Combining Filtering and Cross-Correlation Efficiently for Streaming Time Series

SHENG ZHONG, University of New Mexico
VINICIUS M. A. SOUZA, Pontifícia Universidade Católica do Paraná
ABDULLAH MUEEN, University of New Mexico

Monitoring systems have hundreds or thousands of distributed sensors gathering and transmitting real-time streaming data. The early detection of events in these systems, such as an earthquake in a seismic monitoring system, is the base for essential tasks as warning generations. To detect such events is usual to compute pairwise correlation across the disparate signals generated by the sensors. Since the data sources (e.g., sensors) are spatially separated, it is essential to consider the lagged correlation between the signals. Besides, many applications require to process a specific band of frequencies depending on the event's type, demanding a pre-processing step of filtering before computing correlations. Due to the high speed of data generation and a large number of sensors in these systems, the operations of filtering and lagged cross-correlation need to be efficient to provide real-time responses without data losses. This article proposes a technique named FilCorr that efficiently computes both operations in one single step. We achieve an order of magnitude speedup by maintaining frequency transforms over sliding windows. Our method is exact, devoid of sensitive parameters, and easily parallelizable. Besides our algorithm, we also provide a publicly available real-time system named Seisviz that employs FilCorr in its core mechanism for monitoring a seismometer network. We demonstrate that our technique is suitable for several monitoring applications as seismic signal monitoring, motion monitoring, and neural activity monitoring.

CCS Concepts: • **Information systems → Data streaming;**

Additional Key Words and Phrases: Time series, correlation, filtering, streaming, seismic monitoring, real-time monitoring systems

ACM Reference format:
Sheng Zhong, Vinicius M. A. Souza, and Abdullah Mueen. 2022. Combining Filtering and Cross-Correlation Efficiently for Streaming Time Series. *ACM Trans. Knowl. Discov. Data.* 16, 5, Article 100 (May 2022), 24 pages.
https://doi.org/10.1145/3502738

1 INTRODUCTION

Large monitoring systems such as a network of seismic stations or forest fire detection systems typically have hundreds of distributed sensors gathering and transmitting real-time and streaming

This material is based upon work supported by the National Science Foundation under #OIA-1757207 and #CNS-2008910.
Authors' addresses: S. Zhong and A. Mueen, University of New Mexico, Albuquerque, New Mexico 87131; emails: {zhongs, mueen}@unm.edu; V. M. A. Souza, Pontifícia Universidade Católica do Paraná, Curitiba, Paraná, Brazil; email: vinicius@ppgia.pucpr.br.

1556-4681/2022/05-ART100 $15.00
https://doi.org/10.1145/3502738

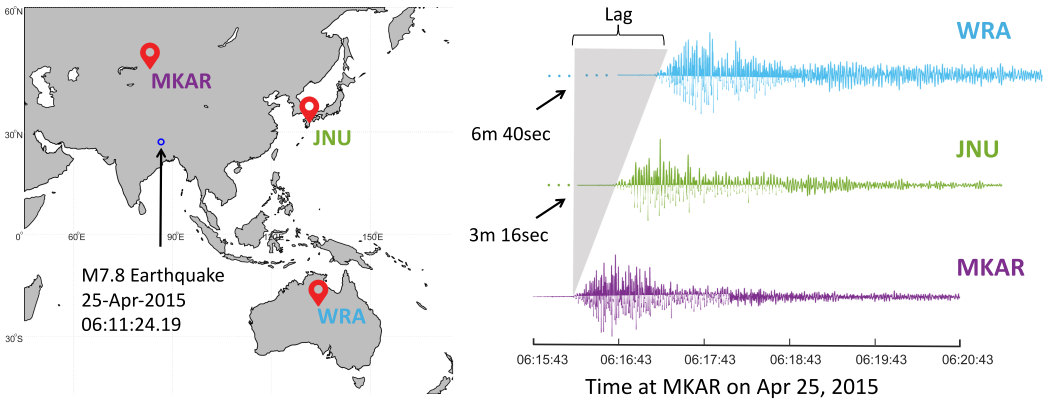

Fig. 1. *(left)* A M7.8 earthquake in Gurkha, Nepal, on Apr-25, 2015. *(right)* The signals recorded at three different stations at different times due to the spatial separation of sensors. Data collected from IRIS [10].

data. An event in these systems, like an earthquake or an abnormal higher temperature, creates dynamic responses at these sensors. The responses can be arbitrarily lagged because of sensors' spatial separation and be limited to a specific band of frequencies depending on the type of event. For these systems, real-time event detection is essential for decision-making or alert generation. *We demonstrate an algorithm to correlate streaming data generated from distributed sensors in real-time to detect events.*

A concrete application where our algorithm can be employed is seismic monitoring. In this application, when an earthquake happens, seismometers (i.e., seismic sensors) across the earthquake region observe the wave at varying times for varying duration, while the signals recorded at these sensors are often correlated. For better understanding, in Figure 1, we show a magnitude 7.8 event in Gurkha, Nepal, on April 25, 2015. Three waveforms recorded at three stations (marked red in the map) show a high correlation when the lag due to propagation delay is considered. The seismic wave generated in Nepal reaches Japan in about eight minutes and Australia in about 11 minutes.

In addition, filtering is a mandatory operation in seismic signal processing to remove undesired noise from data for events. We illustrate the importance of filtering in Figure 2. The raw waveforms rarely demonstrate a correlation between events. In contrast, waveforms filtered between 0.4 Hz to 3 Hz achieve a higher correlation. For many applications, mainly those involving digital signal processing, certain properties of data are made explicit when the signal is represented in the frequency domain [25]. Thus, filtering is also essential to extract descriptive features for machine learning models.

In a monitoring system that consumes streaming data from hundreds of sensors, computing lagged correlation (or asynchronous correlation) can be challenging because of the fast data rate, a large number of stations, and the necessity for accurate correlation values. Although the problem has been studied for decades, none of the existing methods such as BRAID [21], COLR-tree [1], or StatStream [30]), can monitor one hundred seismometers at a 10 Hz rate on a single workstation, a usual setting for many systems. The main reason for the failure of these methods is the data-dependent pruning, projection, or indexing technique. Seismic traces are mostly white noise (except when events happen), stressing the algorithms falling behind the streams. In contrast, none of these algorithms are amenable to requirements of data pre-processing, such as filtering. Hence, pre-processing must always be done before correlation computation, resulting in a loss of efficiency.

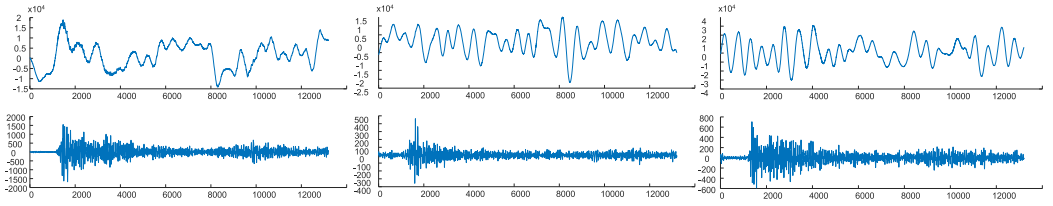

Fig. 2. The top row shows raw waveforms with no clear seismic signal. The bottom row shows filtered waveforms of the corresponding raw waveforms in the top row. Note the decrease in absolute value in the y-axis and the increased visibility of the signal. The filter band is 0.4 Hz to 3 Hz. The average lagged correlation of all three pairs of raw signals is zero. The average lagged correlation of all three pairs after filtering is 0.53.

In this aricle, we propose an algorithm, **FilCorr**—Filtered Lagged Correlation, to merge filtering and lagged correlation computation to extract *data-independent* efficiency. Our algorithm efficiently maintains frequency information over the stream and calculates correlation in the frequency domain. The technique is *exact*, *devoid of sensitive parameters* and *easily parallelizable*. FilCorr was first introduced in our previous work [28]. We experimentally show that our technique is suitable for monitoring applications where the lagged correlation of filtered time series is required, such as seismic event monitoring, motion monitoring, and neural activity monitoring. This article introduces a new real-time seismic events monitoring system (SeisViz) that exploits the FilCorr algorithm to monitor high-speed data streams. Our implementation can monitor one hundred seismic stations at a 10 Hz rate without any delay. A demonstration of Seisviz capturing past earthquakes is available on our website [29], and the system for real-time monitoring is available at www.seisviz.com. This extended version explains algorithmic details to support its implementation using different programming languages. We perform new experiments concerning the parameter sensitivity of FilCorr and discuss their parallelization on multiple processing units and how FilCorr can work with other types of filters. Additionally, we introduce two new case studies on neuroimaging and motion-capture data, demonstrating the applicability of FilCorr.

The main contributions of this work are summarized below:

— We demonstrate a working system to cross-correlate hundreds of high-speed streams in real-time at 10 Hz speed using a conventional workstation;
— We merge digital filters and correlation computation in one combined step to achieve time and space efficiency;
— We show three case studies in which such high-speed lagged correlation help to detect events of interest.

The rest of the article is organized as follows. In Section 2, we present the general definitions related to streaming time series and notation employed throughout the article. In Section 3, we discuss related work. In Section 4, we introduce our proposed algorithm FilCorr. In Section 5, we discuss the performance of our proposed algorithm. In Section 6, we present three case studies in which FilCorr is employed to compute the lagged correlation of filtered data. Finally, our conclusions and future works are presented in Section 7.

2 BACKGROUND AND NOTATION

We define *time series* as a sequence of observations, and they are in the form of real numbers measuring a quantity at a fixed sampling rate. A *streaming time series* is an unbounded sequence of observations generated at a fixed rate. Table 1 shows the symbols and definitions considered in our discussions. The relationships of main variables are also illustrated in Figure 3. We use s^x to

Table 1. Symbols and Definitions

Symbol	Definition
s^x	Streaming time series with ID x
N	Total number of streams that a system is processing
$s^x[t]$	The observation value of stream s^x at time t
f^x	Sampling rate of stream s^x
f_s	Lower bound frequency of the bandpass filter in Hz
f_t	Upper bound frequency of the bandpass filter in Hz
m	Length of basic windows in number of observations
l	Lag size in number of observations
w_t^x	The **basic window** in s^x at time t, includes m observations: $\{s^x[t-m+1]\ldots s^x[t]\}$
$w\prime_t^x$	The **filtered basic window**, filtered version of w_t^x in the time domain
W_t^x	The **frequency window**, discrete Fourier transformed version of w_t^x
$W_t^x\|_{f_s}^{f_t}$	The **filtered frequency window** of W_t^x which only contains coefficients corresponding to frequencies from f_s to f_t
s_t^x	The **sliding window** in s^x at time t, covers observations in range $\{s^x[t-l-m+1]..s^x[t]\}$ includes $(l+1)$ basic windows: $\{w_{t-l}^x..w_t^x\}$
$w_t^x[j]$	jth element of the basic window w_t^x, $w_t^x[j] = s^x[t-m+1+j]$
$lcorr_t^{xy}$	Lagged correlation value between s_t^x and s_t^y at time t
$corr_{t_1t_2}^{xy}$	Correlation value of $w_{t_1}^x$ and $w_{t_2}^y$
$step$	Gap between two successive correlation computation in number of observations. If $step = 10$ and the first output is $lcorr_1^{xy}$ then the next output is $lcorr_{11}^{xy}$

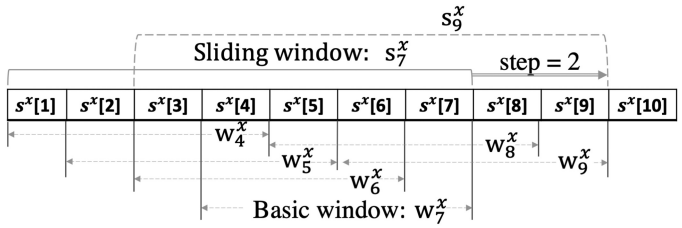

Fig. 3. Examples of various windows on a stream, where $m = 4, l = 3$, and $step = 2$.

represent a particular streaming time series x. We define the *basic window* as a continuous segment of a time series using w^x. A basic window of size m from time series s^x at time t is denoted by w_t^x which contains observations from $s^x[t-m+1]$ to $s^x[t]$. We can extract at most $n-m+1$ basic windows of length m from a long time series of length $n >> m$. Adjacent basic windows are not independent of each other, overlapping basic windows are trivially close in the high dimensional space.

Filtering: An essential pre-processing task for time series required by many applications. Filters are most commonly used to remove undesirable components from a time series. There are many types of filters useful in various domains. All filters have response functions convolved with a signal to apply the filter. Such response functions contain relative weight for each frequency. In an analytical form, the weights are non-zero for all frequencies. However, many weights are significantly smaller than the rest in practice, allowing us to cut off and keep only the necessary frequencies. Most applications employ a band that focuses on the need of the application. For example, seismic monitoring uses up to 10 Hz [23], and EEG monitoring uses a gamma activity band between 30 Hz–50 Hz [5]. In this work, we assume that the given filter band is a contiguous set of frequencies, and all frequencies are equally important. This is also known as a box filter or

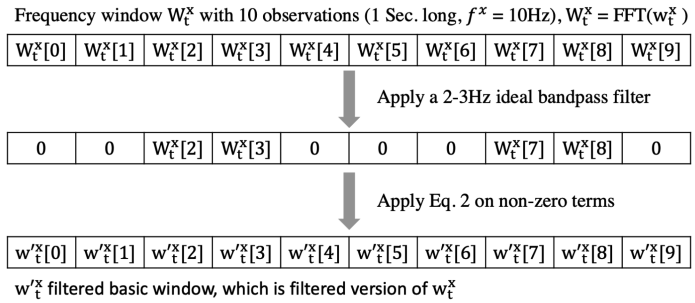

Fig. 4. Running example of how to derive the filtered frequency window and filtered basic window.

ideal bandpass filter. Our proposed method is extendable to a non-contiguous set of frequencies with varying weights, i.e., other types of filters. We define the *frequency window* W_t^x as a window that contains all the frequency components of the basic window w_t^x, and each $W_t^x[k]$ where $k = 0, 1, 2, \ldots, m-1$ are defined in Equation (1). Note $i = \sqrt{-1}$, $W_t^x[k]$ is a complex number.

$$W_t^x[k] = \sum_{j=0}^{m-1} w_t^x[j] e^{\frac{-i2\pi}{m} kj}. \tag{1}$$

We use f^x in Hz to represent the sampling rate of the streaming time series s^x and the *filtered basic window* $w_t^x{}'$ to denote the filtered version of w_t^x. If we apply a box filter with a band from f_s to f_t in Hz ($0 < f_s \leq f_t$), then $w_t^x{}'$ can be derived from Equation (2):

$$w_t^x{}'[j] = \frac{1}{m}\left(\sum_{k=\lfloor \frac{f_t}{f} m \rfloor}^{\lfloor \frac{f_t}{f} m \rfloor} W_t^x[k] e^{\frac{i2\pi}{m} kj} + \sum_{k=m-\lfloor \frac{f_t}{f} m \rfloor}^{m-\lfloor \frac{f_s}{f} m \rfloor} W_t^x[k] e^{\frac{i2\pi}{m} kj} \right); \tag{2}$$

Figure 4 shows a running example of how to derive the $W_t^x|_{f_s}^{f_t}$ and $w_t^x{}'$. Note that there is a corner case when m is even and $f_t = \frac{f}{2}$, then $w_t^x{}'[j]$ is equals to $\frac{1}{m}(\sum_{k=\lfloor mfs/f \rfloor}^{k=m-\lfloor mfs/f \rfloor} W_t^x[k] e^{\frac{i2\pi}{m} kj})$. The Fourier transform of $w_t^x{}'$ will only contains non-zero components that corresponding to frequency from f_s to f_t in W_t^x. If $f_s = 0$ and $f_t = \frac{f}{2}$ (Nyquist frequency), then $w_t^x{}'$ is identical to the basic window w_t^x.

Correlation computation in the frequency domain: If we are given two basic windows $w_{t_1}^x$ and $w_{t_2}^y$, Pearson's correlation coefficient between them is defined in the time domain as Equation (3):

$$corr_{t_1 t_2}^{xy} = \frac{\sum_{j=0}^{m-1} w_{t_1}^x[j] w_{t_2}^y[j] - m\mu(w_{t_1}^x)\mu(w_{t_2}^y)}{m\sigma(w_{t_1}^x)\sigma(w_{t_2}^y)}. \tag{3}$$

To compute the correlation in the frequency domain, we can exploit Parseval's theorem [16, 18], which is expressed as Equation (4) for the discrete Fourier transformation:

$$\sum_{j=0}^{m-1} |w_t^x[j]|^2 = \frac{1}{m} \sum_{k=0}^{m-1} |W_t^x[k]|^2. \tag{4}$$

Based on Equation (4), we can compute Equation (3) in the frequency domain: $\mu(w_t^x)$ equals $\frac{W_t^x[0]}{m}$ since $W_t^x[0] = \sum w_t^x[j]$. $\sigma(w_t^x)$ equals $\sqrt{\frac{\sum w_t^x[j]^2}{m} - [\mu(w_t^x)]^2}$ in the time domain and the $\sum w_t^x[j]^2$ term can be computed in the frequency domain by directly applying Equation (4). Since

the discrete Fourier transform is a linear operation, Parseval's theorem can also be expressed as Equation (5) with two signals.

$$\sum_{j=0}^{m-1} |w_t^x[j] - w_t^y[j]|^2 = \frac{1}{m} \sum_{k=0}^{m-1} |W_t^x[k] - W_t^y[k]|^2. \tag{5}$$

Based on Equation (5), we can get Equation (6) to compute $\sum_{j=0}^{m-1} w_t^x[j] w_t^y[j]$. Thus, we show how to compute each term of Equation (3) in the frequency domain.

$$\frac{1}{2m} \left(\sum_{k=0}^{m-1} |W_t^x[k]|^2 + \sum_{k=0}^{m-1} |W_t^y[k]|^2 - \sum_{k=0}^{m-1} |W_t^x[k] - W_t^y[k]|^2 \right). \tag{6}$$

Intuition: *Computing correlation values in the frequency domain is more efficient when we have a bandpass filter.* Based on Equation (2), the computation of correlation in the frequency domain only takes time $O(B)$, where B is the bandwidth in a number of frequency components.

Problem Statement: Given N streaming time series, a frequency band (f_s, f_t), and a maximum allowable lag l, compute the Pearson's correlation coefficients for all pairs of streaming time series over a sliding window up to the given lag l.

We argue in this article, with the empirical case study, this problem is very practical. In most domains, filtering can get rid of unwanted signals to compute correlation on right signals; and a reasonable maximum lag always exists, beyond which no correlation coefficient is meaningful.

3 RELATED WORK

Lagged correlation is a problem that has been studied for decades [1, 21, 30]. However, none of the existing methods consider a set of features required by modern applications that monitor hundred of sensors in real-time. Based on these requirements and FilCorr properties, we categorize the related works to our proposal according to four groups, as follows.

Lagged Correlation Computation: Computing lagged correlation, or cross-correlation, on offline data is a fundamental operation that is benefited from Fast Fourier Transform. Researchers have proposed an online algorithm for lagged correlation computation named BRAID [21]. However, the method samples frequency coefficients in a logarithmic manner to approximate the correlation value. Moreover, the method computes correlations over the entire stream, unlike our method that computes over a sliding window in real-time. For lagged correlation computation, one may consider offline indexing methods such as iSAX [24] and COLR-tree [1]. However, such offline indexes suffer from many modification (insert or delete) operations over correlation computation.

Real-Time Correlation Computation: Efficient all-pair correlation computation for streaming data is no longer an active research problem. StatStream [30] is a technique that exploits a few Fourier coefficients to prune improbable pairs quickly. Similarly, ParCorr [27] performs random projections to prune improbable pairs without redundancy due to the sliding window. Hardware-based techniques are often used for computation at MHz to GHz rate [9]. However, it is necessary to note that none of these methods consider lagged correlation after filtering.

Frequency Domain Correlation Computation: Frequency domain features are widely employed for various tasks; however, we consider our work a pre-process engine for downstream machine learning tasks, unlike many related works that use frequency domain features in machine learning tasks. For instance, **Random Interval Spectral Ensemble** (**RISE**) [2] uses spectral features for classification tasks, and various techniques from [26] utilize frequency features for deep learning. Many algorithms also exploit frequency as a computation space for dimension reduction

Table 2. Capabilities of FilCorr and Related Work

	FilCorr	ParCorr [27]	BRAID [21]	COLR-Tree [1]	StatStream [30]
Data-independent	✓	×	✓	×	×
Filtering	✓	–	×	×	–
Lagged correlation	✓	×	✓	×	×
Real-time	✓	✓	–	✓	✓
Exact	✓	✓	×	–	✓
Parallel	–	✓	–	–	–

The symbol ✓represents a claimed capability; – represents extendable capability, and × represents unknown.

purposes, such as the techniques proposed by [14, 23] calculate the correlations in the frequency domain. However, most of the works consider the offline nature of computation.

Filtered Correlation Computation: In this category, our method is unique. To the best of our knowledge, no work exploits filtering operation to extract efficiency in correlation computation. It is somewhat surprising considering the widespread filtering usage for processing real-time data captures. The novelty in our technique is that the speedup is not data-dependent, unlike any of the aforementioned work.

In Table 2, we present a comparison of the main capabilities of FilCorr concerning state-of-the-art algorithms for cross-correlation computation over multiple streaming time series. Some capabilities are not demonstrated but are trivially achievable with simple extensions of the algorithms. FilCorr comprehensively covers capabilities across several existing works, making it unique in the suite of correlation computing algorithms.

4 FILCORR: FILTERED LAGGED CORRELATION

The primary motivation of FilCorr is the need for systems to compute lagged correlation of filtered high frequency streaming time series from distributed sensors, such as seismic event monitoring. These systems require monitoring hundreds of sensors responsible for generating data at high speed. Surprisingly, none of the existing methods from the literature can deal with all of these requirements.

In this section, we present our solution in detail. For simplicity but without loss of generality, we assume all streams have the same sampling rate f, no discontinuity (no data loss during the transmission) and the observations are all aligned, which means they all have timestamps in set $\{1, 2, 3 \ldots, t-2, t-1, t, \ldots\}$.

4.1 Lagged Correlation

We use $lcorr_t^{xy}$ to denote the lagged correlation value at time t between s^x and s^y. It is defined in Equation (7).

$$lcorr_t^{xy} = Max(corr_{tt_i}^{xy}, corr_{t_it}^{xy}); t_i \in [t-l, t]. \tag{7}$$

$lcorr_t^{xy}$ is the largest Pearson's correlation coefficient value among the correlations between the most recent basic windows at time t and all the previous basic windows from time $t-l$ to t. Such a strategy can cover all possible cases when an event appears in different streams at different time.

Case 1: No lag. The event appears in both streams at the same time. $lcorr_t^{xy}$ equals to $corr_{tt}^{xy}$ which is the correlation value between basic window w_t^x and w_t^y.

Case 2: The event appears in s^y first then in s^x at time t, such scenario will be captured by computing correlation between w_t^x and $w_{t_i}^y$. $t_i \in [t-l, t-1]$.

Case 3: Similar to case 2, the event appears in s^x first then in s^y at time t. Then $lcorr_t^{xy}$ will be located among correlations between w_t^y and $w_{t_i}^x$. $t_i \in [t-l, t-1]$.

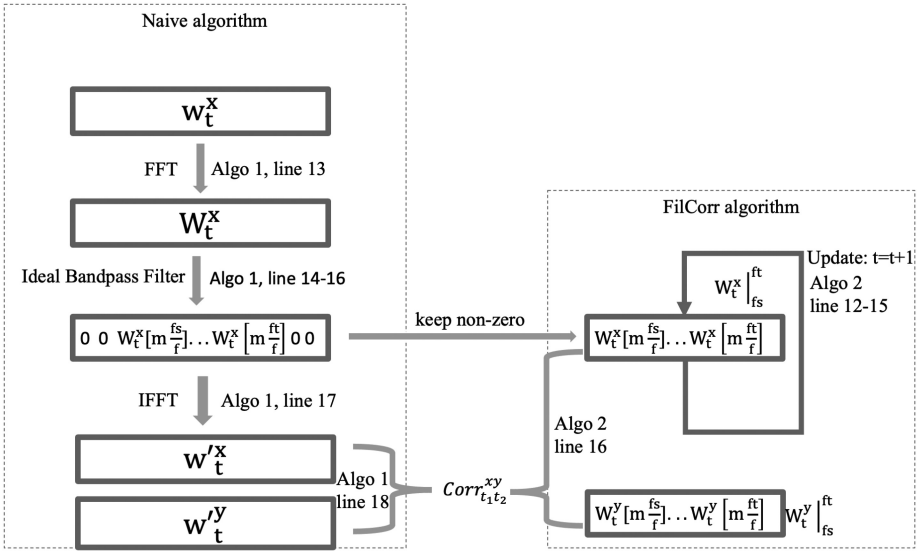

Fig. 5. Comparison between Naive algorithm (left) and FilCorr (right) for computing a correlation between two basic windows. The keep non-zero operation that bridges Naive and FilCorr only needs to perform once to initiate the FilCorr, and then it can update the filtered frequency window from the previous one.

To combine all cases, we define the *sliding window* s_t^x and s_t^y. Each sliding window will cover all $(l+1)$ basic windows that are necessary to compute $lcorr_t^{xy}$. The relations are shown in Figure 3.

To represent this process with code, we have line 3 to line 10 in Algorithm 1 and line 5 to line 11 in Algorithm 2. It takes the last basic window from one sliding window and computes correlation with all basic windows in another sliding window and vice versa. It returns the largest value as the value of the lagged correlation between the two sliding windows.

4.2 Lagged Correlation with Filtering

The following discussions are based on the ideal bandpass filter, which has frequency response values equal to 1 for frequencies in the band, and 0 for frequencies outside the band. For any other known frequency response, we can apply FilCorr trivially. Before diving into the details of algorithms, Figure 5 illustrates the high-level procedures to compute the correlation between two filtered basic windows for both Naive and FilCorr algorithms.

Naive approach: A straightforward approach to filter time series and compute lagged correlation. For the filtering, the naive approach will transform each basic window to the frequency domain, filter out frequency components outside the filter band before transforming back to the time domain. This process shows in $filter$ function with lower cutoff frequency as f_s and higher cutoff frequency as f_t in Algorithm 1.

We define the *filtered basic window* $w\prime_t^x$, as the filtered version of w_t^x. $w\prime_t^x$ and w_t^x have the same length. Unlike two successive basic windows, $w\prime_t^x$ and $w\prime_{t+1}^x$ are independent and do not share the same overlapping values.

Once all the filtered windows in s_t^x and s_t^y are calculated, we can compute $lcorr_t^{xy}$ by calling the function $LagFilterCorr$ in Algorithm 1 with parameters $(s_t^x, s_t^y, l, m, f, f_s, f_t)$. The $oneCorr$ function is based on Equation (8) derived from Equation (3) by substituting $w\prime_t^x$ for w^x.

$$corr_{t_1t_2}^{xy} = \frac{\sum_{j=0}^{m-1} w\prime_{t_1}^x[j] w\prime_{t_2}^y[j] - m\mu(w\prime_{t_1}^x)\mu(w\prime_{t_2}^y)}{m\sigma(w\prime_{t_1}^x)\sigma(w\prime_{t_2}^y)}. \tag{8}$$

We can avoid some redundant computation by storing sums, means and standard deviations of all filtered windows, which can be computed from Equations (9), (10), (11). Thus, only $\sum_{j=0}^{m-1} w\prime_{t_1}^x[j]w\prime_{t_2}^y[j]$ needs to be computed for each pair of filtered windows. In our implementation, only one copy of a filtered window $w\prime_t^x$ and its statistics are stored.

$$\sum w\prime_t^x = \sum_{j=0}^{m-1} w\prime_t^x[j], \tag{9}$$

$$\mu(w\prime_t^x) = \frac{\sum w\prime_t^x}{m}, \tag{10}$$

$$\sigma(w\prime_t^x) = \sqrt{\frac{\sum w\prime_t^x[j]^2}{m} - [\mu(w\prime_t^x)]^2}. \tag{11}$$

ALGORITHM 1: NAIVE

Function `LagFilterCorr`$(s_t^x, s_t^y, l, m, f, f_s, f_t)$
1. $w_t^x\prime \leftarrow$ `filter`(w_t^x, f, f_s, f_t)
2. $w_t^y\prime \leftarrow$ `filter`(w_t^y, f, f_s, f_t)
3. $curMax \leftarrow$ `oneCorr`$(w_t^x\prime, w_t^y\prime, m)$ `//case 1`
4. **for** $t_i = t - l : t - 1$ **do**
5. $\quad w_{t_i}^x\prime \leftarrow$ `filter`$(w_{t_i}^x, f, f_s, f_t)$
6. $\quad w_{t_i}^y\prime \leftarrow$ `filter`$(w_{t_i}^y, f, f_s, f_t)$
7. $\quad tmp1 \leftarrow$ `oneCorr`$(w_t^x\prime, w_{t_i}^y\prime, m)$ `//case 2`
8. $\quad tmp2 \leftarrow$ `oneCorr`$(w_{t_i}^x\prime, w_t^y\prime, m)$ `//case 3`
9. $\quad curMax = max(curMax, tmp1, tmp2)$

   end
10. **return** *curMax*

end

Function `filter`(w_t^x, f, f_s, f_t)
11. $LB = \lfloor mf_s/f \rfloor$
12. $UB = \lfloor mf_t/f \rfloor$
13. $W_t^x\prime \leftarrow FFT(w_t^x)$
14. $W_t^x\prime[0 : LB - 1] \leftarrow 0$
15. $W_t^x\prime[UB + 1 : m - UB - 1] \leftarrow 0$
16. $W_t^x\prime[m - LB + 1 : m - 1] \leftarrow 0$
17. **return** $IFFT(W_t^x\prime)$

end

Function `oneCorr`$(w_{t_1}^x, w_{t_2}^y, m)$
18. **return** $[\sum_j w_{t_1}^x[j]w_{t_2}^y[j] - m\mu(w_{t_1}^x)\mu(w_{t_2}^y)]/[m\sigma(w_{t_1}^x)\sigma(w_{t_2}^y)]$

end

Proposed approach: We propose combining the filtering and correlation computation in one step to speedup the computation of lagged correlation on filtered time series. Such an approach can improve not only time efficiency but also space efficiency.

We define the *frequency window* W_t^x, which is the Fourier transformed version of basic window w_t^x. $W_t^x|_{f_s}^{f_t}$ as the *filtered frequency window* which only contains half coefficients corresponding

to frequencies from f_s to f_t in W_t^x. $W_t^x|_{f_s}^{f_t}$ will only keep one half of elements to save memory space since elements in W_t^x are complex conjugate symmetric. The main idea of our approach is to incrementally updates filtered frequency window $W_t^x|_{f_s}^{f_t}$ from the previous filtered frequency window $W_{t-1}^x|_{f_s}^{f_t}$ instead of calculating from basic window w_t^x. If all filtered frequency windows in a pair of sliding windows are computed, then the lagged correlation coefficients can be directly computed using the frequency components based on Equations (4) and (6). Thus, our proposed method avoids computing repeated Fourier transforms and inverse Fourier transforms on every basic windows and perform correlation computation directly in the frequency domain on fewer data. The length of $W_t^x|_{f_s}^{f_t}$ is $\lfloor \frac{f_t - f_s}{f} m \rfloor + 1$, where f is the sampling rate of the streams.

To maintain frequency components upon receiving a new observation, the algorithm removes the quantities for each frequency that contributed by the first observation $w_{t-1}^x[0]$ of basic window w_{t-1}^x and adds the quantities for each frequency that brought by the new observation $w_t^x[m-1]$ of basic window w_t^x. To account for the slide of the window, the algorithm updates k^{th} coefficient by multiplying $e^{i\frac{2\pi k}{m}}$, for $\lfloor \frac{f_s}{f} m \rfloor \leq k \leq \lfloor \frac{f_t}{f} m \rfloor$. This process is applied to each of the frequencies from f_s to f_t. The steps are precisely represented by $filter$ function in Algorithm 2.

Once filtered frequency windows $\{W_{t-l}^x|_{f_s}^{f_t}, \ldots, W_t^x|_{f_s}^{f_t}\}$ and $\{W_{t-l}^y|_{f_s}^{f_t}, \ldots, W_t^y|_{f_s}^{f_t}\}$ are ready, then the lagged correlation $lcorr_t^{xy}$ can be computed by calling Algorithm 2. The function *oneCorr* in Algorithm 2 will be called to compute each $corr_{t_1 t_2}^{xy}$. Note that the correlation coefficients calculated from filtered frequency windows are *exactly* the same as the coefficients calculated from filtered windows in the naive algorithm. The exactness of our algorithm is directly derived from Parseval's theorem described earlier in Section 2.

To explain the function *oneCorr* in Algorithm 2, we describe how Equation (8) is evaluated using a filtered frequency window instead of a filtered basic window. In the following, we show how each term from Equation (8) can be expressed using frequency terms in detail.

Based on Equation (10), we can calculate the mean of filtered basic window with Equation (12).

$$\mu(w\prime_t^x) = \begin{cases} \frac{W_t^x[0]}{m} & \text{if } f_s = 0 \\ 0 & \text{if } f_s > 0 \text{ (DC term is filtered)} \end{cases}. \tag{12}$$

Based on Parseval's theorem in Equation (4), we can compute the $\sum w\prime_t^x[j]^2$ in Equation (13). We multiply a constant 2 to include the symmetric part of $W_t^x|_{f_s}^{f_t}$. This also applies to Equations (14), (15). There are two special cases, one case is when the $f_s = 0$ and another case is when the length of a basic window is an even number and the f_t equals to the Nyquist frequency, which is one half of the sampling rate f. For both cases, $W_t^x[\lfloor \frac{m}{2} \rfloor]$ or $W_t^x[0]$ need to be subtracted after we multiply 2 since there is no symmetric value to them. Both values need to be subtracted if conditions for both cases hold.

$$\sum_{j=0}^{m-1} w\prime_t^x[j]^2 = \frac{2}{m} \sum |W_t^x|_{f_s}^{f_t}[k]|^2. \tag{13}$$

Then the standard deviations can be calculated as below if $f_s > 0$.

$$\sigma(w\prime_t^x) = \sqrt{\frac{2}{m^2} \sum |W_t^x|_{f_s}^{f_t}[k]|^2} \tag{14}$$

Lastly, the product term $\sum(w\prime^x_{t_1}[j]w\prime^y_{t_2}[j])$ can be expressed in Equation (15) based on Equation (6).

$$\frac{2}{2m}\Big(\sum|W^x_{t_1}|^{f_t}_{f_s}[k]|^2+\sum|W^y_{t_2}|^{f_t}_{f_s}[k]|^2-\sum|W^x_{t_1}|^{f_t}_{f_s}[k]-W^y_{t_2}|^{f_t}_{f_s}[k]|^2\Big). \tag{15}$$

Finally, we can derive the equation that appears in line 16 of Algorithm 2 when $f_s > 0$. For the case when $f_s = 0$, it only needs to include the $\mu(w\prime^x_t)$ term in the computations since it is no longer equals zero.

ALGORITHM 2: FilCorr

```
Function LagFilterCorr(s^x_t, s^y_t, l, m, f, f_s, f_t)
1     LB ← m⌊f_s/f⌋
2     UB ← m⌊f_t/f⌋
3     W^x_t|^{f_t}_{f_s} ← filter(W^x_{t-1}|^{f_t}_{f_s}, m, LB, UB, w^x_{t-1}[0], w^x_t[m-1])
4     W^y_t|^{f_t}_{f_s} ← filter(W^y_{t-1}|^{f_t}_{f_s}, m, LB, UB, w^y_{t-1}[0], w^y_t[m-1])
5     curMax ← oneCorr(W^x_t|^{f_t}_{f_s}, W^y_t|^{f_t}_{f_s}, m) //case 1
      for t_i = t − l : t − 1 do
6         W^x_{t_i}|^{f_t}_{f_s} ← filter(W^x_{t_i-1}|^{f_t}_{f_s}, m, LB, UB, w^x_{t_i-1}[0], w^x_{t_i}[m-1])
7         W^y_{t_i}|^{f_t}_{f_s} ← filter(W^y_{t_i-1}|^{f_t}_{f_s}, m, LB, UB, w^y_{t_i-1}[0], w^y_{t_i}[m-1])
8         tmp1 ← oneCorr(W^x_t|^{f_t}_{f_s}, W^y_{t_i}|^{f_t}_{f_s}, m) //case 2
9         tmp2 ← oneCorr(W^x_{t_i}|^{f_t}_{f_s}, W^y_t|^{f_t}_{f_s}, m) //case 3
10        curMax = max(curMax, tmp1, tmp2)
      end
11    return curMax
end
Function filter(W^x_t|^{f_t}_{f_s}, m, LB, UB, d, a)
      //d is the element that will be deleted
      //a is the element that will be added
12    for q from 0 : UB − LB do
13        k ← q + LB //k is the index of W^x_t
14        W^x_{t+1}|^{f_t}_{f_s}[q] = e^{i 2πk/m} (W^x_t|^{f_t}_{f_s}[q] − d · e^{−i 2π0k/m} + a · e^{−i 2πmk/m})
      end
15    return W^x_{t+1}|^{f_t}_{f_s}
end
Function oneCorr(W^x_{t_1}, W^y_{t_2}, m)
      //Assume 0 < f_s < f_t < f/2
16    return (Σ_q |W^x_{t_1}[q]|^2 + Σ_q |W^y_{t_2}[q]|^2 − Σ_q |W^x_{t_1}[q] − W^y_{t_2}[q]|^2) / (2√(Σ_q |W^x_{t_1}[q]|^2 Σ_q |W^y_{t_2}[q]|^2))
end
```

In order to control the output rate, we use the parameter *step*. FilCorr can output all-pair correlations upon receiving the next set of observations in the streams. However, the output rate does not necessarily have to be the same as the input rate. If the current sliding window is s^x_t, we can slide to s^x_{t+step}, where *step* is the number of observations in a stream the algorithm gathers before

outputting the next set of pairwise correlations. When $step = 1$, the algorithm outputs at the same rate as the input. The larger the $step$, the slower the output rate.

4.3 Computational Complexity

FilCorr algorithm contains two stages of computation, the initial stage and the streaming processing stage. In the initial stage, FilCorr utilizes the FFT algorithm to transform the beginning basic window to the frequency window, then keeping the components within the filter band to get the filtered frequency window. The total time complexity for this stage is $O(mlogm + B)$ in which the FFT algorithm contributes $O(mlogm)$ [6, 7], and filtering contributes $O(B)$, m is the length of the basic window and B is the number of elements within the filter band which is $1 + \lfloor \frac{f_t - f_s}{f} m \rfloor$. The cost for this one-time operation can be amortized among the following streaming processing steps, thus yielding amortized time complexity $O(\frac{mlogm+B}{len(stream)})$, this value is near zero in the long run. Therefore, we do not count this cost to the overall time complexity.

The time complexity for streaming processing of one pair time series at any step is composed of two parts: (i) filtering and (ii) correlation computation. For the filtering, FilCorr only takes $O(B)$ time based on the $filter$ function in Algorithm 2. The worst case is $O(m)$ when B is close to m. Filtering in the naive algorithm will take $O(m \log m)$ to transfer the frequency and time domain when applying FFT and IFFT algorithm. As for the cost of computing lagged correlation coefficients, the *oneCorr* function in both algorithms will be executed l times. Each iteration of the naive algorithm will take $O(m)$ based on the equation in line 18, while the FilCorr only takes $O(B)$ time.

The combined time complexity depends on the number of pairs of time series in the system. We assume all permutations of $O(N)$ time series. Thus, the time complexity to compute one lagged correlation value for naive is $O(Nm \log m + lmN^2)$, and $O(BN + lBN^2)$ for FilCorr, $O(mN + lmN^2)$ in the worst case. In practice, the speedup is more because the number of possible lags is much less than the window size, and the frequency band is much smaller than the number of observations in the signal.

The naive algorithm's space complexity is $O(Nlm)$ to maintain all the filtered basic windows in the most recent sliding window for all streams. The space complexity of FilCorr is $O(NlB)$.

4.4 Extensions to Our Implementation

Our proposal can be easily extended to execute in parallel and to employ different digital filters. In this section, we discuss the properties of FilCorr that provide such flexibilities.

Parallelization: Since lagged correlation computation for one pair of streams is independent of other pairs, we can utilize multiple processing units (i.e., thread, core, processor, etc.) to expand the capacity to calculate multiple pairs in parallel. Each pair will maintain their sliding windows for the streams s^x and s^y. In order to achieve the best real-time performance, one filtered frequency window with its statistics can be accessed by all pairs to save more memory and avoid redundant computation.

Different filters: Our method can adapt to other types of digital filters; for instance, the Butterworth filter illustrated in Figure 6. We can directly apply this filter on top of the box filter by changing each frequency weight. In this way, we can still save time from computing unnecessary frequencies and keep the filter property within the bandwidth. This operation will not change the overall time complexity since it is a linear operation to factor the weights in all frequencies. Note that incrementally updating the DFT coefficients is also a linear time operation. Our method is not limited to the digital frequency-domain filters. A time-domain digital or analog filter can also be applied for streams passing through as pre-processing then passing into our method for correlation computation on the targeted frequencies.

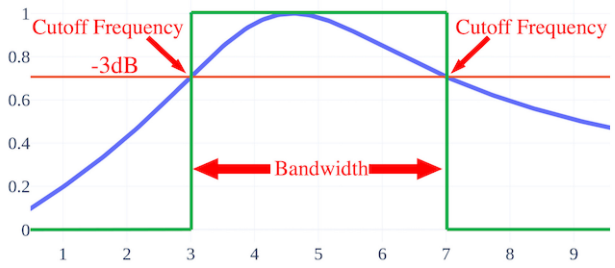

Fig. 6. Butterworth second order bandpass filter 3–7 Hz.

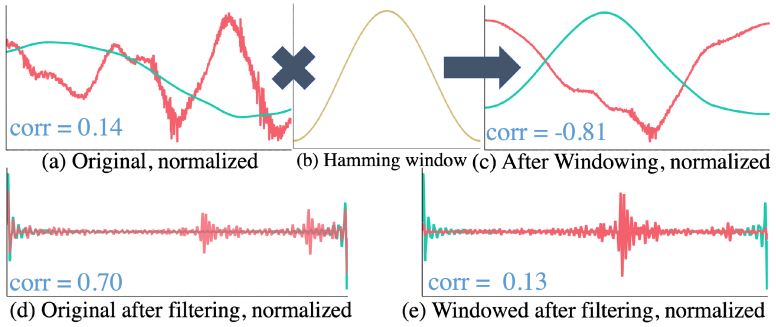

Fig. 7. Example of the Ringing effect on two uncorrelated seismic signals and how the multiplication by a Hamming window can address the false high correlation between them.

4.5 Discussion on Data Independence

The general idea of FilCorr is to exploit the use of digital filters to achieve faster pairwise correlation computation. Besides removing undesired noise, filters allow computing the correlation between time series in a reduced dimensional space provided by the frequency domain. It turns the time cost of our method data-independent and will be affected by neither the sparsity nor the similarity. This cost is related to the lag and the bandwidth assumed by the filter, typically much smaller than the original series. It is also important to note that unlike other methods that speedup the correlation computation by pruning improbable pairs and provide approximate results, our approach efficiently calculates all possible lagged pairs and provides exact results.

4.6 Managing Spurious Correlation

Filtered correlation can occasionally be spurious because of Ringing effect [8]. It occurs due to spectral leakage when the length of the basic window is mutually prime with the period of the signal. For instance, in Figure 7(a), we show two uncorrelated signals (correlation = 0.14) obtained from different seismometers. If we filter such signals by a box filter, we obtain a false high correlation of 0.70 in the resulting signals, as illustrated in Figure 7(d). This high correlation is mainly caused by similar oscillations at the edges of both signals. This effect can be addressed by "windowing" (i.e., multiplying with a window function) the time series before converting them to the frequency domain. The resulting signals after the process of windowing using a Hamming window (Figure 7(b)) are illustrated in Figure 7(c). Finally, the filtered signals after the windowing process are illustrated in Figure 7(e). We note that the correlation of the original signals is reduced to 0.13, eliminating the previously observed false high correlation.

Windowing on each basic window data will change the filter process since we could not maintain the filtered frequency window incrementally. Thus, we need to apply the filter function from the naive algorithm.

5 EXPERIMENTAL EVALUATION

All our experiments are reproducible and the supplementary material such as code, data, and additional results are publicly available on our supporting website [29]. FilCorr is exact and deterministic, and the efficiency is not data-dependent.

5.1 Sanity Check

Before the experimental evaluation and comparisons with existing methods, we show a sanity check to demonstrate that our approach is fast enough to be employed in a real-time system that is capable of monitoring hundreds of sensors with a high sampling rate. To demonstrate this claim, we develop a system named Seisviz (www.seisviz.com) that will render the lagged correlation values computed by FilCorr in real-time. It has been used successfully for monitoring a seismic network at Yellowstone, WY, USA. This network has 30 stations, where each station has up to six channels. Each channel represents an individual stream of observations at 100 Hz. We obtain 670 pairs if we pair the streams from different stations by the same channel type. We demonstrate videos of detected earthquakes in real-time by Seisviz on our website [29]. In Section 6.1, we return to the discussion about our findings on seismic data and a detailed system implementation description.

5.2 Setup

All experiments are performed on a desktop computer with an Intel i9-9900k (8 cores) processor, 32 GB of memory, running a Linux operation system. As FilCorr is data-independent, we use synthetic data for various experiments. The performance on real-world data will be discussed in the case studies presented in Section 6.

We create two testing scenarios to evaluate the performance of naive and FilCorr algorithm: *offline* and *online*. In the offline scenario, all observations are available beforehand, so the system will use its full power to compute until it finishes computation on all data. For the online scenario, the observations are generated in a streaming fashion with the speed of sampling rate.

In the offline scenario, we measure the total execution time, including I/O operations. In the online scenario, we seek to find the maximum number of streams that the system can compute their pairwise lagged correlation without any delay. To measure this, we create an ideal environment where all the streams have an equal length and the same sampling rate specified by the parameter f. We consider the system capable of processing this number of streams if the finish time is ahead of the expected next computation time. There is no need to measure at each step because if the number of streams exceeds the system capability, then the extra time to finish computation will accumulate at each step and reflect at the finish time.

For both offline and online, we run ten trials to confirm an algorithm's capability on a certain number of streams to account for random events in the operating system.

5.3 Efficiency

In Figure 8, we show the execution time of FilCorr and the naive algorithm in the offline scenario. We consider three filter bands (5 Hz, 25 Hz, and 50 Hz), and five sampling rates between 100 Hz to 300 Hz with increments of 50 Hz. The results testify to the time complexity discussed in Section 4.3. The naive algorithm's execution time remains on the same level for the same sampling rate no matter the bandwidth size. This is because the number of observations used for correlation

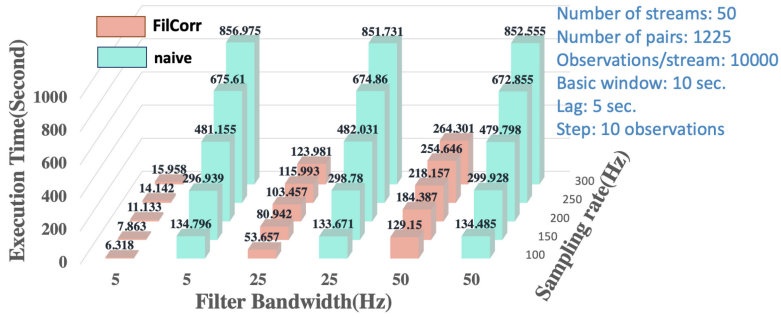

Fig. 8. Offline performance of FilCorr and naive algorithm.

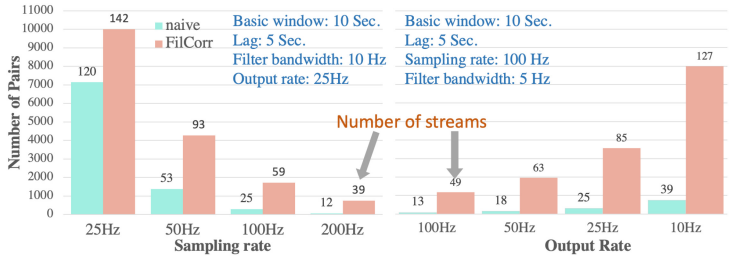

Fig. 9. Online performance of FilCorr and naive algorithm. The left figure shows the results varying the sampling rate and fixed values for the remaining parameters. The right figure shows the results under different output rates. In both figures, the total number of stream pairs are computed as all possible combinations of any two streams.

computation of the naive algorithm does not change along with the filter bandwidth in the time domain. On the contrary, the narrower the band the fewer frequency components for FilCorr to compute, so less execution time. Another conclusion we can draw from the figure is that the growth rate for FilCorr is much slower than the naive for a fixed bandwidth when the sampling rate is increasing. This is because the basic window length is calculated based on the time, which is 10 seconds in here. For FilCorr, the number of frequency components in the filter band remains the same since B is defined as $1 + \lfloor \frac{f_t - f_s}{f} m \rfloor$, m equals $10f$ so B equals $1 + \lfloor 10(f_t - f_s) \rfloor$. The only extra cost for FilCorr is coming from the longer lag. However, for the naive, it is affected by both longer lag and more observations for computing the correlation; thus, it increases at a much higher rate than FilCorr.

In Figure 9, we show the number of pairs that each algorithm can process without delay in the online scenario. We vary the input sampling rate and output rate for both algorithms. In all experimental settings, FilCorr can process (up to 4×) more sensors than the naive algorithm. The performance gap increases with higher input or output rate. For other filter bands, the general performance trends hold.

5.4 Comparison to Existing Method

Based on our previous comparison shown in Table 2, we argue that FilCorr is a comprehensive method for streaming correlation computation. However, although not an ideal match in capabilities, we identify ParCorr [27] as the most recent baseline with state-of-the-art performance. ParCorr calculates pairwise correlation in parallel with the Apache Spark system based on

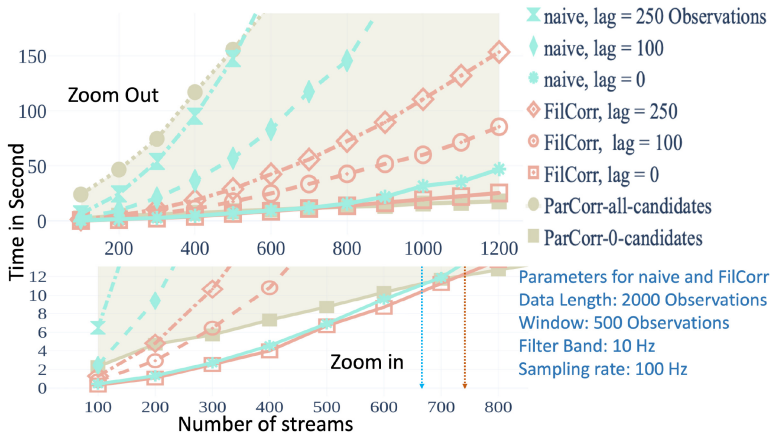

Fig. 10. Comparison with ParCorr fixing *step* = 20. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0, and the vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0.

randomly projected sketches. Note that ParCorr does not compute lagged correlation. The following experiments are conducted on the same setup as previous experiments.

In order to favor ParCorr's implementation in our comparison, we perform all the experiments in the offline scenario. To offset the Spark system's costs, we conduct another set of experiments as offsets. Each offset experiment will only process one time series with window size as 1, step size as 1, and the length of this time series depends on the actual corresponding experiment parameters. Our goal is to make sure the sliding window in both offset experiments and actual corresponding experiments move the same number of times. All the experiment results here are adjusted based on the results of the corresponding offset experiment.

Since the ParCorr is data-dependent, we use two sets of synthetic data with 2,000 observations in each targeted to emulate the best-case and worst-case scenarios for ParCorr. The cost of ParCorr depends on the number of pairs it can prune without computing the correlation coefficients. Our first synthetic dataset contains sequences of uniformly distributed random numbers, which is expected to contain only uncorrelated pairs. Thus, a random noise dataset is the best data for ParCorr where it can prune all possible pairs. To further boost ParCorr's performance, we use a high correlation threshold (*candThreshold*) for better pruning. The purpose of this is to guarantee that no two series will lead to actual correlation computation for ParCorr. Besides, we also set the parameter *candThreshold* with a high value in ParCorr as double insurance. On the contrary, the second dataset is a sinusoid that is expected to have all possible pairs of streams to be highly correlated. In this case, ParCorr computes correlation for all possible pairs, failing to prune and demonstrating the worst-case performance. For FilCorr and Naive, the pairs are generated based on all possible combinations from all the streams.

We show the performance comparison in Figure 10. The light grey shaded area represents the range of performance by ParCorr. The worst-case performance (on sinusoid data) is illustrated by the superior grey line with solid circles, and the best-case (on random data) by ParCorr is illustrated by the inferior grey line with solid boxes. The actual performance of ParCorr on any other dataset should be in between the worst-case and best-case lines. Figure 10 (zoom-out) shows that the time spent by FilCorr for various lags is well inside the shaded area. To be fair to ParCorr, when we consider the synchronous correlation (lag = 0), FilCorr is more efficient than the best-case

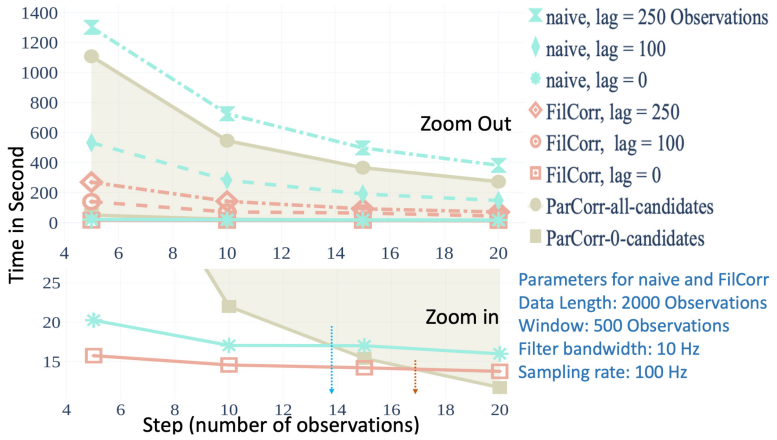

Fig. 11. Comparison with ParCorr varying the *step* from 5 to 20 observations. The vertical red line shows the crossing point between ParCorr and FilCorr with lag = 0; the arrow points the point where the output rate is 6 Hz. The vertical blue line shows the crossing point between ParCorr and naive algorithm with lag = 0, and the arrow points to the output rate as 7 Hz.

of ParCorr up to around 700 streams as shown in Figure 10 (zoom-in). Therefore, we recommend FilCorr on a single desktop computer when the number of streams is less than 700, instead of using a parallel system.

In the second experiment, we fix the total number of streams at 800 and vary the *step* from 5 to 20 observations, which correspond to the output rate of 20 Hz to 5 Hz, respectively. We note in the results shown in Figure 11 that the execution time for all methods increases when the *step* is getting smaller to compute more correlation coefficients for a higher output rate. However, ParCorr increases at a higher rate compared to FilCorr. The zoom-in figure shows that FilCorr with $lag = 0$ has better performance than the best-case of ParCorr when the output rate is higher than 6 Hz.

5.5 Parameter Sensitivity

In this section, we discuss the algorithms' sensitivity to three design parameters: (i) lag, (ii) filter bandwidth, and (iii) window size. We consider the online scenario and measure the performance between the naive algorithm and FilCorr. We fix the sampling rate at $f = 100$ Hz and $step = 10$ observations for all experiments. The results are presented in Figure 12.

Figure 12 shows that doubling either the window size or the lag size has similar effects, where the number of pairs the algorithm can handle shrinks to half for both FilCorr and naive. The performance of FilCorr approaches the naive when the bandwidth increases.

6 CASE STUDIES

This section discusses different scenarios that can benefit from FilCorr. We present three case studies with data from diverse domains, such as seismic signals captured by seismometers, motion signals captured by accelerometers, and brain activity signals from neuroimaging.

Before we dive into case studies, we would like to make a general recommendation about the usage of FilCorr in different domains. In general, any application that needs pairwise Pearson's correlation values for various purposes can utilize FilCorr. However, some cases can enjoy the full benefits from FilCorr while some are negligible. To simplify the decision-making of a domain user,

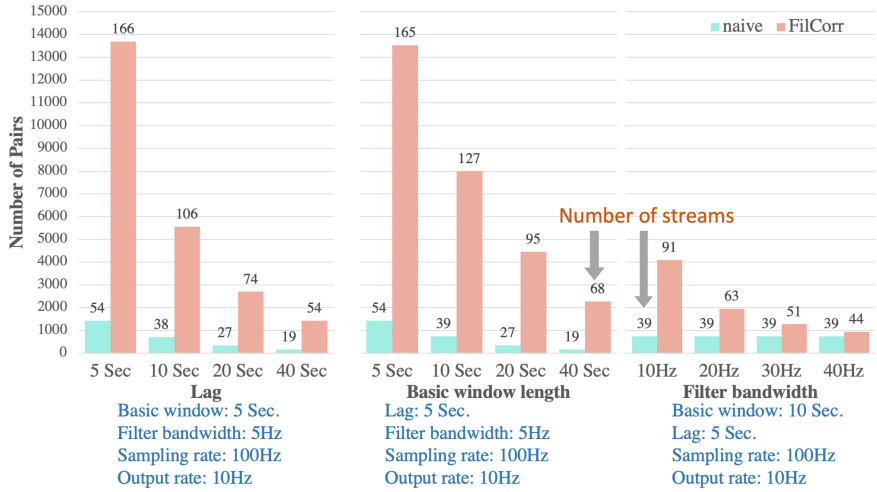

Fig. 12. Results for the parameter sensitivity test considering the online scenario.

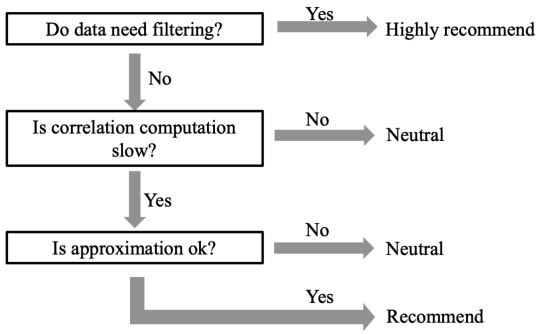

Fig. 13. Different recommendation levels on FilCorr based on the application requirements. Neutral means there are no apparent benefits (No loss either) when compared with time domain implementation. Approximation means results are derived from fewer frequency components.

we draw a flowchart in Figure 13. Comparing the performance of FilCorr to the naive approach, we set different recommendation levels based on the need for filtering and execution speed.

6.1 Seismic Event Monitoring

We have deployed a system for monitoring seismic events in which FilCorr is one of the core components. We designed such a system, named Seisviz,[1] by following the separation of concerns (**SoC**) design principle [17]. The main components of Seisviz are illustrated in Figure 14.

The system has four modules: (*i*) data collector, (*ii*) the Kafka cluster, (*iii*) the FilCorr computing unit, and (*iv*) the Seisviz web server. Such a modular design is useful for developing and maintaining the system, while robustness and scalability are improved because failures can easily be tracked to one of the modules, and scaling each module is easier than scaling the whole system. There are two data pipelines in the system, one path originates at the seismic sensors and ends in the FilCorr computing unit. The other path originates at the FilCorr computing unit and ends in

[1]The system is publicly available at www.seisviz.com.

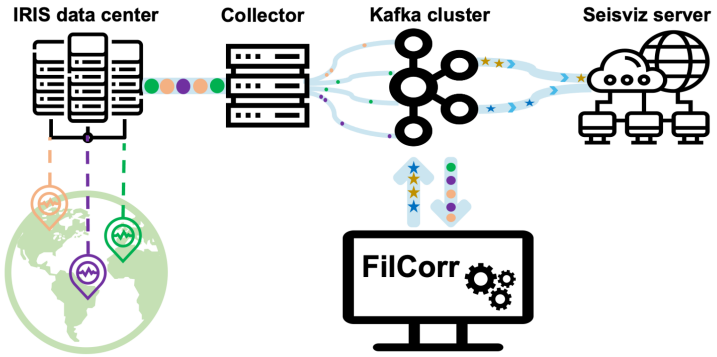

Fig. 14. Main components of Seisviz, a real-time system for seismic event monitoring.

the Seisviz server. The seismic observations are fetched from the **Incorporated Research Institutions for Seismology (IRIS)**—IRIS data center [10] to the Kafka cluster, then consumed by the FilCorr computing unit. The correlation values are computed by the FilCorr computing unit and saved in the Kafka cluster, and finally consumed by the Seisviz web server.

The Kafka cluster was used for temporary data storage and data distribution. Kafka has essential features that meet our requirements, such as processing streaming data in real-time, storing a certain amount of historical data in a durable way, and allow multiple consumptions by several applications and systems. Besides, it provides extra benefits, including fault tolerance and high availability [22], which can enhance the reliability and the scalability of our system.

A few key points are worth discussing when using Kafka for time series data. The first thing is the order of observations since it is natural to keep each observation in a timely order or index-based order for time series; however, no such order can be maintained if we use multiple Kafka topics with one partition or one topic with multiple partitions to store observations from a time series. In other words, there is no guarantee that the order for each observation arriving at the consumer is the same order when they are generated. So the downstream services need to restore the order. This is because Kafka can only guarantee the records from the same topic partition will arrive at the consumer in the same order as they were appended to the partition but not for the records across many partitions. This leads to another approach that uses a topic with only one partition to store observations from a time series. Such an approach can bring time efficiency to the downstream services as they no longer need to restore each observation order. However, this may cause a performance penalty when consuming a large number of records at a time compared with the approach using a topic with multiple partitions on several nodes in a cluster. We choose the latter approach for simplicity, and our system scale is not reaching that performance bottleneck.

Secondly, Kafka only supports millisecond precision for timestamp, generating losses when the period of a stream is less than a millisecond. To solve this, we first look at the structure of the Kafka record. Each record has three attributes: key, value, timestamp. The key attribute is free in our case, so we can combine timestamp and key attributes to store the time information of an observation; time components after millisecond can be saved in the key attribute. This approach can support the precision level up to nanoseconds, which is sufficient for virtually any streaming time series problem.

As previously stated, our system considers IRIS as a data source. IRIS provides a protocol called SeedLink[2] for users to receive real-time data. The streams of time series from IRIS are sent out

[2]https://ds.iris.edu/ds/nodes/dmc/services/seedlink/.

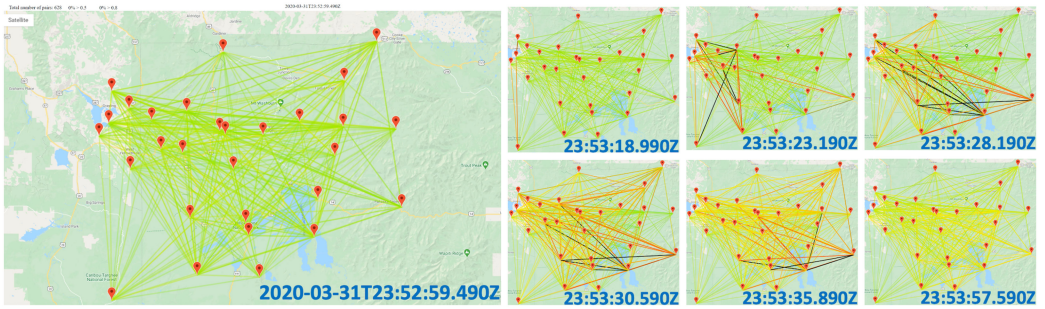

Fig. 15. Pairwise correlation among 29 stations at Yellowstone, WY over different times given an M6.5 earthquake occurred in Challis, Idaho, at 23:52:30 2020-03-31(UTC).

in the form of batches containing a certain number of observations. Although SeedLink is based on the TCP/IP protocol that guarantees packets transmission in order and without any loss, the packets' order may not be consistent with observations' time order. To address this, we develop a collector module to save the streams in the Kafka cluster and, most importantly, recover the original data streams in time order. This process will encounter three different scenarios: (1) There is a time gap between data batches; in other words, a segment of observations of a time range is missing. (2) Two batches are overlapping. (3) There is a time shift among observations; if the time of one observation is shifted, then the time duration between this observation and the previous observation is no longer consistent with the sampling rate. This time shift has to be smaller than the sampling period to distinguish this scenario from the first scenario. Once we have all the necessary observations, we can start computing the filtered lagged correlation values using the FilCorr algorithm.

The back-end server will read the computed correlation value from the Kafka cluster, group the results based on the timestamp, and then send the results in a streaming manner to the front end. The front-end website will render the correlation value on a colored line between two points on the map, as illustrated in Figure 15. The point represents the location of a station, which consists of various seismometers. The correlation value is depicted by the color and transparency of the line between two stations. Since there are usually multiple seismometers at one station, if there are multiple pairs of streams between two stations, then only the correlation with the highest value will be rendered at the moment.

In Figure 15, we illustrate the propagation of a seismic event of magnitude 6.5 that occurred at Challis, Idaho, on March 03, 2020[3] and which was observed by our system about 300 miles away in the stations at Yellowstone. In this representation, each red location symbol represents a station, and a colored line represents the lagged correlation between two stations. Different colors and levels of alpha reflect the lagged correlation value. The correlation values from 0 to 0.5 to 0.9 are mapped from green to yellow to red. A black edge represents a correlation value greater than 0.9. We notice that the correlated edges are appearing between station pairs as the earthquake wave reaches them. Similarly, edges become uncorrelated when the wave has passed through.

In summary, our system is monitoring a seismic network with 30 stations in Yellowstone, Wyoming. There is a total of 98 streams, which can form a total of 670 pairs. Our system is computing the correlation of these 670 pairs at a 10 Hz rate. Correlation coefficients can capture earthquake propagation through a region in real-time, which can easily be converted to a detector with the rule: *If more than Q% of pairs of stations are highly correlated (*>0.8*), an earthquake is propagating.*

[3] https://earthquake.usgs.gov/earthquakes/eventpage/us70008jr5/executive.

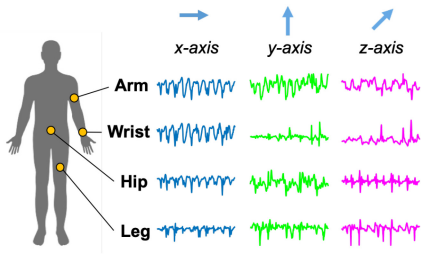

Fig. 16. Motion sensor locations and examples of time series generated by each sensor in the three axes.

The utility of such a detector is massive for early warning systems because seismic wave propagate at 8 km/s, which is much slower than electronic signals carrying the warning.

As we previously have shown in Figure 2, filtering is essential for seismic data in order to remove noise and obtain a representation in the frequency domain that better describes the signals. For this case study, we consider a 20 seconds window size, a box bandpass filter with a cutoff frequency of 3 Hz and 7 Hz, and a lag of 10 seconds.

Although the Seisviz website shows the results with a few minutes lag, we still claim our system is real-time since the delay occurs before the data arrive at our system. The delay is typically caused at the origin (i.e., seismometer) and during the transmission process. The delay time varies and is beyond our control. Seisviz waits initially to accumulate enough data to be able to calculate the first set of correlations. After that, the system processes (i.e., flows data through the modules and computes the correlations) at a faster rate than the rate of streams.

6.2 Motion Monitoring

In recent decades, the emergence of low-cost wearable devices made it easier to measure body parts' motion by accelerometers and gyroscope sensors. One interesting use of motion data is in interactive systems that monitor the movement synchronization between different users in real-time. Synchronous motion is at the heart of many art forms, including dance and music, sports such as swimming and gymnastics, and electronic games. The monitoring of a group of users is useful for physical activities, choreography design, precise movements evaluation, and other tasks.

In this case study, we demonstrate how our method monitors motions in real-time for dance sessions, where slight irregularities are expected between the participants. For evaluation, we consider the Dancestix data [20], which has four participants who performed a Lady Gaga's song with the same choreography with 21 dance steps following the 581 beats of a metronome at 2 Hz. As illustrated in Figure 16, each participant has four three-axis accelerometer sensors placed on the hip, wrist, arm, and leg. The data was recorded at 100 Hz, totaling time series with 29,255 observations for each sensor in a given axis, while we filter the data between 1 Hz and 3 Hz.

Ideally, any pair of dancers should show a strong correlation, while a slight deviation would suggest quality degradation. Since dance motions are repetitive, the allowable lag in detecting correlation must not be more than the duration of a dance step. In Figure 17, we show the resulting pairwise correlation between overall body parts of dancers, considering a 2 seconds window (200 observations), a lag of 50 and 0 (no lag) observations, with a box filter (1–3 Hz) and without filtering. The location of the dance steps is represented by the vertical lines identified by letters (A-T).

For all pairs in Figure 17, we note that when we consider the lagged correlation of filtered data, higher values are observable over time. The correlations are close to zero when we do not consider filter and lag, demonstrating the importance of such features for this problem. From these correlations, we can carry out real-time analysis to identify fatigue, off-rhythm dancers, and so on.

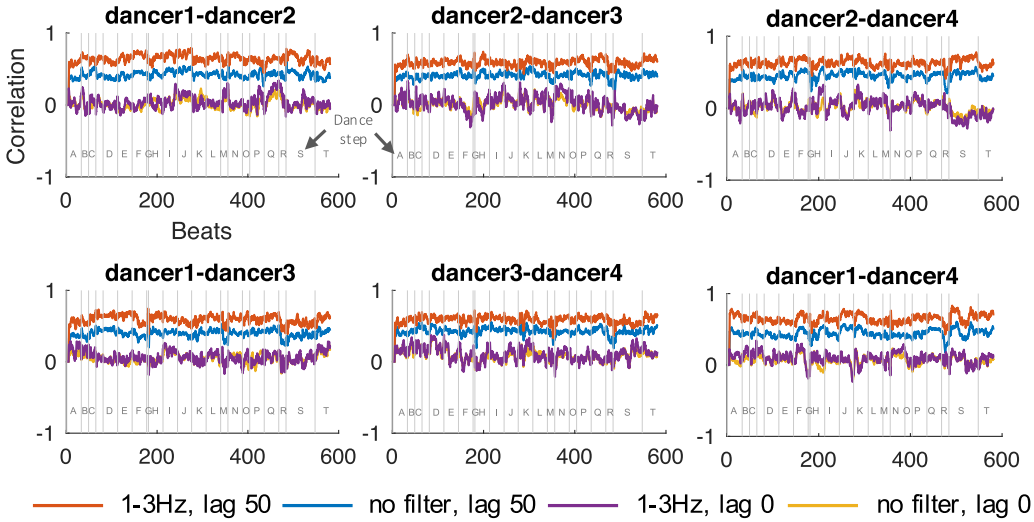

Fig. 17. Cross-correlation between pairs of dancers over all body parts considering the use and absence of filter and lag.

For example, we see that dancer 3 is significantly less correlated to all other participants while performing step M.

6.3 Electrophysiological Monitoring

Parkinson's disease is the second-most common neurodegenerative disorder that affects 2–3% of the population older than 65 years of age [19]. As the disease progresses, people may have difficulty walking and talking. It is caused by a significant decrease in dopamine production, a neurotransmitter that helps us to automatically carry out voluntary body movements without the need for thinking about every single movement that our muscles make. In the absence of dopamine, particularly in a small brain region called the substantia nigra, the individual's motor control is lost [13].

An essential tool for monitoring the disease's progress and its effects is the **electroencephalogram (EEG)** [11]. An EEG records electrical activity produced by the brain via small noninvasive sensors attached to the scalp in different regions, offering a direct measure of neuronal activities. The study of the correlation between EEG signals from different parts of the brain has been an active research area in the last years [4, 12, 15]. In general, a high correlation between the signals from different electrodes indicates similar brain activity, and a low correlation indicates that the brain activity at the different measurement sites is relatively independent [3]. However, patients with conditions such as Parkinson's related dementia and Alzheimer's disease often exhibit different behavior and a slow oscillatory brain activity compared with healthy subjects [12]. In EEG data, it is expected a lag due to the time spent to transfer the signals from one brain region to another. Also, the use of filters is essential to identify the frequencies that compose the signal. In this direction, for better analysis, it is essential to consider the lagged correlation of filtered EEG data, as proposed in this work.

In this case study, we consider a dataset of 25 patients with Parkinson's disease and an equal number of people in the control group [5]. Subjects performed a task of identifying novel sounds in a sequence of known sounds. EEG was recorded continuously by a 64-channel system with a sampling rate of 500 Hz. Here, we only consider the EEG responses to the novel sounds. We first average all the trials and participants in one group then calculate pairwise lagged correlation across

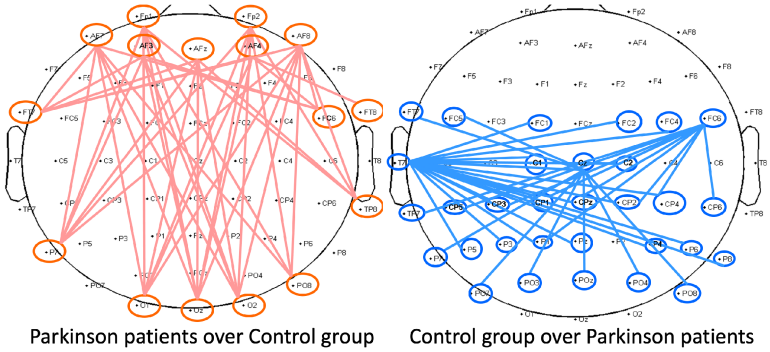

Fig. 18. Lag-correlated pairs of channels present in one group and not present in the other.

the channels. The results are for the one second window, 60 ms lag containing 30 observations, and a box filter with a band from 0 Hz to 20 Hz.

In Figure 18, we only show the pairs that are correlated only with lag and not correlated without lag. The left figure shows the correlated pairs of channels with lagged correlation values greater than 0.5 in the patient group that are not correlated in the control group, and the right figure shows the lagged correlated pairs in the control group but not in the patient group. We observe the front-back lag-correlation in the patient group, while the control group does not show any correlation in the frontal lobe. We do not claim any neurological significance of this finding. However, the difference between patient and control groups suggests that filtered and lagged correlation can support EEG analysis as a diagnostic tool.

7 CONCLUSION

This article demonstrates an algorithm, FilCorr, to compute filtered and lagged correlation over streaming time series. FilCorr combines filtering and cross-correlation computing operations in one step to obtain the lagged correlation between streaming time series efficiently. FilCorr is faster than the state-of-the-art ParCorr algorithm that computes correlation in parallel. We show three case studies where the algorithm achieves promising results towards greater societal impacts. We also provide a publicly available real-time system named Seisviz that employs FilCorr in its core mechanism for monitoring a seismometer network.

REFERENCES

[1] Y. Ahmad and S. Nath. 2008. COLR-Tree: Communication-efficient spatio-temporal indexing for a sensor data web portal. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*. IEEE, 784–793.

[2] Anthony Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. 2020. On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1. 0). In *Proceedings of the International Workshop on Advanced Analytics and Learning on Temporal Data*. Springer, 3–18.

[3] R. Bhavsar, Y. Sun, N. Helian, N. Davey, D. Mayor, and T. Steffert. 2018. The correlation between eeg signals as measured in different positions on scalp varying with distance. *Procedia Computer Science* 123, 1 (2018), 92–97.

[4] P. Bob, M. Susta, K. Glaslova, and N. N. Boutros. 2010. Dissociative symptoms and interregional EEG cross-correlations in paranoid schizophrenia. *Psychiatry Research* 177, 1–2 (2010), 37–40.

[5] J. F. Cavanagh, P. Kumar, A. A. Mueller, S. P. Richardson, and A. Mueen. 2018. Diminished EEG habituation to novel events effectively classifies parkinson's patients. *Clinical Neurophysiology* 129, 2 (2018), 409–418.

[6] FFTW. 2002. FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data. Retrieved 03 January, 2022 from http://www.fftw.org/.

[7] M. Frigo and S. G. Johnson. 1998. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vol. 3. IEEE, 1381–1384. DOI: https://doi.org/10.1109/ICASSP.1998.681704

[8] David Gottlieb and Chi-Wang Shu. 1997. On the gibbs phenomenon and its resolution. *SIAM Review* 39, 4 (1997), 644–668.
[9] J. Hickish, N. Razavi-Ghods, Y. C. Perrott, D. J. Titterington, S. H. Carey, P. F. Scott, K. J. B. Grainge, A. M. M. Scaife, P. Alexander, R. D. E. Saunders, M. Crofts, K. Javid, C. Rumsey, T. Z. Jin, J. A. Ely, C. Shaw, I. G. Northrop, G. Pooley, R. D'Alessandro, P. Doherty, and G. P. Willatt. 2018. A digital correlator upgrade for the arcminute microkelvin imager. *Monthly Notices of the Royal Astronomical Society* 475, 4 (2018), 5677–5687.
[10] IRIS. 2018. The facilities of IRIS Data Services, and specifically the IRIS Data Management Center, were used for access to waveforms, related metadata, and/or derived products used in this study. IRIS Data Services are funded through the Seismological Facilities for the Advancement of Geoscience and EarthScope (SAGE) Proposal of the National Science Foundation under Cooperative Agreement EAR-1261681. Retrieved 03 January, 2022 from https://www.iris.edu/hq/.
[11] N. Jackson, S. R. Cole, B. Voytek, and N. C. Swann. 2019. Characteristics of waveform shape in Parkinson's disease detected with scalp electroencephalography. *Eneuro* 6, 3 (2019), 1–11.
[12] D. Jeong, Y. Kim, I. Song, Y. Chung, and J. Jeong. 2016. Wavelet energy and wavelet coherence as EEG biomarkers for the diagnosis of Parkinson's disease-related dementia and alzheimer's disease. *Entropy* 18, 8 (2016), 1–17.
[13] E. R. Kandel, J. H. Schwartz, T. M. Jessell, S. Siegelbaum, and A. J. Hudspeth. 2000. *Principles of Neural Science*. Vol. 4. McGraw-hill, New York.
[14] A. Mueen, S. Nath, and J. Liu. 2010. Fast approximate correlation for massive time-series data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 171–182.
[15] S. H. Na, S. Jin, S. Y. Kim, and B. Ham. 2002. EEG in schizophrenic patients: Mutual information analysis. *Clinical Neurophysiology* 113, 12 (2002), 1954–1960.
[16] A. V. Oppenheim and R. W. Schafer. 1975. *Digital Signal Processing* (1 ed.). Prentice Hall, Hoboken.
[17] H. Ossher and P. Tarr. 2001. Using multidimensional separation of concerns to (re) shape evolving software. *Communications of the ACM* 44, 10 (2001), 43–50.
[18] Marc-Antoine Parseval. 1806. Mémoire sur les séries et sur l'intégration complète d'une équation aux différences partielles linéaires du second ordre, à coefficients constants. *Mém. prés. par Divers Savants, Acad. Des Sciences, Paris,(1)* 1 (1806), 638–648.
[19] W. Poewe, K. Seppi, C. M. Tanner, G. M. Halliday, P. Brundin, J. Volkmann, A. Schrag, and A. E. Lang. 2017. Parkinson disease. *Nature Reviews Disease Primers* 3, 1 (2017), 1–21.
[20] Henning Pohl and Aristotelis Hadjakos. 2010. Dance Pattern Recognition using Dynamic Time Warping. In *Proceedings of the 7th Sound and Music Computing Conference (SMC'10)*. mdpi, Barcelona, Spain, 183–190.
[21] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. 2005. Braid: Stream mining through group lag correlations. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 599–610.
[22] M. J. Sax. 2018. *Apache Kafka*. Springer International Publishing, Cham, 1–8. DOI: https://doi.org/10.1007/978-3-319-63962-8_196-1
[23] N. S. Senobari, G. J. Funning, E. Keogh, Y. Zhu, C. M. Yeh, Z. Zimmerman, and A. Mueen. 2019. Super-efficient cross-correlation (SEC-C): A fast matched filtering code suitable for desktop computers. *Seismological Research Letters* 90, 1 (2019), 322–334.
[24] J. Shieh and E. Keogh. 2008. iSAX : Indexing and mining terabyte sized time series. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 623–631.
[25] D. F. Silva, V. M. A. Souza, D. Ellis, E. Keogh, and G. E. A. P. A. Batista. 2015. Exploring low cost laser sensors to identify flying insect species. *Journal of Intelligent & Robotic Systems* 80, 1 (2015), 313–330.
[26] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2021. Time series data augmentation for deep learning: A survey. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 4653–4660. DOI: https://doi.org/10.24963/ijcai.2021/631. Survey Track.
[27] D. E. Yagoubi, R. Akbarinia, B. Kolev, O. Levchenko, F. Masseglia, P. Valduriez, and D. Shasha. 2018. ParCorr: Efficient parallel methods to identify similar time series pairs across sliding windows. *Data Mining and Knowledge Discovery* 32, 5 (9 2018), 1481–1507.
[28] S. Zhong, V. M. A. Souza, and A. Mueen. 2020. FilCorr: Filtered and lagged correlation on streaming time series. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 1436–1441.
[29] S. Zhong, V. M. A. Souza, and A. Mueen. 2020. Supporting website. Retrieved 03 January, 2022 from https://sites.google.com/view/filcorr/.
[30] Y. Zhu and D. Shasha. 2002. StatStream: Statistical monitoring of thousands of data streams in real time. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. Elsevier, 358–369.

Received January 2021; revised October 2021; accepted November 2021