (left square matrix) of the linear system given by Eq. (2) is not positive definite (a diagonal element is 0), a legal spatial covariance model must ensure $\mathbf{C_{pp}}$ is a symmetric positive definite matrix (Myers, 1992), which can be inverted more efficiently using the Cholesky decomposition than using any other methods. Hence the weights can be solved more efficiently in the form:

$$\mathbf{W} = \left(\mathbf{C_{pp}^{-1}} + \frac{1}{c}\mathbf{C_{pp}^{-1}}\mathbf{1}_n\mathbf{1}_n^{\top}\mathbf{C_{pp}^{-1}}\right)\mathbf{C_{pp_t}} - \frac{1}{c}\mathbf{C_{pp}^{-1}}\mathbf{1}_n\mathbf{1}_m^{\top}. \tag{3}$$

where $c = -\mathbf{1}_n^{\top}\mathbf{C_{pp}^{-1}}\mathbf{1}_n$. $\hat{\mathbf{z}}$ can then be obtained by substituting Eq. (3) into Eq. (1),

$$\hat{\mathbf{z}} = \mathbf{C_{pp_t}^{\top}}\tilde{\mathbf{W}} - \tilde{\mathbf{c}}, \tag{4}$$

where $\tilde{\mathbf{W}} = (\mathbf{C_{pp}^{-1}} + \frac{1}{c}\mathbf{C_{pp}^{-1}}\mathbf{1}_n\mathbf{1}_n^{\top}\mathbf{C_{pp}^{-1}})\mathbf{z}$ and $\tilde{\mathbf{c}} = \frac{1}{c}\mathbf{1}_m\mathbf{1}_n^{\top}\mathbf{C_{pp}^{-1}}\mathbf{z}$. The computational complexity of the steps for calculating $\mathbf{C_{pp}^{-1}}$, $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{c}}$, and $\hat{\mathbf{z}}$ is summarized in Table 1. Many methods such as the FFT-based algorithm (Fritz et al., 2009) and parallel computation (Guan et al., 2011; Cheng, 2013) have been proposed to deal with the case where $m \gg n^2$. This paper focuses on the case where $m \ll n^2$. In this case the computational complexity of ordinary Kriging is dominated by the $O(n^3)$ computation of $\mathbf{C_{pp}^{-1}}$. Hence, in improving the efficiency of online Kriging, this paper focuses on calculating $\mathbf{C_{pp}^{-1}}$ more efficiently in a stream environment.

2.2. Stream ordinary Kriging

In stream computing, data are structured as a sequence of timestamped tuples (the "stream"). Operations on the stream often use the concept of a *window* to impose additional structure upon the stream. Incoming data are accumulated in the window until some trigger condition is met, such as the number of tuples received (count-based), the time elapsed (time-based), or the magnitude of change found in the received data (delta-based). Upon triggering, pre-defined operations are performed on data in the window, before some or all data in the window is "flushed". The cycle then continues anew.

Two common types of window are *tumbling* and *sliding* windows. In a tumbling window, all data in the window are flushed at a trigger. In a sliding window, only the oldest tuples are expunged at a trigger, akin to a FIFO list. Fig. 1 depicts an ordinary Kriging stream operator diagrammatically. The operator aggregates tuples (source points) within a window. When the trigger policy of the window is satisfied, ordinary Kriging interpolation is conducted at all target points $\mathbf{p_t}$, and the results $\hat{\mathbf{z}}_k$ are output to the result stream, ready for the next cycle. The configuration of the window depends on the specific stream query. For example, typical stream queries can include:

Table 1
The computational complexity of the steps for calculating $\mathbf{C_{pp}^{-1}}$, $\tilde{\mathbf{W}}$ and $\tilde{\mathbf{c}}$, and $\hat{\mathbf{z}}$.

Calculating	$\mathbf{C_{pp}^{-1}}$	$\tilde{\mathbf{W}}$ and $\tilde{\mathbf{c}}$	$\hat{\mathbf{z}}$
Computational complexity	$O(n^3)$	$O(n^2)$	$O(mn)$

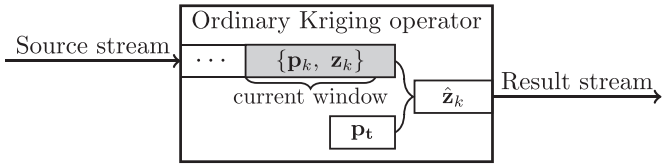

Fig. 1. The schematic of an ordinary Kriging operator that conducts ordinary Kriging spatial interpolation over the tuples within a window and outputs the interpolation at the result stream.

- *Tumbling window*: Interpolate the values at $\mathbf{p}_t$ whenever 1000 new tuples have been received (count-based).
- *Sliding window*: Every 30 s, interpolate the values at $\mathbf{p}_t$ over tuples received in the past 10 min (time-based).

Let $\mathbf{p}_k$ denote the source points in the window at the kth window trigger event. Executing the ordinary Kriging algorithm (Eq. (4)) over the source points in the window as an entirely new batch can become unfeasible as $|\mathbf{p}_k|$ becomes larger, as a consequence of the $O(|\mathbf{p}_k|^3)$ time complexity required to invert $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$.

2.2.1. Incomplete streams

Assuming that locations of the source points in $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ are the same, but only the scalar values at the source points are changed; and that b. the covariance model does not change from the $(k-1)$th trigger to the kth trigger, then the same linear weights calculated at the $(k-1)$th trigger can be used at the kth trigger. This avoids the computation of $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$. In many cases, however, it is to be expected that at any given kth trigger, data from *part but not all* of the locations of the whole source points ($\mathbf{p}_A$) is received ($\mathbf{p}_k \subset \mathbf{p}_A$). As a consequence, the locations of source data points from $\mathbf{p}_{k-1}$ and $\mathbf{p}_k$ may be different, but substantially overlap: $\mathbf{p}_{k-1} \neq \mathbf{p}_k$ and $|\mathbf{p}_{k-1} \cap \mathbf{p}_k| \gg 0$. In the sequel, the notation $\ddot{\mathbf{p}}_{k-1} \equiv \mathbf{p}_{k-1} \cap \mathbf{p}_k$ is used.

Wireless sensor networks typically generate such data steams. The location of the sensor nodes $\mathbf{p}_A$ across the whole network may change only infrequently. However, intermittent and frequent node and link failures are commonplace in sensor networks, and it is usual that data from the network is incomplete (i.e., $\mathbf{p}_k \subset \mathbf{p}_A$). For example, we deployed three RISERnet networks (Zhong et al., 2015) to monitor weather conditions in wildfire-prone areas. Wildfire spread simulation tools, such as PHOENIX RapidFire (Tolhurst et al., 2008), require regular, gridded inputs. Hence in order to apply the RISERnet data to such tools, the measurements need to be interpolated spatially. The RISERnet networks only consist of 70 nodes currently, but similar networks are expected to scale to thousands or even millions of nodes in the future. Hence it is challenging to interpolate the data efficiently to support online applications. Although the location of the sensor nodes of our RISERnet networks is static, link failures are inevitable due to significant signal attenuation caused by trees and undergrowth. This produces the incomplete streams described previously.

To illustrate, Fig. 2 depicts an example of 20 static nodes. In each window k, the measurement from a node is received with a probability of 80%. In Fig. 2, the filled dots indicate those nodes whose measurements are received and stored in $\mathbf{p}_k$ at the kth trigger of the window.

2.2.2. Assumptions

This paper aims to improve the efficiency of ordinary Kriging operations over aggregated spatiotemporal data streams when compared with the traditional batch algorithm. To simplify the description of the algorithms, the approaches explored in the following section assume that covariance model is static (unchanging). Kerwin and Prince (1999a, 1999b) and Vargas-Guzmán and Yeh (1999) assumed static covariance model to utilize the autocorrelation of the covariance matrix $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$ for accuracy gain. In this paper, static covariance model is assumed for efficiency gain. In practice, the covariance model of a random field may change over time. The algorithms proposed in this paper can handle time-varying covariance model if the changes in the model satisfy the criteria described in Section 5.2. It is further assumed that either:

1. $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|, \ \forall\, k > 1$, i.e., the source points at any given iteration k refer to a majority of spatial locations processed from iteration 1 to iteration k; or
2. $|\bar{\mathbf{p}}'_{k-1}| \ll |\mathbf{p}_k|$ and $|\bar{\mathbf{p}}''_k| \ll |\mathbf{p}_k|, \ \forall\, k > 1$, i.e., the source points at any given iteration k refer to a majority of spatial locations processed at iteration $k-1$;

where:

$$\bar{\mathbf{p}}_{1:k} \equiv \{p | p \in \mathbf{p}_{1:k} \text{ and } p \notin \mathbf{p}_k\}$$

$$\bar{\mathbf{p}}'_{k-1} \equiv \{p | p \in \mathbf{p}_{k-1} \text{ and } p \notin \ddot{\mathbf{p}}_{k-1}\}$$

$$\bar{\mathbf{p}}''_k \equiv \{p | p \in \mathbf{p}_k \text{and} p \notin \ddot{\mathbf{p}}_{k-1}\}$$

and where $\mathbf{p}_{1:k}$ is the source points that have been received up to the kth trigger event of the window. An efficient incremental algorithm is firstly proposed for assumption 1, but it is not suitable for assumption 2. Thus a recursive algorithm, which can also in some situations improve efficiency in assumption 1, is proposed to handle assumption 2.

3. Methods

This section presents the incremental approach and the recursive approach to efficient stream Kriging when the source points satisfy one of the key two assumptions made in Section 2.2.2.

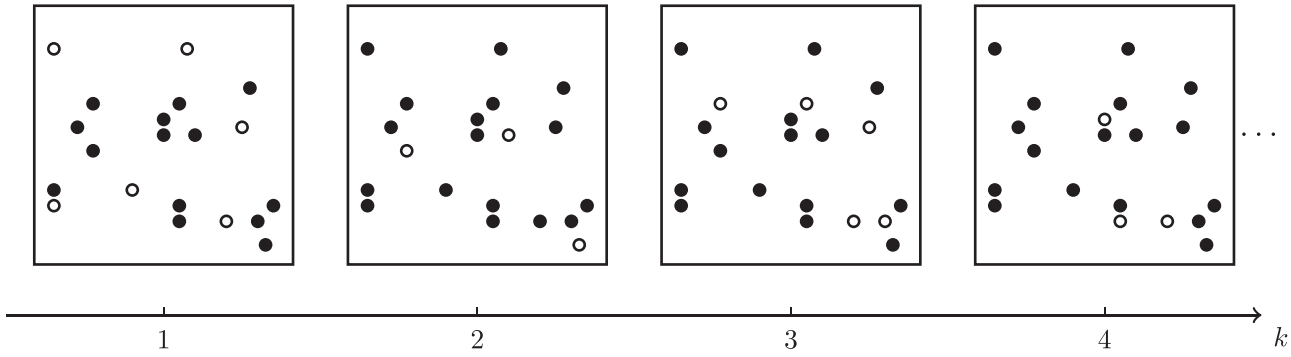

Fig. 2. An example of 20 immobile nodes, where the measurement of each node is received by the stream processor with a probability of 80%. The filled dots indicate the subset $\mathbf{p}_k$ of nodes $\mathbf{p}_A$ whose measurements are received by the stream processor at the kth trigger of the window.

3.1. Incremental computation of $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$

Let $\bar{\mathbf{p}}_k = \{p | p \in \mathbf{p}_k \text{ and } p \notin \mathbf{p}_{1:k-1}\}$ denote the new source points in the window at the kth trigger of the window. $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ can be computed incrementally from $\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}$, called the *addition* step. $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ can then be obtained from $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$, called the *extraction* step. As this algorithm needs to calculate $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ incrementally, it is called the *incremental* algorithm. The details of this algorithm are described below.

3.1.1. Incorporating $\bar{\mathbf{p}}_k$ into the Kriging system: the addition step

For $k > 1$, when the window contains new source points at the kth trigger of the window ($\bar{\mathbf{p}}_k \neq \varnothing$), $\mathbf{p}_{1:k}$ needs to be updated as $\mathbf{p}_{1:k} = \mathbf{p}_{1:k-1} \cup \bar{\mathbf{p}}_k$. The covariance matrix $\mathbf{C}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ of $\mathbf{p}_{1:k}$ can then be written in block form as:

$$\mathbf{C}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}} = \begin{bmatrix} \mathbf{C}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}} & \mathbf{C}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k} \\ \mathbf{C}^{\top}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k} & \mathbf{C}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k} \end{bmatrix}. \tag{5}$$

A legal covariance model must ensure that $\mathbf{C}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}$ and $\mathbf{C}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k}$ are symmetric positive definite matrices, thus $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ can be obtained using the following block matrix inverse formula:

$$\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}} = \begin{bmatrix} \mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}} + \bar{\mathbf{C}}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k}\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k}\bar{\mathbf{C}}^{\top}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k} & -\bar{\mathbf{C}}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k}\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k} \\ -\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k}\bar{\mathbf{C}}^{\top}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k} & \tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k} \end{bmatrix}, \tag{6}$$

where $\bar{\mathbf{C}}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k} = \mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}\mathbf{C}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k}$ and $\tilde{\mathbf{C}}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k} = \mathbf{C}_{\bar{\mathbf{p}}_k\bar{\mathbf{p}}_k} - \mathbf{C}^{\top}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k}\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}\mathbf{C}_{\mathbf{p}_{1:k-1}\bar{\mathbf{p}}_k}$. After inverting the initial covariance matrix $\mathbf{C}_{\mathbf{p}_1\mathbf{p}_1}$, $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ can be deduced incrementally from $\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}$.

3.1.2. Computing $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ from $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$: the extraction step

As $\mathbf{p}_k \subseteq \mathbf{p}_{1:k}$, $\mathbf{C}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ can be permuted so that:

$$\mathbf{P}^{\top}\mathbf{C}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\mathbf{P} = \begin{bmatrix} \mathbf{C}_{\mathbf{p}_k\mathbf{p}_k} & \mathbf{Y} \\ \mathbf{Y}^{\top} & \mathbf{Z} \end{bmatrix}, \tag{7}$$

where $\mathbf{Y} \in \mathbb{R}^{|\mathbf{p}_k| \times |\bar{\mathbf{p}}_{1:k}|}$ and $\mathbf{Z} \in \mathbb{R}^{|\bar{\mathbf{p}}_{1:k}|^2}$ are the blocks resulted from the permutation; and $\mathbf{P} \in \mathbb{R}^{|\mathbf{p}_{1:k}|^2}$ is the permutation matrix. The first $|\mathbf{p}_k|$ columns of $\mathbf{P}$ are determined by which elements of $\mathbf{p}_{1:k}$ are in $\mathbf{p}_k$ as:

$$\mathbf{P}[i,\ j] = \begin{cases} 1 & \text{if } \mathbf{p}_{1:k}[i] = \mathbf{p}_k[j], \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j = 1,\ 2,\ \ldots,\ |\mathbf{p}_k|, \tag{8}$$

where $(\cdot)[i,\ j]$ is the entry of $(\cdot)$ at row i and column j. The $|\mathbf{p}_k| + 1$ to $|\mathbf{p}_{1:k}|$ columns of $\mathbf{P}$ can be filled with the rest columns of a $|\mathbf{p}_{1:k}| \times |\mathbf{p}_{1:k}|$ permutation matrix in any order. For example:

$$\mathbf{P}[i,\ |\mathbf{p}_k| + j] = \begin{cases} 1 & \text{if } \mathbf{p}_{1:k}[i] = \bar{\mathbf{p}}_{1:k}[j], \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j = 1,\ 2,\ \ldots,\ |\bar{\mathbf{p}}_{1:k}|. \tag{9}$$

As the inverse of a permutation matrix is its transpose, Eq. (7) and the block matrix inverse formula yield

$$\mathbf{P}^{\top}\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\mathbf{P} = \left(\mathbf{P}^{\top}\mathbf{C}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\mathbf{P}\right)^{-1} = \begin{bmatrix} \mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k} + \mathbf{U}\mathbf{X}^{-1}\mathbf{U}^{\top} & \mathbf{U}.4em \\ \mathbf{U}^{\top} & \mathbf{X} \end{bmatrix}, \tag{10}$$

where $\mathbf{U} = -\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}\mathbf{Y}\mathbf{X}$, and $\mathbf{X} = (\mathbf{Z} - \mathbf{Y}^{\top}\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}\mathbf{Y})^{-1}$. Thus in order to calculate $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$, $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ is firstly permuted as $\mathbf{C}'_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}} = \mathbf{P}^{\top}\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\mathbf{P}$. Then according to Eq. (10):

$$\mathbf{U} = \mathbf{C}'_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\left[1{:}\ |\mathbf{p}_k|,\ |\mathbf{p}_k| + 1{:}\ |\mathbf{p}_{1:k}|\right] \tag{11}$$

$$\mathbf{X} = \mathbf{C}'_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\left[|\mathbf{p}_k| + 1{:}\ |\mathbf{p}_{1:k}|,\ |\mathbf{p}_k| + 1{:}\ |\mathbf{p}_{1:k}|\right], \tag{12}$$

where $(\cdot)[r_1{:}\ r_2,\ c_1{:}\ c_2]$ denotes the sub-matrix of $(\cdot)$ from row r_1 to row r_2 and column c_1 to row c_2. $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ can be deduced from Eq. (10) as,

$$\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k} = \mathbf{C}'_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}\left[1{:}\ |\mathbf{p}_k|,\ 1{:}\ |\mathbf{p}_k|\right] - \mathbf{U}\mathbf{X}^{-1}\mathbf{U}^{\top}. \tag{13}$$

Eq. (13) solves the inverse of the $|\mathbf{p}_k| \times |\mathbf{p}_k|$ covariance matrix $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$ by inverting the $|\bar{\mathbf{p}}_{1:k}| \times |\bar{\mathbf{p}}_{1:k}|$ matrix $\mathbf{X}$ with the overhead of matrix permutation and two matrix multiplications. Further, some extra time is needed to update the matrix $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ when the addition step is required. If $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$, $\forall\ k > 1$, calculating $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ using this incremental algorithm is faster than inverting $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$ directly. For example, assume the set of all source points $\mathbf{p}_A$ does not change over time, and the window stores the data from a random subset of $\mathbf{p}_A$ at every trigger of the temporal window. Then, the addition step is needed only at the first k_f triggers of the window, where k_f is the index of the window trigger when $\mathbf{p}_{1:k_f} = \mathbf{p}_A$. For $k > k_f$, only the extraction step is needed. If a large enough part of $\mathbf{p}_A$ is received and stored in the window at every trigger, the assumption $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$, $\forall\ k > 1$ can be satisfied. An experimental evaluation of this expectation is explored in Section 4.2.

In the incremental algorithm, $\mathbf{p}_{1:k}$ and $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ are maintained to accelerate the ordinary Kriging interpolation at the next trigger event of the window. If the assumption $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$, $\forall\ k > 1$ is satisfied, the memory cost of the incremental algorithm is similar to that of the original ordinary Kriging algorithm. However, the incremental algorithm is problematic when $\mathbf{p}_A$ changes over time. In practice this might occur if sensor nodes are occasionally redeployed or removed, or if a sliding temporal window is used in combination with moving sensors. These circumstances can result in unbounded sizes of $\mathbf{p}_{1:k}$ and $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$, which invalidates the assumption $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$ as k increases. The recursive algorithm below is proposed to address this issue for high computational efficiency even if $\mathbf{p}_A$ slowly changes.

3.2. Recursive computation of $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$

To address the problem of changing $\mathbf{p}_A$, a recursive algorithm is developed to calculate $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ recursively from $\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}$. As $\ddot{\mathbf{p}}_{k-1} \subseteq \mathbf{p}_{k-1}$, $\mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}}$ can be calculated from $\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}$ using the extraction step described in Section 3.1.2. In this step, the first $|\ddot{\mathbf{p}}_{k-1}|$ columns of the permutation matrix $\mathbf{P}_r \in \mathbb{R}^{|\mathbf{p}_{k-1}|^2}$ are determined by which elements of $\mathbf{p}_{k-1}$ are in $\ddot{\mathbf{p}}_{k-1}$ as

$$\mathbf{P}_r[i,\ j] = \begin{cases} 1 & \text{if } \mathbf{p}_{k-1}[i] = \ddot{\mathbf{p}}_{k-1}[j], \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j = 1,\ 2,\ \ldots,\ |\ddot{\mathbf{p}}_{k-1}|. \tag{14}$$

The $|\ddot{\mathbf{p}}_{k-1}| + 1$ to $|\mathbf{p}_{k-1}|$ columns of $\mathbf{P}_r$ can be filled as

$$\mathbf{P}_r[i,\ |\ddot{\mathbf{p}}_{k-1}| + j] = \begin{cases} 1 & \text{if } \mathbf{p}_{k-1}[i] = \bar{\mathbf{p}}'_{k-1}[j], \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } j = 1,\ 2,\ \ldots,\ |\bar{\mathbf{p}}'_{k-1}|. \tag{15}$$

$\mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}}$ can be calculated in a similar fashion to Eq. (13) by

$$\mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}} = \mathbf{C}'_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}\left[1{:}\ |\ddot{\mathbf{p}}_{k-1}|,\ 1{:}\ |\ddot{\mathbf{p}}_{k-1}|\right] - \mathbf{U}_r\mathbf{X}_r^{-1}\mathbf{U}_r^{\top},$$

where:

$$\mathbf{C}'_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}} = \mathbf{P}_r^{\top}\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}\mathbf{P}_r \tag{16}$$

$$\mathbf{U}_r = \mathbf{C}'_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}\left[1{:}\ |\ddot{\mathbf{p}}_{k-1}|,\ |\ddot{\mathbf{p}}_{k-1}| + 1{:}\ |\mathbf{p}_{k-1}|\right] \tag{17}$$

$$\mathbf{X}_r = \mathbf{C}'_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}\left[|\ddot{\mathbf{p}}_{k-1}| + 1{:}\ |\mathbf{p}_{k-1}|,\ |\ddot{\mathbf{p}}_{k-1}| + 1{:}\ |\mathbf{p}_{k-1}|\right]. \tag{18}$$

Then $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ can be calculated from $\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}$ by incorporating $\bar{\mathbf{p}}''_k$ using the addition step described in Section 3.1.1 as

$$\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k} = \begin{bmatrix} \mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}} + \bar{\mathbf{C}}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k}\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k}\bar{\mathbf{C}}^{\top}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k} & -\bar{\mathbf{C}}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k}\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k} \\ -\tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k}\bar{\mathbf{C}}^{\top}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k} & \tilde{\mathbf{C}}^{-1}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k} \end{bmatrix}, \quad (19)$$

where $\bar{\mathbf{C}}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k} = \mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}}\mathbf{C}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k}$ and $\tilde{\mathbf{C}}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k} = \mathbf{C}_{\bar{\mathbf{p}}'_k\bar{\mathbf{p}}'_k} - \mathbf{C}^{\top}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k}\mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}}\mathbf{C}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}'_k}$.

In this recursive algorithm, the extraction and the addition steps need to invert a $|\bar{\mathbf{p}}'_{k-1}| \times |\bar{\mathbf{p}}'_{k-1}|$ matrix and a $|\bar{\mathbf{p}}''_k| \times |\bar{\mathbf{p}}''_k|$ matrix, respectively. Hence when $|\bar{\mathbf{p}}'_{k-1}| \ll |\mathbf{p}_k|$ and $|\bar{\mathbf{p}}''_k| \ll |\mathbf{p}_k|$, computing $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ with this recursive algorithm can be faster than inverting $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$ directly. Compared to the incremental algorithm, the recursive algorithm only saves $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ and $\mathbf{p}_k$. Thus the memory load does not increase as $\mathbf{p}_A$ changes over time and maintains the same with that of the original ordinary Kriging algorithm. Moreover, the computational load depends on $|\bar{\mathbf{p}}'_{k-1}|$ and $|\bar{\mathbf{p}}''_k|$ rather than $|\bar{\mathbf{p}}_{1:k}|$. Hence, in practice the computational efficiency is expected to depend on the number of source locations that change between successive window triggers, but not to decrease as $\mathbf{p}_A$ changes over time. An experimental investigation of these expectations is presented in Section 4.3.

4. Results

In this section, the efficiency of the incremental and recursive algorithms developed in this paper is evaluated and compared experimentally. The algorithms were implemented within a commercial stream-processing platform: IBM InfoSphere Streams. The ability to use an existing, off-the-shelf stream processing platform was a key objective for our work, ensuring our approach is practically usable. Two sets of experiments were conducted, to evaluate and compare the efficiency of the incremental and recursive algorithms under first static and then dynamic $\mathbf{p}_A$.

4.1. Experimental setup

Dynamic scalar fields were simulated in the experiment. Each of the fields was simulated using a mixture of 200 Gaussian functions. The signs of these functions obey a first-order binomial distribution $B(1, 0.5)$. The centers of the Gaussian functions were drawn from a uniform distribution across the test area. The ranges of the Gaussian functions are also random numbers that follow a uniform distribution. This surface generation scheme provides a close-form function of the experimental surfaces, which allows the source points to distribute anywhere in the study area. The semivariogram of the surfaces was generated, and was found to be fitted best by the Stable model (Wackernagel, 1995) with exponent of 1.5. The covariance model was then deduced from this fitted variogram model. This covariance model is assumed to be static over all the sampling cycles.

In the experiments, the fields are sampled by sensor nodes ($\mathbf{p}_A$) randomly located with a uniform distribution. The input port of the spatial interpolation operator for receiving source points was configured with a tumbling window (i.e., all tuples are flushed after each sample cycle). In the first experiment, a random subset of nodes was drawn from $\mathbf{p}_A$ with a uniform probability P_v to simulate node and link failures. In the second experiment, $\mathbf{p}_A$ was allowed to change over time, to simulate node movement and redeployment. In either case, $\mathbf{p}_k$ denotes the set of the sensor nodes drawn in the kth sampling cycle. In both experiments, a 50×50 grid of target points was used to generate the interpolated surface.

It is important to note that no temporal interpolation is performed by either of the incremental algorithm or the recursive algorithm. As a result, changes to the actual scalar values across the field over time have no effect upon the computational characteristics of the algorithms. The algorithms take advantage of static spatial covariance and slow evolution in the spatial locations of source data points to accelerate Kriging, but make no assumptions about changes to the scalar values of the field.

4.2. Efficiency of incremental and recursive algorithms under static $\mathbf{p}_A$

The first experiment examined the efficiency of the incremental algorithm and the recursive algorithm, under the assumption of static (unchanging) $\mathbf{p}_A$. Five different sizes of $|\mathbf{p}|$ (200, 400, 600, 800, 1000) were tested in order to investigate the scalability of the algorithms. For each of these, six different values of P_v (60%, 70%, 80%, 90%, 98%, and 100%) were tested, effectively varying the rate of change in the spatial locations of source points in each window. For each different combination of $|\mathbf{p}|$ and P_v, 30 repetitions of the simulation were performed. The execution time for interpolation (RT_k) for each cycle was measured for a total of 10 sampling cycles.

Fig. 3 illustrates the raw results for a typical set of simulations, with P_v=80%. The figures show the changing execution time over

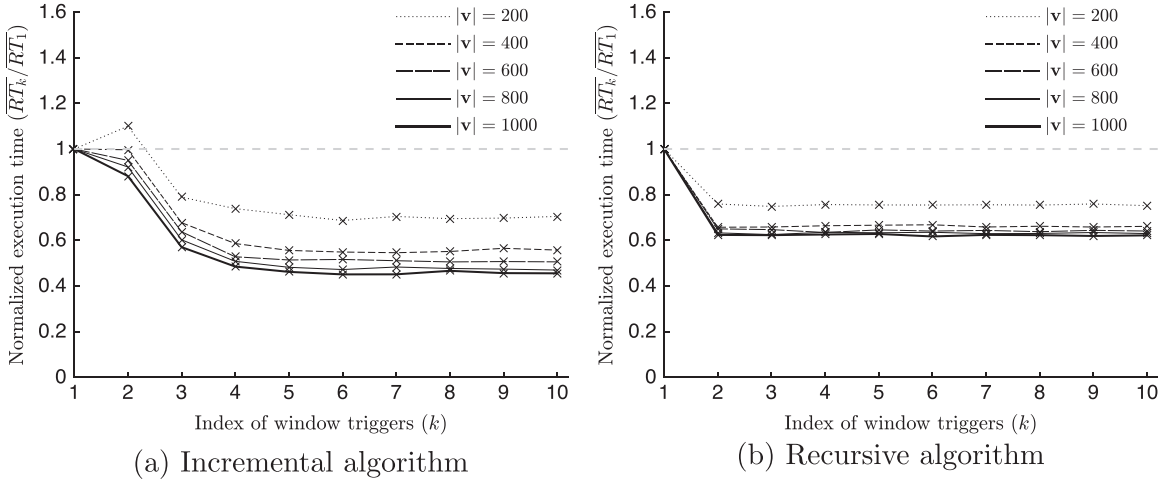

Fig. 3. Comparison of the normalized execution time (ratio between mean RT_k and mean RT_1) of the incremental algorithm (a) and the recursive algorithm (b) over $k = 1, 2, \ldots, 10$ windows for $P_v = 80\%$ and $|\mathbf{p}| \in \{200, 400, 600, 800, 1000\}$. (a) Incremental algorithm. (b) Recursive algorithm.

the first $k=1,2,\ldots,10$ windows for the five sizes of source point data set. The ratio of mean RT_k ($\overline{RT_k}$) to mean RT_1 ($\overline{RT_1}$) of the 30 replications is used as the metric to evaluate the improvement in efficiency of our algorithms (when $k > 1$) compared to direct batch interpolation (when $k=1$). Thus, an $\overline{RT_k}/\overline{RT_1}$ ratio of 0.5 means that our algorithm executed in 50% of the time required by naive batch Kriging recomputation at every iteration. In the case of Fig. 3, the recursive algorithm rapidly stabilizes to below 80% of the $\overline{RT_1}$ execution time. The incremental algorithm slightly outperforms the recursive algorithm, reaching below 50% of the $\overline{RT_1}$ execution time for larger sets of source points.

As is clear from Fig. 3, the execution time rapidly stabilizes after 2–5 windows, a common feature across all scenarios tested. For the incremental algorithm, the addition step is needed at the initial k_f triggers of the window when $P_v < 100\%$. As $\mathbf{p}_{1:k}$ approaches $\mathbf{p}_A$ for $k \le k_f$, fewer and fewer nodes need to be incorporated into $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$. Hence, as anticipated, the execution time reduces gradually and reaches the asymptotic level when $\mathbf{p}_{1:k} = \mathbf{p}_A$. The recursive algorithm does not need to calculate $\mathbf{C}^{-1}_{\mathbf{p}_{1:k}\mathbf{p}_{1:k}}$ incrementally, hence it delivers the same extent of improvement in efficiency from $k=2$.

By using the average execution time for $k \in \{7, \ldots, 10\}$, Fig. 4 enables us to show the change in execution time across all P_v values and $|\mathbf{p}|$ tested. The figure shows that for both incremental and recursive algorithms, proportional improvements in execution time increase with larger data sets. The graphs show that the recursive algorithm is more efficient than the incremental algorithm in the 60–70% range because $|\bar{\mathbf{p}}_{1:k}|$ is large in the extraction step of the incremental algorithm when P_v is low. However, when P_v is 60–70%, the recursive algorithm provides only limited improvement in efficiency compared to naive batch Kriging.

When $80\% \le P_v < 100\%$, both algorithms are more efficient than direct batch computation. In this range, the asymptotic execution time of the incremental algorithm is less than that of the recursive algorithm. This is because only the extraction step is needed in the incremental algorithm for $k > k_f$, and $|\bar{\mathbf{p}}_{1:k}|$ is small enough. On the other hand, the recursive algorithm needs both the addition and extraction steps for all $k > 1$. When $P_v = 100\%$, neither the addition nor the extraction step is needed in either algorithm. Thus, they both have the same performance.

As P_v becomes closer to 100% the efficiency gains for both algorithms increase, with the incremental algorithm broadly superior to the recursive algorithm, when $\mathbf{p}_A$ does not change over time and $P_v \ge 80\%$. This fits the objective of more efficient processing in cases where some but not all of $\mathbf{p}_A$ is received in each window, $\mathbf{p}_k \subset \mathbf{p}_A$.

4.2.1. Scalability

Although neither the incremental algorithm nor the recursive algorithm alters the overall $O(n^3)$ worst-case scalability in cases where $P_v < 100\%$, the algorithms reduce the size of the matrix computations, and so increase the average-case scalability. Table 2 compares and summarizes the average-case scalability of the incremental algorithm and the recursive algorithm across the experiments. For each algorithm and value of P_v, a polynomial regression of the form $y = ax^b$ was fitted to the average-case scalability curves (i.e., measured execution time in seconds as a function of $|\mathbf{p}_k|$). In all cases, the coefficient of determination (R^2) indicated a good fit ($R^2 > 0.99$ in all cases). The exponent of the curve b provides an estimate of the average case scalability. Table 2 shows that as expected, average-case scalability is always better than worst case scalability. The difference is more pronounced for the incremental algorithm, although both algorithms achieve average-case scalability approaching $O(n^2)$ for cases where P_v is high (e.g., $P_v = 98\%$ yields average-case scalability of $O(n^{2.162})$ for incremental and $O(n^{2.092})$ for recursive algorithms).

4.3. Efficiency of incremental and recursive algorithms under changing $\mathbf{p}_A$

As discussed previously, the incremental algorithm is not suitable for changing $\mathbf{p}_A$. This can be illustrated through a simple experiment, the results of which are summarized in Fig. 5. In the second experiment, the set up of the scalar surfaces, sampling cycles, and target points are the same as those in the previous experiment. But $\mathbf{p}_A$ is manipulated to be time-varying. At the first trigger of the window, the window contains 600 non-coincident sample points ($|\mathbf{p}_1| = 600$). From the $(k-1)$th to the kth trigger of the window for all $k > 1$, 20 nodes were removed and 20 new nodes were added, i.e., $|\bar{\mathbf{p}}'_{k-1}| = |\bar{\mathbf{p}}''_k| = 20$. Ordinary Kriging interpolation is conducted over $\mathbf{p}_k$ for $k=1,2,\ldots,41$ using the incremental algorithm and the recursive algorithm. In Fig. 5, the execution time of these two algorithms is compared. Because 20 new nodes are incorporated at every trigger of the window for $k > 1$, $|\mathbf{p}_{1:k}|$ increases by 20 at every trigger in the incremental algorithm. As a result, the execution time of the addition and the extraction steps of the incremental algorithm keeps increasing. For $k > 12$, the execution time becomes even longer than RT_1. In contrast, the execution time RT_k of the recursive algorithm $\forall\, k > 1$ is shorter than RT_1, and it is stable because $|\mathbf{p}_k|$ does not change over time.

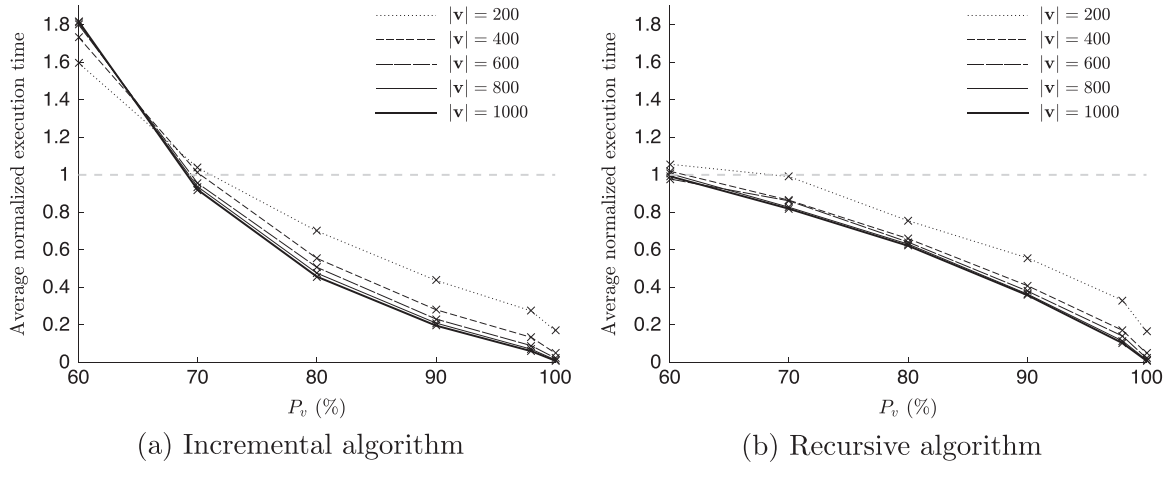

Fig. 4. Comparison of the normalized execution time (ratio between mean RT_k and mean RT_1) of the incremental algorithm (a) and the recursive algorithm (b) for average of $k \in \{7, .., 10\}$ windows for $P_v \in \{60, 70, 80, 90, 98, 100\%\}$ and $|\mathbf{p}| \in \{200, 400, 600, 800, 1000\}$. (a) Incremental algorithm. (b) Recursive algorithm.

Table 2
Coefficients for regressions of the form $y = ax^b$ upon scalability (growth in execution time in seconds as a function of $|\mathbf{p}_k|$).

P_v (%)	Incremental			Recursive		
	a	b	R^2	a	b	R^2
60	2.2×10^{-7}	2.932	0.99996	1.5×10^{-7}	2.896	0.99995
70	1.7×10^{-7}	2.872	0.99995	1.5×10^{-7}	2.868	0.99998
80	1.2×10^{-7}	2.819	0.99990	1.0×10^{-7}	2.886	0.99996
90	2.5×10^{-7}	2.667	0.99979	8.4×10^{-7}	2.839	0.99991
98	1.5×10^{-7}	2.162	0.99926	9.5×10^{-7}	2.029	0.99828
100	3.8×10^{-7}	1.080	0.99955	4.0×10^{-7}	1.078	0.99985

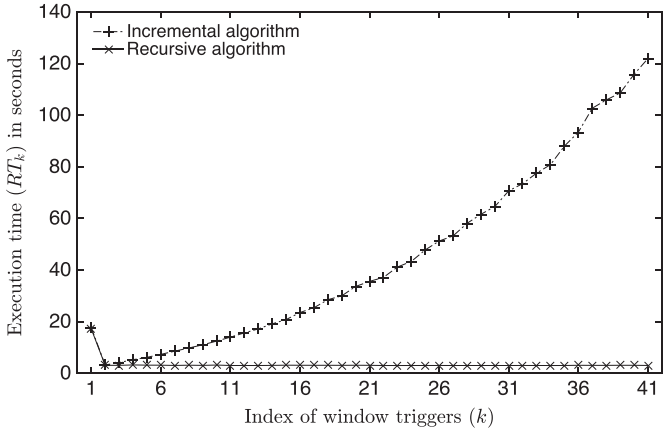

Fig. 5. Comparison of execution time of the incremental algorithm and the recursive algorithm when $\mathbf{p}_A$ continually changes. At the first trigger of the window, the window contains 600 non-coincident source points ($|\mathbf{p}_1|=600$). From the $(k-1)$th to the kth trigger of the window for all $k > 1$, 20 nodes are removed and 20 new nodes are added, i.e., $|\bar{\mathbf{p}}'_{k-1}| = |\bar{\mathbf{p}}''_k| = 20$.

5. Discussion and future work

This section discusses the applicability of the incremental algorithm and the recursive algorithm when the assumptions described in Section 2.2.2 are not satisfied, and potential further improvement of the efficiency of ordinary Kriging.

5.1. Extending to sliding temporal windows

The discussions above frame the problem, common in streaming data, where incomplete data are received on the stream, requiring computation with different but overlapping subsets of locations in consecutive windows. In order to satisfy the assumption $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$, $\forall\, k > 1$ for the incremental algorithm and the assumptions $|\bar{\mathbf{p}}'_{k-1}| \ll |\mathbf{p}_k|$ and $|\bar{\mathbf{p}}''_k| \ll |\mathbf{p}_k|$, $\forall\, k > 1$ for the recursive algorithm, the data sources that generate the spatiotemporal streams are assumed to be immobile in these discussions. Although today's sensor technology most frequently involves immobile sensor nodes, mobility is increasingly common. In fact, the assumptions for the recursive algorithm in certain cases may hold even for such mobile data sources.

Specifically, for moving source points, a sliding temporal window also may result in $\mathbf{p}_{k-1} \neq \mathbf{p}_k$ and $|\ddot{\mathbf{p}}_{k-1}| \gg 0$. Fig. 6 shows a count-based sliding window with window size w and trigger size s. The window maintains its size to be w by evicting a tuple when a new tuple is inserted. $\mathbf{p}_k$ can be maintained as a set of pointers to the tuples in the window k. Spatial interpolation is conducted over $\mathbf{p}_k$ when the trigger policy of the window (i.e., s tuples inserted) is satisfied. The tuples that are not evicted between windows $k-1$ and k represent the overlapping set of source points that make up

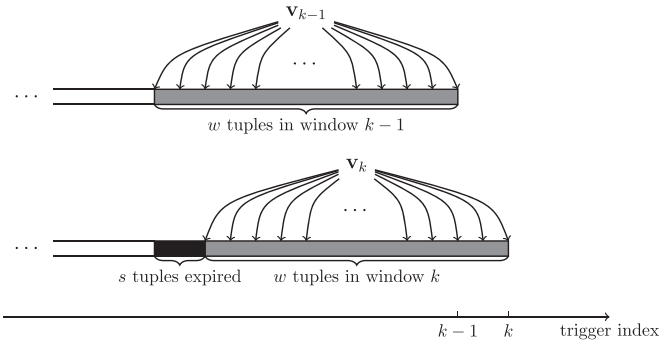

Fig. 6. An example of a count-based sliding window over moving objects with extent size w and step size s.

$\mathbf{p}_{k-1}$ and $\mathbf{p}_k$.

For the sliding window shown in Fig. 6, at the window trigger event, the extraction step is firstly run to obtain the inverse of the covariance matrix ($\mathbf{C}^{-1}_{\bar{\mathbf{p}}_{k-1}\bar{\mathbf{p}}_{k-1}}$) of the non-coincident tuples that are still in the window ($\bar{\mathbf{p}}_{k-1}$) at t_k. Then the addition step is executed to incorporate $\bar{\mathbf{p}}''_k$ in the s inserted tuples into the interpolation system, which provides $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ for generating the interpolation results. The effectiveness of this scheme depends on the relationship between s and w and the occurrence of coincident tuples. If the tuples in the stream are rarely coincident and $s \ll w$, this scheme can be more efficient than processing the source points in the window at every trigger as an entirely new batch. For example, as shown in Fig. 5, this scheme can be as about 5.5 times as efficient as batch interpolation when there are no coincident tuples in the stream, $s=20$, and $w=600$. The same procedure can be applied to other variants of sliding temporal windows as well, such as time-based and delta-based sliding windows. For other variants of sliding windows, the effectiveness of this scheme can also be affected by changes in the data rate.

5.2. Extending to dynamic covariance models

In the previous sections, the spatial covariance model is assumed not to change over time to deduce $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$ from $\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}$ or $\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}$ $\forall\, k > 1$. In fact, the incremental algorithm and the recursive algorithm can handle dynamic covariance models if the temporal evolution of the covariance model satisfy the following two criteria:

1. the function of the model does not change but only the parameters $\boldsymbol{\theta}$ change, and
2. the ratio of the preceding covariance to the current covariance between any two points is independent on the locations of the points, i.e., $\mathrm{Cov}(p_i,\ p_j;\ \boldsymbol{\theta}_{k-1}) = d(\boldsymbol{\theta}_{k-1},\ \boldsymbol{\theta}_k)\mathrm{Cov}(p_i,\ p_j;\ \boldsymbol{\theta}_k)$, $\forall\ p_i$ and p_j, where d is a scalar function independent on p_i or p_j.

These two assumptions are often satisfied in practice. For example, when dynamic stationary Gaussian random fields are interpolated, it is common to use the same form of kernel function to model the spatial variation of the fields (Kerwin and Prince, 1999a, 1999b). This satisfies the first criterion. The variation of the spatial model is captured by fitting the kernel function to the variograms of the fields, which leads to different parameterizations of the kernel function. Many well-established kernel functions such as the exponential, Gaussian, and power kernel functions satisfy the second criterion when parameterization is changed. In this situation, the changes in the covariance matrix between two points sets $\mathbf{p}$ and $\mathbf{q}$ can be written as:

$$\mathbf{C}_{\mathbf{pq}}(\theta_{k-1}) = d(\theta_{k-1},\ \theta_k)\mathbf{C}_{\mathbf{pq}}(\theta_k). \tag{20}$$

Hence when the parameters of the covariance model changes, the inverse of the preceding covariance matrix can be easily updated by:

$$\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}(\theta_k) = d(\theta_{k-1},\ \theta_k)\mathbf{C}^{-1}_{\mathbf{p}_{1:k-1}\mathbf{p}_{1:k-1}}(\theta_{k-1}),$$
$$\text{for the incremental algorithm,} \tag{21}$$

$$\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}(\theta_k) = d(\theta_{k-1},\ \theta_k)\mathbf{C}^{-1}_{\mathbf{p}_{k-1}\mathbf{p}_{k-1}}(\theta_{k-1}),$$
$$\text{for the recursive algorithm.} \tag{22}$$

Then $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}(\theta_k)$ can be calculated using the previous two algorithms with the inverse of the updated preceding covariance matrix.

5.3. Combining our algorithms with modified ordinary Kriging algorithms

In both the addition and extraction steps of our algorithms, one matrix inverse needs to be calculated. In addition, $\mathbf{C}^{-1}_{\mathbf{p}_1\mathbf{p}_1}$ needs to be computed in the batch fashion. These are the major computational and space bottlenecks of our algorithms towards online ordinary Kriging spatial interpolation. There are several modified ordinary Kriging algorithms available which can compute the approximated inverse of the covariance matrix efficiently and accurately (see Section 1). These approximated algorithms can be combined with our algorithms to further reduce execution time. They can be adopted to calculate $\mathbf{C}^{-1}_{\mathbf{p}_1\mathbf{p}_1}$ and the matrix inverse in the addition step. In the future works, a scalable online ordinary Kriging spatial interpolation operator will be investigated by combining the incremental and recursive algorithms developed in this paper with currently available modified ordinary Kriging algorithms.

6. Conclusion

An incremental algorithm and a recursive algorithm to conduct ordinary Kriging interpolation over spatiotemporal data streams were proposed and evaluated in this paper. Computing $\mathbf{C}^{-1}_{\mathbf{p}_k\mathbf{p}_k}$, $\forall\, k > 1$, with the incremental algorithm is more efficient than inverting the entire $\mathbf{C}_{\mathbf{p}_k\mathbf{p}_k}$ when $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$. When the size of $\mathbf{p}_{1:k}$ keeps increasing over time, the assumption $|\bar{\mathbf{p}}_{1:k}| \ll |\mathbf{p}_k|$ can become invalid as k increases. In this case, if the assumptions $|\bar{\mathbf{p}}'_{k-1}| \ll |\mathbf{p}_k|$ and $|\bar{\mathbf{p}}''_k| \ll |\mathbf{p}_k|$, $\forall\, k > 1$ can be satisfied, the recursive algorithm can deliver lower computational load than batch Kriging computation.

The results demonstrate the significant average-case efficiency gains that can be accrued in cases where data from part but not all of $\mathbf{p}_A$ is expected to be received in any given window. This situation is endemic to stream processing data sources such as wireless sensor networks, where node movements or replacements are typically infrequent, but intermittent node and communication failures are commonplace. In addition, the applicability of our algorithms under sliding temporal windows or dynamic covariance models is discussed. Based on our algorithms, further investigation for developing a scalable online ordinary Kriging spatial interpolation operator is also described.

Acknowledgments

This work is fully funded by the ARC Linkage Project (Grant no. LP120200584) “Resilient Information Systems for Emergency Response (RISER)”.

References

Acharya, S., Lee, B.S., 2014. Incremental causal network construction over event streams. Inf. Sci. 261, 32–51.
Albers, S., 2003. Online algorithms: a survey. Math. Program. 97 (12), 3–26.
Ali, M.H., Mokbel, M.F., Aref, W.G., Kamel, I., 2005. Detection and tracking of discrete phenomena in sensor-network databases. In: SSDBM. pp. 163–172.
Ali, M.H., Aref, W.G., Kamel, I., 2006. Scalability management in sensor-network phenomenabases. In: 18th International Conference on Scientific and Statistical Database Management, Vienna, 2006. IEEE, pp. 91–100.
Ali, M.H., Mokbel, M.F., Aref, W.G., 2007. Phenomenon-aware stream query processing. In: 2007 International Conference on Mobile Data Management, Mannheim, IEEE, pp. 8–15.
Appice, A., Ciampi, A., Malerba, D., Guccione, P., 2015. Using trend clusters for spatiotemporal interpolation of missing data in a sensor network. J. Spat. Inf. Sci. (6), 119–153.
Banerjee, S., Gelfand, A.E., Finley, A.O., Sang, H., 2008. Gaussian predictive process models for large spatial data sets. J. R. Stat. Soc.: Ser. B (Stat. Methodol.) 70 (4), 825–848.
Cheng, T., 2013. Accelerating universal Kriging interpolation algorithm using cuda-enabled gpu. Comput. Geosci. 54, 178–183.
Cressie, N., 1992. Statistics for spatial data. Terra Nova 4 (5), 613–617.
Cressie, N., Johannesson, G., 2008. Fixed rank Kriging for very large spatial data sets. J. R. Stat. Soc.: Ser. B (Stat. Methodol.) 70 (1), 209–226.
De Baar, J., Dwight, R.P., Bijl, H., 2013. Speeding up Kriging through fast estimation of the hyperparameters in the frequency-domain. Comput. Geosci. 54, 99–106.
de Ravé, E.G., Jiménez-Hornero, F.J., Ariza-Villaverde, A.B., Gómez-López, J., 2014. Using general-purpose computing on graphics processing units (gpgpu) to accelerate the ordinary Kriging algorithm. Comput. Geosci. 64, 1–6.
Falivene, O., Cabrera, L., Tolosana-Delgado, R., Sáez, A., 2010. Interpolation algorithm ranking using cross-validation and the role of smoothing effect. A coal zone example. Comput. Geosci. 36 (4), 512–519.
Fritz, J., Neuweiler, I., Nowak, W., 2009. Application of fft-based algorithms for large-scale universal kriging problems. Math. Geosci. 41 (5), 509–533.
Furrer, R., Genton, M.G., Nychka, D., 2006. Covariance tapering for interpolation of large spatial datasets. J. Comput. Graph. Stat. 15 (3).
Guan, Q., Kyriakidis, P.C., Goodchild, M.F., 2011. A parallel computing approach to fast geostatistical areal interpolation. Int. J. Geogr. Inf. Sci. 25 (8), 1241–1267.
Guccione, P., Appice, A., Ciampi, A., Malerba, D., 2012. Trend cluster based Kriging interpolation in sensor data networks. In: Modeling and Mining Ubiquitous Social Media. Springer, Boston, pp. 118–137.
Hartman, L., Hössjer, O., 2008. Fast Kriging of large data sets with Gaussian Markov random fields. Comput. Stat. Data Anal. 52 (5), 2331–2349.
Hessami, M., Anctil, F., Viau, A.A., 2001. Delaunay implementation to improve Kriging computing efficiency. Comput. Geosci. 27 (2), 237–240.
Hu, H., Shu, H., 2015. An improved coarse-grained parallel algorithm for computational acceleration of ordinary kriging interpolation. Comput. Geosci. 78, 44–52.
Jin, R., Goswami, A., Agrawal, G., 2006. Fast and exact out-of-core and distributed k-means clustering. Knowl. Inf. Syst. 10 (1), 17–40.
Jung, H., Kim, Y.S., Chung, Y.D., 2014. Qr-tree: an efficient and scalable method for evaluation of continuous range queries. Inf. Sci. 274, 156–176.
Kamel, I., Al Aghbari, Z., Awad, T., 2010. Mg-join: detecting phenomena and their correlation in high dimensional data streams. Distrib. Parallel Databases 28 (1), 67–92.
Kerwin, W.S., Prince, J.L., 1999a. The kriging update model and recursive space-time function estimation. IEEE Trans. Signal Process. 47 (11), 2942–2952.
Kerwin, W.S., Prince, J.L., 1999b. Tracking mr tag surfaces using a spatiotemporal filter and interpolator. Int. J. Imag. Syst. Technol. 10 (2), 128–142.
Li, L., Zhang, X., Holt, J.B., Tian, J., Piltner, R., 2011. Spatiotemporal interpolation methods for air pollution exposure. In: SARA.
Lu, G.Y., Wong, D.W., 2008. An adaptive inverse-distance weighting spatial interpolation technique. Comput. Geosci. 34 (9), 1044–1055.
Memarsadeghi, N., Raykar, V. C., Duraiswami, R., Mount, D.M., 2008. Efficient kriging via fast matrix-vector products. In: Aerospace Conference, Big Sky, MT, 2008 IEEE. IEEE, pp. 1–7.
Mokbel, M.F., Xiong, X., Hammad, M.A., Aref, W.G., 2005. Continuous query processing of spatio-temporal data streams in place. GeoInformatica 9 (4), 343–365.
Myers, D.E., 1992. Kriging, cokriging, radial basis functions and the role of positive definiteness. Comput. Math. Appl. 24 (12), 139–148.
Nittel, S., Leung, K.T., 2004. Parallelizing clustering of geoscientific data sets using data streams. In: 16th International Conference on Scientific and Statistical Database Management, Santorini Island, 2004. Proceedings. IEEE, pp. 73–84.
Paige, C.C., Saunders, M.A., 1975. Solution of sparse indefinite systems of linear equations. SIAM J. Numer. Anal. 12 (4), 617–629.
Pebesma, E.J., 2004. Multivariable geostatistics in s: the gstat package. Comput. Geosci. 30 (7), 683–691.
Pesquer, L., Cortés, A., Pons, X., 2011. Parallel ordinary kriging interpolation incorporating automatic variogram fitting. Comput. Geosci. 37 (4), 464–473.
Rivoirard, J., Romary, T., 2011. Continuity for kriging with moving neighborhood. Math. Geosci. 43 (4), 469–481.
Romary, T., 2013. Incomplete Cholesky decomposition for the kriging of large datasets. Spat. Stat. 5, 85–99.
Sakata, S., Ashida, F., Zako, M., 2004. An efficient algorithm for Kriging

approximation and optimization with large-scale sampling data. Comput. Methods Appl. Mech. Eng. 193 (3), 385–404.
Sang, H., Huang, J.Z., 2012. A full scale approximation of covariance functions for large spatial data sets. J. R. Stat. Soc.: Ser. B (Stat. Methodol.) 74 (1), 111–132.
Stein, M.L., 2002. The screening effect in Kriging. Ann. Stat., 298–323.
Tolhurst, K., Shields, B., Chong, D., 2008. Phoenix: development and application of a bushfire risk management tool. Emerg. Manag. Aust. 23 (4), 47–54.
Umer, M., Kulik, L., Tanin, E., 2008. Kriging for localized spatial interpolation in sensor networks. In: Scientific and Statistical Database Management. Springer, Hong Kong, pp. 525–532.
Ünal, A., Saygín, Y., Ulusoy, Ö, 2006. Processing count queries over event streams at multiple time granularities. Inf. Sci. 176 (14), 2066–2096.
Vargas-Guzmán, J., Yeh, T.-C.J., 1999. Sequential Kriging and Cokriging: two powerful geostatistical approaches. Stoch. Environ. Res. Risk Assess. 13 (6), 416–435.
Wackernagel, H., 1995. Variogram and covariance function. In: Multivariate Geostatistics. Springer, Berlin Heidelberg, pp. 35–40.
Zhang, X., Furtlehner, C., Germain-Renaud, C., Sebag, M., July 2014. Data stream clustering with affinity propagation. IEEE Trans. Knowl. Data Eng. 26 (7), 1644–1656.
Zhang, X., Furtlehner, C., Sebag, M., 2008. Data streaming with affinity propagation. In: Machine Learning and Knowledge Discovery in Databases. Springer, Antwerp, pp. 628–643.
Zhong, X., Kealy, A., Sharon, G., Duckham, M., 2015. Spatial interpolation of streaming geosensor network data in the riser system. In: Web and Wireless Geographical Information Systems. Springer, Grenoble, pp. 161–177.