LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR STATISTIK

Analysis of Anomaly Detection Methods for Streaming Data

Bachelor Thesis

Wissenschaftliche Arbeit zu Erlangung des akademischen Grades
Bachelor of Science

Hyeyoung Park

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
INSTITUT FÜR STATISTIK

Analysis of Anomaly Detection Methods for Streaming Data

Bachelor Thesis

Wissenschaftliche Arbeit zu Erlangung des akademischen Grades
Bachelor of Science

Author Hyeyoung Park
Supervisor Prof. Dr. Christian Heumann
Date München, April 23, 2019

I confirm that this bachelor thesis is my own work and I have documented all resources and material used.

München, 23rd April 2019

Hyeyoung Park

"We are all outliers, depending on the context."

Abstract

This paper gives an overview over several techniques of anomaly detection in time series data, which can be utilized e.g. for real time sensor data from IoT appliances. The following anomaly detection methods are chosen depending on the type of an anomaly. For contextual anomalies, the method using ARIMA models is adopted. The HOT SAX algorithm detects discords, which can be considered as collective anomalies. In order to implement the two models, simulation data sets are generated with customized functions. Afterwards, the anomaly detection techniques are evaluated via precision, recall, and F_1.

Contents

List of Figures

List of Tables

1 Introduction

Nowadays, from portable devices to transportation systems, it is hard to find a field which does not utilize the Internet. Major companies are trying to put the Internet into almost every product, even home appliances like fridges, lamps, and music boxes. Along with it, the rapid progress of WiFi-connection technology is accelerating the Internet of Things (IoT) era. All devices connected with the Internet are called "smart", however they seem not as smart as we think, since they are easily exposed to security breaches. As an notorious example, *Dyn cyber attack in 2016* targeted major services like Amazon, Paypal, Twitter and GitHub, causing them to be not available for several hours. Such incident could result in significant financial losses. This attack was realized by using a *botnet*, which consisted mainly of IoT devices.[13] With *Network-based Intrusion Detection Systems* (NIDS) one can identify malicious behavior through anomaly detection and prevent such attacks. Because of this and many other applications in business and research, discovering anomalous instances needs to gain more attention. This paper will give an overview over several techniques of anomaly detection in time series data, which can be utilized e.g. for real time sensor data from IoT appliances. Afterwards, the anomaly detection methods will be simulated using generated time series data sets and subsequently evaluated.

1.1 Introduction to Anomaly Analysis

1.1.1 Types of Anomalies

Numerous researcher have already well defined outlier[1]. Instead of enumerating those definitions, this work will clarify the different types of anomalies, since the type of data and anomaly has considerable effects on the quality of anomaly detection methods.

Above all things, the terms *noise*, *outlier* and *anomaly* are used in different ways. According to Aggarwal [1, S.3], outlier covers abnormal data points as well as noise that slightly deviates from the normal data, but not interesting enough to be specially handled. Anomaly, on the other hand, indicates the data points that could be of interest due to the strong deviation from other data points. Anomalies can be considered as a special type of outlier. The present task of anomaly detection consists of making a distinction between noise and anomalies.

In *Outlier Analysis 2nd Edition* by Aggarwal [1] anomalies are divided into *contextual* and *collective anomalies*. If an individual data point looks not deviated from the rest of the other data points, but shows a different pattern as compared with the adjacent data instances, the data point can be defined as a *contextual anomaly*. In other words, each data point needs to be understood in the context of a given time series data, since each instance in a time series is correlated with the rest of the instances regardless of the extent. For example, a creditor has regularly used

[1] For example, "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism" by Hawkins [4]

his credit card. However, one day his credit card was used at dawn, when he is normally not suppose to be awake. Then the use of the credit card will be suspected as stolen, even thought the amount of the expense is not high enough to attract the attention of the bank. A *collective anomaly* is a collection of neighbouring data points in a data set, which are prone to show an abnormal trend. Those anomalies display a different shape in comparison to the regularly and frequently appearing patterns in time series. Here, an issue has arisen: can all different types of anomalies always be categorized by their type? The answer has been concluded by Chandola et al. [2, S.10]:

> It should be noted that while point anomalies can occur in any data set, collective anomalies can occur only in data sets in which data instances are related. In contrast, occurrence of contextual anomalies depends on the availability of context attributes in the data. A point anomaly or a collective anomaly can also be a contextual anomaly if analyzed with respect to a context.

Another approach for classifying the type of an anomaly separates outliers into *Additive Outlier* (AO), *Level Shift* (LS), *Temporary Change* (TC), *Innovational Outlier* (IO), and *Seasonal Level Shift* (SLS). This approach will be concretely discussed in the following chapter.
Note that this paper will mainly focus on contextual anomalies and collective anomalies. Thus, the following anomaly detection methods are selected in order to discover those types of anomalies.

1.1.2 Outline

There are two types of anomaly detection methods: supervised and unsupervised methods. Supervised methods can only be feasible, when anomalies are known in a given data set. But anomalies happen rarely in practice and it may be difficult to define given data points as true anomalies due to either lack of information or technique. Hence, unsupervised methods could be more applicable in reality and this paper will only handle unsupervised methods.
For the same reason as mentioned above, generated data sets will be used in this work instead of real data sets. The undeniable problem of real data sets is that those data sets do not always include certain types of outlier which we want to investigate. The rare occurrence of outliers can cause unreliable results. The following generated data sets will show a considerable amount of anomalies, which makes it easier to analyze the anomaly detection methods introduced in the following chapter.
After detecting the anomalies in the generated data, the next issue is analyzing which method is more effective and accurate to detect anomalies. The result of anomaly detection can be

described in two ways: [1, S.26]

- Each data point can be rated based on the level of *outlierness*.
- Normal data points and anomalies can be classified by binary flagging.

In this work anomalies will be flagged as 1 and otherwise as 0 in order to vectorize each data set. The vectorized data sets will be used to compute the metrics such as precision and recall, which are commonly adopted for the evaluation of statistic methods.

2 Data Generation for Simulation

Despite of the huge impact of outliers, abnormal instances can easily be ignored in practice because of their rareness. In this work, simulated data sets have been used for evaluating the anomaly detection methods, thus possible outliers can be effectively treated. Each data set emulates possible real time streaming data from reality. First of all, normal time series data sets, i.e. without anomalies, were generated. After that, several types of anomalies have been randomly planted. In the real world, what kind of data instances should be named as an anomaly, strongly depends on the user of a project. For instance, extreme values are one of the most primary concern in sales forecast. On the other hand, horrendous traffic jams in the middle of night should be handled in the context of the situation. An embedded anomalous data point in the simulated data sets is either a point anomaly or a contextual anomaly. The sequential anomalies are made based on the above definition of collective anomalies.
To generate time series data sets, it is essential to understand the fundamental backgrounds of time series models.

2.1 Generating Time Series Data without Anomalies

The definition of time series models can be easily found in diverse sources related to time series. According to Shumway and Stoffer [14]:

Definition 1: *An **autoregressive model** of order p, abbreviated **AR**(p), is of the form*

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + w_t, \tag{2.1.1}$$

where x_t is stationary, white noise $w_t \sim wn(0, \sigma^2)$ and $\phi_1, \phi_2, ..., \phi_p$ are constants $(\phi_p \neq 0)$. Autoreressive models are based on the idea that the current value of the series, x_t, can be explained as a function of p past values, $x_{t-1}, x_{t-2}, ..., x_{t-p}$, where p determines the number of steps into the past needed to forecast the current value.

Definition 2: *The **moving average model** of order q, or **MA**(q) model, is defined to be*

$$x_t = w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + \theta_q w_{t-q}, \tag{2.1.2}$$

where $w_t \sim wn(0, \sigma_q^2)$*, and* $\theta_1, \theta_2, ..., \theta_q$ *are parameters.*

Definition 3: *A time series* $\{x_t;\ t = 0, \pm 1, \pm 2, ...\}$ *is **ARMA**(p,q)*

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_p x_{t-p} + w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + ... + \theta_q w_{t-q}, \tag{2.1.3}$$

with $\phi_p \neq 0, \theta_q \neq 0$ *and* $\sigma_w^2 > 0$*. The parameter p and q are called the autoregressive and the moving average orders, respectively.*

Each data set has been generated on the basis of 10 customized methods using the function arima.sim() in the programming language **R**, which produces automatically one of the time series models as defined above. Figure 1 shows 10 randomly selected data sets out of 500. The time series generating functions produce AR models, MA models, ARMA models and ARIMA models with different parameters. The parameters, namely the values ϕ and θ in the definitions above, are chosen randomly in a sequence from -0.9 to 0.9, considering the stationarity of the time series. The graph (a), (b), (c), (d) and (e) in Figure 1 were the result of the function arima.sim() without further manipulation.

The function arima.sim() is a straightforward tool to generate a time series. However, a user must decide appropriate parameters for ϕ and θ in Definition 1, 2 and 3 one by one. If any numbers are recklessly chosen, the function could return an error message. In addition, it is hard to get an expected time series shape by only adjusting parameters. Thus, the graph (f) in Figure 1 was created by applying the cosine waves to display regular patterns. For instance the below equation,

$$4 \cdot \cos\left(\frac{2\pi}{i} + 0.3\pi\right), \quad \text{i = Index of a data point} \tag{2.1.4}$$

was used to create a time series with a regular shape. In Formula 2.1.4, which was arbitrarily designed, the coefficients in front of π and cos were randomly chosen, thus each round of simulation generates a different cosine function with varying amplitude. In order to preserve the property of time series, i.e. the correlation of data instances, the indices of the data points were used in the formula. Especially, graphs (g) and (j) in Figure 1 were generated for the purpose of testing anomaly detection methods, whether they can discover anomalies despite of the consistently increasing or decreasing trend. Both of the graphs were created by rotating the time series with cosine waves. To drift the time series artificially, the stationary time series has been rotated with randomly generated degrees, which covers the range $\pi/8$ to $\pi/20$.

For rotating, the rotation matrix,

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.1.5}$$

was multiplied to the time series data. This idea arose from imitating the effect of a linear trend, which is a more reasonable scenario for the real world, where non-stationary time series are more prevalent.

The graph (h) in Figure 1 shows a sudden change of mean. For this scenario, two randomly generated AR(1) time series with a different mean of the noise term were combined into a single time series, in which the point of contact plays the roll of a level shift. The abrupt level shift could be considered as an anomaly depending on the situation. However, simply because level shifts are not a main interest in this work, they will not be regarded as an anomaly.

Totally 500 data sets were created, and each of them includes 300 data points, thus every time series has the same length. A unit of time could be considered as a second, a day, or a month but for convenience, each time point has been consecutively numbered.

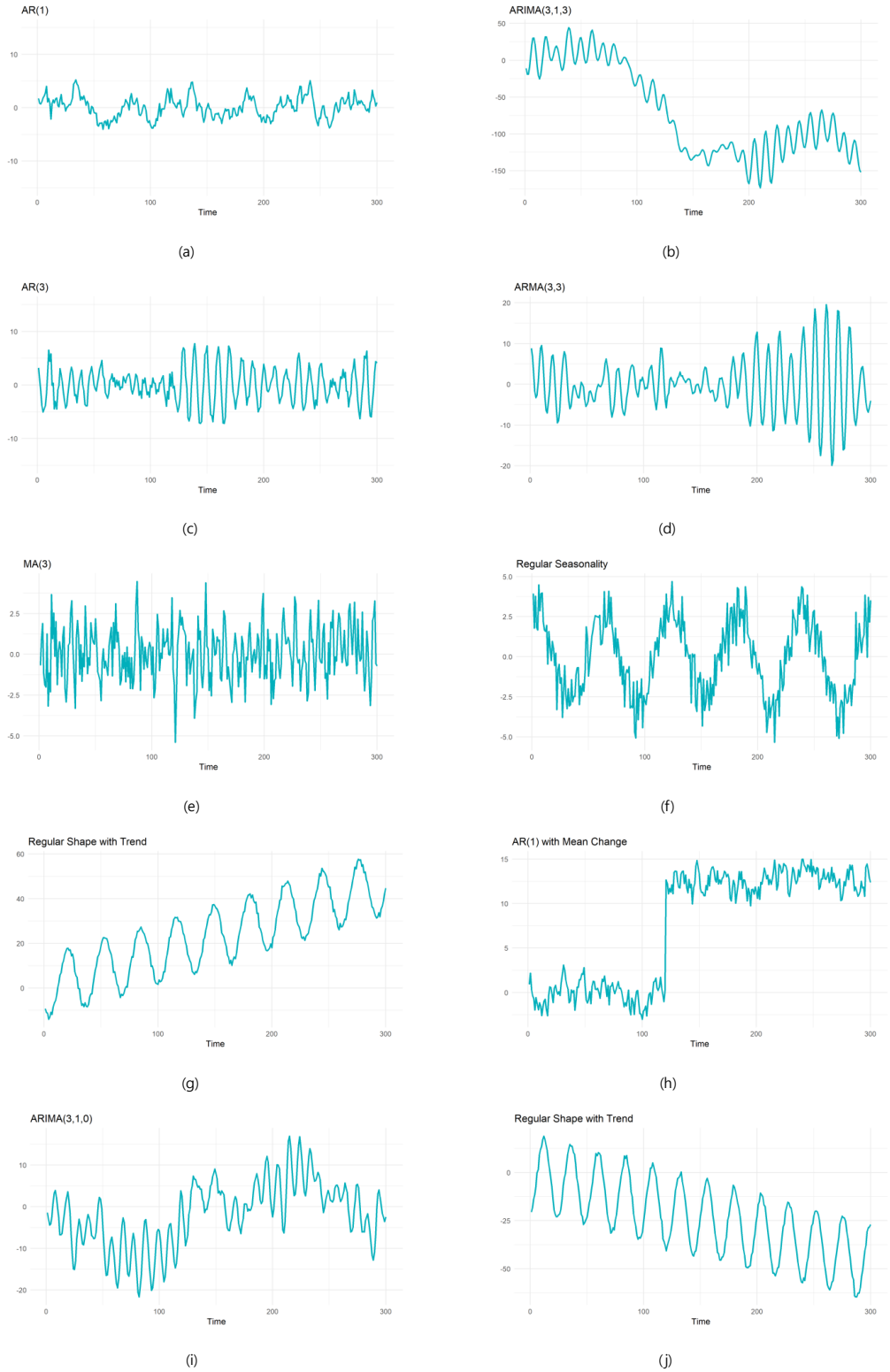


Figure 1: Different types of time series were generated without anomalies in the first place. It is assumed that all generated time series behave normally.

2.2 Embedding Anomalies in the generated Time Series Data

2.2.1 Generating a single point contextual Anomalies

The deluge of real time data and IoT sensor data through the growth of wireless technologies is urging more critical and accurate data analysis. For the reliability of data analysis, developing tools to detect various types of anomalies is one of the key issues.
Point anomalies, such as large spikes in sensor data, are simply detected by means of visualization. The main problem arises in case of contextual anomalies. Although contextual anomalies include useful information for businesses or research, they are often neglected due to the difficulty of identifying them. Thus, this work will handle both situations, not only point anomalies but also contextual anomalies which have been planted in the normal time series data as seen in Figure 2 and 3. The left side of the figures shows 10 randomly chosen normal time series data sets. On the right side the arbitrarily planted anomalies are marked with navy-coloured points. Some of the anomalies display evident deviations from the rest of the data points as seen in graph (r) in Figure 3, which are simple point anomalies. The other anomalies, i.e. contextual anomalies are not particularly deviated as compared to the rest of the data points, but show different behaviors in comparison to the neighbouring data points. The point and contextual anomalies have been generated by the idea:

$$\text{size of anomaly} := \frac{abs(max(data) - min(data))}{2} \tag{2.2.1}$$

The distance between the maximum and the minimum of a data set divided by two has been added, if a randomly chosen data point is smaller than the mean of the data set and otherwise subtracted. The idea came from the shape of a wave. For instance, when the chosen data point is the maximum of a period in the time interval, the value at the data point shrinks by the size of the anomaly as defined in Equation 2.2.1 and the shape of the period is not the form of a smooth wave anymore. As a result, the context in the time series is broken and the point is regarded as an abnormal instance. Nevertheless, the data point does not break the boundary of the interval of the entire data points. As an example, graphs (d) in Figure 2 and (l) in Figure 3 present contextual anomalies, whereas graph (r) in Figure 3 shows simply large deviated point anomalies.
For each data set three or four anomalies have been planted (1,750 in total), which amounts to approximately 1.16% of the total number of data points. All locations of the generated anomalies were visualized with a navy-coloured point, and they will be called true anomalies in this work. Also, all data points were vectorized with the numbers 0 and 1, in a way that the location of the true anomalies and detected anomalies can be precisely computed and compared for the evaluation of the anomaly detection methods.

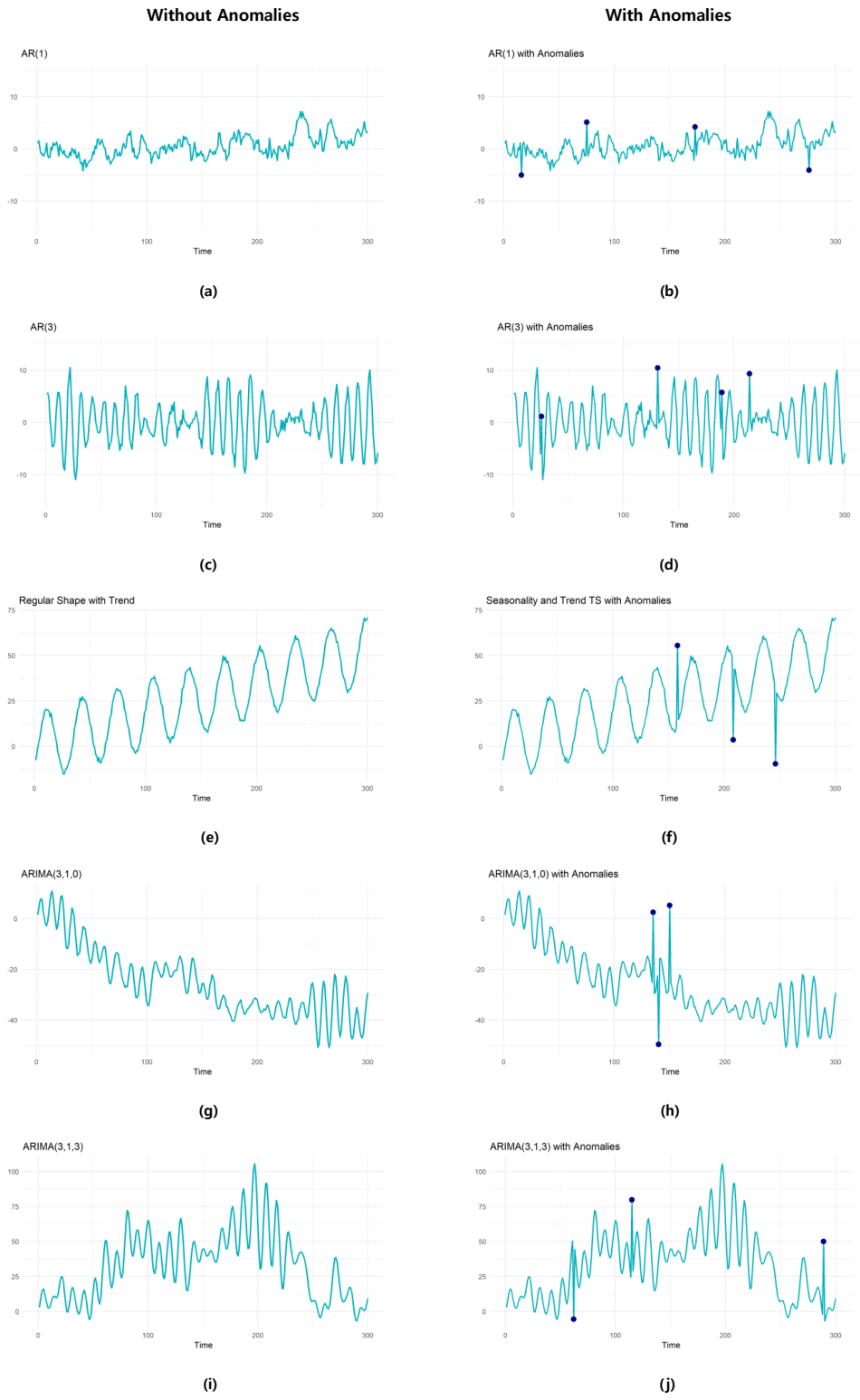


Figure 2: Generated time series without anomalies are on the left-hand side, while time series with anomalies are on the right-hand side. The navy-coloured points indicate the true contextual anomalies.

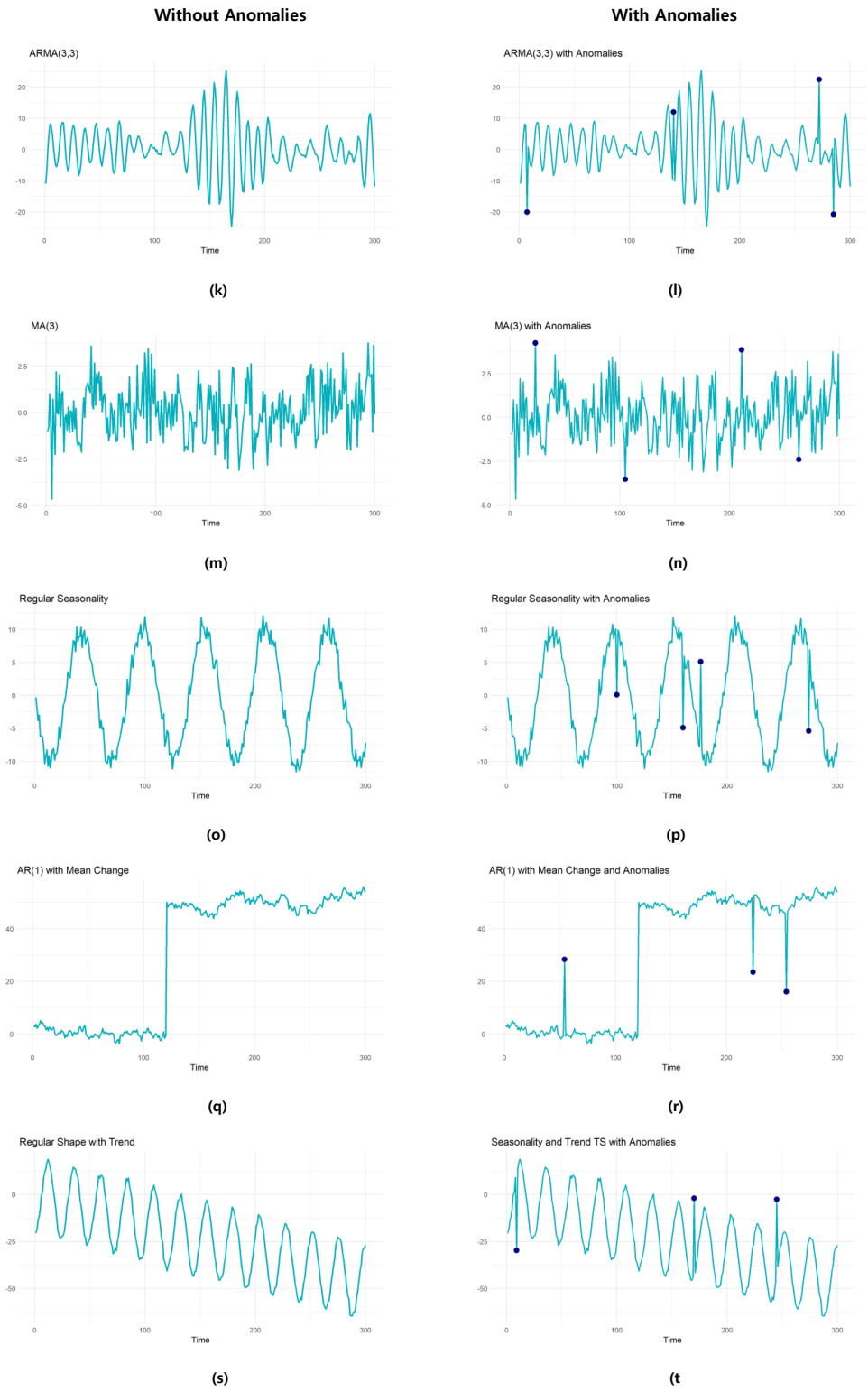


Figure 3: Generated time series without anomalies are on the left-hand side, while time series with anomalies are on the right-hand side. The navy-coloured points indicate the true contextual anomalies.

2.2.2 Generating Anomalies with an unusual shape

In order to describe the performance of the following collective anomaly detection, clearly distinguishable, and repetitive shapes of time sequences are essential. Thus, time series without any anomalies like graph (f), (g), and (j) in Figure 1 were used for basic time series, which show regular shapes with or without a linear trend over time. 200 time series were created by the function that was used for graph (f) in Figure 1 and 300 time series were generated by the function that was used for graphs (g) and (j) in Figure 1. The only difference between graphs (f), (g) and (j) is whether the function includes the process of rotating a time series or not. The length of a time series is 300 as before. Each time series has a single collective anomaly, which is a part of the time series.

The total number of collective anomalies is 500. If too many collective anomalies are generated in a single time series, it can impact the performance of the shape anomaly detection, since it is ambiguous, which time sequences should be considered as normal data. Hence, relatively short time series with a single shape anomaly were generated to simplify the interpretation of the simulation.

In order to plant a collective anomaly in a time series, a subsequence of a temporarily generated time series replaced a part of the original one, and plays the roll of the collective anomaly. The size of a collective anomaly was randomly chosen in the first place. However, due to the property of the following collective anomaly detection method, the size is set to 15 time points and this will be discussed in detail in chapter 4.

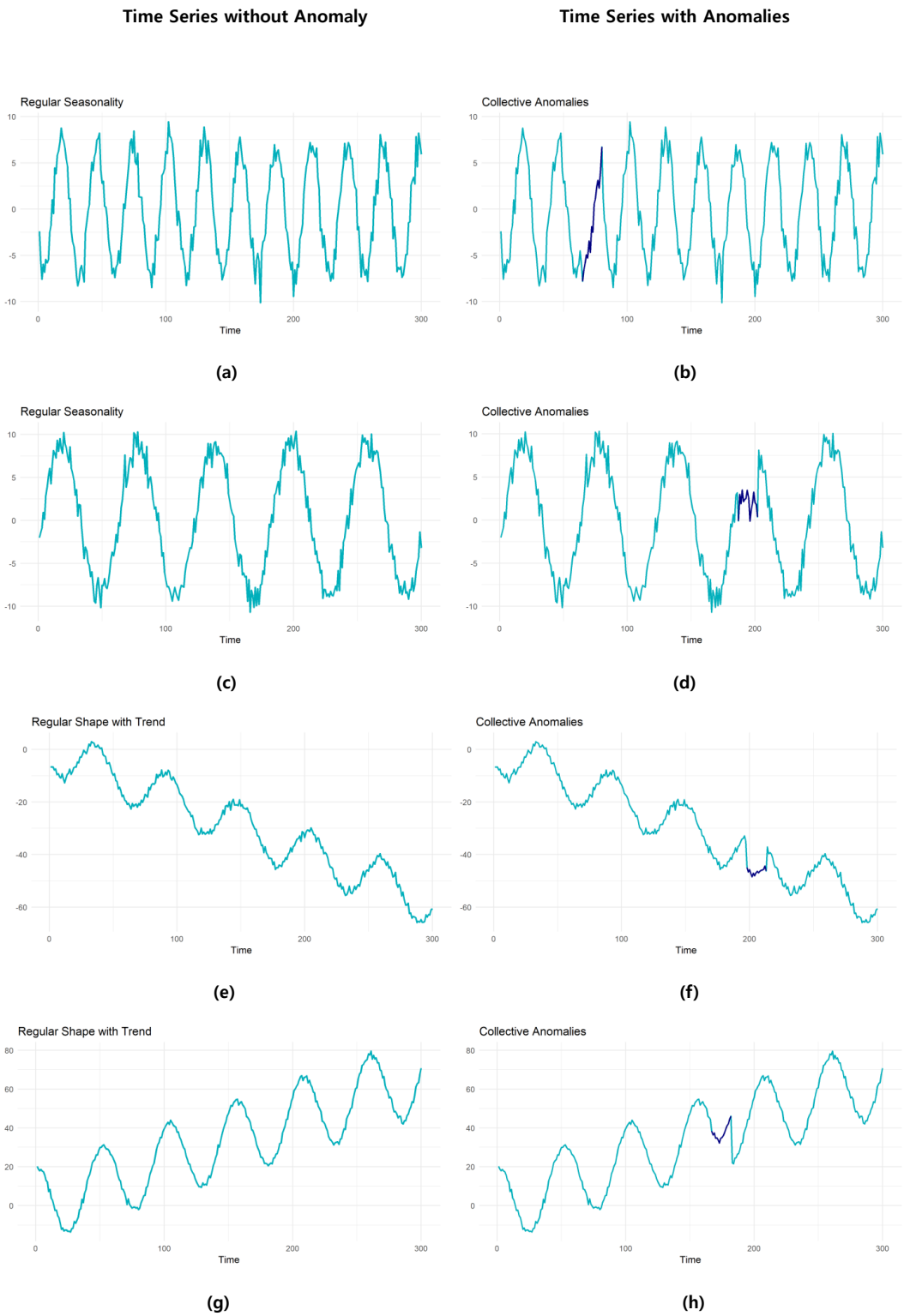


Figure 4: Generated time series without anomalies are on the left-hand side, while time series with anomalies are on the right-hand side. The navy-coloured lines indicate the true collective anomalies.

3 Prediction-based Anomaly Detection: ARIMA Model

Each data point in a time series model is influenced by the previous data points over time. Using this feature, a forecast informs about forthcoming events on the basis of accumulated data sets. In other words, abnormal situations can be uncovered through forecasting, although it strongly depends on the context of the given data set or a user decides, whether the detected data points should be classified as anomalies or not.
The most familiar approach to detect radical changes in time series data is using a forecasting model, such as ARIMA. [1, S.276] There are already a countless number of approaches about forecasts taking advantage of regression models, but slightly modified methods are constantly introduced and inspected in the business field due to the type of outliers and the reliability of existing methods. For instance, Twitter has released an open source **R** package **AnomalyDetection** whose algorithm is called *Seasonal Hybrid ESD (S-H-ESD)* based on *Seasonal and Trend decomposition (STL)* . The package makes it possible to find both *global* as well as *local anomalies* in time series.[5] Another case is the open source software **Prophet** created by Facebook. Prophet is being used for forecasting time series data which shows conspicuous seasonality and is applicable to both outlier detection as well as handling missing data. However, the Prophet forecasting model is built on a generalized additive model (GAM) instead of a ARIMA model.[15] Both Twitter and Facebook are social network platforms which handle the most formidable real time streaming data. Those two cases display the importance of outlier detection in real time streaming data and it reflects how important the development of techniques to find any type of outliers is. In this chapter the concept of anomaly detection methods based on the AR, ARMA, and ARIMA models, which are suggested by Aggarwal [1], will be introduced and implemented. As a reminder, the anomalies that previously have been designed are based on the definition of contextual anomaly, however a simple large spike might result due to the random simulation.

3.1 Background of the Approach: ARIMA Model

AR, MA, and ARMA models are desirable for *stationary* time series, which assume that the mean and variance of a time series are constant over time. The book *Time series analysis and its applications* [14] defines that *stationarity* needs the consistency when it comes to the mean and autocorrelation functions. However, real data is often not stationary. For forecasting *nonstationary* time series data, an additional time series model is necessary.

Shumway and Stoffer [14, S.132] say, *the integrated model, i.e. ARIMA is*:

Definition 4: *A process x_t is said to be **ARIMA**(p,d,q), if*

$$\nabla^d x_t = (1-B)^d x_t \tag{3.1.1}$$

is ARMA(p,q). In general, we will write the model as

$$\phi(B)(1-B)^d x_t = \theta(B) w_t \tag{3.1.2}$$

If $E(\nabla^d x_t) = \mu$, we write the model as

$$\phi(B)(1-B)^d x_t = \delta + \theta(B) w_t \tag{3.1.3}$$

where $\delta = \mu(1-\phi_1 - ... - \phi_p)$.

A *non-stationary* time series can be divided into two parts: a *non-stationary trend component* and a *zero-mean stationary component*.[14, S.131] Before forecasting, *non-stationary* data should be converted into *stationary* data by differencing each data point x_t. The ARIMA model contains the process of differencing as seen in Definition 4. Since the differentiated X_t equals the ARMA model, the data can be handled as *stationary* data in forecasting[14, S.132].

For implementing the ARIMA method to find anomalies, **R** package **tsoutliers**, which was derived from the package **forecast** in **R**, has been chosen, since this package is based on an approach described by Chen and Liu [3], and being maintained by Javier López-de-Lacalle. In *Joint estimation of model parameters and outlier effects in time series*[3] four types of outliers are introduced: *Additive Outlier* (AO), *Level Shift* (LS), *Temporary Change* (TC), and *Innovational Outlier* (IO). But due to the restriction of four outlier types considering seasonality of time series, a different type of outlier, *Seasonal Level Shift* (SLS) has been added by Kaiser and Maravall [8]. In *Seasonal outliers in time series* [8] the 5 types of outliers are defined as follows:
"An AO represents an isolated spike, a LS a step function, a TC a spike that takes a few periods to disappear and an IO a shock in the innovations of the mod."
It is crucial to understand the various types of outliers, since the 5 different types of outliers should be reinterpreted in contextual anomalies.

According to the description in *Seasonal outliers in time series*[8]:

Definition 5: *Let y_t be a time series that is the output of the ARIMA model and y_t^* denotes the observed contaminated series that contains k outliers, their combined effect can be expressed as:*

$$y_t^* = \sum_{j=1}^{k} \xi_j(B)\omega_j I_t^{(\tau_j)} + y_t \tag{3.1.4}$$

where B denotes the lag operator; ω_j is the initial impact of the outlier at time $t = \tau_j$; $I_t^{(\tau_j)}$ is an indicator variable such that it is 1 for $t = \tau_j$, and 0 otherwise; $\xi_j(B)$ determines the dynamics of the outlier occurring at $t = \tau_j$, according to the following scheme:

$$\textbf{\textit{AO}}: \quad \xi_j(B) = 1 \tag{3.1.5}$$

$$\textbf{\textit{TC}}: \quad \xi_j(B) = 1/(1-\delta B) \tag{3.1.6}$$

$$\textbf{\textit{LS}}: \quad \xi_j(B) = 1/(1-B) \tag{3.1.7}$$

The three outlier types seen in Definition 5 above are the default values for the **types** parameter in the function tso(). A TC can become an AO or a LS depending on the value of δ, as seen in Figure 5. If δ equals zero, the TC can be considered as an AO. When δ is one, the TC takes the property of a LS. In **R**, the default value of δ is set to 0.7. [7]

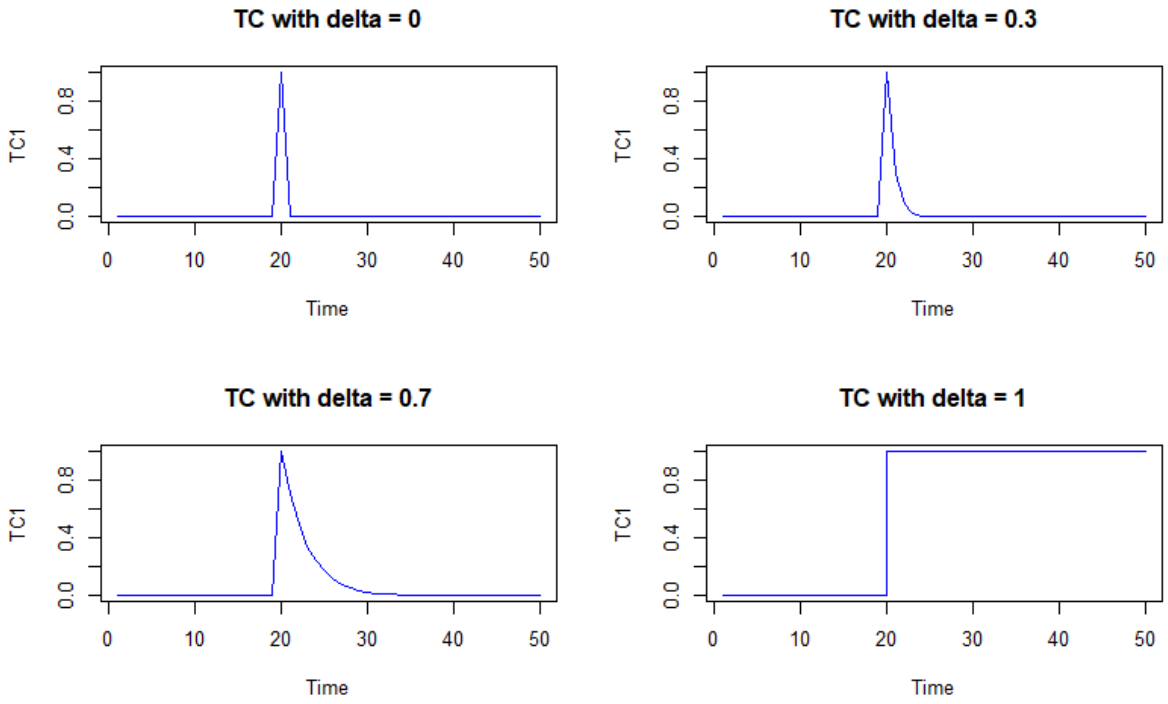


Figure 5: Description of a Temporary Change (TC) with different values for δ. $0 < \delta < 1$

The anomalies that have been generated before, can be considered as an AO, since the anomalies are produced simply by adding the arbitrarily defined distance to a normal data point, thus they

become a sudden change in a part of the time series or a large spike in the entire time series. For this reason, when it comes to the **R** package **tsoutliers**, only AO type anomalies will be detected, and are called a point anomaly or contextual anomaly in this work.

3.2 Implementation of Anomaly Detection using ARIMA

As mentioned before, the function tso() in the **R** package **tsoutliers** has been used to automatically implement the anomaly detection using ARIMA models. The automatic procedure of anomaly detection can be described in 3 phases. Manual running each step is also feasible by means of the embedded functions in the package **tsoutliers**. In the first step, an ARIMA model is fitted to the given time series data. In order to locate outliers, the significance of pre-established types of outliers (in this work, for instance, an AO type) are computed via t-statistics[2]. In the second step, a new ARIMA model is fitted with the potential outliers, which are chosen in the first step, and the t-statistics are calculated again. In this stage, not significant outliers are removed and a new ARIMA model is fitted afresh. For fitting an ARIMA model, the function auto.arima() in the **R** package **forecast** can be used for manual running. The best ARIMA model is chosen by the *Bayesian information criterion (BIC)*. The t-statistics are computed for all types of outlier at every time point. If the absolute values of the t-statistics are bigger than the previous defined critical value, than the data points are classified into the outlier candidates group. One of the outlier types with the biggest absolute value of t-statistics is chosen and manifested as an outlier with its type. In this paper, the critical value was set to the default value 3.5, and the AO type was solely used, thus, the other types of outliers were ignored.
The last step repeats step 1 and step 2 for original time series and for adjusted ARIMA models.[12] The adjusted ARIMA model will be described with a deep color line in the following Figures 6, 7, and 8, whereas the original time series data will be marked with a light color line. The left side of Figures 6, 7, and 8 shows the original location of anomalies ,which is marked with the navy-coloured points and the right side describes the results of the outlier detection using the ARIMA model. The red-marked points indicate the outliers that have been computed by the function tso() in the **R** package **tsoutliers**. As seen in graphs (d), (h) in Figure 6 and in graph (b) in Figure 7, the method using ARIMA model generates the best outcomes. However, in graphs (d) and (h) in Figure 7, the function could not find any single anomaly. Here a common interesting factor can be found. The bad performances are mostly shown in data sets with a extremely fluctuating variance. The obvious large spikes, i.e. point anomalies in graph (a) of Figure 7 have been well detected, as expected. The contextual anomalies in graph (e) of Figure 7 were also detected, but the contextual anomalies in Figure 7 graph (g) could not been found, although Figure 7 graph (g) shows a regular seasonal shape like Figure 7 graph (e). It makes the

[2] For the detailed equations, refer to [12]

crucial roll of variance more clear.

The ARIMA model method is prone to define outliers in the cosine wave data sets more often than in other data sets as described in Figure 7 graph (f). This phenomenon was also observed in the data sets with changing mean of a time series as seen in graph (b) in Figure 8. It is simply due to the level shift, which this paper decided not to concern in the function tso(). The aggregated results of the prediction-based anomaly detection method is described in Table 1.

The Result of Anomaly Detection using ARIMA			
Time Series	Data Points	Anomalies	declared Outliers
AR(1)	15,000	200	83
AR(3)	15,000	200	196
Shape with positive Trend	15,000	150	156
ARIMA(3,1,0)	15,000	150	149
MA(3)	15,000	200	90
ARIMA(3,1,3)	15,000	150	151
ARMA(3,3)	15,000	200	195
Shape	15,000	200	173
Mean Change	15,000	150	163
Shape with negative Trend	15,000	150	152
Sum	150,000	1,750	1,508

Table 1: The number of true anomalies and the detected outliers using ARIMA model

The 500 time series data sets are categorized according to the type of the time series. Each time series consists of 50 data sets with 300 data points respectively. 3 or 4 anomalies were planted in each time series data, thus totally 1750 anomalies were produced. The anomaly detection using ARIMA model defined 1508 data points as outliers. Whether the outliers are the genuine anomalies or not will be discussed in chapter 5.

The total number of detected outliers in AR(1) and MA(3) data sets is substantially smaller than in the other types of the time series, whereas more than the number of the true anomalies were detected in the time series with the mean change and cosine wave data sets with or without a trend. The result in Table 1 corresponds to Figure 6, 7, and 8. For example, the AR(1) time series in Figure 6 graph (b) with a short wave length and high frequency displays a poor performance, while the time series with cosine waves in Figure 8 graph (f) overdetects outliers.

Figure 6: The navy-coloured points in the left side graphs show the locations of true anomalies and the red-coloured points in the right side graphs indicate the detected outliers by the function tso(). The faint lines in the right side graphs describe an original time series.

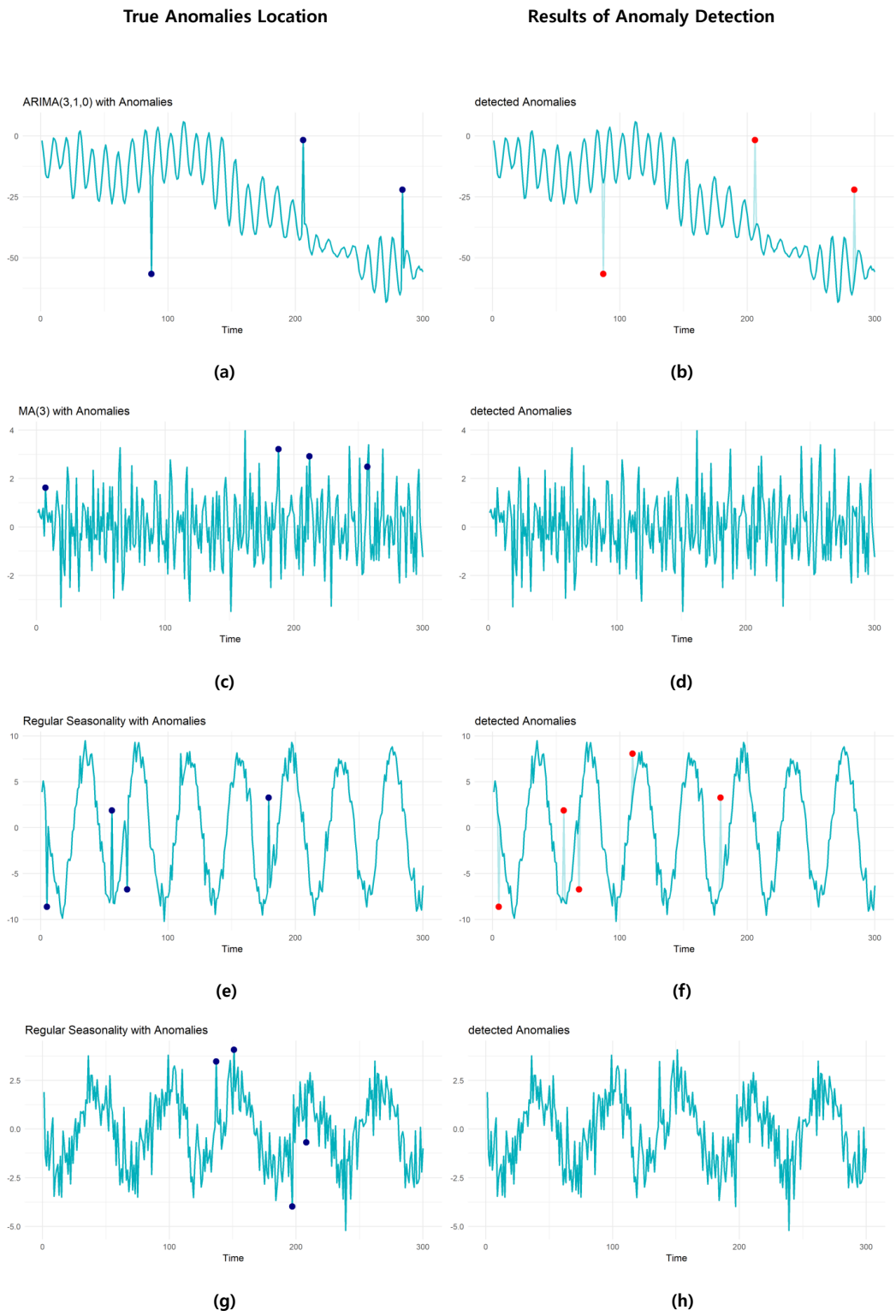


Figure 7: The navy-coloured points in the left side graphs show the locations of true anomalies and the red-coloured points in the right side graphs indicate the detected outliers by the function tso(). The faint lines in the right side graphs describe an original time series.

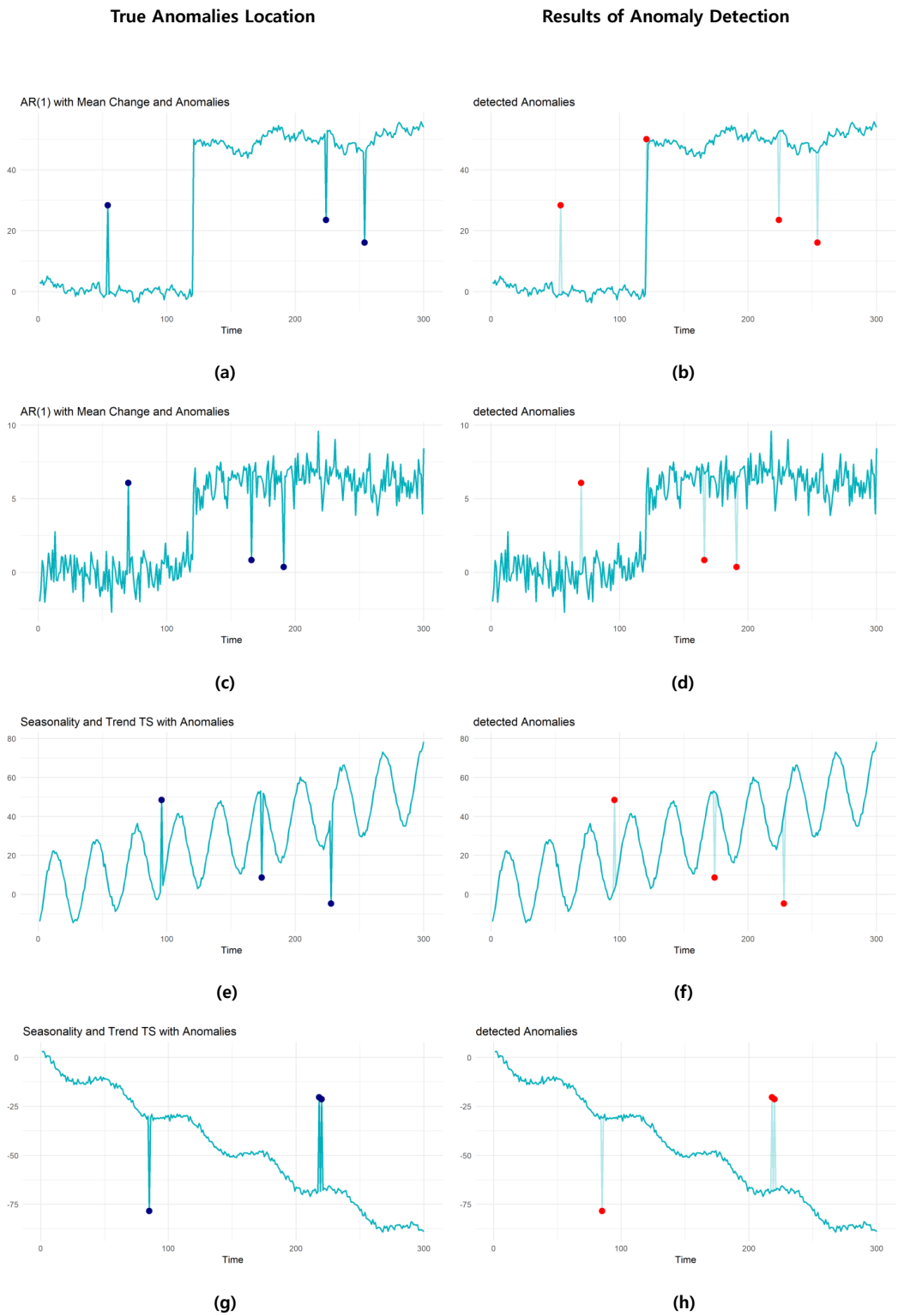


Figure 8: The navy-coloured points in the left side graphs show the locations of true anomalies and the red-coloured points in the right side graphs indicate the detected outliers by the function tso(). The faint lines in the right side graphs describe an original time series.

4 HOT SAX Algorithm

Outlier can be shown in a single point in time series as seen before, but they can also appear with a certain shape. For instance, the cardiac cycle of a human with an usual state of health presents a regular pattern. The sudden appearance of abnormal shapes in the cardiac cycle can correspond to the symptoms of his illnesses. For the purpose of detecting the sequences of outliers, the patterns of the entire sequences of time-stamps must be investigated. We call this kind of anomalies *collective anomaly*. According to Aggarwal [1], there are two scenarios within the collective anomaly detection. If a whole time series is considered as an anomaly as a result of comparing it with other similar time series, it is called *Full-series anomaly*. If a time series is long enough to be split into windows, in which certain patterns are conspicuous, the anomalous sequence of the time series is called *Subsequence-based anomaly*[1, S.287]. This chapter introduces ***H**euristically **O**rder **T**ime series using the **S**ymbolic **A**ggregate approo**X**imation* (HOT SAX) approach, which can be applied for the collective anomaly detection with the help of equidistant windows in a time series. Before detecting anomalies, a given time series must be transformed either into a *numeric multidimensional transformation* or a *discrete sequence transformation*. The HOT SAX algorithm is based on the discrete sequence transformation. According to this theory, continuous time series data can be transformed into discrete data. [1, S.290] The form of discrete data can be represented by the means or slopes of certain windows in time series data. [1, S.290] Remarkable information from the discrete data makes it possible to distinguish the subtle difference between normal and abnormal time series sequences.

4.1 Background of the Approach: HOT SAX

The HOT SAX algorithm suggested by Keogh et al. [9] uses the similarity of time series sequences. The term, "time series *discords*" indicates the most different subsequence compared with all other parts of the time series. Thus, time series discords can be understood as collective anomalies in this work. The formal definition is written in Definition 11. In order to implement the HOT SAX approach the overall understanding of the algorithm and several notations are required. According to Keogh et al. [10]:

Definition 6: *Time Series: A time series $T = t_1, ..., t_m$ is an ordered set of m real-valued variables.*

Definition 7: *Subsequence: Given a time series T of length m, a subsequence C of T is a sampling of length $n \leq m$ of contiguous position from T, that is, $C = t_p, ..., t_{p+n-1}$ for $1 \leq p \leq m-n+1$ Since all subsequences may potentially be discords, any algorithm will eventually have to extract all of them;this can be achieved by the use of a sliding window.*

Definition 8: *Sliding Window: Given a time series T of length m, and a user-defined subsequence length of n, all possible subsequences can be extracted by sliding a window of size n across T and considering each subsequence C_p.*

The HOT SAX algorithm uses the concept of the distances between time series subsequences in order to discover the least similar subsequence. For computing distance the *Euclidean distance* was chosen.

Definition 9: *Euclidean Distance: Given two time series Q and C of length n, the Euclidean distance between them is defined as:*

$$Dist(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \tag{4.1.1}$$

The key point in this algorithm is excluding *trivial matches*, which indicates that a subsequence in a time series T includes a part of a matching subsequence or the subsequence itself corresponds perfectly to the matching subsequence. Trivial matches can cause unintuitive results[10]. The definition of *Non-self match* is briefly quoted below. A detailed description and examples are available in *Finding the most unusual time series subsequence: algorithms and applications* [10] for further understanding of the problem of trivial matches and why non-self match is used for the HOT SAX approach.

Definition 10: *Non-Self Match: Given a time series T, containing a subsequence C of length n, beginning at position p and a, matching subsequence M, beginning at q, M is called as a non-self match to C at distance of Dist(M,C), if $|p - q| \geq n$.*

Definition 11: *Time Series Discord: Given a time series T, the subsequence D of length n beginning at position l is said to be the discord of T, if D has the largest distance to its nearest non-self match. That is, $\forall$ subsequence C of T, non-self match M_D of D, and non-self match M_C of C, min(Dist(D, M_D))> min(Dist(C,M_C)*

Before realizing the HOT SAX approach, two algorithms in a similar context need to be compared in order to prove time efficiency, since the implementing time has become a crucial obstacles in the era of big data. The two algorithms are introduced as a form of pseudo code in Table 2 and 3.

Due to its simplicity and accuracy, *Brute force* algorithms are a common approach in research and development areas to solve a problem. With the aid of the brute force algorithm in Table 2, time series discords can be easily implemented. For this algorithm a single parameter, i.e, the length of subsequence is required exclusively [10]. All possible subsequences become candidates to be concerned in the outer loop of the pseudo code. The nested loop computes the distance to the *nearest non-self match*. The subsequence with the largest distance is defined as time series

1	**Function** [dist, loc] = Brute_Force(T,n)
2	best_so_far_dist = 0
3	best_so_far_loc = NaN
4	
5	**For** $p = 1$ **to** $\|T\| - n + 1$ // Begin Outer Loop
6	nearest_neighbor_dist = infinity
7	**For** $q = 1$ **to** $\|T\| - n + 1$ // Begin Inner Loop
8	**IF** | p – q | $\geq n$ // non-self match?
9	**IF** $Dist(t_p,...,t_{p+n-1},t_q,...,t_{q+n-1}) <$ nearest_neighbor_dist
10	nearest_neighbor_dist = $Dist(t_p,...,t_{p+n-1},t_q,...,t_{q+n-1})$
11	**End**
12	**End** // End non-self match test
13	**End** // End Inner Loop
14	**IF** nearest_neighbor_dist > best_so_far_dist
15	best_so_far_dist = nearest_neighbor_dist
16	best_so_far_loc = p
17	**End**
18	**End** // End Outer Loop
19	**Return** [best_so_far_dist, best_so_far_loc]

Table 2: Brute Force Discord Discovery Pseudo Code. Extracted from [10]

discord [10]. Nevertheless, the brute force algorithm increases the processing time just as the name implies, which is not preferable for enormous data sets like sensor data. This Algorithm needs to be modified as seen in Table 3 which describes the HOT SAX method.
The former algorithm computes the nearest neighbor to each subsequence, which is not always required. Therefore, if a more adjacent subsequence to the present candidate than the *best-so-far-distance* is discovered, the present candidate must be excluded from the list of time series discords. In addition, considering the order of each subsequence in the outer loop and the order of finding a closer subsequence in the inner loop have a huge influence on improving the running time [10]. By altering the pseudo code, as seen on line 5 to 11 of Table 3, the time complexity can be ameliorated by a considerable margin. However, two heuristics still need to be specified.

1	**Function** [dist, loc] = Heuristic_Search($T, n, Outer, Inner$)
2	best_so_far_dist = 0
3	best_so_far_loc = NaN
4	
5	**For Each** p in T ordered by heuristic Outer // Begin Outer Loop
6	nearest_neighbor_dist = infinity
7	**For Each** q in T ordered by heuristic Inner // Begin Inner Loop
8	**IF** | p – q | $\geq n$ // non-self match?
9	**IF** $Dist(t_p, ..., t_{p+n-1}, t_q, ..., t_{q+n-1}) <$ best_so_far_dist
10	**Break** // Break out of Inner Loop
11	**End**
12	**IF** $Dist(t_p, ..., t_{p+n-1}, t_q, ..., t_{q+n-1}) <$ nearest_neighbor_dist
13	nearest_neighbor_dist = $Dist(t_p, ..., t_{p+n-1}, t_q, ..., t_{q+n-1})$
14	**End**
15	**End** // End non-self match test
16	**End** // End Inner Loop
17	**IF** nearest_neighbor_dist > best_so_far_dist
18	best_so_far_dist = nearest_neighbor_dist
19	best_so_far_loc = p
20	**End**
21	**End** // End Outer Loop
22	**Return** [best_so_far_dist, best_so_far_loc]

Table 3: Heuristic Discord Discovery Pseudo Code. Extracted from [10]

Keogh et al. [10] has mentioned, "The input has been augmented by two heuristics, one to determine the order in which the outer loop visits the subsequences, and one to determine the order in which the inner loop visits the subsequences."
And three heuristic approaches are introduced for optimal subsequence ordering: *Random*, *Perverse*, and *Magic*. Among them, the *Magic heuristics* has been chosen as the most ideal strategy by Keogh et al. [9], since it shows the best performance to time complexity. In addition, the HOT SAX algorithm is based on the approximation of the Magic ordering, this work will exclusively focus on the Magic heuristics. Any further explanations of the two other approaches,

Random ordering and *Perverse ordering*, can be found in *Finding the most unusual time series subsequence: algorithms and applications* [10].
In the same paper the Magic heuristics has been described as:

> In this hypothetical situation, we imagine that a friendly oracle gives us the best possible orderings. These are as follows: For outer, the subsequences are sorted by descending order of the non-self distance to their nearest neighbor, so that the true discord is the first object examined. For inner, the subsequences are sorted in ascending order of distance to the current candidate. For the Magic heuristic, the first invocation of the inner loop will run to completion.Thereafter,all subsequent invocations of the inner loop will be abandoned during the very first iteration. [10]

As mentioned before, what the Magic heuristics requires is a ideal scenario, which can not be perfectly implemented in the real world. However, approximation to the Magic ordering is feasible. In order to understand the approximation procedure, it is inevitable to learn the ***S**ymbolic **A**ggregate Appro**X**imation*(SAX) representation described by Lin et al. [11]. SAX representation is one of the substitutions to express a time series in its special way. Above all, a given time series needs to be transformed into the ***P**iecewise **A**ggregate **A**pproximation*(PAA) as shown in Figure 9. The time series C with the length n has been split into 8 equidistant frames (w) and the true values of the time series C have been substituted by the mean value of the partial time series in each frame. Each mean value $\bar{C}_i$ can be computed by Equation 4.1.2 introduced in *A symbolic representation of time series, with implications for streaming algorithm* [11].

$$\bar{C}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} C_j \tag{4.1.2}$$

In order to attain the SAX representation, the discretized time series in Figure 9 must be further modified. For processing the modification, Keogh et al. [10] defined the term *Breakpoints* as:

Definition 12: *Breakpoints: Breakpoints are a sorted list of numbers* $B = \beta_1, ..., \beta_{a-1}$ *such that the area under a* $N(0,1)$ *Gaussian curve from* β_i *to* $\beta_{i+1} = 1/a$ *(*β_0 *and* β_a *are defined as -*∞ *and* ∞*, respectively).*

According to Lin et al. [11], normalized subsequences are usually normal distributed, which does not crucially affect the *correctness* of the HOT SAX algorithms. Therefore, the assumption of a Gaussian distribution was used for defining breakpoints. 'a' in Definition 12 indicates the number of used symbols. Thus, breakpoints are highly dependent on the symbol size. Symbols are not restricted, as long as the chosen symbols have the same probability to appear. As seen in Figure 10, the characters 'a', 'b', and 'c' are used as symbol. Lin et al. [11] named a sequence of symbols *Word*.

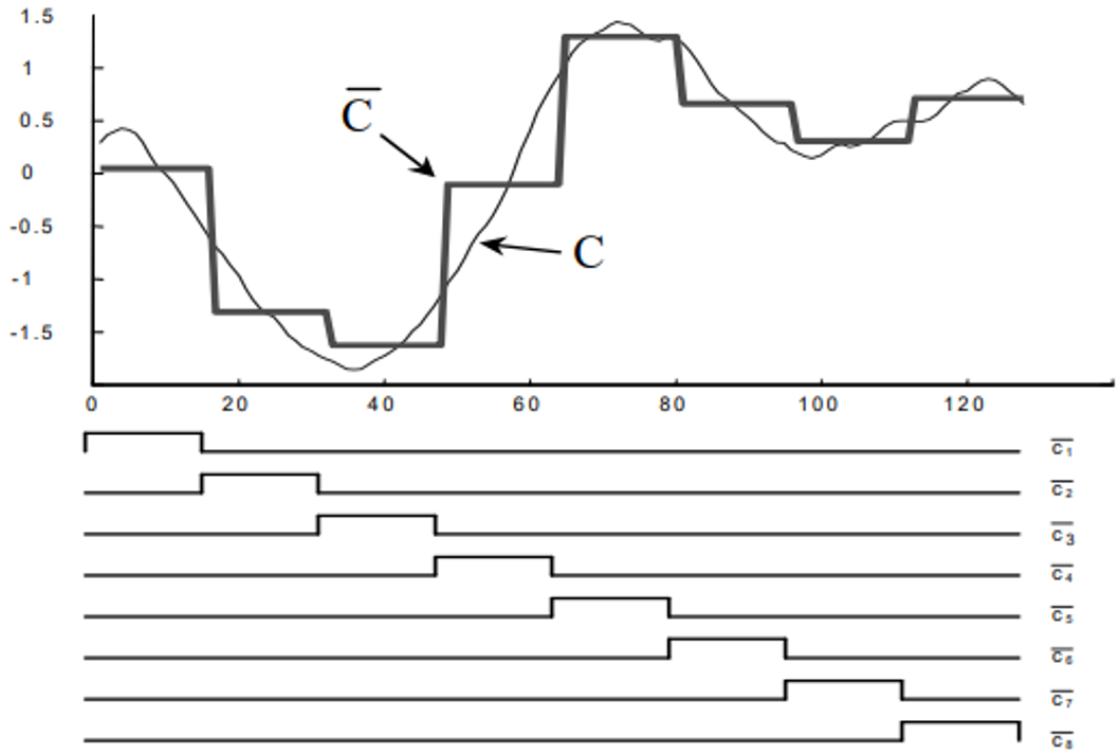


Figure 9: The PAA representation of a time series: The time series with the length 128 is split into 8 segments. Each $\bar{C}_i$ has been obtained by Equation 4.1.2. Extracted from Figure 3 in [11].

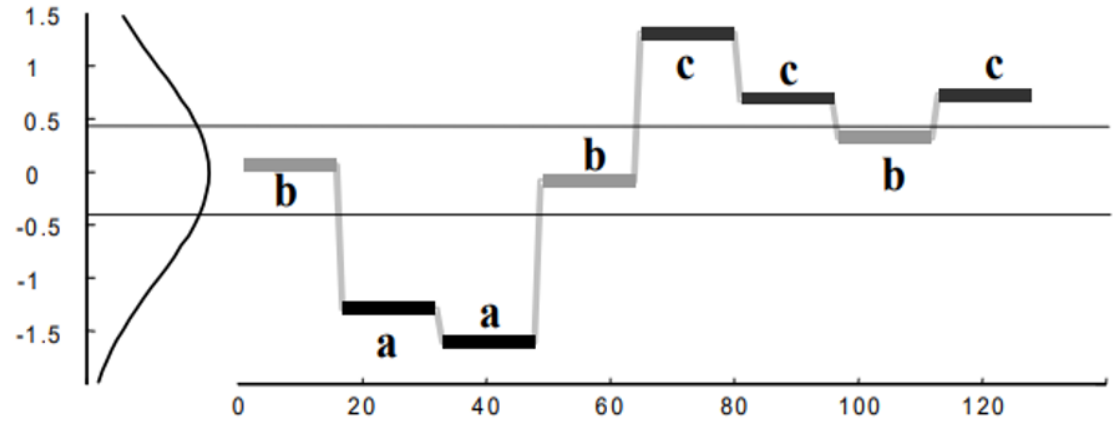


Figure 10: The SAX representation of a time series: The time series C with the length n = 128, the number of segments w = 8, and the symbol size a = 3 is matched with the word baabccbc. Extracted from Figure 5 in [11]

Now, almost all essential knowledge has been stated for implementing the HOT SAX. The last stage before implementing the code is to explore the mechanism of approximating the Magic ordering.

First, a given time series T must be transformed into a SAX representation. Note that Keogh et al. [9] assumed that the length of a time series discord is known beforehand. Through this procedure, each subsequence with a unique shape will be matched with an unusual sequence of symbols. The frequency of each symbols is recorded and the indices of the SAX representation with the smallest number of the frequency are visited first by the outer loop [10]. The *Outer* heuristic has now been acquired. To sum up, few possible subsequences are extracted as candidates for discords by the outer loop of the heuristic discord discovery algorithm. It allows an earlier stop of the inner loop, which contributes to improved running time.
After the outer loop sets the first candidate by using the index of the Word subsequence, the other indices, that are mapped to the same Word of the current candidate, can be found. The inner loop visits these indices first and the rest of the subsequences are visited randomly [10]. Soon after finding a subsequence with a smaller distance than best_so_far_dist, the inner loop is ceased.

4.2 Implementation of Anomaly Detection using the HOT SAX algorithm

This paper takes advantage of the **R** package **jmotif** that allows both pseudo codes in Table 2 and 3 to be implemented. To find a discord via the brute force algorithm, the function find_discords_brute_force is available. However, this work will not discuss the brute force algorithm. Instead, the function find_discord_hotsax will be consistently used to detect the collective anomalies.
In order to implement the HOT SAX algorithm, all time series data sets must be z-normalized. The z-normalization process is already embedded in the function find_discord_hotsax, and the user can control through a threshold, whether the process must be accomplished or not. In this work the threshold value is set to 0.01, which is the default value of the function znorm() in the package **jmotif**. When completing the z-normalization, one of the time series data set results in:

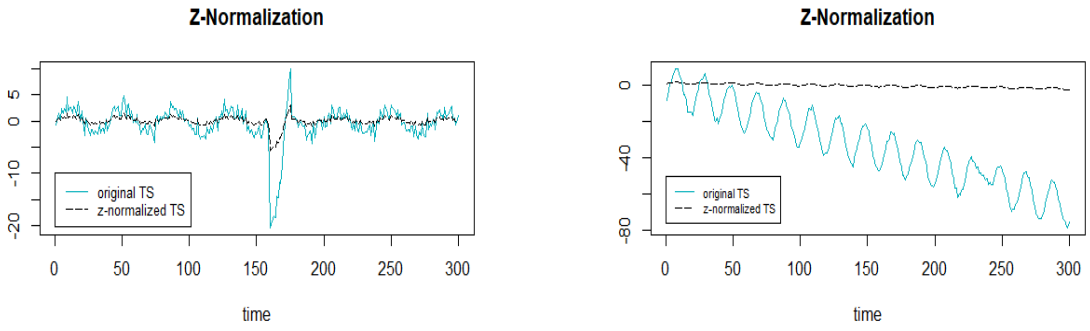
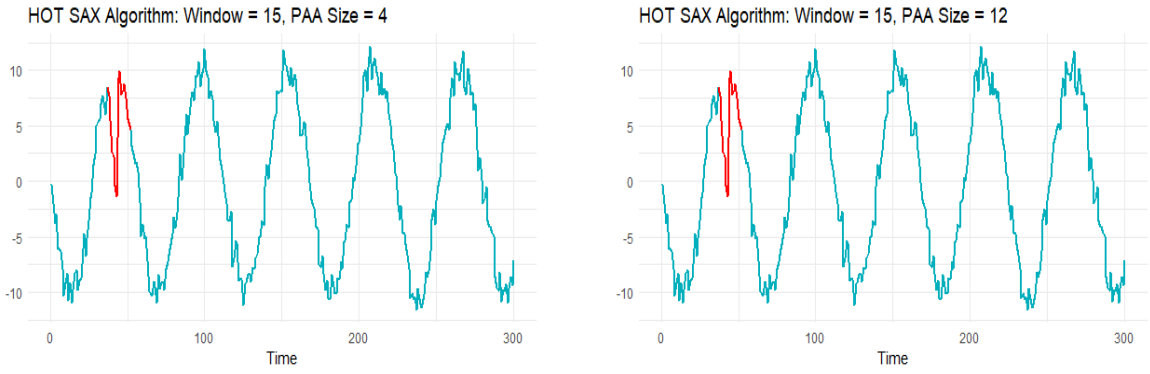


Figure 11: Z-normalization.

Now two parameters must be chosen: one is the dimension of the PAA and the other one is the symbol size of the SAX. According to Keogh et al. [9], fortunately the word size and alphabet

size do not change the result of the anomaly detection. Keogh et al. [9] suggests, to set the alphabet size to three or four, since it is the best value regarding any data set. But,the word size should be chosen depending on the data set. It is mentioned in *Finding the most unusual time series subsequence: algorithms and applications* [10] that, "In general, relatively smooth and slowly changing datasets favor a smaller value of w, whereas more complex time series favor a larger value of w." To decide on the word size, the parameter was changed each time during implementation. Actually it was proved correct that the alphabet size and the word size do not change the best discord candidate as seen below in Figure 12. Hence, the alphabet size and word size will be set to 4 in the following implementation of the algorithm.



(a) Result with a PAA size 4

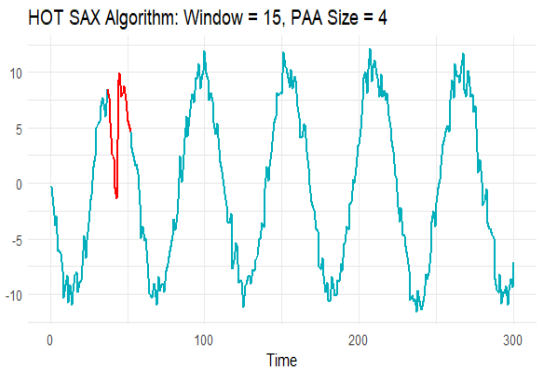
(b) Result with a PAA size 12

Figure 12: A time series was implemented with two different PAA sizes under identical conditions: one was set to 4 and the other to 12. The red line indicates a detected anomaly sequence. Both implementations produced an identical and correct result.

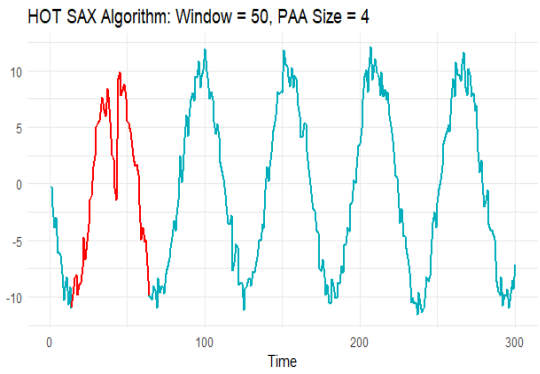
Before or after deciding a word size and alphabet size, a time series needs to be cut into equidistant windows. Each subsequence obtained by *sliding a window* is compared with each other and the best discord is computed. Note that while the notation w has been called PAA size, the argument **w_size** in **R** indicates the length of the window, whereas the word size is represented by the argument **paa_size**.

Now a question may arise. On what criteria can a user decide the length of the window? Unfortunately, the length of the window was simply defined as the length of the discords in *Hot sax: Efficiently finding the most unusual time series subsequence* [9] without further explanation, which is only possible, when the lengths of the discords are all well-known. If the discords are well-known in advance, anomaly detection may signify nothing in practice. However, because generated data has been used for this paper, it is known that the length of a discord is set to 15. The reason for embedding only fixed length of anomalies stems exactly from the lack of information on deciding the window size. Moreover, the length of the window has crucial effects on the result of anomaly detection.

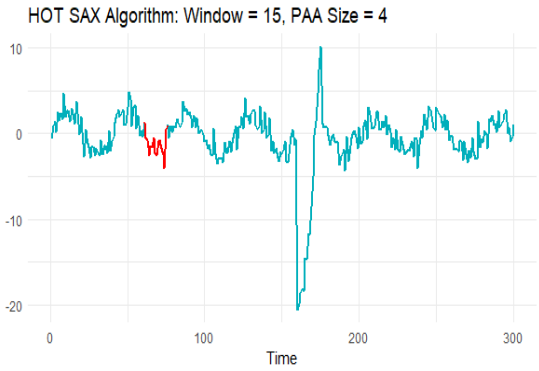
The graphs (a) and (d) in Figure 13 demonstrate the correctly detected location of a discord that



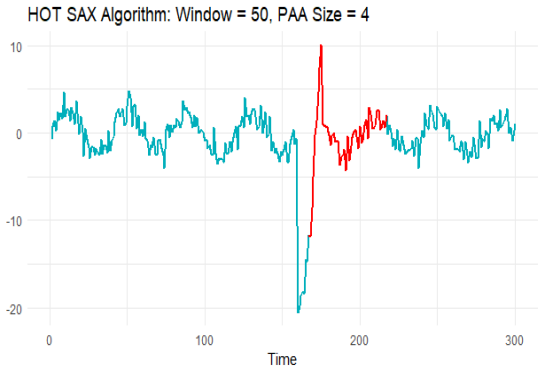
(a) Result with window size 15



(b) Result with window size 50



(c) Result with window size 15



(d) Result with window size 50

Figure 13: Anomaly detection produces a different result depending on the window size. The correct position of a discord is red-colored with window size 15 in graph (a), whereas the HOT SAX algorithm failed to detect the right discord position in graph (c) with window size 15.

displays abnormal behavior in comparison to the other shapes of the time subsequences. On the other hand, graphs (b) and (c) show one of the cases of falsely recognized anomalies, when the window size has been changed. In the first place, the window size was set to 15 in this work, and few time series with the failed cases were detected. Since then, the window size was changed to 30, 50, 70, 100, and so on. However, when considering the size of a time series, which is set to 300, the length of window more than 100 is meaningless to analyze, since it is too big to be a size of an anomaly. Thus, the Figure 14 reflects only the results up to the window size 100.

As seen in Figure 14, the result with window size 15 has been reversed into a better result, which means, that the HOT SAX found more of the right discords locations, when increasing the length of window. Assigning 50 to the window length, the HOT SAX algorithm provided the most optimal result among the trials with different window sizes. The aggregation of the result was accomplished through vectorization of the positions of the discords. In a true anomaly vector, the true anomalous subsequences were denoted by the number one, and the rest of the data points were marked as the number zero. In a result vector, the points defined as a discord were

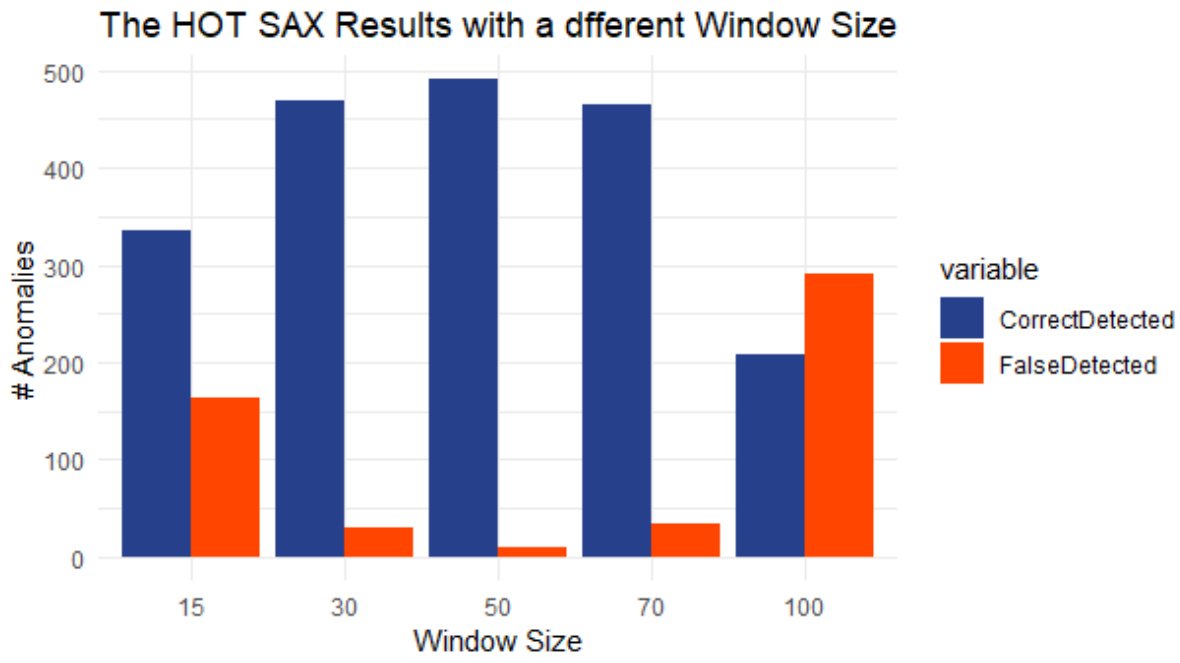
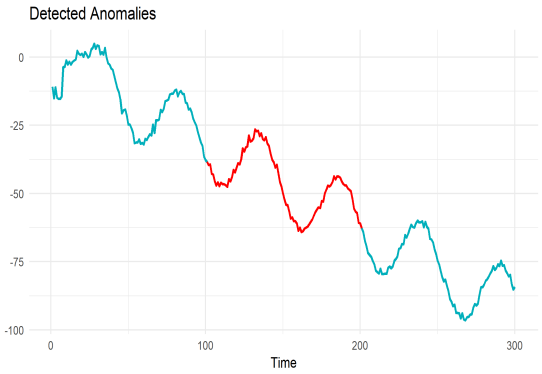


Figure 14: The HOT SAX algorithm depends strongly on the window size. The blue bars display the numbers of correctly detected anomalies, whereas the red bars indicate the number of falsely detected anomalies. When adjusting the window size to 50, it is possible to obtain an optimal result. In total, 491 of the true anomalies are successfully discovered at window size 50.

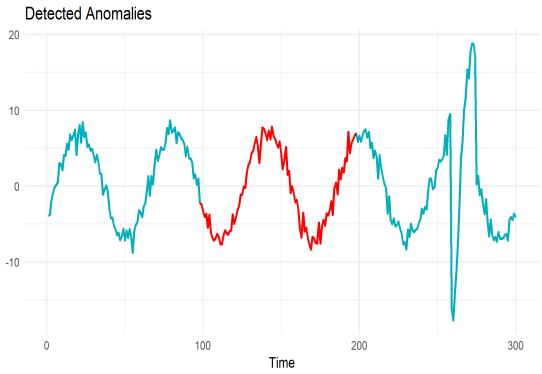
denoted by the number one, and the other were marked as the number zero, as the true anomaly vector. Then, the two vectors, the true anomaly vector and the result vector, are added. If the number 2 appears in the added vector at least one time, it is considered that a discord is found correctly. 491 anomalies were correctly discovered under the condition of the window size 50, which amounts to 98.2% of the true anomalies. On the other hand, 336 anomalies were detected, which comes to 67.2%, when setting the window size to 15, just as the size of a discord. In the paper *Finding the most unusual time series subsequence: algorithms and applications* [10], the size of discords is assumed to be known in advance, and was used for sliding windows. It has now been proven that, in order to achieve a better result the size of a window is not necessarily equal to the length of a discord.

However, the result of the HOT SAX algorithm becomes gradually worse, when the window size becomes bigger. Figure 15 demonstrates the poor performances of the HOT SAX algorithm, and it corresponds to the result in Figure 14 and 18.

It is now apparent that the performance of the HOT SAX algorithm fluctuates strongly, when adjusting the window size. For this reason Keogh et al. [10] did not suggest a concrete criteria to decide the size of window, an painstaking exploration is inevitable. Nonetheless, once a decent size of window is chosen, the HOT SAX algorithm shows a remarkable potentiality as a shape anomaly detection in real time data. As seen in Figure 16, the HOT SAX algorithm shows a



(a) Result with window size 100



(b) Result with window size 100

Figure 15: The true discord is placed between time points 1 and 50 in graph (a). In graph (b) the abnormal time subsequence lies between time points 150 and 300. Both graphs demonstrates the failure of the anomaly detection with window size 100.

better performance with respect to relatively small anomalies that can often be neglected. When it comes to a situation in which a subtle change can cause serious consequences, as a heart rate sensor, this anomaly detection can be especially advantageous. Furthermore, it is available for the time series data sets that involves a trend. A discord is correctly detected in the time series with a regular seasonality as well as a trend in graph (c) of Figure 16. In addition, it is possible to control the number of candidates to be discovered in the function find_discords_hotsax(). Thus, not only a single discord, but also few candidates of an anomaly can be under consideration. Even thought a candidate with the farthest distance is not a true anomaly, a user of the outlier detection can be forewarned by the other candidates.

On the contrary of the case of the subtle abnormality, obvious shape anomalies with a large deviation were not well spotted. Graphs (f) and (h) in Figure 16 present a distinct irregularity respectively. However, t ation of the discord. After changing the window size to 50, the true position of the anomalies were accurately found as seen in Figure 17 graphs (f) and (h).

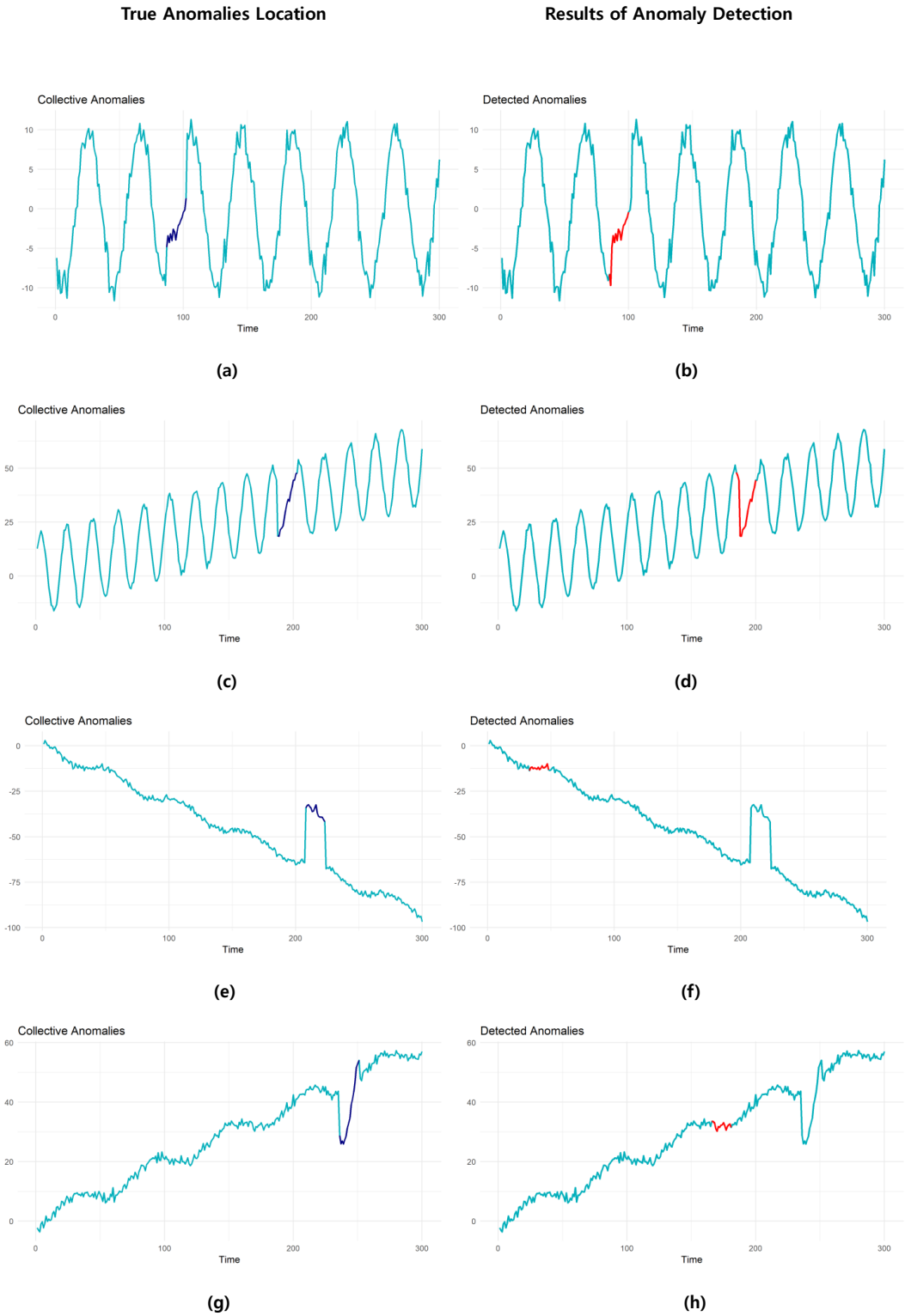


Figure 16: The result of the HOT SAX algorithm with window size 15. The left-hand side indicates the true shape anomalies(discords) and the right-hand side shows the result of the HOT SAX algorithm with window size 15, alphabet size 4, and word size 4.

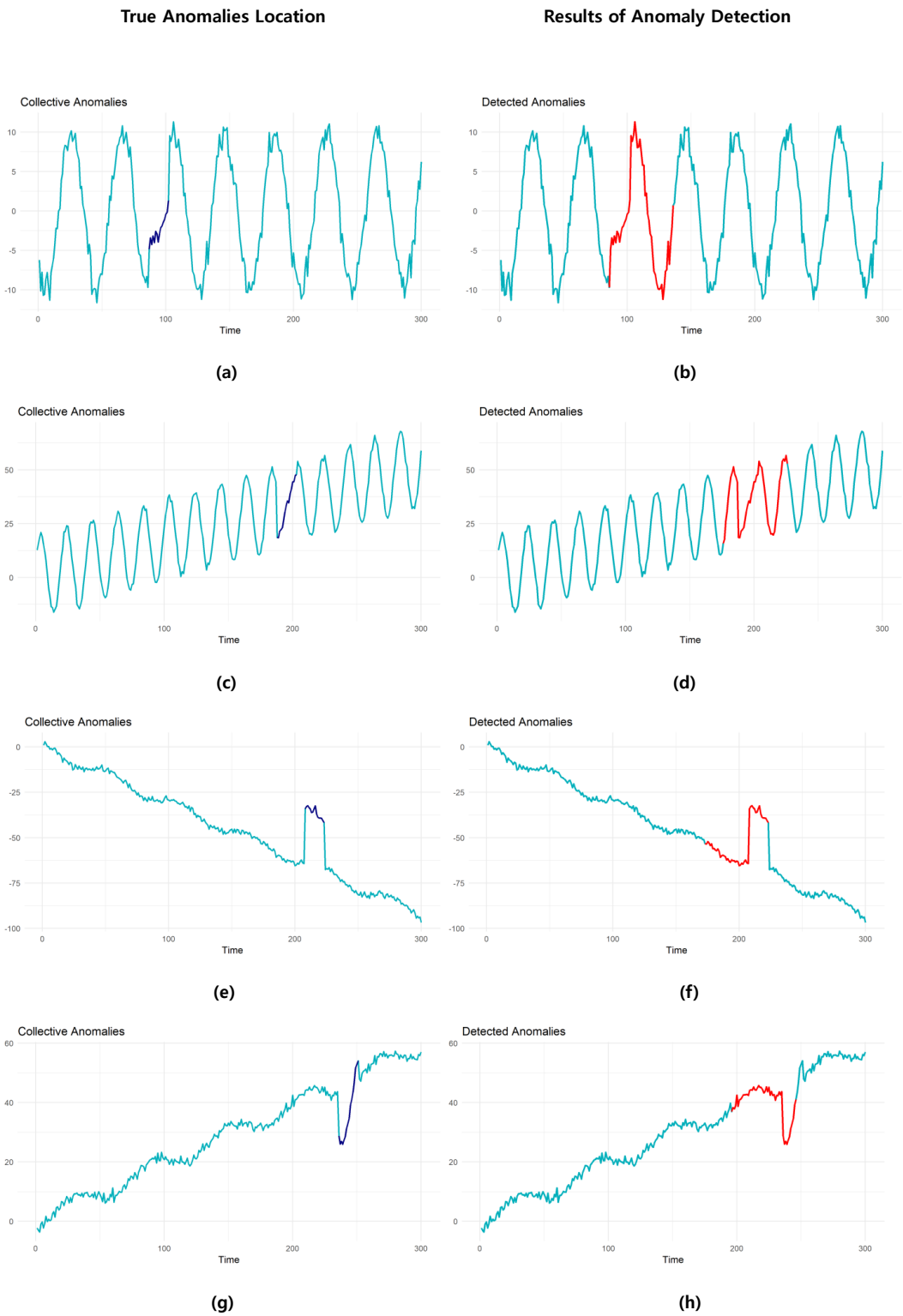


Figure 17: The result of the HOT SAX algorithm with window size 50. The left-hand side indicates the true shape anomalies(discords) and the right-hand side shows the result of the HOT SAX algorithm with window size 50, alphabet size 4, and word size 4.

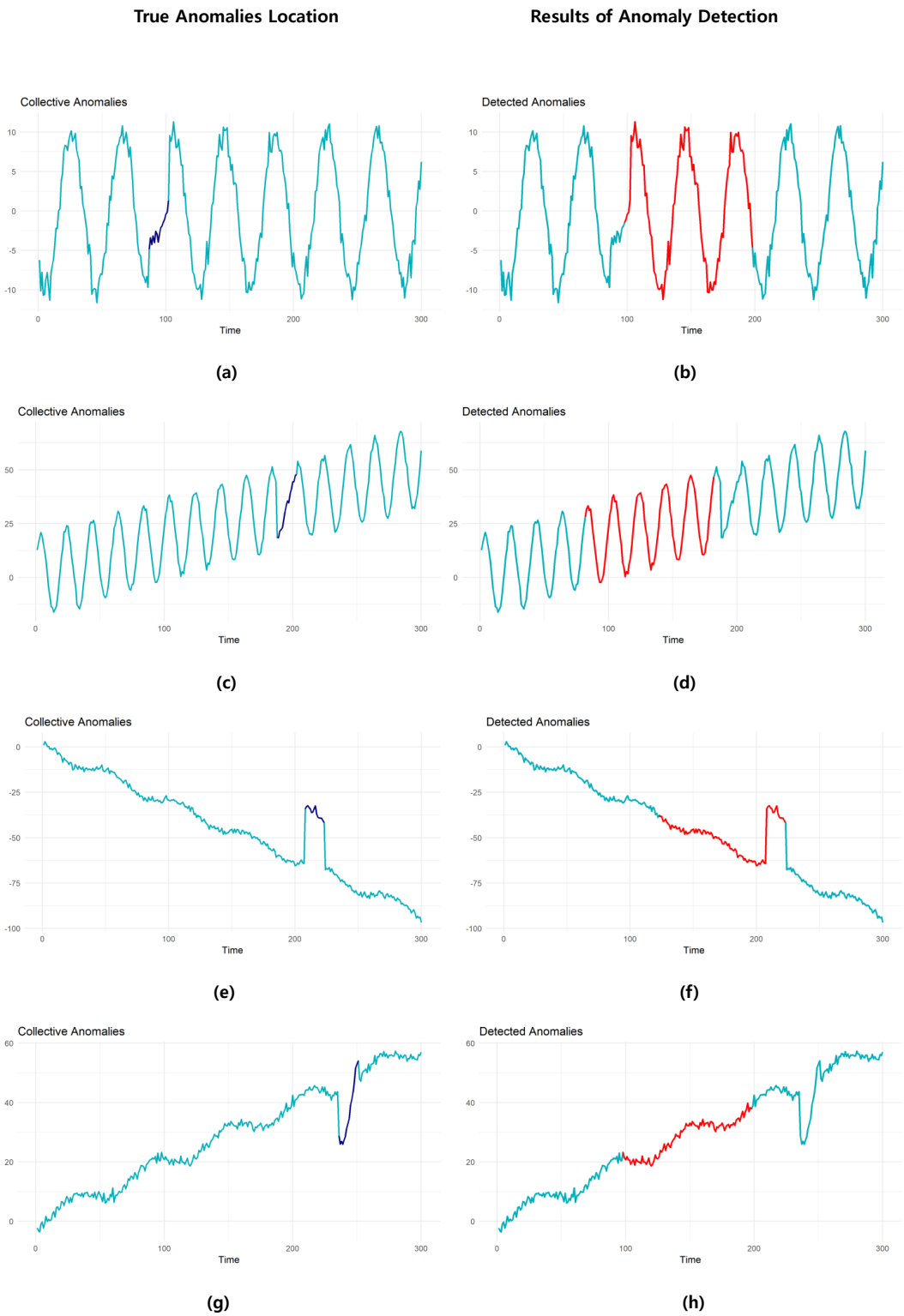


Figure 18: The result of the HOT SAX algorithm with window size 100. The left-hand side indicates the true shape anomalies(discords) and the right-hand side shows the result of the HOT SAX algorithm with window size 100, alphabet size 4, and word size 4.

5 Evaluation and further possible Approaches

5.1 Evaluation of the anomaly detection using ARIMA Models

To evaluate the result of the ARIMA model, falsely anomaly-defined data points and correctly detected anomalies need to be aggregated. Table 4 presents the aggregated result of the anomaly detection using the ARIMA model.

Aggregated Result of Anomaly Detection using ARIMA Model					
# Data Points	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
150,000	1,750	1,508	304	62	1,446

Table 4: The aggregated Result of Anomaly Detection using ARIMA Model. There are 500 time series data sets which include respectively 300 data points. Column 1 describes the time series data points in total and column 2 the number of true anomalies. Column 3 indicates anomaly-defined data points, column 4 falsely not detected anomalies, column 5 falsely anomaly-defined data points and column 6 correctly detected anomalies.

500 time series data sets include 1,750 collective anomalies in total, which is ca. 1.16% of data points. Column 4 in Table 4 displays the cases of *false negatives*, which indicates that a data point is a true anomaly in reality, but not declared as an anomaly in an anomaly detection. The anomaly detection method using an ARIMA model could not find 304 out of 1,750 anomalies, which is approx. 17.3%. On the other hand, a *false positive* in Table 4 means that a data point is normal in reality, but defined as an anomaly in an anomaly detection. 62 normal data points were improperly declared as an anomaly in this case. In total, 1,446 out of 1750 anomalies were accurately found by ARIMA models. With those information *precision* and *recall* can be computed. Precision and recall are defined in *Outlier Analysis Second Edition* [1] as follows:

Definition 13: *For any given threshold t on the outlier score, the declared outlier set is denoted by $S(t)$. G represent the true set (ground-truth set) of outliers in the data set. Then, for any given threshold t, the precision is defined as the percentage of reported outliers that truly turn out to be outliers.*

$$Precision(t) = 100 \cdot \frac{|S(t) \bigcap G|}{S(t)} \tag{5.1.1}$$

Definition 14: *The recall is correspondingly defined as the percentage of ground-truth outliers that have been reported as outliers at threshold t.*

$$Recall(t) = 100 \cdot \frac{|S(t) \bigcap G|}{G} \tag{5.1.2}$$

Additionally, F_1 *Score* can be calculated by the aid of precision and recall. F_1 Score is as follows:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.1.3}$$

The numerator $|S(t) \bigcap G|$ in Definition 13 corresponds simply to true positives and the denominator is the sum of true positives and false negatives. According to Definition 13, the precision rate of the anomaly detection using an ARIMA model is $100 \cdot \frac{1446}{1446+62} \approx 95.8\%$, which can be considered "good". The recall rate can be attained from the proportion of properly detected anomalies to true anomalies. It is calculated as $100 \cdot \frac{1446}{1750} \approx 82.6\%$. F_1 is approximately 88.71%. Note that those rates do not indicate that the anomaly detection using prediction model promises always good performances. As mentioned in a previous chapter, only additive anomalies were considered , which are detected easier than other types of anomaly. The rate of precision and recall need to be interpreted along with the other conditions of the experiments in this work.
The anomaly detection using ARIMA models has achieved fairly good result as seen above. As already known, ARIMA models are still being used in many areas by virtue of the relatively good performance and simplicity. It can recognize different types of outliers and among them especially additive outliers are outstandingly well detected. However, it performs well only when the statistical assumptions are fulfilled, since prediction-based models like ARIMA are found on statistics theories. In addition, ARIMA models are restricted when it comes to dealing with seasonality. To overcome this disadvantage of ARIMA models, *Seasonal Decomposition of Time Series by Loess* (STL) can be regarded as an alternative. STL can treat any types of seasonality including multiple seasonalities. Although this paper has not handle data sets with multiple seasonalities, it is a fairly possible scenario in reality. A data can have not only annual seasonality but also monthly, weekly, and daily seasonality. [6]
Another option is the technique using *Seasonal Hybrid Extreme Studentized Deviate* (S-H-ESD) by Twitter. S-H-ESD is based on Seasonal-ESD (S-ESD), which decomposes a time series to verify seasonal components, and applies *Extreme Studentized Deviate* (ESD)[3] to discover anomalies. The main difference between S-ESD and S-H-ESD is that S-H-ESD uses the median and median absolute deviation (MAD), whereas S-ESD uses the mean and deviation which are vulnerable to a large amount of outliers.[5] In regard to data sets with a sizable number of outliers, S-H-ESD method may be more appropriate. This technique can be implemented with the **R** package **AnomalyDetection** from GitHub.

5.2 Evaluation of the HOT SAX algorithm

As seen in chapter 5.1, the aggregated result of the anomaly detection using the HOT SAX algorithm can be used for evaluating. In contrast to point anomalies or contextual anomalies,

[3] For further information on ESD, please refer to [5].

shape anomalies consist of several data points. In other words, a set of data points is considered as an anomaly. Hence, the aggregated result is slightly distinct from the results in Chapter 5.1.

Aggregated Result of Anomaly Detection using the HOT SAXl					
Window Size 15					
# Data Sets	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
500	500	500	164	164	336
Window Size 30					
# Data Sets	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
500	500	500	31	31	469
Window Size 50					
# Data Sets	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
500	500	500	9	9	491
Window Size 70					
# Data Sets	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
500	500	500	34	34	466
Window Size 100					
# Data Sets	**# Anomalies**	**# Declared**	**# False Neg**	**# False Posi**	**#True Posi**
500	500	500	292	292	208

Table 5: The aggregated Result of Anomaly Detection using the HOT SAX model. There are 500 time series data sets which include respectively 300 data points. Column 1 describes the time series data sets in total and column 2 the number of true anomalies. Column 3 indicates anomaly-defined data points, column 4 falsely not detected anomalies, column 5 falsely anomaly-defined data points and column 6 correctly detected anomalies.

Each time series data set was designed to have a single shape anomaly that has a length of 15, and the HOT SAX algorithm always declares at least one discord, since it computes all possible distances between subsequences. Hence, unlike ARIMA models, the number of declared anomalies corresponds exactly to the number of true anomalies. Additionally, due to the existence of a single discord, the number of false negatives is equal to the number of false positives. When a data point is improperly defined as a discord, automatically the true discord is left not to be exposed. Consequently, it causes the equivalence of the recall and precision rate. With a window size of 15, the precision and recall rate are equal to 67.2%. Adjusting the window size to 50, the precision and recall rate become 98.2%, which is the best result in this paper. When setting the window size to 100, the precision and recall rate decrease to 41.6%, which represents the

worst case in this paper. Since F_1 indicates the average of precision and recall, F_1 has no need to be considered in this case. Window-based anomaly detection such as the HOT SAX algorithm produces different results depending on the window size. Hence, many researchers are trying to design window-size-independent anomaly detection algorithms. For example, Yang and Liao [16] introduced the *Adjacent Mean Difference (AMD) segmentation* method, which segments a data set without considering parameters and slides windows with different lengths. Although this paper will not analyze the AMD method, such an approach could be a milestone towards better shape anomaly detection techniques.

6 Conclusion

Even though many unsupervised models have been devised and are in use in order to investigate anomalies, there is no perfect technique that can detect all types of anomalies in reality. According to the types of anomalies different methods were examined and accessed in this paper.
One of the commonly used methods to detect contextual anomalies is the prediction-based model using ARIMA. The main principle of ARIMA models is built on comparing actual data points and predicted values. When an actual data point is bigger or smaller than a predicted value and threshold, the data point is declared as an anomaly. In order to implement the method using ARIMA models, the **R** package **tsoutliers** was adopted. It shows fairly good performances though, further tests with the time series involving multiple seasonalities need to be accomplished in the future work.
In case of detecting collective anomalies, the HOT SAX algorithm is applicable. The idea of this algorithm arose from transforming time series data into a discrete form. The result of the HOT SAX algorithm is very sensitive to the length of the window though, it returns satisfactory outcomes, once a user discovers the proper window size. The HOT SAX algorithm is executable in the **R** package **jmotif**.
In order to invent more robust and outlier-type-independent methods, an ensemble of various anomaly detection methods is worth consideration in future works. As proposed by Aggarwal [1], a combination of supervised and unsupervised techniques is also expectable.

Bibliography

[1] C. C. Aggarwal. *Outlier Analysis Second Edition*. Springer, 2016. ISBN 978-3-319-47577-6.

[2] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[3] C. Chen and L.-M. Liu. Joint estimation of model parameters and outlier effects in time series. *Journal of the American Statistical Association*, 88(421):284–297, 1993.

[4] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[5] J. Hochenbaum, O. S. Vallis, and A. Kejariwal. Automatic anomaly detection in the cloud via statistical learning. *arXiv preprint arXiv:1704.07706*, 2017.

[6] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[7] javlacalle (https://stats.stackexchange.com/users/48766/javlacalle). Detecting outliers in time series (ls/ao/tc) using tsoutliers package in r. how to represent outliers in equation format? Cross Validated. URL `https://stats.stackexchange.com/q/104946`. URL:https://stats.stackexchange.com/q/104946 (version: 2014-06-26).

[8] R. Kaiser and A. Maravall. Seasonal outliers in time series. 1999.

[9] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. Ieee, 2005.

[10] E. Keogh, J. Lin, S.-H. Lee, and H. Van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11(1): 1–27, 2007.

[11] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

[12] J. López-de Lacalle. tsoutliers r package for detection of outliers in time series. *CRAN, R Package*, 2016.

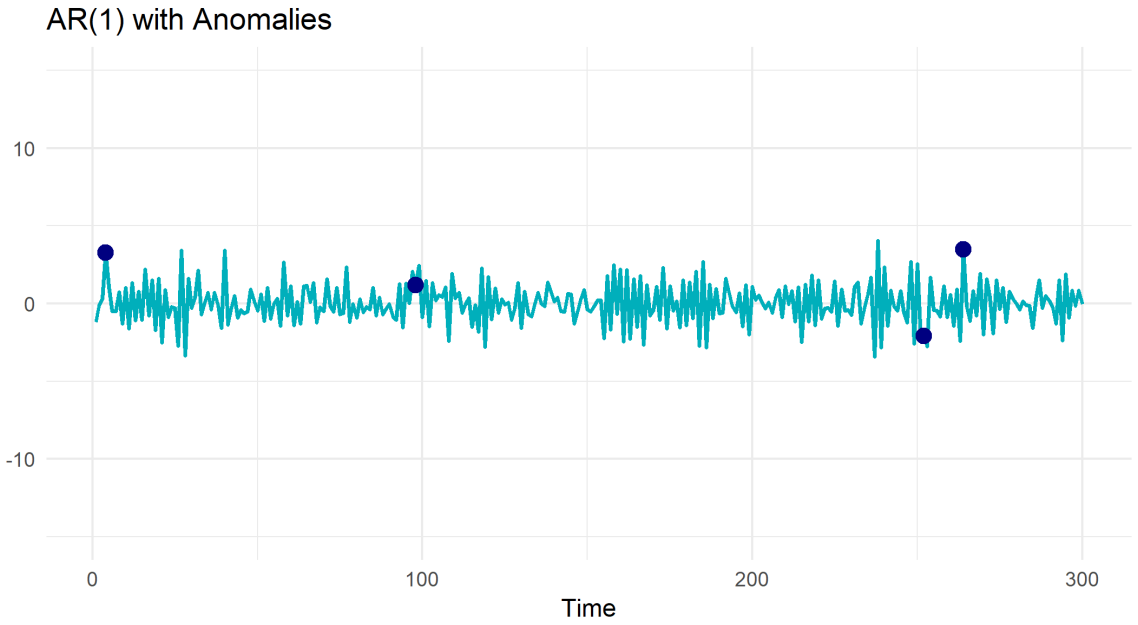
[13] N. Perlroth. Hackers used new weapons to disrupt major websites across us. *The New York Times*, 21, 2016.

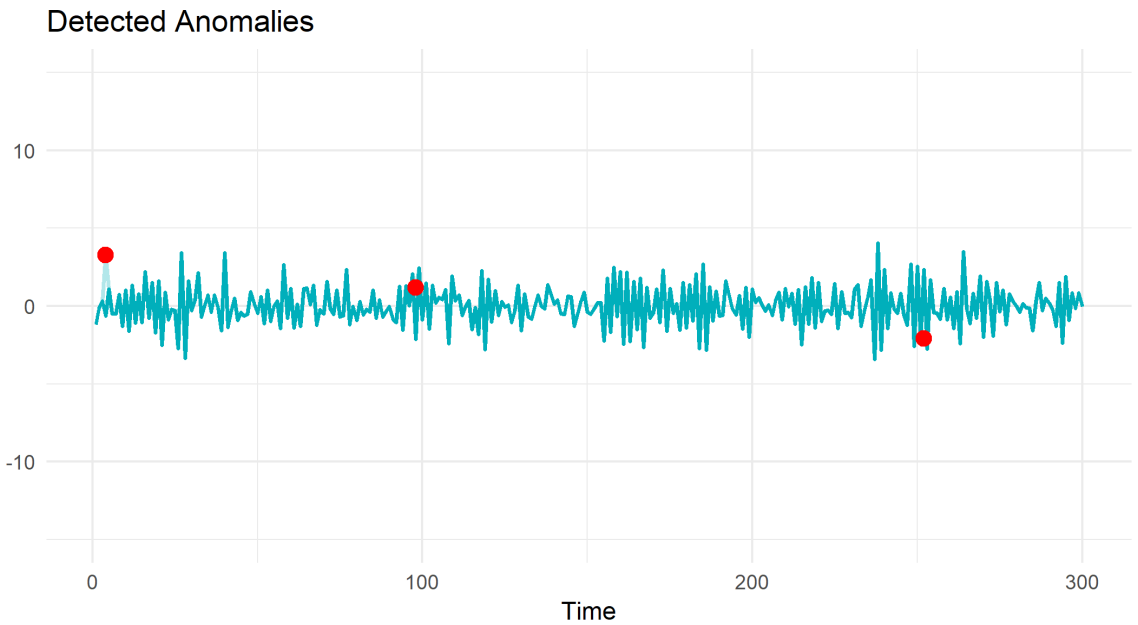
[14] R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications: with R examples*. Springer, 2017.

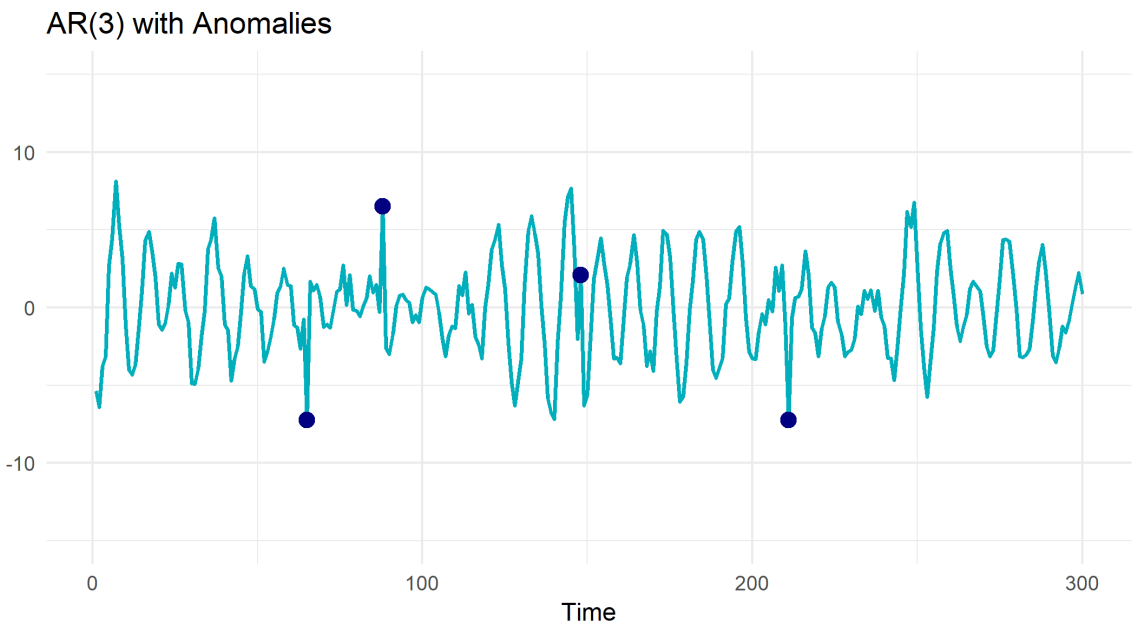
[15] S. J. Taylor and B. Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

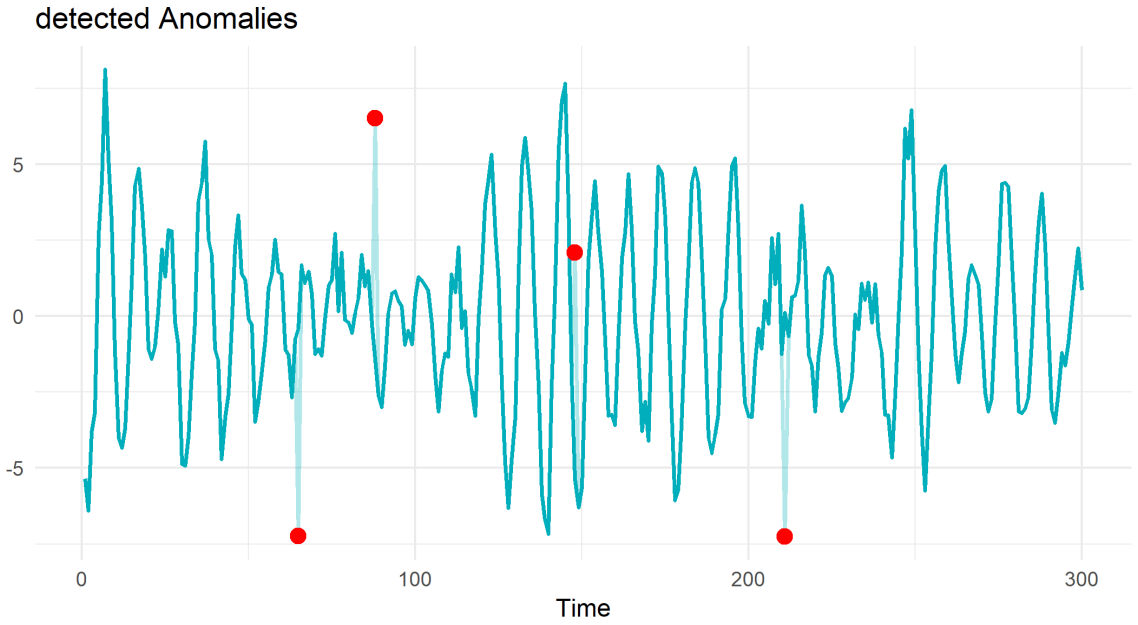
[16] C.-L. Yang and W.-J. Liao. Adjacent mean difference (amd) method for dynamic segmentation in time series anomaly detection. In *2017 IEEE/SICE International Symposium on System Integration (SII)*, pages 241–246. IEEE, 2017.

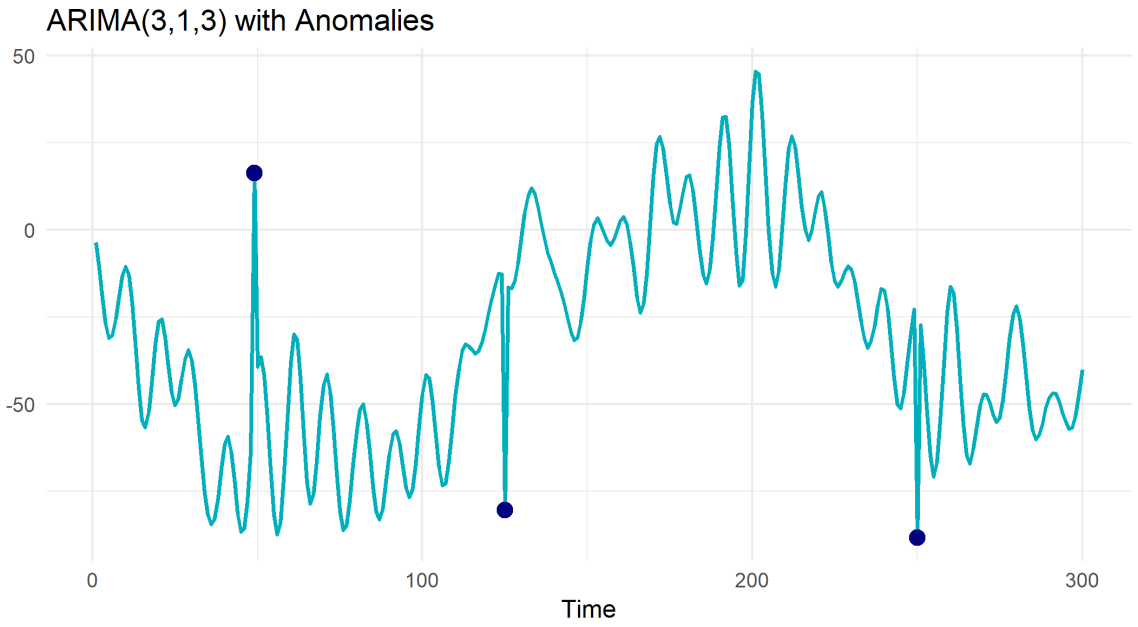
A Appendix: Detecting Contextual Anomalies using ARIMA

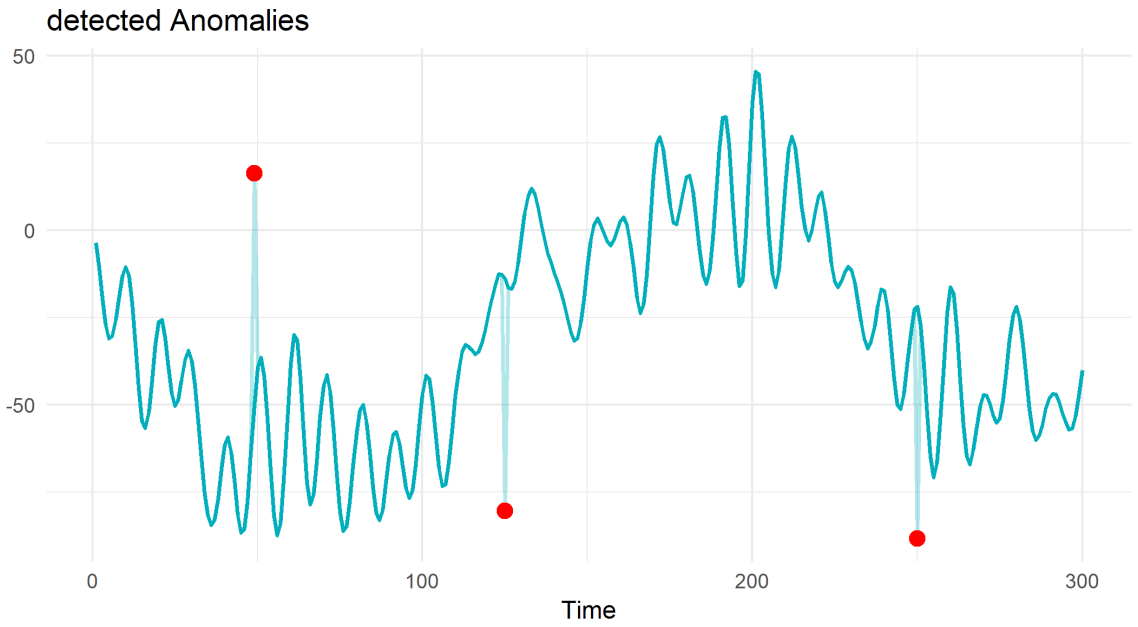


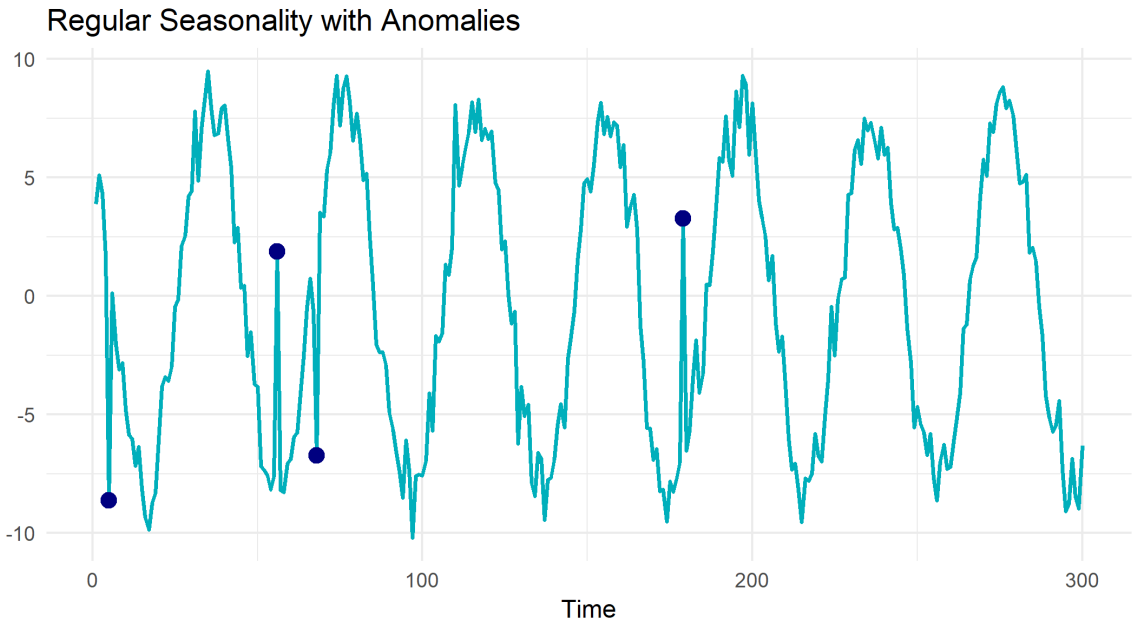



AR(3) with Anomalies
10
0
-10
0
100
200
300
Time

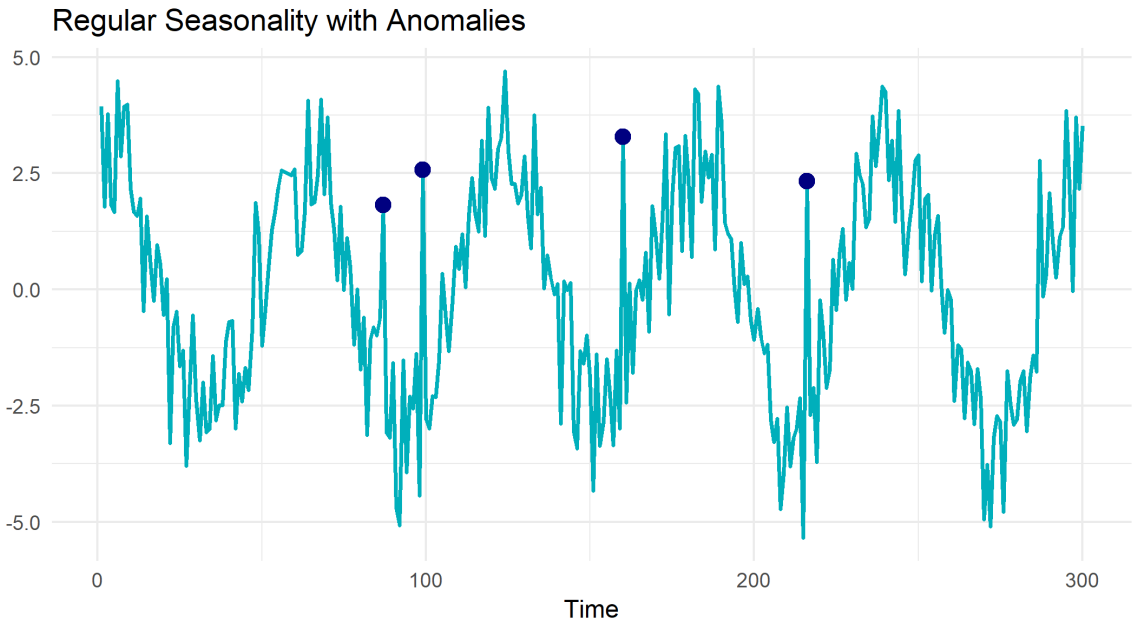

detected Anomalies
5
0
-5
0
100
200
300
Time

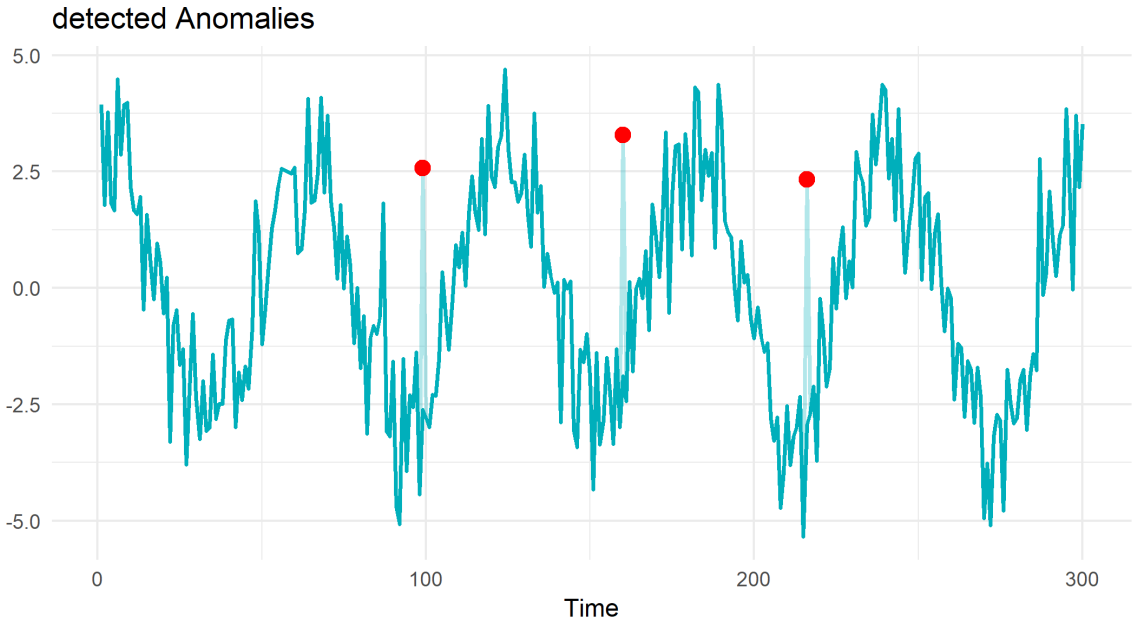

ARIMA(3,1,3) with Anomalies
50
0
-50
0
100
200
300
Time

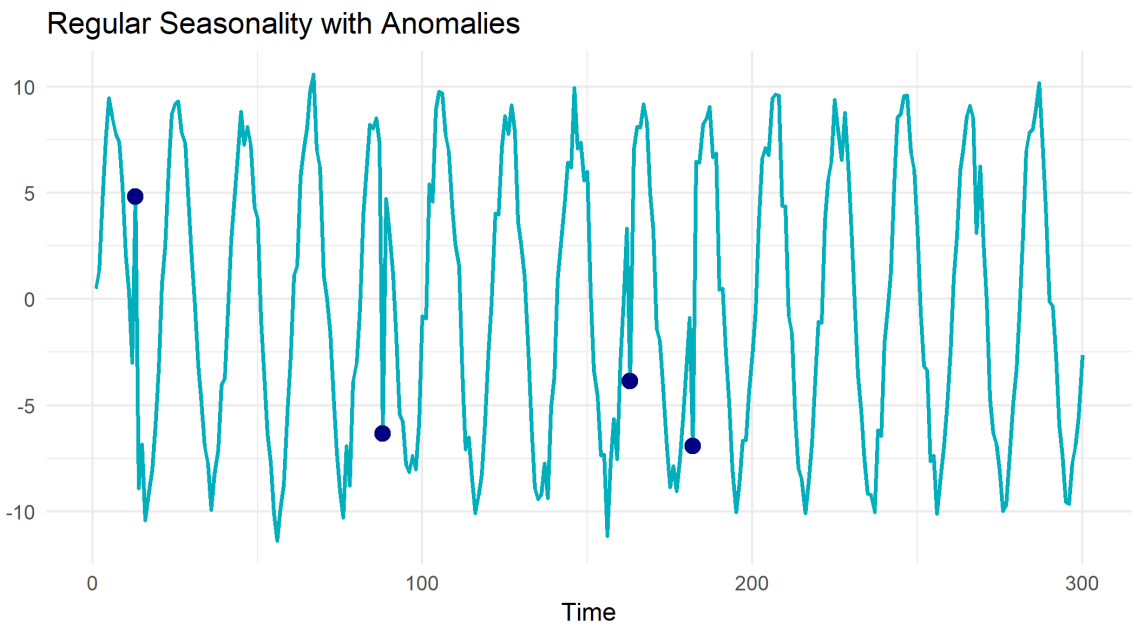

detected Anomalies
50
0
-50
0
100
200
300
Time

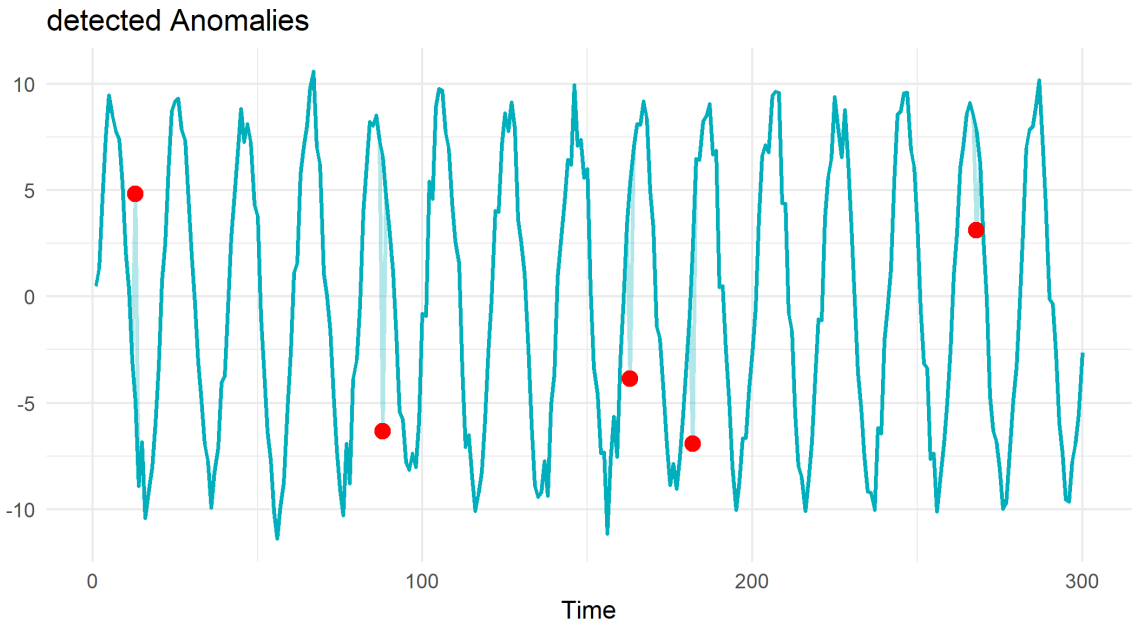

Regular Seasonality with Anomalies
10
5
0
-5
-10
0
100
200
300
Time

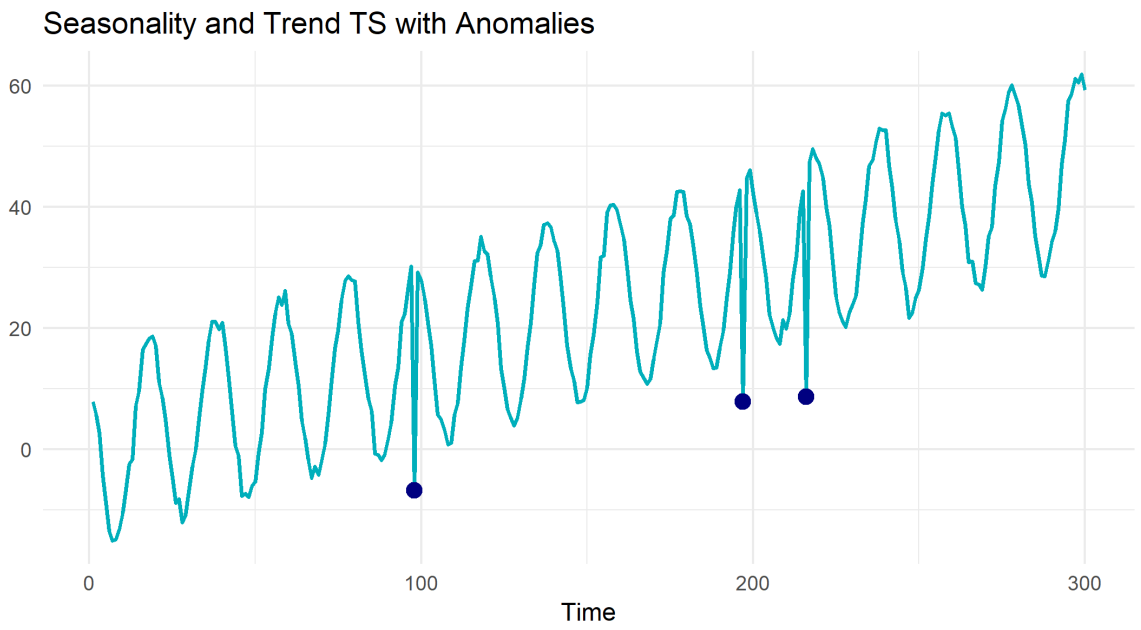

detected Anomalies
10
5
0
-5
-10
0
100
200
300
Time

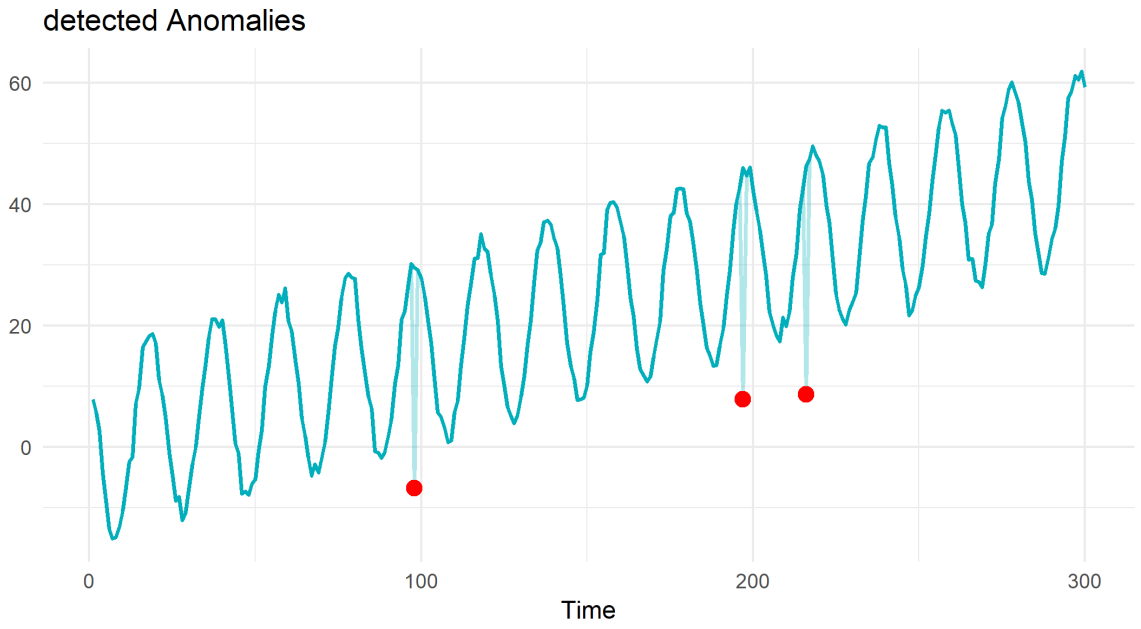

Regular Seasonality with Anomalies
5.0
2.5
0.0
-2.5
-5.0
0
100
200
300
Time

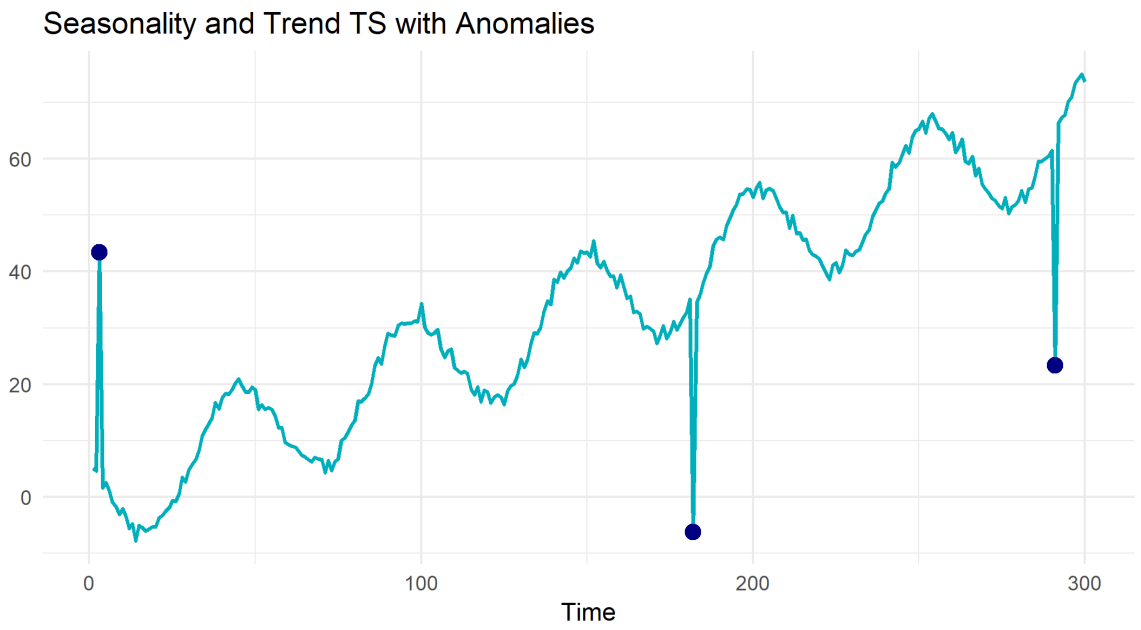

detected Anomalies
5.0
2.5
0.0
-2.5
-5.0
0
100
200
300
Time

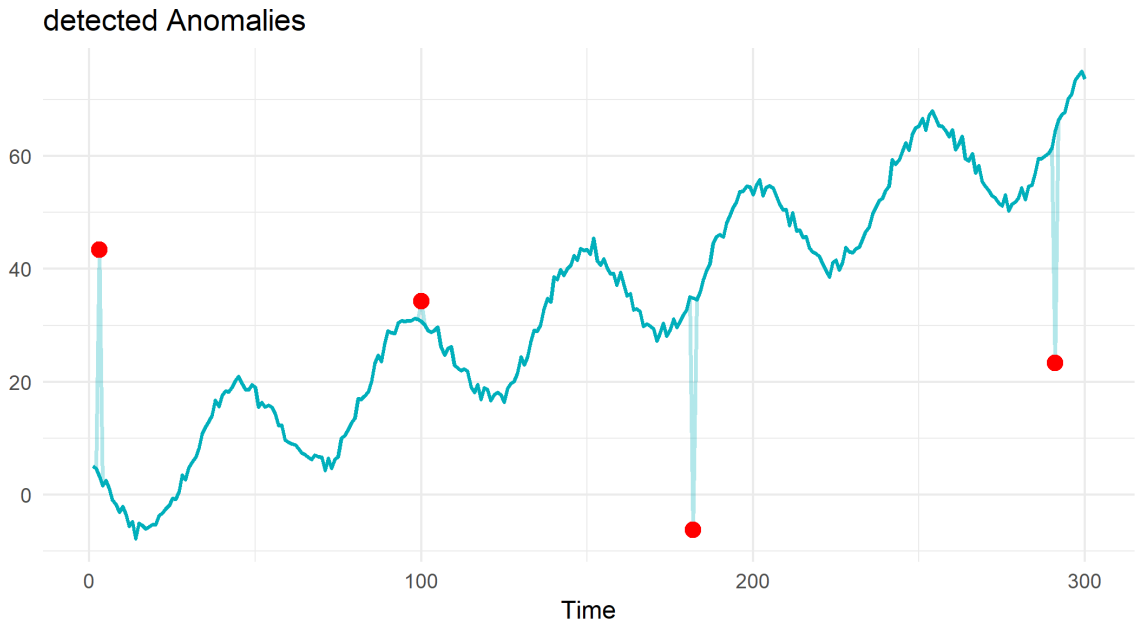

Regular Seasonality with Anomalies
10
5
0
-5
-10
0
100
200
300
Time

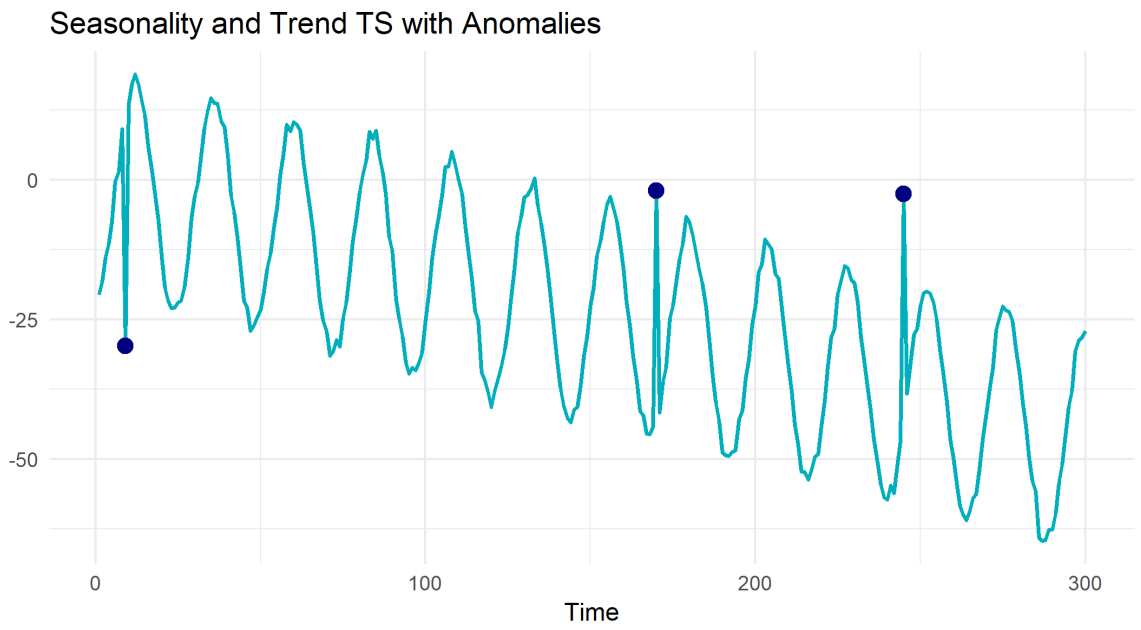

detected Anomalies
10
5
0
-5
-10
0
100
200
300
Time

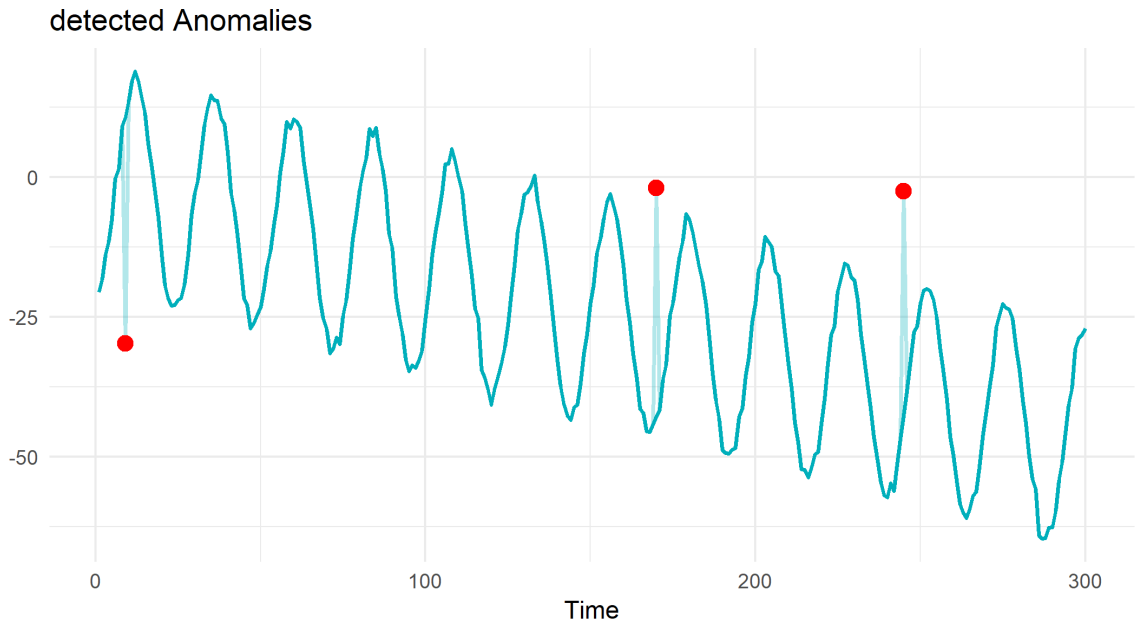

Seasonality and Trend TS with Anomalies
60
40
20
0
0
100
200
300
Time

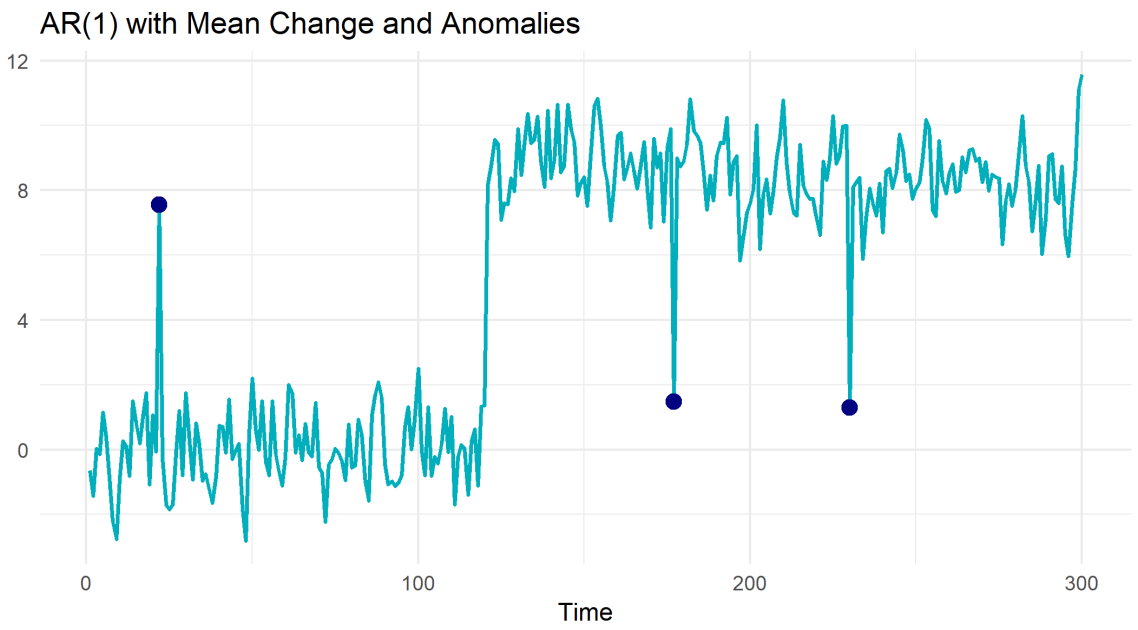

detected Anomalies
60
40
20
0
0
100
200
300
Time

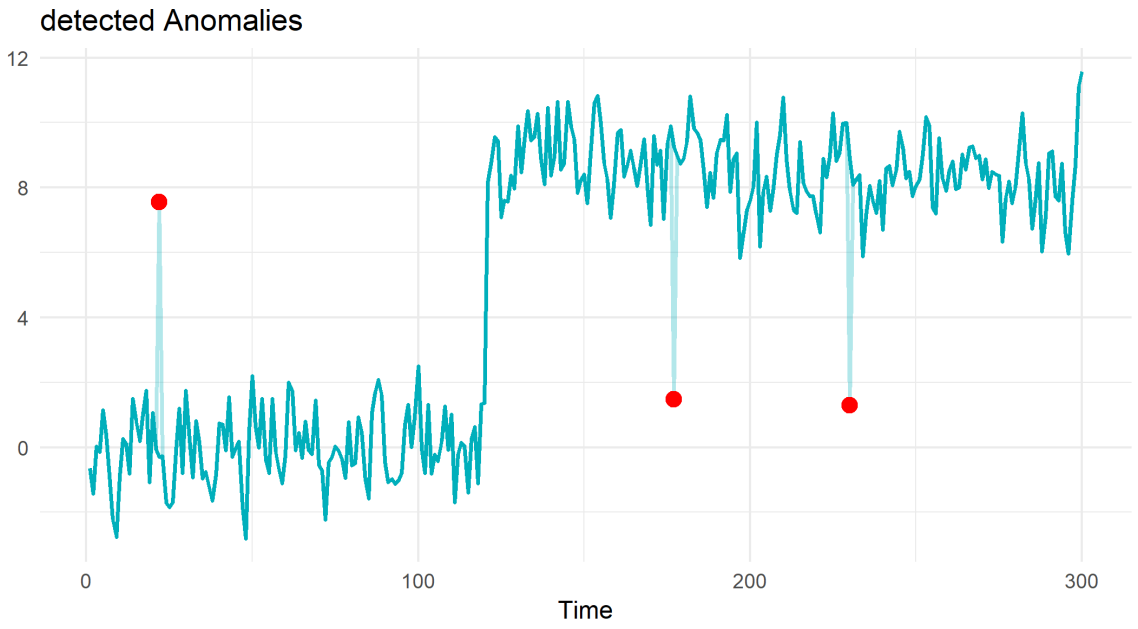

Seasonality and Trend TS with Anomalies
60
40
20
0
0
100
200
300
Time


detected Anomalies
60
40
20
0
0
100
200
300
Time

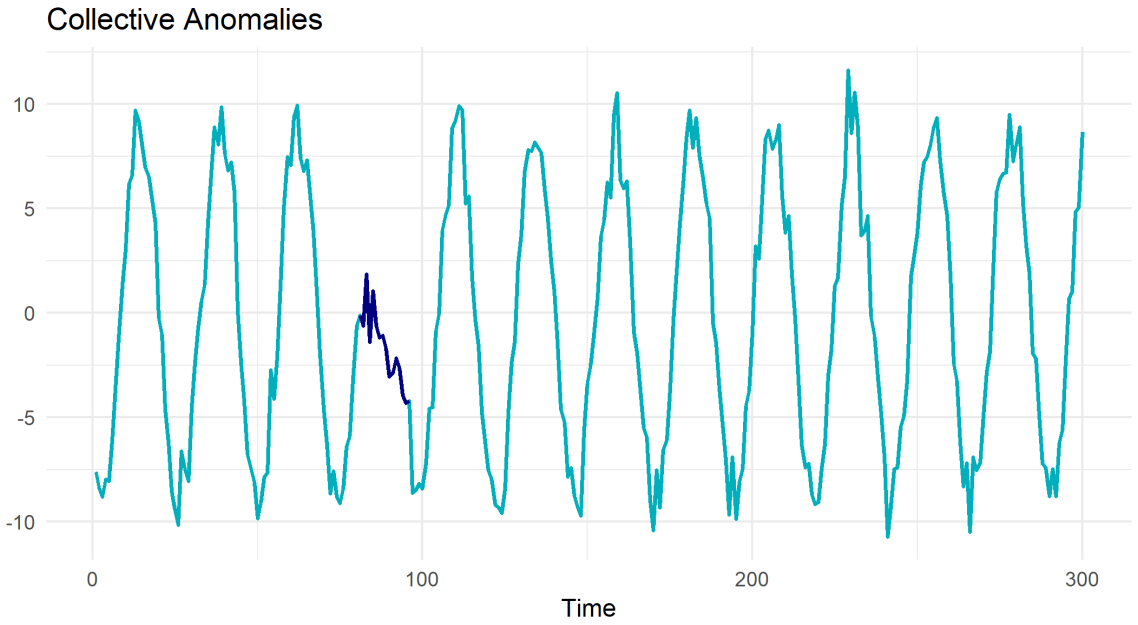

Seasonality and Trend TS with Anomalies
0
-25
-50
0
100
200
300
Time

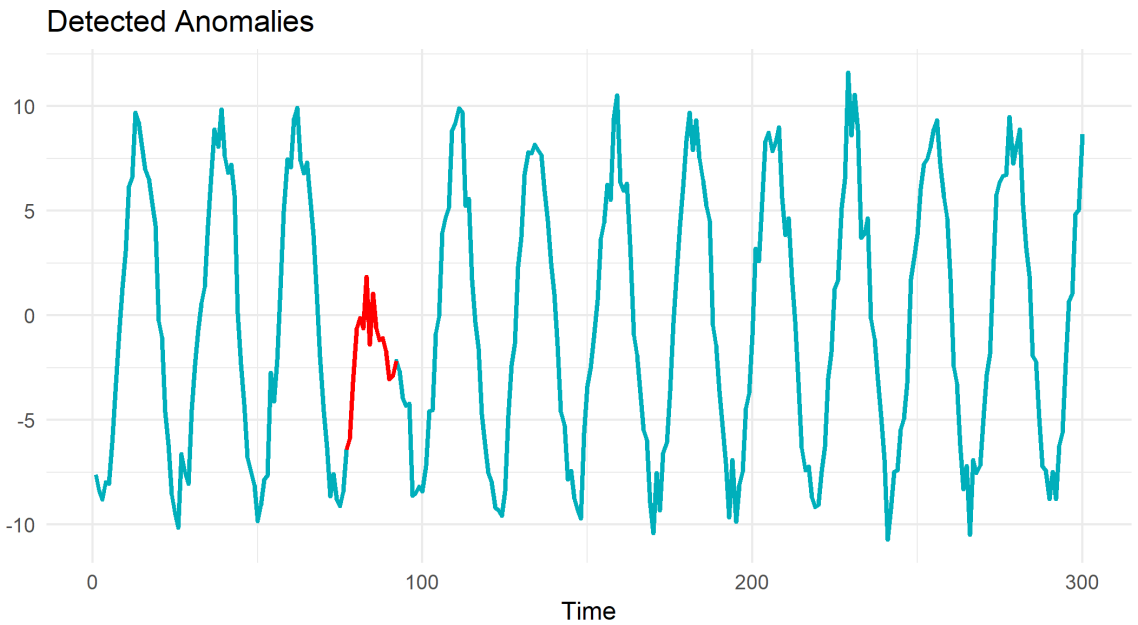

detected Anomalies
0
-25
-50
0
100
200
300
Time

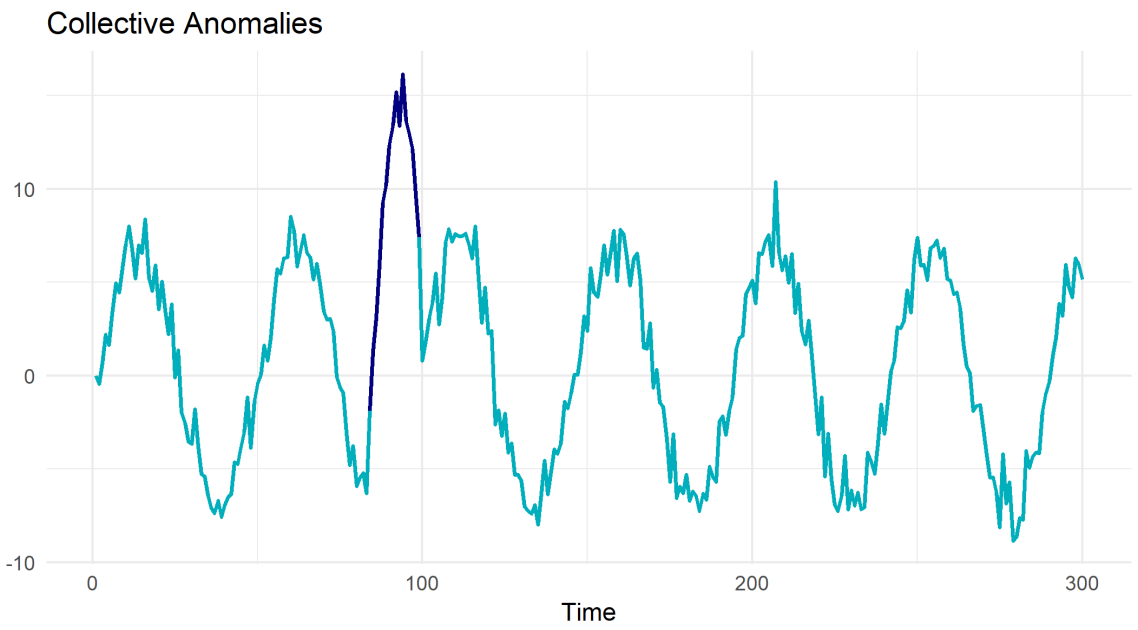

AR(1) with Mean Change and Anomalies
12
8
4
0
0
100
200
300
Time

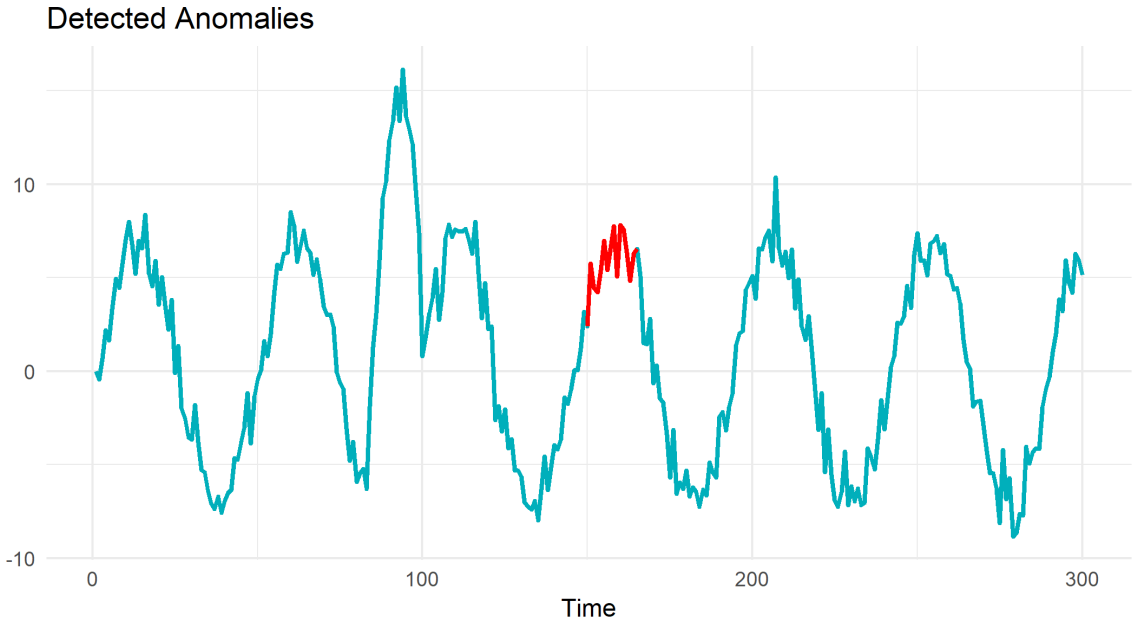

detected Anomalies
12
8
4
0
0
100
200
300
Time

B Appendix: The HOT SAX with Window Size 15

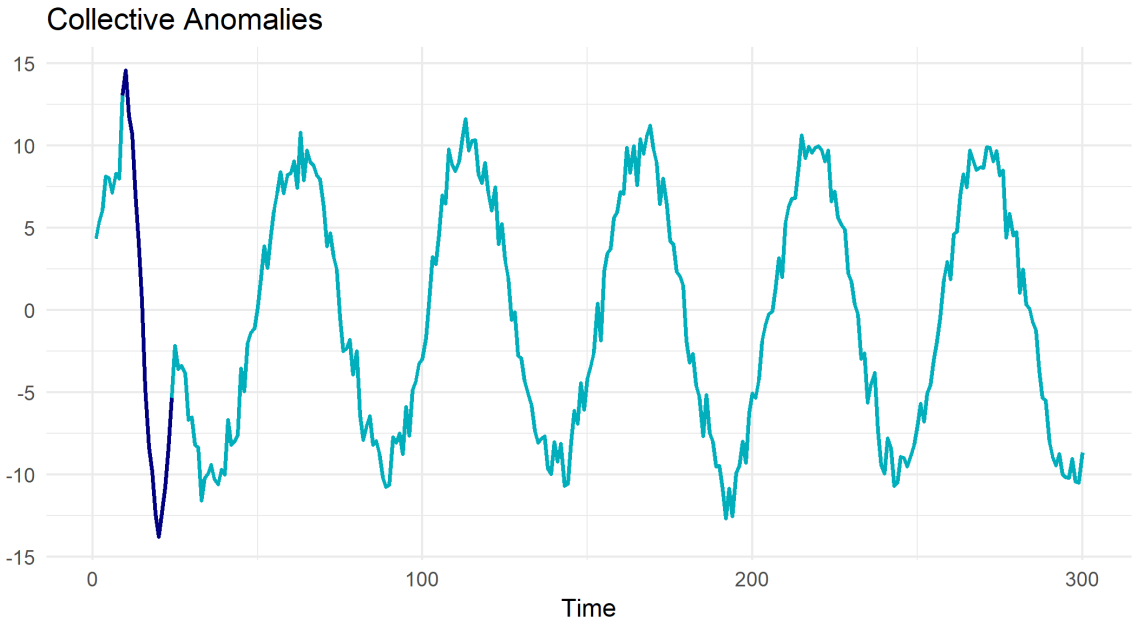


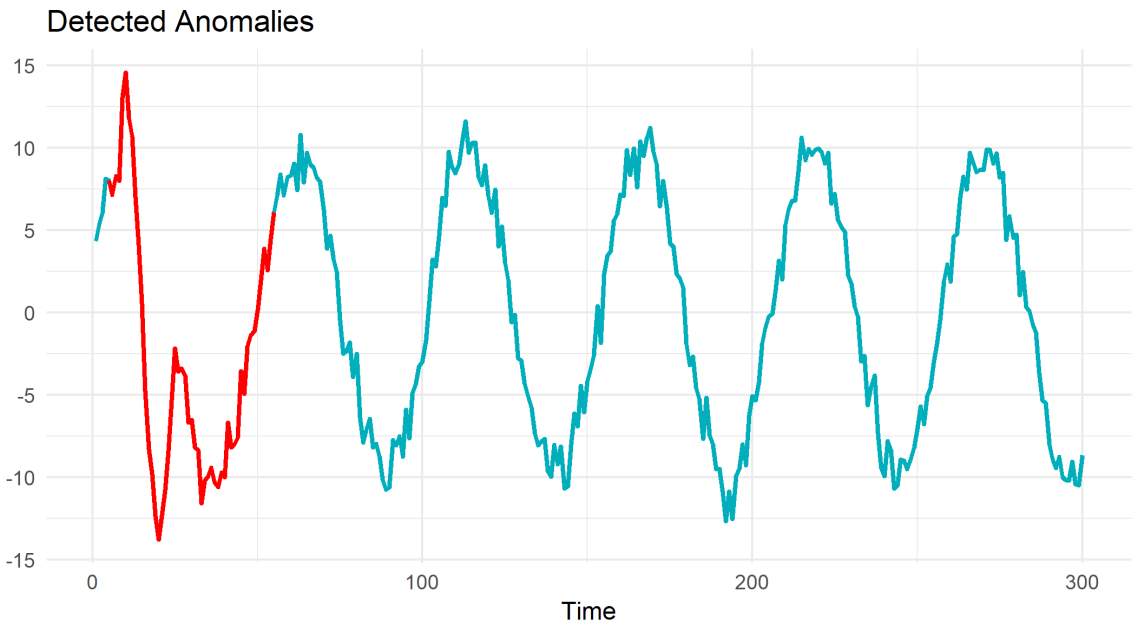


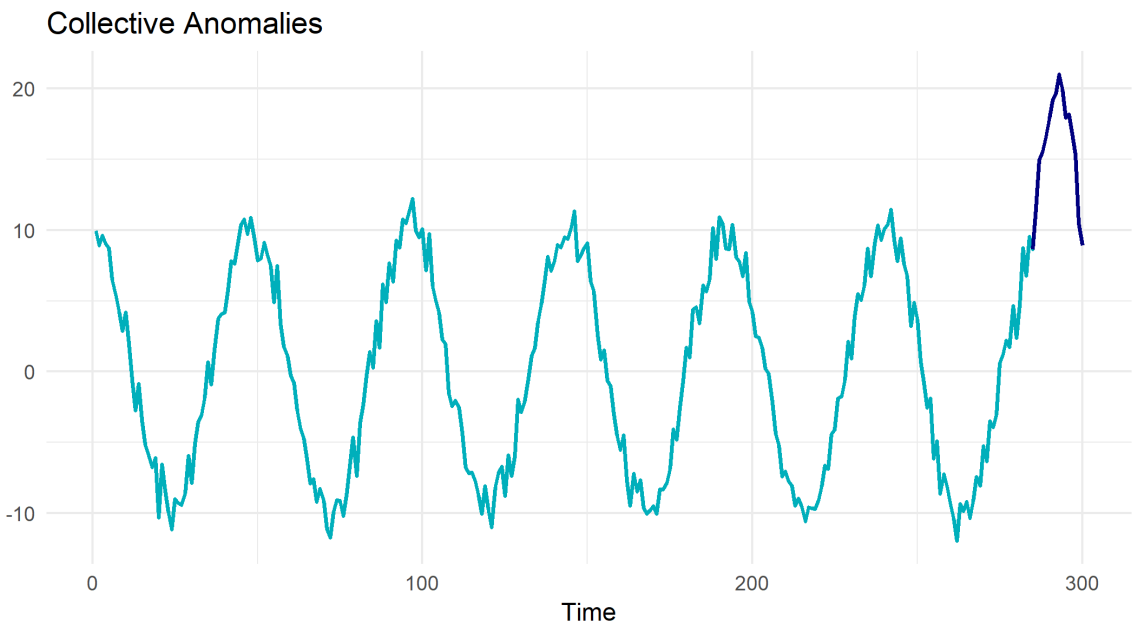

Collective Anomalies
10
0
-10
0
100
200
300
Time

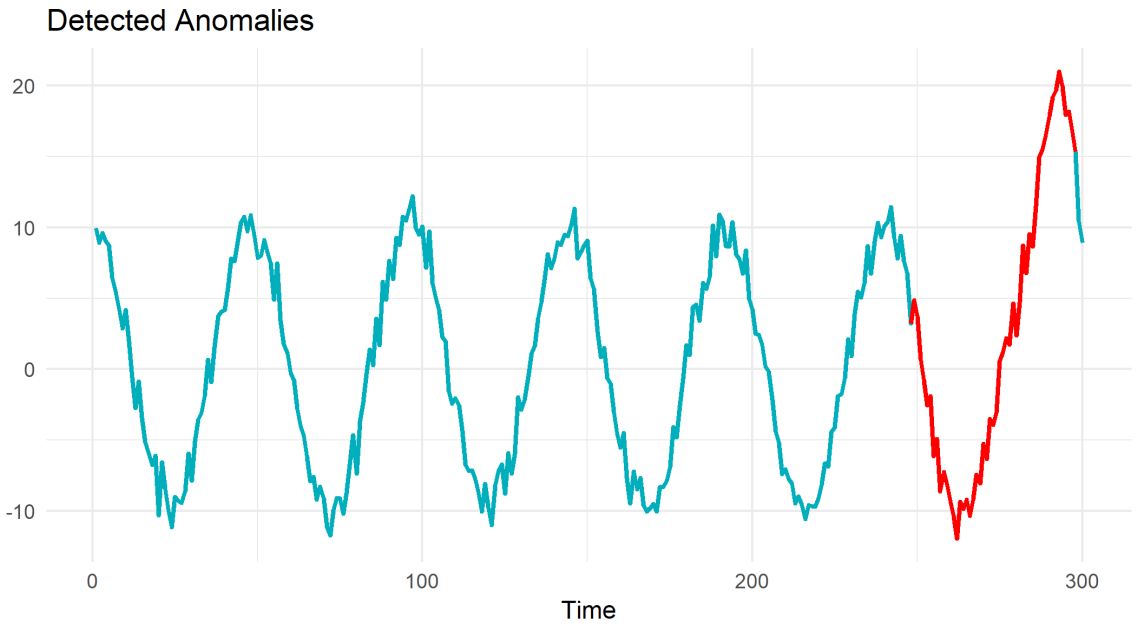

Detected Anomalies
10
0
-10
0
100
200
300
Time

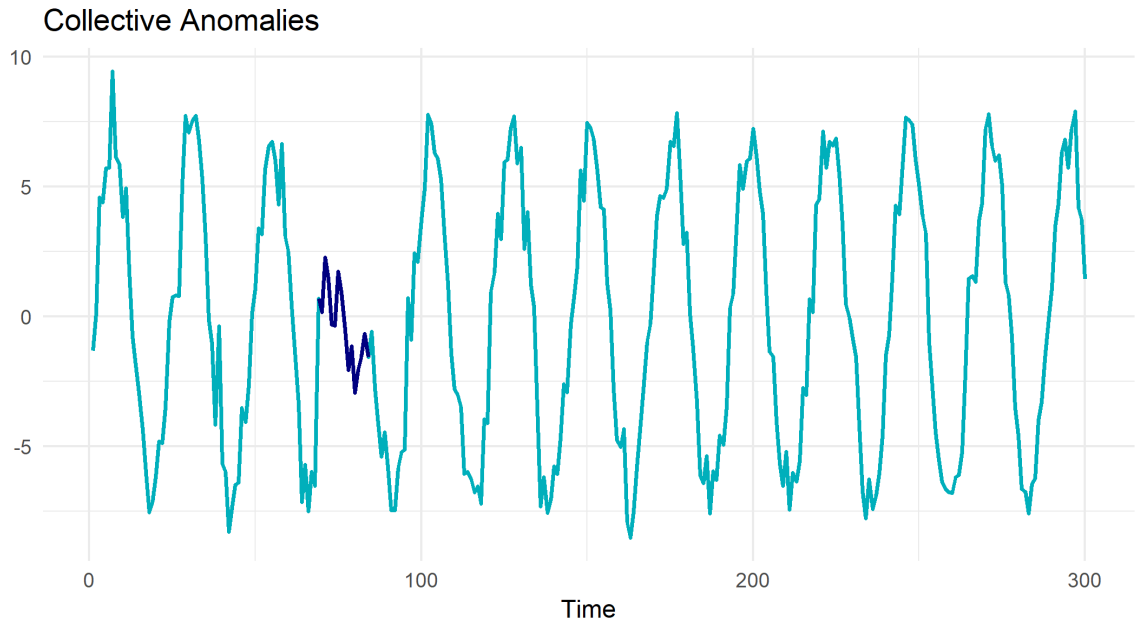
C Appendix: The HOT SAX with Window Size 50

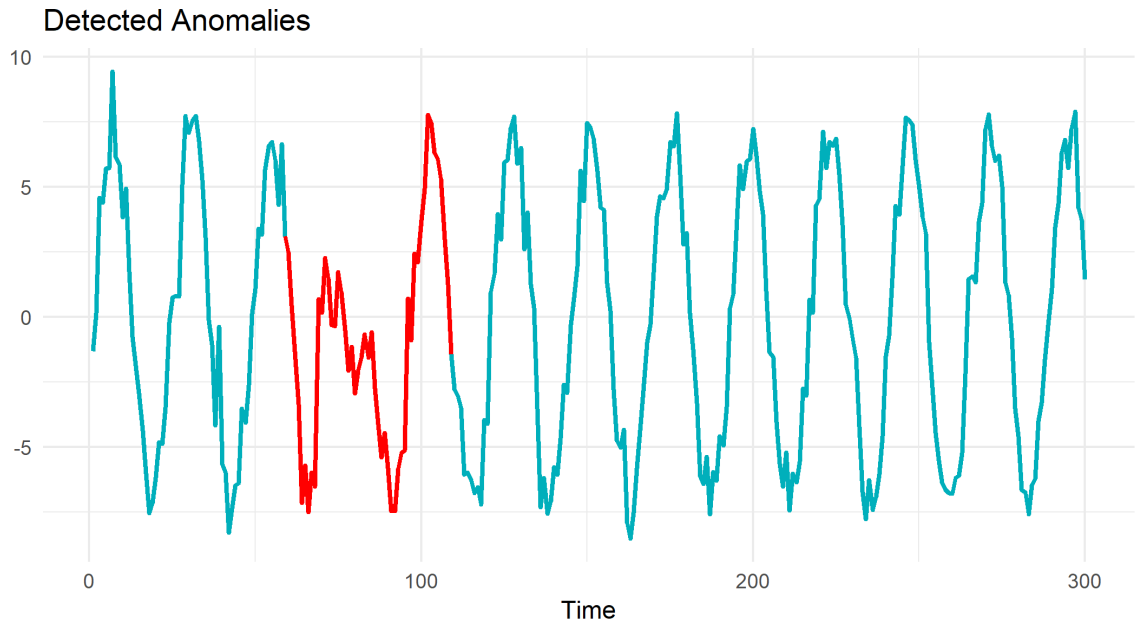


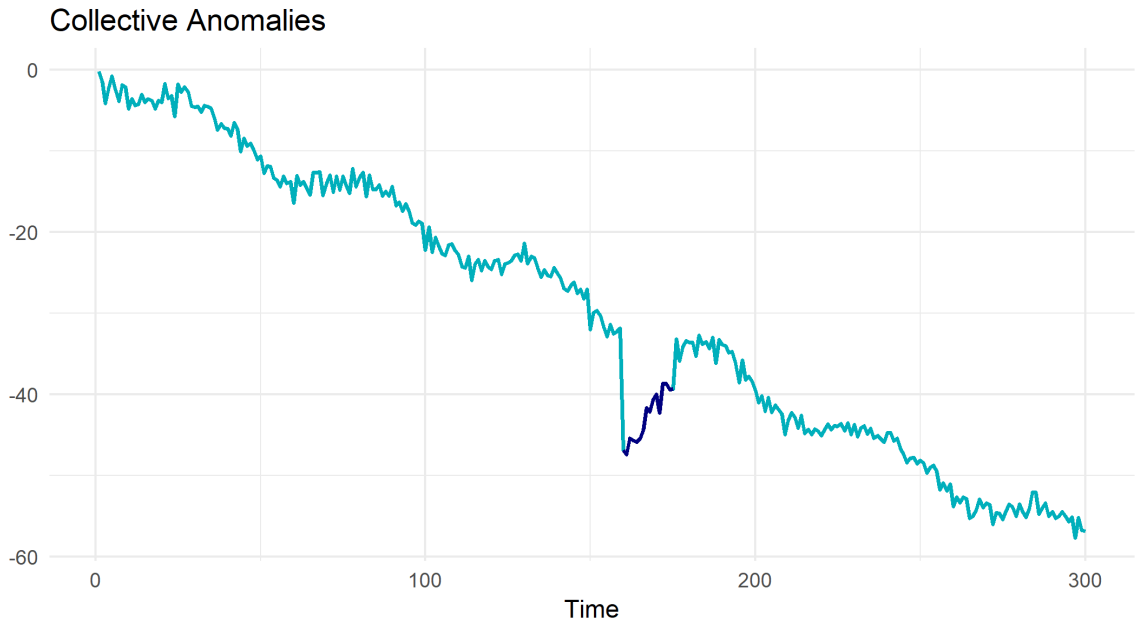


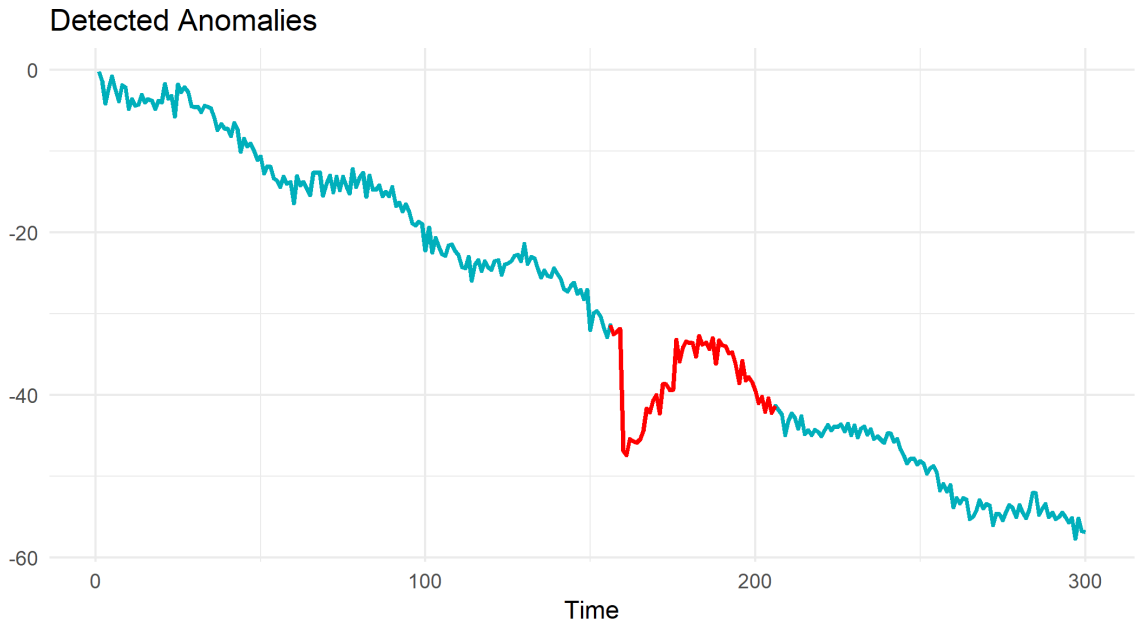

Collective Anomalies
20
10
0
-10
0
100
200
300
Time

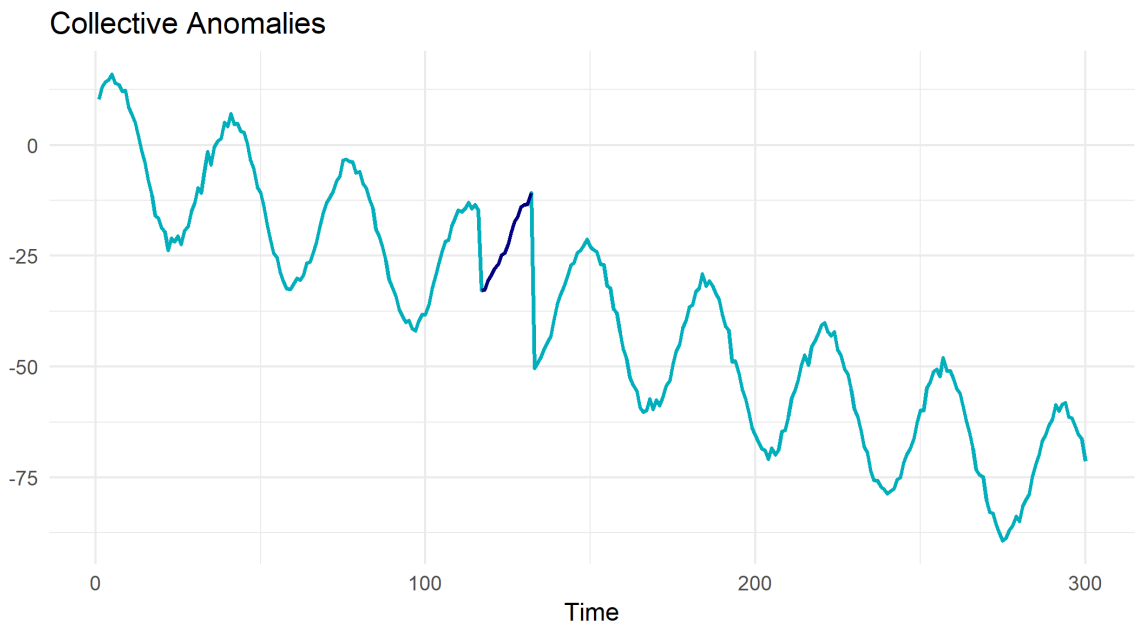

Detected Anomalies
20
10
0
-10
0
100
200
300
Time

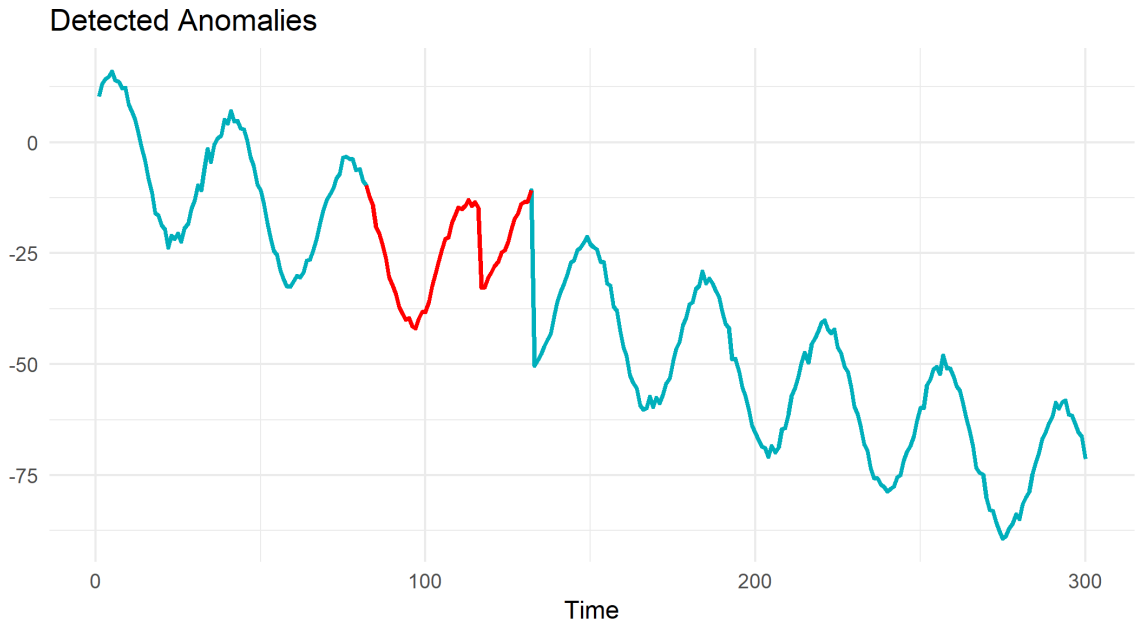

Collective Anomalies
10
5
0
-5
0
100
200
300
Time

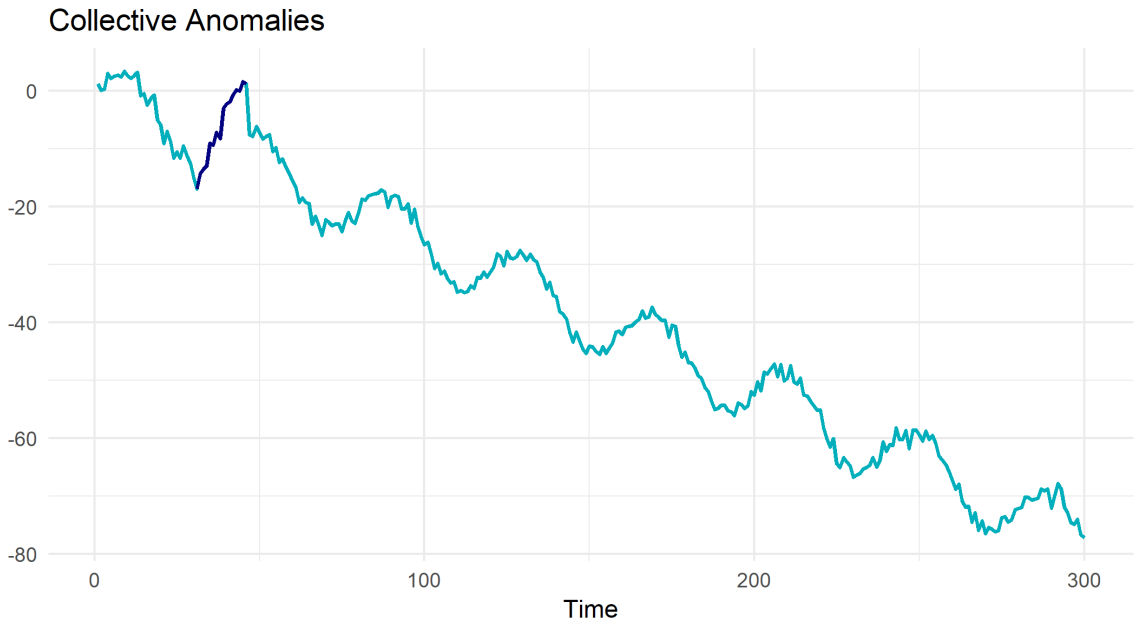

Detected Anomalies
10
5
0
-5
0
100
200
300
Time

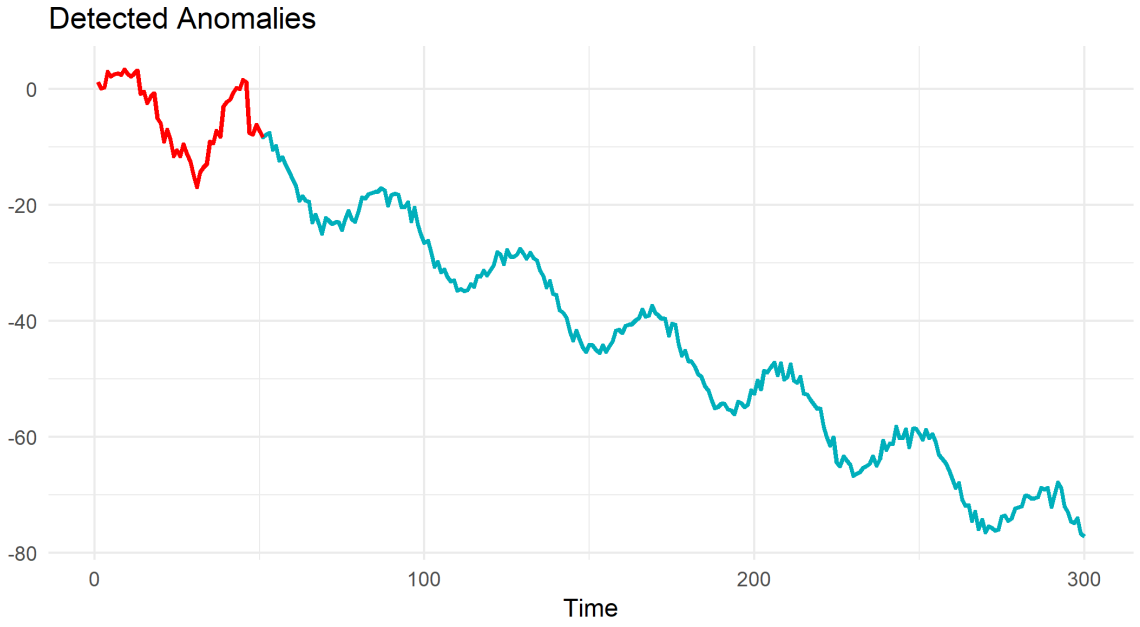

Collective Anomalies
0
-20
-40
-60
0
100
200
300
Time

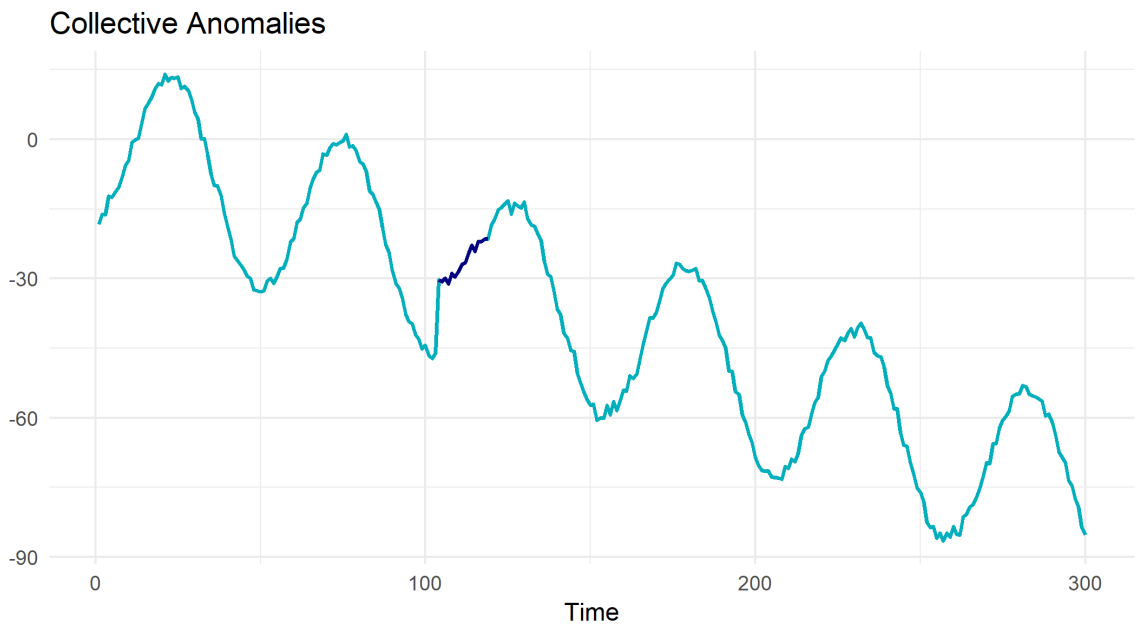

Detected Anomalies
0
-20
-40
-60
0
100
200
300
Time

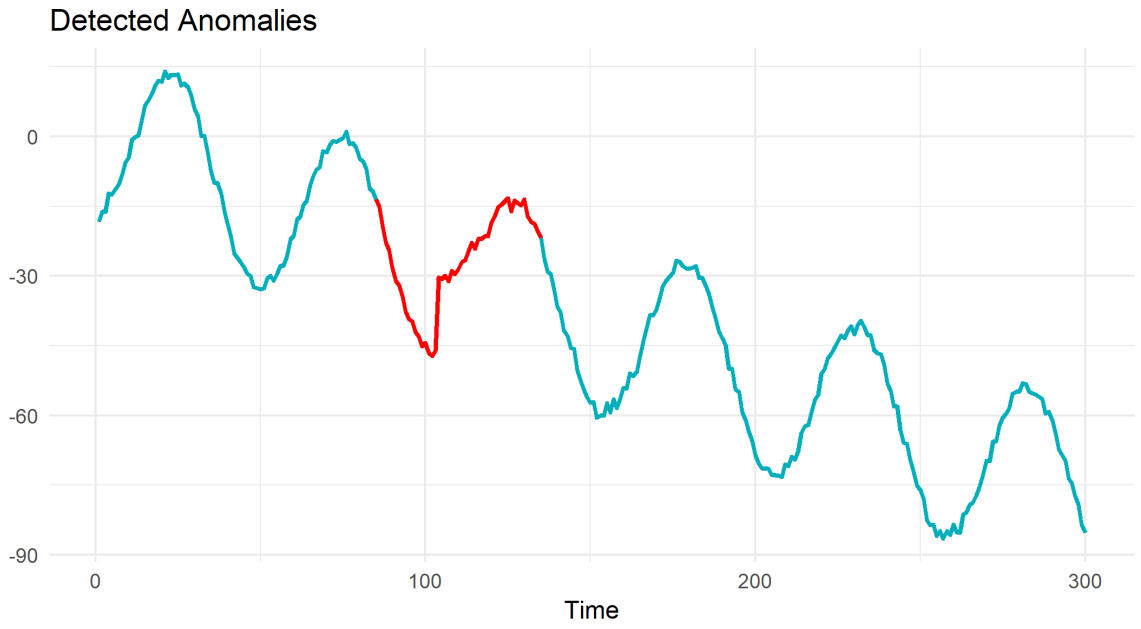

Collective Anomalies
0
-25
-50
-75
0
100
200
300
Time

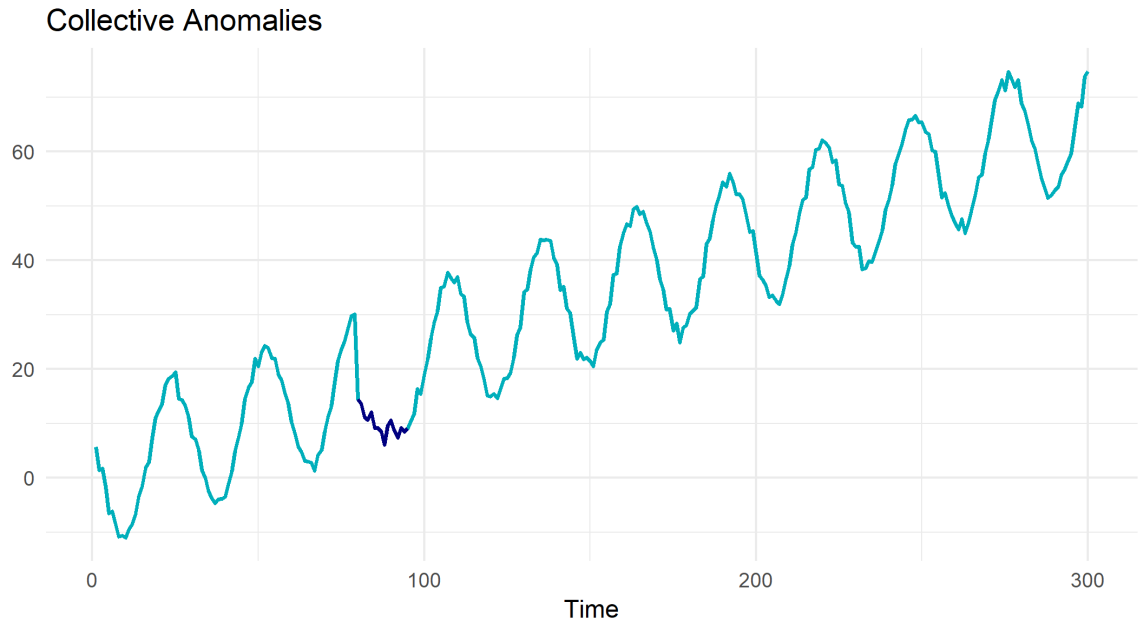

Detected Anomalies
0
-25
-50
-75
0
100
200
300
Time

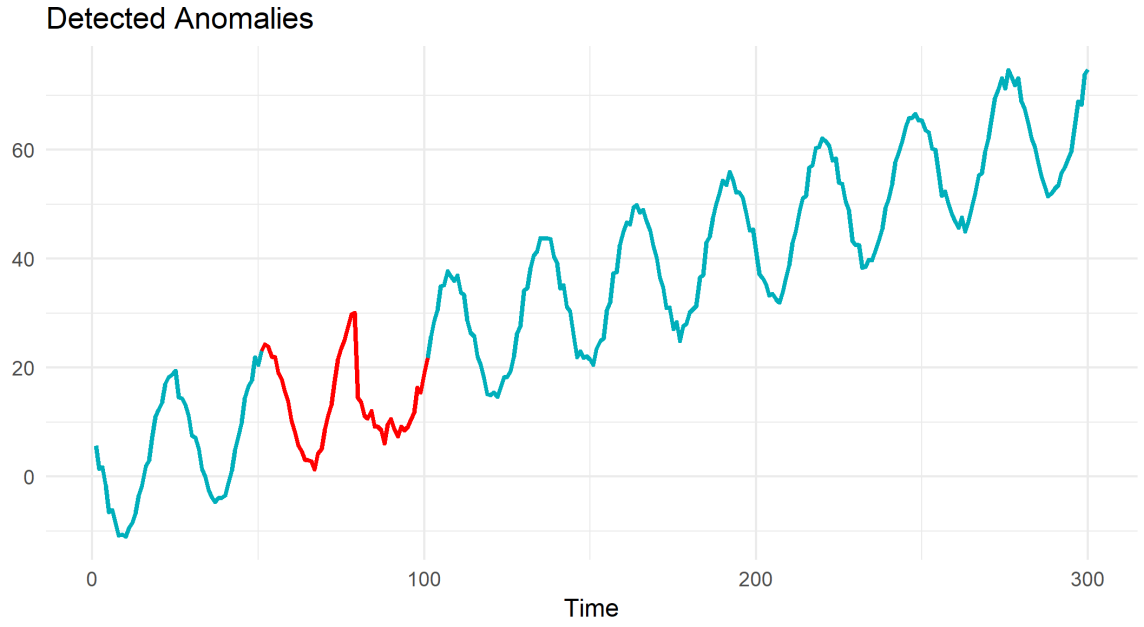

Collective Anomalies
0
-20
-40
-60
-80
0
100
200
300
Time

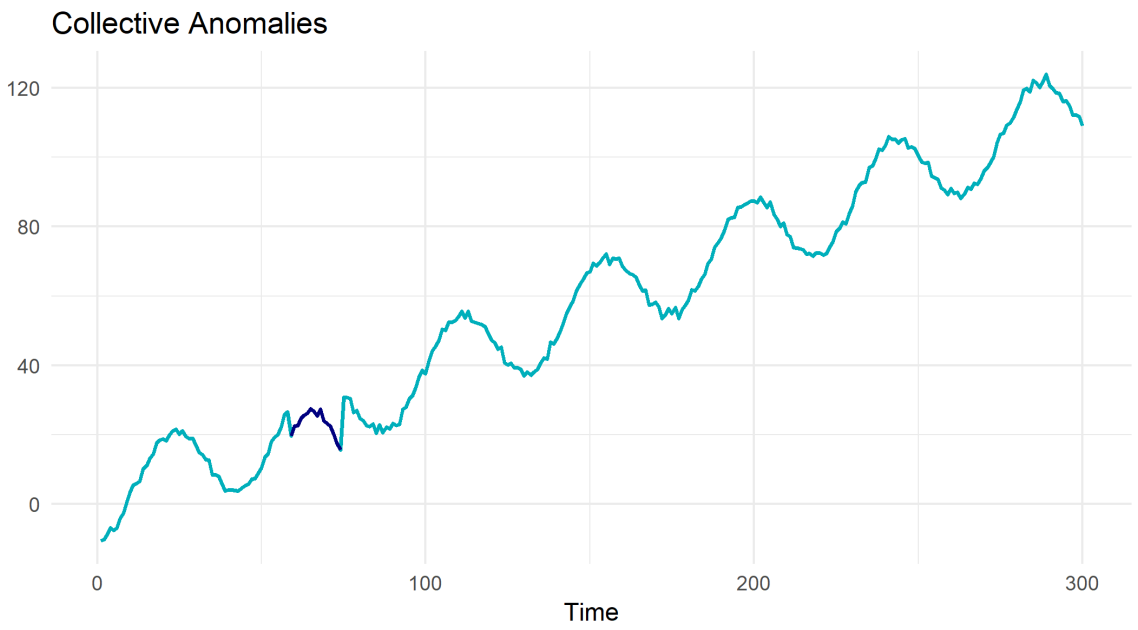

Detected Anomalies
0
-20
-40
-60
-80
0
100
200
300
Time

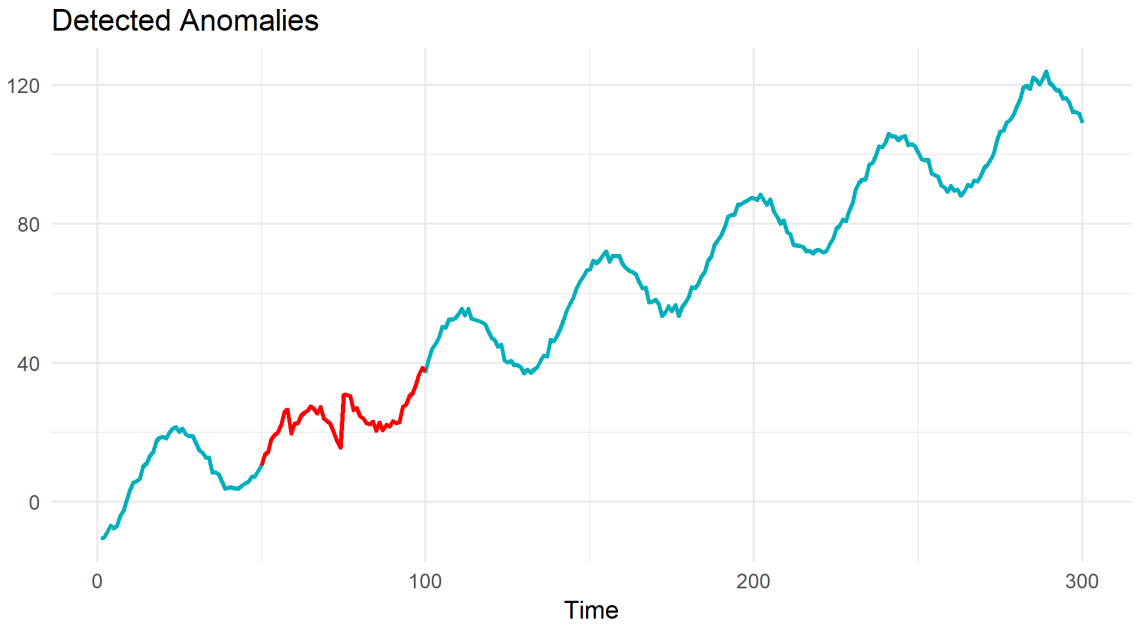

Collective Anomalies
0
-30
-60
-90
0
100
200
300
Time

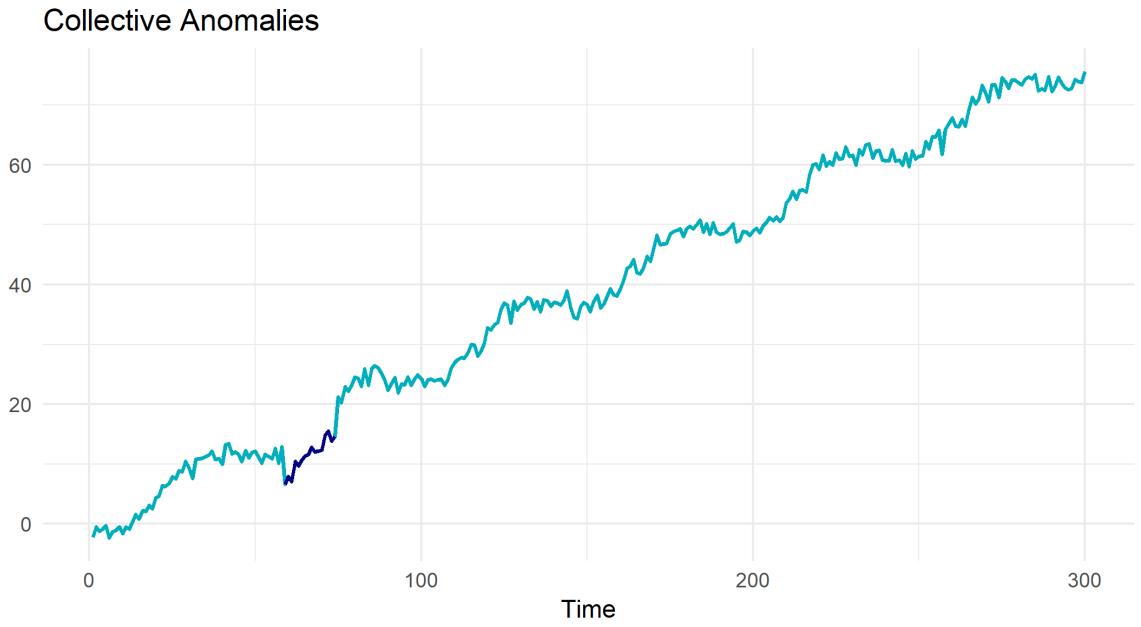

Detected Anomalies
0
-30
-60
-90
0
100
200
300
Time

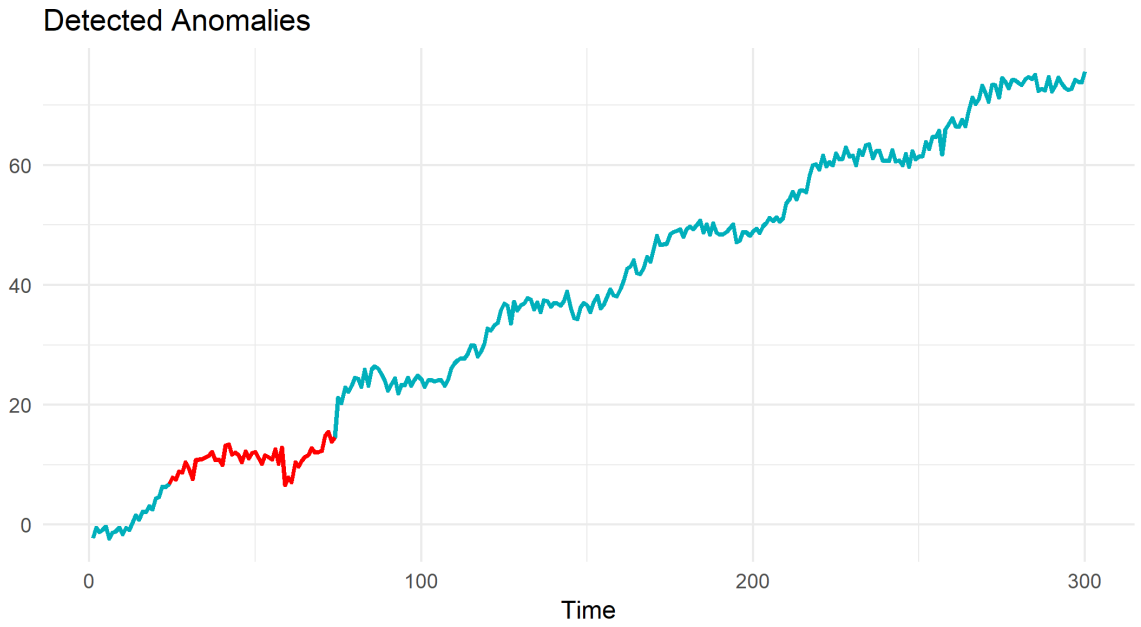

Collective Anomalies
60
40
20
0
0
100
200
300
Time


Detected Anomalies
60
40
20
0
0
100
200
300
Time

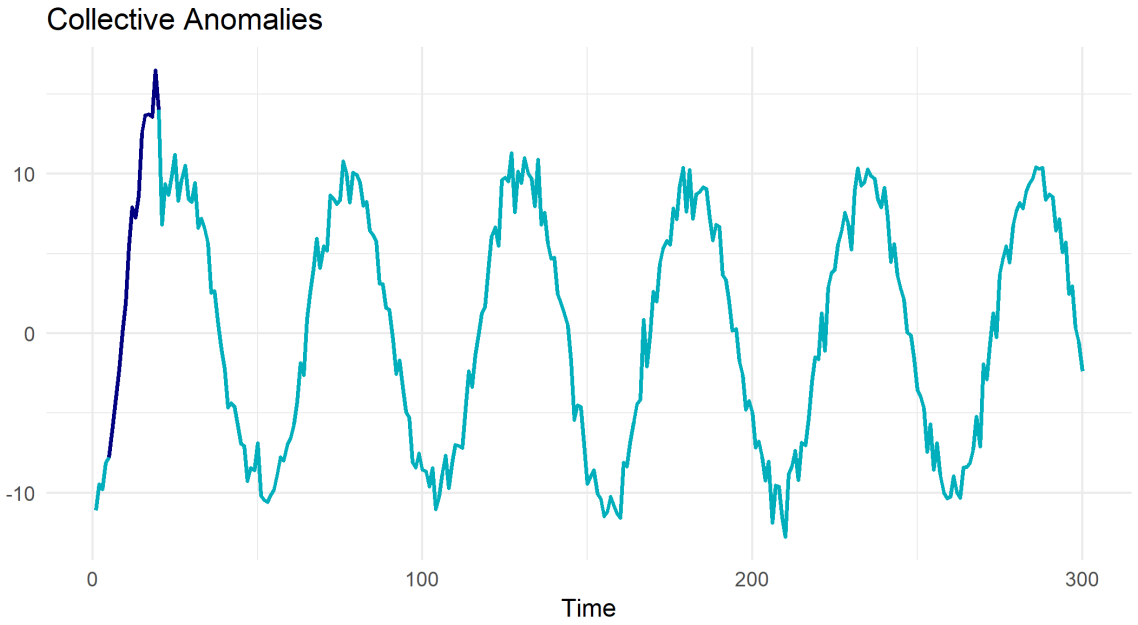

Collective Anomalies
120
80
40
0
0
100
200
300
Time

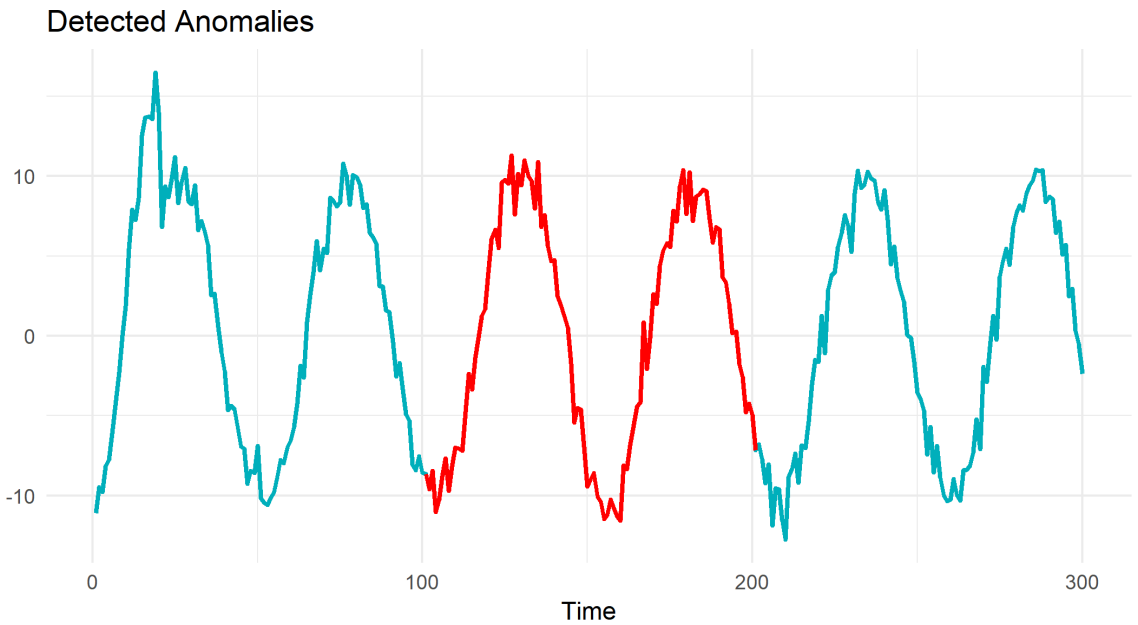

Detected Anomalies
120
80
40
0
0
100
200
300
Time

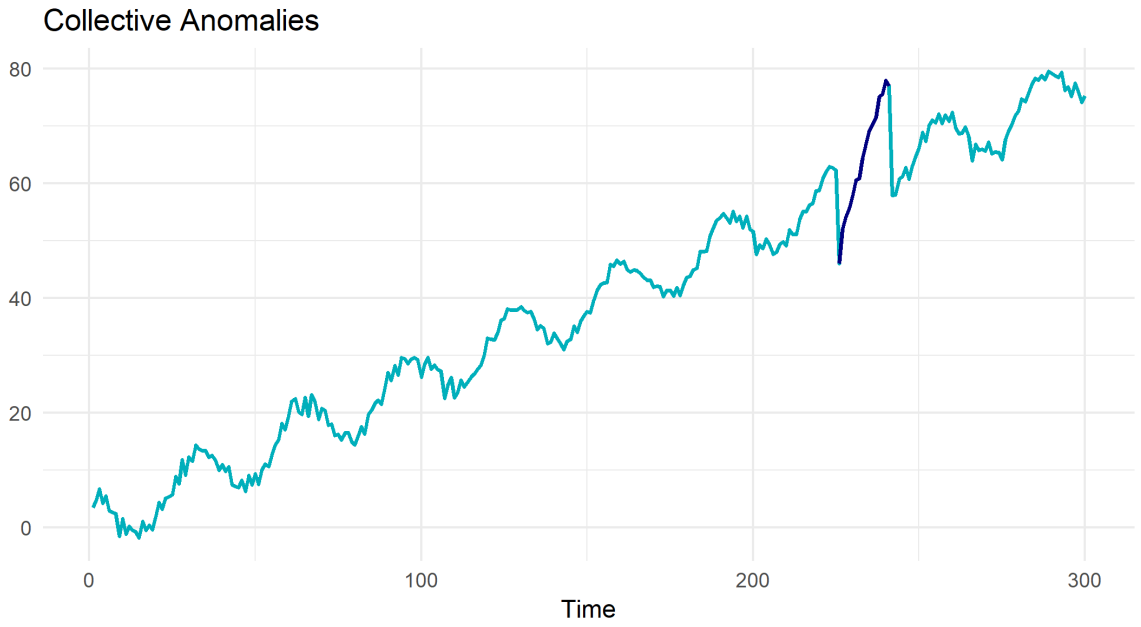

Collective Anomalies
60
40
20
0
0
100
200
300
Time

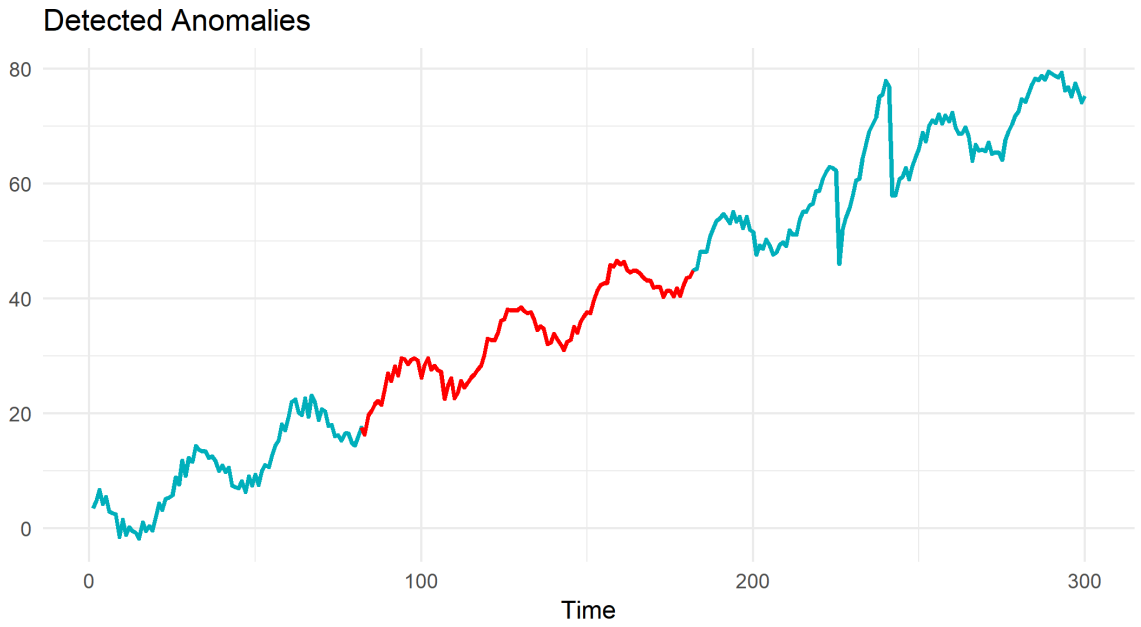

Detected Anomalies
60
40
20
0
0
100
200
300
Time

D Appendix: The HOT SAX with Window Size 100






Collective Anomalies
80
60
40
20
0
0
100
200
300
Time


Detected Anomalies
80
60
40
20
0
0
100
200
300
Time