Time series outlier detection in spacecraft data

Time series outlier detection in spacecraft data
Master-Thesis von Dang Quoc Hien aus Darmstadt
Dezember 2014

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group

Time series outlier detection in spacecraft data
Time series outlier detection in spacecraft data

Vorgelegte Master-Thesis von Dang Quoc Hien aus Darmstadt

1. Gutachten: Prof. Johannes Fürnkranz
2. Gutachten:

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den December 12, 2014

_____

(Hien Q. Dang)

Contents

1 Introduction

1.1 What is outlier detection

As it is formally defined by Hawkins [1], in data mining, Outlier Detection is the task of finding data points that deviate so much from the remaining data points. It is also known as *Anomalies* or *abnormalities detection* [2]

Outlier detection is often used to discover unusual events or properties from a data set. These unusual characteristics may be caused by changes or malfunction in the system. Outlier detection, on account of that, is applied in many areas such as credit card fraud detection, medical diagnosis, environment monitoring and intrusion detection systems. Applications in these areas generate a lot of data in a long time, however most of them represent the normal behaviors (of users, environment or hardware system). Therefore, outlying data are usually more interesting since they represent the abnormal behavior of the actuators (user, environment or hardware system) that affects the data generation process. Such abnormal behaviors can be merely a new event or still sometimes cause serious damages to the system. For that reason, they need to be reported to and reviewed by human for appropriate actions.

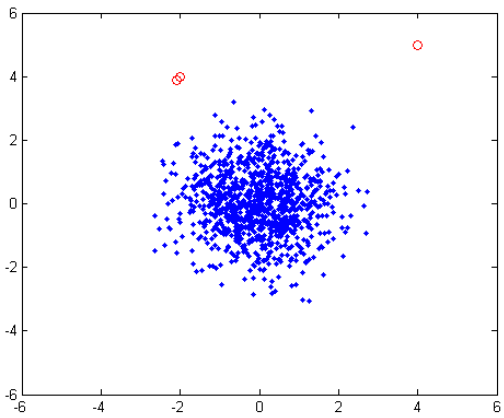


Figure 1.1.: Example of outliers in a 2-dimensional Gaussian distribution. Three outlying data points are marked with red circles

Figure 1.1 shows an example of outliers where data are generated from a 2-dimensional distribution with zero-mean and unit variance. The red data points lie significantly far away from the center (0,0) so it is in doubt whether they are generated by the same distribution. Those data points are considered to be outliers.

Depending on the type of application and whether there are already samples of outliers, the problem of detecting outlier could be solve by supervised method or unsupervised one.

In case of supervised method we assume that there are already some domain knowledge about the data and outliers available. The system that we build should detect outliers based on that knowledge. On the contrary to supervised methods, in unsupervised outlier detection, a statistical approach will be applied to find outliers.

For illustration, let's take a look at an example of credit card fraud detection, in which we are trying to find abnormal behavior from a list of payments (this illustration is taken from [2]):

$$2, 4, 9, 98, 3, 8, 10, 12, 4, 5, 97, 88, 102, 79 \tag{1.1}$$

Unsupervised methods may find payments with amount of 98$ and 97$ suspicious because they noticeably deviate from the recent payments. However payments or transactions with a large amount of money are in reality not so seldom, user may sometimes make a big purchase. Knowing that, supervised methods don't see these payments/transactions as outliers. They only pay attention to a continuous sequence of these payments or transactions, which is not frequent behavior and may imply something abnormal. Thus, 88, 102 and 79 are seen by supervised methods as outliers.

1.2 Classification of Outlier Detection Problems

While anomaly detection is applied in many areas with different type of data, in this work, we will focus on temporal data.

According to the survey from Gupta et al. [3], temporal data can be divided into five main categories:

1. Time Series Data
2. Data Streams
3. Distributed Data
4. Spatio-Temporal Data
5. Network Data

For the first type of data (time series data), application is given a set of time series and tries to find which time series or which subsequences of a time series are outliers. This is the simplest form of finding outliers because the algorithm can visit a time series or a sequence of a time series multiple times. The runtime of the algorithm for this type of data is not critical. Methods for finding outliers in time series data set are usually off-line.

A time series, which has an infinite length, is called streaming data. To process this kind of data, an application must be able to adapt to drifts in each stream. Drifts could be new value range, new behaviors or a new established/disappeared correlation across different streams. Approaches for this type of data usually make use of a model that can be simply updated to capture the characteristics of new data and slowly forget the impact of the old data. In comparison to the first type

of data, algorithms for dealing with streaming data can be seen as online methods. Outliers in streaming data are usually data points since windows based approaches mostly are used in offline task [3]

Outlier detection methods used for the last three types of temporal data (Distributed Data, Spatio-Temporal Data and Network Data) are specialized for different aspects. They are more systems-oriented than the first two. Distributed data focus on the scenario where the data are collected in different nodes (for example of a wireless sensor network). The challenge is how to detect outliers without transmitting much data between nodes [4, 5]. Spatio-temporal data takes into account the additional information of the location where the data points are also recorded. In network data, each data point is a graph of the network. Which means, each data point is basically a set of nodes and vertices. The problem is now to find outlying graphs in the given streams of graphs.

1.3 Problem Statement

Nowadays we are using more and more artificial satellites. They are orbiting the earth and provide us a lot of information that facilitates everyday life activities, researches, military and a lot of different purposes.

One of the most important task of operating satellites is to keep track of the satellites and make sure they are healthy and working properly. In order to do that, a lot of sensors are installed in each satellite to monitor the environment, state of on board devices as well as of the whole system.

Satellites will continuously send these sensor measurements to ground stations. Here the measurements will be scanned by experts and they will decide if it is about a strange behavior, and maybe some appropriate actions should be taken to bring them back to normal state.

Scanning the sensor measurements could be very stressful, since each satellite consists of thousands sensors which generate a large amount of data every few seconds. On the other hand, strange behaviors or malfunctioning of on board devices occur rarely, which means most of the time we observe the data from normal behaviors.

Since we are interested in the abnormal behaviors which are reflected in unusual sensors' measurements, so one can think of using the computational power of computer and outlier detection techniques to help and reduce the works for the monitoring experts.

1.4 Organization of the thesis

The remainder content of this thesis is structured as follows. In chapter 2 we will get to know the Solenix's data set that is used in this works. Furthermore, the general characteristics of the data set are also analysed so that we can better understand it. We also address the difficulties and problems of finding outlier in this data set.

In chapter 3 we will take a look at the background literatures in the field of time series analysis and outlier detection. The techniques and approaches which are either related to this work or used in the data processing pipeline are also described in this chapter.

Our proposed approach and its implementation details are explained in chapter 4. Here we will show how the algorithm deals with multiple dimension streaming data and how it finds outliers. Afterwards, we will describe how the data processing pipeline can be configured so that it can be applied on different data set.

Chapter 5 presents an evaluation of the algorithm and reviews the outliers that are found in the Solenix data set. The performance of the algorithm and set up of each experiment are also discussed in this chapter.

Chapter 6 provides the summarization and a conclusion for the thesis. We will also discuss about possible directions of future works

2 Characteristics of Data Set

Since the goal of this thesis is mining time series data to detect outliers, so we should firstly get to know the data. In this chapter, we will examine the general characteristics of the Solenix's data set which is mainly used in this work.

This data set consists of measurements from a satellite. They were recorded in about 12 days and stored in form of multiple time series. Each time series in this data set is a sequence of measurements and their recorded time stamp generated by a sensor or an on-board device.

In total, the data set consists of 1288 time series with 43,918,229 data points (on average there are $\tilde{3}4098$ data points per time series). However, this is still considered as a small data set since the number of time series could be much bigger (up to 40,000 time series). So it is required that the outlier detection algorithm can be scaled up to deal with larger number of time series.

2.1 Value Range and Sample Rate

2.1.1 Value Range

One notable thing in the data set is that the values of each time series fluctuate widely. The minimum and maximum values of a time series can be very small from 4.2×10^{-4} to 5.2×10^{-4} or large from 0 to 10^{12}. This also implies that the other statistical characteristics of each time series such as variance, mean and range are also distributed in a wide domain. By computing the variances of 1288 time series we can see that the smallest variance is about 10^{-15} and the maximum variance is greater than 10^{17}. In figure 2.1, the variances are plotted in increasing order (Note that because of the variation is too big, we have plotted the variances using logarithmic scale on the y-axis). Still, the extreme variances are found in a small number of time series, most of the them (about 84%) has variance smaller than 1000 (see table 2.1).

This characteristic is caused by many reasons, one of them is the difference in measurement's units. Each time series represents a sensor which measures different type of data like temperature, coordination, voltage, light or gyroscope.... The units of these data type are incomparable thus it causes the huge different between time series.

The enormous difference in scale of time series could effect the performance of algorithms in data mining. Thus, it is one of the problems that we have to deal with in later part of this work.

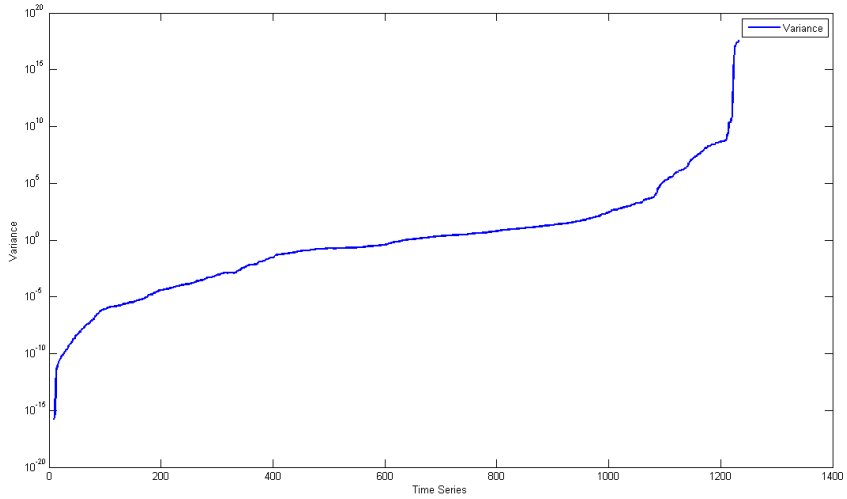


Figure 2.1.: Variance of 1288 time series from Solenix data set. Y-axis is set to logarithmic scale

Variance	Number of TS	Percentage
0-1	636	51.58%
1-1000	402	32.60%
>1000	195	15.82%

Table 2.1.: Variance of Time series

2.1.2 Sample Rate and Synchronization

Despite all time series are recorded from the same satellite, they have different sample rates and not synchronized. The time series with highest sample rate has 259,199 data points (it is four data points every second) and the one with lowest sample rate contains only 6 data points. However most of them have high sample rate, 1192 out of 1288 time series (about 92.55%) have more than 8,000 data points (see table 2.2). As the most approaches for outlier detection in streaming data [6, 7, 8], we also don't want do deal with flexible arrival rate. Since processing and finding outliers in multiple time series in such case is stressful and not effective. Solutions for

Data points per TS	Number of TS	Percentage
>100.000	29	2.25%
>50.000	332	25.78%
>20.000	688	53.42%
>8.000	1192	92.55%
>5	1288	100.00%

Table 2.2.: Distribution of the sample rates within 1288 time series. The first column is the constraint for the number of data points. The second column shows how many time series fulfill that constraint. Third column show the result in percentage

dealing with such time series is usually to try to bring all them to the same level of sample rate (by doing up/down-resampling).

Another important characteristic of the dataset is that most of the time series are stationary. By simple plotting out the each time series we can observe that they often contain six repeated periods (Figure 2.2). This observation is used later on when we want to determine how long the algorithm should wait before starting to find outliers. The waiting time is needed for some parameters to converge.

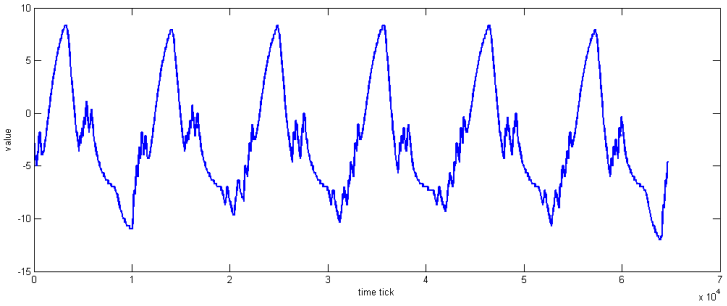
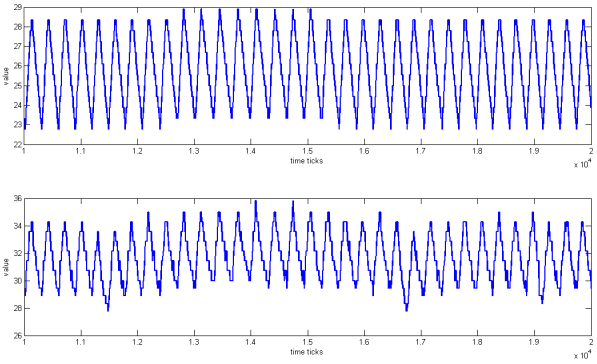


Figure 2.2.: Most of time series consists of repeated pattern

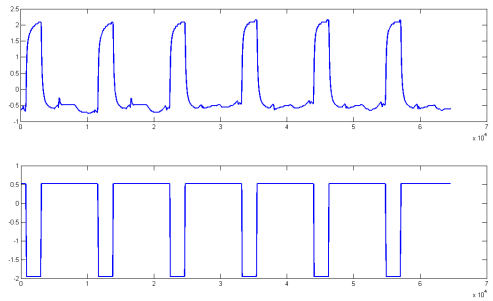
2.2 Correlation between Time Series

Because the time series are recorded from sensors in the same satellite, they reflect the same state or behaviors of the satellite and are usually correlated. For example two temperature sensors in the same part of the satellite should produce similar measurements and therefore they are positively correlated. Contrariwise, the measurements of two temperature sensors placed in opposite side of a satellite are usually negatively correlated (When one side is facing the sun, its temperature will increase and temperature on the other side will decrease).

By plotting each time series from the data set we can obverse this characteristic in a more intuitive way. In figure 2.3 we show two examples of positive correlation (figure 2.3a) and negative correlation (figure 2.3b).



(a) Two positively correlated time series



(b) Two negatively correlated time series

Figure 2.3.: Examples of correlated time series from Solenix's data set

In this work we will exploit the correlation to compress the enormous number of input time series into fewer linearly uncorrelated time series and try tracking the correlation to find outliers.

3 Theoretical Backgrounds

3.1 Feature Scaling

3.1.1 Why scaling?

Feature scaling is one of the well-known techniques that is commonly used in data pre-processing. It is used to bring different features to the same scale, whereas the *scale* could be interpreted as range of possible values, mean, variance or length (in case of scaling vectors). The different in scale of features may have negative impacts on many algorithms in machine learning and data mining, thus it is considered as a requirement for those algorithms [9].

For example clustering methods like k-means suffers from unequal variance of feature. That is because k-mean is isotropy and when the variances of some features are much bigger than the others', k-mean is more likely to split the clusters along those features and ignore the features with small variance. In the case of gradient descent, unequal variance of features makes the convergence process become slower.

The importance of normalization in neural network is reported by Sola et al. in [10]. Also, the roles of feature scaling in support vector machine are reported in [11] by Hsu et al. In general, to equalize the effect of each feature has on the results, rescaling should be used on the numeric features [12].

Based on the statistical properties of the features, there are different scaling methods. In the following sections we will introduce briefly two conventional re-scaling methods: the Min-Max Normalization and Z-Score Standardization. There are more scaling techniques of different fields in literatures like Decimal scaling [12], Bi-Normal Separation (BNS) feature scaling [13], Rank normalization [14] or Transformation to Uniform random variable [15]

3.1.2 Min-Max Normalization

Min-Max Normalization is the simplest form of feature scaling and usually denoted shortly as 'normalization'. Min-Max Normalization re-scales features in the mean of range of possible values. After normalizing all features will lie in the same range.

The common ranges are $[0, 1]$ and $[-1, 1]$. Another range could also be used and can be derived from results of normalizing into $[0, 1]$. Min-Max Normalization of a random variable x is based on its minimum and maximum values and is given by form 3.1. In general, the minimum and maximum values are taken from the training set to normalize the new data (in test set).

$$\bar{x}_t = \frac{x_t - min(x)}{max(x) - min(x)} \tag{3.1}$$

3.1.3 Z-Score Standardization

Z-Score Standardization is a method for re-scaling a feature in the mean of variance and mean value. It aims to transform all features so that they have the same variance and mean. Usually the features are re-scaled to zero-mean and unit variance. In such case, Z-score standardization of a random variable x is given by following form:

$$\bar{x}_t = \frac{x_t - \mu}{\sigma} \tag{3.2}$$

Where μ and σ are the mean and standard deviation of x. Z-Score is often used when the features are measured in different metrics or varies from each others.

For example, a feature represents room's temperature in Celsius which may vary from and 0° to 40° and another feature represents light intensity in luminous flux (lux), which may vary from 0 lux (complete darkness) to 100000 lux (direct sunlight). An clustering algorithm may split this data set just according to the light feature since it has the most variance.

3.2 Moving Average

Moving average is a method used in time series analysis to cancel the temporary changes or noise and highlight the main trend.

Simple Moving Average

In the simplest from of moving average the value of current data point is computed as the average value of the last **n** data points.

$$SMA(x_t) = \frac{x_t + x_{t-1} + \ldots + x_{t-(n-1)}}{n} \tag{3.3}$$

For simple moving average, we need to remember the last n value of each time series. It can also be implemented more efficiently using the incremental update form:

$$SMA(x_{t+1}) = SMA(x_t) - \frac{x_{t-(n-1)}}{n} + \frac{x_{t+1}}{n} \tag{3.4}$$

Weighted Moving Average

Weighted moving average also computing the average of recent data points but it places different factors on data points. The weight of each data point usually a fixed function of its relative position to the current data point. The purpose of this method is often to put higher weight on recent values and therefore the current value is up to date.

$$WMA(t) = \frac{\alpha_1 x_t + \alpha_2 x_{t-1} + \ldots + \alpha_n x_{t-n+1}}{\sum_{i=1}^{n} \alpha_i} \tag{3.5}$$

We choose $\alpha_i, i = 1...n$ so that $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_n$.

Exponential moving average is a special case of weighted moving average where $n \to \infty$ and $\alpha_i = \alpha^i$ with α is a value in range (0,1)[16]. The weight of a data point exponentially decreases to its relative position from the current data point

$$EMA(t) = \frac{x_t + \alpha^1 x_{t-1} + \alpha^2 x_{t-2} + \ldots + \alpha^i x_{t-i} \ldots}{\sum_{i=0}^{\infty} \alpha^i} \tag{3.6}$$

$$\approx (1-\alpha)x_t + \alpha \times EMA(t-1) \tag{3.7}$$

One of the advantages of exponential moving average is that the computation process is very simple. According to equation 3.7, it needs to remember the last EMA value (at time $= t-1$).

3.3 Dimension Reduction

3.3.1 Principal Component Analysis

In machine learning, Principal Component Analysis (PCA) is a well-known technique for reducing the dimension of data. It can capture the linear correlation between variables (or features) and transforms them into fewer linearly uncorrelated hidden variables. When dealing with multi-dimensional data we may try to use PCA to reduce the number of dimension first and then we apply our learning methods on the reduced data. The number of dimension needed for capturing the major characteristics of a given high-dimension data set is usually much smaller than in the original data. In the worst case, it can only reach the number of dimension of original data.

The idea of PCA is to find out, on which subspace of the feature space, the data could be best represented. For example if we have two correlated random variables (see Figure 3.1). We project this 2-dimensional data set on vector $\vec{u_1}$ so that each data point can be now represented as $\alpha\vec{u_1}$. This representation will capture the most significant information of the data set and the reconstruction error is minimal. The best direction to project the data on is the direction with the most variation of the data.

Formally, let x be a M-dimensional random variables and we have N samples of x: $\{x^1, x^2, ..., x^N\}$ where $x^i \in \mathbb{R}^M,\ i = 1...N$
Let y_1 is the projection of x on u_1 so y_1 is the product of x and u_1 ($y_1 = x \times u_1$). The first principal components is the vector u_1 so that the projection of x on u_1 has maximum variance:

$$u_1 = \arg\max_{u_1}(var(y_1)) \tag{3.8}$$

$$= \arg\max_{u_1} \sum_{i=1}^{N} \frac{1}{N}(y_{1,i} - \bar{y}_1)^2 \qquad = \arg\max_{u_1} \frac{1}{N} \sum_{i=1}^{N} (x_i u_1 - \bar{x} u_1)^2 \tag{3.9}$$

$$= \arg\max_{u_1} \frac{1}{N} \sum_{i=1}^{N} u_1^T (x_i - \bar{x})(x_i - \bar{x})^T u_1 \qquad = \arg\max_{u_1} u_1^T \Sigma u_1 \tag{3.10}$$

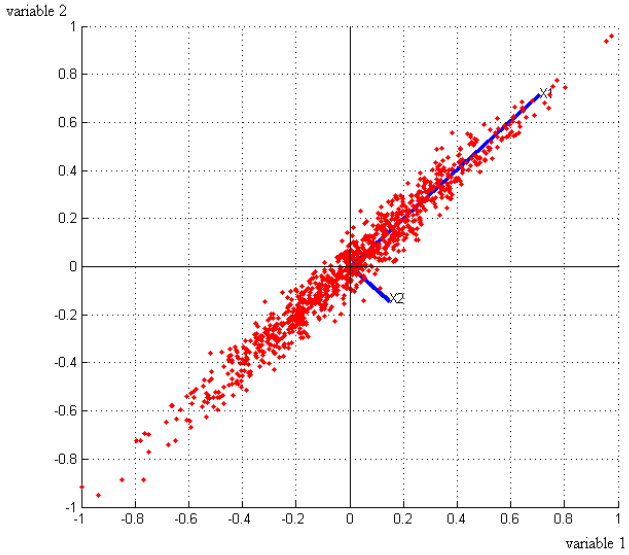


Figure 3.1.: Example of PCA on 2 dimensional data space

with the normalization constraint that $\|u_1\| = 1$ (or $u_1' u_1 = 1$) and Σ is the covariance matrix of x:

$$\Sigma = E[(x - \bar{x})(x - \bar{x})^T] \tag{3.11}$$

This is a constrained maximization problem so we can use Lagrange multiplier to transform it to dual problem of maximizing [17]:

$$L(u_1) = u_1^T \Sigma u_1 - \lambda(u_1^T u_1 - 1) \tag{3.12}$$

solve it by derivative with respect to u_1 we have

$$\Sigma u_1 - \lambda u_1 = 0 \tag{3.13}$$

when u_1 and λ fulfill this equation, they are the eigenvector and eigenvalue of Σ. The eigenvalue λ is also the variation of the data set along the principal direction u_1. Therefore the first principal direction is the eigenvector with greatest eigenvalue λ.

To sum up, for doing PCA on a data set x we have to:

1. Compute the covariance matrix Σ from the data set x. The complexity of this operation is $O(M^2 N)$

2. Compute the eigenvalue and its eigenvector of Σ. This take $O(M^3)$

Additionally in the case of streaming data we have to access the past data points for computing Σ.

3.3.2 SPIRIT

In this section we present an alternative approach, called SPIRIT, proposed by Papadimitriou et al.[18] The name of this method (SPIRIT) stands for Streaming Pattern Discovery in Multiple Time-Series. What is can do is similar to PCA but it is specialized to multiple time series and more importantly, it works in an online manner and therefore is suitable for dealing with streaming data.

Instead of computing the eigenvector and eigenvalue from the dataset, SPIRIT learns the eigenvector incrementally from the data set. It firstly initiates some dummy eigenvectors as unit vectors

$$u_1 = \begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}, u_2 = \begin{pmatrix} 0 \\ 1 \\ \dots \\ 0 \end{pmatrix}, \dots, u_k = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \end{pmatrix}$$

For each new incoming data point, the eigenvectors are updated according to the error that it makes when using current eigenvectors to project the new data point. The first principal direction will be updated first and the process continues in the order of decreasing eigenvalues. Error of a data point that cannot be captured by the first principal direction is left as input for updating the next principal directions. The i-th eigenvector is updated as follows:

- Project input data point to the i-th eigenvector: $y_{i,t} = u_i^T \hat{x}_{i,t}$
- Compute the error that it made by using i-th eigenvector: $e_i = \hat{x}_{i,t} - y_{i,t} u_i$
- Estimate energy of all data points along i-th eigenvector: $d_i = \lambda d_i + y_{i,t}^2$
- Update the principal component based on the error, projection and learning rate: $u_i = u_i + \frac{1}{d_i} y_{i,t} e_i$
- Error that cannot be captured by the first i principal components is left as input for updating next eigenvectors $\hat{x}_{i+1} = \hat{x}_i - y_i u_i$

Using this update process is actually applying gradient descent to minimize the squared error e_i^2 with a dynamic learning rate d_i. Therefore, u_i will eventually converge to the true i-th principal direction. We can show it formally by deriving the squared error function as follows:

$$E_i = \frac{1}{2}e_i^2 = \frac{1}{2}(\hat{x}_i - y_i u_i)^2 \quad \text{(squared error function)}$$

$$\nabla E_i = \frac{1}{2}2(\hat{x}_i - y_i u_i)(-y_i) = -e_i y_i \quad \text{(take the derivation of squared error with respect to } u_i\text{)}$$

$$u_i = u_i - \gamma \nabla E_i(u_i) = u_i - \gamma(-e_i y_i) = u_i + \frac{1}{d_i}e_i y_i \quad \text{(using the gradient descent update form with the learning rate } \gamma = 1/d_i\text{)}$$

As we see that the learning rate The way d_i is computed makes it proportional to the variation of the past data point a long u_i. The learning rate is set dynamically to $\frac{1}{d_i}$ means that if the variation of the past data points a long the i-th principal component is great, then gradient descent will make a small step. If the variation is small, so gradient descent will make a big step. In general it is desirable to have adaptive learning rate that makes a big step when it is far from the optimal value and slows down when it gets closer to the optimal value.

Another advantage of SPIRIT over conventional PCA is that SPIRIT can automatically detect the number of needed hidden variables on the fly. SPIRIT tracks the number of hidden variables by comparing the energy of the original time series (E) and of the hidden variables ($\tilde{E}$). The algorithm will adjust the number of hidden variables to keep ratio of $\tilde{E}/E$ in a certain window. In [18], the window is set to $[0.95, 0.98]$. This condition could be formulated as:

$$0.95E \leq \tilde{E} \leq 0.98E \tag{3.14}$$

When $\tilde{E}$ is greater than $0.98E$, it means that we do not need that many principal components to capture the significant information of the data set. Therefore SPIRIT will decrease the number of hidden variables. If $\tilde{E}$ is smaller than 0.95, it means that the original time series are poorly represented by current principal components and thus SPIRIT will increase the number of hidden.

However the way SPIRIT detects number of principal components does not always work because the contribution of each time series to the total energy is different. In general, it requires that the input time series have the same energy level. We consider it as an additional requirement for SPIRIT and will discuss about this in detail on section 4.2.

3.3.3 Autoencoder

Autoencoders are also known as autoassociators or Diabolo networks [19, 20, 21]. They are trained to compress a set of data so that we could have a better representation in a lower dimensional space.

An autoencoder is a special neural network in which its input-layer and output-layer have the same size (**N**-dimensions) and there is at least a hidden layer in the middle that connect two parts. The hidden layer's size is smaller (**K**-dimensions, $\mathbf{K} < \mathbf{N}$) than input and output layer. The middle-hidden-layer is also called the code-layer since it is the new representation of compressing the input data. A simplest form of autoencoders with one hidden layer is show in figure 3.2. The first part of the network (from the input to code-layer) is called encoder since it encodes the input. The second part (from the code-layer to output) is called decoder because it reconstruct the input from the code-layer.

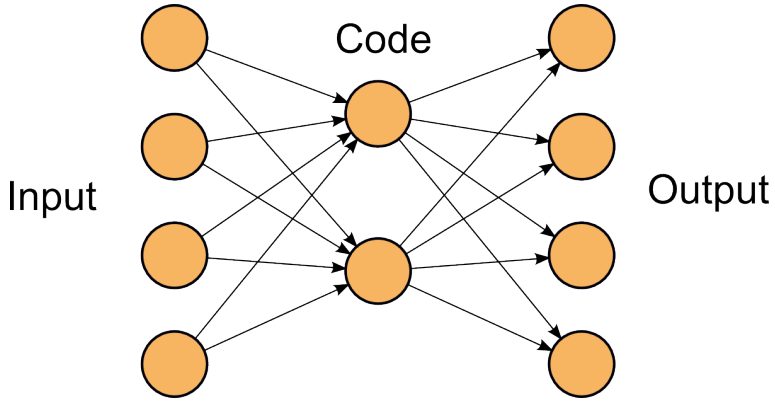


Figure 3.2.: An autoencoder with one hidden layer consists of 2 nodes. The input/output layer have 4 nodes. This autoencoder can be used to compress 4 dimensional data to 2 dimensional data

If we use the simplest form of autoencoder (one hidden linear layer) and the mean squared error for training it, then **K** units from the code layer span the same subspace as the first **K** principal components by PCA [22]. However the principal components from PCA are not equal to the hidden units of code-layer. The principal components are orthogonal and are picked so that the i-th component contains the most variance which is not covered by the first $i-1$ components. Whereas the hidden units of an autoencoder are not orthogonal and tend to be equal.

As we can see that the difference between an autoencoder and a normal neural network for other task is that, the size of input and output layer is equal. Furthermore, The training process of autoencoders aims to find the optimal weights, so that the output is as close as possibly to the input. If it is successful to do such job, that means the autoencoder can compress the input from **N** dimensions to **K** dimensions (as the size of hidden layer). And from this **K**-dimensions data, it can also reconstruct back the original data in N dimensions without losing much information.

From the approach of one-hidden-layer autoencoders, we can generalize it by adding extra hidden non-linear layers to the architecture and form a deep autoencoder (see Figure 3.3). With these additional hidden layers, deep autoencoders can capture the non-linear correlation of the data and therefore provide a better result in data compression and representation. However a deep neural network is hard to optimize. In recent researches in deep learning, Hinton et al. have successfully trained deep autoencoders for image and text document data [23]. In this publication, the deep autoencoders are pre-trained layer-by-layer using restricted Boltzmann machine.

Afterwards, all pre-trained layers are assembled (unrolling) and then backpropagation is used to train the whole deep network. The results of deep autoencoders in representation and reconstruction are much better than by using PCA. However because

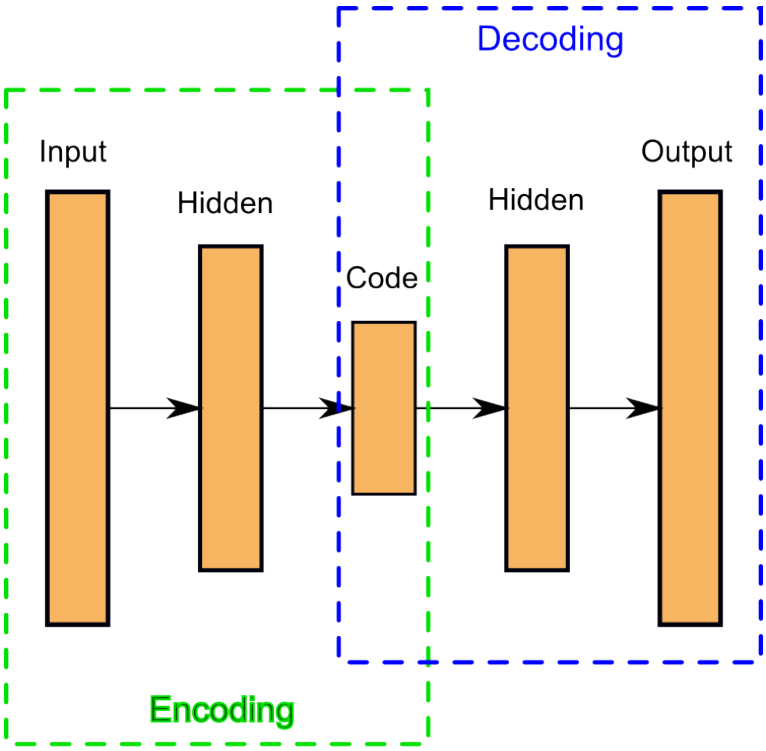


Figure 3.3.: Structure of a deep autoencoder

of the complex structure of deep autoencoders, it is hard to track the correlation between time series and to explain the outliers. This could be considered as a drawback of deep autoencoders.

3.4 Alternative Approaches

In this section we will review two alternative approaches in the context of detecting outliers in streaming data.

3.4.1 AnyOut

Most of approaches for detecting outliers in streaming data assume that the data streams are synchronized and have a fixed sample rate. Recently, Assent et al. introduced an algorithm called Anyout [24](Anytime Outlier Detection), which also deals with flexible arrival rate of streaming data. It finds outliers without assumption about sample rate.

The approach of Anyout is based on hierarchical clustering algorithm (here ClusTree from Kranen et al.[25] is used). The clusters are ordered in a tree structure in which the children nodes are clusters that contain more fine grained information than the parent node. As a data point arrived, Anyout will traverse the tree to determine two outlier scores:

- Mean Outlier Score
- Density Outlier Score

By this approach, Anyout also takes into account available time to process a data point. As long as the there is no new data point arrived, the current data point is still processed and the algorithm is able to go to deeper level of the clustering tree, therefore, result of outlier score is more accurate.

3.4.2 SmartSifter

SmartSifter was presented by Yamanishi et al. in [26, 27]. It is an unsupervised algorithm which use online sequential discounting approach and can incrementally learn the probabilistic mixture model from the time series data.

To encounter concept drift, SmartSifter also includes a forgetting factor which make the algorithm gradually forget the effect of past data points.

Furthermore, SmartSifter consists of two algorithms for dealing with categorical variables and continuous variables.

With categorical variables, the Sequentially Discounting Laplace Estimation (SDLE) algorithms is used. It divides the feature space in to cells. The probability that a data point x lies in a cell is simply the ratio of total number of data points in that cell divided by the total number of data points with Laplace smoothing applied on top it.

To deal with continuous variables, SmartSifter used two models, one independent model for the data distribution and one model for time series.

For the data model, they proposed two algorithms for learning mixture parametric models and non-parametric model

- Sequentially Discounting Expectation and Maximizing (SDEM) for learning the Gaussian mixture model (parametric)
- Sequentially Discounting Prototype Updating for learning Kernel mixture model (non-parametric)

For time series model, the Sequentially Discounting AutoRegressive (SDAR) algorithm is used that learns AR model from the time series with decay factor.

4 Approach and Implementation

4.1 Approach

As we see in section 3.3.2, the SPIRIT method introduced by Papadimitriou et al. provides a fast and incremental way to find the principal directions. On that we can reduce the number of time series to hidden variables and reconstruct them back. When the input time series are correlated with each other, this method works effectively and can compress the inputs into a few output time series. Based on this, we approach the problem of finding outlier in multiple correlated time series, which is caused by one (or some) time series suddenly break the correlation. This type of outlier could be interpreted as a disturbance of some sensors or malfunction in a part of the system.

Assume that we have a set S of k correlated time series $x_1, x_2, \ldots, x_k$. Further, assume that from time tick t to $t+p$, the i-th time series (x_i) suddenly changes and breaks the correlation (outlying data points). At the beginning of these outlying data points (time tick t), the old principle directions are used for projecting and reconstructing. Since outlier in x_i breaks the correlation, the old principle directions cannot capture these outlying data points. Therefore the reconstruction error at this point will be significantly higher than at normal data points. As the time tick increase from $t+1$ to $t+p$, SPIRIT will slowly adapt to the change by adjusting the principal directions and possibly also by adjusting the number of hidden variables if the error is too high. However the adjusting process also takes into account the past data point, so it is quite slow and therefore the reconstruction error are still higher at these time ticks.

For illustration, we apply SPIRIT on five correlated time series which are generated by sinus function with Gaussian noise. The generated time series have zero mean and variance of 0.5. We choose one time series from those and put an outlier into it by setting the values of a small window to zeros (here we used index from 800 to 815, see Figure 4.1). Note that, zero is in the range of sinus function $[-1, 1]$ and not a new value of the time series.

Because all the input time series are correlated so they can be compressed into one hidden variable. From this one, input time series that do not contains outliers can be reconstructed with relatively small errors. For time series with the outlier from $t = 800$ to 815, the break in correlation results in a high peak in its reconstruction error. By running a simple extreme value analysis on the reconstruction error we can find the outlier.

As we have explored the characteristics of the time series data recorded from a satellite in chapter 2 and see that there are many correlated time series. The problem of detecting outliers in this data set can be approached by using SPIRIT and the reconstruction error as described above.

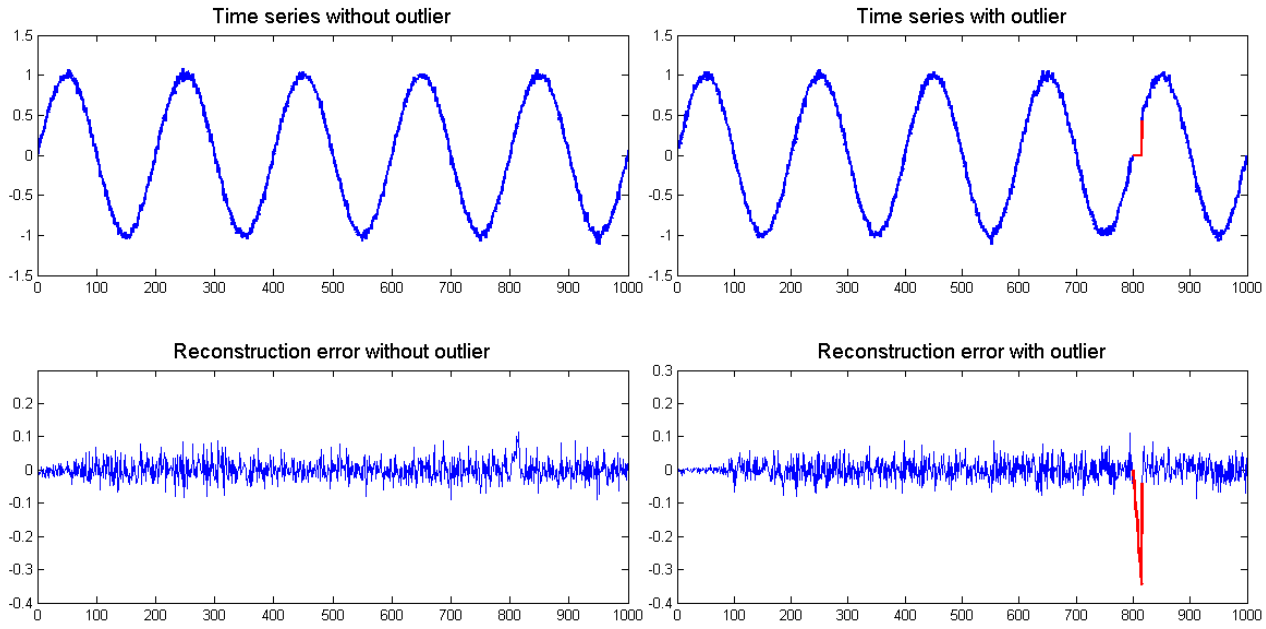


Figure 4.1.: Detecting outliers in time series data which are generated by sinus function

4.2 Requirements for SPIRIT

As mentioned in Section 3.3.2, a very desirable property of SPIRIT is that it can adjust the number of hidden variable on the fly. This could be considered as an advantage of SPIRIT over PCA and autoencoder since we can start with any number of principal components and it will automatically find out how many principal components are needed.

The criterion for finding the number of hidden variables is to use energy thresholds. Where the total energy of original time series and of the reconstructed time series at time t (E_t and $\widetilde{E}_t$) are defined as the average of square values of each data point [18]:

$$E_t = \frac{1}{t}\sum_{\tau=1}^{t}\left\|x_{\_,\tau}\right\|^2 = \frac{1}{t}\sum_{\tau=1}^{t}\sum_{i=1}^{n}\left\|x_{i,\tau}\right\|^2 \tag{4.1}$$

$$\widetilde{E}_t = \frac{1}{t}\sum_{\tau=1}^{t}\left\|y_{\_,\tau}\right\|^2 = \frac{1}{t}\sum_{\tau=1}^{t}\sum_{i=1}^{n}\left\|\tilde{x}_{i,\tau}\right\|^2 \tag{4.2}$$

Here $x_{i,\tau}$ is the data point of i-th in put time series at time tick τ; $y_{\_,\tau}$ is the projections on principal directions at time τ and $\tilde{x}_{i,\tau}$ is the reconstruction of $x_{i,\tau}$ from the projections.

The algorithm adjusts number of hidden variables so that the total energy of hidden variables retain in the range from 95% to 98% the total energy of original time series:

$$0.95 \times E_t \leq \widetilde{E}_t \leq 0.98 \times E_t \tag{4.3}$$

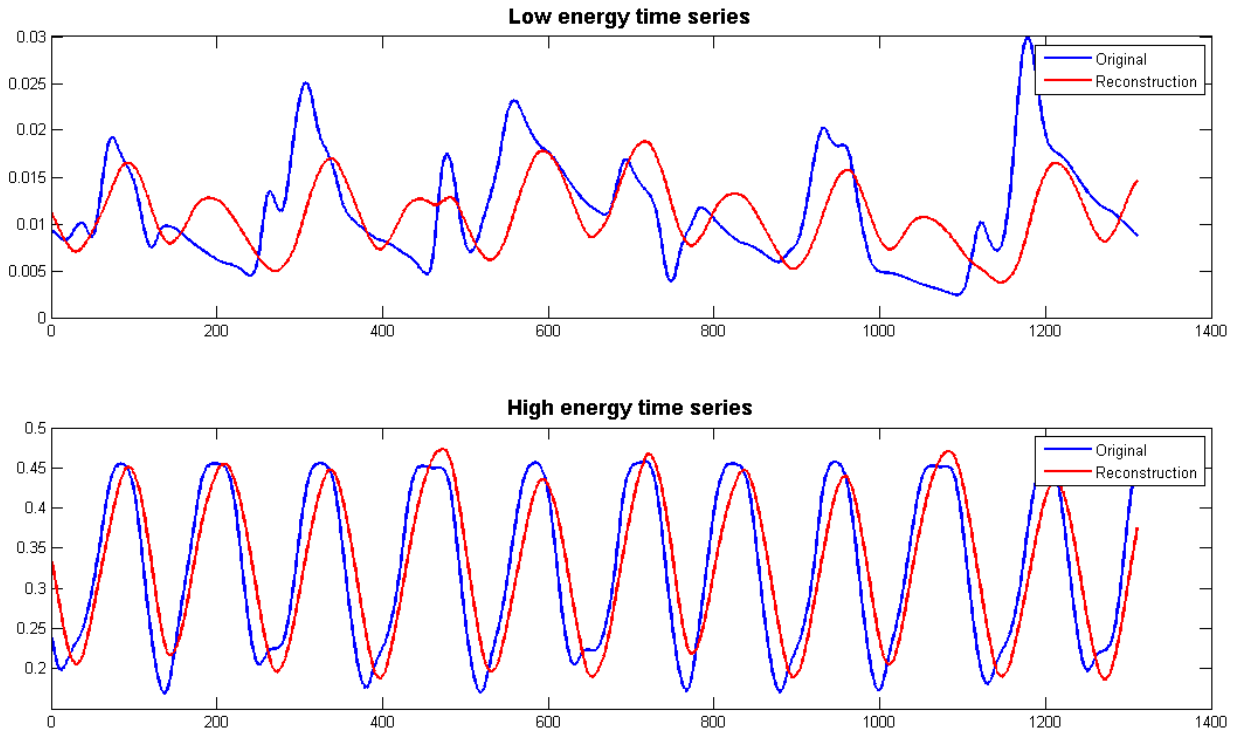


Figure 4.2.: High and low-energy time series with their reconstruction. Data are taken from chlorine dataset[28]

As it is proven in [18], this criterion is equivalent to keep the relative reconstruction error of all time series in the range from 2% (1-0.98) to 5% (1-0.95):

$$1 - F_E \leq \frac{\sum_{\tau=1}^{t} \left\| \tilde{x}_{\_,\tau} - x_{\_,\tau} \right\|^2}{\sum_{\tau=1}^{t} \left\| x_{\_,\tau} \right\|^2} \leq 1 - f_E \tag{4.4}$$

However as we have examined the data set in chapter 2, time series in our data set have very different scales. Thus, their energy is varying, it could lead to a problem that the algorithm treats each time series unequally.

With that concern, we run a test on the chlorine data set[1] which is one of the data sets used for evaluating SPIRIT [28]. On this test, the algorithm also discovered two hidden variables from 166 input time series (the same result as reported in [18]). When we plot the reconstructions and compare it with the original time series, only time series with high energy could be well reconstructed. The reconstructions of low-energy time series do not approximate very well their original time series. An illustration is showed in figure 4.2, where the plots represent two original time series (blue lines) with very different energy levels. The contribution to total energy of the first time series is about 681 times more than the contribution of the second one. The reconstructions of each time series are showed in red lines. We can see that the the time series with high energy and its reconstruction are quite alike (they are well reconstructed), while the one with lower energy and its reconstruction are not.

[1] Sensor measurements of chlorine concentration of drinking water in the distribution network

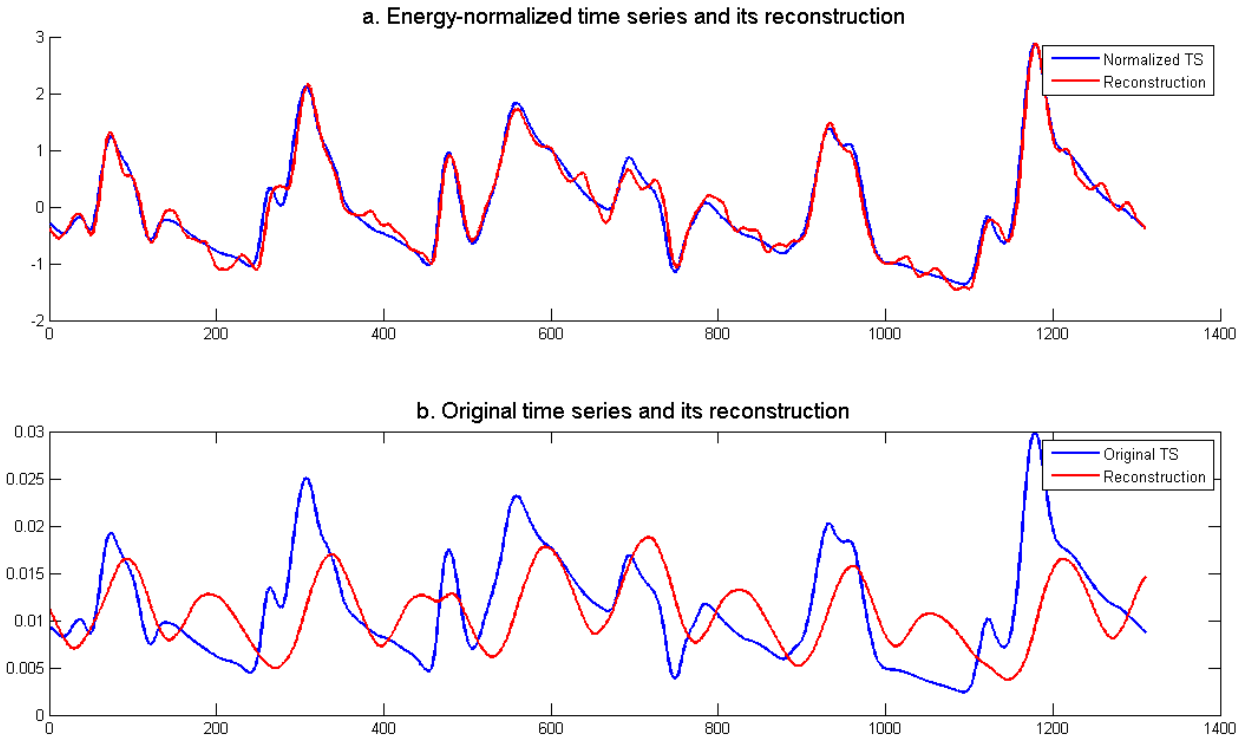


Figure 4.3.: Original time series and reconstructed time series from chlorine dataset [28]

This could be explained that if we apply the algorithm on time series with different scales, they will contribute differently to the total energy. The contribution of time series with higher energy dominates the lower ones, thus the number of hidden variables are regulated by the higher-energy time series. As a result, the projections are more sensitive to high-energy time series rather than of the low-energy ones.

To deal with this problem, one could think of using a different criterion for determining the number of hidden variable or trying to rescale each time series so that they all have a same energy level, and thereby equalize the role of each time series.

We run the second experiment on the same chlorine data set[28], but before putting the data into SPIRIT, all time series are re-scaled using Z-Score standardization using the formal 3.2. For standardization, the mean (μ) and standard deviation (σ) of each time series are computed directly from the data set. After this preprocessing step, all time series are standardized and have zero mean and unit variance. The other parameters needed for running SPIRIT are kept the same as in the first experiment.

The result of second experiment shows that the algorithm now needs 14 hidden variables for compressing 166 input time series (instead of just two as in the first experiment). However not only time series, which originally have high-energy, are well reconstructed but also low-energy time series. In figure 4.3 we can compares the result of reconstruction before applying standardization (figure 4.3b) and after standardization (figure 4.3a).

So by normalizing the energy, the algorithm treats all time series equally. It takes into account the low-energy time series, therefore new hidden variables are needed to be able to capture all time series into hidden variables.

4.3 Data Processing Pipeline

This section gives you, firstly, an overview of how the system work. We will then briefly describe the functionality of all steps in the data processing pipeline as well as how they are connected to the others. In the following subsections, each step of the pipeline is also discussed in details.

The first phase is preprocessing. In this phase, the incoming data points from each data stream is smoothed out by applying triangular moving average. As we have discussed in 4.2, we need to make sure the all time series are treated equally so the data points are rescaled by using z-score standardization. Output of this step are rescaled data streams with approximately same mean and variance.

In second phase, the standardized data streams are put into SPIRIT. The input streams are projected to the principal directions and the weight matrix are updated for each new data point. Outputs of this phase are streams of hidden variables, reconstructed data stream and the weight matrix. In the third phase, the re-

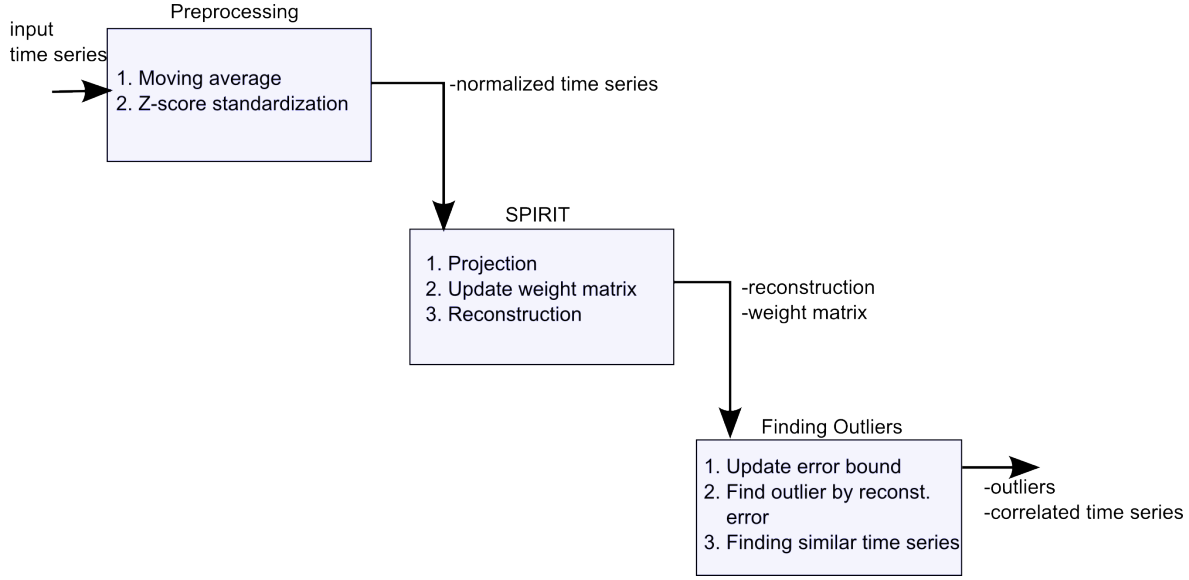
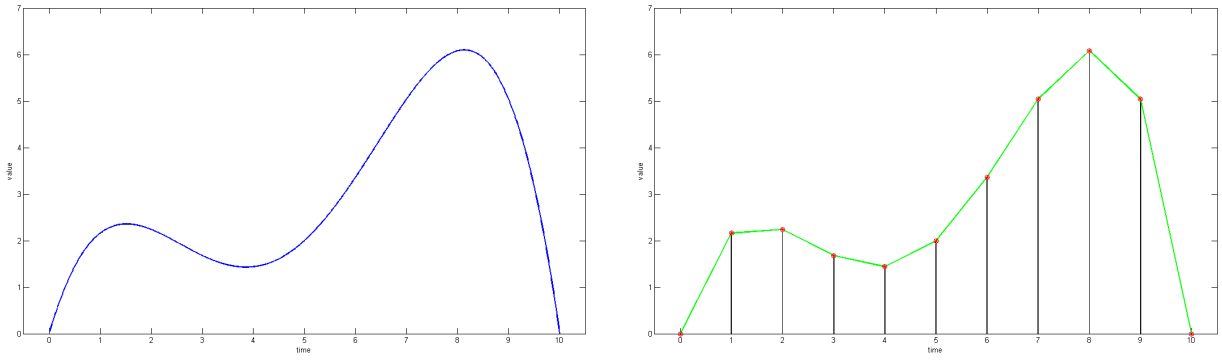


Figure 4.4.: Three phases in the data processing pipline

construction error is used to determine whether there is outlier or not. If it was detected that there is an potential outlier, the index of time series which are causing this outlier are thrown as the output. System could then use this and raise a warning so that experts can review them and decide if that was an actual outlier. Also in this step, a list of time series, which have the similar shape as the one with outliers, are also thrown out as additional output. This extra information could be used as an explanation for the outlier. Details about finding similar time series when outlier occurs is discussed in section 4.4

4.3.1 Preprocessing

We already see in section 4.2 that standardization should be applied to each time series before running SPIRIT. Together with standardization, we also use other pre-



(a) Original function **(b)** Resampled data points

Figure 4.5.: Down-resampling of a continues function to 11 data points

processing techniques in the data processing pipeline. In this section we will discuss about the each preprocessing steps, how they can be implemented to deal with streaming data and and their effects to the results.

Resampling

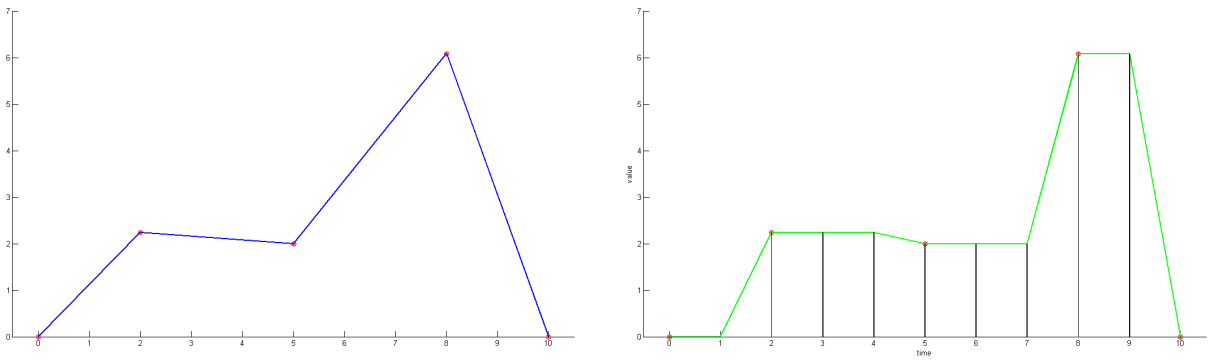
As we have described in section 2.1, data points of each time series are recorded at different rate, so the first thing to do is to re-sample them so that they are synchronized. This process makes sure that there is a new data point in each time series after every time tick t. In the mean of stream processing, the value of a time series at time tick t is the last value that the system received since the last time tick. In other words, the last value in time interval $(t-1, t]$. In case there is no data point arrived within $(t-1, t]$, we simply take the resampled data point at time $t-1$.

When the time series has higher frequency than the resample rate, it is called upsampling. An example of downsampling is showed in figure 4.6, in which, the original is a continuous function (figure 4.6a) and on the right side, figure 4.6b shows the its resampled time series with 11 data points.

An other case is when the number of data points in original time series is lower than the resampling rate, so it is called upsampling. The example in figure 4.6a is the original time series with 5 data points. Using last-known value to resample it we get a new time series with 11 data points (see figure 4.6b).

This resampling method is simple to implement and it does not require much computational resource and for each time series it only has to remember the last known value. The complexity and storage requirement are both O(n), where n is the number of time series.

The disadvantage of this method is that information is getting lost by resampling. However, after resampling, we can ignore the time stamp and do not have to deal missing value. Furthermore, the representation of time series data after resampling is matched to the input form of SPIRIT.



(a) Original function **(b)** Resampled data points

Figure 4.6.: Up-resampling from a function with 5 data points to 11 data points, using last known value if value for current time tick is missing

Moving Average Smoothing

As the second preprocessing step, triangular moving average is applied on each time series to remove the temporary trends or noise and make the main trends clearer to notice.

Using moving average does not require much resource since it just compute the average value of the recent history data point. With window size of w, we have to store the last w data points for each time series (w is usually smaller than 100).

Standardization

As we have showed in section 4.2, before feeding data to SPIRIT, all time series should be rescaled so that they have the same amount of energy. The z-score standardization which is used in section 4.2 is an appropriate approach. However, this method requires that the mean and variance of each time series are known and the application that we aim to build must deal with streaming data. It means that the true mean and variance of each time series are unknown. So what we can do is to estimate the mean and variance from the history data points and use them as true mean and variance to standardize the new data point. Computing the mean and variance from history data point can be implemented very efficiently in an incremental manner. Since the standard deviation of a time series could be formulated as:

$$\begin{aligned} Var(X) &= E\left[(X - E[X])^2\right] \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

so the only two variables that need to be stored are the sum of history values (sum) and the sum of squared history values ($square\_sum$). For an incoming value x_t at time t, the estimated mean and variance are updated as follow:

$$sum = sum + x_t$$
$$square\_sum = square\_sum + {x_t}^2$$
$$estimated\_mean = sum/t$$
$$estimated\_var = square\_sum/t - estimated\_mean^2$$

There are other method for incrementally estimating mean and variance which are claimed to be more robust[29, 30]. However all methods provide the same empirical result when applying on our dataset, the mean and variance converge after few data points.

A drawback of using incrementally updated mean and variance for z-score standardization is that it cannot deal with constant time series or time series which begins as a constant. As long as the values are constant, the estimated variance will be zero and z-score standardization cannot be applied (division by zero). This observation is important because such time series also exist on the data set from Solenix. A simple solution for this problem is to use a constant instead of z-score standardization when the variance is zero. With that, the standardized value of time series x at time t with the estimated variance (est_var) and estimated mean (est_mean) is computed as follow:

```
if (est_var < eps)
  standardized_x(t) = 0;
else
  standardized_x(t) = (x(t) - est_mean)/sqrt(est_var);
end
```

Listing 4.1: Standardization with estimated mean and variance

Results of Incremental Standardization

The results of incremental standardization will eventually converge to the normal standardization. Because as the time t increases, the number of sample data points also increases and therefore incremental mean and variance get closer to the true mean and variance.

So the question is how good is the convergence of incremental standardization and how good the convergence need to be. The second question is that how quick is the convergence process.

To answer the first question, we apply the incremental standardization to 100 time series from Solenix data set. In these time series, ratio between the greatest and smallest variance is more than 600 (see histogram a of Figure 4.7). Since we are estimating the variance and mean from the past data points to standardize the current data point, the variances of time series after standardization are still not equal to one but they are distributed in a small area around one(see histogram b of Figure 4.7). The distribution of variances is much narrower than in original time series. In fact, the greatest ratio between variances is now about 10.

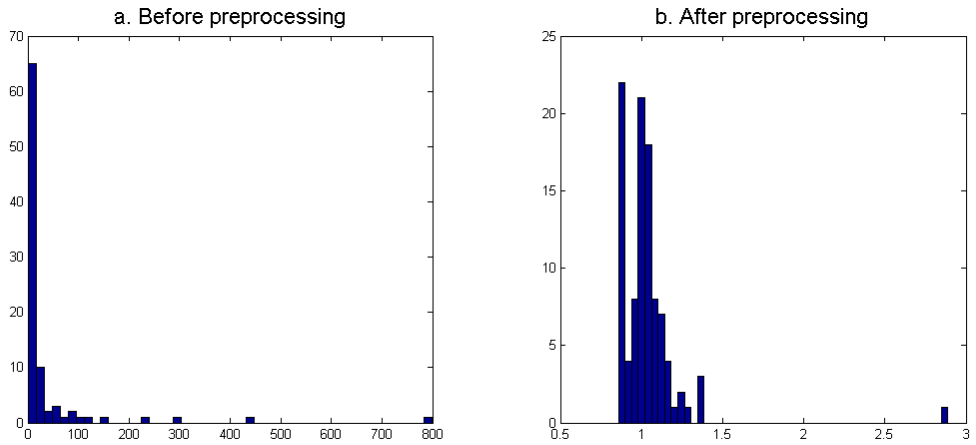


Figure 4.7.: Histogram of energy distribution before and after preprocessing

We are doing standardization because of the huge different in time series scales (one time series could have more than 600 times the energy of the others). The practical result shows that rescaling is not importance if the difference between variances is small. We set apply SPIRIT on 100 time series, which are rescaled with incremental standardization. By comparing the rescaled time series with its reconstruction we will see that SPIRIT is able to reconstruct well the time series with smallest variance (see Figure 4.8).

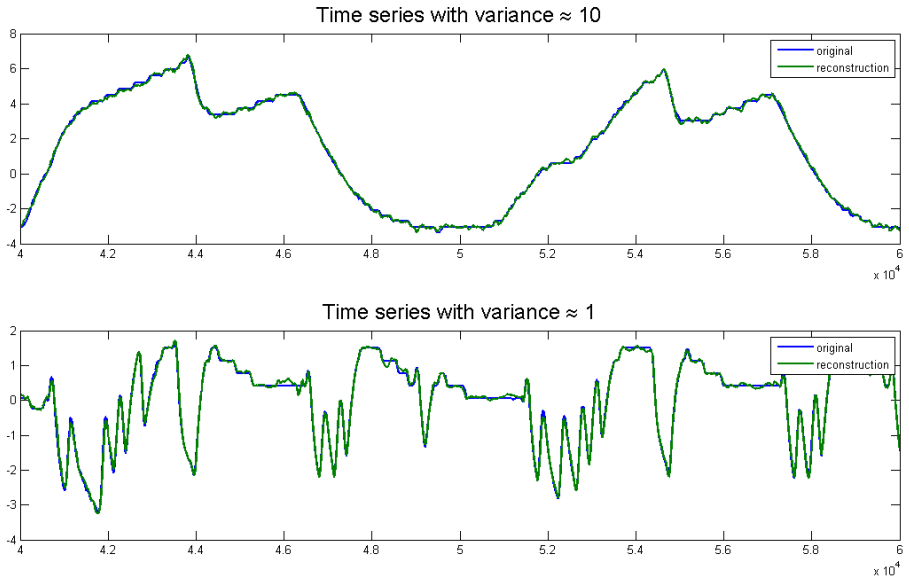


Figure 4.8.: The reconstruction of the lowest energy versus greatest energy time series after applying standardization. The maximal energy time series in the used data set is 10 times bigger than this time series

Since most of time series in Solenix dataset is periodic (its shape repeats itself after one period), the incrementally standardized time series gets close to the normal standardized time series after a period (see Figure 4.9a).

There are also some extreme case in the data set where the first half is constant see Figure 4.9b). At $t \approx 30000$, the time series starts to changes, therefore, result of the standardization process deviates from desired time series.

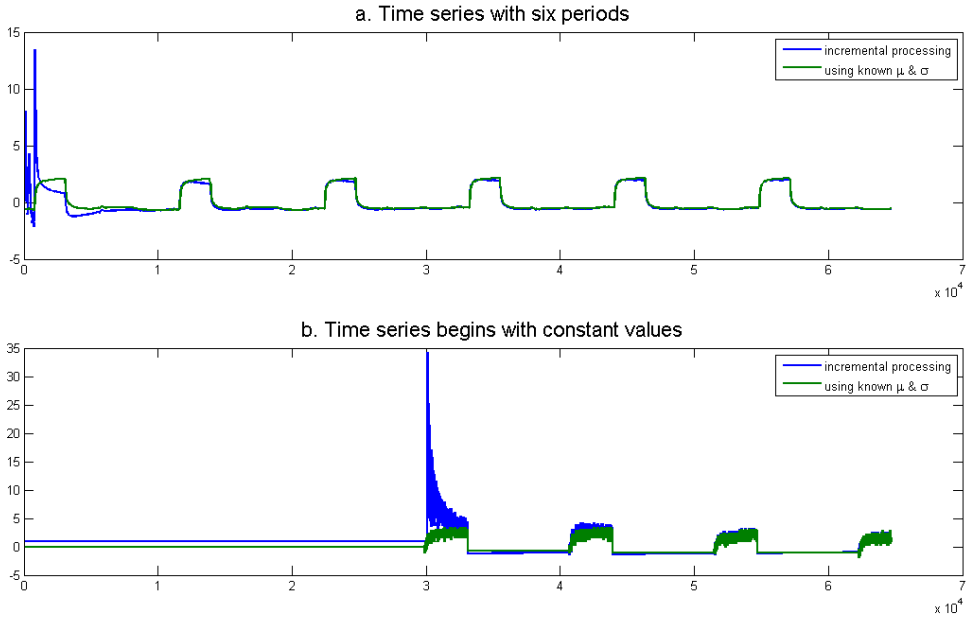


Figure 4.9.: Comparing results of incremental standardization with z-score standardization with pre-computed mean and variance

4.3.2 Putting data to SPIRIT

On this phase, we simply put the preprocessed data streams into SPIRIT. The following steps will be applied for each new data point x_t:

1. Project x_t to each principal direction, update the weight matrix W based on the error that the old weight matrix make with x_t
2. Orthonormalising the weight matrix by using Gram-Schmidt process.
3. Update the energy of input streams and of the reconstructed streams. Adapt the number of hidden variables (increase/decrease or keep)

As results of this step we will get hidden variables - y_t , reconstructed data stream - $\tilde{x_t}$ and a weight matrix - W. These information is passed to the third phase for finding outliers.

4.3.3 Finding Outliers

In this last step of the processing pipeline, we make use of information provided by SPIRIT to find out which time series contains contains outliers. We will describe how to interpret this information and make use of it for tracking the break of correlation in each individual time series and therewith find the actual time series causing outliers.

Reconstruction Error

The reconstruction error of a time series contains information about how well can it be reconstructed from the hidden variables. For detecting when the reconstruction error indicates that there is an outlier we need to specify a threshold. Whenever the error is higher than this threshold it can be said there are changes in the data streams so that the old weight vector can no longer compress the input streams into hidden variables without losing significant information. Finding an appropriate threshold can be done in three different ways:

- Static fixed threshold
- Dynamic threshold
- A mixed version of static and dynamic threshold

Fixed Threshold

Using a fixed threshold is the easiest way to finding outlier. The advantage of this method is that no computation is needed except comparing current error with the predefined threshold. Because input time series are already standardized in the pre-processing step which mean their variances are expected to stay close to 1. Therefore one can think about using the same fixed threshold for all data stream. Applying on the data set from Solenix with a threshold from 0.7 to 1 can discover very interesting unusual data points. Note that this window for threshold (form 0.7 to 1) is drawn from empirical results where the standard deviation of the reconstruction error is about x. For other dataset, a different threshold may be used if the reconstruction error is bigger (or even smaller).

Dynamic Threshold

The idea of dynamic thresholding is to determine the error threshold based on the mean and standard deviation of the errors. Since the standard deviation of reconstruction error is different for each data stream, we can make use of the incremental process (section 4.3.1) to continuously updating the variance and then the threshold.

Assume that error is a normal distribution with mean μ and standard deviation σ. A traditional approach for detecting outlying points in such data set is to use the three-sigma rule. For each data point x_i the quantity z_i is computed as:

$$z_i = \frac{x_i - \mu}{\sigma} \tag{4.5}$$

Any data point x_i with $\left|z_i\right| > 3$ lies outside the interval three standard deviation of the mean $[\mu - 3 * \sigma, \mu + 3 * \sigma]$. For a normal distribution, such a data point is considered to be potential outliers [31].

In statistical point of view, the interval $[\mu - 3 * \sigma, \mu + 3 * \sigma]$ contains the most data point of a normal distribution $N(\mu, \sigma^2)$. According to the three-sigma rule (or 68/95/99.7 rule), if x_i is drawn from $N(\mu, \sigma^2)$ then with the probability of 99.7

$$P(\mu - 3\sigma \leq x \leq \mu + 3\sigma) \approx 0.997 \tag{4.6}$$

So the probability that $\left|z_i\right| > 3$ is about 0.3% and it is significantly small enough to be suspected as an outlier.

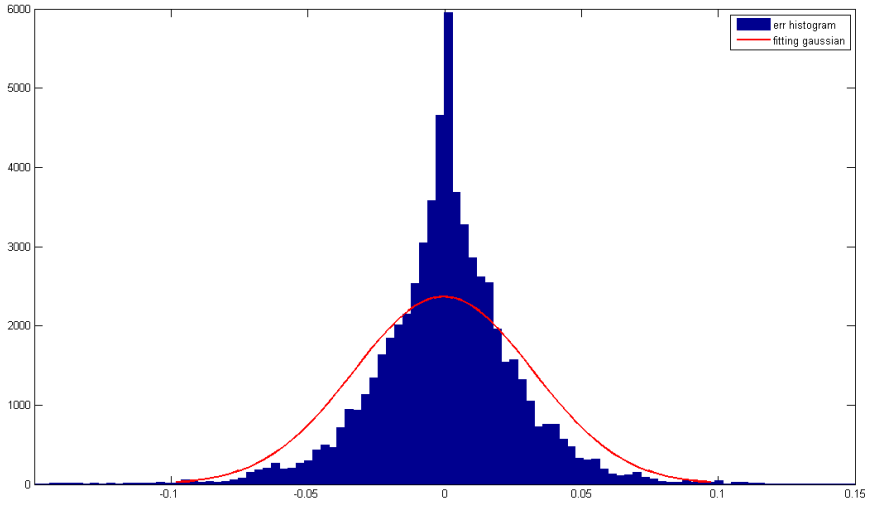


Figure 4.10.: Histogram of reconstruction error and a Gaussian distribution with the same mean and variance

However we should use the threshold of three standard deviations around the mean only if we are sure that the data is drawn from a normal distribution. If the error has a different distribution, this test may perform poorly when used for detecting outlying data points [2].

By creating a histogram of the reconstruction error and comparing it to the normal distribution (with same mean and standard deviation as the errors), we can see that distribution of reconstruction error is more slender with very high peak at the mean and two long tails (see Figure 4.10). So the reconstruction error is not really a Gaussian distribution. By counting for this particular reconstruction error (see Figure 4.10), the three-sigma interval of the mean contains about 98.65% the number of data points. To archive the same ratio as three-sigma interval has in a normal distribution (which is 99.7%), the window must be expanded to 6.5 times standard deviation around the mean:

$$P(error \in [\mu - 6.5\sigma, \mu + 6.5\sigma]) = 99.71\%$$

In this frame work, we use α-sigma interval for detecting outlier where α is configurable and should be in range of $[6, 10]$. We will discuss about how to tune this parameter in Section 4.5.

In Figure 4.11 we can see a illustration of using dynamic thresholds on the reconstruction error for detecting outliers. At $t = 46000$ and $t = 57000$, the reconstruction error jumps out of the lower- and upper-bound. Two points are potential outliers.

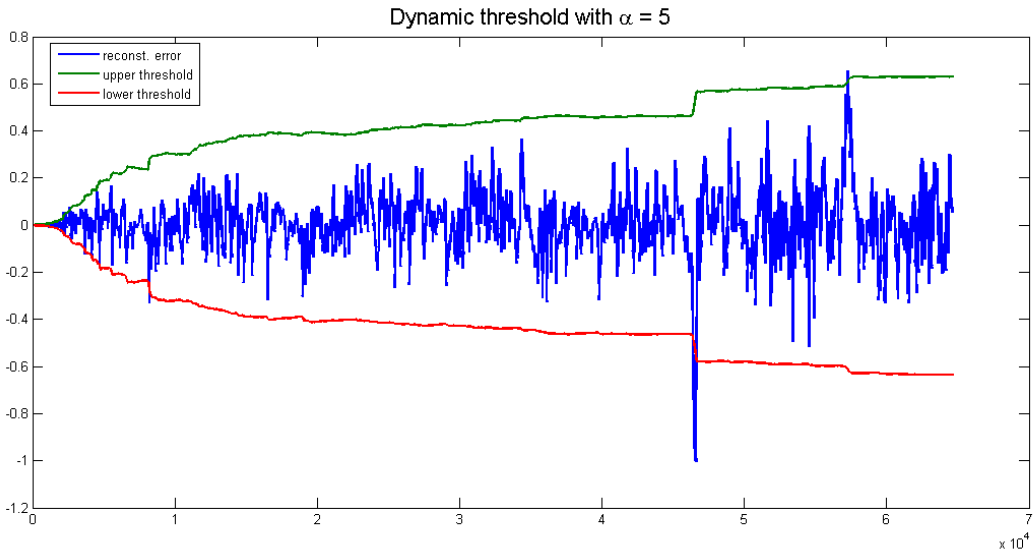


Figure 4.11.: Reconstruction error and dynamic thresholds

Mixed Threshold

Dynamic threshold of a time series can be very small if the past data points are really well reconstructed. In this case a small spike in reconstruction error could be greater than the dynamic threshold and therefore it will trigger the outlier alarm. For example a reconstruction error of 0.1 is relatively small in compare to other time series but it is still considered as outlier if the history errors are even smaller.

We want to encounter that problem by combining fixed threshold (Th_{fixed}) with dynamic threshold ($Th_{dynamic}$), Whereas Th_{fixed} is set to a small value and used as the lower-bound for $Th_{dynamic}$.

A data point at time t of i-th time series is considered as an outlier if its reconstruction error is greater than both thresholds:

$$e_{i,t} \geq max(Th_{fixed}, Th_{dynamic}) \tag{4.7}$$

Number of Output Time Series

Since the number of hidden variables is adjusted on the fly to keep the ratio of energy of hidden variables and original time series in a sufficient level. When outliers occur in one or more time series at time t, the correlations are broken and the energy ratio may drop out of the accepted range. It will make the algorithm increase number of hidden variables.

So in general, the number of hidden variables also provides information about outliers in all time series. However, from this information we cannot find out which time series actually contains outliers.

4.4 Explaining The Outliers

After found the outliers, one more thing that we could do is to explain it. This can be done by finding the correlated time series and showing them together with the one contains outlier. We will then see the correlation in the past data points and how it breaks at the current data points.

As output of SPIRIT, at time tick t we have k output time series as well as a matrix of weight vectors $W \in R^{n \times k}$ where $w_{i,j}$ $(1 \leq i \leq n, 1 \leq j \leq k)$ tells us how much the reconstruction of i-th data stream depends on the j-th hidden variable. So each row of the weights matrix W contains the information of how to reconstruct the a input data stream from k hidden variables. The j-th row $W_{j,\_}$ in W corresponds to j-th time series.

$$x = W * y \tag{4.8}$$

$$x_i = W_{i,\_} * y \tag{4.9}$$

$$= W_{i,1} * y_1 + W_{i,2} * y_2 + \ldots + W_{i,k} * y_k \tag{4.10}$$

We can prove that if two time series x_m and x_n are linearly correlated, then $W_{m,\_}$ and $W_{n,\_}$ are parallel (they either have same or opposite direction). Because of the linear correlation between x_n and x_m, we can represent one as a linear function of the other with some noise added.

$$x_m = \alpha x_n + \beta + \epsilon \tag{4.11}$$

By assuming that two time series is perfectly correlated so we can get rid of the noise term ($\epsilon = 0$). Furthermore, because x_m and x_n are already standardized to zero-mean, the constant β is also zero. The hidden variables y_i is linearly uncorrelated therefore equation 4.11 is equal to

$$W_{m,j} = \alpha W_{n,j}, 1 \leq j \leq k \tag{4.12}$$

With that we have proven that if x_m is correlated with x_n then $W_{m,\_}$ is parallel to $W_{n,\_}$. However, in practice, time series are not perfectly correlated thus the row vectors are not truly parallel, but the angle between them should be close to 0 or 180 grad. An example of three row vectors of correlated time series are show in figure 4.12. Using this observation, we can find the correlated time series by computing the angle between their weight vectors $\theta = \angle(\mathbf{W_{m,\_}}, \mathbf{W_{n,\_}})$.

$$cos(\theta) = \frac{W_{m,\_} \times W_{n,\_}}{\left\|W_{m,\_}\right\| \times \left\|W_{n,\_}\right\|} \tag{4.13}$$

The value of $cos(\theta)$ lies in the range $[-1, 1]$, where 0 means two vector are perpendicular and two time series are uncorrelated, 1 and -1 means they are parallel and two time series are correlated. The closer it gets to 1 (or -1) the more correlated two time series are.

Finding the correlated time series with the one that contains outliers is useful information for explaining the outliers. Since the outliers imply that the correlation is going to be broken, showing the correlated time series will point out which correlation is broken.

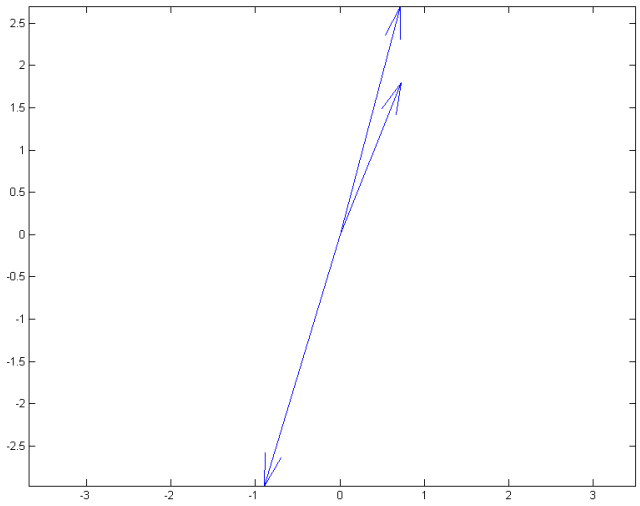


Figure 4.12.: Row vectors of three correlated time series. Although the row vectors are in high dimension, the plot only shows the 2-D hyperplane that contains these 3 vectors

4.5 Parameters Configuration

As mentioned in the previous parts, there are some parameters that can be configured before putting the system to work. In this section we will discuss about the meaning of those parameters and how they can be changed to adapt to a new dataset.

Exponential Forgetting Factor

The first parameter that is configurable is the exponential forgetting factor λ. It is used in SPIRIT to specify how a data point in the history will affect the update process of SPIRIT. The value range of λ is [0,1], where 0 means not taking the history data point into account and 1 means using all history data points. Normally λ should be set close to 1 (typically from 0.96 to 0.98[32]) so that the algorithm can deal with trends drifting. However in our data set, the time series have very high sample rate and mostly periodic (each time series consists of 6 periods) therefore we set the value for λ even closer to 1. For the most experiments we used $\lambda \geq 0.999$.

Energy Levels

The energy levels are used to tracking the number of hidden variables. There are two energy levels: the lower-bound f_E and the upper-bound F_E. Because f_E and F_E is the percentage of how good the variances that retain in hidden variables, it is actually not critical to change these parameters when we apply the pipeline to a new data set. However f_E should be greater than 0.95 whereas F_E must be smaller than 1. In this work we used $f_E = 0.97$ and $F_E = 0.99$.

Smoothed Error

For detecting outliers of a time series, the reconstruction error is a good starting point. It is the difference between the original time series and the reconstructed time series. A high reconstruction error at time t indicates that there is an potential outlier at time t. We also may want to detect it if when the reconstruction error is significant high at some consecutive time stamp rather than just a single time stamp. One way to deal it is to make use of moving average. it is interesting to examine the reconstruction error at not only a particular time stamp but also the,a the exponentially weighted error is used. It is computed from the reconstruction error with the following formal [33]:

$$expE_0 = e_0 \tag{4.14}$$

$$expE_t = e_t + \alpha * expE_{t-1} \tag{4.15}$$

$$= e_t + \alpha * e_{t-1} + \alpha^2 * e_{t-2} + ... + \alpha^{t-1} * e_1 + \alpha^t * e_0 \tag{4.16}$$

Where e_t is the reconstruction error and $expE_t$ is exponentially weighted error at time t. Despite the exponential weighted error at time t is computed from the entire history errors, the equation 4.15 is a nice update rule which results the same exponential weighted error and we just have to store the last computed error ($expE_{t-1}$).

The parameter α can be configured in range of $[0, 1)$. A special case when λ is equal to 0 then the exponentially weighted error is the reconstruction error ($expE_t = e_t$)

Outlier Thresholds

As we already mentioned in section 4.3.3, the dynamic thresholds are set proportional to the standard deviation of history errors : $\alpha \times \sigma$. The parameter α indicates expected deviation of reconstruction error as a window around the mean. The reconstruction error of a normal data point will most likely fall inside of this window. α can be interpreted as the sensitivity of the algorithm with outliers and noises.

This parameter can be adjusted on the fly by users. In case of too many suspicious data points are found but they are not actual outliers but noises, we should increase λ. The algorithm will then become less sensitive and do not recognize noise as outlier. However choosing a too high value for α may make the algorithm also ignore the actual outliers and therefore abnormal data points would remains undiscovered.

From the empirical results by applying on the full data set, we found that λ should be set in the range from 5 to 11.

5 Evaluation and Review Outliers

In this chapter we evaluate the performance of the proposed data processing pipeline on different data sets. The data set of each experiment and the set up parameters are described in detail. We will then review some of the outliers that are found by the data processing pipeline.

5.1 General set-up

As described in above sections, data processing pipeline are initiated with some dummy values (principal directions, estimated means and variances...). Eventually, these values will then converge but at the beginning, they may be varied and not stable. It means that the algorithm needs some time to find those parameters and during this time it should not be used to detect outliers.

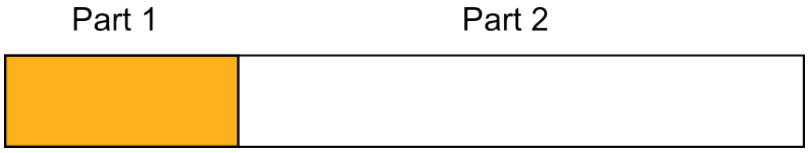


Figure 5.1.: Time series are divided into two parts. Detecting outliers are apply to the part 2 of the time series

In the following experiments, we divide each time series into two parts as show in Figure 5.1. We run the data processing pipeline on the whole time series but try to find outliers on the second part of the time series. The first part is meant for finding and converging principal directions and other parameters.

The policy of how to split the data set into two parts is set based on the characteristic of the data set itself. When the time series are periodic, their means and variances converge after about one period. As we have seen in section 2, the time series from Solenix usually consists of 6 periods. Therefore in the experiments with this data sets, the length of first part is set to 20000 data points, which will cover more than 1 period of each time series. A detail information about how each data set is split can be found in table 5.1.

Data set	Number of TS	Length	Length of Part 1
Generated data set	5	1000	300
Small Solenix data set	200	64650	20000
Full Solenix data set	1200	64650	20000

Table 5.1.: General information about data sets

Parameter	Value	Description
λ	0.99	Exponential forgetting factor
f_E, F_E	0.97,0.99	Range for excepted energy of projection time series in comparison to original time series
α	0.6	Exponential moving average factor is set to a small value because we want to find outlier in a small window

Table 5.2.: Parameters configuration for the generated data set

For detecting outliers, the dynamic threshold method is used with the factor of 10 times of standard deviation (σ). So a data point is considered as outlier if its reconstruction error is greater than $10 \times \sigma$.

The exponential forgetting factor needed for running SPIRIT is chosen according to section 4.5 and will be discussed in each experiment in following sections.

5.2 Results for generated data set

We have already seen the generated data set in section 4.1 as an illustration to show how the algorithm works. This data set consists of 5 time series and one outlier. The snippet of Matlab code for generating this data set can be found in Listing 5.1.

```
% generate a normal time series with Gaussian noise
data = sin((1:1000)*pi/100)' + randn(1000,1)/20;

% generate a time series with outlier
data_outlier = sin((1:1000)*pi/100)' + randn(1000,1)/20;
data_outlier(800:815) = 0;
```

Listing 5.1: Matlab code for generating data

The configurable parameters of this experiment are set to match the length of generated data set. A table of configurable parameters that were used in this experiment together with a short description:

Because the low sample rate in all time series (200 data points per period), the forgetting factors λ and exp. moving average factor α are smaller than in next experiments.

This data set is the optimal case where all time series are correlated. The outlier causes a high peak in the reconstruction error and is detected.

5.3 Results for a part Solenix's data set with artificial outliers

We set up the second with 200 time series of the data set from Solenix. In this experiment, we run the algorithm two time. On the first time on the original 200

time series. On the second time, before feeding data to the pipeline, we pick some time series that has a repeated pattern and modify them at some small intervals (called artificial outliers). The purpose of this experiment is to find out how the algorithm performs on a real data set and if it is capable of discovering the artificial outliers.

In one time series, an outlier is created by removing a peak at t = 45000. In figure 5.2, the plots on the left side show the original time series and its reconstruction error, on the right side are the modified time series and its reconstruction error. As

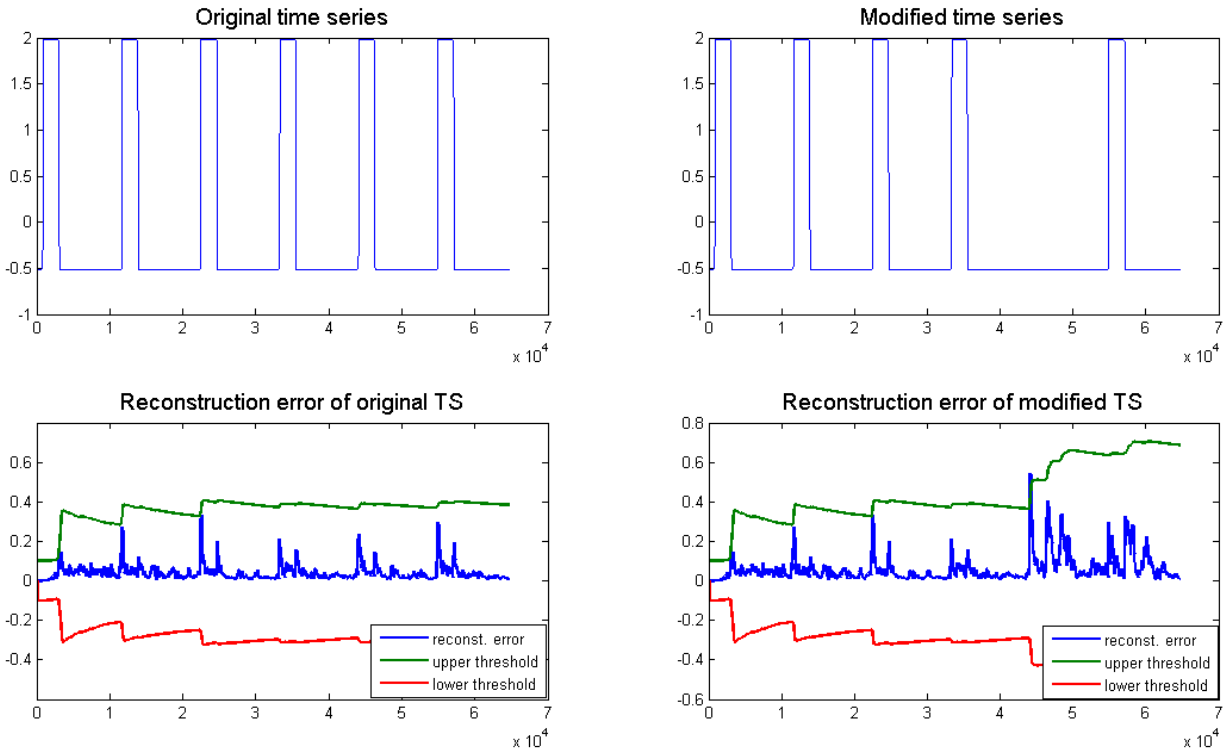


Figure 5.2.: Detecting the removed peak from original time series. Reconstruction error is significantly higher than at other data points

we can see, the reconstruction error of both time series before the artificial outlier are the same. At the beginning out the outlier interval, the reconstruction error of the modified time series is getting higher and jumps over the threshold. It indicates that the injected outlier is found.

In another time series, we create two artificial outliers by removing one peak at time tick $t \approx 43000$ and adding a new peak to another location $t \approx 25000$. In figure 5.3, the outliers are marked with red circles. Despite having visited many similar peaks in the past data point of this time series, the algorithm still make a high reconstruction error when it visits the peak that was created at time tick $t = 25000$. Similar to the previous artificial outlier, the missing peak at $t = 43000$ also results a high reconstruction error. Using dynamic threshold with factor of 10 times standard deviation, both outliers are detected.

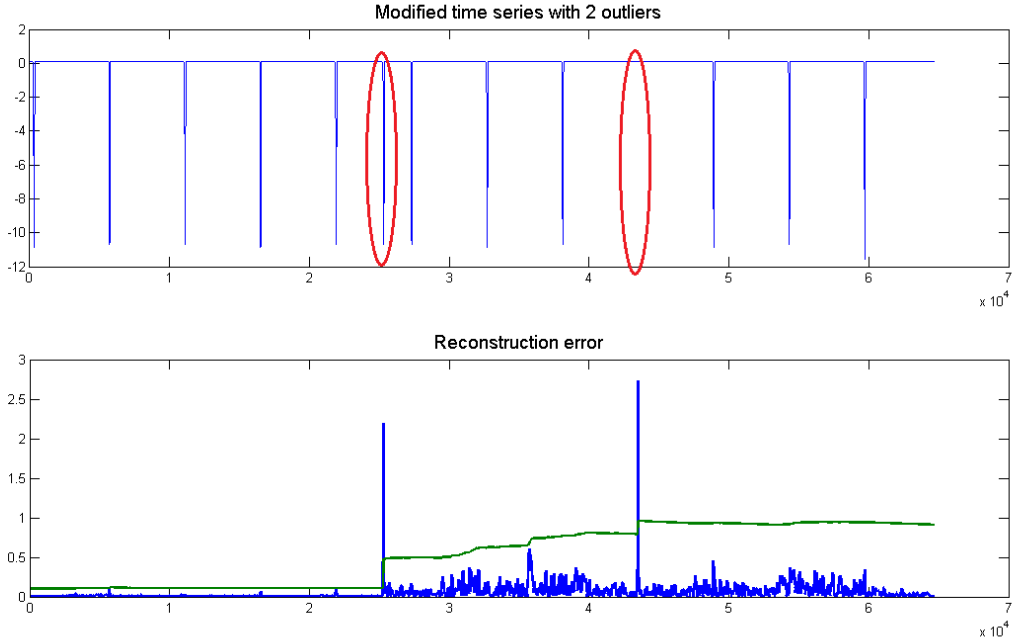


Figure 5.3.: Modified time series with two artificial outliers at $t = 25000$ and $t = 43000$. The outliers are marked with red circles

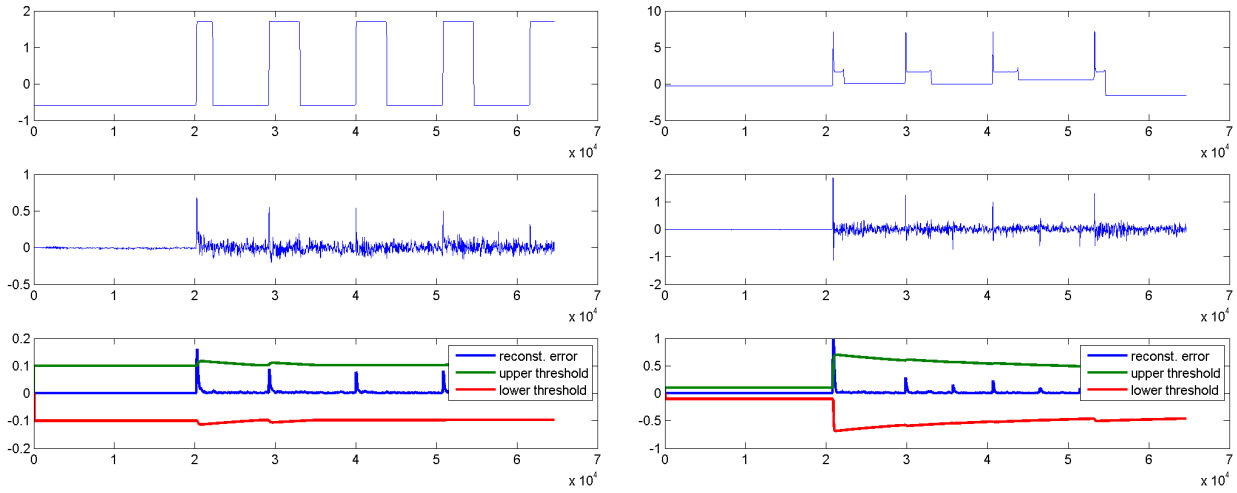
5.4 Results for full Solenix's data set

In this experiment, the big data set from Solenix is used with 1200 time series. The time series are resampled so that they are synchronized with time tick of 16 seconds (there is a new data point every 16 seconds). The data set was recorded in 12 days, so after resampling, each time series has 64800 data points and in total there are 77'760'000 data points.

The parameters configuration on this experiment are showed in table 5.3.

Parameter	Value	Description
λ	0.9997	λ was set very close to 1 because the sample rate is high and we want to remember the recent data points
$[f_E, F_E]$	$[0.97, 0.99]$	Energy thresholding. the energy of projections should lie between 97% and 99% of the input time series
α	0.9	Exponential moving average factor is greater than generated data set since we want to detect outlier in a large window

Table 5.3.: Parameters configuration for full data set



(a) time series 123 **(b)** time series 602

Figure 5.4.: Two from 17 time series that contain outliers at time tick 20000. The time series are placed in the first row. The second row are the reconstruction errors and the last row are exponential smoothed reconstruction error with outlier thresholds

5.4.1 Performance

The experiment on full data set runs in about 1 hour. In contrast to 12 days, in which the data set was recorded, performance of the pipeline is suitable for deal with real time streaming data.

5.4.2 Review Outliers

In following section we will review the outliers that were discovered during the runtime. Out of 1200, the algorithm found 225 time series which contains outliers. Note that an outlier could also occur in multiple time series and one time series could contains multiple outliers.

The first outlier was found at the beginning of part 2 of the data set. It was detected in 17 time series simultaneously. These 17 time series have the same shape as they are almost constant in the first part of the data set and start to change together at time tick around $t \approx 20000$ (see Figure 5.4). At this point, the algorithm cannot reconstructed well these time series and we can observe a high peak in the reconstruction error. However it learns these changes by adapting its principal directions. So when the same pattern occurs in some latter points, they are not considered as outliers.

Another similar outlier was found in 16 time series at time tick around 30000. These time series also start with constant values and only begin to change together at $t = 30000$ (see Figure 5.5). However, this type of outliers are quite easy to be captured, a simple extreme value analysis would also detect these abnormal behaviors.

As we mentioned in the implementation that, along with detecting outliers, the data pipeline can also explain them. In Figure 5.6, we show another outlier that was

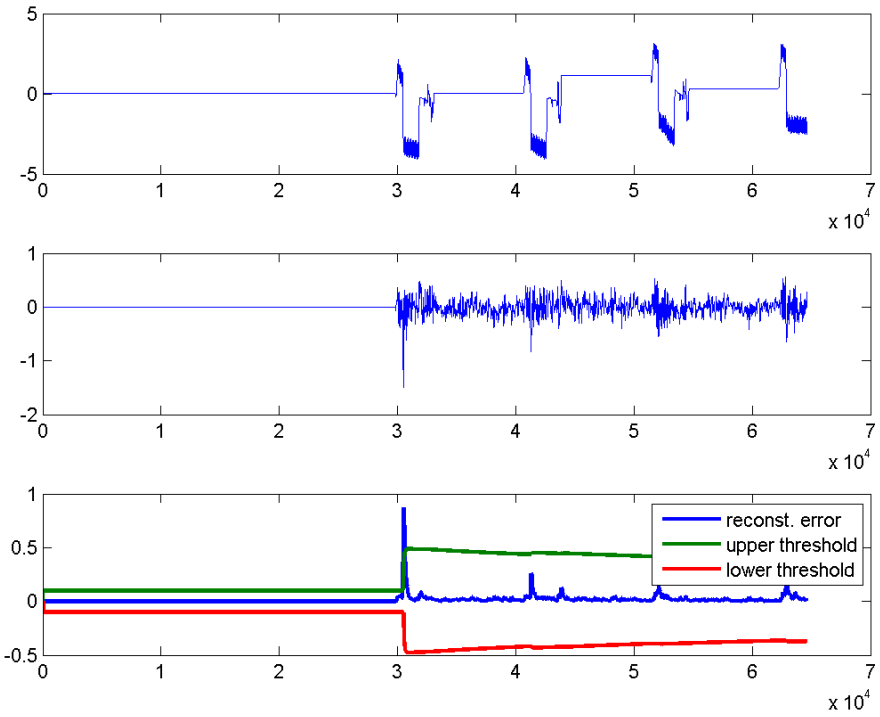


Figure 5.5.: Time series 561. An outlier is detected at $t \approx 30000$ together with 15 other time series. Only one outlier was found as the time series firstly changes. In the next appearance of the similar pattern, no outlier was indicated

found on the data set (the outlier is marked with red circle). Two correlated time series which contain no outliers are also outputed by the pipeline as an explanation to the outlier in first time series. With that we can see more clearly that there is a correlation in the past data points and how this correlation is broken. .

Furthermore, the processing pipeline also detects time series that contain multiple outliers. An example of those is showed in figure 5.7. Here we can see that the correlation is broken at several areas (marked with red circles), they cause high peaks in the reconstruction error. in this experiment, the dynamic thresholds are used for detecting the peaks is set to 10 times standard deviation, however the algorithm is also able to detect those anomalies with lower thresholds. While changing the thresholds from 5 to 11, the same outliers are still detected.

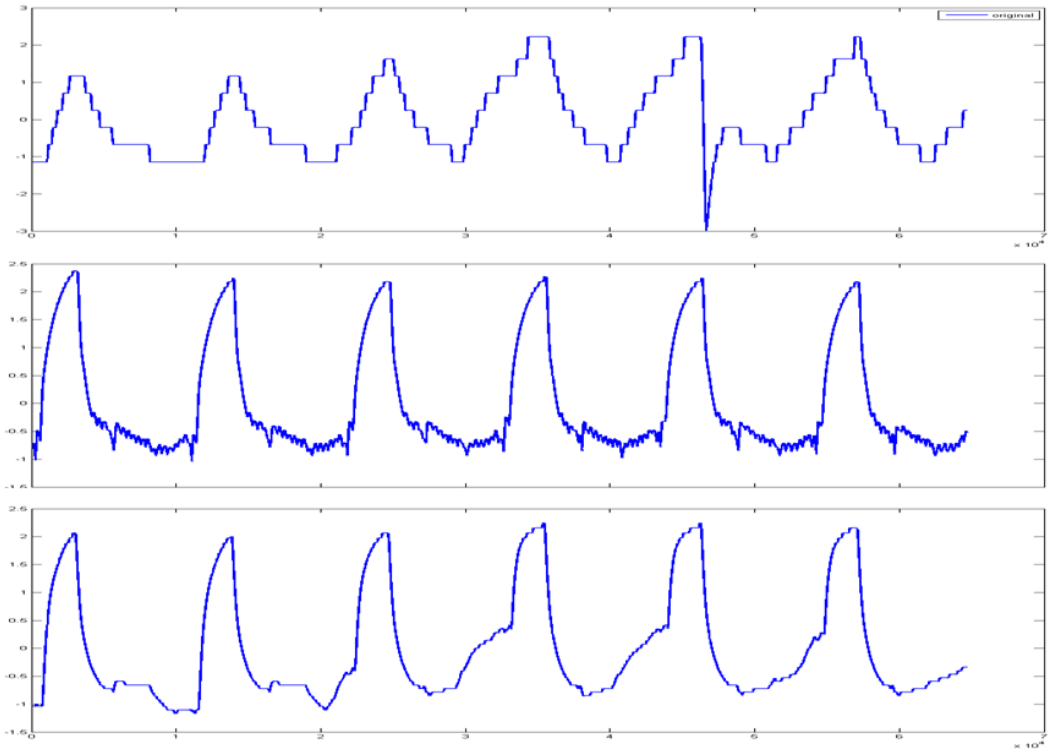


Figure 5.6.: Time series with outlier and its correlated time series. The first time series contains an outlier at $t \approx 46000$. Two other time series correlates with the the first one at all data points but the outliers

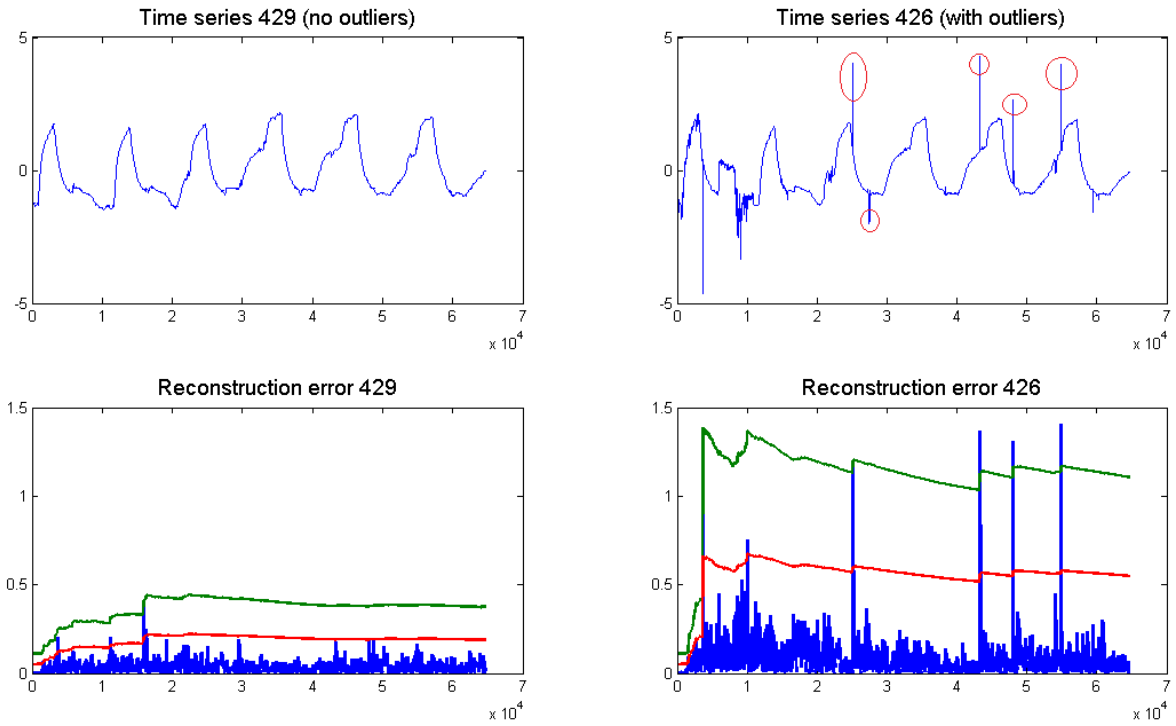


Figure 5.7.: Time series with multiple outliers (left) in comparison to its correlated time series which contains no outlier (right). The blue lines in the reconstruction error diagrams show dynamic threshold of 11 times of the standard deviation. The red lines show the dynamic threshold of 5 time of the standard deviation

6 Conclusion and Future Work

6.1 Conclusion

The goal of this work is to develop an application for helping domain experts in monitoring the daily routines of artificial satellites. In order to identify potential problems and report them to experts as early as possible, it is required that the algorithm is able to process the data stream in real time. Also the algorithm should be able to deal with high number of data streams as it can be up to 40,000 from a satellite.

By the observation that time series are highly correlated, we focus on finding the breaks in these correlations which may indicate new behaviors or changes/malfunction of on-board devices.

We have examined and compared different methods and then proposed a data processing pipeline for detecting outliers. In the pipeline we make use of SPIRIT for compressing the time series into main trends. The breaks of correlation in time series are found by scanning the reconstruction error. To summarize, the data processing pipeline consists of three following steps:

- Pre-processing
- SPIRIT
- Finding outliers

In additional, the algorithm also tries to find an explanation for each found outlier. This is done by outputting not only the outliers but also theirs correlated time series. Furthermore, we have also proven that the linear correlation of two time series is captured in the weight vectors (output from SPIRIT). Therefore, the process of finding correlated time series is implemented efficiently by comparing the their weight vectors.

To evaluate the proposed method above, we carried out two experiments. The first experiment was to work with a small set of time series, which are extracted from the given data set. Since there was no sample of outliers available, so we create artificial outliers by modifying some time series. The data pipeline worked well in small size data set and all the modified time series got detected. In the second experiment, we run on the whole data set and verified that the data pipeline is able to cope with real time data. The outliers found in this experiment were afterwards reviewed and explained.

Finally, we have provided a guideline for determining needed parameters of the algorithm. Based on that, users can configure the pipeline to match with a new data set when it has different characteristics.

6.2 Future Works

One of the disadvantages of the proposed data pipeline is that it can only capture the linear correlation between time series. We have also discussed about the ability of deep autoencoders for dealing with non-linear correlation. It is interesting to know if there is any non-linear correlation between time series and if deep autoencoders could be applied to improve the results of PCA in the time series case.

In this work, we have focused on finding the breaks of correlation. There is an another possible type of outliers, which may occur when the correlated time series suddenly change together and form new unsual shapes ,however, the correlations are preserved, this kind of outlier is not reflected in the reconstruction error and therefore cannot be detected by proposed method. If there are such outliers in the data set, the changes in all correlated time series will be reflected in some hidden variables and therefore could on account that be track down by examining the hidden variables. This is feasible since the number of hidden variables are much smaller than the number of original time series.

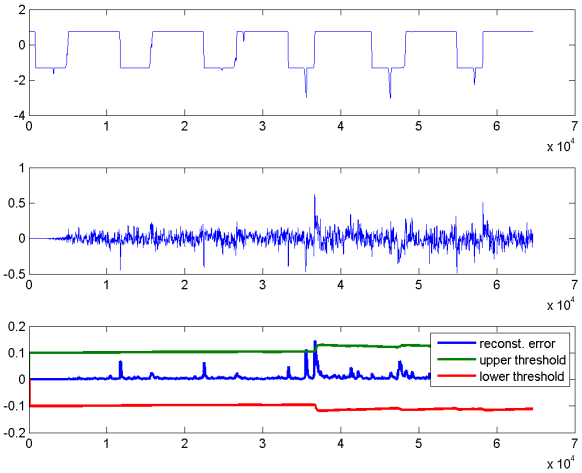
A Table of Symbols

List of symbols and descriptions which are used in this thesis.

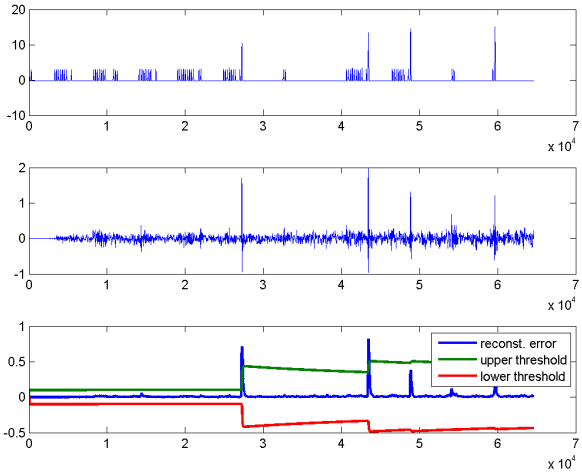
Symbol	Description
x	Input time series
$\tilde{x}$	Reconstructed time
y	Projection of x on principal directions
W	Weight matrix
$W_{\_,j}$	weight vector for projecting j-th hidden variables (j-th principal directions)
$W_{i,\_}$	weight vector for reconstructing i-th time series
$\alpha, \beta, \lambda, \theta, \ldots$	configurable parameters or constants
t, τ	Time ticks
E_t	Energy of all time series at time t
$\tilde{E}_t$	Energy of all reconstructed time series at time t
f_E, F_E	Lower- and Upper-Energy thresholds
$e_{i,t}$	Reconstruction error of i-th time series at time tick t
$expE_i$	Exponentially smoothed reconstruction error
d_i	Estimated energy of data along i-th principal component

B Time Series with Outliers

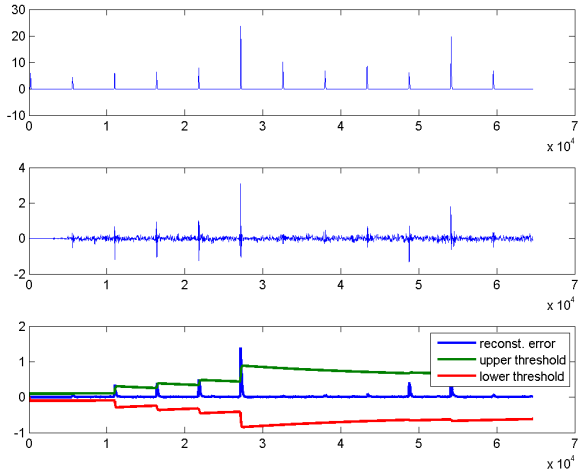
In this appendix we show some more time series which contains outliers and are found from the full data set. In each figure, from top to bottom: Original time series, reconstruction error, exponentially smoothed error with outlier thresholds.



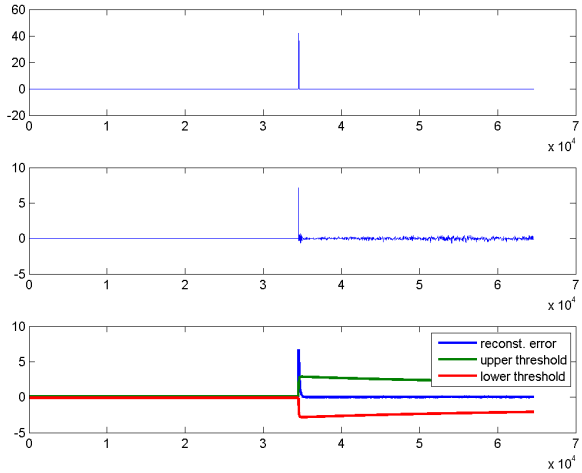
Outlier in time series number 502



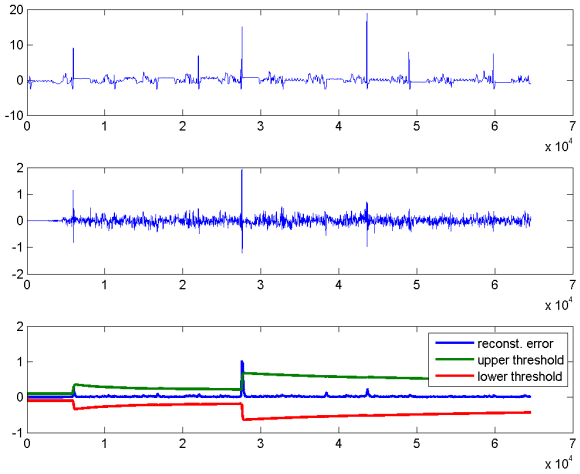
Outlier in time series number 645



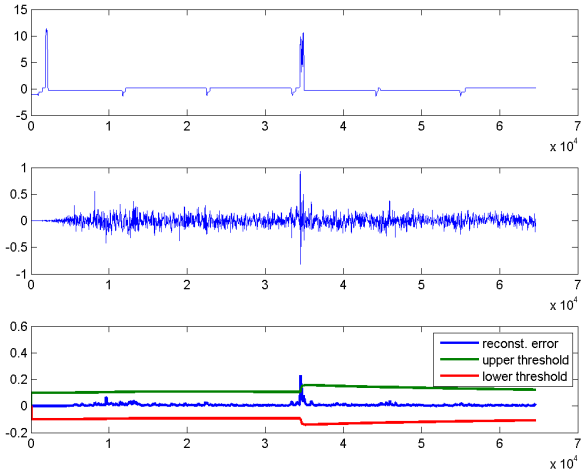
Outlier in time series number 864



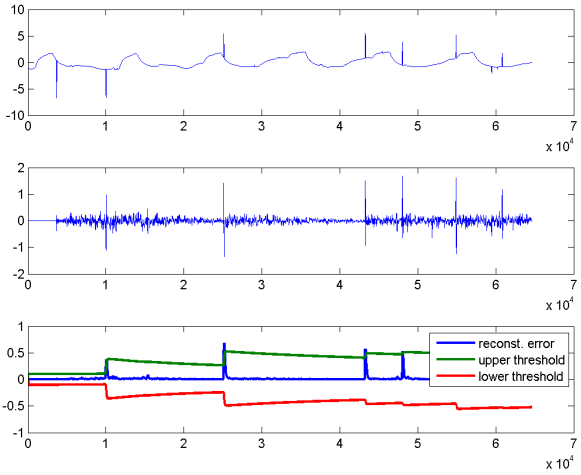
Outlier in time series number 940



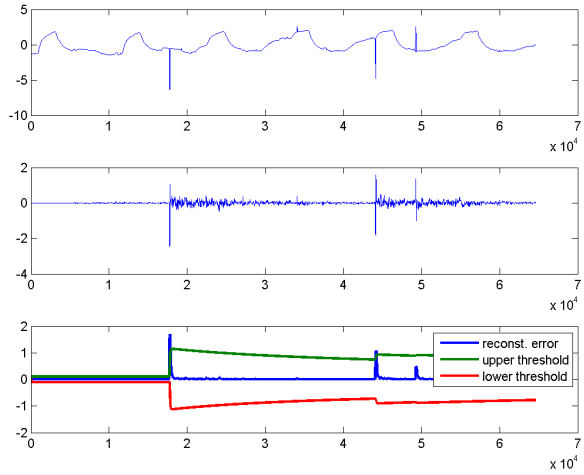
Outlier in time series number 1180



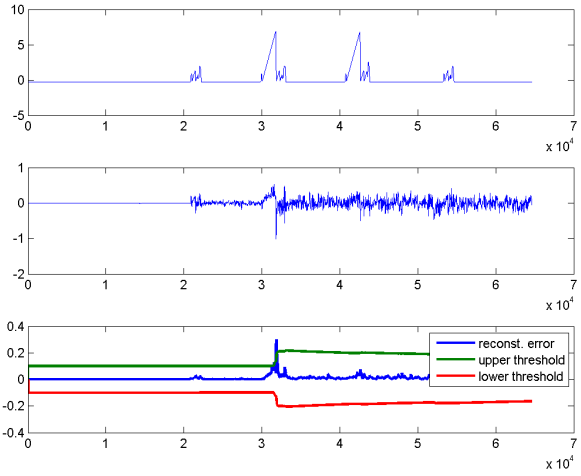
Outlier in time series number 41



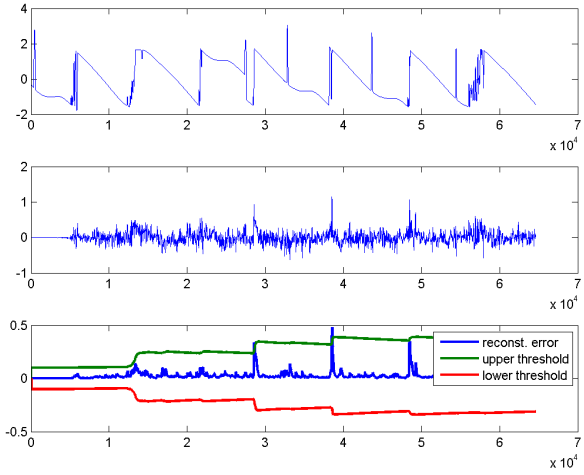
Outlier in time series number 227



Outlier in time series number 239



Outlier in time series number 634



Outlier in time series number 937

List of Figures

List of Tables

Bibliography

[1] D. M. Hawkins, *Identification of outliers*, vol. 11. London: Springer, 1980.

[2] C. C. Aggarwal, *Outlier Analysis*. New York, NY: Springer New York, 2013.

[3] M. Gupta, Hyderabad, J. Gao, C. Aggarwal, and J. Han, "Outlier Detection for Temporal Data : A Survey," vol. 26, no. 9, pp. 2250 – 2267, 2014.

[4] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," *Proceedings of the 32nd international conference on Very large data bases*, pp. 187–198, 2006.

[5] Y. Zhang, N. Meratnia, and P. Havinga, "Outlier detection techniques for wireless sensor networks: A survey," *Communications Surveys & . . .*, vol. 12, no. 2, pp. 159–170, 2010.

[6] S. Subramaniam and T. Palpanas, "Online outlier detection in sensor data using non-parametric models," *. . . on Very large data . . .*, 2006.

[7] C. Aggarwal, "On Abnormality Detection in Spuriously Populated Data Streams.," *SDM*, pp. 80–91, 2005.

[8] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, p. 811, 2007.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," vol. 12, pp. 2825–2830, Jan. 2012.

[10] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on Nuclear Science*, vol. 44, pp. 1464–1468, June 1997.

[11] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," no. 1, pp. 1–16, 2003.

[12] D. T. Larose, *Discovering knowledge in data: an introduction to data mining*. Hoboken, NJ, USA: John Wiley & Sons, Inc., June 2014.

[13] G. Forman, "BNS feature scaling: an improved representation over tf-idf for svm text classification," *Proceedings of the 17th ACM conference on . . .*, pp. 263–270, 2008.

[14] S. Aksoy and R. Haralick, "Feature Normalization and Likelihood-based Similarity Measures for Image Retrieval," *Pattern Recognition Letters*, no. October 2000, 2001.

[15] A. Papoulis and S. U. Pillai, *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education, 4 ed., 2002.

[16] J. Hunter, "The exponentially weighted moving average.," *J. QUALITY TECHNOL.*, vol. 18, no. 4, 1986.

[17] C. M. Bishop, *Pattern recognition and machine learning*. New York: Springer, 2006.

[18] S. Papadimitriou, J. Sun, and C. Faloutsos, "Streaming pattern discovery in multiple time-series," *Proceedings of the 31st . . .*, pp. 697–708, 2005.

[19] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, 1988.

[20] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[21] G. E. Hinton and S. Z. Richard, "Autoencoders, minimum description length, and Helmholtz free energy," *Advances in neural information processing systems*, 1994.

[22] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decomposition," *Biological cybernetics*, vol. 294, pp. 291–294, 1988.

[23] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks.," *Science (New York, N.Y.)*, vol. 313, pp. 504–7, July 2006.

[24] I. Assent, P. Kranen, C. Baldauf, and T. Seidl, "AnyOut: Anytime Outlier Detection on Streaming Data," in *Database Systems for Advanced Applications*, vol. 7238, pp. 228–242, Springer Berlin Heidelberg, 2012.

[25] P. Kranen, I. Assent, C. Baldauf, and T. Seidl, "Self-Adaptive Anytime Stream Clustering," *2009 Ninth IEEE International Conference on Data Mining*, pp. 249–258, Dec. 2009.

[26] K. Yamanishi and J. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002.

[27] K. Yamanishi, J.-i. Takeuchi, G. Williams, and P. Milne, "On-Line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms," *Data Mining and Knowledge Discovery*, vol. 8, pp. 275–300, May 2004.

[28] S. Thompson and J. M. VanBriesen, "Chlorine data: generated using EPANET http://www.cs.cmu.edu/afs/cs/project/spirit-1/www/#Download."

[29] R. F. Ling, "Comparison of Several Algorithms for Computing Sample Means and Variances," *Journal of the American Statistical Association*, vol. 69, pp. 859–866, Dec. 1974.

[30] T. F. Chan, G. H. Golub, and R. J. Leveque, “Algorithms for Computing the Sample Variance : Analysis and Recommendations,” vol. 37, no. 3, pp. 242–247, 2014.

[31] Maronna, Ricardo, Douglas Martin and V. Yohai, *Robust statistics*. Chichester: John Wiley & Sons, 2006.

[32] S. S. Haykin, *Adaptive filter theory*. Pearson Education India, 2008.

[33] C. C. Holt, “Forecasting seasonals and trends by exponentially weighted moving averages,” *International Journal of Forecasting*, vol. 20, pp. 5–10, Jan. 2004.