ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams

Mourad Khayati, Ines Arous, Zakhar Tymchenko, Philippe Cudré-Mauroux
University of Fribourg
Switzerland
{firstname.lastname}@unifr.ch

ABSTRACT
With the emergence of the Internet of Things (IoT), time series streams have become ubiquitous in our daily life. Recording such data is rarely a perfect process, as sensor failures frequently occur, yielding occasional blocks of data that go missing in multiple time series. These missing blocks do not only affect real-time monitoring but also compromise the quality of online data analyses. Effective streaming recovery (imputation) techniques either have a quadratic runtime complexity, which is infeasible for any moderately sized data, or cannot recover more than one time series at a time.

In this paper, we introduce a new online recovery technique to recover multiple time series streams in linear time. Our recovery technique implements a novel incremental version of the Centroid Decomposition technique and reduces its complexity from quadratic to linear. Using this incremental technique, missing blocks are efficiently recovered in a continuous manner based on previous recoveries. We formally prove the correctness of our new incremental computation, which yields an accurate recovery. Our experimental results on real-world time series show that our recovery technique is, on average, 30% more accurate than the state of the art while being vastly more efficient.

PVLDB Reference Format:
doi:10.14778/3430915.3430920

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/eXascaleInfolab/orbits.

1 INTRODUCTION
Time series are ubiquitous in many domains, e.g., finance, hydrology, network monitoring, or the Internet of Things (IoT). In such applications, time series often contain a large number of missing values that occur as blocks of consecutive values because of sensor failure, power outages, transmission problems, etc. [4, 37]. Data management systems assume no such gaps exist in the data. Even if a system can work with incomplete data (e.g., NULLs in databases), leaving missing values untreated can cause incorrect or ill-defined results [7, 9]. Missing values can alter time series statistical properties, such as the correlation. This in turn, might significantly affect further analysis tasks, e.g., data sampling, or exploration, rendering their results pointless [14]. Moreover, repairing dirty series substantially improves downstream tasks such as classification [39].

The recovery of these missing blocks is challenging, as in many modern applications data arrives in a streaming fashion. The stream could be infinite, rendering batch recovery techniques that attempt to process the entire stream for analysis impractical [22]. In many mission-critical applications, the missing blocks need to be recovered as they are encountered. We describe below two real-world applications that require the continuous recovery of missing blocks in time series streams. We empirically evaluate these two applications in Section 6.

Example 1. We consider the example of an IoT-based sports analytical use-case, where millions of sensors are unobtrusively integrated into the clothing and equipment of players to monitor their activities in real-time [13]. The monitoring aims to examine the players’ performance and minimize the risk of injury during practice. Missing blocks occur here as the sensors often detach from the players. Recovering these holes on-the-fly allows coaches to immediately adjust their tactics by repositioning the players on the field, which in turn might improve the overall performance of the team.

Example 2. Consider as a second example the case of the Swiss Federal Office for the ENvironment (FOEN) [1], an organization responsible for protecting against natural hazards and triggering alerts in case of floods in Swiss lakes and rivers. FOEN operates more than 260 water stations spread all over the country, each continuously collecting water discharge measurements. These water stations frequently suffer sensor failures or errors in the transmission of the data leading to missing blocks. However, the continuous monitoring of the current water level is crucial to swiftly warn about imminent threats such as floods.

Online recovery of missing values is a well-studied problem and a number of algorithms have been proposed in the past to tackle it. There is, however, a lack of practical techniques able to efficiently recover multiple incomplete streams by achieving real-time complexity, which is an essential requirement for any streaming algorithm. Existing online algorithms can be classified into two classes, according to the underlying method they use. On one hand, matrix completion techniques operate by constructing a matrix of series and then applying a low-rank reduction of this matrix to derive the principal dimensions that represent the streams. Each time the stream is updated, an approximated reduced matrix is computed out of the previous one, which limits the accuracy of these techniques. On the other hand, pattern-based techniques...
apply similarity techniques to detect repeating trends and use them to
derive replacements for the missing blocks. These algorithms
operate in an online manner by evaluating dependencies across
streams. The calculation of the exact set of dependencies requires a
large number of iterations rendering the recovery inefficient.

In practice, time series streams often exhibit temporal and spatial
similarity that can be used to summarize the main properties of
the data. We leverage this observation in our work by incrementally
computing a handful of dimensions, which conceptually represent
the up-to-date streams, and use them to recover missing values in
a continuous fashion. The proposed online recovery efficiently
implements an exact incremental version of the Centroid Decom-
position (CD) technique. It achieves an accurate recovery thanks
to the ability of CD to embed the correlation across time series
as part of its decomposition process [19, 21]. Our incremental ap-
proach can be beneficial to various applications where CD has been
applied such as factor analysis [10], text classification [15], image
compression [17], or anomaly detection [23].

At a technical level, CD decomposes an input matrix $X$ into the
product of two matrices, $X = LR^T$, where $L$ and $R$ are called the
loading and relevance matrix, respectively. The most expensive
operation in CD is the computation of maximizing sign vectors, $Z$,
which consist of $1$s and $-1$s and are used as part of an optimization
problem that must be solved to obtain the decomposition. The only
online CD approach, called UpdateCD [33], recomputes the sign
vectors from scratch each time new data arrives, thus achieving the
same quadratic time complexity as the batch algorithm [20].

In this paper, we propose to vastly improve the efficiency of
the Centroid Decomposition technique. Our algorithm extends the
traditional iterative way of calculating the sign vectors using two
techniques. First, it uses an anticipatory technique that speeds up
the CD computation by predicting the termination of the sign vector
search and thus reduces the number of iterations. Second, it utilizes
the matrix similarity before and after new data arrives to compute
the sign vectors incrementally.

In summary, the main contributions of this paper are as follows:

- We introduce a new algorithm called ORBITS for the Online
  Recovery of missing Blocks In multiple Time Series streams.
  ORBITS relies on an anticipatory computation of the CD
  technique, which reduces its time complexity from quadratic
to linear (Section 4).
- We prove the dependence between sign vectors before and
  after each stream update. We use this property to prove the
correctness of our decomposition, which guarantees an
accurate recovery of missing values (Section 5).
- We evaluate ORBITS on various time series datasets and
  show that our technique substantially outperforms the state
  of the art both in accuracy and efficiency (Section 6).

2 RELATED WORK
We describe, in turn, related work on (a) streaming recovery tech-
niques and (b) Centroid Decomposition.

2.1 Streaming Recovery Techniques
Online matrix completion algorithms have been used for the
continuous recovery of missing values. They assume that the
streams present a temporal continuity and apply matrix decompo-
sition/factorization techniques for the recovery.

Incremental SVD (ISVD) [8] is the first online matrix completion
technique that was proposed. It continuously recovers a matrix
with missing values using the Singular Value Decomposition (SVD)
method [34]. ISVD first constructs an orthogonal random matrix $S$
and uses it to approximate the observed values in $X$. Then, the
columns of $S$ are appended with the data and decomposed using
SVD, recovering missing values in the process. To preserve the
orthogonality of the decomposition, ISVD requires column updates,
and cannot properly handle row updates, i.e., stream values (which
is the main focus of this paper).

SAGE (aka online GROUSE) [5, 40] is an online recovery algo-
rithm that relies on PCA (Principal Component Analysis) [16]. PCA
takes an $n \times m$ matrix and finds $n$ principal components vectors,
which better represent the dimensions of the initial data. SAGE
updates the principal components with the appended matrix rows
and applies a Stochastic Gradient Descent (SGD) procedure to
derive a new matrix such that their product approximates the input
matrix. The resulting matrix is used to replace the missing values
in the case of column or row updates. SAGE outperforms ISVD both
in accuracy and efficiency on column updates [18]. The gradient
process makes SAGE inefficient in long time series, and inaccurate
in the presence of outliers or lowly correlated time series.

In [27], the authors introduce another online PCA-based recov-
ergy technique, which we refer to as pcaMME. It first constructs
samples of a covariance matrix using a memory-efficient parti-
toning technique [26]. The samples are obtained by partitioning
the components of the input matrix into separate blocks. Then,
pcaMME incrementally derives the largest principal components
from the matrix samples, recovering initialized missing values in
the process. pcaMME relies on a fast approximation of the principal
components, which limits its accuracy.

Online pattern matching techniques have been used in this con-
text as well. These algorithms assume that close sensors can present
trend similarity, which is used to derive replacement values.

TKCM [36] identifies and uses repeating patterns (seasonality)
in the history of time series to recover missing blocks. It creates
a query pattern composed of the most recent measurements and
searches for the missing value in the most similar (non-overlapping)
pattern to the query pattern. TKCM is able to recover missing values
as they appear in time series that exhibit seasonal patterns. The
pattern search assumes a single base time series rendering the
technique unable to recover more than one series at a time.

OGDImpute [3] uses an auto-regressive (AR) model to recover
missing values in data streams. It initializes the missing values with
zeros and applies stochastic gradient to compute the AR coeffi-
cients. These coefficients are incrementally used to compute two
predictions: the first one for the observed values and the second
one for the missing values. Such a two-fold prediction allows for
learning the trend from the base time series and its similarity to the
reference time series. OGDImpute assumes autoregressive input
data, which is not the case in most real-world time series.

Yoon et al. [38] introduce a Multi-directional Recurrent Neural
Network (MRNN) technique to recover missing blocks. The pro-
posed technique contains an interpolation block and an imputation
block, which are simultaneously trained using a fixed size sliding
window. It learns the data dependencies by leveraging both the correlation within time series and the correlation across time series. MRNN resorts to a forward and backward propagation process, which is expensive to compute and can only handle time series that are linearly dependent on one another. Additionally, the performance of the MRNN model deteriorates in short time series because of the limited number of training samples.

Hybrid online techniques that use both matrix completion and pattern matching have also been investigated. For instance, SPIRIT [30, 31] combines PCA with an AR model to reduce a set of co-evolving streams to a small number of hidden variables. These variables summarize the most important features of the original data. For each variable, SPIRIT fits an AR model on historical values and lower-case letters to refer to elements of vectors/matrices. For example, X is a matrix, X_{ij} is the i-th row of X, X_{ij} is the j-th column of X and x_{ij} is the j-th element of X_{ij}. The isolated vectors that do not belong to a matrix will be denoted with a capital letter, e.g., V.

A time series X = \{(t_1, v_1), \ldots, (t_n, v_n)\} is a set of n temporal values v_i that are ordered according to their timestamps t_i. In the rest of the paper we omit the timestamps, since the values are ordered, and write the time series X_t = \{(0, 2), (1, 0), (2, -4)\} as the sequence X_1 = [2, 0, -4]. A time series stream S is an unbounded (i.e., never ending) time series. We write X = [X_1| \ldots | X_m] (or X_{n \times m}) to denote an n \times m matrix having m time series X_{ij} as columns and n values for each time series as rows.

A sign vector Z \in \{1, -1\}^n is a sequence [z_1, \ldots, z_n] of n unary elements, i.e., |z_i| = 1 for i = 1, \ldots, n. We use \times for scalar multiplications and \cdot for matrix multiplications. The symbol \| \| refers to the l-2 norm of a vector. Assuming X = [x_1, \ldots, x_n], then \|X\| = \sqrt{\sum_{i=1}^{n} (x_i)^2}.

### 3 Background

As we mentioned earlier, ORBITS implements an incremental version of the Centroid Decomposition (CD) technique (cf. Figure 1). In what follows, we first describe CD and its most challenging part, the maximizing sign vector. Then, we introduce our problem definition.

#### 3.2.1 Definition

CD is a matrix decomposition technique that decomposes an input n \times m matrix X into an n \times m loading matrix L and an m \times m relevance matrix R, such that X = LR^T. CD is computed iteratively where at each iteration i (i \in [1, m]), a new maximizing sign vector Z_i (described below) is computed and used to compute the i-th relevance and loading vectors, i.e., R_{ij} = \frac{X^T \cdot z_j}{\| z_j \|^2} and L_{ij} = X \cdot R_{ij}. Next, a matrix reduction step is applied in order to obtain the new relevance and loading vectors, i.e., X = X - L_{ij} \cdot R_{ij}^T. The decomposition returns m loading and relevance vectors, of size n and m respectively. The most challenging part of the CD of a matrix X is the computation of the vector Z_i.

---

**Figure 1**: Graphical illustration of the recovery of the missing block (MB) in X_3 using ORBITS. (a) A plot of three input time series X_1, X_2 and X_3 with a missing block represented by a dashed line at 13 \leq t \leq 21. (b) ORBITS’S recovery, where CD decomposes the original data, exposing the matrix rank with dimensions L_1, L_2 and L_3.
3.2.2 Maximizing Sign Vector. Given an \( n \times m \) matrix \( X \), the maximizing sign vector \( Z \) is the sign vector that maximizes the centroid value \( \|X^T \cdot Z\| \), i.e., \( Z \) satisfies the following equation:

\[
\arg \max_{Z \in \{1, -1\}^n} \|X^T \cdot Z\|. 
\] (1)

The SSV algorithm \([20]\) is the most efficient batch solution to solve Eq. (1). SSV is based on the derivation of a new and equivalent optimization problem:

\[
\arg \max_{Z \in \{1, -1\}^n} \|X^T \cdot Z\| = \arg \max_{Z \in \{1, -1\}^n} Z^T \cdot \text{diag}^{\oplus}(X \cdot X^T) \cdot Z
\]

where \( \text{diag}^{\oplus}(\cdot) \) is an auxiliary function that sets the diagonal values of a square matrix to 0, and \( X \cdot X^T \) is the correlation matrix in case \( X \) is z-score normalized.

To solve the new optimization problem, the SSV algorithm starts by initializing a column vector \( Z = [1, \ldots, 1]^T \) of length \( n \), and iteratively computes a weight vector \( V = \text{diag}^{\oplus}(X \cdot X^T) \cdot Z \). The elements of \( V \) are defined as:

\[
v_i = z_i (z_i \cdot X_{i\cdot} - (X_{i\cdot} - (X_{i\cdot})^T))
\] (2)

where \( v_i \) is the \( i \)-th element of \( V \) and \( S = \sum_{i=1}^{n} z_i \cdot (X_{i\cdot})^T \).

Next, the signs of \( Z \) and \( V \) are compared, for each of their values \( i \in \{0, \ldots, n\} \). SSV iteratively flips the sign of \( z_{p_{\text{pos}}} \) (the element at position \( p \) in \( Z \)) from plus to minus, such that \( |\sum_{p_{\text{pos}}} z_{p_{\text{pos}}} x_{p_{\text{pos}}}^T| \) is maximized. Once \( Z \) and \( V \) have the same pairwise sign in all positions, SSV terminates and returns the correct maximizing sign vector. Note that SSV flips the sign of only one element at a time.

3.3 Problem Definition

Let \( X \) be an \( n \times m \) matrix of \( m \) time series each of length \( n \) and let \( \Delta X \) be an \( r \times m \) increment to \( X \) with \( r \ll n \). Let CD(X) be the Centroid Decomposition of \( X \). Our goal is to infer CD(\( X + \Delta X \)) directly from CD(X) without recomputing it from scratch. Unlike incremental linear algebra frameworks such as Linview \([29]\), which compute the change introduced by \( \Delta X \), our goal in this paper is to efficiently compute the whole decomposition of \( X + \Delta X \).

4 CD OPTIMIZATION

Before introducing our incremental decomposition algorithm, we present a new Anticipatory Sign Vector (ASV) technique to speed up the sign vector search and, subsequently, the CD computation. Our approach for optimizing the sign vector search consists of reducing the number of weight vectors and thus reducing the number of iterations to compute the sign vectors. ASV is based on an incremental computation of weight vectors (Section 4.1) and makes use of an anticipatory termination (Section 4.2).

4.1 Incremental Weight Vector

The optimization of sign vectors computation resorts to efficiently finding weights vectors (cf. Eq. (2)). We propose an incremental technique to compute weight vectors. This technique makes it possible to anticipate the termination of the maximizing sign vector search, which considerably reduces the number of iterations.

Definition 1 (Weight vectors are incremental). Let \( Z(k) \) be \( Z \) at iteration \( k \), \( P \) the set of positions flipped in \( Z(k) \) and let \( v_i \) be the \( i \)-th weight value in \( V \). For any two consecutive iterations of sign vectors, the weight vectors are linearly dependent, i.e.,

\[
v_i^{(k+1)} = v_i^{(k)} - 2 \times \sum_{p \in P} (X_{i\cdot} \cdot X_{p\cdot}^T) 
\] (3)

In the case where only one sign is flipped, \( \forall i \in [1, n] \setminus \{p\} \), Eq. (3) can be rewritten as follows:

\[
v_i^{(k+1)} = v_i^{(k)} - 2 \times (X_{i\cdot} \cdot X_{p\cdot}^T) 
\] (4)

with \( v_i^{(k+1)} = v_i^{(k)} \).

The incremental computation of weight vectors allows us to include a sequence of termination checks. Consequently, many weight vector computations do not have to be fully processed but can be interrupted once a termination condition is met. This incremental definition makes it possible to compute the correct maximizing sign vectors but in fewer iterations (compared to the SSV technique \([20]\)). Note that in order to incrementally compute the weight vectors, the first weight vector is computed using Eq. (2).

Example 1. To illustrate the incremental computation of the weight vectors, consider an input matrix \( X \) that contains two time series of five elements each, i.e.,

\[
X =\begin{bmatrix}
5 & 1 \\
-10 & 5 \\
-9 & 4 \\
4 & 6 \\
2 & -4
\end{bmatrix}
\]

For the sake of simplicity, we illustrate the case where only one sign flip is performed. First, \( Z \) is initialized with \( Is \), i.e., \( Z(1) = \{1, 1, 1, 1, 1\}^T \) and the initial weight vector is computed using Eq. (2) to get \( V(1) = \{-54, 15, 23, -12, -84\}^T \). Three elements of \( Z(1) \) have a different sign from the corresponding elements in \( V(1) \) and among them the element in the 5th position has the highest absolute value. Using \( p = 5 \), the next weight vector is incrementally computed (using Eq. (4)) as follows

\[
v_1 = -54 - 2 \times \left(\begin{bmatrix} 5 & 1 \end{bmatrix} \times \begin{bmatrix} 2 \\ -4 \end{bmatrix}\right) = -66
\]

\[
\vdots
\]

\[
v_5 = -84
\]

i.e.,

\[
Z(2) = \begin{bmatrix} 1 \\
1 \\
1 \\
1 \\
-1
\end{bmatrix}
\]

and \( V(2) = \begin{bmatrix} -66 \\
95 \\
20 \\
-84
\end{bmatrix} \).

4.2 Anticipatory Sign Vector

Anticipatory sign vector is a new algorithm to efficiently compute the maximizing sign vector by including a sequence of termination checks. After every weight vector has been found, an anticipatory sign vector is computed, using the signs of the weight vector, and the flipping process is evaluated. This process can therefore be seen as a series of sign multi-flips where the final step returns the maximizing sign vector.
And finally, \( \mathbf{v}^{(k)} \) be the weight vector obtained at iteration \( k \). The vector \( \mathbf{Z}_A^{(k)} \) where each \( z_i \in \mathbf{Z}_A^{(k)} \) gets the same sign as \( v_i \in \mathbf{v}^{(k)} \) is called anticipatory sign vector.

**Lemma 1 (Anticipatory termination).** Let \( \mathbf{Z}_A^{(k)} \) be the anticipatory sign vector at iteration \( k \) and let \( \mathbf{v}^{(k)} \) be its corresponding weight vector. We can safely terminate the flipping process if the pairwise elements in \( \mathbf{Z}_A^{(k)} \) and \( \mathbf{v}^{(k)} \) have the same sign.

**Proof.** We prove that the anticipatory termination yields the correct maximizing sign vector. This proof follows from the definition of the anticipatory sign vector. Since \( \mathbf{Z}_A^{(k)} \) contains all possible sign flips, then \((\mathbf{Z}_A^{(k)})^T \mathbf{v}_A^{(k)} \geq (\mathbf{Z}_A^{(k+1)})^T \mathbf{v}^{(k+1)}\). In case \( \mathbf{Z}_A^{(k)} = \mathbf{Z}_A^{(k+1)} \) (termination condition), then \( \exists \; p \) s.t. \((\mathbf{Z}_A^{(k+p)})^T \mathbf{v}_A^{(k+p)} > (\mathbf{Z}_A^{(k)})^T \mathbf{v}^{(k)}\). Subsequently, \( \forall p \in \mathbb{N}, (\mathbf{Z}_A^{(k)})^T \mathbf{v}_A^{(k)} = \max((\mathbf{Z}^{(k)})^T \mathbf{v}^{(1)}, \ldots, (\mathbf{Z}^{(k)})^T \mathbf{v}^{(t)}, \ldots, (\mathbf{Z}^{(k+p)})^T \mathbf{v}^{(k+p)}) \). Thus, \( \mathbf{Z}_A^{(k)} \) is the maximizing sign vector. □

The anticipatory sign vector is computed as outlined in Algorithm 1. Given a matrix of \( m \) time series and an initial sign vector \( \mathbf{Z} \), we start by computing the initial weight vector \( \mathbf{V} \). We assume for now that \( \mathbf{Z} \) contains only 1s and we show in Section 5.1 how to optimally initialize it. Next, the anticipatory sign vector is computed (line 3) and the weight vector \( \mathbf{V}_A \) is initialized with the weight vector \( \mathbf{V} \) (line 4). Then, we search for the set of positions of \(-1\)s in \( \mathbf{Z}_A \), called \( P \), that have not been flipped yet (line 5). \( P \) is used to determine the positions that need to be flipped in the weight vector. And finally, \( \mathbf{V}_A \) is updated according to Eq. (3) (lines 7-8). If no position is left to be flipped in \( \mathbf{Z}_A \), i.e., \( pos = 0 \), then the anticipatory sign vector is returned as the maximizing one. Otherwise, the next weight vector is incrementally computed (lines 14-16). By using the anticipatory technique, we are able to find the required sign flips without going through the full sign flipping process.

Let us illustrate the anticipatory termination in the example depicted in Figure 2. The upper part of the figure shows the standard flipping process, SSV, while the lower part shows the anticipatory termination, ASV. We consider the same input matrix \( \mathbf{X} \) of our running example. As illustrated in the upper part of the figure, we start computing the weight values for each weight vector using Eq. (2). The last element in the first weight vector \( \mathbf{v}^{(1)} \) has the smallest negative (and sign opposite) value and thus, the sign of the element in position 5 in \( \mathbf{Z} \) is flipped. The standard approach applies the same procedure until no more elements need to be flipped (i.e., \( \mathbf{Z}^{(4)} \)). Using ASV on the other hand, as illustrated in the lower part of the figure, \( \mathbf{Z}_A^{(1)} \) gets the same sign as the elements in \( \mathbf{v}^{(1)} \) and subsequently \( \mathbf{V}_A^{(1)} \) is computed. The termination condition is met, i.e., \( \mathbf{Z}_A^{(1)} \) and \( \mathbf{v}^{(1)} \) have the same pairwise element sign. Hence, ASV may immediately stop the flipping process, whereas the standard flipping process takes three additional iterations to terminate. As illustrated in this example, ASV returns the same maximizing sign vector while requiring less than half of the number of iterations.

**Algorithm 1: ASV(\( \mathbf{X}, \mathbf{Z} \))**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>maximizing sign vector ( \mathbf{Z} \in {1, -1}^m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbf{V} )</td>
<td>( \mathbf{Z}_A )</td>
<td>( \mathbf{V}_A )</td>
</tr>
</tbody>
</table>

**Proof.**

**Algorithm 1:**

1. Compute initial weight vector \( \mathbf{V} \) using Eq. (2).
2. Repeat
   - \( \mathbf{Z}_A \) := \( \{ z_i \in \mathbf{Z}_A | z_i = \frac{v_i}{\mathbf{V}_A} \} \);
   - \( \mathbf{V}_A := \mathbf{V} \);
   - \( P := \{ i \in \mathbf{Z}_A | z_i \neq \mathbf{V}_A \} \);
   - For each \( k \in P \)
     - \( \mathbf{V}_A := \mathbf{V}_A - 2 \times \mathbf{X} \times \mathbf{v}_k \);
     - Using Eq. (3);
     - \( \mathbf{v}_k := \mathbf{v}_k + 2 \times \mathbf{X}_k \times \mathbf{X}_k \);
     - If \( \mathbf{v}_k \) is the \( k \)-th element of \( \mathbf{V}_A \)
       - \( pos := \{ i \in \mathbf{Z}_A | \mathbf{v}_j \in \mathbf{V}_A \} \);
       - Anticipatory termination
   - If \( pos = 0 \)
     - \( \mathbf{P} := pos \);
     - \( \mathbf{Z} := \mathbf{Z}_A \);
   - Else
     - \( \mathbf{P} := \{ i \in \mathbf{Z}_A | \mathbf{v}_j \in \mathbf{V}_A \} \);
     - \( \mathbf{Z}_A := \mathbf{V}_A - 2 \times \mathbf{X} \times \mathbf{v}_j \);
     - Compute incremental weight vector;
3. Until \( \mathbf{P} = 0 \); return \( \mathbf{Z} \in \{1, -1\}^m \);

**Figure 2: Anticipatory Termination Example.**

5 INCREDENTAL CENTROID DECOMPOSITION

Our incremental Centroid Decomposition is a two-fold procedure. First, it applies an incremental initialization of sign vectors, which leverages the matrix similarity before and after each time series update. Second, it uses the ASV algorithm to efficiently find the corresponding sign vectors. We begin by introducing the initialization technique, before introducing our incremental algorithm and proving its correctness. Finally, we present ORBITS, which uses the incremental CD technique to recover blocks of missing values.

5.1 Sign Vector Initialization

The intuition behind our incremental initialization revolves around the idea that matrices before and after each row update share similar properties. This similarity reflects the temporal continuity in time series. Our proposed approach leverages this matrix similarity to
compute the maximizing sign vectors of the updated matrix using the sign vectors cached from the previous CD computation.

**Lemma 2 (Matrix Similarity).** Let $\tilde{X}$ be the matrix resulting from incrementing an $n \times m$ matrix $X$ with an $r \times m$ matrix $\Delta X$ and let $A_{\text{r}i}$ be the $i$-th row of $\Delta X$. Let also $\tilde{Z} \in \{-1,1\}^n$ and $\tilde{Z} \in \{-1,1\}^{n+r}$ be two sign vectors. Then, the following holds:

$$\max \| \tilde{X}^T \cdot \tilde{Z} \| \leq \max {\| X^T \cdot Z \| + \sum_{i=1}^{r} \| A_{\text{r}i} \|}$$

**Proof.** The proof is given in Appendix A. □

Lemma 2 shows the relationship between the maximizing sign vector before any row update of $X$ and the one after the update. It states that each update of the input matrix augments the centroid values with the norm of the added data. We use the sign vector computed before the update as input to find the maximizing sign vector of the incremented matrix $\tilde{X}$. This initialization takes into consideration the fact that the length of the initial sign vector might not match the dimensions of $\tilde{X}$; the input matrix might have grown since the last computation of the algorithm and might thus require larger sign vectors than those already computed. We append additional $1$s (since the sign vectors consist of $1$s and $-1$s only) at the end of the sign vector until reaching the correct length. Note that appending $-1$s yields the opposite sign vector.

We now use the matrix similarity property to prove that our technique computes the correct sign vector, which guarantees an accurate recovery.

**Theorem 1 (Correctness).** Let $\tilde{X}$ be the resulting matrix of incrementing $X$ with $\Delta X$. Then, our technique returns the sign vector, $\tilde{Z}$, that maximizes $\| \tilde{X}^T \cdot \tilde{Z} \|$.

**Proof.** Using $Z^{(1)}$ as $Z$ at the first iteration of the algorithm, we introduce the two following vectors. Let $\tilde{Z}$ be the resulting sign vector obtained by batch CD (i.e., $\tilde{Z}^{(1)} = \{1, \ldots, 1\}$) and let $\hat{Z}_{\text{c}}$ be the resulting sign vector obtained by vector technique with (i.e., $\hat{Z}^{(1)}_{\text{c}} = [Z_{\text{c}}, 1, \ldots, 1]$; $Z_{\text{c}}$ is the cached sign vector). Proving the correctness of our technique boils down to proving that:

$$\arg \max \| \tilde{X}^T \cdot \hat{Z}_{\text{c}} \| \equiv \arg \max \| \tilde{X}^T \cdot \tilde{Z} \|$$

Let $I$ be an identity matrix, $D$ be a diagonal matrix containing $Z_{\text{c}}^{(1)}$, i.e., $D = \text{diag}(Z_{\text{c}}^{(1)})$, and let $\tilde{X}_{\text{D}}$ be an $(n+r) \times m$ matrix s.t. $\tilde{X}_{\text{D}}^T = \tilde{X}^T \cdot D$. Let also $\tilde{Z}_{\text{D}}$ be an $(n+r)$ sign vector s.t. $\tilde{Z}_{\text{D}} = D \cdot \hat{Z}_{\text{c}}$.

First, we prove the following:

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \| \equiv \arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

Since $D$ is a signature matrix where the diagonal elements are $+1$ or $-1$, then $D \cdot D = I$. It follows:

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot \hat{Z}_{\text{c}} \| \equiv \arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

By definition of $D$, we have $d_{ij} \cdot \hat{Z}_{\text{c}}^{(1)} = 1$, $\forall i \in \{1, \ldots, (n+r)\}$ where $\hat{Z}_{\text{c}}^{(1)} \subseteq \hat{Z}_{\text{c}}$, which yields $D \cdot \hat{Z}_{\text{c}}^{(1)} = [1, \ldots, 1]$. Since $\hat{Z}_{\text{D}}^{(1)}$,

$$D \cdot \hat{Z}_{\text{c}}^{(1)}, \text{ we replace the argument } \hat{Z}_{\text{c}} \text{ by } \tilde{Z}_{\text{D}} \text{ and get}$$

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot D \cdot \tilde{Z}_{\text{D}} \| \equiv \arg \max \| (\tilde{X}_{\text{D}}^T \cdot D) \cdot \tilde{Z}_{\text{D}} \|$$

Putting (5) into (6), we get

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \| \equiv \arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

Next, we prove the following

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \| \equiv \arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

By definition of $\tilde{X}_{\text{D}}$ we have

$$\tilde{X}_{\text{D}} = \tilde{X}^T \cdot D = \begin{bmatrix} X & \text{diag}(Z_{\text{c}}) \cdot 0_{n \times r} \end{bmatrix} \begin{bmatrix} 0_{r \times n} \text{\; 1}_{1 \times r} \end{bmatrix} = \begin{bmatrix} (\tilde{X}^T \cdot \text{diag}(Z_{\text{c}})) \; \Delta X^T \end{bmatrix}$$

From (8), we can see that $\tilde{X}_{\text{D}}$ is $\text{diag}(Z_{\text{c}})^T \cdot X$ incremented with $\Delta X$. By applying Lemma 2 on $\tilde{X}_{\text{D}}$, we get

$$\max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \| = \max \| \tilde{X}_{\text{D}}^T \cdot \text{diag}(Z_{\text{c}}) \cdot Z_{\text{c}} \| + \sum_{i=1}^{r} \| A_{\text{r}i} \|$$

Using (9) and Lemma 2, we get

$$\max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \| = \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

Since the two equations in (10) use the same initial maximizing sign vector that contains only $1$s. Thus, (10) can be rewritten as follows:

$$\arg \max \| \tilde{X}_{\text{D}}^T \cdot \hat{Z}_{\text{c}} \| \equiv \arg \max \| \tilde{X}_{\text{D}}^T \cdot \tilde{Z}_{\text{D}} \|$$

Therefore, our technique computes the same maximizing sign vector as the batch CD, which concludes the proof. □

**Example 2.** To illustrate the incremental initialization after appending multiple rows, consider the input matrix of our running example. The two time series have been sequentially appended with (0, 8) and (2,7), respectively. Figure 3 illustrates the initialization and the computing of the maximizing sign vector. For the sake of simplicity, we illustrate the flipping at the same time of one element only. To compute the first maximizing sign vector of $X$, we use the sign vectors computed before the update. The incremental initialization first takes the (cached) sign vector $Z_1$ as the initial sign vector. Then, it appends $1s$ at the end of each cached sign vector and starts flipping the signs to find the maximizing sign vectors for $X$. At each iteration $k$ and for each sign vector, a pair of vectors $\tilde{Z}(k)$ and $\tilde{V}(k)$ is computed. The weight vectors $V$ are incrementally computed according to Eq (3) (similarly to Example 1).
We now present the InCD algorithm that implements our incremental solution. First, the sign vector used to initialize the search for a minimal cache footprint. Assuming that the input matrix \(X\) is already loaded in memory, we apply the same procedure to compute the second maximizing sign vector. The runtime complexity of InCD is dominated by the call of AsV(\(\Delta X\)), which is performed \(m\) times. AsV uses the same definition of weight vectors as the SSV algorithm and, thus, an element that has been flipped can not be flipped again (see Lemma 5 in [20]).

Algorithm 3: ORBITS (X, \(\Delta X, T^e\))

```
Input : \(n \times m\) input matrix \(X\), \(r \times m\) matrix of appended rows \(\Delta X\); List of missing time points \(T^e\)
Output: Matrix with recovered values \(\hat{X}\)
1 \(\hat{X} \leftarrow \) increment \(X\) with \(\Delta X\);
2 Linearly interpolate/extrapolate all missing values in \(\hat{X}\);
3 repeat
4 \(\hat{X} \leftarrow \hat{X}\);
5 compute reduction factor \(k\) of \(\hat{X}\);
6 \(L_k, R_k \leftarrow \) InCD(\(\Delta X, r, m - k\));
7 \(\hat{X}_k \leftarrow L_k \cdot (R_k)^T\);
8 // Update missing values
9 foreach \((i, j) \in T^e\) do
10 \(\hat{x}_{ij} \leftarrow y_{ij};\)
11 // \(y_{ij}\) element of \(\hat{X}_k\) at timestamp \(i\)
12 until \(||\hat{X} - \hat{X}\|_F < \epsilon||\);
13 return \(\hat{X}\);
```

Algorithm 3 describes the pseudocode of ORBITS. It takes as input a matrix \(X\) incremented with \(\Delta X\) (both matrices might contain a set of missing blocks, \(B\)) and a list \(T^e\) of pairs indicating the rows and columns of the missing values in \(X\). The recovery starts by initializing the missing values in \(X\) using either linear interpolation or zero, depending on the position of the missing values in \(X\). Then, we apply a reduction to the decomposition of \(X\) to return \(L_k\) and \(R_k\), which contain the first \(m - k\) columns of \(L\) and \(R\) respectively. We use the same procedure that applies to SVD [2], but we consider the centroid values instead of the eigenvalues [21]. Next, the values in \(\hat{X}\) with positions in size required to store \(X\), preventing any cache overflow. Finally, the found maximizing sign vector is used to sequentially compute the decomposition of \(\hat{X}\).
We rewrote all algorithms in C++, except for MRNN (inextricable from Python), using an advanced linear algebra library called Armadillo [33]. We re-engineered the original implementations, which led to gains in performance across all the algorithms – in one case making an algorithm (SPIRIT) 110x faster. We implemented all recovery algorithms in a non-distributed fashion as they rely on incremental computations, which are not easy to parallelize.

All the experiments were ran on a Linux machine with a 3.4 GHz Intel Core i7 and 128GB of RAM. All code and data (and additional experiments omitted for brevity) are publicly available.

The BAFU and Gas datasets were used in a recent benchmark that evaluates missing values recovery techniques [22]. The values of the observations are stored as 4-byte floating numbers while the sign vectors are binary arrays. All the time series have been z-score normalized.

### Table 1: Description of time series.

<table>
<thead>
<tr>
<th>Name</th>
<th>TS length</th>
<th># of TS</th>
<th>Main features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gas</td>
<td>1000</td>
<td>100</td>
<td>mixed correlations</td>
</tr>
<tr>
<td>Motion</td>
<td>10'000</td>
<td>20</td>
<td>non-periodic, local similarity</td>
</tr>
<tr>
<td>BAFU</td>
<td>50'000</td>
<td>10</td>
<td>periodic, time-shifts</td>
</tr>
<tr>
<td>Soccer</td>
<td>500'000</td>
<td>10</td>
<td>sudden change, anomalies</td>
</tr>
</tbody>
</table>

### Figure 4: Three sample time series from each dataset.

### 6.2 Recovery of Missing Values

In this section, we evaluate ORBITS under a variety of recovery scenarios. We compare its accuracy and efficiency against the state of the art in streaming recovery techniques, i.e., OGDImpute [3], SAGE [5], pcaMME [27], SPIRIT [31], TKCM [36] and MRNN [38]. For the parametric techniques, we use the parameters recommended by the authors. We report the results over three runs. To measure the accuracy, we adopt the most commonly used measure in this

$$T^\ast$$ are updated with their corresponding ones in $\tilde{X}_k = L_k \cdot R_k^T$. The recovery process terminates if the difference in the relative Frobenius norm $\|\tilde{X} - \tilde{X}'\|_F / \|\tilde{X}'\|_F = \sqrt{\sum_{i=1}^{|B|} \sum_{j=1}^{|B|} (\tilde{x}_{ij} - \tilde{x}'_{ij})^2 / |B|}$ between $\tilde{X}'$ and $\tilde{X}$ (where $\tilde{x}_{ij} \in \tilde{X}$, $\tilde{x}'_{ij} \in \tilde{X}'$ and $|B|$ is the length of the missing block) falls below a small threshold value $\epsilon$. We empirically set the value of $\epsilon$ to $10^{-6}$, which yields the best accuracy/precision trade-off. Since the entropy, used to find the reduction value $k$, minimizes the Frobenius norm as does the iterative process, then our recovery algorithm quickly converges (see Figure 4 in [21]).

In this section, we report our experimental results. We aim to evaluate: (1) the performance of ORBITS against the state-of-the-art online recovery techniques and (2) the impact of our sign vector computation on ORBITS’s performance.

### 6.1 Setup and Datasets

We rewrote all algorithms in C++, except for MRNN (inextricable from Python), using an advanced linear algebra library called Armadillo [33]. We re-engineered the original implementations, which led to gains in performance across all the algorithms – in one case making an algorithm (SPIRIT) 110x faster. We implemented all recovery algorithms in a non-distributed fashion as they rely on incremental computations, which are not easy to parallelize.

All the experiments were ran on a Linux machine with a 3.4 GHz Intel Core i7 and 128GB of RAM. All code and data (and additional experiments omitted for brevity) are publicly available.

Our evaluation was performed on real-world time series from a broad range of applications and which cover a wide range of characteristics and sizes (cf. Table 1). We used the following datasets:

- **Soccer**. This dataset was introduced in the DEBS challenge 2013 [28] and contains the position of players during a football match. The data originates from sensors located near the players’ shoes and the goal keeper hands. This dataset provides values recorded at very a high rate, i.e., the tracking frequency is equal to 200Hz yielding 15’000 position events per second. Soccer time series are bursty with lots of outliers.

- **MotionSense**. This dataset includes time series data generated by accelerometer and gyroscope sensors (attitude, gravity, userAcceleration, and rotationRate) [24]. The sensors measure different human activities at a high sampling rate of 50Hz using an iPhone 6s kept in the users’ front pockets. The motion time series are non-periodic, but exhibit partial trend similarities.

- **BAFU**. This dataset consists of water discharge time series provided by the BundesAmt Für Umwelt (BAFU) [1] for a platform facility situated at the ChemoSignals Laboratory at the University of California in San Diego [32]. This dataset is used to measure the impact of sustainable green infrastructures and contains different gases that present mixed correlation (positive/negative and high/low).

The BAFU and Gas datasets were used in a recent benchmark that evaluates missing values recovery techniques [22]. The values of the observations are stored as 4-byte floating numbers while the sign vectors are binary arrays. All the time series have been z-score normalized.

### Figure 4: Three sample time series from each dataset.
field [25]: the root mean square error (RMSE) between the original blocks and the recovered ones, i.e.,

$$RMSE = \sqrt{\frac{1}{T} \sum_{t \in T} (x_t - \hat{x}_t)^2}$$

where \( T \) is the set of missing values, \( x_t \) is the original value and \( \hat{x}_t \) is the recovered value.

6.2.1 Accuracy. We first consider the simplest case where a single time series contains only one missing block. To simulate an online recovery, we set the missing block to appear at the tip of a randomly chosen series and we vary the size of the missing block from 10% to 80% starting from the end of the series. We keep the length and number of the time series to their maximum per dataset. Figure 5 shows the RMSE results when varying the missing rate. We exclude MRNN and TKCM from the soccer experiment as both algorithms take more than 15 hours to complete one tick.

Next, we evaluate the recovery accuracy on different datasets when increasing the dataset size. We assume one incomplete series with a missing block of size 10% of the maximum series’ length. When the sequence length varies, the number of series is set to 10, while the sequence length is set to 1k values when the number of series varies. We use the average RMSE values across different datasets, with standard deviation as the confidence intervals. Figure 6 depicts the results.

First, we observe that ORBITS outperforms in general all recovery baselines and yields 25% and 30% RMSE improvement compared to SPIRIT and SAGE, respectively. The only exception is on the Motion dataset, where our technique achieves a similar recovery as SPIRIT. This is explained by the fact that this dataset presents local similar trends to which both techniques respond in a similar fashion. The hybrid principal components computed by SPIRIT in this case carry similar information to the loading vectors computed by ORBITS. Second, in the datasets with a small number of series, i.e., Motion, BAFU and Soccer, as expected, the RMSE of ORBITS increases with the missing rate. Surprisingly, in the Gas dataset, the RMSE shows a bell shape as the missing rate grows. This is explained by the fact that in the presence of a high number of series, larger missing blocks require a higher number of iterations which, in turn, yield better recovered values.

In the experiments of Figure 6b, we observe that all techniques take advantage of using longer time series per dataset. The results show also that ORBITS outperforms all techniques in all datasets. The only exception is when the number of time series is equal to five where MRNN achieves a slightly lower RMSE than ORBITS (0.59 vs 0.63). This is explained by the fact that, for such a low number of time series, MRNN produces a recovery that mimics the average, which turns out to yield a low error. However, compared to ORBITS, the recovery of MRNN does not preserve the shape of the missing block (as we will show later in this section).

In the previous set of experiments, we assumed that a missing block occurs in a single series in a dataset. In real-world applications, however, more than one sensor might break during multiple time periods. This yields multiple incomplete time series with one or more missing blocks each.

We observe that most techniques take advantage of using longer series to achieve a lower error (cf. Figure 6a). The improvement is more noticeable when we vary the length either from 1k to 5k or from 5k to 10k, depending on the technique. This is expected, because using more data helps these techniques to better capture the main features of the data. We observe also that ORBITS achieves the lowest standard deviation as its RMSE is steady across datasets.

In the experiments of Figure 6b, we observe that all techniques take advantage of using more time series per dataset. The results show also that ORBITS outperforms all techniques in all datasets. The only exception is when the number of time series is equal to five where MRNN achieves a slightly lower RMSE than ORBITS (0.59 vs 0.63). This is explained by the fact that, for such a low number of time series, MRNN produces a recovery that mimics the average, which turns out to yield a low error. However, compared to ORBITS, the recovery of MRNN does not preserve the shape of the missing block (as we will show later in this section).

In the previous set of experiments, we assumed that a missing block occurs in a single series in a dataset. In real-world applications, however, more than one sensor might break during multiple time periods. This yields multiple incomplete time series with one or more missing blocks each.

In the experiments in Figure 7, we evaluate the case of multiple incomplete time series where each has one missing block at the end (of size of 10% of the series). We keep the length and number of series to their maximum per dataset and we vary the number of incomplete series. TKCM and SPIRIT cannot handle more than one incomplete series and are hence discarded (cf. Section 2). SAGE requires every row of the input matrix to contain at least \( k \) non-missing values, where \( k \) is the reduction factor. Thus, this technique can process only a certain number of incomplete series.
series in the dataset. We exclude MRNN from the soccer experiment for the same reason explained before, completion time.

The results show that our technique achieves the lowest RMSE when varying the number of incomplete series. As expected, the RMSE of ORBITS increases with the number of incomplete series. This is explained by the fact that more incomplete series means more values to recover, which in turn yields a higher error. We note that the last point in the plot corresponds to a blackout, i.e., when all time series lose data at the same time. In this case, ORBITS terminates in few iterations, incurring a high RMSE. The recovery of ORBITS in the case of blackout is the same as pcaMMe because both techniques use similar initialization of the missing blocks.

Next, we consider the most frequent scenario where each incomplete time series has multiple missing blocks (of the total size of 10% per series). We set the missing blocks to appear completely at random (MCAR), i.e., missing blocks can be overlapping, disjoint or both at the same time. We keep the length and number of series to their maximum per dataset and vary the number of incomplete series. The results are depicted in Figure 8.

We observe that ORBITS achieves the lowest RMSE in all datasets and is on average between 30% (cf. Figure 8d) and 60% (cf. Figure 8a) more accurate than its closest competitor. Surprisingly, increasing the number of series did not always increase the RMSE of ORBITS. The reason is that the MCAR scenario yields relatively small blocks, which poses no challenge to our technique.

In addition to the RMSE analysis, we also compare the accuracy of the evaluated techniques to reconstruct the shape of the missing blocks. We compute the correlation between the original blocks and the recovered ones using Pearson and Spearman coefficients [11]. The two metrics capture different trends. The former captures linear relationships (e.g., aligned trends) while the latter captures non-linear relationships (e.g., shifted trends). The two metrics range between 1 and -1, where 1 stands for perfect positive correlation, -1 for perfect opposite correlation, and 0 for no correlation.

Figure 9 depicts the results of the correlation experiment where each time series has missing MCAR multiblocks of a total size of 10% per series. The results show that ORBITS achieves the highest Pearson and Spearman correlation values, preserving the shape of the original block. This is attributed to the weight vectors, computed
during the decomposition, which embed the correlation across the input series. We can also observe that only ORBITS and SAGE have on par Pearson and Spearman results: time shifts do not affect the accuracy of the recovery of both techniques. The remaining techniques are sensitive to non-linear relationships (e.g., OGD and MRNN have a close to zero Spearman correlation on the BAFU dataset). All techniques achieve their lowest correlation on the motion dataset, because motion time series contain a high number of irregular fluctuations.

6.2.2 Efficiency. We now evaluate the efficiency of the recovery techniques by measuring the runtime while varying the sequence length and number. We report the time on a log scale, since the results widely vary among algorithms.

In Figure 10 we incrementally increase the stream length, using all the series from the dataset. We consider the MCAR scenario on the BAFU and the Soccer datasets, which have the longest time series, 50k and 500k respectively.

The results show that ORBITS and pcaMME are the two most efficient techniques and are orders of magnitude faster than the remaining techniques. pcaMME relies on an approximation of the principal components yielding a low runtime. However, this approximation incurs a high recovery error (as shown earlier). Unlike pcaMME, our technique relies on a fast incremental computation of the centroid values rendering our recovery very efficient. The incremental computation computes the correct decomposition, which explains the low recovery error. ORBITS achieves a linear time complexity and takes 1.4 sec to recover 10 time series each with 500k (cf. Figure 10b).

![Figure 10: Efficiency with increasing series length.](image)

(a) BAFU dataset ($n = 50k$).  
(b) Soccer dataset ($n = 500k$).

In the experiments of Figure 11, we gradually increase the number of time series in a dataset while keeping their length fixed at their maximum. We choose the Gas and the Motion datasets for the same reason as above (their size). We observe the same trends as in Figure 10. ORBITS stands out among the faster peers when using a few series per dataset. Our technique achieves a sub-linear complexity with the number of series. The efficiency of our technique is explained by the careful computation of the appropriate reduction factor. At each iteration, ORBITS finds the reduction factor that yields the best accuracy/precision trade-off. For larger number of series, the centroid vectors (ORBITS) and the principal components (pcaMME and SAGE) require a similar number of iterations to be computed and, subsequently, the same runtime.

![Figure 11: Efficiency with increasing series number.](image)

(a) Motion dataset ($m = 20$).  
(b) Gas dataset ($m = 100$).

Figure 12: Efficiency with increasing the number of series on augmented datasets.

Finally, we evaluate the impact of series’ number on the recovery runtime using large-scale datasets. We exclude MRNN from this experiment for the same reason mentioned earlier. We augment both the BAFU and Soccer datasets with more time series. To do so, we first generate synthetic time series that have similar properties as the existing ones using the Weighted_DBA technique [12]. Next, we alter the time series with noise following a Gaussian distribution $N(0,0.2)$ and with exponential smoothing. This allows us to obtain time series that exhibit similar shape but with some local dissimilarities. Finally, we apply a full permutation of blocks of ten series. The results are depicted in Figure 12.

We observe a similar trend as in Figure 11, ORBITS stands apart among the rest of the techniques when using a few series (cf. Figures 12a and 12b). The runtime difference between ORBITS and pcaMME becomes, however, indistinguishable as the number of series increases. SAGE becomes slower because SGD is more expensive to compute on larger matrices. It is worth to notice that the efficiency of pcaMME comes with the cost of a low accuracy.
We observe similar results when varying the missing rate. The good performance of ASV is attributed to its ability to leverage two inherent properties of time series data: the spatial continuity (similarity between time series) and the temporal continuity (similarity between the existing data and the appended one). In the unlikely case where the two conditions do not hold at the same time, ASV would not early terminate and would require the same number of iterations as SSV. The incremental initialization would, however, still improve the runtime of ORBITS by up to 2x.

By transposing both sides of (12), multiplying each side by $\hat{Z}$ and applying the $l_2$ norm, we get

$$
\|\hat{X}^T \cdot \hat{Z}\| = \|(Y + M)^T \cdot \hat{Z}\|
$$

The computation of the right hand side of (13) gives

$$
Y^T \cdot \hat{Z} = \left[ X^T 0_{1}^{T}, \cdots, 0_{r}^{T} \right] \cdot \begin{bmatrix} 
\tilde{z}_{n+1} \\
\vdots \\
\tilde{z}_{n+r} 
\end{bmatrix}
= X^T \cdot Z + \tilde{z}_{n+1} \times 0_{1}^{T} + \cdots + \tilde{z}_{n+r} \times 0_{r}^{T}
= X^T \cdot Z.
$$

Equation (16) is valid for all $\hat{Z}$ and $Z$ including the maximizing one, i.e., $Z_{\text{max}}$. It follows:

$$
\|\hat{X}^T \cdot \hat{Z}_{\text{max}}\| = \|X^T \cdot Z_{\text{max}} + \tilde{z}_{n+1} \times A_{1}^{T} + \cdots + \tilde{z}_{n+r} \times A_{r}^{T}\|.
$$

The application of time series temporal continuity to (17) gives

$$
\|\hat{X}^T \cdot \hat{Z}_{\text{max}}\| = \|X^T \cdot Z_{\text{max}}\| + \sum_{i=1}^{r} \|\tilde{z}_{n+i} \times A_{i}^{T}\|.
$$

Since $\hat{Z}_{\text{max}}$ is the maximizing vector of $\|\hat{X}^T \cdot \hat{Z}\|$, and $Z_{\text{max}}$ is the maximizing sign vector of $\|X^T \cdot Z\|$, then we get

$$
\max \|\hat{X}^T \cdot \hat{Z}\| = \max \|X^T \cdot Z\| + \sum_{i=1}^{r} \|\tilde{z}_{n+i} \times A_{i}^{T}\|.
$$

We have $\tilde{z}_{n+i} = \pm 1$, hence $\forall i \in \{1, \cdots, r\}$, $\|\tilde{z}_{n+i} \times A_{i}^{T}\| = \|A_{i}^{T}\|$ and $\|A_{i}^{T}\| = \|A_{i}\|$. Therefore, we have

$$
\max \|\hat{X}^T \cdot \hat{Z}\| = \max \|X^T \cdot Z\| + \sum_{i=1}^{r} \|A_{i}\|
$$

which concludes the proof.\[\square\]