Department of Informatics, University of Zürich

**Master Projektarbeit**

# Recovery of Missing Values based on Centroid Decomposition

Eszter Börzsönyi

Matrikelnummer: 12-736-161

Email: sb.eszti@gmail.com

January 29, 2013

Supervised by Prof. Dr. M. Böhlen and M. Khayati

University of Zurich
Department of Informatics

**Abstract**

In the area of information technology large amount of data are generated and stored day by day. Observation data - like measurements - often contain missing values by nature. The goal of this work is to investigate the application of the Centroid Decomposition algorithm for the recovery of missing values in shifted time series. We provide an extensive set of experiments to evaluate the scalability of our implementation. We apply our implementation to recover shifted missing blocks in real world hydrological time series and identify the classes of time series that can be recovered using this technique. As part of the implementation, we propose a graphical tool that can perform the Centroid Decomposition of matrices and recover missing blocks in hydrological time series.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Context of Work

My master group project was carried out as a guest student within the Database Technology Group at the University of Zürich. The main motivation of the work was to handle blocks of missing values [1] that frequently occur in hydrological datasets. The data we have worked on, was provided by the a hydrological company [2]. This data contains hydrological measurements such as temperature, precipitation, humidity, air pressure, wind speed and water level computed over the time in different areas around the region of South Tyrol in Italy. Due to sensor malfunction, both single missing data and blocks of missing data occur frequently in the datasets. The percentage of missing data in the different time series [3] sets is follows: for the temperature set is $\sim 20$ %, for the humidity set is $\sim 6$ %, for the wind speed set is $\sim 23$ %, for the precipitation set is $\sim 16$ %, for the air pressure set is $\sim 11$ %, for the water level set is $\sim 66$ %.

The main goal of our work was to implement a scalable algorithm, and using it to be able to recover the missing values. Within the project a recovery tool has been implemented and different experiments have been executed.

## 1.2 Motivation and Contribution

Our motivation was the lack of tools that are able to recover efficiently blocks of missing values in time series, e.g., REBOM [4] is a graphical tool that takes more than 15 minutes to recover missing blocks in time series containing 100000 observations. This long run time makes this tool non usable for long time series. In this report, we firstly worked on a scalable implementation of the algorithm Centroid Decomposition, used later for the recovery. We developed a tool with a graphical user interface for the recovery of missing data, based on iterative refinement of missing values by applying Centroid Decomposition [5] and dimensionality reduction technique [6]. Our recovery tool gives possibility to display and discover time series, explore and recover missing values. We found out that using our algorithm, the efficient recovery of missing values in shifted time series is also possible. Then, we evaluated the accuracy of our recovery algorithm for missing values in shifted time series, and identified the classes of time series that can be recovered using our solution.

## 1.3 Hydrological Time Series

Our hydrological time series are grouped into sets. Each set represents different weather phenomena like temperature, precipitation, humidity, air pressure, wind speed and water level. Each time series set contains measurements from different measurement stations around the region of South Tyrol in Italy. We consider a sequence of measurements as a time series. Each set of time series has its own granularity, i.e., the time frequency measurements. Time series from temperature set have the granularity 2 while the other sets are measured with granularity 1. Each time series is defined by an id, and dispose a description about its measurement area. Time series contain sequence of observations. Each observation has a timestamp and a corresponding value.

## 1.4 Structure of Report

This report is structured as follows: in Chapter 2, we define the concepts time series and missing values. In Chapter 3, we give a detailed introduction of the Centroid Decomposition technique. In Chapter 4, we introduce our implementation and we show the results of our experiments to evaluate the scalability of our implementation. In Chapter 5, we introduce our recovery algorithm and the developed recovery tool. We apply our implementation to recover missing blocks of time series and identify the classes of time series that can be efficiently recovered using this technique. In Chapter 6, we summarize the main contributions of this work and enumerate some limitations that can be extended as future work.

# 2 Background

## 2.1 Notations

| Symbol | Definition |
|---|---|
| $\mathcal{S}_i$ | denotes a time series |
| $t_j$ | denotes a timestamp |
| $(y_{t_j}, t_j) \in S_i$ | denotes observation in any time series $\mathcal{S}_i$, where $y_{t_j}$ is the measured value at the timestamp $t_j$ |
| $\mathcal{M}_{\mathcal{S}_i}$ | denotes the sequence of missing values in any time series $\mathcal{S}_i$ |
| $\mathcal{T}_{\mathcal{S}_i}$ | denotes the sequence of timestamps contained by $\mathcal{S}_i$ |

## 2.2 Time Series

A time series is an ordered sequence of observations captured over time. We denote an observation that belongs to a time series $\mathcal{S}$ as: $(t, v_t) \in \mathcal{S}$, where $t$ denotes a timestamp, $y_t$ the observed value. A times series contains a finite number of observations and is denoted as follows:

$$\begin{aligned} \mathcal{S} &= \{(y_{t_1}, t_1), (y_{t_2}, t_2), \dots (y_{t_n}, t_n)\} \\ \textbf{If } t_i < t_j &\Rightarrow y_{t_i} \text{is measured at a timepoint before } y_{t_j} \end{aligned}$$

Time series can be classified into two groups: aligned time series and shifted time series.

### 2.2.1 Aligned time series

Two time series $\mathcal{S}_1$ and $\mathcal{S}_2$ are considered aligned on the time window $[t_i, t_k]$, if they contain exactly the same sequence of timestamps on the time interval $[t_i, t_k]$ and each observation in $\mathcal{S}_1$ has a corresponding observation in $\mathcal{S}_2$. Formally, $\mathcal{S}_1$ and $\mathcal{S}_2$ are aligned on the time interval $[t_i, t_k]$, if

$$\forall t_j \in [t_i, t_k], \quad (y_{t_j}, t_j) \in \mathcal{S}_1 \rightarrow (y'_{t_j}, t_j) \in \mathcal{S}_2 \tag{2.1}$$

### 2.2.2 Shifted time series

Time series $\mathcal{S}_2$ is shifted with a time shift $\delta$ with respect to time series $\mathcal{S}_1$ on the time interval $[t_i, t_k]$, if $\mathcal{S}_2$ contains exactly the same number of timestamps on the time interval

$[t_i + \delta, t_k + \delta]$ and each observation in $\mathcal{S}_1$ on time interval $[t_i, t_k]$ has a corresponding observation in $\mathcal{S}_2$ on time interval $[t_i + \delta, t_k + \delta]$. Formally $\mathcal{S}_1$ and $\mathcal{S}_2$ are shifted if:

$$\forall (y_{t_j}, t_j) \in [t_i, t_k] \in \mathcal{S}_1 \rightarrow (y'_{t_j}, t_j + \delta) \in [t_i + \delta, t_k + \delta] \in \mathcal{S}_2$$

## 2.3 Missing Data

Missing data occur in real world datasets mainly due to a damage that affects the sensors that capture the measurements. Missing data may introduce some inconsistencies in performing any data analysis.

Given two time series $\mathcal{S}_1$ and $\mathcal{S}_2$ measured over the same time interval $[t_1, t_n]$.

Time series $\mathcal{S}_2$ contains missing values respect to time series $\mathcal{S}_1$, if $\mathcal{S}_2$ misses some of the timestamps, what times series $\mathcal{S}_1$ contains. Let us denote the sequence of missing values in any time series $\mathcal{S}_i$ with $\mathcal{M}_{\mathcal{S}_i}$, and denote with $\mathcal{T}_{\mathcal{S}_i}$ the sequence of timestamps contained by $\mathcal{S}_i$.

$$\mathcal{T}_{\mathcal{S}_i} = \{\forall t_j \,|\, (y_j, t_j) \in \mathcal{S}_i\}$$

Formally, the sequence of missing values in $\mathcal{S}_2$ respect to $\mathcal{S}_1$ can be written as follows:

$$\mathcal{M}_{\mathcal{S}_2} = \{t_j \mid t_j \in \mathcal{T}_{\mathcal{S}_1} \wedge t_j \notin \mathcal{T}_{\mathcal{S}_2}\}$$

Missing data can occur as single missing observations and also as blocks of missing observations. We propose to implement a method, that can recover accurately and missing observations using existing observations in shifted time series.

# 3 Centroid Decomposition

## 3.1 Notations

| Notation | Description |
|---|---|
| $A$ | denotes the $n \times m$ input matrix of decomposition |
| $\mathbf{a}_i$ | denotes the $i^{th}$ row of $A$ |
| $n$ | denotes the number of rows |
| $m$ | denotes the number of columns |
| $B$ | denotes the $n \times m$ loading matrix |
| $V$ | denotes the $m \times m$ factor matrix |
| $\mathbf{v}_i$ | denotes the $i$th column of $V$ called centroid factor |
| $\mathbf{b}_i$ | denotes the $i$th column of $B$ called loading vector |
| $\mathbf{z}$ | denotes the sign vector |

## 3.2 Centroid Decomposition

### 3.2.1 Input matrix

Let us assume, we have $m$ entities, what can be measured (like temperature in $m$ different place). We have $n$ variables what can take values from a defined value set (like $n$ different time point in a day what can take the values of temperature in that time point). We construct a matrix, whose columns represent the $m$ entities and whose rows represent the $n$ variables.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \cdots & a_{1m} \\ a_{21} & \ddots & \cdots & \cdots & a_{2m} \\ \vdots & \cdots & a_{ij} & ... & \vdots \\ \vdots & \cdots & \cdots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{nm} \end{bmatrix}$$

In this terminology every element of the matrix means

$$a_{ij} = \text{the measured value of variable } i \text{ on entity } j.$$

### 3.2.2 Decomposition

Centroid Decomposition is a matrix decomposition technique [7], it decomposes an input matrix $A^{n \times m}$ into the product of two other matrices $B^{n \times m}$ and $V^{m \times m}$. Formally a matrix

$A$ can be decomposed as follows:

$$A = BV^T$$

,where $B$ called as **loading matrix** and $V$ called as **factor matrix**.

### 3.2.3 Factor Matrix $V$

Obtained matrix $V$ is an orthogonal matrix (Property 1 of Appendix) with dimension $m$. We call the column vectors of $V$ as **centroid factors**.

$$V = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_j & \cdots & \mathbf{v}_m \end{bmatrix}$$

We call $\mathbf{v}_j$ as the $j$th centroid factor.

$$V = \begin{bmatrix} v_{11} & \cdots & v_{1j} & \cdots & v_{1m} \\ \vdots & \ddots & \vdots & \cdots & \vdots \\ v_{i1} & \cdots & v_{ij} & \cdots & v_{im} \\ \vdots & \cdots & \vdots & \ddots & \vdots \\ v_{m1} & \cdots & v_{mj} & \cdots & v_{mm} \end{bmatrix}$$

The following constraints are valid for the elements of $V$.

**Constraint 1.**

$$v_{1j}^2 + \cdots + v_{ij}^2 + \cdots + v_{mj}^2 = 1$$

**Constraint 2.**

$$v_{i1} + \cdots + v_{ij} + \cdots + v_{im} < m$$

### 3.2.4 Loading Matrix $B$

Obtained matrix $B$ has exactly the same dimension as $A$: $n \times m$. We call the n-dimensional column vectors of $B$ as **loading vectors**.

$$B = \begin{bmatrix} \mathbf{b}_1 & \cdots & \mathbf{b}_j & \cdots & \mathbf{b}_m \end{bmatrix}$$

Written in matrix form:

$$B = \begin{bmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots & \cdots & \vdots \\ b_{i1} & \cdots & b_{ij} & \cdots & b_{im} \\ \vdots & \cdots & \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nj} & \cdots & b_{nm} \end{bmatrix}$$

We call the elements of the loading matrix as **loadings**.

Let us call the sum of the square of the elements in the $j$th loading vector $\mathbf{b}_j$ as the **significance** of the $j$th centroid factor, and denote it with $\psi_j$.

$$\psi_i = b_{1j}^2 + \cdots + b_{ij}^2 + \cdots + b_{nj}^2 \tag{3.1}$$

As the outcome of the decomposition the following equation is valid:

$$A = \mathbf{b}_1\mathbf{v}_1^T + \mathbf{b}_2\mathbf{v}_2^T + \cdots + \mathbf{b}_m\mathbf{v}_m^T \tag{3.2}$$

### 3.2.5 Extraction Procedure of the Centroid Factors

Centroid decomposition aims to find factors with *high siginifance*. It uses a greedy method to iteratively define candidate factors. This method is called the centroid method, therefore the factors which are produced by this algorithm are called centroid factors. In order to have a better understanding on the meaning of the centroid factors, we give a geometric interpretation of the problem. Each row of the input matrix $A_1$ denotes a point in the $m$-dimensional space, which means our measurement entities are represented by points in $\mathbb{R}^m$ space. The basic idea of finding a suitable factor is that we take the arithmetic mean (centroid) of all the points, and use it to calculate the first centroid factor. We call the centroid point $\mathbf{c}_1$ the first *centroid* of $A_1$ and calculate as the following:

$$\mathbf{c}_1 = \frac{A_1^T\mathbf{1}_n}{n}$$

The first *centroid factor* is defined as the normalized vector (Property 2 of Appendix).

$$\mathbf{v}_1 = \frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} = \frac{A_1^T\mathbf{1}_n}{\|A_1^T\mathbf{1}_n\|}$$

Where $||c_1||$ is the norm (Property 3 of Appendix) of vector $c_1$. We are searching for factors with high significance, and from the above equation of significance (3.1) follows, that a factor has high significance if the absolute value of the loadings belonging to that factor is high. The centroid factor calculated that way, would be a significance factor under special circumstances:

1. if all the variables (represented by points) residue near the line determined by the centroid factor
2. and the centroid $\mathbf{c}_1$'s distance from the origin (or in other means, its vector's length) is large.

The larger the centroid vector's length $\|\mathbf{c}_1\|$ is, the more variable are dispersed around it and the more loading the centroid factor $\mathbf{v}_1$ has on it, so the higher it is its significance.

After determining the centroid factor candidate, it is then tweaked with respect to the above remarks. We introduce the vector $\mathbf{z}$ in the candidate factor's tweaked formula, and call it *sign vector*.

**Definition 1** *The **sign vector** denoted with **z** is an n-dimensional vector containing only -1 and 1 values in its coordinates.*

This vector will modify the signs of the rows in $A_1$. The remarks above make us sure, that changing these signs doesn't modify a factor's significance property, because only the signs of the loadings will change, not their absolute value. So with the proper choice of $\mathbf{z}$ we can find the best possible centroid, and centroid factor belonging to $A_1$. The algorithm of finding the proper sign vector is described in details in Chapter 4.

After modifying the formula of the first centroid it will be like the following:

$$\mathbf{c}_1 = \frac{A_1^T \mathbf{z}}{n} \tag{3.3}$$

The modified formula of the first centroid factor:

$$\mathbf{v}_1 = \frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} = \frac{A_1^T \mathbf{z}}{\|A_1^T \mathbf{z}\|} \tag{3.4}$$

After we have discovered the first centroid factor (which will be $\mathbf{v}_1$, the first column of factor matrix $V$), we then calculate the first loading vector (which will be $\mathbf{b}_1$, the first column of the loading matrix $B$) respect to it as the follows:

$$\mathbf{b}_1 = A_1 \mathbf{v}_1 \tag{3.5}$$

Now we have the first centroid factor and the loadings respective to it. Now our job is to subtract the "information" along the new factor $\mathbf{v}_1$ from $A_1$, in other words, to eliminate its impact from the data matrix, in order to be able to find the second centroid factor, and so on. This step is equivalent with a *rank reduction* on matrix $A_1$.

$$A_2 := A_1 - A_1 \mathbf{v}_1 \mathbf{v}_1^T \tag{3.6}$$

As we have $A_2$ we can start to search for the second centroid factor. Iteratively repeating the whole procedure will extract the centroid factors and loading vectors respective to each from the original matrix $A_1$. Each factor extraction step will reduce the rank of $A_1$ by one, so the procedure will stop in $r$ step, where $r$ denotes the rank of $A_1$.

## 3.3 Algorithm

The centroid decomposition is an iterative process, in which each step finds a centroid factor and the corresponding loading vector.
The algorithm of centroid factor retrieval is described as follows:

**Algorithm 1**: Centroid Decomposition

**Input**: $A^{n \times m}$

**Output**: loading matrix $B$ and factor matrix $V$

1 $i = 1$ ;
2 $A_i := A$ ;
3 **repeat**
4 $\quad \mathbf{z} = FindSignVector(A_i)$ ;
5 $\quad \mathbf{c}_i = A_i^T \mathbf{z}$ ;
6 $\quad \mathbf{v}_i = \frac{\mathbf{c}_i}{\|\mathbf{c}_i\|}$;
7 $\quad \mathbf{b}_i = A_i \mathbf{v}_i$ ;
8 $\quad A_i := A_i - \mathbf{b}_i \mathbf{v}_i^T$ ;
9 $\quad$ **if** *i = 0* **then**
10 $\quad\quad B = \mathbf{b}_i$, $V = \mathbf{v}_i$ ;
11 $\quad$ **else**
12 $\quad\quad B = \text{append}(\text{B}, \mathbf{b}_i)$ `// append` $\mathtt{b}_i$ `to the right side of` $B$;
13 $\quad\quad V = \text{append}(\text{V}, \mathbf{v}_i)$ `// append` $\mathtt{v}_i$ `to the right side of` $V$;
14 $\quad i = i + 1$;
15 $\quad m = m - 1$;
16 **until** $m = 0$ ;

Key question is the implementation of the function in the 4. line, we called $FindSignVector(A_i)$. In the next chapter we review the algorithm of finding a sign vector, and show our different implementations and their's computational complexity.

# 4 Sign Vectors Computation

## 4.1 Definitions

For simplicity, we use $A$ to denote the input matrix obtained out of the extraction of any factor, i.e, $A_i$. Choosing the proper sign vector means, finding the sign vector that generates the largest sized centroid (3.3), i.e., that has the maximal $\|c\|$, in each step. Recall the definition of $\mathbf{c}$ and write down the formula of $\|c\|$:

$$\|\mathbf{c}\| = \frac{\|A^T\mathbf{z}\|}{n}$$

The following formula gives the scalar multiple of the size of the centroid:

$$n^2\|\mathbf{c}\|^2 = \|A^T\mathbf{z}\|^2 \tag{4.1}$$

This type of formula is difficult to handle by an optimization. For an exhaustive search would take $2^n$ steps to find the proper sign vector. That's why we are looking for an equivalent formula, which we can maximize in less steps. Let $R$ denote the product of $A$ and it's transpose: $R = AA^T$. Then we can write the following equation:

$$n^2\|\mathbf{c}\|^2 = \|A^T\mathbf{z}\|^2 = \mathbf{z}^T R\mathbf{z} \tag{4.2}$$

Maximization of the formula $\mathbf{z}^T R\mathbf{z}$ is an integer programming problem. [5] To show the proof of the equation (4.2), let's initialize sign vector $\mathbf{z}$ with $\mathbf{1}_n$, i.e., the vector of 1 values. Then, we have the following:

$$\|A^T\mathbf{z}\|^2 = \mathbf{z}^T R\mathbf{z} \tag{4.3}$$

,what is the scalar multiple (4.1) of $\|\mathbf{c}\|$. This means that finding the largest size centroid is equivalent with the following maximization problem:

$$\mathbf{z} = \underset{|\mathbf{z}_j|=1}{\arg\max}\, \mathbf{z}_j^T R\mathbf{z}_j \tag{4.4}$$

Where index $j$ denotes the number of iteration during the maximization process. It is seen, from the equation (**??**) , that the value of elements in the diagonal of $R$ are independent from the choice of $\mathbf{z}$. Let's consider matrix $P$ has all values of $R$ except that the diagonal values set to 0. Thus, we can write the following:

$$\mathbf{z}_j^T R \mathbf{z}_j = \mathbf{z}_j^T P \mathbf{z}_j + \sum_{i=1}^{n} r_{ii} = \mathbf{z}_j^T P \mathbf{z}_j + k \tag{4.5}$$

, where $k$ denotes a constant value.

It follows that the maximization problem of equation 4.4 is equivalent to the following:

$$\arg\max_{|\mathbf{z}_j|=1} \mathbf{z}_j^T P \mathbf{z}_j \tag{4.6}$$

In what follows, we explain why we use $P$ instead of $R$. In the first iteration of the algorithm we initialize $\mathbf{z} = \mathbf{1}_n$. Our initial $\mathbf{w}_1$ is equal to:

$$\mathbf{w}_1 = P\mathbf{z}_1 = \begin{bmatrix} w_1^{(1)} \\ w_1^{(2)} \\ \vdots \\ w_1^{(n)} \\ \end{bmatrix}$$

If we use $R$ instead of $P$ our initial $\mathbf{w'}_1$ can be written as follows:

$$\mathbf{w'}_1 = R\mathbf{z}_1 = \begin{bmatrix} w_1^{(1)} + r_{11} \\ w_1^{(2)} + r_{22} \\ \vdots \\ w_1^{(n)} + r_{nn} \end{bmatrix}$$

where $r_{ii}$ denotes the $i$th element in the diagonal of $R$.

In any $j$ iteration our algorithm is looking for the index $k$ where:

$$k = \arg\max_{\{i \mid w_j^{(i)} \cdot z_j^{(i)} < 0\}} |w_j^{(i)}| \tag{4.7}$$

The next sign vector $\mathbf{z}_{j+1}$ in the iterative process will be created by changing the sign of it's $k$th component, i.e., $z_{j+1}^{(k)} := -z_j^{(k)}$.

The problem with using $R$ instead of $P$ is that our algorithm's decision on $k$ would be different than compared to the one based on $P$. The reason is that the $k$th component in $\mathbf{w'}_j$ is $w'^{(k)}_j = (w_j^{(k)} + r_{kk})$, so the decision of $k$ depends not only on $w'^{(k)}_j$ but on $r_{kk}$ as well, which can mislead the choice. The following table summarizes the cases of decisions on $k$ as the result of the misleading formula: $\arg\max_{\{i \mid w'^{(i)}_j \cdot z_j^{(i)} < 0\}} |w'^{(i)}_j|$. We show if the decision would differ from the correct result based on (4.7).

| | | |
|---|---|---|
| 1st case: | $(w_j^{(k)} + r_{kk}) < 0$ and $\lvert w_j^{(k)} + r_{kk}\rvert > \lvert w_j^{(i)} + r_{ii}\rvert$, $\forall i \{1 \le i \le n, i! = k\}$ | *no* |
| 2nd case: | $(w_j^{(k)} + r_{kk}) < 0$, but $\exists i$, that $\lvert w_j^{(i)} + r_{ii}\rvert > \lvert w_j^{(k)} + r_{kk}\rvert$ | *yes* |
| 3rd case: | $(w_j^{(k)} + r_{kk}) > 0$ | *yes* |

Now we have proved that the decision of $k$ will be different in the 2nd and 3rd cases, when $R$ is used as input.

The algorithm finds a maximum of $\mathbf{z}^T P\mathbf{z}$. It can occur that there are more optimal sign vectors for $P$. The reason is, more than one choice of $k$ can satisfy (4.7). The algorithm can get to every optimal sign vector, if by running, it decides on a random possible $k$.

An important feature of the algorithm is, that it never descends, i.e. it never chooses the same $k$ in different iterations, or in other words, it never changes the sign of an element which has been already changed.

Let us assume there are $r$ optimal sign vectors for a given $A$ input matrix, and they are denoted with $\mathbf{z}_1^*, \mathbf{z}_2^*$ ... $\mathbf{z}_r^*$.

If the algorithm changes the sign of an element $z_j^{(k)}$ for negative, while that element is non-negative in any of the optimal sign vectors $\mathbf{z}_i^*$, then from that step on, the process converges towards a suboptimal $\mathbf{z}$, because that sign will never be changed again during the run. Thus it is proved that if $R$ is used as the input instead of $P$, the decision on $k$ in the $j$th step will be different in the 2nd or 3rd cases, and this leads to a suboptimal result.

## 4.2 Algorithm

### 4.2.1 Steps

We present in this section the steps of the maximization process that computes the right sign vectors.

**Step 1** Choose an initial sign vector $\mathbf{z}_1 = \mathbf{1}_n$ and define the vector $\mathbf{w}_1$ as follows: $\mathbf{w}_1 = P\mathbf{z}_1$

**Step 2** Choose the $k$th index of the component in $\mathbf{w}_j$, where $k = \arg\max_{\{i \mid w_j^{(i)} \cdot z_j^{(i)} < 0\}} |w_j^{(i)}|$.

**Step 3** Take $\mathbf{z}_1$ and change the sign of it's $k$th component to create $\mathbf{z}_2$.

**Step 4** Repeat the whole procedure from **Step 1** with using the newly created $\mathbf{z}_2$, until the signs of all corresponding components in $\mathbf{w}_j$ and $\mathbf{z}_j$ will be the same.

**Step 5** Choose $\mathbf{z}_j$ as the proper sign vector $\mathbf{z}$ for $A$.

### 4.2.2 Pseudocode

The pseudocode of these steps is described in Algorithm 2.

**Algorithm 2**: $FindSignVector(A)$

**Data**: $A$
**Result**: the corresponding $\mathbf{z}$ sign vector for $A$

```
1  j = 1 ;
2  k = 1;
3  maxValue = 0, bestIndex = 0;
4  z_1^T = [1  1  ...  1]^n;
5  compute P := setDiagNull(AA^T)  // set 0 the diagonal elements in AA^T;
6  compute w_1 = P z_1 ;
7  repeat
8      if j > 1 then
9          w_j = w_{j-1} - 2 · col(P, bestIndex)   // col() returns the column with
           index bestIndex from P;
10     maxValue = 0;
11     bestIndex = 0;
12     for k = 1 to n do
13         if (w_j^(k) · z_j^(k) < 0) then        // w_j^(k) means the kth component in w_j
14             if maxValue <|w_j^(k)| then
15                 maxValue = w_j^(k) ;
16                 bestIndex = k ;
17             else
18                 k = k + 1 ;
19
20     if maxValue != 0 then
21         z_{j+1} := changeSign(bestIndex, z_j)  // change the sign of element with
           index "bestIndex" in z_j ;
22         j = j + 1;
23
24 until maxValue = 0 ;
25 return z_j
```

The search space of sign vectors if $\Omega = \{2^n\}$ since the possible values of $\mathbf{z}$ are 1 and -1. In the worst case the value of $\mathbf{z}$ will be changed $n$ times and the algorithm will take linear time to compute the correct values of $\mathbf{z}$. This explained by the fact that we start by the initial sign vector $\mathbf{1}_n$ and in each step we change the sign of one element. Changed elements will not be updated again. The experiment of Figure 4.1 shows that in an average case the algorithm takes $\frac{n}{2}$ steps to find the sign vector. The values of the x-axis and the y-axis are a power of $k = 10^3$.

## 4.3 Computation of z

### 4.3.1 Using $2 - d$ Array to store P

We propose to use a 2-dimensional array to store matrix $P$. We have following run time and space complexities:

**Space complexity**: The matrix $P$ is calculated as $AA^T$, with all of the elements in the diagonal set to 0. This implementation uses 2-dimensional arrays to represents matrices. Hence $P$ is an $n \times n$ square matrix (n equals to the number of rows in $A$), the space complexity increases **quadratically** with the number of rows of the input matrix

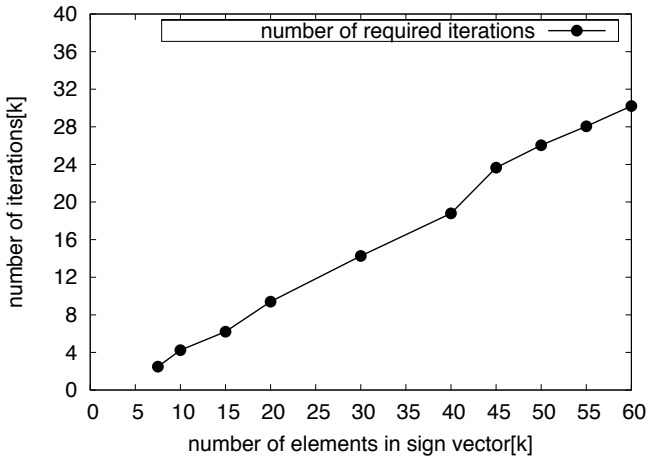



Figure 4.1: Number of iterations to find sing vector

$A$.

| Stored object | Space complexity |
|---|---|
| $A$ | $n \cdot m$ |
| $\mathbf{z}$ | $n$ |
| $P$ | $n^2$ |
| $\mathbf{w}$ | n |
| **Total space complexity:** | $n^2 + n \cdot (m + 2)$ |

Table 4.1: Space complexity of $1 - d$ array

If the elements are stored in floats, the used memory space is: $(n^2+n\cdot(m+2))\cdot 4$ *byte*. In Table 4.2, we give an example how the space increases for input matrices with increasing number of rows and fixed number of columns.

| $n \times m$ | Space complexity |
|---|---|
| $10000 \times 5$ | $400,28$ MB |
| $20000 \times 5$ | $\sim 2,8$ GB |
| $30000 \times 5$ | $\sim 3,6$ GB |
| $40000 \times 5$ | $\sim 6,4$ GB |
| $\vdots$ | $\vdots$ |

Table 4.2: Memory used for $2 - d$ array

**Run time complexity**:
**Line 4** - initialize $\mathbf{z}_j$ : $n$ steps
**Line 5** - computeP$(A_i)$ : $\frac{n^2 m}{2} + n$ steps
**Line 6** - compute $\mathbf{w}_1 = P\mathbf{z}_1$ : $n^2 steps$

**Loop 7-24:** $x$-times: in worst case $x = n$, average: $x = \frac{n}{2}$
**Line 9** - compute $\mathbf{w}_j$ : $n$ steps
**Loop 12-19** : $6n$ steps

**Total complexity:** $n + \frac{n^2m}{2} + n + n^2 + x \cdot (n + 6n) = \frac{n^2m}{2} + n^2 + 2n + x \cdot 7n$ steps
**Total worst case complexity:** $\frac{n^2m}{2} + 8n^2 + 2n$ steps $= O(\frac{n^2m}{2} + n^2 + n)$
**Average complexity:** $\frac{n^2m}{2} + 4.5n^2 + 2n$ steps $= O(\frac{n^2m}{2} + n^2 + n)$

The high memory complexity inspired us to use another data structure to store $P$.

### 4.3.2 Using $1 - d$ Array to store P

This implementation follows the same pseudocode, but aims to reduce the memory complexity of the previous implementation.

**Space complexity**: This implementation exploits the symmetric property of P and that the diagonal contains only 0 values. We store only the necessary part of $P$ in a 1-dimensional array. It means, that instead of $n^2$ elements, only $\frac{n^2}{2} - n$ elements have to be stored. In order, to identify the element of $P$ in the 1-dimensional array, we give an index to the elements, what is calculated from their row index and column index in the matrix form of $P$. However, the computation of $\mathbf{w}_j = P\mathbf{z}_j$ needs more time, because of the index calculation used to access $P$.

| Objects need to be stored | Space complexity |
|---|---|
| $A$ | $n \cdot m$ |
| $\mathbf{z}$ | $n$ |
| $P$ | $\frac{n^2}{2} - n$ |
| $\mathbf{w}$ | n |
| **Total space complexity:** | $\frac{n^2}{2} + n \cdot (m + 1)$ |

Table 4.3: Space complexity of $1 - d$ array

If the elements are stored in floats, the space needed is the following: $(\frac{n^2}{2} + n \cdot (m + 1)) \cdot 4$ *byte*. Table 4.4 summarizes the space complexity for given input matrices with increasing number of rows and fixed number of columns.

| $n \times m$ | Space complexity |
|---|---|
| $10000 \times 5$ | $200,24$ MB |
| $20000 \times 5$ | $\sim 0,8$ GB |
| $30000 \times 5$ | $\sim 1,8$ GB |
| $40000 \times 5$ | $\sim 3,2$ GB |
| $\vdots$ | $\vdots$ |

Table 4.4: Memory Used for $1 - d$ array

**Run time complexity**:
**Line 4** - initialize $\mathbf{z}_j$ : $n$ steps
**Line 5** - computeP($A_i$) : $\frac{(n^2-n)m}{2}$ steps
**Line 6** - compute $\mathbf{w}_1 = P\mathbf{z}_1$ : $3n^2 steps$
**Loop 7-24:** $x$-times: in worst case $x = n$, average: $x = \frac{n}{2}$
**Line 9** - compute $\mathbf{w}_j$ : $3n$ steps
**Loop 12-19** : $6n$ steps

**Total complexity:** $n + \frac{(n^2-n)m}{2} + 3n^2 + x \cdot (3n + 6n) = n + \frac{n^2m}{2} - \frac{nm}{2} + 3n^2 + x \cdot 9n$ steps.
**Total worst case complexity:** $\frac{n^2m}{2} + 12n^2 - \frac{nm}{2} + n$ steps $= O(\frac{n^2m}{2} + n^2 - \frac{nm}{2} + n)$
**Total average case complexity:** $\frac{n^2m}{2} + 7.5n^2 - \frac{nm}{2} + n$ steps $= O(\frac{n^2m}{2} + n^2 - \frac{nm}{2} + n)$

The $1-d$ data structure reduced the space complexity, but the memory needed is still high. Our goal to reduce the space complexity to make it linear. We propose to change the algorithm 2 in order not to store anymore matrix $P$.

### 4.3.3 Without storing $P$

The idea of this implementation is, that we don't really need to store $P$ at all, because all information contained in $P$ can be calculated on the fly from original matrix A. In the previous implementations, we were using $P$ to be able to compute the vector $\mathbf{w}$ in the subalgorithm $FindSignVector()$. It means, that we used $P$ in the following two steps:

- To compute the initial $\mathbf{w}_1 = P\mathbf{z}_1$, where $\mathbf{z}_1$ is the initial sign vector containing only 1 values.
- To compute the following $\mathbf{w}_j = \mathbf{w}_{j-1} - 2 \cdot col(P, k)$.

Matrix $P$ is computed out of $R$ by setting its diagonal elements to 0. The $\mathbf{r}_{ij}$ element in $R$ is calculated exactly with multiplying the $i$th row-vector of $A$ , i.e., $\mathbf{a}_i$), with the transpose of the $j$th row vector of $A$, i.e., $\mathbf{a}_j^T$). Therefore, we can write $P$ as the following:

$$P = \begin{bmatrix} 0 & \mathbf{a}_1 \cdot \mathbf{a}_2^T & \cdots & \mathbf{a}_1 \cdot \mathbf{a}_i^T & \cdots & \mathbf{a}_1 \cdot \mathbf{a}_n^T \\ \mathbf{a}_2 \cdot \mathbf{a}_1^T & 0 & \cdots & \mathbf{a}_2 \cdot \mathbf{a}_i^T & \cdots & \mathbf{a}_2 \cdot \mathbf{a}_n^T \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ \mathbf{a}_i \cdot \mathbf{a}_1^T & \mathbf{a}_i \cdot \mathbf{a}_2^T & \cdots & 0 & \cdots & \mathbf{a}_i \cdot \mathbf{a}_n^T \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ \mathbf{a}_n \cdot \mathbf{a}_1^T & \mathbf{a}_n \cdot \mathbf{a}_2^T & \cdots & \mathbf{a}_n \cdot \mathbf{a}_i^T & \cdots & 0 \end{bmatrix}$$

### Computation of $\mathbf{w}_1$

The initial $\mathbf{w}_1 = P\mathbf{z}_1$, where $\mathbf{z}_1^T = \begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix}^n$. This means that the components of $\mathbf{w}_1$ will be the sum of the components in the corresponding row of $P$:

$$\mathbf{w}_1 = \begin{bmatrix} 0 + \mathbf{a}_1 \cdot \mathbf{a}_2^T + \cdots + \mathbf{a}_1 \cdot \mathbf{a}_i^T + \cdots + \mathbf{a}_1 \cdot \mathbf{a}_n^T \\ \mathbf{a}_2 \cdot \mathbf{a}_1^T + 0 + \cdots + \mathbf{a}_2 \cdot \mathbf{a}_i^T + \cdots + \mathbf{a}_2 \cdot \mathbf{a}_n^T \\ \vdots \\ \mathbf{a}_i \cdot \mathbf{a}_1^T + \mathbf{a}_i \cdot \mathbf{a}_2^T + \cdots + 0 + \cdots + \mathbf{a}_i \cdot \mathbf{a}_n^T \\ \vdots \\ \mathbf{a}_n \cdot \mathbf{a}_1^T + \mathbf{a}_n \cdot \mathbf{a}_2^T + \cdots + \mathbf{a}_n \cdot \mathbf{a}_i^T + \cdots + 0 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \cdot (0 + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) \\ \mathbf{a}_2 \cdot (\mathbf{a}_1^T + 0 + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) \\ \vdots \\ \mathbf{a}_i \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + 0 + \cdots + \mathbf{a}_n^T) \\ \vdots \\ \mathbf{a}_n \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + 0) \end{bmatrix}$$

We can transform the formula as the following:

$$\mathbf{w}_1 = \begin{bmatrix} \mathbf{a}_1 \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) - \mathbf{a}_1 \cdot \mathbf{a}_1^T \\ \mathbf{a}_2 \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) - \mathbf{a}_2 \cdot \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_i \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) - \mathbf{a}_i \cdot \mathbf{a}_i^T \\ \vdots \\ \mathbf{a}_n \cdot (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T) - \mathbf{a}_n \cdot \mathbf{a}_n^T \end{bmatrix}$$

Let's take $\mathbf{s} = (\mathbf{a}_1^T + \mathbf{a}_2^T + \cdots + \mathbf{a}_i^T + \cdots + \mathbf{a}_n^T)$ is a constant vector. $\mathbf{s}$ has to be calculated only once instead of $n$ times. The computational complexity of calculating $\mathbf{s}$ is $O(nm)$. Rewriting the formula with using $\mathbf{s}$ gives the following:

$$\mathbf{w}_1 = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{s} - \mathbf{a}_1 \cdot \mathbf{a}_1^T \\ \mathbf{a}_2 \cdot \mathbf{s} - \mathbf{a}_2 \cdot \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_i \cdot \mathbf{s} - \mathbf{a}_i \cdot \mathbf{a}_i^T \\ \vdots \\ \mathbf{a}_n \cdot \mathbf{s} - \mathbf{a}_n \cdot \mathbf{a}_n^T \end{bmatrix} \tag{4.8}$$

The computational complexity of calculating $\mathbf{w}_1$ is $O(2nm)$. After computing the initial $\mathbf{w}_1$, we show how to compute the rest of $\mathbf{w}_{j\{1<j\leq n\}}$ values.

### Computation of $\mathbf{w}_j$

As described above, $\mathbf{w}_j = \mathbf{w}_{j-1} - 2 \cdot col(P, k)$, so we have to find the formula which gives us the $k$th column of $P$ directly from the input matrix $A$. Using the equation (4.3.3),

we can write the $k$th column of $P$ as follows:

$$\begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{a}_k \\ \mathbf{a}_2 \cdot \mathbf{a}_k \\ \vdots \\ \mathbf{a}_i \cdot \mathbf{a}_k \\ \vdots \\ \mathbf{a}_n \cdot \mathbf{a}_k \end{bmatrix}$$

Having $\mathbf{w}_j = \mathbf{w}_{j-1} - 2 \cdot col(P, k)$, it follows:

$$\mathbf{w}_j = \mathbf{w}_{j-1} - 2 \cdot \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{a}_k^T \\ \mathbf{a}_2 \cdot \mathbf{a}_k^T \\ \vdots \\ \mathbf{a}_i \cdot \mathbf{a}_k^T \\ \vdots \\ \mathbf{a}_n \cdot \mathbf{a}_k^T \end{bmatrix} = \begin{bmatrix} \mathbf{w}^1{}_{j-1} - 2 \cdot \mathbf{a}_1 \cdot \mathbf{a}_k^T \\ \mathbf{w}^2{}_{j-1} - 2 \cdot \mathbf{a}_2 \cdot \mathbf{a}_k^T \\ \vdots \\ \mathbf{w}^i{}_{j-1} - 2 \cdot \mathbf{a}_i \cdot \mathbf{a}_k^T \\ \vdots \\ \mathbf{w}^n{}_{j-1} - 2 \cdot \mathbf{a}_n \cdot \mathbf{a}_k^T \end{bmatrix} \tag{4.9}$$

The computational complexity of calculating $\mathbf{w}_j$ using the formula (4.9) is $O(nm)$.

### Pseudocode

**Space complexity**: Because we don't compute $P$ in this implementation, only the following objects are stored:

| Objects need to be stored | Space complexity |
|---|---|
| $A$ | $n \cdot m$ |
| $\mathbf{s}$ | $m$ |
| $\mathbf{z}$ | $n$ |
| $\mathbf{w}$ | n |
| **Total space complexity:** | $n \cdot (m + 2) + m$ |

Table 4.5: Space complexity of the final implementation

If the elements are stored in floats, the needed memory space is $(n \cdot (m+2)+m) \cdot 4$ *byte*. Thus, the memory used by this implementation increases **linearly** with size of the input matrix.

Table 4.6: Memory need

| $n \times m$ | Space complexity |
|---|---|
| $10000 \times 5$ | $\sim 0,28$ MB |
| $20000 \times 5$ | $\sim 0,56$ MB |
| $30000 \times 5$ | $\sim 0,84$ MB |
| $40000 \times 5$ | $\sim 1,12$ MB |
| $\vdots$ | $\vdots$ |
| $100000 \times 5$ | $\sim 2,8$ MB |
| $\vdots$ | $\vdots$ |

**Computational complexity**:
**Line 4** - initialize $\mathbf{z}_j$ : $n$ steps $= O(n)$
**Line 5** - compute $\mathbf{s}$ : $nm$ steps $= O(nm)$
**Line 6** - compute $\mathbf{w}_1$ : $nm = O(nm)$
**Loop 7-24:** $x$-times: worst case $x = n$, average $x = \frac{n}{2}$
**Line 9, 10** - compute $\mathbf{w}_j$ : $nm$ steps $= O(nm)$
**Loop 12-19** : $6n$ steps

**Total complexity:** $n + 2nm + x \cdot (nm + 6n)$ steps
**Total worst case complexity:** $n+2nm+n \cdot (nm+6n)$ steps $= n^2m+6n^2+2nm+n=$ $O(n^2m + n^2 + nm + n)$ steps
**Average complexity:** $n + 2nm + \frac{n}{2} \cdot (nm + 6n)$ steps $= \frac{n^2m}{2} + 3n^2 + 2nm + n=$ $O(\frac{n^2m}{2} + n^2 + nm + n)$ steps

## 4.4 Comparison of Different Data Structures

Figure 4.2 shows a graphical computational complexity comparison of the three implementations where x-axis represents the number of rows of the input matrix and y-axis represents the measured computational time in milliseconds. The values of the two axes are a power of $k = 10^3$. The first and second implementations could not finish their computation with input matrices containing respectively more rows than 10000 and 20000 and generated an out of memory error. We use four time series for to compute the run time of the decomposition ($m = 4$).

Using the final implementation, the total complexity of centroid decomposition algorithm of an input matrix $A^{n \times m}$ is the follows :
**Loop 4-19:** $l - times$
**Line 6** - $B = \mathbf{b}_i$ : $m$ steps $O(m)$
**Line 7** - $V = \mathbf{v}_i$ : $n$ steps $O(n)$
**Line 9** - append$(B, \mathbf{b}_i)$ : $(mn + m)$ $steps O(mn + m)$
**Line 10** - append$(V, \mathbf{v}_i)$ : $(nm + n)$ steps$=O(nm + n)$
**Line 13** - findSignVector$(A_i)$ - in average case $\frac{n^2m}{2} + 3n^2 + 2nm + n$

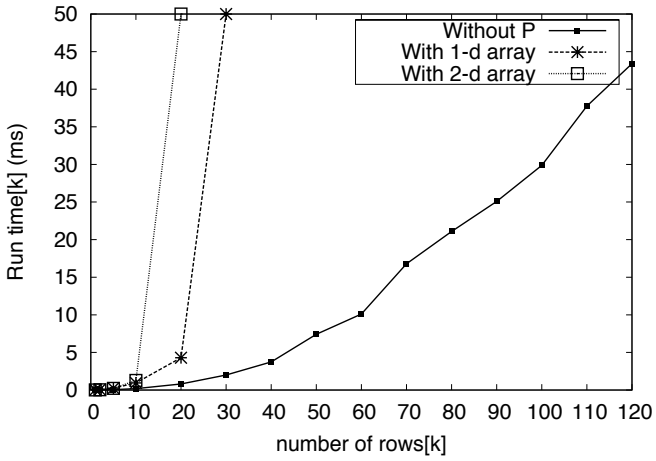



Figure 4.2: Computational complexity of different implementations

- in worst case $n^2m + 6n^2 + 2nm + n$
**Line 14** - compute $\mathbf{c}_i$ : $nm$ steps=$O(nm)$
**Line 15** - compute $\mathbf{v}_i$ : $2m$ steps=$O(m)$
**Line 16** - compute $\mathbf{b}_i$ : $nm$ steps=$O(nm)$
**Line 17** - compute $A_{i-1}$ : $nm$ steps=$O(nm)$

**Total in average case:** $m + n + (l-1) \cdot (2nm + m + n) + l \cdot (\frac{n^2m}{2} + 3n^2 + 2nm + n) = l \cdot O(\frac{n^2m}{2} + n^2 + nm + n + m)$

## 4.5 Running Example

Let's take the example of matrix $A = \begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$, we illustrate the computation of the Centroid Decomposition of $A$. The number of columns $m = 2$, that's why 2 centroid factors will be extracted. In what follows we assume that $A_1 := A$.

**I. Extraction of the first centroid factor**

First, we look for a corresponding sign vector for $A_1$ with using the above described $FindSingVector(A)$ algorithm. The computation of $R_1$ gives the following:

$$R_1 = A_1 A_1^T = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Matrix $P_1$ is the same as $R_1$ with 0 on its diagonal: $P_1 = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

**Step 1** Choose an initial sign vector $\mathbf{z}_1$ and compute the vector $\mathbf{w}_1$ as follows: $\mathbf{w}_1 = P\mathbf{z}_1$

*First iteration*: $\mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $\mathbf{w}_1 = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix}$

*Second iteration*: $\mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$ $\mathbf{w}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$

**Step 2** Choose the $k$th index of the component in $\mathbf{w}_j$, where $k = \arg\max_{\{i \mid w_j^{(i)} \cdot z_j^{(i)} < 0\}} |w_j^{(i)}|$.
*First iteration*: $k = 2$
*Second iteration*: We don't have elements with different signs. Jump to **Step 5**

**Step 3** Take $\mathbf{z}_1$ and change the sign of it's $k$th component to create $\mathbf{z}_2$.

*First iteration*: $\mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$

**Step 4** Repeat the whole procedure from **Step 1** with using the newly created $\mathbf{z}_2$, until the signs of all corresponding components in $\mathbf{w}_j$ and $\mathbf{z}_j$ will be the same.

**Step 5** Choose $\mathbf{z}_j$ as the proper sign vector $\mathbf{z}$ for $A$.

*Second iteration*: The proper sign vector is found: $\mathbf{z}_2 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$

As we found the sign vector $\mathbf{z}$ we can calculate the first centroid factor $\mathbf{v}_1$ (which will be the first column of factor matrix $V$). Using the formula (3.4), we have the following:

$$\mathbf{v}_1 = \frac{A_1^T \mathbf{z}}{\|A_1^T \mathbf{z}\|} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix}$$

Since we obtained the first centroid factor, we can calculate the first loading vector $\mathbf{b}_1$ (which will be the first column of the loading matrix $B$). Using the equation (3.5), we have the following:

$$\mathbf{b}_1 = A_1 \mathbf{v}_1 = \begin{bmatrix} \sqrt{2} \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

We have the first centroid factor and the loadings respective to it. Now we subtract the "information" along the first centroid factor $\mathbf{v}_1$ from $A_1$, in order to be able to find the second best centroid factor. This step is equivalent with a rank reduction on matrix $A_1$. We are using the formula (3.6).

$$A_2 = A_1 - A_1 \mathbf{v}_1 \mathbf{v}_1^T = \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

**II. Extraction of the second centroid factor**

We compute the second centroid factor from $A_2$. Thus, we repeat all of the steps of the algorithm on $A_2$. We start by computing the proper sign vector with respect to $A_2$. The computation of $R_2$ gives the following:

$$R_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

We Construct $P_2$ out of $R_2$:

$$P_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{bmatrix}$$

**Step 1** *First iteration*: $\mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $\mathbf{w}_1 = \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}$

**Step 2** *First iteration*: We don't have elements with different signs. Jump to **Step 5**

**Step 5** *First iteration*: The proper sign vector is found: $\mathbf{z}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

Using the computed sign vector, we calculate the second centroid factor and loading and loading vector:

$$\mathbf{v}_2 = \frac{A_2^T \mathbf{z}}{\|A_2^T \mathbf{z}\|} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

and

$$\mathbf{b}_2 = A_2 \mathbf{v}_2 = \begin{bmatrix} 0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

The algorithm terminates and we obtain the Centroid Decomposition of the original matrix $A = BV^T$ as follows:

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}$$

We can also write the decomposition in the following form: $A = \mathbf{b}_1 \mathbf{v}_1^T + \mathbf{b}_2 \mathbf{v}_2^T$:

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix}$$

# 5 Missing Values Recovery

## 5.1 Method

The proposed recovery algorithm is based on an iterative application of the Centroid Decomposition method. In every iteration of the algorithm, a dimensionality reduction is applied: only $k < m$ factors are computed, where $m$ is the number of columns in matrix $A$. We compute the difference in every step between the Frobenius norm (Property 4 of Appendix) [7] of the currently obtained recovery matrix and the previous one. The algorithm iterates until this difference is smaller than a threshold value $\epsilon < 10^{-5}$. We consider that the reference time series contains the missing block that will be recovered and that test time series are used to recover the missing values in reference time series. Algorithm 3 describes the recovery process.

**Algorithm 3**: *Recovery_Process*

**Input**: $\mathcal{S}_{ref}$: reference time series with imputed missing values
**Input**: $\{\mathcal{S}_{test_1}, \mathcal{S}_{test_2} \dots \mathcal{S}_{test_n}\}$: set of test time series
**Input**: $f$: number of factors
**Input**: $R$: set of timestamps of missing values

**Output**: $M_i$: matrix of recovered values

$i = 1$ ;
$j = 1$;
$M = [\mathcal{S}_{ref}|\mathcal{S}_{test_1}|\mathcal{S}_{test_2} \dots \mathcal{S}_{test_n}]$;

**repeat**
  $B_i, V_i = CentroidDecomposition(M_i, \text{f})$;
  $\mathbf{b}_i = \text{col}(B_i, 1)$ // $col(B_i, 1)$ returns first column of $B_i$;
  $\mathbf{v}_i = \text{row}(V_i, 1)$ // $row(V_i, 1)$ returns first row of $V_i$;
  $\mathbf{x} = \mathbf{b}_i \cdot \mathbf{v}_i$ ;
  **foreach** $j \in R$ **do**
    $M_{i+1}[j, 1] = \mathbf{x}[j]$ ;
      // $M_{i+1}[j, 1]$ means the element in jth row,1st column of $M_{i+1}$;
      // x[$j$] means the element in jth row of vector x ;
  $i = i + 1$
**until** $|(\|\boldsymbol{M}_i - \boldsymbol{M}_{i-1}\|_F)| < \epsilon$ ;
**return** $M_i$

## 5.2 Recovery in Shifted Time Series

We construct synthetic shifted time series in order to check the accuracy of the proposed recovery technique. The constructed time series are periodic functions, e.g., sine, triangle, square, etc. We drop a block of 100 values from a reference time series and we recover it using our technique. We use the *Mean squared error (MSE)* to measure the recovery accuracy as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{M}_i - M_i)^2$$

Where $\hat{M}_i$ is the vector of recovered values and $M_i$ is the vector of the real values. We describe in what follows the result of a bunch of experiments we ran using different setups of synthetic time series. We enumerate the main factors that influence the accuracy of the recovery. We compute for each experiment the $MSE$ to measure the accuracy of the recovery. The plotted graphs have the following characteristics:

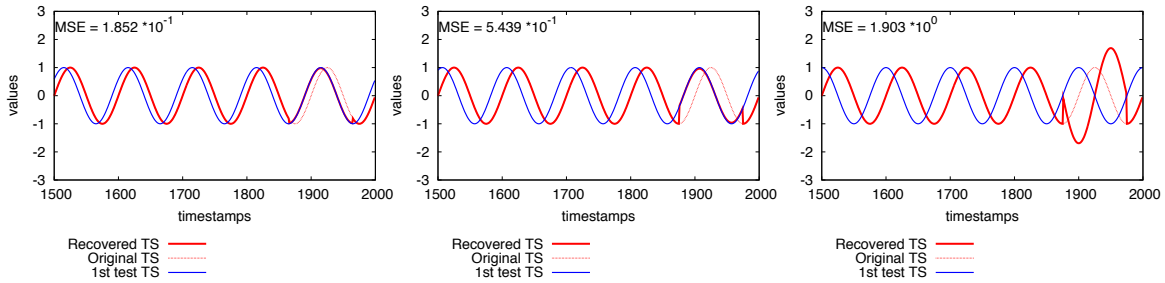
- The $x$-axis represents the timestamps and
- The $y$-axis the represents the observed values and the recovered ones
- The *thin black* line represents the original values of the reference time series.
- The *blue* line represents the test time series. If more test time series are used for the recovery, they are represented with *green* and *orange* lines.
- The *red bold* line represents the recovered values of the reference time series.

### 5.2.1 Impact of the length of time shift

In the experiment of Figure 5.1, we perform the recovery using a shifted test time series with respect to the reference time series. The two time series have the same shape, i.e., sine wave. We increase the time shift in the three results of Figure 5.1. In Figure 5.1(a), we show the result of recovery with a time shift equal to 10. In Figure 5.1(b), we show the result of recovery with a bigger time shift equal to 18 and in Figure 5.1(c) with a time shift equal to 25. The three figures show that the recovery imitates the shape of test time series where the missing block is. The smaller the length of time shift is, the more accurate the recovery is. Figure 5.1(c) shows also that if the length of the time shift is high, the amplitude of the original values is less accurately recovered.

### 5.2.2 Impact of the amplitude

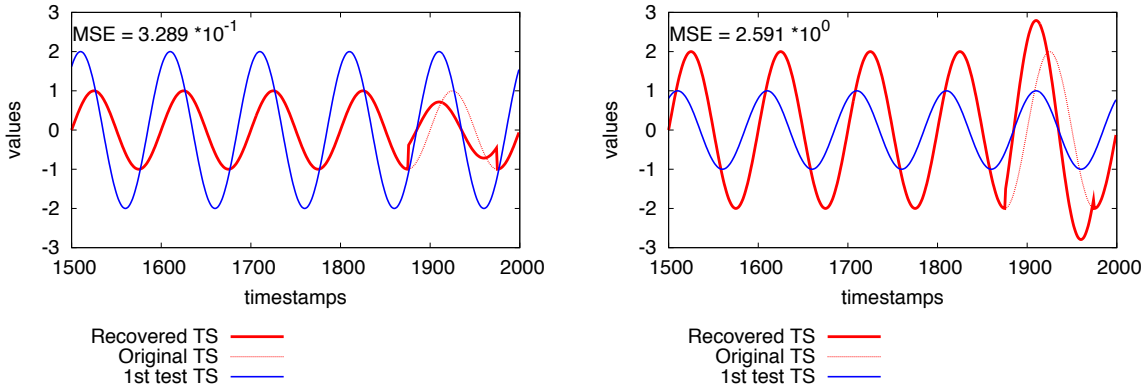
**a) Different amplitudes between two time series**: In the experiment of Figure 5.5, we perform the recovery using shifted test time series and a reference time series with different amplitude. In Figure 5.2(a) the test time series has a higher amplitude than the




(a) Recovery with time shift = 10 (b) Recovery with time shift 18 (c) Recovery with time shift 25

Figure 5.1: Impact of the length of shift

reference time series. The amplitude of the missing block is not accurately recovered. In Figure 5.2(b) the reference time series has a higher amplitude than the test time series. The amplitude of the missing block is not accurately recovered.



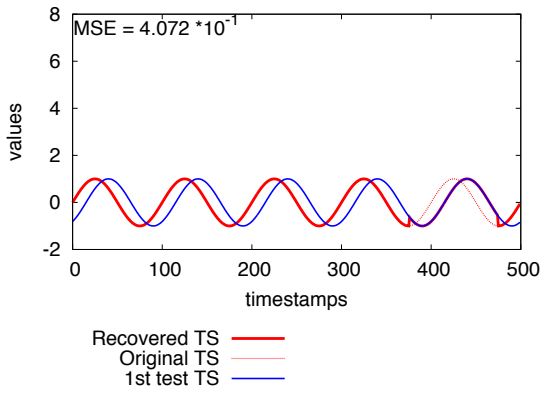

(a) Higher amplitude in test time series (b) Higher amplitude in reference time series

Figure 5.2: Impact of the amplitude

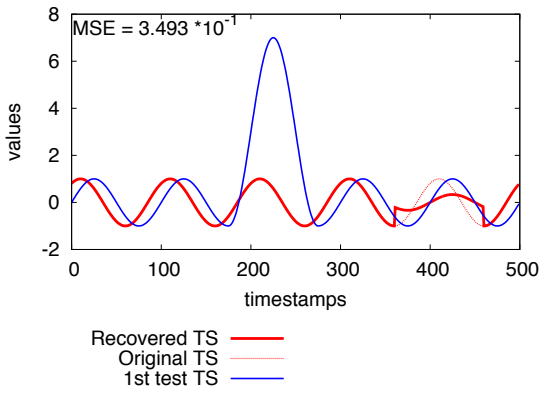
**b) Different amplitudes within the same time series**: In the experiment of Figure 5.3, we perform the recovery using shifted test time series and a reference time series. Goal of this experiment is to test the impact of the different amplitudes within the same time series. In Figure 5.3(a) the test time series has same amplitude along the whole time series. In Figure 5.3(b) the test time series contains a higher peak i.e., it has different amplitudes within itself. Although the assessment based on mean squared error find the recovery accuracy in Figure 5.3(b) better, it is seen that the amplitude of the reference time series are recovered more accurate. If we take a longer history from both time series the impact of the extreme higher peak will reduced, see Figure 5.3(b)

### 5.2.3 Impact of the shape of test time series

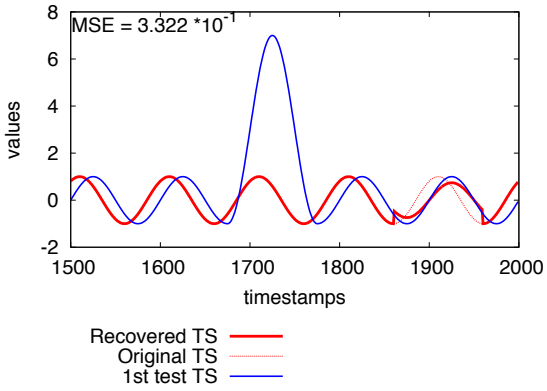
In the experiment of Figure 5.4, we perform the recovery using a shifted test time series and a reference time series with different shape. The result of Figures 5.4(a), 5.4(b) and




(a) Same amplitudes within the same time series




(b) Different amplitudes within the same time series




(c) Different amplitudes within the same time series taking longer history

Figure 5.3: Impact of length of history for the different amplitudes within the same time series

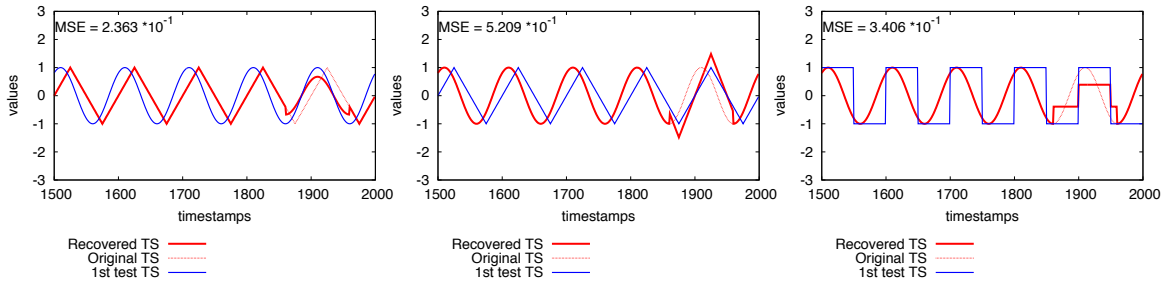
5.4(c) show that the recovered values always imitates the shape of the test time series.

### 5.2.4 Impact of the number of time series

In the experiment of Figure 5.5, we perform the recovery with increasing number of test time series. Test time series are aligned to each other. The result of Figures 5.5(a), 5.5(b) and 5.5(c) shows that the recovery imitates the shape of the test time series. If more than one test time series are used, the amplitude of the recovered block is smaller than the originally one.

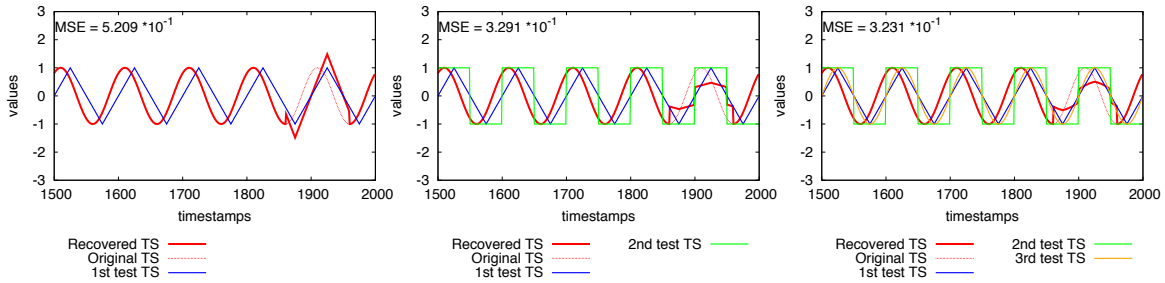
### 5.2.5 Impact of the number of factors

In the experiment of Figure 5.6, we perform the recovery using three test time series and increased the number of factors used for the recovery. The best case is obtained for one used factor as shown in Figure 5.6(a).




(a) Recovery using test time series "sine" (b) Recovery using test time series "triangle" (c) Recovery using test time series "square"

Figure 5.4: Impact of the the shape of test time series



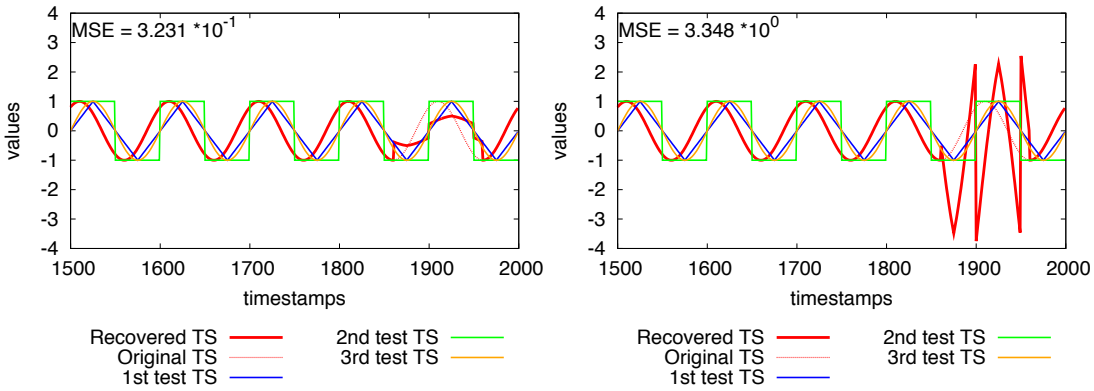

(a) Recovery using one test TS (b) Recovery using two test TS (c) Recovery using three test TS

Figure 5.5: Impact of the number of time series

### 5.2.6 Interpretation of results

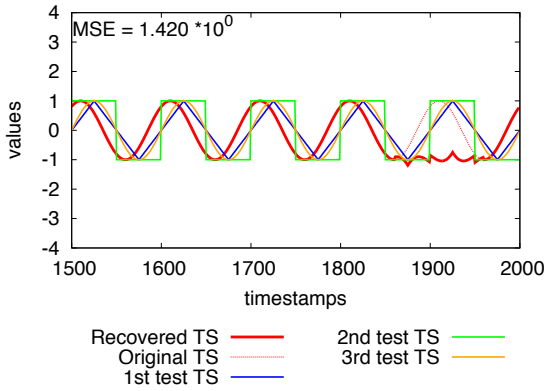
The experiments that we ran yield us to draw some conclusions about the recovery accuracy of the proposed recovery techniques. The main conclusions are the following:

- The recovered values imitate the shape of test time series where the missing block is.
- The smaller the shift is, the more accurate is the amplitude of the recovered values.
- The longer the history of the time series is, the less the effect of extreme values is in the recovery.




(a) Recovery using 1 factor

(b) Recovery using 2 factors




(c) Recovery using 3 factors

Figure 5.6: Impact of the number of time series

# 6 Conclusion

We have presented in this report a graphical recover tool that incorporates the Centroid Decomposition method in order to perform the recovery of missing values in hydrological time series and display it. The implemented tool offers additional functionalities such as to display the entire history of time series and to show the result of the Centroid Decomposition of input matrices. We presented different possibilities to implement the Centroid Decomposition. We described analytical results about the scalability of our technique that has linear space complexity and is able to perform the recovery of time time series containing up to 100'000 values in less than 5 minutes. We applied the recovery techniques for time series that are shifted in time. The experiments we ran show satisfactory results in case of time series with small time shifts. But, the accuracy of our techniques deteriorates with bigger time shifts. We listed the cases where the recovery gives good results in time series shifted in time. We used the Mean Square Error to measure the recovery accuracy of our technique. This measure turned out to be not effective if the technique recovers the same shape of original block but shifted in time. A future direction of this work is propose an effective accuracy measure that could work in the case where the result of the recovery is shifted in time. A more extensive comparison with other recovery techniques is thought to be an interesting point to investigate. Last but not least, proposing a formula that takes as input the time series and the length of the time shift and predicts the shape of the recovery could be part my final Master thesis.

# Appendix

## A. Properties

**Property 1** *Two vectors are* ***orthogonal*** *to each other if their inner product equals zero. An* ***orthogonal matrix*** *is a square matrix with real entries whose columns and rows are orthogonal unit vectors.*

**Property 2** *The* ***normalized vector*** *of* $\boldsymbol{x}$ *is a vector in the same direction but with norm 1.*

**Property 3** *The* ***norm*** *of the n-dimensional vector* $\boldsymbol{x}$ *is denoted as* $\|\boldsymbol{x}\|$ *and equals to the scalar value* $\sqrt{x_1^2 + x_2^2 + ... + x_n^2}$*, where* $x_1, x_2 ... x_n$ *are the components of* $\boldsymbol{x}$*.*

**Property 4** *The* ***Frobenius Norm*** *of a matrix* $A \in \mathbb{R}^{n \times m}$*, denoted with* $\|A\|_F$ *is defined by the equation* $\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2}$

# References


[1] Benjamin, M.,M., :*Missing Data Problems in Machine Learnings*, PhD Thesis, 2008.

[2] HydroloGIS, `http://www.hydrologis.eu/`

[3] Fried R., Gather, U., :*On rank tests for shift detection in time series*, Technical report, 2007.

[4] Khayati, M., and Böhlen, M.H., : *REBOM: Recovery of Blocks of Missing Values in Time Series*, in COMAD, 2012

[5] Chu, M.T., and Funderlic, R.E., : *The Centroid Decomposition: Relationships Between Discrete Variational Decompositions and SVDs*, in SIAM J. Matrix Analysis and Applications, 2002

[6] Fodor, I.,K., :*A survey of dimension reduction techniques*, Technical report, 2002.

[7] Meyer, C.D., :*Matrix Analysis and Applied Linear Algebra*, Book, SIAM publisher, 2001.