Automated Load Curve Data Cleansing in Power Systems

by

Jiyi Chen
B.Sc. Peking University, China, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

© Jiyi Chen 2010
SIMON FRASER UNIVERSITY
Spring 2010

APPROVAL

Name:	**Jiyi Chen**
Degree:	**Master of Science**
Title of Thesis:	**Automated Load Curve Data Cleansing in Power Systems**

Examining Committee:

Chair: **Dr. Qianping Gu**

Dr. Ke Wang
Senior Supervisor

Dr. Jiguo Cao
Co-Supervisor

Dr. Fred Popowich
SFU Examiner

Date Approved: March 31, 2010

Abstract

Load curve data refers to power consumption recorded by meters at certain time intervals at delivery points or end user points, and contains vital information for day-to-day operations, system analysis, system visualization, system reliability performance, energy saving and adequacy in system planning. It is unavoidable that load curves contain corrupted data and missing data due to various random failure factors in meters and transfer processes. In this thesis, nonparametric smoothing techniques are proposed to model the load curve data and detect corrupted data. An adapted multiplicative model is built to correct corrupted data and fill in missing data. In implementation, an incremental training procedure is proposed to enhance the performance. The experiment results on the real BCTC (British Columbia Transmission Corporation) load curve data demonstrated the effectiveness of the presented solution.

Keywords: Time series, Dynamic Time Warping, Load management, Load modelling, Power systems, Power quality, Smoothing methods.

Acknowledgements

I would like to thank my senior supervisor Dr. Ke Wang for his invaluable guidance and support. This work would not have been a reality without his insightful comments and valuable suggestions. I benefit from his extensive knowledge, extraordinary research experience and his warm encouragement. I am indebted to the other members of my supervisory committee: Dr. Jiguo Cao and Dr. Fred Popowich. Dr. Cao led me to the area of the Functional Data Analysis, which is of great importance to my thesis. Dr. Popowich greatly encouraged my working in this direction and provided me in-depth comments in my thesis.

I am grateful to BCTC (BC Transmission Corporation) for the access to its data set, BCTC's Principal Engineer Dr. Wenyuan Li and Senior Engineer Dr. Adriel Lau for their valuable help during my research.

I would also like to express my gratefulness to all other students under the supervision of Dr. Ke Wang for their kind help and encouragement. They are Junqiang Liu, Yabo Xu, Rhonda Chaytor, Mohammad Tayebi, Reza Shahidi-Nejad, Amin Milani Fard, Zhihui Guo and Hua Huang.

I thank Bin Jiang for his valuable comments for my paper. I thank Bin Zhou, Ming Hua, Zhenhua Lin, Asqar Saypil, Haiyang Wang, Xu Cheng, Feng Wang and all other friends for making my stay at SFU an enjoyable period of time.

Finally, I thank my parents and my brother for their forever love and support.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

In British Columbia, the BC Transmission Corporation (BCTC) is the provincial Crown Corporation that plans, builds, operates and maintains the province's publicly-owned electrical transmission system. ***Load curve data*** refers to the power consumptions recorded by meters at certain time intervals at delivery points or end user points. Load curve data is the “heartbeat” of electricity systems and is one of several most important data sets collected and retained in utilities. The analysis of load curve data would greatly improve day-to-day operations, system analysis, system visualization, system reliability performance, energy saving, and accuracy in system planning [1]. Thus, the quality of load data is of vital importance to BCTC. In practice, it is unavoidable that load curves contain corrupted data and missing data due to various random failure factors in meters and transfer processes. In this thesis, we aim to improve the load curve data’s quality, i.e. correct the corrupted data and filling missing data with reasonable estimates.

1.1 Background

Two key features in the global vision of a smart grid [2] are self-healing from power disturbance events and enabling active participation by consumers in demand response. The collection of valid load curve data is critical for supporting decision making in a smart grid system. For example, smart meters are an important initiative in smart grid and the quality of data is essential for the success of smart meters.

Collecting all load data accurately in fine granularity is a challenging and costly task. There is often missing and corrupted data in the process of information collection and transfer. This is caused by various reasons including meter problems, communication failures, equipment outages, lost data, and unknown factors. Other reasons include unexpected interruption or shutdown in power use due to strikes, unscheduled maintenance, and temporal closure of production lines. Such events cause a significant deviation in load and do not repeat regularly, resulting in load data records being unrepresentative of actual usage patterns. The term ***corrupted data*** refers to data records significantly deviated from its regular patterns.

Poor quality load curve data can lead to misleading data analysis and incorrect decision making. For instance, the total system load estimated by direct summing up corrupted load data at delivery points can result in an over-estimation or under-estimation of the actual system, which in turn leads to either over-investment on costly transmission lines or operation risk of overloading transmission lines. It is important that corrupted data is identified and corrected. Currently, most utilities handle corrupted data manually in an ad hoc manner. This approach actually does not work, particularly after considerable smart meters come into place, as it is impossible to handle a huge data pool using a manual process.

Missing data can be treated as a special case of corrupted data. The problem of detecting and correcting corrupted data is referred to as the ***load cleansing problem***. Basically the goal of the load cleansing problem is to improve the quality of the load curve data. And it is done by correcting the corrupted data. There are three major challenges in resolving this problem. First, if a relatively large portion of data is corrupted or missing, most standard statistical methods cannot be applied. Second, due to the uncertainty in power consumption, randomness of outage events and dynamism of customers, it is difficult to judge whether a relatively large deviation represents corrupted data or an underlying change in data patterns. Third, the underlying patterns such as trends, periodicities, and autocorrelations should be kept when correcting the corrupted data such that the information behind the load curve data is not distorted.

The contributions of this work are as follows. First, two types of corrupted data are formalized for load curve time series data, which are called *local* and *global trend corrupted data*, respectively. Basically global trend corrupted is the data deviating too much (markedly) from global trends while local corrupted data is the data deviating too much (markedly) from local trends. They are formalized in Chapter 4. Though related, detecting corrupted data is different from the traditional load forecasting problem. It is very similar to outlier detection in time series. More discussion will be given in the Section 1.2. Second, a principled solution is presented by modelling the underlying structure of load curve data and using specific *nonparametric regression* techniques. This solution provides a common basis for detecting corrupted data, estimating replacing data, and deriving the confidence level for detection. The solution deals with both global and local trend corrupted data in a uniform way and is robust to a relatively large portion of missing data. Third, an incremental training algorithm is developed to enhance the performance, which incorporates the user feedback at an early stage with minimum user effort. Fourth, the Dynamic Time Warping distance and curvature of the underlying function of the load curves are used to find the ***yearly discords*** (defined in Chapter 5) of the load curves. Finally, a multiplicative model, incorporating global trends and yearly seasonality is applied to estimate missing data. All the methods and models are tested using real BCTC load curve data and the results demonstrated the effectiveness and high performance of the proposed methods.

1.2 Related Work

In this section, we review the popular related techniques in the literature. A closely related area to our load cleansing problem is *outlier detection*, which has been extensively studied in data mining and statistics research. The other closely related work is load forecasting. Below we first introduce outlier detection literature and then review the load forecasting works.

In the domain of data mining, a broad spectrum of techniques has been developed to detect outliers, among which, proximity-based techniques such as k-nearest neighbor classification [3], k-means clustering [4], and neural network methods such as RNNs [5] are frequently used [6]. Most of these techniques are designed for structured relational data instead of for time series. When dealing with outliers in time series, data mining researchers are more interested in determining whether a whole time series is abnormal with respect to a database of "normal" time series (multiple time series). Most techniques ([7][8][9][10][11]) compute an anomaly score of a coming time series based on its distance to the database of normal time series, using different distance measures (e.g. Euclidean Distance, Dynamic Time Warping and Cross Correlation). These elegant methods are not applicable to our problem because we do not have a database of explicitly labeled "normal" time series. Instead, our goal is to find corrupted data within a single long time series.

Keogh et al develops a suite of techniques to detect “discords” in a time series [12][13][14][15], that is to find the time series subsequence that is maximally different from all the rest of the time series subsequences. A representative technique in [12] has the following steps. First, it splits a time series into multiple subsequences with equal length by sliding a window in the time dimension. Second, it computes the pair-wise distance between subsequences to find the nearest subsequence (called the nearest neighbor) for each subsequence. Third, the subsequence with the maximum distance to its nearest neighbor is the top-1 discord. This technique is efficient in finding discords, but is not suitable for solving our problem. First of all, it requires the prior knowledge on the length of the sliding window, which corresponds to the length of corrupted data in our time series. This knowledge is difficult to obtain because corrupted data in load curve could have different length. Secondly, the so-called discords are not necessarily corrupted data or outliers for load curve data, and vice verse. Consider a load curve with an obvious increasing trend. Subsequences with larger values tend to have larger distances to their nearest neighbors, therefore, are more likely to be detected as discords. But such subsequences can be normal because a load curve typically has an increasing trend over time.

Outlier detection in time series data is also studied in the field of statistics [23]. Well known statistical tests for outliers include Z-value, Box Plot, Rosner test, Dixon test and Grubbs test. The assumption underlying these tests is the normal distribution of data [16], which is far from true for our load data. Other works [17][18][24][25][26] are based on the ARMA model, which impractically assumes that the time series is stationary. Moreover, traditional time series methods, including the ARMA model, can only be applied for equally-spaced time series data, therefore, are hard to be used for load curve data where many time points have no measurements available due to missing data. To our best knowledge, nonparametric regression methods are seldom used for outlier detection.

The other closely related field is *time series forecasting* [27] or *load forecasting* [29]. The load cleansing problem is different from load forecasting problem. In load forecasting, historical loads are used to forecast the load at a future time. In load cleansing, historical loads are used to detect whether the load recorded at a particular historical time is corrupted and decide what estimated value should be used to replace it. In other words, while load forecasting trusts all historical data and uses them to predict a data value at a future time point, load cleaning deals with the possibility that some historical data might be corrupted and has to detect such corrupted data and replace them. Although different, they share the following similarities. First, for both problems, it is usually necessary to model the underlying patterns of the load curve data. In load forecasting, the patterns should be modelled to predict the load value at a future time point while in load cleansing the patterns are considered as reference from which corrupted data deviate too much. Second, in load cleansing, an important task is to fill in the missing data. The process of filling in missing data could be considered as a type of load forecasting where both historical data and future data are available. Thus, we also review the main techniques in load forecasting. They can be adapted to assist solving load cleansing problem.

Exponential smoothing is a popular scheme to produce a smoothed time series and a popular technique that has been applied to load forecasting. The basic idea of exponential smoothing is to assign exponentially decreasing weights as the observation gets older

[31]. In other words, when estimating current observation, recent observations are given relatively more weight in forecasting than the older observations. There are three types of exponential smoothing, namely *single*, *double* and *triple* exponential smoothing. The single exponential smoothing simply model the smoothed value of the current observation as a weighted sum of historical observations, where the weights are exponentially decreasing as the observation grows older. The double exponential smoothing incorporates a **trend** factor and the triple exponential smoothing (a.k.a. Holt-Winters Method) [35] takes into account both **trend** factor and **seasonality** factor. We take a closer look at the multiplicative version of Holt-Winters Method because we are going use the idea of "multiplicative model" in **missing data filling** in Chapter 6. The multiplicative Holt-Winters Methods is represented as

$$\begin{aligned}
S_t &= \alpha \frac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + b_{t-1}) \\
b_t &= \gamma (S_t - S_{t-1}) + (1-\gamma) b_{t-1} \\
I_t &= \beta \frac{y_t}{S_t} + (1-\beta) I_{t-L} \\
F_{t+m} &= (S_t + m b_t) I_{t-L+m}
\end{aligned} \quad , \qquad (1)$$

where S_t is the smoothed value at time t, b_t is the trend factor at time t, I_t is the seasonal index and F_{t+m} is the forecasting value at time $t+m$ and α, β, γ are smoothing constants [32]. Chatfield discussed the properties of Holt-Winters Method and the computation of prediction interval in [33][34][35]. Taylor applied the different exponential smoothing methods with certain constraints to forecast short-term electricity demand in [37][39] [40], and supermarket sales [38]. Although the results in the papers show the effectiveness of the forecasting, exponential smoothing cannot be applied directly to the load cleansing problem because of following reasons. First, exponential smoothing is more capable of very short term forecasting. When there is a reasonably large portion of the missing data, the accuracy of the prediction will be significantly influenced. Second, the exponential smoothing assumes that the periodicity periods are staying the same. However, the periods may change over time.

ARMA [27] models are another set of techniques that can be used for load forecasting in time series analysis. An AR (autoregressive) model is simply a linear regression of the current value of the series against one or more prior values of the time series. An MA (moving average) model is conceptually a linear regression of the current value of the series against the white noise or random shocks of one or more prior values of the series. The ARMA model is a combination of the AR and MA models. The ARMA models suffer from the similar problems as the exponential smoothing, such as not robust to missing data.

Although different, many ideas and methodologies in the load forecasting can be applied to load cleansing problem. For example, the **multiplicative model** can be adapted to fill in the missing data and the **prediction interval** can be adapted to detect trend corrupted data.

The rest of the thesis is organized as follows. In Chapter 2, the load cleansing problem itself is anatomized associated with the real BCTC data. Complicated as it is, the load cleansing problem is decomposed into several sub-problems and addressed in a divide and conquer manner. From Chapter 3 to Chapter 6, the detailed solutions to every sub-problem are presented and tested on the real BCTC data. We discuss the extensions and future work in Chapter 7.

Chapter 2. Problem Anatomy

A ***load curve*** is a time series where a load value is collected at a certain time frequency such as every five minutes or hourly. Typically, load curve data follows certain patterns and behaves with a daily, weekly and seasonal periodicity with an increasing or decreasing tendency over years. Load curve data is also influenced by random factors such as outages, meter failures, communication interruptions or errors and dynamism of customers. As a result, a load curve consists of not only white noise but also some ***corrupted data***.

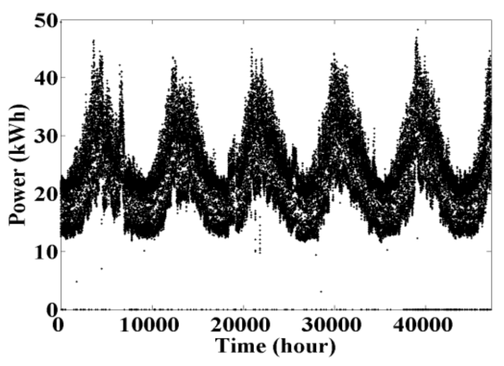
In this work, we aim to improve the quality of a load curve. Specifically, given a time series representing a load curve, we aim to cleanse it, which includes two major tasks: detection of corrupted data and repair of corrupted data. We refer it to as the ***load cleansing*** problem. In the process of load cleansing, the underlying regular patterns of the load curve, for example, periodicity, trends and autocorrelations, must be kept while assuring the quality of replaced data. **Missing data** can be considered as a special case of corrupted data where a load value of zero is collected.

Definition 1. *Corrupted Data*: Consider a time series $\{(t_i, y_i)\}_{i=1}^{n}$, where y_i is the data (observation) at the time t_i. A data point is corrupted if it deviates *markedly* from the *underlying patterns* of the time series.

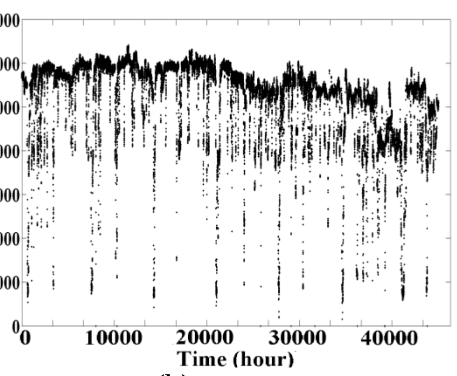
To fully describe the corrupted data, two questions need to be answered. What **patterns** are underlying a time series? How much deviation could be considered a "**marked deviation**"? The first question is answer in Chapter 3, in which pattern modelling method is introduced. The second question is answered in Chapter 4. To make more sense of the load cleansing problem, we need to look into the real data.

2.1 Problematic Data

In the whole BCTC data set, there are around 300 load curves and almost each load has its unique shape and other characteristics. The data has been categorized into two types according to different usages, namely *distribution* (or residential) data that represents from residential power consumption and *industrial* data that represents industrial power consumptions. Fig. 1(a) and Fig. 1(b) give the typical examples of distribution data and industrial data. In general, the *distribution* data behaves more regularly, following some obvious rules with relatively smaller variance. In contrast, the *industrial* data has much more irregularity as industrial power consumption is very sensitive to the economic conditions. A whole factory may shut down suddenly due to the lack of orders made from its customers. However, there are situations that some *distribution* data behaves much more irregularly than *industrial* data. If load curve data, no matter from *distribution* or *industrial*, follows certain rules or patterns, abnormal deviations from the rules will appear due to various reasons such as strikes and sudden weather change. Data patterns and abnormal deviations are always obvious to see when the data is visualized. Some examples will be given in following sections.



(a)



(b)

Fig. 1 Distributional (a) and industrial data example (b)

2.1.1 Patterns

The load curve data typically has **patterns** such as **periodicity**, **trends** and **normal fluctuations**.

Periodicity, meaning similar or equal values occurring over a period regularly, is a very common pattern in load curve data. As mentioned before, there exist periodicities with different periods in the same load curve. Looking at data sets in different granularities can reveal different periodical patterns. In some load curve, when daily data is under consideration, as illustrated in Fig. 2(a), a simple rule can be observed: the power consumption from 8am to 10pm of a day is usually higher than the rest time of the day and data values forms a shape similar to the letter “n”. In some other load curves, such day-to-day repetition patterns may not appear any more. On the other hand, a periodicity with a period of one week can be observed, just as shown in Fig. 2(b). The weekends’ power consumption is regularly lower than weekdays’ consumption. If five years’ data is under consideration, as shown in Fig. 2(c), another rule can be captured: the values at the similar time of every year are very close to each other, and thus the shapes of every year are very similar to each other.

However, this is not the whole story. Practically, periodical patterns do not indicate exact repetitions of data values. Instead, they represent *similar* data values or the shapes of data repeating over periods. It is easy for a human being to judge whether two shapes are similar or not. However, it is much more difficult to mathematically define similarity between two shapes. This is because a human being could easily ignore minor differences without a precise definition of what is "minor difference", while mathematical model usually require an exact threshold or splitting line to separate "minor difference" and "major difference".

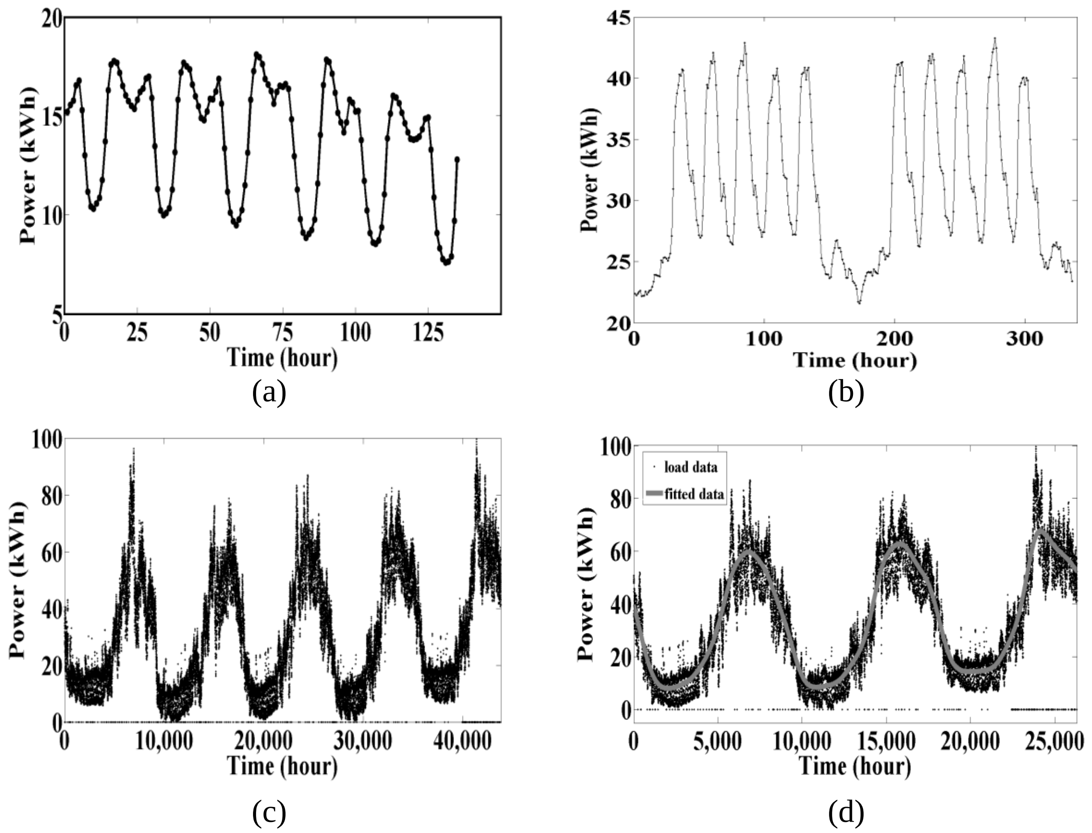
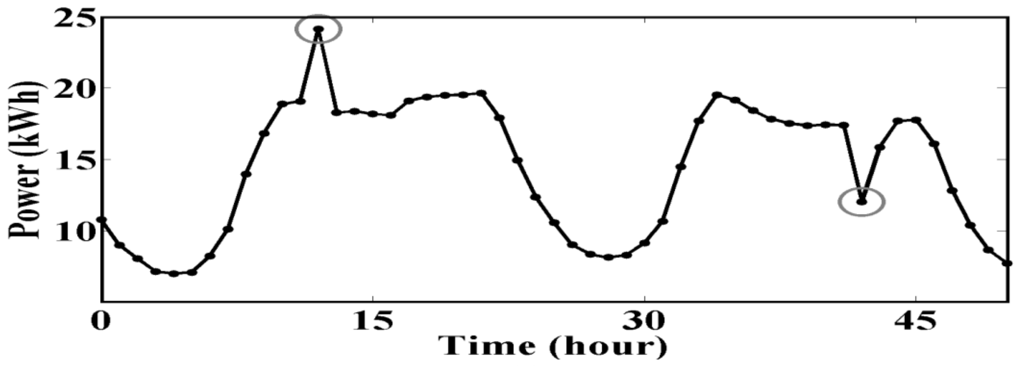


Fig. 2 Data patterns example

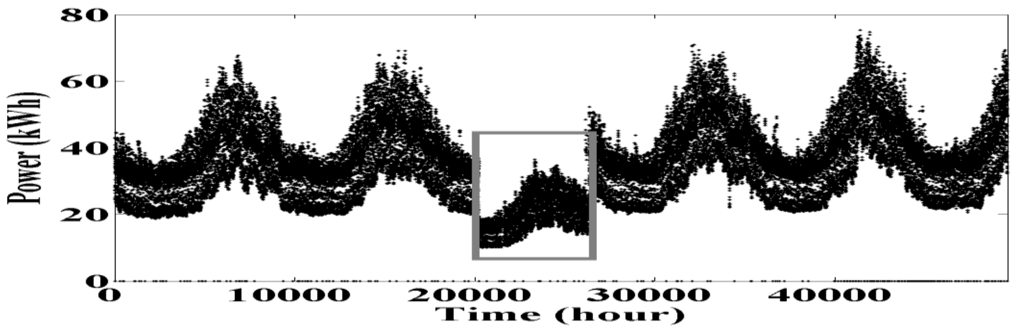
Besides periodical patterns, the load curve data may follow specific **trend function** or we call that the load curve data have certain **trends**. ***Trend**s* represent the general moving directions of the load curve. The solid line in Fig. 2(d) can be considered the trends of the load curve. In addition, the load curves usually have some reasonable fluctuations or normal fluctuation. Still in Fig. 2(d), we can see the data fluctuating around the solid line (the trends).

2.1.2 Corrupted Data

As defined in **Definition 1**, a data point is considered corrupted if it deviates markedly from the underlying patterns. Thus, any data corruption or abnormal deviation should be considered with respect to the underlying data structures or patterns the data follows. Different kinds of corruptions can be observed when aligning data with different patterns. For example, if a load curve follows a periodical pattern with the period of one day, data points that deviate too much from the daily repetition are considered corrupted with respect to the daily periodicity. Similarly, if it deviates abnormally from yearly periodicity (if the load curve has a one-year periodicity), it is considered corrupted from yearly periodicity. Fig. 3 shows data corruptions with respect to different periodical patterns. The two circled data points in Fig. 3(a) represent corruptions with respect to the daily periodicity; in Fig. 3(b), the data in the rectangle shows a corruption with respect to the yearly periodicity.



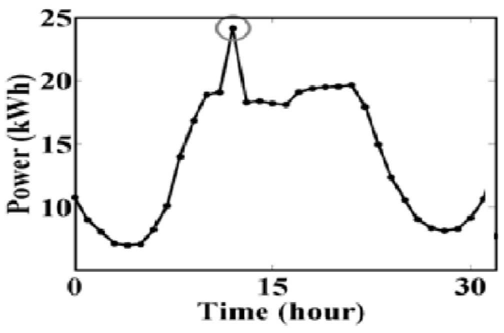
(a)

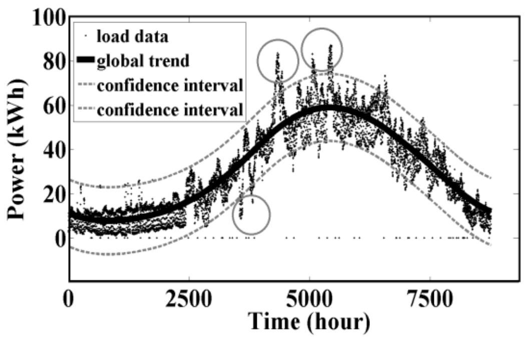


(b)

Fig. 3 Corrupted data with respect to daily (a) and yearly (b) periodicity

Moreover, there is corrupted data that deviates too much from **trends** instead of periodicities. Load data that is too far away from the underlying trends should be considered corrupted. Such abnormal deviations can often be observed when the data values are suddenly much higher or lower than their neighbours. For instance, the circled point in Fig. 4(a) is obvious corrupted data as it is much higher than its previous and next data point. Similarly, in Fig. 4(b), the three-circled groups of data are considered corrupted since they deviate from their neighbours "too much". It is natural to have the following questions. First, indeed, how many data points nearby should be counted as neighbours? Second, how much deviations should be considered as "too much" (a question asked in previous section)? These questions will be answered in Chapter 4.





(a) (b)

Fig. 4 Corrupted data with respect to trends

2.1.3 Missing Data

Missing data is the data that the meters fail to collect due to various reasons, for example the meters. Filling missing data is another important task in the load cleasing problem. The load curves may contain different amount of missing data. Sometimes missing data appear as a single point, and more often they appear as an time interval in which all data points are missing. Fig. 5 shows an example of missing data problem. Two obvious blank intervals marked by rectangles indicate two intervals of missing data. Missing data is also considered as a special type of corrupted data where zero is collected as data value. And filling missing data is also necessary whenever any corrupted data are detected -- the detected corrupted data need to be deleted and replaced with estimated data.

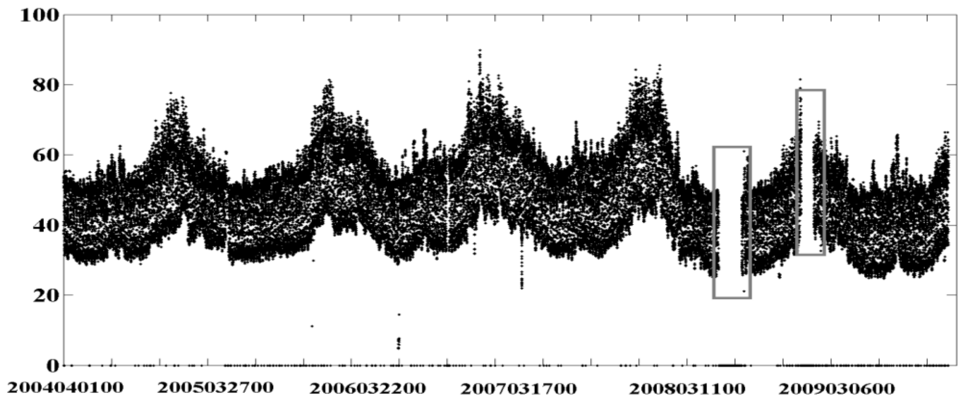


Fig. 5 Missing data example

A most intuitive way and also the currently used way in BCTC to fill in the missing data is to pick up the data from the corresponding historical data. For example, select the data at same time of previous week, previous month or previous year to replace the missing data. This simple method works fine if the load curve has an perfect periodicity. However, this is not always the case. First, a load curve generally has certain trends (increasing or decreasing). Using the raw historical data to fill in the missing data at current time may be an underestimation or overestimation. Second, the patterns of the load curve may vary from time to time.

2.2 Challenges

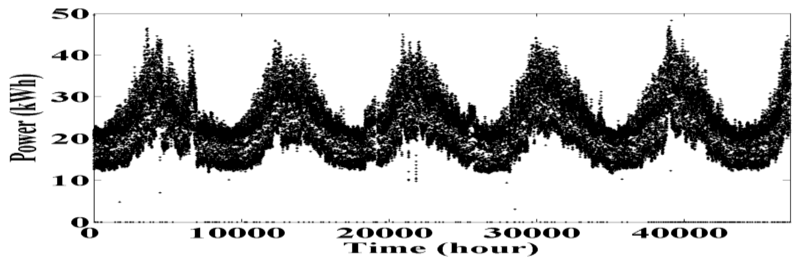
The main challenges of the detection of corrupted data are how to model the underlying patterns of a load curve and how to quantify the degree of the deviation of data from the patterns.

2.2.1 Pattern Modelling

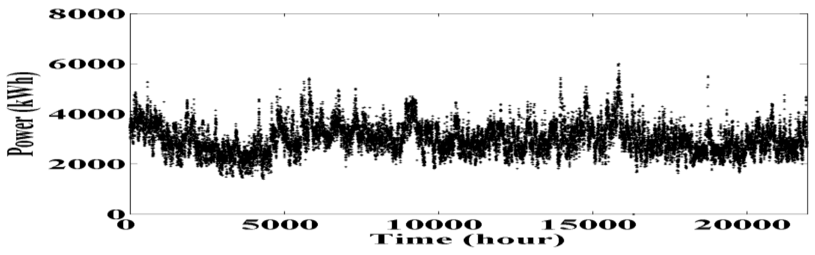
Different load curves may have different patterns, and patterns inside a load curve would change from time to time. For example, a load curve may be periodical with different periods such as a day, a month, a year and other lengths. Modelling periodical patterns of all different granularities (some unknown length of periods) is a challenging task. Fig. 6(a) shows a typical residential load curve that has an obvious yearly periodicity. It is shown that, although the data values at same time of every year are very similar, they are not exactly the same. A pure periodical function is not capable of modelling all the characteristics of the curve. Besides, the lengths of the periods are not exactly the same – even if the periods are always one year, not every year has the same length (a leap year has one day more than a regular year). It is also important to note that there are load curves that do not have obvious periodicities. An intuitive solution is to judge the periodicity of the load curve before apply the mathematical model to it. This is intuitive but may not be a good solution. It will be the best if there is a model that does not require prior knowledge such as periodicity of the data.

Besides periodicity, the trend features are also very important patterns that need to be modelled. The load curves generally have an increasing trend over years. In addition, some load curves do not have obvious periodicity; however, their trends are still following some mathematical formulae form. Again, when considering trends at different granularities (daily, weekly, or yearly), different trends characteristics can be observed.

Moreover, there may be patterns that are obvious to human eyes, however, are not so obvious when considered mathematically. For example, the Fig. 6(b) shows a regular industrial load curve. From an observer's perspective, the load curve is quite normal in the sense that it fluctuates "regularly" or "normally". However, there are no gold standard rules that can help computers judge whether a load curve is normal or not.



(a)



(b)

Fig. 6 Load curve examples with different patterns

2.2.2 Degree of Deviation

In previous sections, different types of corrupted data deviating from different patterns have been introduced. Assuming that different patterns have been modelled by a certain mathematical formula. Real load curves will never exactly follow the mathematical model. Now the questions are how to quantify the deviations of load curves from the models and how much deviation should be considered irregular or corrupted. The answer to this question in our solution is to compute confidence intervals for normal data at every time point. The confidence limits of the confidence intervals constitute the boundary between normal data and corrupted data. More details will be presented in Chapter 4.

2.3 Problem Decomposition

The whole load curve cleansing problem is decomposed into the following sub-problems: *Pattern modelling*, *trend corrupted data detection*, *yearly discords detection* and *missing data filling*.

Pattern modelling, as it sounds, is to model the underlying structure of the load curves. Load curves have all kinds of characteristics or patterns without perfect predefined forms. To address this issue, nonparametric smoothing techniques are utilized in Chapter 3. The nonparametric smoothing techniques provide a ***smoothing parameter*** that can control the smoothness of the curve; or in other words, the smoothing parameter decides how many nearby data points are considered to estimate a current data point.

Trend corrupted data detection is to detect the data that deviates too much from the **trends**. The load curves have local and global trends and accordingly the trend corrupted data are classified *local* trend corrupted data and *global* trend corrupted data. This trend corrupted data detection problem is addressed in Chapter 4. The **smoothing parameter** introduced in Chapter 3 makes it possible to detect the two types of corruption in a same manner.

Yearly discords detection is to find the most unusual year in a load curve. Discords detection is a hot topic in time series data mining [12]. It can be considered to be a step in the detection of anomalies in time series. The yearly discords detection is addressed in Chapter 5.

Filling missing data is addressed by utilizing a multiplicative model, taking into account of trend factor and seasonality (periodicity) factor [41] in Chapter 6. The trend factor represents the “scale level” of the load curves and seasonality factor represents the normal deviation of the load data from the trends.

Chapter 3. Pattern Modelling

The first and most critical step for the whole *load cleansing problem* is to model the intrinsic patterns or structure of load data. The idea to the modelling is to consider that there exists a continuous function that generated the data. And **trends** and **periodicities** of the load curves are all implicitly modelled in that function. Assume that n data points $\{(t_i, y_i)\}_{i=1}^{n}$ of a load curve have been collected. The underlying data generation process is modelled as [19]

$$y_i = m(t_i) + \varepsilon_i \tag{2}$$

where y_i is the data value at time t_i, $m(t)$ is the underlying function and ε_i is the error term. It is assumed that the error term ε_i is *white noise* [44], i.e. it is normally and independently distributed with the mean of zero and constant variance σ^2. Our main task is to find an appropriate estimate of the function $m(t)$, namely $\hat{m}(t)$ (or **fitted curve**), using the collected load data as a sample of observations.

The question is: how to find such a function that can just fit a new coming load curve? Following we introduce **nonparametric regression** techniques to answer this question.

3.1 Nonparametric Regression

The aim of a regression analysis is to estimate the unknown response function (or curve) *m(t)* from the observed data $\{(t_i,\ y_i)\}_{i=1}^{n}$. This approximation procedure is called *smoothing*. The smoothing task can be done essentially in two ways: *parametric regression* and *nonparametric regression*. The former assumes that the curve *m(t)* has some pre-specified functional form, whereas the latter does not. As we have seen in the previous sections, there are more than 300 load curves in the data set and almost each of them has unique shape and other features. Pre-specifying a function form for every load curve is too challenging, if not impossible. Thus, the parametric regression is not applicable. Instead, the nonparametric regression (nonparametric smoothing) is used to model the load curves.

The basic idea of nonparametric smoothing is the *local averaging procedure*. Specifically, the curve can be estimated by

$$\hat{m}(t) = \frac{1}{n}\sum_{i=1}^{n} W_i(t) y_i \,, \tag{3}$$

where $\{W_i(t)\}_{i=1}^{n}$ denotes a sequence of weights which depend on the whole vector $\{t_i\}_{i=1}^{n}$. Among most well accepted nonparametric smoothing techniques are *Spline* smoothing and *Kernel* smoothing. In this paper, the *B-Spline smoothing* [20], which is commonly used in the Spline smoothing family, and the popular *Nadaraya-Watson* estimator in the Kernel smoothing family [19] are used.

The remaining issue of pattern modelling is in what granularity patterns should be modelled. To answer this question, a smoothing parameter introduced to control the curve smoothness. The basic idea is that a smoother curve *m(t)* tends to model global patterns since it is less sensitive to local deviations, whereas a rougher curve *m(t)* is more capable of modelling local patterns. Different settings of the smoothing parameter can be chosen to model global or local patterns.

3.2 B-Spline Smoothing

3.2.1 Basis Function System

To estimate the function *m(t)*, the B-Spline smoothing makes use of a basis function system consisting of a set of known basis functions $\{\phi_k(t)\}_{k=1}^{K}$ that are mathematically independent of each other. The idea is to approximate the function *m(t)* by taking a weighted sum or linear combination of a sufficiently large number K of basis functions $\phi_{\mathrm{k}}(t)$

$$m(t) = \sum_{k=1}^{K} c_k \phi_k(t)\,, \tag{4}$$

or in the form of vectors

$$m(t) = \vec{c}^{\,T} \vec{\phi}(t)\,, \tag{5}$$

where $\vec{c} = (c_1, \ldots, c_K)$ is the coefficient vector and $\vec{\phi}(t) = (\phi_1(t), \ldots, \phi_K(t))$ is the vector of basis functions. There are different basis function systems. The B-Spline basis system developed by de Boor [21] is adopted as it is among the most popular and powerful basis systems [20]. Fig. 7 shows how a B-Spline basis system looks like. There are 11 basis functions forming a basis function system in Fig. 7. Each wave crest represents a basis function. Each function $\phi_k(t)$ in a B-Spline basis system is positive over a short interval of time and is zero in the rest. This property, called compact support property, guarantees that mainly local information is considered when estimating the coefficients $\vec{c}$.

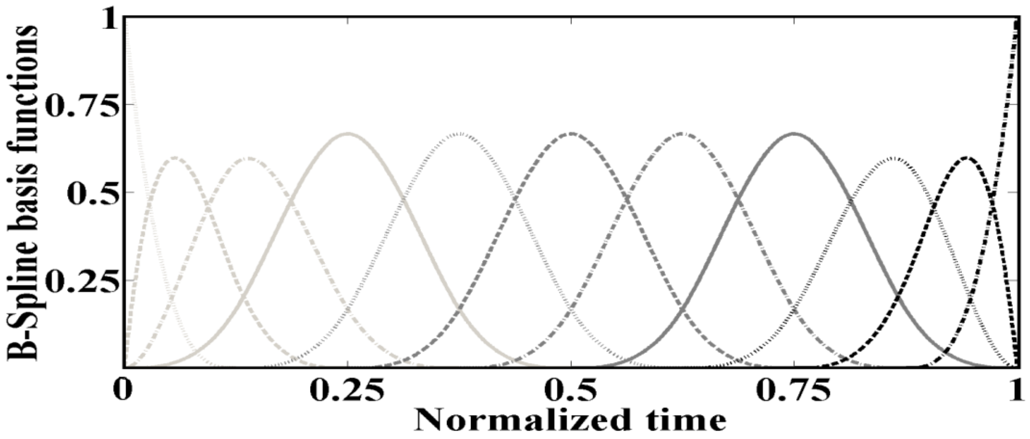


Fig. 7 B-Spline basis system example

3.2.2 Estimating Coefficients

To estimate the coefficients $\vec{c}$ from the observations $\{(t_i, y_i)\}_{i=1}^{n}$, we define an n by *K* matrix

$$\Phi = \begin{bmatrix} \phi_1(t_1) & \phi_2(t_1) & \cdots & \phi_K(t_1) \\ \phi_1(t_2) & \phi_2(t_2) & \cdots & \phi_K(t_2) \\ \vdots & \vdots & & \vdots \\ \phi_1(t_n) & \phi_2(t_n) & \cdots & \phi_K(t_n) \end{bmatrix}, \quad (6)$$

where $\Phi[i, j] = \phi_j(t_i)$ represents the value of the *jth* basis function at time t_i. By treating all vectors as column vectors, the function values *m(t)* at time $(t_1, ..., t_n)$ are given by $\vec{m} = \Phi\vec{c}$. A simple smoother could be obtained if the coefficients $\vec{c}$ are determined by minimizing the *sum of squared error* (*SSE*) as

$$SSE = \sum_{j=1}^{n}[y_j - \sum_{k=1}^{K} c_k \phi_k(t_j)]^2 , \quad (7)$$

or in the form of vectors

$$SSE = (\vec{y} - \Phi\vec{c})^T (\vec{y} - \Phi\vec{c}), \quad (8)$$

where $\vec{y}$ is the vector form of $\{(t_i, y_i)\}_{i=1}^{n}$. The *SSE* can always be decreased by using an enough number of basis functions, to make the fitted curve go through all data points. However, a larger *K* can lead to more risk of over-fitting, i.e., the noise that we wish to remove has more chance to be fitted.

To solve the over-fitting problem, the coefficients can be estimated by minimizing the *penalized sum of squared errors* (*PENSSE*) as follows

$$PENSSE_{\lambda} = SSE + \lambda \times PEN_2(t), \tag{9}$$

where λ is the smoothing parameter and $PEN_2(x)$ is the roughness measure which is defined as the integral of square of second derivative or curvature of the curve *m(t)*

$$\begin{aligned} PEN_2(t) &= \int [D^2 m(t)]^2 dt \\ &= \int [D^2 \vec{c}^{\,T} \vec{\phi}(t)]^2 dt, \\ &= \vec{c}^{\,T} R \vec{c} \end{aligned} \tag{10}$$

where

$$R = \int D^2 \vec{\phi}(t) D^2 \vec{\phi}(t)^T dt. \tag{11}$$

Generally, the rougher the curve is, the larger curvature it tends to have. The smoothing parameter λ controls the scale of roughness penalty that will be put on and thus controls the smoothness (or roughness) of the curve. By combining (8), (9) and(10), the *PENSSE* can be expressed in the form of vectors as

$$PENSSE_{\lambda} = (\vec{y} - \Phi\vec{c})^T (\vec{y} - \Phi\vec{c}) + \lambda \vec{c}^{\,T} R \vec{c}. \tag{12}$$

The estimate of the coefficients $\vec{c}$ can be obtained by setting the derivative of *PENSSE* with respect to $\vec{c}$ to be zero

$$\hat{\vec{c}} = (\Phi^T\Phi + \lambda R)^{-1}\Phi^T\vec{y}\,. \tag{13}$$

From (5) and (13), the fitted value vector $\hat{\vec{y}}$ (i.e. estimated values for observations $\vec{y} = (y_1, ..., y_n)$) is computed by

$$\hat{\vec{y}} = \Phi\hat{\vec{c}} = \Phi(\Phi^T\Phi + \lambda R)^{-1}\Phi^T\vec{y}\,, \tag{14}$$

or

$$\hat{\vec{y}} = S\vec{y}\,, \tag{15}$$

where

$$S = \Phi(\Phi^T\Phi + \lambda R)^{-1}\Phi^T\,. \tag{16}$$

S is referred to as hat matrix as it converts the dependent variable vector $\vec{y}$ into its fitted value $\hat{\vec{y}}$. The hat matrix S in (15) plays the role of the weight functions $\{W_i(t)\}_{i=1}^{n}$ in (3). The number of degrees of freedom of the fit is the trace of the hat matrix

$$df = trace(S)\,. \tag{17}$$

In linear algebra, the trace of an n-by-n square matrix A is the sum of the diagonal elements of A. The degrees of freedom will be used in calculating the confidence interval for trend outlier detection.

3.3 Kernel Smoothing

In Kernel smoothing, the shape of the weight function $W_i(t)$ in (3) is described by a density function (*Kern*) called a kernel (e.g. a normal distribution density function) [19]. The weight sequence $\{W_i(t)\}_{i=1}^n$ is defined by

$$W_i(t) = \frac{Kern_h(t - t_i)}{n^{-1}\sum_{i=1}^{n} Kern_h(t - t_i)}, \tag{18}$$

where

$$Kern_h(t) = \frac{1}{h} Kern(\frac{t}{h}) \tag{19}$$

is the kernel with the scale factor h. A well accepted kernel estimator is Nadaraya-Wastson estimator

$$\hat{m}_h(t) = \frac{n^{-1}\sum_{i=1}^{n} Kern_h(t - t_i) y_i}{n^{-1}\sum_{i=1}^{n} Kern_h(t - t_i)}. \tag{20}$$

The shape of the weights is determined by *Kern* and the size of the weights is parameterized by *h*, which is called the bandwidth. In our case, *Kern* is chosen to be the probability density function of the standard normal distribution

$$Kern(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2}, \tag{21}$$

such that $Kern_h$ is the density function of a normal distribution with a mean of 0 and a variance of h^2. According to (3) and (18), the fitted value vector $\hat{\vec{y}}$ can be rewritten in the form of (15), i.e., $\hat{\vec{y}} = S\vec{y}$, where the hat matrix S is defined as

$$S = \begin{bmatrix} n^{-1}W_1(t_1) & n^{-1}W_2(t_1) & \cdots & n^{-1}W_n(t_1) \\ n^{-1}W_1(t_2) & n^{-1}W_2(t_2) & \cdots & n^{-1}W_n(t_2) \\ \vdots & \vdots & & \vdots \\ n^{-1}W_1(t_n) & n^{-1}W_2(t_n) & \cdots & n^{-1}W_n(t_n) \end{bmatrix}, \tag{22}$$

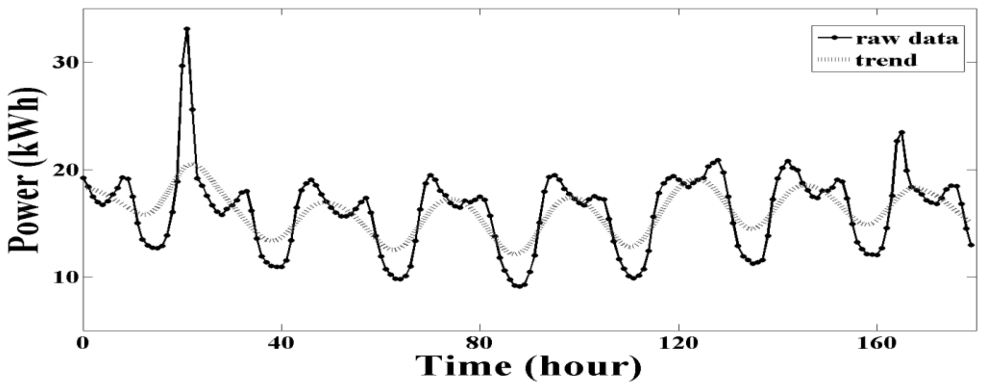
where $W_i(t_j)$ is the weight of observation at time t_i for estimating data value at time t_j. Again, the number of degrees of freedom of the fit is computed by the trace of the hat matrix, i.e.(17).

The bandwidth h controls the roughness of the fitted curve. This plays the same role as the smoothing parameter λ does in the B-Spline smoothing. The larger the h is, the more neighboring information is taken into account and the smoother the fitted curve would be. For simplicity, both h and λ will be referred to as the smoothing parameter.

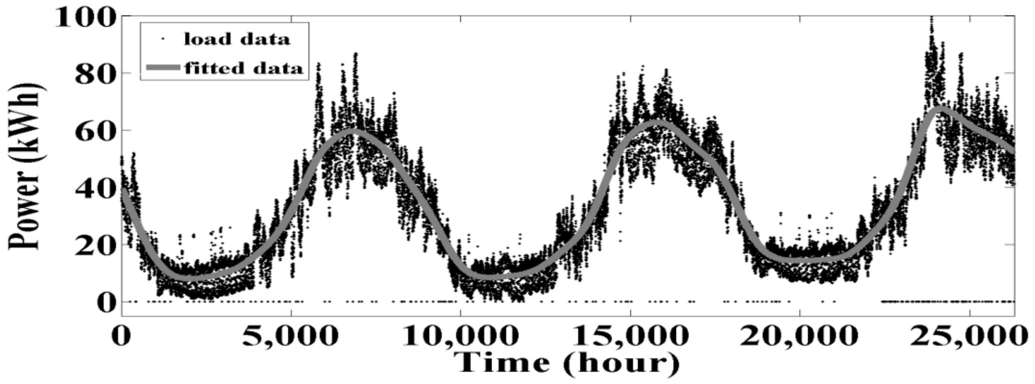
Chapter 4. Trend corrupted data detection

In this Chapter, we focus on the detection of trend corrupted data. The **trends** here refer to the general moving directions of the load curves.

> **Observation 1.** A load curve generally has two types of trends, namely **local trends** and **global trends**.



(a)



(b)

Fig. 8 Load trends examples

Fig. 8(a) and Fig. 8(b) give examples of local and global trends respectively. The dashed line in Fig. 8(a) and solid line in Fig. 8(b) represent the trends of the load curves. The local trends capture more detailed information of the load curve, for example, the information about daily variation of the load curve; the global trends focus on the big picture of the load curve, for example, the general moving directions of the curve in a year. However, there is no clear separating line between local and global trends. But it is also interesting to note that the proposed methods do not require such a separating line.

Deviating from different types of trends (local trends and global trends), corrupted data can be classified into *local trend corrupted data* and *global trend corrupted data*.

Definition 2. *Local Trend Corrupted Data*: Given a load curve time series $\{(t_i, y_i)\}_{i=1}^{n}$, a load data point y_i is local trend corrupted data if it deviates markedly from the local trends of the load curve.

Definition 3. *Global Trend Corrupted Data*: Given a load curve time series $\{(t_i, y_i)\}_{i=1}^{n}$, a load data point y_i is global trend corrupted data if it deviates markedly from the global trends of the load curve.

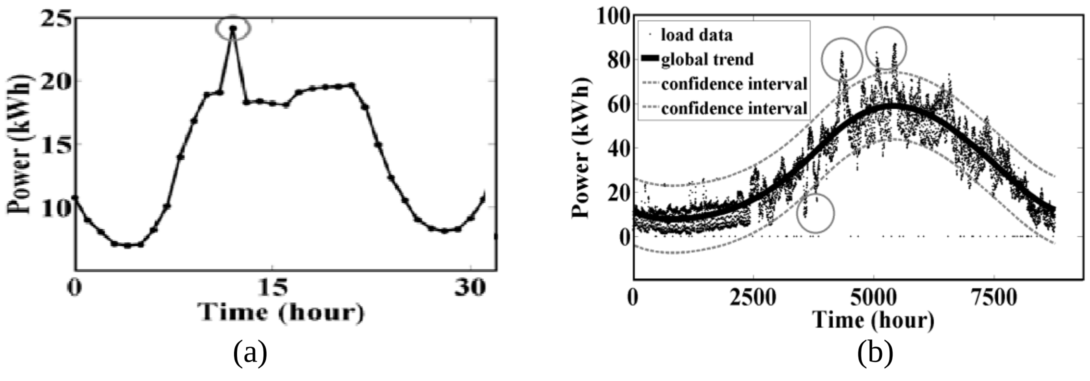


Fig. 9 Local and global corrupted data

Fig. 9 (a) and Fig. 9 (b) show examples of the local and global trend corrupted data. It is important to note that observations $\{(t_i, y_i)\}_{i=1}^{n}$ are not necessarily equally spaced in time dimension because some of them may be missing. Luckily, the presented B-Spline smoothing and Kernel smoothing are robust to a reasonable large number of missing data.

Sometimes corrupted data appears as a single point, and more often it appears as a time interval in which most of data points are corrupted. Therefore, the term "*corrupted region*" are frequently used in this thesis to represent the time-interval in which most data is corrupted. When the region contains only one data point, the corrupted region is actually a corrupted data point. For example, Fig. 9 (a) shows two *local trend corrupted regions* and Fig. 9 (b) shows three *global trend corrupted regions* indicated by circles.

The modelling methods for the underlying structure of a load curve were presented in the previous Chapter. Two remaining issues are how to set the parameters in the model and how to utilize the **fitted curve** $\hat{m}(t)$ (estimated underlying function) to detect trend corrupted data.

> **Observation 2**. With different smoothness levels, the fitted curve of a load curve can capture the local and global trends of the load curve. The fitted curve with lower smoothness level (the rougher curve) tends to capture the local trends; the fitted curve with higher smoothness level (the smoother curve) tends to capture the global trends.

The dashed line in Fig. 8(a) and solid line in Fig. 8(b) show local and global trends that are captured by the fitted curves with different smoothing parameters.

Once an appropriate fitted curve $\hat{m}(t)$ is found for a certain load curve, the basic idea of detecting trend corrupted data is to create a point-wise confidence interval for normal data based on $\hat{m}(t)$. An observation within the confidence interval is considered normal and an observation outside the confidence interval is considered corrupted. Both local and global trend corrupted data are detected in a same manner. The only difference is the setting of the smoothing parameter.

Now we can answer the two questions raised in Section 2.1.2.

Question 1: How many data points nearby should be counted as neighbors?

Answer: We do not need to specify a certain number to be considered as neighbors. Appropriate smoothing parameter automatically takes into account of the number of appropriate neighborhoods. Section 4.3 introduces how to select an appropriate smoothing parameter.

Question 2. How much deviations from the trends should be considered as “too much” or markedly?

Answer: The data falls outside the confidence interval should be considered should be considered deviating too much.

4.1 Confidence Interval

The idea of *prediction interval* in linear statistical model [28] is borrowed to create the confidence interval for our model. As mentioned earlier, the error term ε_i in (2) is assumed to be normally and independently distributed with the mean of zero and constant variance σ^2. Under this assumption, the predicted confidence interval of a data point can be computed by its estimated value plus or minus a multiple of predicted errors. The estimated predicted errors is given by [28]

$$s_i^2\{pred\} = MSE + s_i^2(\hat{y}_i)\,, \tag{23}$$

where the *MSE* is the mean square error

$$MSE = \frac{1}{n-df}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\,, \tag{24}$$

and $s_i^2(\hat{y}_i)$ is the sampling variance to the fit at time t_i and is given by the entry *(i, i)* of the matrix $Var[\hat{\vec{y}}]$,

$$Var[\hat{\vec{y}}] = S * S^T * MSE\,, \tag{25}$$

where S is the hat matrix introduced in Section Chapter 2. For the given α significance level, the $100*(1-\alpha)\%$ confidence interval at time t_i is estimated by

$$[\hat{y}_i - Z_{1-\alpha/2} * s_i\{pred\}, \hat{y}_i + Z_{1-\alpha/2} * s_i\{pred\}]\,, \tag{26}$$

where $Z_{1-\alpha/2}$ is the $100*(1-\alpha/2)$ percentile of the standard normal distribution, which can be obtained by looking up from a standard normal table. In the following experiment, the 0.05 significance level ($\alpha = 0.05$) is chosen and the corresponding $Z_{1-\alpha/2}$ is 1.96. This means that a normal observation would fall in the confidence interval with a probability of 95%, or equivalently, the probability that a data point located outside the interval is corrupted is 95%.

4.2 K – The Number of Basis Functions

For the B-Spline smoothing, the number of basis functions, i.e. *K* needs to be determined. As we mentioned in the previous section, a larger *K* can make the curve fit the actual data better, and also result in a higher risk of over-fitting and more computations. There is no gold criterion for the selection of *K* [20]. The strategy used in this work is to only let the smoothing parameter λ control the smoothness of the curve -- In modelling local patterns where only local data (e.g. data of one week) are used, K is selected as the number of observations, whereas in modelling global trends where more data (e.g. data of a whole year) are considered, K is selected as large as the computation allows.

4.3 Smoothing Parameter

Finally, it is the time to unveil the mystery of selecting the smoothing parameter. As we already know, the smoothing parameter controls the smoothness (roughness) of the fitted curve. The basic requirement is that the smoothing parameter should be selected such that the fitted curve can capture the patterns of the load curve that the user would like to see; or, in the context of trend corrupted data detection, the smoothing parameter should be set to make the corrupted data outstanding so that it can be filtered out by confidence interval.

For both the B-Spline smoothing and the Kernel smoothing, the choice of the smoothing parameter depends not only on the data but also on the type of patterns being modelled. If global trends of a time series are focused, a relatively smoother fitted curve with a larger smoothing parameter is preferred; if local patterns are focused, a rougher curve with a relatively smaller smoothing parameter will fit the data better. In the literature, different criteria such as minimizing Mean Squared Error, MISE (mean integrated squared error), CV (cross-validation error sum of squares), and GCV (generalized cross-validation error) have been proposed to find an optimal smoothing parameter [19][20][22]. Unfortunately none of these guarantees to produce the "best result" because ultimately it relies on the user's needs.

To address this issue, an incremental training algorithm is proposed in this paper. The idea is to let the user label some small portion of the data that is corrupted and using the labeled data to search for the “optimal” smoothing parameter. In the process, care must be taken to minimize the user’s effort required for labeling the data. This algorithm is given in Table 1. The algorithm applies to both the B-Spline smoothing and the Kernel smoothing. The range of the smoothing parameter is divided into 10 different levels. For the B-Spline smoothing, for local trend corrupted data detection, the 10 levels are defined as

$$\lambda = 10^{(i-1)/2-9}, i = \{1, 2, ..., 10\}, \tag{27}$$

and for global trend corrupted data detection, they are

$$\lambda = 10^{i-11}, i = \{1, 2, ..., 10\}. \tag{28}$$

The difference between two neighbor values of λ for local trend corrupted data is smaller than global trend corrupted data. This is mainly because local trends modelling is more sensitive to the smoothing parameter since it only consider much less data in the modelling process. For the Kernel Smoothing, these are defined as

$$h = (1 + i / 2) * space, i = \{1, 2, ..., 10\}, \tag{29}$$

where the space is the normalized time lag between two data points in the time series. The incremental training procedure is as follows.

First, the length of the whole load time series T is divided into *k* partitions $T_1,\ldots,T_k$ (see Line 1 in Table 1), equally spaced along the time. Initially, the user only needs to label the first partition T_1 (see Line 2). A smoothing method presented in Chapter 3 is run on T_1 for each of the predefined 10 smoothness levels to identify the trained optimal smoothing parameter (denoted by OSP), which results in the most accurate detection of the labeled corrupted data in the first partition and is done by the function Find_OSP on Line 6 in Table 1. Next, with this trained optimal smoothing parameter, the smoothing method is run to identify the candidates of corrupted data in the second partition (see Line 7). These candidates and smoothing results with the confidence interval are then presented to the user for confirmation (see Line 8).The user's confirmation enables us to label the corrupted data in the second partition, thus, expanding the labeled data by one partition (see Line 9).

Table 1. Incremental Training Algorithm

```
Algorithm Incremental_Training (T, k)
1.      Partition T into $T_1, T_2, ..., T_k$;
2.      User_Label($T_1$);
3.      Labeled_T ← $T_1$;
4.      j ← 2; OSP ← NaN;
5.      While (User not satisfied and $j \le k$)
6.          OSP ← Find_OSP (Labeled_T) ;
7.          Smoothing (OSP, $T_j$);
8.          User_Confirm ($T_j$);
9.          Labeled_T ← Labeled_T ∪ $T_j$;
10.         j++;
11.     End
12.  Unlabeled_T ← T – Labeled_T;
13.  Smoothing(OSP, Unlabeled_T);
```

Until now, the first iteration is completed, in which the labeled data in the first partition is utilized to help label the corrupted data in the second partition. In each iteration, the trained optimal smoothing parameter OSP is updated by rerunning the smoothing program on the all the labeled partitions and the labeled data is expanded by one partition (Lines 6-9). The training process can be stopped at any iteration as long as the user is satisfied with the accuracy of the current model. If the user stops at the end of the *jth* run, the first (*j+1*) partitions have been labeled and the most updated trained optimal smoothing parameter OSP will be used to construct the model to label the remaining (*k-j-1*) partitions (see Line 10 and 11).

The essence of the incremental training algorithm is the dialogue between human being and the computer program. The initial labeling and feedback from the user provide the information for the computer program to find a better smoothness level. The user's effort required is minimized in that the labeling of corrupted data is required for the first partition and for each subsequent partition, only confirmation is required. As more partitions are labeled, the updated model gradually becomes more robust. Another feature of this algorithm is that it can be implemented in an online environment where data continuously arrives in real time since only one new partition of data is required to process with the incremental training algorithm.

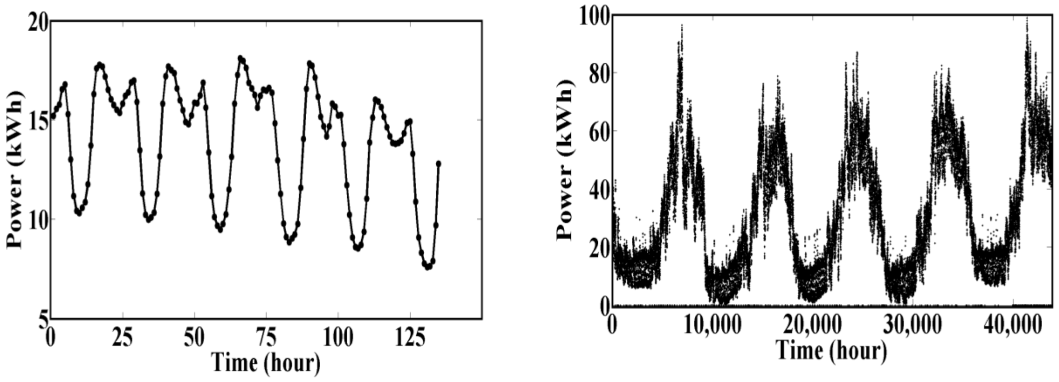
The incremental training algorithm in Table 1 is mainly used for detecting local trend corrupted data. For global trend corrupted data detection, the data is only partitioned into two partitions (training data and testing data) because more global information should be taken into account. The algorithm is shown in Table 2, which simply traverses every smoothness level to find the optimal one based on the training data and then applies the optimal smoothness level to the testing (remaining) data.

Table 2. Global Trend Corruption Detection Algorithm

```
Algorithm Global_Trend_Corruption_Detection (T)
1.    Partition T into $T_1$,$T_2$
2.    OSP ← NaN;
3.    For $\lambda$ in [$\lambda_1$,…,$\lambda_{10}$]
4.        Smoothing($\lambda$ , $T_1$);
5.        User_Confirm($T_1$);
6.        If $\lambda$ leads better results;
7.            OSP ← $\lambda$;
8.    Smoothing(OSP, $T_2$);
```

4.4 Experiment

In this section, the proposed trend corruption detection methods are tested. The real electricity power load data from BCTC (British Columbia Transmission Corporation) was used for our experimental evaluation. The data set includes hourly residential power consumption records in a certain area for the five years from April 2004 to March 2009. Fig. 10(a) shows the data in one week and Fig. 10(b) shows five-year data distribution. The data exhibits both local patterns (by day) and global patterns (by year).



(a) (b)

Fig. 10 Local patterns and global patterns in load curve data

For both locally and globally corrupted data detection, we follow the same experiment procedure. First, the incremental training process described in Section 4.3 is carried out to obtain the trained optimal smoothing parameter. The data used in this training process is called the training data set. Second, the smoothing methods with the trained optimal smoothing parameter are run on a separate pre-labeled testing data set for accuracy evaluation. The testing data set simulates the data on which the user wants to detect corrupted data except that the labeling would not be available in real applications. The standard precision (P), recall (R) and F-measure (F) are used as the accuracy metrics. Precision is the percentage of correctly detected corrupted regions with regard to the total detected regions; recall is the percentage of correctly detected regions with regard to pre-labeled corrupted regions; the F-measure is a harmonic mean of precision and recall, i.e.,

$$F = \frac{2 * precision * recall}{precision + recall}. \tag{30}$$

A larger F-measure indicates more accurate detection.

4.4.1 Detecting Local Trend Corrupted Data

To facilitate the evaluation for locally corrupted data detection, 25 weeks' data was selected from the five-year BCTC data with 168 hourly data points per week. For simplicity, each corrupted data point was considered as one region. The data in the first 12 weeks was used to carry out the incremental training following the procedure in Table 1 where the data in each week was treated as one partition T_j. The trained optimal smoothing parameter OSP was obtained from the incremental training process. The data in the remaining 13 weeks served as the testing data for evaluation, in which all corrupted data points were manually labeled in advance. The model on the testing data was constructed using the OSP obtained from the incremental training process. The precision, recall and F-measure were recorded. For comparison, the other 9 predefined smoothness levels were also tested on the testing data. The results produced by all 10 smoothness levels are shown in Table 3.

In Table 3, the left half shows the results obtained by using the B-Spline smoothing and the right half shows those obtained by using the Kernel smoothing. The "Level" column represents the smoothness level, with 1 being the least smooth and 10 being the smoothest; the “P” column represents precision; the “R” column represents recall; the “F” column represents F-measure. The row with * shows the result produced by the trained optimal smoothing parameter, whereas the row with ** shows the best result that can be produced by any of the 10 predefined smoothness levels. It can be seen from Table **3** that the trained optimal smoothing parameter (labeled by *) produces the result that is very close to the best one (labeled by **). The experiments show that more data in both training data and testing data will lead to better results.

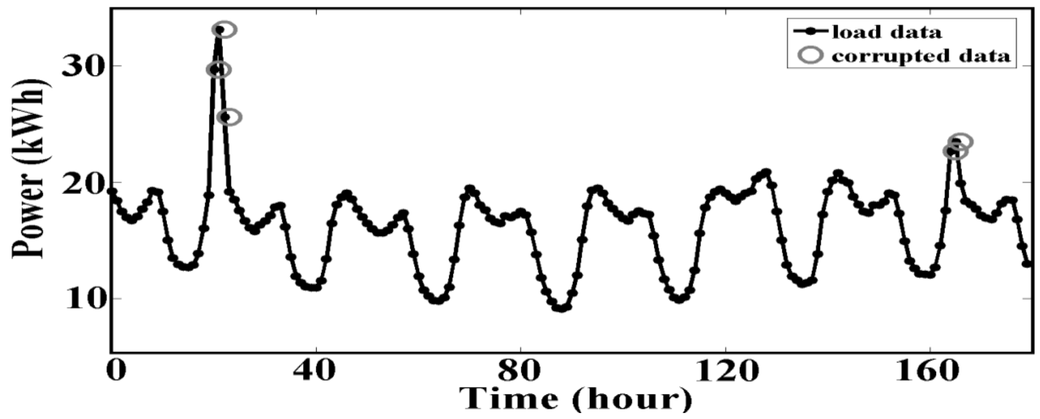
Table 3 Locally Corrupted Data Detection Results

B-Spline Smoothing				**Kernel Smoothing**			
Level	**P**	**R**	**F**	**Level**	**P**	**R**	**F**
1	53%	74%	0.62	1	55%	74%	0.63
2	50%	70%	0.58	2	67%	78%	0.72
3	52%	70%	0.59	3	71%	74%	0.72
4	52%	70%	0.59	**4	90%	78%	0.84
5	73%	70%	0.71	5	89%	74%	0.81
**6	86%	78%	0.82	*6	94%	74%	0.83
*7	85%	74%	0.79	7	94%	74%	0.83
8	78%	61%	0.68	8	88%	65%	0.75
9	82%	61%	0.7	9	88%	65%	0.75
10	81%	57%	0.67	10	88%	61%	0.72

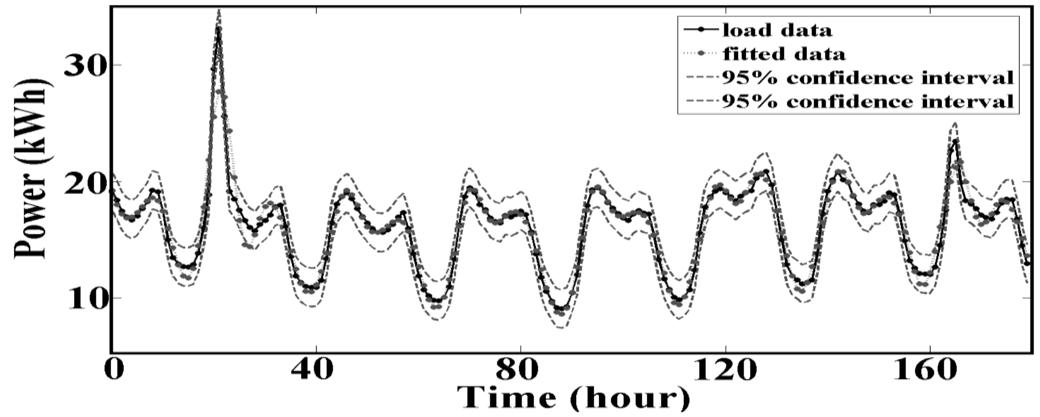
It can be observed that as the smoothness level increases, the F-measure increases and reaches the peak, and then begins to decrease. To understand this behaviour, the fitted curve and confidence interval produced by the B-Spline smoothing for one week’s testing data at the smoothness level 1, 6 and 10 are presented in Fig. 11. The solid line with points represents real observations, the dashed line with points represents the fitted curve, and the dashed lines without points represent the upper bound and lower bound of the confidence interval. Fig. 11 (a) shows the raw data with 5 labeled locally corrupted data points. When the smoothing parameter is selected at Level 1 (least smooth), as shown in the Fig. 11 (b), the curve tends to fit both normal data and corrupted data. When the smoothing parameter increases to Level 6, indicated in Fig. 11 (c), the curve becomes to reveal the real local patterns of the data and filter out corrupted data. Fig. 11 (d) indicates that an overly smooth curve fails to identify some corrupted data because of failure to model the local patterns.

A closer look reveals the following interesting behaviours of precision and recall with respect to different smoothness levels. The precision metric largely increases as the fitted curve gets smoother (i.e., at a higher smoothness level). This increase is contributed by the decrease in the number of wrongly detected corrupted data. As a matter of fact, a smoother curve causes more fitting errors, thus, a larger MSE and a wider confidence interval. In this case, fewer data points fall outside the confidence interval and these data points are most likely true corrupted data. Therefore, a higher smoothness level helps increase the precision metric.

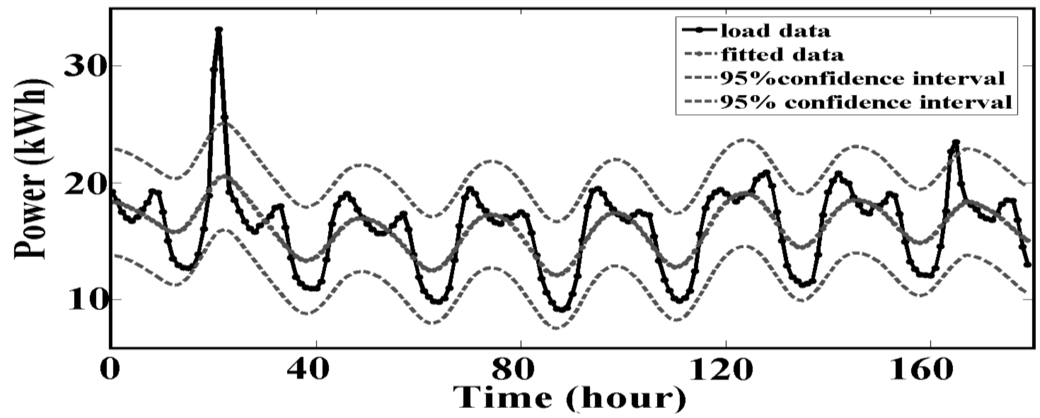
In contrast, the recall metric behaves differently. When the smoothing parameter is set at a low smoothness level, the curve is rough and tends to over-fit the data. In this case, the recall metric is low because corrupted data is also fitted instead of being detected. As the smoothness level increases, the curve is getting smoother and gradually approximating the intrinsic structure of the real data, thus, more abnormal deviations from the function are detected. After reaching the "peak point" (i.e., Level 6 for the B-Spline smoothing and Level 4 for the Kernel smoothing), further increasing smoothness level produces an overly smooth curve, thus, a large MSE. This causes a too wide confidence interval that fails to exclude the corrupted data and therefore leads to a decrease of the recall metric.



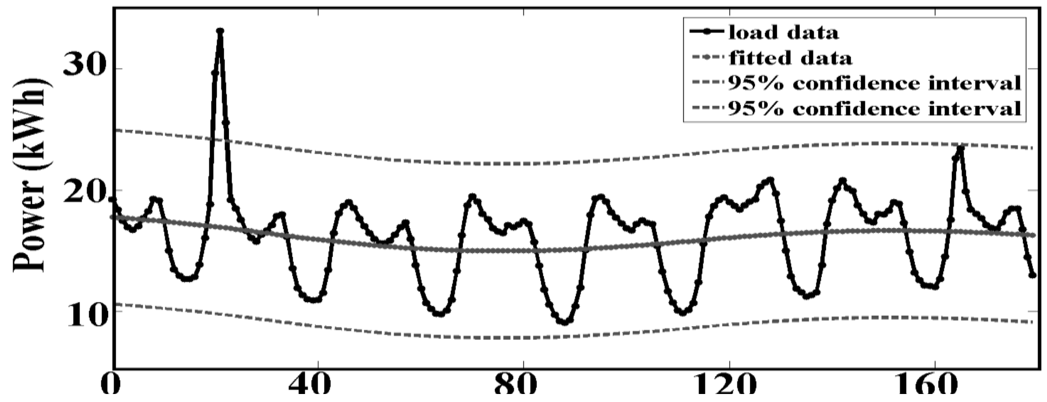
(a) Real observations with 5 corrupted data points in circles



(b) Result for smoothness Level 1



(c) Result for smoothness Level 6

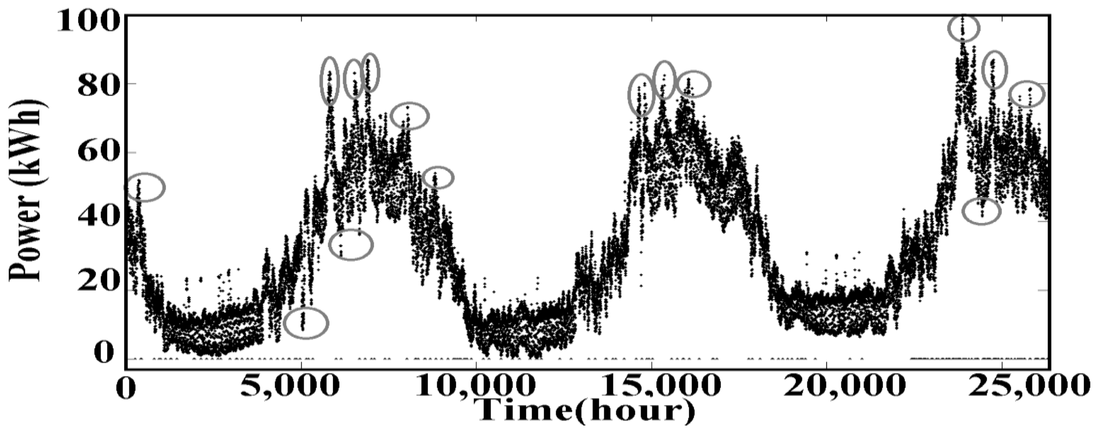


(d) Result for smoothness Level 10

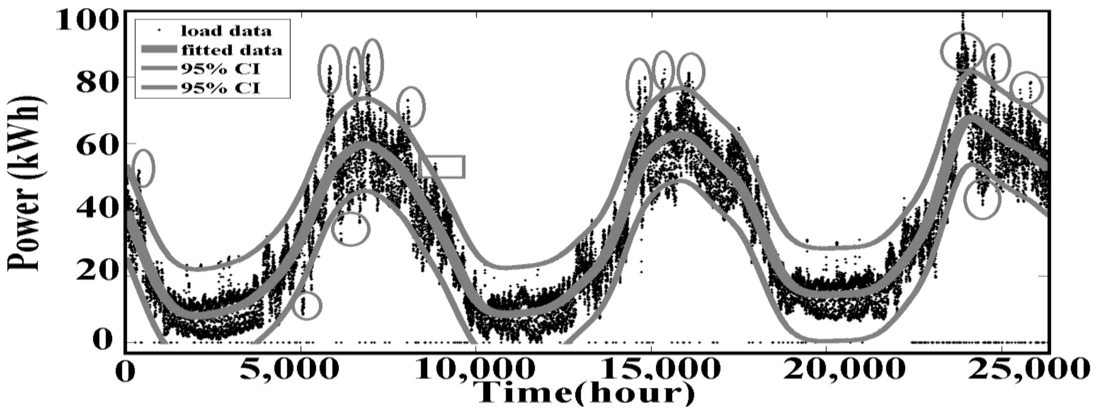
Fig. 11 Local trend corrupted data detection – B-Spline smoothing

4.4.2 Detecting Globally Corrupted Data

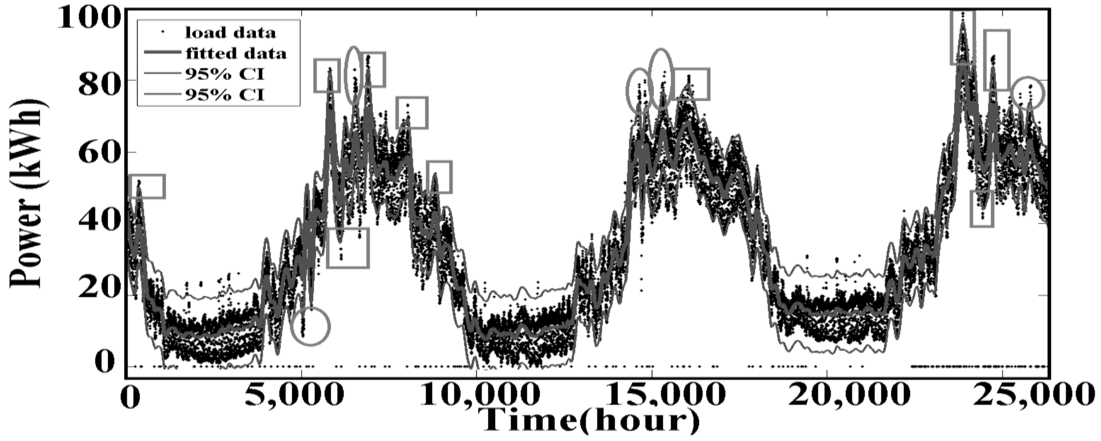
A globally corrupted region is a region that includes a group of nearby corrupted data. For globally corrupted data detection, the data in all the five years was considered. The data in the first two years was used as training data and the rest three-year data was used as testing data. 15 globally corrupted regions in the testing data are manually pre-labeled. The incremental training algorithm was run on the training data to obtain the trained optimal smoothing parameter (OSP). Then the smoothing program was carried out on the testing data with the OSP. The results using the B-Spline smoothing are shown in Fig. 12. The circles in Fig. 12(a) represent pre-labeled globally corrupted regions in the three-year testing data; the circles in Fig. 12(b) and Fig. 12(c) represent correctly detected corrupted regions; the rectangles represent corrupted regions that were not detected. In Fig. 12(b), the fitted curve and confidence interval produced by the OSP (Level 7) are presented. As shown, 14 out of 15 globally corrupted regions are detected. For comparison, Fig. 12 (c) gives the result with the smoothness Level 4, in which only 5 out of 15 are correctly detected.



(a) three-year testing data with 15 globally corrupted regions



(b) B-Spline smoothing result with smoothness Level 7



(c) B-Spline smoothing result with smoothness Level 4

Fig. 12 Global trend corrupted data detection for the three-year testing data

As expected, when the smoothing parameter is at a low smoothness level, such as in Fig. 12(c), the curve tends to model too much detailed information or local patterns. As a result, globally corrupted data is fitted as normal observations rather than being detected. In contrast, when the curve becomes smoother, as modelled in Fig. 12(b), the performance improves accordingly as the local information is ignored and more general data behaviours are modelled. Compared to locally corrupted data detection, a higher smoothness level (i.e., a smoother curve) is usually preferred to detect globally corrupted data. It is also interesting to note that the smoother fitted curve in Fig. 12(b) correctly preserves the yearly seasonality.

Chapter 5. Yearly Discords Detection

As we know, many load curves in the BCTC data set have “un-perfect” yearly periodicity, that is, the periodicity with a period of year. The periodicity is considered “un-perfect” because of the following reasons. First, the length of the periodicity does not strictly stay the same due to different factors. For example, the leap year has one more day than non-leap year. Second, the data in different years is similar to each other, however, not exactly the same.

In load curves with these “un-perfect” yearly periodicities, there are always years or months in which most data deviates very much from the yearly periodicity. They should be considered corrupted with respect to yearly periodical patterns. In this case, trend corruption detection methods may not work because the corrupted regions may be so large that the corrupted data influences the trends of the data.

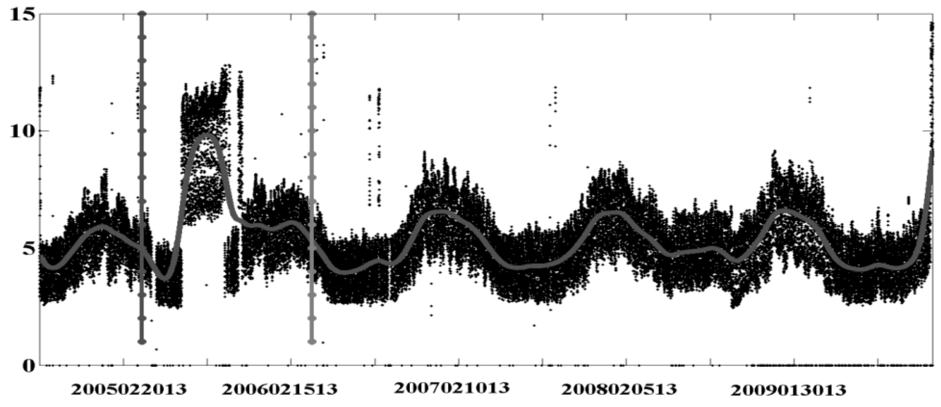


Fig. 13 Yearly discord example

To detect such relatively large regions of corrupted data, the main challenge is that we do not have prior knowledge of the length of the corrupted region. To be a first step of this challenging work, we aim to find the ***yearly discord*** of a long time series. The formal definition of yearly discord will be given shortly. Basically **yearly discord is the most unusual year in the long time series**. An example of yearly discord is illustrated in Fig. 13. The concept of “discord” was proposed by Keogh in [12].

Definition 4. *Distance Function*: Denoted as *Dist*, *Distance Function* is a function that has two time series *S1* and *S2* as inputs and returns a nonnegative value as the distance from *S1* to *S2*. Distance function measures the similarity of the two time series.

Definition 5. *Nearest Match*: Given a collection of time series *S* and a distance function *Dist*, the time series *D* and M_D belonging in *S*, M_D is the *nearest match* of *D* in *S* if for any time series *C* in S, $Dist(D,M_D) <= Dist(D, C)$.

Definition 6. *Time Series Discord*: Given a collection of time series *S* and a distance function *Dist*, the time series *D* in *S* is the time series discord of *S* if *D* has the largest distance to its nearest match. That is, for any time series *C* in *S*, the nearest match M_C of *C* and the nearest match M_D of *D*, $Dist(D, M_D) >= Dist(C, M_C)$.

Definition 7. *Yearly Discord*: Given a long time series *T* containing *n* years’ data, each year’s data is considered as an individual *yearly* time series and all yearly time series form a collection *S*. The time series discord D of the collection S is the *yearly discord* of the time series T.

As we can see from the definition of the discord, the distance function *Dist* plays a key role in defining the discord. The principal of selecting *Dist* is to “mimick human sense”. That is, the distance between two time series should be large when the user thinks they are very much different when she sees the visualized data. The yearly discord discovered by using the distance function should be the really most unusual year that can be confirmed when the users sees the whole n years time series. For example, in Fig. 13, the year between the two solid vertical lines should be detected as yearly discord.

5.1 Distance Function Selection

Based on the observation on the load curves, we find that to mimic human sense, two basic requirements should be kept in mind when selecting the *Dist* function. **First**, the distance function should be robust to data shift and noise. For example, in the Fig. 14(a), the dashed curve is actually a shift of the solid curve. A user would consider the two curves are quite similar, and thus an appropriate distance function should think in the same way. The *Euclidean Distance* is not applicable as it is too sensitive to the noises and data shifting. Measuring the distance between the two curves in Fig. 14(a) by Euclidean Distance would result in a misleading conclusion that the two curves are far away from each other. In terms of noise, human eyes would ignore it naturally, and so should the distance function. **Second**, the distance function should be able to model both “shape” distance and “scale” distance between two time series subsequences. For example, the two curves in the Fig. 14(b), they are the same except that the scale of the dashed curves is constantly larger than that of the solid curves. The some users may consider the scale difference as a huge difference and hence the two curves differ much from each other. On the other hand, some other users may think the two shapes are so similar that the scale difference is ignorable. A good distance function could offer the flexibility that the user can put weights on the shape difference and scale difference.

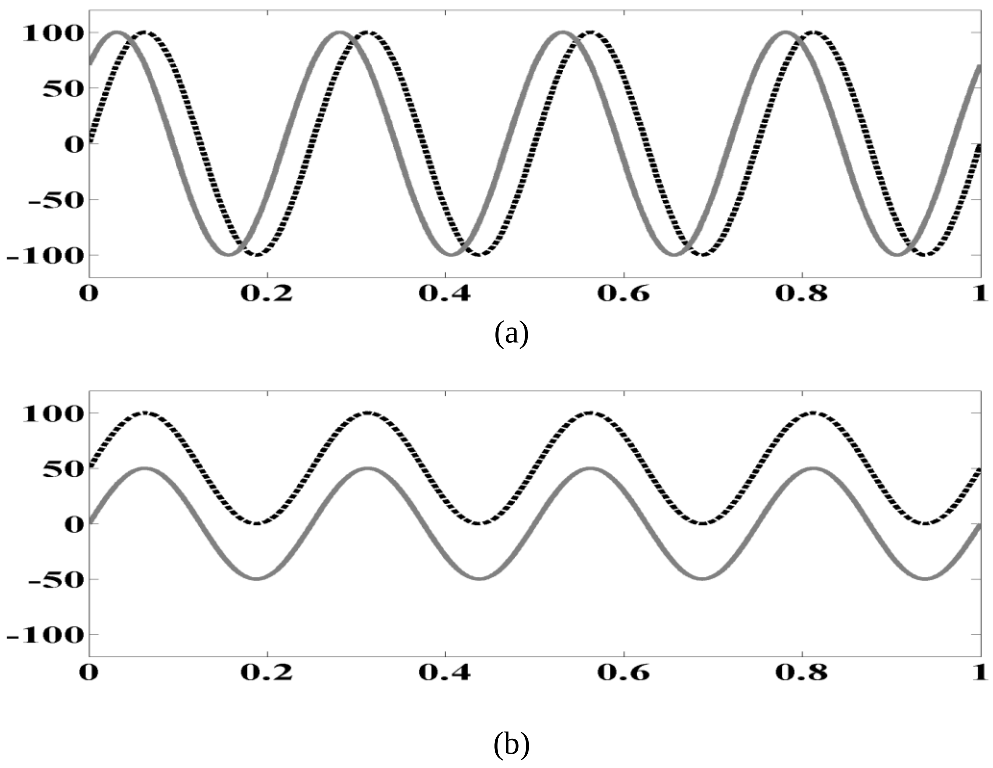


Fig. 14 Data shifting and level difference examples

The key to finding the yearly discord in the load curves is finding a suitable distance function. Basic requirements are robust to noises and data shift, and capability of modelling both shapes and scales.

The load curves contain a lot of noises. To prevent the influence of noise, the proposed method tries to find the yearly discord based on smoothed load curves instead of the raw load curves. It is important to note that the smoothing algorithm should keep those significant patterns and ignore minor deviations caused by noises. The B-Spline Smoothing is chosen to be the smoothing algorithm because of its advantages introduced in previous sections**.** So the distance between two time series S1 and S2 is defined as the distance between the corresponding smoothed time series SS1 and SS2, i.e.

$$Dist(S1, S2) \equiv Distance(SS1, SS2) \qquad (31)$$

To incorporate both shape distance and scale distance, we model the distance between two time series subsequences *SS1* and *SS2* as

$$Distance(SS1, SS2) = \alpha * Shape\_Dist(SS1, SS2) + (1-\alpha) Scale\_Dist(SS1, SS2), \qquad (32)$$

where the *Shape_Dist(SS1,S2)* and *Scale_Dist(SS1,SS2)* measure the shape distance and scale distance respectively, and $\alpha\ (0 \le \alpha \le 1)$ is the weight parameter. To represent the shape of a curve, the proposed method makes use of *curvature* or second *derivative of the curve*. The second derivative measures how the slope of a curve is changing. It contains information of the shape of a curve. After conducting B-Spline Smoothing on a load curve, a continuous function becomes available. The second derivatives at the every data point can be computed based on the function. Thus, we define *Shape_Dist(SS1,SS2)* in (32) as

$$Shape\_Dist(SS1, SS2) \equiv Curvature\_Dist(SS1, SS2) \qquad (33)$$

Where *Curvature_Dist(SS1,SS2)* is the distance between the corresponding curvatures of *SS1* and *SS2*. For the *Scale_Dist(SS1,SS2),* we use the simple, straightforward and effective measure: the difference of “average” value of the *SS1* and *SS2*.

The only issue remaining is the computation of *Curvature_Dist(SS1,SS2),* and in this computation process, the requirement that the distance measure should be robust to data shifting should be enforced. In this work, the data shift problem is addressed by

Dynamic Time Warping (DTW) algorithm, which will be introduced shortly. Thinking a time series as a vector, assuming C1 and C2 are the corresponding curvature vectors of *SS1* and *SS2*, then the *Curvature_Dist(SS1,SS2)* is computed by

$$Curvature\_Dist(SS1, SS2) = DTW(C1, C2)\,. \qquad (34)$$

Combining (31), (32), (33), (34), we obtain the final distance function for two time series S1, S2 as

$$Dist(S1, S2) = \alpha * DTW(C1, C2) + (1-\alpha) * Average\_Diff\,(SS1, SS2) \qquad (35)$$

where SS1 and SS2 is the corresponding smoothed time series of S1 and S2 after B-Spline Smoothing, and C1 and C2 are the corresponding curvature vectors of *SS1* and *SS2*.

5.2 Dynamic Time Warping

It has long been known that Dynamic Time Warping (DTW) is superior to Euclidean Distance (ED) as a distance measure for time series classification and clustering [30]. It aims to find the best mapping (best alignment) between two sequences (or time series). Specifically, given two time series, Q with length of n and C with length of m, where

$$\begin{aligned} Q &= q_1 q_2 \ldots q_n \\ C &= c_1 c_2 \ldots c_m \end{aligned},$$

the goal of DTW is to find a mapping path $\{(i_1, j_1), (i_2, j_2), \ldots, (i_e, j_e)\}$ such that the distance on the mapping path $\sum_{k=1}^{e} | dist(q_{i_k}, c_{j_k}) |$ is minimized, with the following constraints[43]:

- Boundary conditions: $(i_1, j_1) = (1,1), (i_e, j_e) = (n,m)$

- Local constraint: For any given node *(i_k, j_k)* in the path, the possible fan-in nodes are restricted to *(i_k-1, j_k)*, *(i_k, j_k-1)*, and *(i_k-1, j_k-1)*.
- *dist(x,y)* is the distance measure of single points in time series, e.g.

  $dist(x, y) = \sqrt{(x - y)^2}$

Fig. 15 gives an example showing the difference between ED and DTW distance. ED requires one-to-one mapping while the DTW allows many-to-many mapping. DTW can be computed using dynamic programming listed in Table 4 [42].

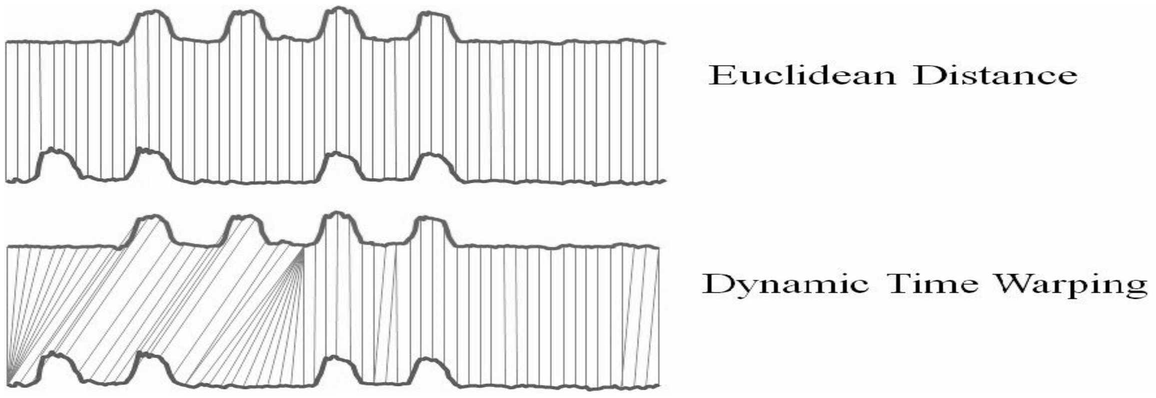


Fig. 15 Euclidean Distance and Dynamic Time Warping

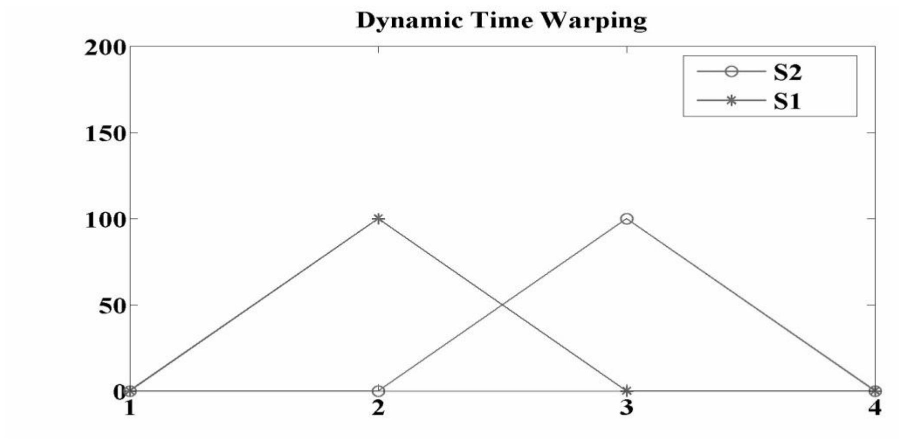
Table 4. DTW-Distance Algorithm

```
Algorithm dtw_distance = DTWDistance(Q, C)
1.   DTW[n+1, m+1];
2.   i ←NaN, j ←NaN, cost ← NaN;
3.   for i ← 1:n
4.       DTW[i, 0] ← infinity;
5.   for j ← 1:m
6.       DTW[0, i] ← infinity;
7.   DTW[0, 0] ← 0
8.   for i ← 1:n
9.       for j ← 1:m
10.          cost:= distance(s[i], t[j])
11.          DTW[i, j] := cost + minimum(DTW[i-1, j  ],
12.                                      DTW[i  , j-1],
13.                                      DTW[i-1, j-1]);
14.      dtw_distance  = DTW[n,m];
```

The **Example 1** illustrates the process of computing DTW distance.

Example 1

Given time series *s1 = (0,100,0,0)* and *s2 = (0,0,100,0)*. Then we have $ED = \sqrt{(0-0)^2+(100-0)^2+(0-100)^2+(0-0)^2} = 100\sqrt{2}$. But for DTW, the distance is 0. The mapping process is given in Fig. 16



S2 \ S1		1	2	3	4
		0	**0**	**100**	**0**
1	**0**	0 →	0 ↘	100	100
2	**100**	100	100	0 ↘	100
3	**0**	100	100	100	0 ↓
4	**0**	100	100	200	0

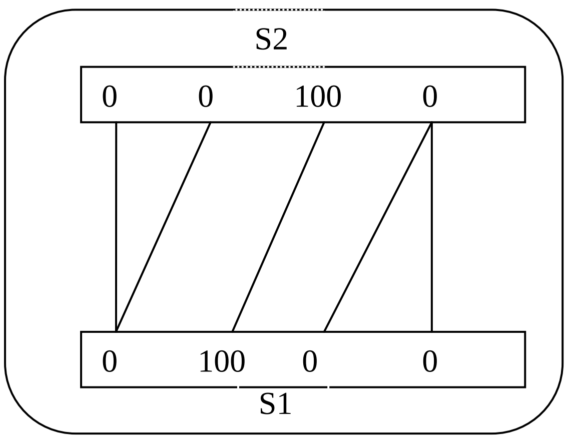


Fig. 16 DTW computing process

The best mapping path is {(1,1), (1,2), (2,3),(3,4),(4,4)}. So the DWT distance is

$$DTW\_Dist = \sqrt{(S1[1]-S2[1])^2+(S1[1]-S2[2])^2+(S1[2]-S2[3])^2+(S1[3]-S2[4])^2+(S1[4]-S1[4])^2}$$
$$= 0 \qquad (36)$$

The complexity of the dynamic programming algorithm is O(nm). As we are going to compute distance between two years' hourly data (around 8760 data points each year), obtaining a distance would need 76737600(or 8760*8760) times' addition computation. And this is too demanding. In practice, it is not necessary to evaluate all possible warping paths since many of them correspond to pathological wrappings. We can add a locality constraint. That is, if q_i is matched with c_j, then $|i-j|$ is no larger than a window's size, *w*. Thus, the complexity becomes *O(nw)*. The locality-adapted version is listed in Table **5**. With the local constraint, the DTW-distance may no longer be strictly symmetric. However, since in our application, the lengths of Q and C are almost the same, the DTW-distance is still approximate-symmetric.

Table 5. Locality Constrained Dtw-Distance

```
Algorithm dtw_distance =  DTWDistance_LC(Q, C, int w)
1.      DTW[n+1, m+1];
2.      i ←NaN, j ←NaN, cost ← NaN;
3.      for i ← 1:n
4.              DTW[i, 0] ← infinity;
5.      for j ← 1:m
6.              DTW[0, i] ← infinity;
7.      DTW[0, 0] ← 0
8.      for i ← 1:n
9.              for j ← max(1,i-w): min(i+w, m);
10.                     cost:= distance(s[i], t[j])
11.                     DTW[i, j] := cost + minimum(DTW[i-1, j  ],
12.                                                 DTW[i  , j-1],
13.                                                 DTW[i-1, j-1]);
14.             dtw_distance  = DTW[n,m];
```

5.3 The whole Algorithm

To sum up, the whole process of yearly discord detection can be illustrate in Table 6 . The function *BSpline_Smoothing* in line 1 smooth the load curve and break it into years of smoothed data. The function *Curvature* in line 2 is used to transform the raw load curve into curvatures of different years. The function *Normalized* in line 7 and line 8 normalize the shape distance and scale distance such that they are comparable and addable. Two for loops compute the distances between any two years in the load curve and find the year that has the largest distance to its **best match**.

Table 6. Yearly Discord Detection Algorithm

```
Algorithm discord_year_id = YearDiscord(LoadCurve, α, w)
1.   YearList = BSpline_Smoothing(LoadCurve);
2.   YearListCurvature = Curvature(LoadCurve);
3.   YearCount ← length(YearList);
4.   i ←NaN, j ←NaN; global_max_min_dist ← -1;  local_min_dist [1:YearCount] ← infinity
5.   for i ← 1: YearCount
6.       for j ← i+1 : YearCount;
7.           shape_dist ← Normalize(DTW_LC(YearListCurvature [i], YearListCurvature [j], w));
8.           scale_dist ← Normalize (AVG(YearList[i]) – AVG(YearList[j]));
9.           tmp_dist ← α* shape_dist + (1- α)* scale_dist;
10.          if  tmp_dist < local_min_dist[i]
11.              local_min_dist [i] ←  tmp_dist;
12.          if  tmp_dist < local_min_dist[j]
13.              local_min_dist[j] ← tmp_dist;
14.      If  local_min_dist[i] > global_max_min_dist
15.          global_max_min ← local_min_dist[i];
16.          discord_year_id ← i;
```

5.4 Experiment

The goal of yearly discords detection is to detect most unusual year of load curve. The similarity measure between yearly data should be consistent with human sense instead of common similarity measure such as Euclidean distance. When the yearly discord of the load curve is deviating much from other years' data, the user could easily find the yearly discord by visualize the load curve. For example, when the user sees the Fig. 17, she can immediately tell the year "2006 -- 2007" is the yearly discord of the load curve. As human sense is the best judge for the accuracy of yearly discords detection, we integrate it into the evaluation of the experiments.

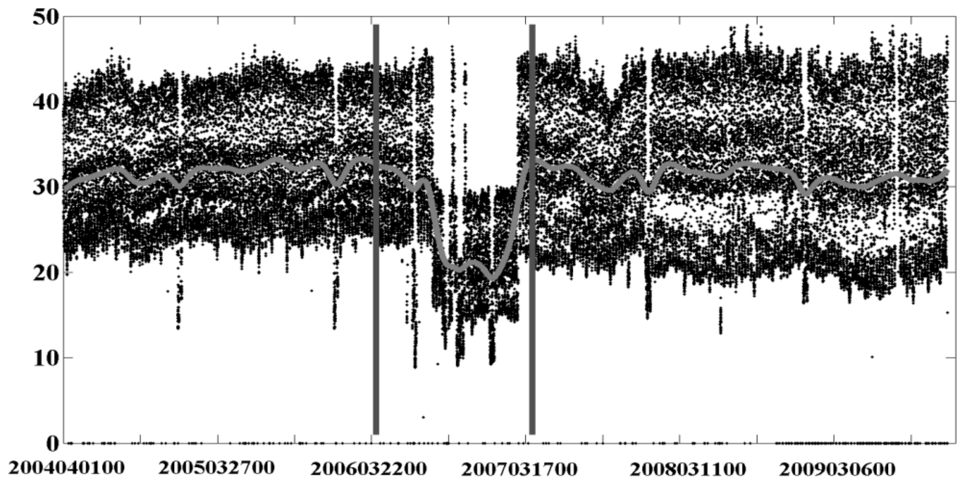


Fig. 17 Yearly discord example 2

Ten load curves with clear yearly discords in the BCTC data set are selected as our experimental data. The yearly discords are "clear" in the sense that the user can immediately tell they are discords when she sees the data. For clarity, the proposed yearly discords detection algorithm is referred to as "dtw_d2" as it incorporates **Dynamic Time Warping** distance measure and **the second derivatives** depicting the shape of the curve.

As introduced previously, there is a weight parameter α between the shape and scale. In the experiment, we set α to be 1, meaning mainly only the shape fact is considered. It is set in this way because the users is more interested in "shape" discords and actually the shape measurements actually to some extends take into account of the scale factor— the smoothed curve is a continuous function and the shape will change much if there is a sudden scale (value) change.

For comparison, we also consider the following algorithms. The "avg_raw" uses the average of raw data to compute distance between two years data; and the "eu_raw" uses the Euclidean distance based on raw load data; the "eu_smooth" uses the Euclidean distance based on the data after B-Spline data; the "dtw_raw" uses the Dynamic Time Warping distance based on the raw load data; the "dtw_smooth" uses the Dynamic Time Warping distance based on the data after B-Spline smoothing. The testing results are shown in Table 7.

Table 7. Yearly Discords Experiment Results

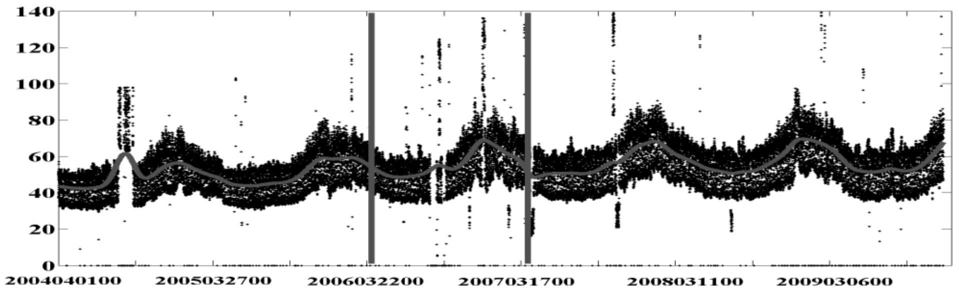
RD: # of real discords; TP: true positive – # of correctly detected discords; FP: false positive -- # of wrongly detected discords; AC: the accuracy of the yearly discords detection – TP/RD

Metrics / Algorithm	RD	TP	FP	AC
avg_raw	**10**	**7**	**3**	**70%**
eu_raw	**10**	**7**	**3**	**70%**
eu_smooth	**10**	**7**	**3**	**70%**
dtw_raw	**10**	**7**	**3**	**70%**
dtw_smooth	**10**	**9**	**1**	**90%**
dtw_d2	**10**	**10**	**0**	**100%**

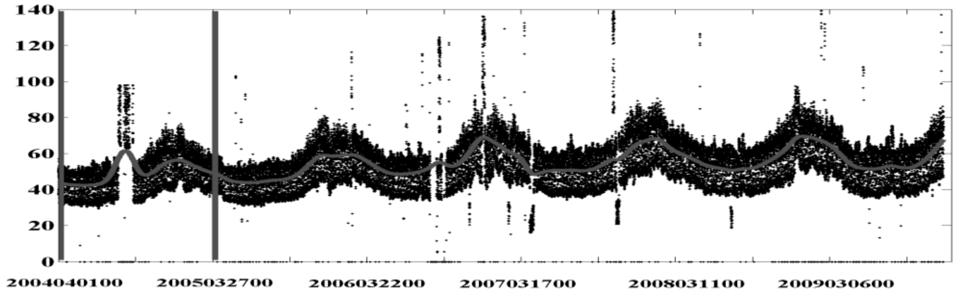
Based on Table 7, the following facts can be observed. First, the proposed method "dtw_d2" outperforms other methods. Second, the avg_raw, eu_raw, en_smooth and dtw_raw have the same accuracy, which is significantly less than the best one. Third, the Dynamic Time Warping distance works better than Euclidean distance, and the working on smoothed curves results in better results than working on raw data. In addition, following interesting questions need to be answered.

First, although lower than the best one, the accuracy obtained by using avg_raw (using simple average of the whole year's data) is not bad – 70% percent. The performance is as good as other more sophisticated methods like eu_raw, eu_smooth and dtw_raw. Why? The reason is that many testing load curves happen to have yearly discords with "outstanding" data values (much lower or much higher), such as the year 2006-2007 in Fig. 17.

Second, why can the smoothed curves lead to better accuracy? The main reason is that the smoothed curves have much less noise than the raw data such that the trends are more obvious. Fig. 18(a) shows the yearly discord detected by using the raw data and the Fig. 18(b) shows the yearly discord detected by using the smoothed data. As we can see, the discord year in Fig. 18(a) has so much noisy data and it mislead the computation of the yearly data distance. Another reason is that the smoothed curve filled in the missing data.

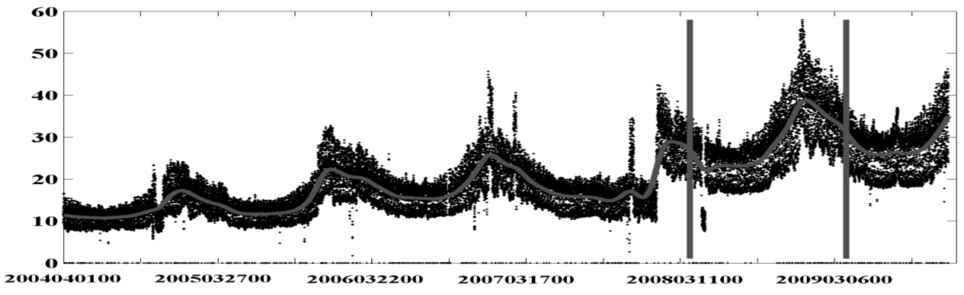


(a) dtw_raw result

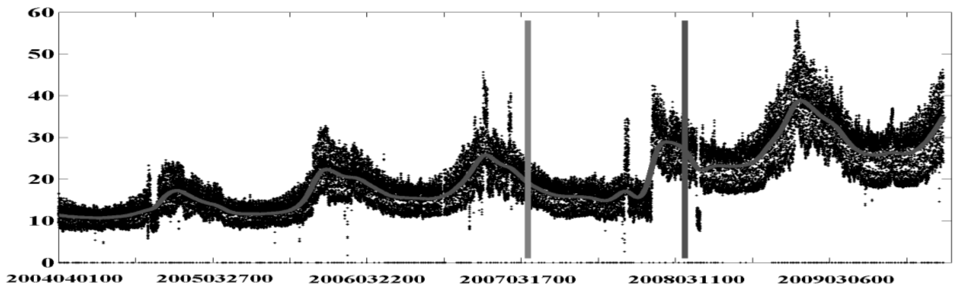


(b) dtw_smooth result

Fig. 18 Different results for dtw_raw and dtw_smooth



(a) dtw_smooth result



(b) dtw_d2 result

Fig. 19 Different results for dtw_smooth and dtw_d2

Third, why would the use of second derivative of the smoothed curve help in detecting yearly discord? The reason is as follows. The load curves may have a natural increasing or decreasing trend. Users usually consider the trends are legal as long as it the shapes between years are similar. Comparing second derivatives instead of the data typically meets this human sense. The Fig. 19(a) and Fig. 19(b) show the different yearly discords detected by using data and second derivatives.

Chapter 6. Filling Missing Data

6.1 Modelling

To make up the missing data in load curves, the following three factors need to be taken into account. First, load curves have increasing or decreasing long-term trends over time. Second, the shapes of load curves are “similar” over years. Third, there are some noise in the data. So the data value at a specific time can be decomposed into 3 components: long-term trend factor, seasonality (perodicity) factor and Irregularity factor. Using a multiplicative model, we can remodel the load curve as [41]

$$\begin{aligned} &\mathrm{Y}(t_i) = Trend(t_i) * Seasonal\_Index(t_i) * \mathrm{Irregularity}(t_i) \\ or \\ &Y = TSI \end{aligned} \tag{37}$$

Where *Y(t*$_i$*)* represents the modelled load value at time *t*$_i$, the *Trend(t*$_i$*)* represents the *long-term trends* of the load curve and *Seasonal_Index(t*$_i$*)* represents the *seasonal index*, i.e. how much the load curve deviates from the long-term trends at time t. For simplicity, we do not consider the irregularity for now.

As mentioned in **Observation 2** in Chapter 4, the Trend(t) could be estimated by the fitted curve $\hat{m}(t)$ with appropriate smoothness level. That is

$$Trend(t_i) = m(t_i) = \sum_{k=1}^{K} c_k \phi_k(t_i)\,. \tag{38}$$

Again, the smoothing parameter selection is a key and can follow the same paradigm of incremental training process introduced in Chapter 4.

In order to estimate *Seasonality(t)*, an important observation needs to be made .

Observation 3. The deviations of normal data from the long-term trends of a load curve are similar at the same times of different years.

The **Observation 3** is illustrated in Fig. 20. The solid horizontal line in the figure reprensents the long-term trends of the load curve data and solid verticle lines represent the deviations of the data from the long term trends.

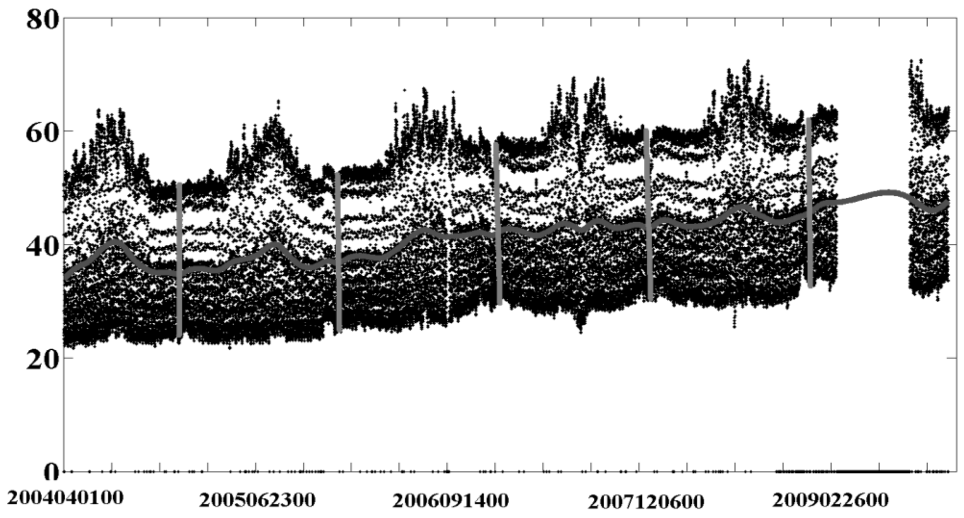


Fig. 20 Deviation from the trends example

For the load data that is **not** missing, the seasonal index can be computed by its definition, i.e.

$$Seasonal\_Index(t_i) = \frac{y_i}{Trend(t_i)} . \qquad (39)$$

Where y_i is the load data value at time t_i. For the data that are **missing**, the seasonal index is estimated by the average of the seasonal index at the same time of its previous and next year, i.e.

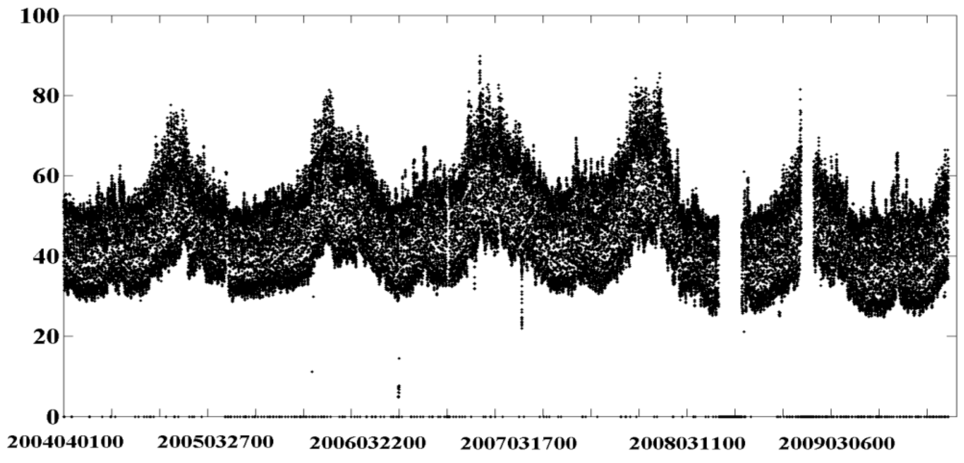
$$Seasonal\_Index(t_i) = \frac{1}{2}(Seasonal\_Index(t_i - Year\_Length) + Seasonal\_Index(t_i + Year\_length)), \quad (40)$$

where Year_length is **length of a year** with respect to the time unit.

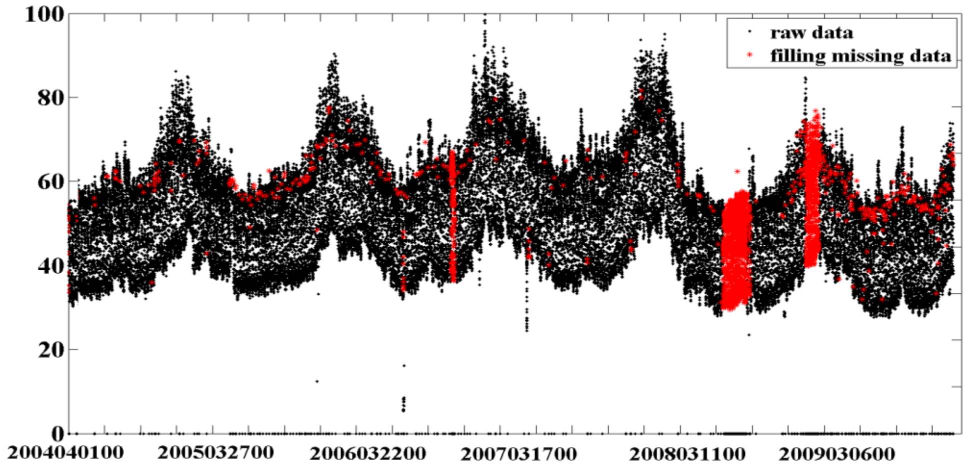
This method can be adapted to fit all load curves, even for those without well periodicities. When dealing with data without obvious yearly seasonality (periodicity), the seasonal index component for missing data in (37) can be computed in different ways. For example, the seasonal index for the missind data could be the average of the seasonal index at the same time of previous month and next month. In effect, the experiments suggest that there are not much difference between different ways of computing seasonal index because the trends component has already taken into account some seasonal effects by fitting the load the curve well.

6.2 Experiment

For a pre-defined specific data set, it is always possible to find an ad-hoc way to make up missing data perfectly. For example, if the data set follows a perfect periodicity, a simple periodical model would be sufficient. However, the practical data set is usually skewed and variable. Such simple periodical mothed cannot work well. The proposed method is generic method which is designed to fit all load curves, even for those without well periodicities. Except proposed method, we have not found an comparision algorithm that have the same capabilities. It does not make any sense to compare a generic method with ad-hoc methods. So the best way to evaluate the proposed the methods would be testing the proposed methods on different load curves with different characteristics. The results are shown to users (experts) to judge the effectiveness. In effect, the proposed methods were tested on the whole BCTC data set, including around 300 load curves and the results are approved by BCTC experts.



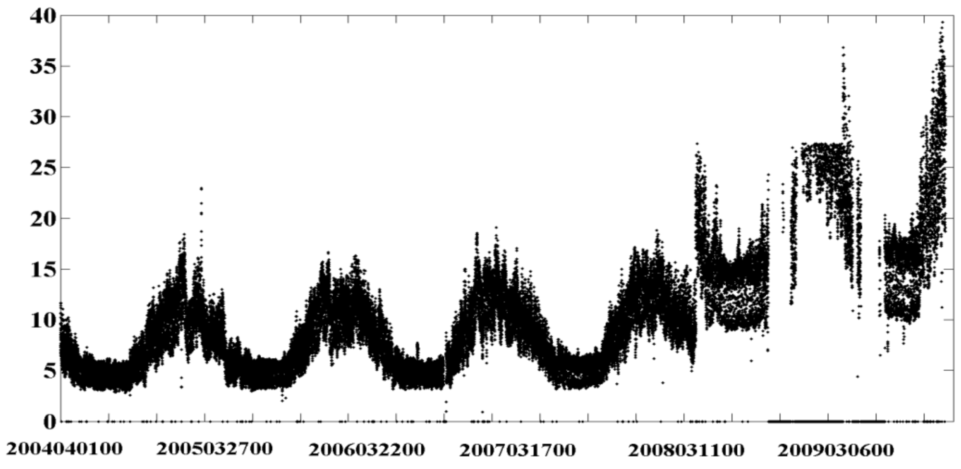
(a) before filling in missing data



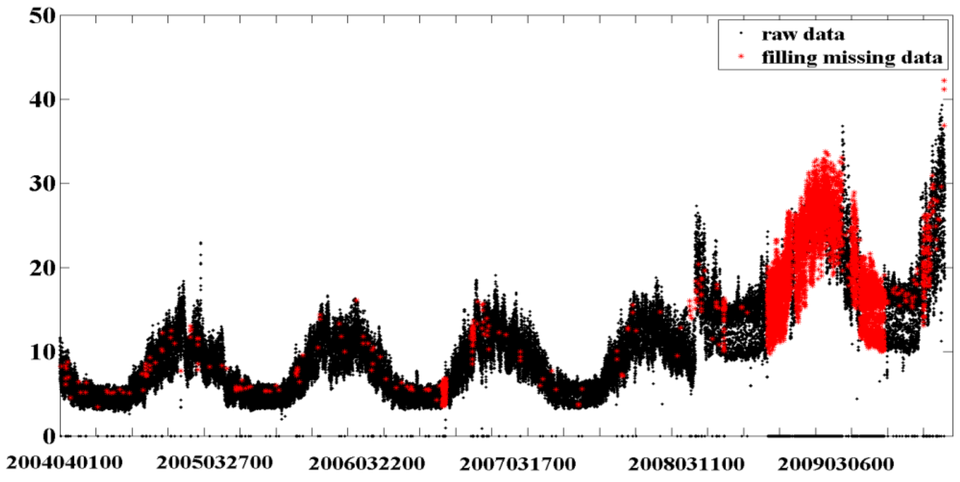
(b) After filling missing data

Fig. 21 Missing data filling in example 1

The first example is a load curve with five and a half years' data. The blank intervals in Fig. 21(a) represent missing data and the points marked with stars (*) in Fig. 21(b) represent the data filled in using the proposed methods. The filling values fit the original trends very well. As shown in the figure, the first four years data values are relatively higher than the last 2 years. Thus, simply using the previous years' corresponding data to make up the missing data would cost overestimate of the missing data.



(a)



(b)

Fig. 22 Missing data filling in example 2

A better example is shown in Fig. 22, which better presents the capability of the proposed method. There is a great amount of missing data in the last two years. Besides, there is a significant trend change the end of the fourth year. Within all these unpleasant challenges, the proposed method still produces very reasonable results shown in Fig. 22(b).

Chapter 7. Conclusion

Load curve data represents the heartbeat of a power system and is crucial in system analysis, real time operations, system visualization, reliability assessment, energy saving, and system planning. Load curve data is noisy and contains corrupted and missing data, which is unavoidable due to random factors and errors in metering and data transfer process. Detecting and correcting corrupted data is the first step for further data analysis and particularly important for the smart grid that will be in place in the future. In this thesis, this problem is addressed as load cleansing problem. The load cleansing problem is decomposed into 4 sub-problems, namely pattern modelling, trend corrupted data detection, yearly discords detection and filling missing data.

7.1 Summary

For pattern modelling and trend corrupted data detection, a practical solution based on well-founded nonparametric regression techniques has been presented. The solution handles both local trends corrupted data and global trend corrupted data in a uniform way through a single smoothing parameter. A challenge in implementation is how to determine the best smoothness level for the smoothing parameter, which ultimately requires user involvement. To address this, the presented solution takes into account of user input while minimizing the user effort required. The nonparametric smoothing methods are incrementally performed in multiple runs so that the user can provide judgments at the initial stage or any stage. The automatic detection and the user input form a man-machine dialogue mechanism. This greatly improves overall performance of the presented methods.

For yearly discord detection, the proposed method aims to mimic human sense in judging corruption of the yearly patterns. Assuming human sense is sensitive to the shape and scale of the load curve. The scale is easy to model while the shape is more challenging. To model the shape of the load curve, the proposed method first smoothes it using the B-Spline smoothing techniques to obtain a continuous curve (fitted curve); then the second derivatives (curvature) of the fitted curve at the time points (forming a vector of curvature or curvature vector) are used to represent the shape. To compute the distance between two curvature vectors, Dynamic Time Warping distance is applied.

For filling in missing data, a multiplicative model incorporating both yearly seasonality and the long-term trends is proposed. The trends are again modelled by using B-Spline smoothing. This intuitive method works very well on almost every load curve in the BCTC data set, which contains more around 300 curves with different characteristics.

7.2 Future Work

The load cleansing problem is actually a real ongoing project at BCTC and the ideal goal is make the whole process of load cleansing fully automatic. In other words, the ideal goal is when there comes a load curve, our algorithm outputs a cleaned load curve, in which all corrupted data and missing data are fixed and the original patterns of the load curve is retained. The work in this thesis is preliminary work and an important step to achieve the goal. Our future work will focus on following points.

First, certain quality control processes should be created for trend corrupted detection. The proposed solution to corrupted data detection makes the judgment of corruption of the data based on the data's deviation from its neighbours. The questions of how much neighbour data should be taken into account and how much deviation should be considered abnormal, are addressed by selection of the smoothing parameter and the confidence interval. However, there exists normal data that deviates very much from its neighbours and it is perfectly normal. For example, the peak values of the load curve of a year usually appear around December, which are much higher than their neighbours. Since they appear every year, they should not be considered corrupted. Thus, the corrupted data detected by the proposed method is better to be considered as potential corrupted data. The potential corrupted need to go through certain quality control processes (such as periodicity checker, which checks whether the corrupted data with very similar characteristics appear repetitively over a period) before it is judged as corrupted.

Second, variable corrupted regions should be addressed specifically. There is corrupted data that cannot be detected in the trend corrupted data detection process because the corrupted regions are large enough to significantly influence the trend. In this thesis, we temporarily avoid the variation of the corrupted region and try to find the yearly discord of a load curve. We have following issues remaining. First of all, discords are not necessarily corrupted. If the discord of a time series is itself normal, then replacing it may affect the quality of the load curve. To address this question, a promising solution is to find a threshold of the distance between a discord to its nearest neighbour (best match). If the distance exceeds the threshold, the discord can be judged as corrupted. Secondly, the lengths of corrupted regions are variable. There is no prior knowledge of the lengths of the corrupted regions. Scanning all possible lengths is too costly and is too sensitive to noises. A possible solution is to predefine some specific lengths of the corrupted regions, such as a month, 3 months, half a year and a year. The algorithms just need to go through the predefined lengths and find corrupted regions with corresponding lengths.

Chapter 8. Bibliography

[1] Wenyuan Li. *Risk Assessment of Power Systems: Models, Methods, and Applications*. IEEE Press and Wiley, 2005.

[2] Smart Grid. Available via http://www.oe.energy.gov/smartgrid.htm.

[3] Ramaswamy Sridhar, Rastogi Rajeev, and Shim Kyuseok. Efficient Algorithms for Mining Outliers from Large Data Sets. *SIGMOD*, 2000.

[4] A. Nairac, N. Townsend, S. King R. Carr, P. Cowley, and L. Tarassenko. A System for the Analysis of Jet Engine Vibration Data. *Integrated Computer-Aided Engineering*, 1999.

[5] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier Detection Using Replicator Neural Networks. *DaWaK*, 2002.

[6] Victoria J. hodge and Jim Austin. A Survey of Outlier Detection Methodologies. *Artificial Intelligence Review*, 2004.

[7] Li Wei, Eamonn Keogh, and Xiaopeng Xi. SAXually Explicit Images: Finding Unusual Shapes. *ICDM*, 2006.

[8] Dragomir Yankov, Eamonn Keogh and Umaa Rebbapragada. Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets. *ICDM*, 2007.

[9] Philip K. Chan and Matthew V. Mahoney. Modelling Multiple Time Series for Anomaly Detection. *ICDM*, 2005.

[10] Matthew V. Mahoney and Philip K. Chan. Trajectory Boundary Modelling of Time Series for Anomaly Detection. *KDD*, 2005.

[11] Stan Salvador, Philip Chan and John Brodie. Learning States and Rules for Time Series Anomaly Detection. *Applied Intelligence*, 2005.

[12] Eamonn Keogh, Jessica Lin and Ada Fu. HOT SAX: Finding the Most Unusual Time Series Subsequence: Algorithms and Applications. *ICDM* 2005.

[13] Yingyi Bu, Tat-Wing Leung, Ada Wai-Chee Fu, Eamonn Keogh, Jian Pei, and Sam Meshkin. WAT: Finding Top-K Discords in Time Series Database. *SDM*, 2007.

[14] Ada Wai-Chee Fu, Oscar Tat-Wing leung, Eamonn Keogh, and Jessica Lin. Finding Time Series Discords Based on Haar Transform. *ADMA*, 2006.

[15] Eamonn Keogh, Stefano Lonardi and Chotirat Ann Ratanamahatana. Towards Parameter-Free Data Mining. *KDD*, 2004.

[16] Agata Fallon and Christine Spada. Detection and Accommodation of Outliers in Normally Distributed Data Sets. [Online]. Available via http://www.cee.vt.edu/ewr/environmental/teach/smprimer/outlier/outlier.html.

[17] A. J. Fox. Outliers in Time Series. *Journal of the Royal Statistical Society. Series B (Methodological), volume 34*, pages 350-363, 1972.

[18] Greta M. Ljung. On Outlier Detection in Time Series. *Journal of the Royal Statistical Society*, Series B (Methodological), volume 55, pages 559-567. 1993.

[19] Wolfgang Hrdle. *Applied Nonparametric Regression.* Cambridge University Press, 1990.

[20] J.O. Ramsay and B.W. Silverman. *Functional Data Analysis, Second Edition.* Springer, 2005.

[21] Carl de Boor. *A Practical Guide to Splines*. Springer, 2001.

[22] M.C. Jones M.P. Wand. *Kernel Smoothing*. Chapman & Hall, 1995.

[23] Barnett, V. and Lewis, T. *Outliers in Statistical Data*, 3rd Edition, John Wiley & Sons, New York, pages 397-415, 1994.

[24] Abraham, B., and Yatawara, N. A Score Test for Detection of Time Series Outliers. *J. Time Ser. Anal*., 9, 109-119, 1988.

[25] Abraham, B., and Chuang, *A.* Outlier Detection and Time Series Modelling. *Technometrics*, 31, 241-248. 1989.

[26] Schmid, W. The Multiple Outlier Problem in Time Series Analysis. *Australian*. *J. Statist.*, 28, 400-413. 1986.

[27] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control, 4th Edition*. Wiley, 2008.

[28] Michael H Kutner, Christopher J. Nachtsheim, John Neter, William Li. *Applied Linear Statistical Models, 5th Edition*. McGraw-Hill/Irwin, 2004.

[29] James W. Taylor. An Evaluation of Methods for Very Short-Term Load Forecasting, Using Minute-by-Minute British Data. *International Journal of Forecasting*, 2008, Vol. 24, pp. 645-658.

[30] Chotirat Ann Ratanamahatana, Eamonn Keogh. Making Time-series Classification More Accurate Using Learned Constraints. *SDM*, 2004

[31] NIST. [Online]. Available via: http://www.itl.nist.gov/

[32] T.M.J.A. Cooray. Applied time series : analysis and forecasting. Oxford, U.K. 2008.

[33] Chris Chatfield and Mohammad Yar. Holt-Winters Forecasting: Some Practical Issues. *The Statistician*, Vol. 37, No. 2, Special Issue: Statistical Forecasting and Decision Making (1988), pp. 129-140.

[34] Chris Chatfield. Calculating Interval Forecasts. Journal of Business & Economic Statistics, Vol. 11, No. 2 (Apr., 1993), pp. 121-135.

[35] Christopher Chatfield. The *Analysis of Time Series: An Introduction*. Boca Raton, FL : Chapman & Hall/CRC, 2003.

[36] Taylor, J.W. 2008. Using Exponentially Weighted Quantile Regression to Estimate Value at Risk and Expected Shortfall. Journal of Financial Econometrics, 6, 382-406.

[37] Taylor, J.W., P. E. McSharry. 2007. Short-Term Load Forecasting Methods: An Evaluation Based on European Data. IEEE Transactions on Power Systems, 22, 2213-2219.

[38] Taylor, J.W. 2007. Forecasting Daily Supermarket Sales Using Exponentially Weighted Quantile Regression. European Journal of Operational Research, 178, 154-167.

[39] Taylor, J.W., R. Buizza. 2003. Using Weather Ensemble Predictions in Electricity Demand Forecasting. International Journal of Forecasting, 19, 57-70.

[40] Taylor, J.W. 2003. Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing. Journal of Operational Research Society, 54, 799-805.

[41] David M. Bourg. *Excel Scientific and Engineering Cookbook*. O'REILLY. 2006

[42] Ralph Niels. Dynamic Time Warping, An Intuitive Way of Handwriting Recogonition? *International Graphonomics Society*, 2005.

[43] Keogh, E. Exact indexing of dynamic time warping. *28th International Conference on Very Large Data Bases*. 2002.

[44] Hui-Hsiung Kuo. White Noise Distribution Theory. CRC Press LLC. 1996.