UNIVERSITY UTRECHT

MASTER THESIS

Time Series Machine Learning Technique with Application to Barcelona Metro Station Energy Minimization

Author:
Jeroen Jansze

Supervisors:
dr. G.A.W. Vreeswijk
dr. S. Guo

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

in the

Intelligent systems group
Information and Computing Sciences

May 2013

Declaration of Authorship

"The best way to predict the future is to invent it."

Alan Kay

UNIVERSITY UTRECHT

Abstract

Faculty of Science
Information and Computing sciences

Master of Science

Time Series Machine Learning Technique with Application to Barcelona Metro Station Energy Minimization

by Jeroen Jansze

This thesis is part of the SEAM4US project, which goal is to minimize the energy consumption of the Barcelona metro station. The energy minimization is done by so-called model predictive control, i.e. management of the energy using systems based on a time series prediction. Here we focus on such a prediction. We make use of the popular Fourier transformation in combination with trend detection and validate our method by testing on different data sets and comparing with well-known techniques. Furthermore we conclude that this technique is very usable for the SEAM4US project and probably for a lot of time series prediction.

Acknowledgements

I would like to thank my supervisors, dr G.A.W. Vreesijk and dr. S. Guo for the guidance during this thesis project. Furthermore I thank my parents for financial en mental support.

Contents

List of Figures

List of Tables

Abbreviations

ANN	**A**rtificial **N**eural **N**etwork
BN	**B**ayesian **N**etwork
CEMA	**C**entered **E**xponential **M**oving **A**verage
CSA	**C**ross **S**pectral **A**nalysis
DAG	**D**irectedl **A**cyclic **G**raph
DBN	**D**ynamic **B**ayesian **N**etwork
DFT	**D**iscrete **F**ourier **T**ransformationr
FFT	**F**ast **F**ourier **T**ransformation
GPD	**G**rossl **D**omestic **P**roduct
IFT	**I**nverse **F**ourier **T**ransform
MAPE	**M**ean **A**bsolute **P**ercentage **E**rror
MLP	**M**ulti **L**ayered **P**erceptron
MPC	**M**odel **P**redictive **C**ontrol
MSE	**M**ean **S**quared **E**rror
NRMSE	**N**ormalized **R**oot **M**eam **S**quared **E**rror
PCA	**P**rincipal **C**omponent **A**nalysis
PDF	**P**robability **D**ensity **F**unction
RMSE	**R**oot **M**eam **S**quared **E**rror
SEAM4US	**S**ustainable **E**nergy m**A**nage**M**ent for **U**nderground **S**tations
SFA	**S**low **F**eature **A**nalysis
SG-smoothing	**S**avitzky-**G**olay smoothing
SNN	**S**piking **N**eural **N**etwork
SSE	**S**um of **S**quared **E**rrors
TDNN	**T**ime **D**elayed **N**eural **N**etwork
TVR	**T**otal **V**ariation **R**construction

Chapter 1

Introduction

Predicting events in time is something that all organisms do on a large part of every day [1]. It helps them to decide which sequence of actions they need to perform to survive. Humans for example need to decide when and how much to eat every day based on a estimation of how much time to the next meal. Because the reality is far too complex to predict directly, humans predict only on a few things they believe to be true. This is implicitly a model of those variables and how they are related to each other. If I miss the train for example I predict that I will also be late for the next train in the planning I made at home. This is based solely on the departing times I read and believe to be true. However there are numerous variables I could use to make a prediction. In my personal opinion predicting is a phenomenon that is strongly connected to intelligence, since it influences beliefs and therefore planning. Even in a tighter view on human estimation [2].

Predictions with a large number of variables in time, the so-called time series, are done mainly with numerical systems [3] [4] [5] [6]. As humans have difficulty finding patterns directly from time series, machines can process numbers faster and can therefore find those patterns in reasonable time. Popular choices of prediction methods include ARIMA methods [6] and methods based on networks [3]. Both systems are based on a reinforcement learning paradigm. That is a prediction gets corrected by a (partial) next observation.

In this project we explore the possibilities of predictions directly from the underlying patterns. An applicable system is created only very recently [7]. The basic idea is to see a batch of data as the outputs of a function (just like with ARIMA and neural networks). If we can find or learn this function we can predict the next time steps. More specific [7] breaks a sequence of data values, here after called a signal, into component signals which are easier to predict. Moreover this method in Chapter 5.

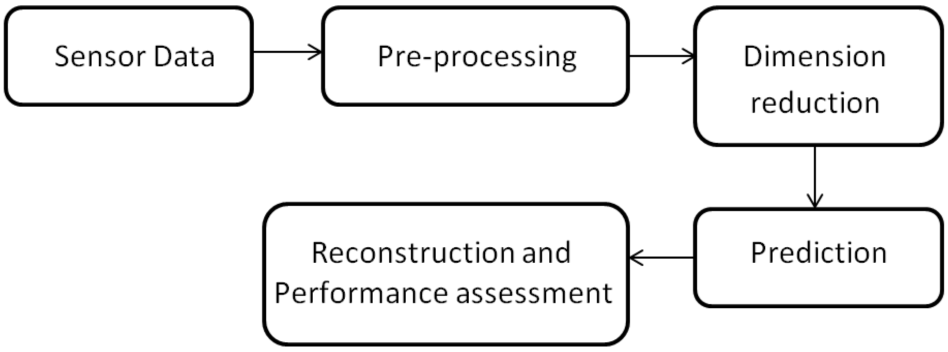

Figure 1.1: View of the prediction scheme.

We compare our method with and ARIMA method and a network method. All networks are modules in our proposed prediction scheme. This scheme incorporates features that are normally used for time series prediction where data is obtained through sensors. First we denoise the incoming data. Second we reduce dimensions (if there are a lot of them) in order to keep the system fast. After that we perform the prediction on the reduced dimensions. And finally generalize the prediction to all dimensions and calculate performances. Figure 1.1 gives a schematic view of the scheme. We use this scheme in order to make the resulting system applicable for the SEAM4-US project, that is described in Chapter 2.

1.1 Time line

This thesis project will take 9 months in total to finish. From December to February we will survey literature for techniques that can be used to build up the scheme. After that we will conduct research on the theory of the prediction techniques. Since this is the main part of the project we reserve about 3 or 4 months for this. In this time we also want to test the solution on simulations with data from the Barcelona metro station. After this step we want to implement our findings, review our work and finally give suggestions for future research.

Chapter 2

Problem formulation and background

In this chapter we formulate the problem we want to solve in this thesis project. First some background and a general description of the problem. Then a translation into a formal constrained based problem. And finally a strategy of solving the problem and on which part we will focus. Almost all examples and simulations in this project are based on data from the project described below.

2.1 SEAM4US

This thesis project is part of the SEAM4US-project[1]. SEAM4US stands for Sustainable Energy mAnageMent for Underground Stations. As the name does suspect the objective of the project is to minimize energy consumption on a regional level. Since underground transportation systems are among the largest energy consumers in regions, reducing their energy consumption have a large impact on the energy consumption as a whole. A way of reducing energy consumption is by optimising the management of systems that use most of the energy. For example lightning, ventilation, elevator and temperature regulation systems. On this project work people all over Europe together in order to create a modular system that manages subsystems of metro stations.

2.1.1 Goal

The goal of the SEAM4US project is a reduction of 5% of the energy consumption of a metro station on yearly basis. This is a relative small percentage, but with a metro

[1]http://seam4us.eu/

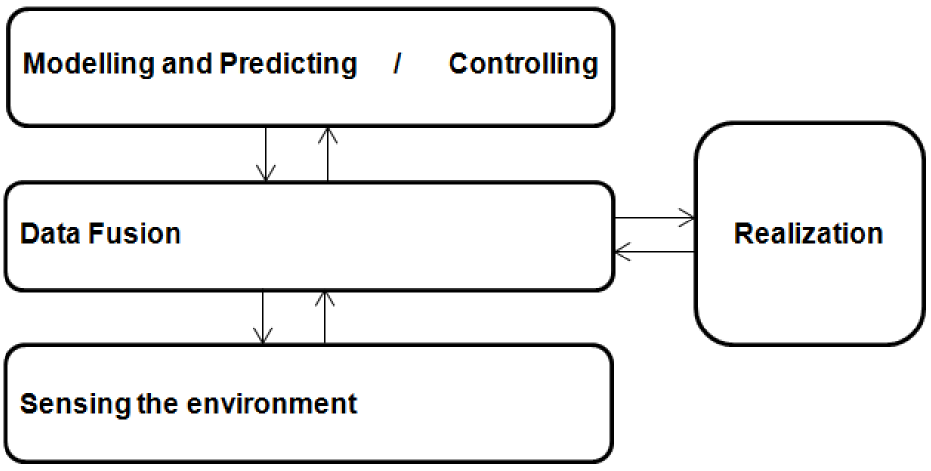

FIGURE 2.1: Schematic view for the SEAM4US Approach.

station the size of Barcalona metro station, it is equivalent to the electricity consumption of 700 households of a whole year. If every metro station in Europe makes use of this system we would have an immense energy saving.

2.1.2 Focus

Figure 2.1 gives a schematic overview of the modules of the system. It starts of course with sensor data from the environment, which is in our case the Barcalona metro station. That data is then collected at the next stage and structured, which we will call data fusion. This is done by a system called hydra, which is the core of the entire system. From there the data will be transferred to modelling and prediction. This will be the main focus of this project. In this part we will make a prediction model based on the data provided. This prediction is an estimation whether we can turn down some system in order to save energy, with adhering to certain constraints. This prediction is then used in controlling the systems. For this purpose we want to make use of an agent based approach. The prediction part and the control part are closely related because the prediction part is dependent on feedback from the controlling system and the controlling system is dependent on the prediction. Finally the commands for the subsystems are collected by the core system and then send to the subsystems themselves. Also in this stage we get feedback from the systems, e.g. in case of system malfunction.

Our focus will be on the prediction and the control of the subsystems. The input for this module is the structured data and the output are commands for subsystems.

2.2 Scientific problem formulation

We will define the general problem more formally as a objective-constraint problem.

2.2.1 Objective

The objective of the SEAM4US project is to realize energy consumption minimizing strategies s_i, where i is an index of a subsystem (i.e. a fan or a light). A strategy for a fan for example can be on which times during the day it is on the highest frequency or for a light at which time steps of the day it is off. Strategies are thus sequences of commands and are based on the prediction of context information, e.g. temperature and airflow. More formally the objective is:

$$\int_t \sum_i e(s_i) dt \tag{2.1}$$

Where $e(s_i)$ is the energy consumption with use of s_i. Which states we look for the best combination of strategies, that in our case minimize energy consumption.

2.2.2 Constaints

If we minimize the energy consumption for a metro station, we should of course make sure that the people using this facility keep a certain level of comfort. Turning off ventilation for example saves a lot of energy, but makes the place very inhospitable in a matter of hours. For this purpose we define constraints that are divided into two categories, namely the comfort level constraints and the operational constraints. The comfort Level constraints are to keep the metro fully operational and hospitable. We define these constrains as:

- $Temp_L \leq Temp(x,t) \leq Temp_H$
- $Airflow_L \leq Airflow(x,t) \leq Airflow_H$
- $Hum_L \leq Hum(x,t) \leq Hum_H$
- $Co2_L \leq Co2(x,t) \leq Co2_H$
- $Lum_L \leq Lum(x,t) \leq Lum_H$

Where $Temp$ stands for the temperature, Hum stands for humidity, $Co2$ the percentage of CO_2 in the air and Lum for lumination. L and H are the minimum and maximum respectively. Furthermore x is a value and t the time at which that value occurs.

In order to keep all equiment running smoothly we also have one operational constraint. When a fan for example, gets turned off/on too many times in a short period, it will has a higher chance of breaking down. To make sure that strategies don't change to radically between time steps we have te constraint:

$$|s_i(t+1) - s_i(t)| < C \tag{2.2}$$

C is a maximum at which states may change between time steps.

The solution we want to create unifies the objective with the constraints.

2.2.3 General solution strategy

The solution strategy we want to pursue lies in Model Predictive Control (MPC) [8]. As the name does suspects MPCs use models that represent the behaviour of complex processes. Based on the models a prediction is made which can be used for influencing the system in some way. A simple example is the case where the model predicts high temperature for tomorrow and we make sure the fans are on high frequency. Furthermore MPC is a multivariable process consisting of a dynamic model, past observations and a cost function over strategies on all time steps. Our main focus is creating the predictive model.

2.3 Model construction and prediction

Since there are a lot of fans, elevators, lights and so on, we get a huge amount of data that keeps growing over time. In order to cope with this we have to have an efficient algorithm that updates our model incrementally, i.e. as data comes in the model gets updated. The model itself consists of state of the system. With it we want to predict the dependent variables, e.g. the temperature, in order to influence the independent variables, e.g. the fan frequency. In the next chapters we will describe how we can create such a model and draw predictions from it.

Chapter 3

Survey of smoothing and filtering techniques

In this chapter we give a broad overview of techniques that can be used for prepossessing sensory data. Measurements from sensors are often noisy. If for example someone stands before a temperature sensor at the time the sensor measures we get a spike in the data. Of course we dont want this spike to influence our prediction. Two common ways to do this are by smoothing or filtering. Because we have high dimensional data it is important we look for techniques that can be used incrementally, i.e. process new data as it becomes available. Also the computational costs of techniques becomes important on bigger data sets.

3.1 Smoothing

In order to de-noise the data we can smooth the data. This simply means we give an approximation function in which every data point gets modified so that the outcome is a smoother signal and the heights of spikes are lessened. This will leave us with a more robust system. In our case we also look for something that is incremental since we have high dimensional data that keeps growing over time, i.e. as time progresses more data points come in. Also on the point of high dimensional data is that we want an smoothing algorithm that has as low complexity as possible. Many different techniques are introduced and this section we will give a review of some of them, namely:

- moving average
- exponential smoothing

- Savitzky-Golay smoothing
- total variation reconstruction

3.1.1 Moving average

We start off with one of the oldest and best-known techniques called the moving average. Smoothing the signal is done by taking the averages of sequences of raw data. The length of those sequences have to be odd, because we want for every data point a modification based on the average of a sequence where that data point is the middle. The length of the sequence over which we take the average is called the filter width f. For example if we have a f of 5 and we modify the 3rd data point in an data array, we take the average of the data point 1 to 5. The formula of the moving average is given by:

$$y_t = \frac{\sum_{i=-k}^{k} x_{t+i}}{f} \tag{3.1}$$

Where y_t is the moving average of a sequence of which x_t is the middle and $f = 2k + 1$.

Examples are given in appendix A for a couple of filter widths. Larger filter widths smooth the data more, i.e. the smoothed signal reacts less to fluctuations in the original signal. From this the trade-off between reducing noise and distorting the signal becomes clear. The moving average can be used incrementally, which means that if new data comes in it can be used for prediction. However there will be a lag of the size $(f - 1)/2$, since we need those data points to calculate the moving average for the middle data point.

3.1.2 Exponential smoothing

Exponential smoothing [9] is in some way similar to the moving average. It also calculates averages but gives weights to every data point. More specifically as the values get older values they get assigned exponentially decreasing weights. Predictions with the exponential smoothing technique will be influenced more by the recent data points. Our only use of exponential smoothing will be the smoothing itself, which can be done with the following formula:

$$y_t = \alpha x_t + (1 - \alpha) y_{t-1} \tag{3.2}$$

Where y_t is the smoothed value of data point x_t and α is a smoothing parameter, the higer this parameter the more it reacts to fluctuations in the original signal. Initially, $y_0 = x_0$.

The lower the smoothing parameter the more the signal is smoothed as can be seen in appendix A. These examples also show that the more the signal is smoothed the more lag we introduce, i.e. the smoothed signal is behind the original signal. This means that we might lose information about trends in the data which will be very important in the prediction step. With the moving average we just shifted the signal $filterwidth - 1/2$. To cope with this problem for exponential smoothing we can apply the exponential smoothing also backwards and average over the two [10], called Centered Exponential Moving Average ($CEMA$). The complete formula for the smoothing then becomes:

$$y_t = \alpha \left(x_t + \frac{\sum_{i=1}^{N} (1-\alpha)^i \left(y_{t-i} + y_{t+i} \right)}{2} \right) \tag{3.3}$$

Where N is the number of data points.

Examples in appendix A show indeed that the lag is gone as to the normal Exponential smoothing. Since there is no theoretical way to get a good α, researchers rely on experimentation for this.

3.1.3 Savitzky-Golay smoothing

Another smoothing technique that has properties in common with the moving average is called Savitzky-Golay smoothing (SG-smoothing) [11], [12]. While moving average takes the average of a certain window, the SG-smoothing fits a polynomial to the data points in that window. This is preferred over the moving average since it distorts the original signal less.

For SG-smoothing we need two parameters. One is the filter width and the other is the polynomial degree we want the data to fit too. Most common degrees are two to five. Smoothing with degree two is also called quadratic smoothing.

As SG-smoothing makes use of least squares regression [13], we can efficiently compute the smoothed value for every data point with the formula:

$$y_t = \frac{\sum_{i=n_L}^{n_R} c_i x_{t+i}}{\sum_{i=n_L}^{n_R} c_i} \tag{3.4}$$

Where n_L is the number of data points to the left, n_R the data points on the right and c_i is a polynomial coefficient.

It is computationally very inefficient to fit a polynomial for every data point, but since the least squared regression takes only matrix inversions we can calculate the coefficients in advance [12].

This technique allows for non-linear smoothing, i.e. we take a window with more points on the right than on the left of the data value we want to smooth. Which means that with a little more computational effort we can eliminate lag. Of course under the basic assumption that values are dependent on past and future values.

Examples of SG-smoothing are in appendix A.

3.1.4 Total variation reconstruction

Total Variation Reconstruction (TVR) [14] is somewhat different from the techniques introduces so far. First of all it assumes that:

$$y_t = x_t + w_t, \qquad t = 0, ..., t = N - 1 \tag{3.5}$$

Where y_t is the noisy original signal, x_t a piece wise constant signal and w_t is white noise, i.e. noise with a zero mean. That noise is white is a reasonable assumption, since noise is probably normally distributed.

Secondly, TVR uses all the data points instead of a moving window, making it less practical for large data sets.

TVR minimizes the total variation and a distance measure between x and y. The total variation for a signal x_t is defined as:

$$TV(x) = \sum_{i=1}^{N-1} |x_{i+1} - x_i| \tag{3.6}$$

And a distance measure, e.g. the Sum of Squared Errors (SSE) between the approximated signal y and the input signal x:

$$SSE(x, y) = \frac{1}{2} \sum_{i=1}^{N-1} (x_i - y_i)^2 \tag{3.7}$$

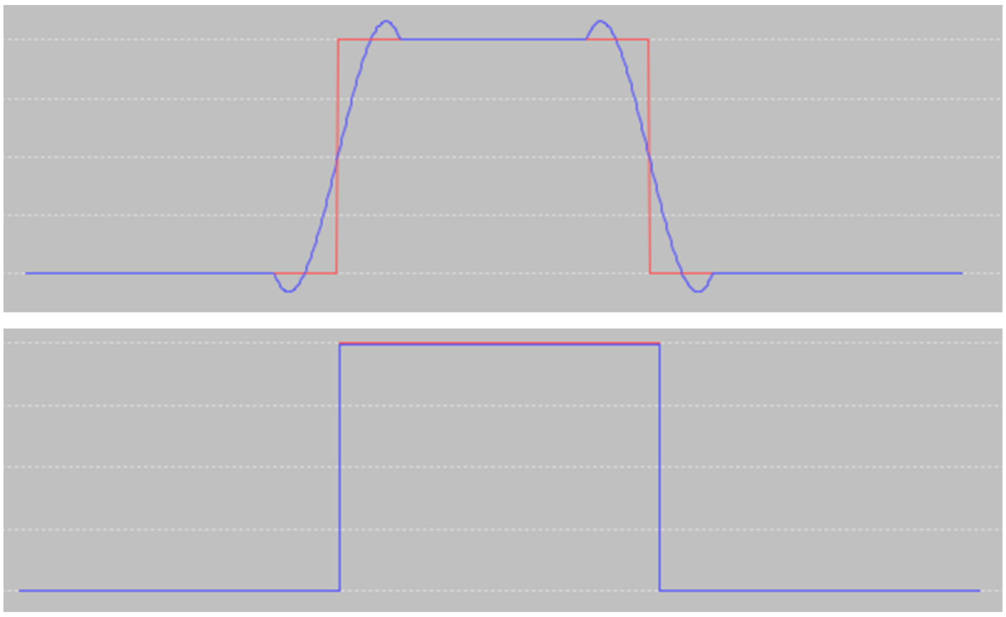
FIGURE 3.1: A rapid variation in data. For the top figure quadratic smoothing was used and for the bottom figure TVR. While the variation is in this case important to model, the TVR method works well. The quadratic smooth however loses some of the signal, i.e. distorts the signal

Which leads to the minimization problem:

$$arg \min_x \sum_{i=1}^{N-1} (x_i - y_i)^2 + \lambda \sum_{i=1}^{N-1} |x_{i+1} - x_i| \tag{3.8}$$

Where λ is a regulation parameter, which emphasizes the total variation term, in this case a higher λ means more smooting.

The solution to the minimization problem 3.8 is the smoothed signal. TVR was designed for data with rapid variations. While quadratic smoothing (SG-smoothing with polynomial degree 2) smooth these variations out, TVR preserves the occasional rapid variations in the original signal. Figure 3.1 shows this clearly. The Figure and the examples of TVR in appendix A were made by a java implementation of a recent developed fast algorithm [15].

3.2 Filtering techniques

Another way of de-noising data is by applying filters. Using filters is mostly the same as smoothing. Most filters can be used for smoothing and smoothing techniques can be written as filters but not all. There are a few differences. Firstly, filters are applied during evaluation, which means as new data comes in it can be processed directly. Some smoothing techniques like the Exponential smoothing do this too, but moving average for example does not. Secondly, filters are computationally less expensive that moving window techniques in general. An example of this is already given, namely the SG-smoothing. If we would try to fit a polynomial for every data point we would have an algorithm with an high run time complexity. However if we write the technique as a filter, that is as a set of coefficients with linear combinations, we can make a faster algorithm. Thirdly, some smoothing techniques cannot be undone, while filters are reversible. An obvious example is the moving average. Fourthly, there is a small difference in approach. Adjusting the parameters on filters relate to reducing the frequency of noise in data, while smoothing parameters relate to the reduction of the magnitude of noise. Another difference is that filters are also used for data compression, while most smoothing techniques are not. This has also to do with reversibility of the smoothing operations.

Here we will review the following filters:

- Fourier filter
- wavelet filter
- Kalman filter
- particle filter

3.2.1 Fourier Filter

The Fourier filter is based on the Discrete Fourier Transformation (DFT) [16]. The theory was devised in the 19th century by Joseph Fourier for a chemistry application. It shows that every discrete function can be written as the sum of complex sinusoids also known as frequencies. Time series are said to be in the time domain, since they are time dependent. Now we transform the data into frequencies and project it on the frequency domain. Since noise in data has high frequency, i.e. peaks in short time span, we can easily filter our data if it is written as such a function. Instead of looking for the total function there is a way to quickly find the coefficients of such a function with the formula:

$$Y_k = \sum_{t=0}^{N-1} x_t \cdot e^{-i2\pi k/N} \tag{3.9}$$

And its inverse:

$$x_t = \frac{1}{N} \sum_{t=0}^{N-1} Y_k \cdot e^{i2\pi k/N} \tag{3.10}$$

Where Y_k is the new sequence known as the Fourier series and i is the complex part of a number.

The coefficients that are relatively low represent the short peaks and the higher peaks represent the longer waves in the data. We can now just change every coefficient between two certain values to zero. This technique is called thresholding. The threshold values determine the smoothing, i.e. if more values are set to zero we get a smoother result. After this we use the Inverse Fourier Transformation (IFT) to get an approximation of our input data.

The examples in appendix B are an implementation of the Fast Fourier Transform (FFT) [17] of the commons math library[1]. This algorithm has a runtime complexity of $O(nlogn)$, which is an significant improvement over a nave implementation which takes $O(n^2)$ runtime. However this algorithm has one extra assumption, namely the input data has to be a power of two. Since our example data has 2293 data points we show filtering for 2048 data points. Furthermore we only use the real part of the complex output, since the input is purely real valued.

3.2.2 Wavelet filter

The wavelet filter is based on an approach similar to the Fourier filter [18]. As the Fourier transformation is based on infinite waves, the wavelet transformation is based on wavelets. Wavelets are finite waves that most of the time start high and die out over time. An example is given in Figure 3.2. This wavelet can then be scaled and dilated with the data. Again we are looking for coefficients rather than a function. In the case of a wavelet it is an square-integrable sequence called the wavelet series. The wavelet we start with is referred to as the mother wavelet and for every mother wavelet the coefficients are calculated differently. As an example we take the simplest of mother wavelets called the Haar wavelet. The formula for the Haar wavelet (plot in Figure 3.3:

$$y_t = \begin{cases} 1 & 0 \leq t \leq 1/2, \\ -1 & 1/2 \leq t \leq 1, \\ 0 & otherwise \end{cases} \tag{3.11}$$

A simple algorithm for calculating the Haar Wavelet Transform (HWT) on data:

[1]http://commons.apache.org/math/

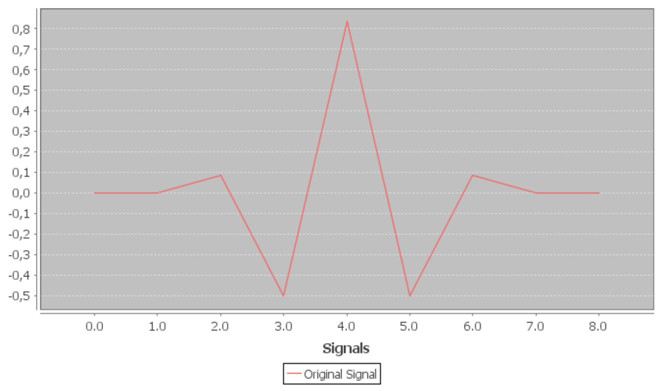

FIGURE 3.2: Linear B-spline wavelet.

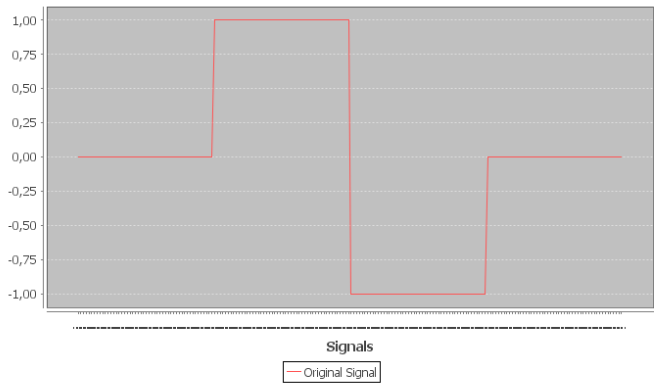

FIGURE 3.3: Haar wavelet.

1. Initialize an Array with the same size as the input data.
2. Calculate the average of each consecutive pair of data values.
3. Calculate the difference between the average and the first value of each pair.
4. Fill the first half of the array with the averages and the second with the differences.
5. Repeat with the first half of the Array.

Since we divide every iteration by 2, our input size needs to be a power of 2 just as with the Fourier transform.

Then we can do thresholding just as with the Fourier transform and reverse the algorithm above to get an approximation of our input data. Examples are given in appendix B.

3.2.3 Kalman Filter

The Kalman filter is a filter that is studied more extensively [19]. The filter recursively minimizes the Mean Squared Error (MSE) of the estimated state with the actual state of a process or signal. Furthermore it keeps track of the current state of the process and the measurements (the data), so we can use it incrementally. The algorithm for Kalman filtering:

1. Initialize:
   (a) A, the state transition matrix.
   (b) B, the control input
   (c) H, the vector representing the state of the measurements.
   (d) Q, the covariance matrix for the noise of the process.
   (e) R, the covariance matrix for the noise of the measurements.
2. Compute the Kalman gain K_t.
3. Update the estimate using K_t and a measurement.
4. Update the covariance matrix with the estimate.
5. Calculate the state for t + 1.
6. Go to 2.

Formula for the Kalman gain:

$$K_t = P'_t H^T (H P'_t H^T + R)^{-1} \tag{3.12}$$

Where P_t is the differential of the MSE on time t.

Formula for updating the estimate:

$$\hat{x}_t = \hat{x}'_t + K_t(z_t - H\hat{x}'_t) \tag{3.13}$$

Where $\hat{x}_t$ is the updated estimate of data point x, $\hat{x}'_t$ is the old estimate of x and z_t is the actual measurement. The term $z_t - H\hat{x}'_t$ is called the innovation.

Formula for updating the error covariance with maximal gain:

$$P_t = (I - K_t H)P'_t \tag{3.14}$$

Where I is the vector representing the innovation.

Formulas for calculating the next state:

$$\begin{aligned} \hat{x}'_{t+1} &= A\hat{x}_t \\ P_{t+1} &= AP_tA^T + Q \end{aligned} \tag{3.15}$$

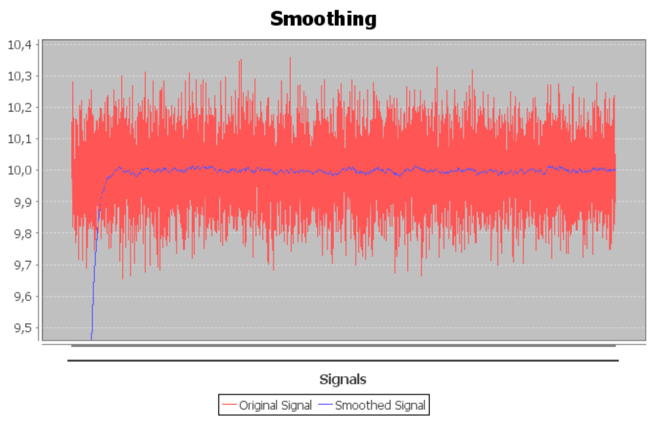

FIGURE 3.4: Kalman filter on very noisy data.

An example on the example data is given in appendix B. However this filter works better on data that has noise of bigger magnitude. This is shown in Figure 3.4. These examples where implemented with the commons math library.

3.2.4 Particle Filter

The last filter we want to discuss is the particle filter [20]. Like the Kalman filter it is based on estimates from states. A Kalman filter however assumes that noise is Gausian and the system is linear, while this is not needed for the particle filter. The state for a particle filter is a Probability Density Function (PDF) represented as a set of weighted samples. As the number of samples is going to infinity, this set converges to the PDF. The sum of the weights has to be 1. The algorithm for the particle filter:

1. Initialize: sample particle values $x_{i,t}$ from an initial distribution $p(x_0)$.
2. Preform Importance sampling step:
   (a) Sample $x_{i,t}$ from a freely chosen distribution $q(x_t|x_{i,t-1}, z_t)$ (z_t is a data point)
   (b) Update corresponding weights with: $w_{i,t} \propto w_{i,t-1} \frac{p(z_t|x_{i,t})p(x_{i,t}|x_{i,t-1})}{q(x_{i,t}|x_{i,t-1},z_t)}$
3. Normalize weights so they sum up to 1.
4. Extract state estimate.
5. Resample if necessary.
6. Go to 2.

Resampling is a solution for the degeneracy problem, i.e. after a few iterations most particles have weights very close to zero and we waste computational effort on them. Resampling throws away all particles with weights that are too small and draws a new set of particles from the old set of particles according to the weight importances. A method that looks like natural evolution.

A disadvantage of the Particle filter is that is computationally expensive filter as compare to the Kalman filter. The complexity of the algorithm above grows linearly with the number of particles which relate to the accuracy of the estimates. So there is a clear trade-off between the accuracy and the complexity. For large data sets it is reasonable to assume that the noise is Gaussian which leaves us with the Kalman filter as the better choice.

Two examples are given in appendix B.

Chapter 4

Survey of prediction techniques

In this chapter we will review some techniques that can be used for building the model and get useful predictions from it. We review techniques that shrink the data, finds underlying patterns in the data and techniques that use past observations in order to predict.

4.1 Interdependencies

First we want to shrink the data if possible, in order to make other techniques applicable. A general way to do this is looking for strong interdependencies in the data. An example of this are the temperature sensors in the Barcelona metro station. If one sensor measures an increase in temperature it is likely that other temperature sensors will do the same. In this case we dont have to make a prediction for all variables related to temperature but only one and then maybe scale it to the interdependent variables.

4.1.1 Principal Component Analysis

Principal Component Analysis (PCA) [21] is a tool for shrinking the dimensions of large data sets. In this analysis we look for the variable that is the best representation of a given data set. PCA consists of the following steps:

1. Subtract the mean of the values of a variable from every data point.
2. Calculate the covariance matrix.
3. Calculate the eigenvectors and the eigenvalues of the covariance matrix.
4. Mark the eigenvectors with the highest eigenvalues as the principle components and put them in a so called feature vector.

5. Multiply the transposed feature vector with the mean subtracted transposed data.

The number of principle components can be chosen freely in range of 1 to the number of dimensions. The trade off with PCA is the variance information left in the transformed set and the reduction of dimensions. The total variance that is accounted for is exactly the sum of the eigenvalues (computed in step 3) in the feature vector divided by the sum of all the eigenvalues.

In order to get the old data back we multiply the transposed feature vector with the data we got from step 5 and add the mean.

4.1.2 Cross-Spectral Analysis

As PCA looks for cross-correlations in the time domain Cross-Spectral Analysis (CSA) [22] does this for the frequency domain. In other words CSA looks for correlations at different frequencies. If we have two variables X_t and Y_t then the cross-covariance is defined as:

$$\gamma_i(XY) = \sum_j X_j Y_{i+j} \tag{4.1}$$

The cross-spectrum $\Gamma_{XY}(\omega)$ is the Fourier transformation (see section 3.2.1) of $\gamma_{XY}(\omega)$, where ω is the given frequency. $\Gamma_{XY}(\omega)$ can be decomposed in a real part $\Lambda_{XY}(\omega)$ and an imaginary part $\Psi_{XY}(\omega)i$. From those components we can calculate the amplitude spectrum with the formula:

$$A_{XY}(\omega) = \sqrt{\Lambda_{XY}(\omega)^2 + \Psi_{XY}(\omega)^2} \tag{4.2}$$

The amplitude spectrum is like a normal set of amplitudes but now in the frequency domain. With these amplitudes we can check if two signals correlate. This is done by use of the coherency spectrum which can be calculated with:

$$\kappa_{XY}(\omega) = \frac{A_{XY}(\omega)^2}{\Gamma_{XX}(\omega)\Gamma_{YY}(\omega)} \tag{4.3}$$

$\kappa_{XY}(\omega)$ is a dimensionless number that works the same as the normal correlation coefficient.

4.2 Patterns

If we take for example a week consisting of 7 days of 24 hours with observations every 5 minutes (the example data to make the figures in Chapter 3), then probably there is some pattern that shows days. This is probable because at night there are less people which means that many subsystems will need less energy. Of course it is very useful to know these patterns if we are to make a good prediction.

4.2.1 Fourier Analysis

Fourier analysis is already described in section 3.2.1. The only difference is that our goal is different. In section 3.2.1 we wanted to filter out the high frequency noise and here we want to isolate certain components of the signal which will be mostly the low frequencies. The low frequencies often show periodicity in data. As is shown in the Fourier filter examples in appendix A we can isolate the day filter easily since it is the most low frequency component. We can do this for every part of the week and look for example for rush hour patterns that maybe important for the energy consumption of the subsystems.

4.2.2 Wavelet Analysis

Just like the Fourier analysis we can do a wavelet analysis (see section 3.2.2). That is we wavelet transform the data and we set certain wavelet coefficients to zero. This basically eliminates those localized frequencies. If we then transform the data back the frequencies that were not eliminated can be shown. The advantage over the Fourier transformation is that we can find the frequency at different time steps and not only find some wave over the whole timespan. In a sense the wavelet analysis is a more refined version of the Fourier analysis. Another advantage is that wavelet transforms are often computationally less expensive and in that way better applicable for large datasets.

4.2.3 Slow Feature Analysis

A more recent developed analysis that looks for patterns in data is the Slow Feature Analysis (SFA). The purpose of SFA is extracting the slow varying components of a quickly varying signal. We do this by searching for a function $g(x)$ that given the input signal has the most slowly varying component as output. We do this by applying a vector of basic functions $h(x)$, which are all monomials of degree 1 or 2, to the input signal. Functions in $h(x)$ maybe nonlinear and the result is a nonlinear expanded signal

$z(x)$. After this we sphere the expanded signal to make sure we get a zero mean and a unit covariance matrix. This is basically choosing the matrix S in:

$$z'(t) = S(z(t) - \langle z \rangle) \tag{4.4}$$

Where $\langle z \rangle$ is the average of $z(x)$.

Choosing S can be done with PCA (see section 4.1.1). After that we take the derivative of $z(t)$ and look for the eigenvector with the smallest eigenvalue a. Again this can be done with PCA. Now the output signal $y(t)$ is given by:

$$y(t) = a^T z'(t) \tag{4.5}$$

Which shows the most slow varying component.

4.3 Time series prediction

Now we have arrived at the main part of the predictive model, namely the function that actually makes the prediction. Although we have already introduced some simple functions that can make predictions, like the exponential smoothing, the Kalman filter and the Particle filter, this mini survey is more about Artificial Neural Networks (ANNs) [23]. Reasons for this choice are that ANNs are used in prediction systems before with success and the empirical prove that these networks are far more robust that most techniques. Moreover ANNs can be used for online learning on large data sets but are rather costly to train.

4.3.1 MultiLayer Perceptron

One of the simplest type of ANN is the MultiLayer Perceptron (MLP) [3]. It consist of an input layer, one or more hidden layers and an output layer in that topological order. The nodes of each layer are fully connected to the next. The connections represent weights. Each node has an activation function which describes the information flow through the network. If this is a linear function the whole MLP will be linear, but for nonlinear prediction we need a nonlinear activation function. The training of the network is done by a supervised learning technique called back propagation which is based on the minimization of the SSE. The algorithm is as follows:

1. Feed the information, e.g. time series, to the input layer and calculate via all layers the output for the output layer.

2. Calculate backwards through the network the errors for each node.

3. Update the weights with $w'_{ij} = w_{ij} + \alpha(error_j \cdot output_i)$, where α is a learning parameter $0 < \alpha < 1$.

Commonly the Sigmoid function is used and in that case the error is calculated as:

$$error_i = output_i(1 - output_i)(target_i - output_i) \tag{4.6}$$

This is because we want to use the gradient, derivative of the activation function, instead of the absolute value to make sure we have the lowest SSE over the data in the end. After the training we feed the latest data points to the network and the output values of the output layer is the prediction of the next time step.

4.3.2 Spiking Neural Network

A more sophisticated ANN is the Spiking Neural Netwok (SNN) [4]. This network model is said to be more biological plausible since SNN allows for much faster data processing. This is because every signal is encoded as a number of spikes called the spiketrain, rather than a real value. So instead of neurons that fire every propagation cycle we have a threshold for every neuron and firing happens only when the sum of its inputs is above the threshold. This sum is called the potential and is leaky, meaning that newer spikes make higher contributions.

The activation function in a SNN is some step-function, which results in every output being the same. However after the firing of a neuron it is scaled by the weight so that every spike may have a different effect on the potential of the next neuron. The information through the network is encoded in the timing of the spikes and not in the height of the spikes.

The training algorithm can look like that of the MLP. But other novel methods have been developed [24] or another based on the Hebbian rule [25]:

$$w'_{ij} = \frac{w_{ij} + \alpha\bar{v}\bar{x})}{\kappa} \tag{4.7}$$

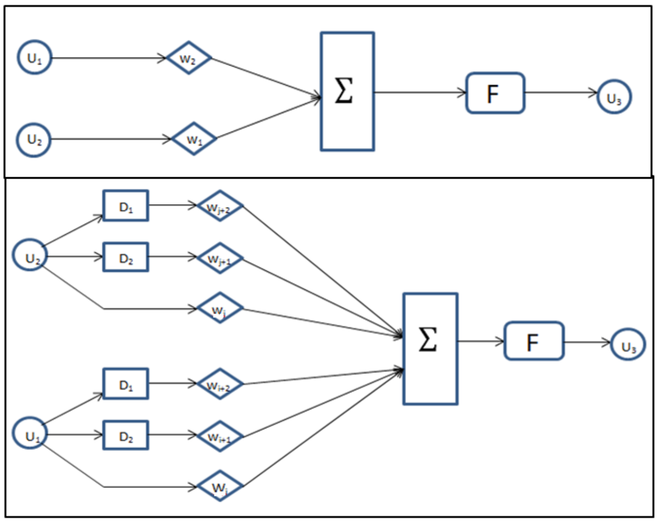

FIGURE 4.1: The top figure are the standard units and the bottom the units for a TDNN. U_i are nodes, w_i are weights, and F is the activation function.

Where $\bar{v}$ and $\bar{x}$ are the average spikes at a stable firing state and κ is a normalization factor. The use of this rule makes neurons fire synchronously which is useful for detecting image features.

4.3.3 Time Delay Neural Network

A network that was specially designed for recognizing and predicting temporal patterns is the Time-Delay Neural Network (TDNN) [5]. The TDNN introduces the concept of short term and long term memory in the network. We still try to minimize the SSE by back propagation which represents the long term memory but we introduce delays in the network that are equivalent to short term memory. Figure 4.1 shows the difference in basic units in a MLP and a TDNN. The number of delays decides the window of the short term memory. If we have for example three delays of one cycle each the network compares the current input with the history of the three other events in the window.

Because of the number of weights and the large number of iterations for training, the back propagation is computationally expensive. But it is possible to start the learning process offline and put the weights in the online model which than can continue to learn online.

4.3.4 Bayesian Network

In order to understand the Dynamic Bayesian Network (DBN) we first have to describe a Bayesian Network (BN) [26]. BN is a graphical model of conditional independences

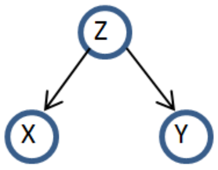

FIGURE 4.2: A simple Bayesian network. X and Y are conditionally independent given Z. The joint probability distribution is: $P(XYZ) = P(X|Z)P(Y|Z)P(Z)$

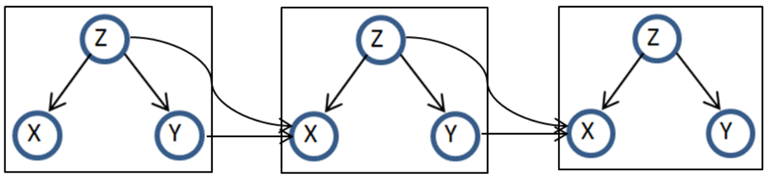

FIGURE 4.3: A DBN in three time steps. The edges between the states are the temporal components.

among variables. The model is represented as a Directed Acyclic Graph (DAG). In the graph the nodes represent variables and the edges the dependencies. An example is given in Figure 4.2.

Dependencies are based on probability theory and the Bayesian rule:

$$P(X, Y) = P(X|Y)P(Y) \tag{4.8}$$

Where $P(X, Y)$ is the joint probability of $X \wedge Y$.

Learning in a BN is done by a process called inference. It is a propagation of beliefs about some of the variables in the network and computing the probability of each state of a node. In other words we infer the probability distribution over some variables in the network given observations of other variables in the network. This strategy is NP-hard so in practise it is done by approximation algorithms.

In order to use a BN for time series analysis, we allow the BN to change over time. The states of the BN are often called time slices. Time slices are connected by temporal relationships that are represented by edges from variables in one slice to variables in the next. The edges in the BN itself are still representations of dependencies. Figure 4.3 shows this more clearly.

The DBN satisfies the Markovian condition, that is the present is only dependent on the direct past, i.e. the state on $t - 1$. In total the DBN now consists of probability distribution of hidden-state variables and observable variables. From this distribution and the observable variables we can predict the variables in the future state.

4.3.5 Support Vector Regression

A more recent prediction method makes use of the Support Vector Machine (SVM) [27] and is called Support Vector Regression (SVR) [28]. This technique is mostly used for classification problems but is successfully adjusted to handle regression. In order to train the SVM we solve the Quadratic Optimization Problem (QOP):

$$\min_{w,b} \tfrac{1}{2} w^T \cdot w + C \textstyle\sum_{t=0}^{N} (\xi_t + \xi_t^*)$$

$$s.t. \begin{cases} y_t - (w^T \phi(t) + b) \leq \varepsilon + \xi_t \\ (w^T \phi(t) + b) - y_t \leq \varepsilon + \xi_t^* \\ \xi_t, \xi_t^* \geq 0, \quad t = 0, .., N \end{cases} \tag{4.9}$$

Where w is the margin (moreover below), y_t the value on time step t, $\phi()$ a kernel function, ε the error tolerance, ξ_t, ξ_t^* and C are boundaries on the error tolerance, and b is the intercept of w.

In the classification case we try to find a separation line between two groups of samples, when projected onto a hyper plane. The samples that are closest to the line are inherently harder to classify. In order to minimize the error we want a rectangular instead of a line and we search for a rectangular with biggest possible width. This rectangular is called the margin w.

In order to make non-linear classification possible we make use of a kernel function $\phi()$ which is commonly chosen as the Gaussian kernel.

The extension to the SVR is made by introducing the ε. Basically we try to find continuous function with the maximal number of samples on it. The ε widens the function into a tube. So we search for a function where most samples lie in this tube. Samples that fall in this tube are not seen as errors, while all others are. As in the classification case we try to minimize the error.

Finally ξ_t, ξ_t^* and C are the boundaries on ε. They give us theoretical guarantees about the prediction performances.

Chapter 5

Fourier prediction with trend compensation

5.1 Fourier prediction

The following section is about the main prediction method we explore in this project. It is related to [7]. The basic idea is that we decompose a complicated signal in to two or more simple signals. As shown in Chapter 3 we can do this by performing a Fourier transformation. The theory of Fourier shows that every complex signal can be described by an infinite number of sinoid signals. Since for our practical solution we can only use a finite number of signals we make use of the DFT which is an approximation of the complex signal. As explained in Chapter 3.2.1 we can filter out every one of those components. The components that we get from the Fourier transform are represented by complex coefficients. If we order those complex coefficients on the value of the real part (from high to low), we obtain the frequencies in leading order. This is the order were the simpler components with the highest amplitudes are on top. Signals with higher amplitudes contribute more to the original signal and are therefore more important from an prediction perspective.

If we got all the components we can extrapolate them. Since all components are sinusoids we need three things from every component:

- The amplitude, the height of the waves
- The phase, the horizontal shift
- The period (or frequency), the number of waves in one time unit

All of the above can be calculated directly from the coefficients themselves. If we have the formula of the DFT:

$$Y_k = \sum_{t=0}^{N-1} x_t \cdot e^{-i2\pi k/N} \tag{5.1}$$

We can calculate the amplitude with:

$$A_k = \frac{\sqrt{Re(Y_k)^2 + Img(Y_k)^2}}{N} \tag{5.2}$$

Where $Re()$ is the real part of the complex component and $Img()$ the imaginary part.

The phase can be calculated with:

$$P_k = \text{atan2}(Img(Y_k), Re(Y_k)) \tag{5.3}$$

The atan2(y, x) is the angle (in radians) between the positive x-axis and the point (x, y) on a two dimensional plane, which is positive for counter clockwise and negative for clockwise.

And the frequency with:

$$f_k = k/N \tag{5.4}$$

We can then fill them in the following formula and extrapolate as long as we want:

$$L_k(t) = \bar{x} + A_k(\sin(f_k t) - P_k) \tag{5.5}$$

Where t is the time step and $\bar{x}$ is the mean of the input data that takes care of the needed vertical shift.

Hereafter we sum L:

$$Pre(t) = \sum_{i=0}^{k} L_k(t) \tag{5.6}$$

which is basically the inverse of the Fourier transform to reconstruct the signal and form the prediction. This signal the represents a complex extrapolation of the original signal

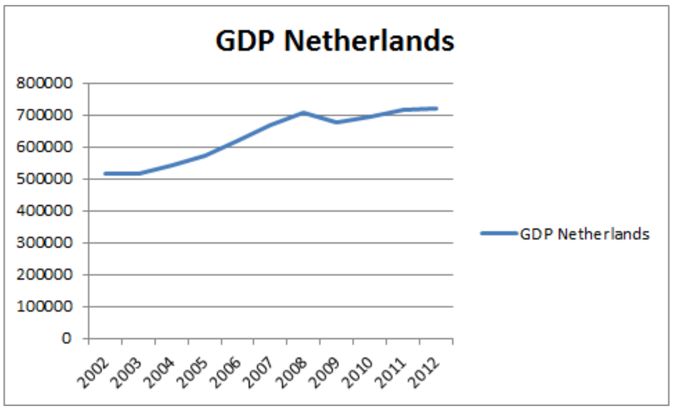

FIGURE 5.1: GDP of the Netherlands, economical time series. Source: OECD, http://stats.oecd.org/

made up from is periodical components. In order to add low pass filtering we can use only a few components for reconstruction which leaves us with a smoother extrapolated prediction signal that could be a better representation of the underlying signal.

5.2 Regressional trend compensation

The method in 5.1 is shown to be very effective at predicting signals with high periodic patterns [7]. Prediction with trends however will be very poorly in theory because this information is not in a discrete windowed frequency representation of the signal. In many time series however there is a trend, see e.g. Figure 5.1 of the Gross Domestic Product (GDP) of the Netherlands. Not correcting for this trend would leave us with a much less generic method (or low accuracy).

This correction is done as follows. First we test if there is an statistical significant trend in the data that we use for prediction. This is done with the so-called Mann-Kendall trend test [29] which is widely used in analysis of hydrologic time series. It is a fairly simple test based on hypothesis testing:

- H_0: There is no trend.
- H_1: There is a trend.

We test our H_0 via the Mann-Kendall statistic:

$$S = \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} sign(T_j - T_i) \tag{5.7}$$

Where T_i are time series values and:

$$sign(x) = \begin{cases} 1 & if \quad x > 0 \\ 0 & if \quad x = 0 \\ -1 & if \quad x < 0 \end{cases} \tag{5.8}$$

We assume that the S statistic is normally distributed with mean:

$$\bar{S} = 0 \tag{5.9}$$

and variance, as derived by [30]:

$$\sigma^2 = \frac{n(n-1)(2n-5)}{18} \tag{5.10}$$

With this we can calculate the $Z(S)$ score as:

$$Z(S) = \begin{cases} \frac{S-1}{\sigma} & if \quad S > 0 \\ 0 & if \quad S = 0 \\ \frac{S+1}{\sigma} & if \quad S < 0 \end{cases} \tag{5.11}$$

If $|Z(S)|$ is greater than $Z_{\alpha/2}$, with $\alpha = 0.05$, there is a 95% probability that H_0 is false. This means that we should adopt H_1. If α is chosen at 5 percent, which is common in statistics, the $Z_{0.025}$ score is 1.96.

When we have a significant trend we compensate for it by:

$$PreCom(t) = Pre(t) + Regr(t) - \bar{Pre} \tag{5.12}$$

Where $Regr(t)$ is a regression line and $\bar{Pre}$ is the average of the unadjusted prediction. This regression line can be any kind of regression. In most economical and natural time series a linear regression will suffice. $PreCom(t)$ is a prediction for t based on the periodical information and a trend.

Chapter 6

Emperical Analysis

In this chapter we describe the experiments we performed in order to analyse the quality of methods introduced in Chapter 4. We test our methods by the prediction scheme proposed in Chapter 1. Furthermore we compare against two other successful methods.

6.1 Scheme details

In order to make a comparison with other methods based on the sensory data of Barcelona metro station we propose a scheme with the following components:

- Data denoising
- Dimension reduction
- Prediction
- Data reconstruction

For our data smoothing is not really needed since we have low-pass filtering in the prediction step. However in order to have more generic scheme we add smoothing as the first step. In order to decide the best smoothing technique we tested on a bell shaped function (see Figure 6.1) with Gaussian noise. With this test we assume that noise is normally distributed. Since we know which part is noise and the shape of the function we can look how much the noise is reduced but the signal is not distorted with a simple SSE. The result is in Figure 6.2. From the techniques of Chapter 3 only the CEMA (section 3.1.2) and the quadratic smoothing (section 3.1.3) are shown because the other techniques performed much worse. We think this is due to lag those methods introduced.

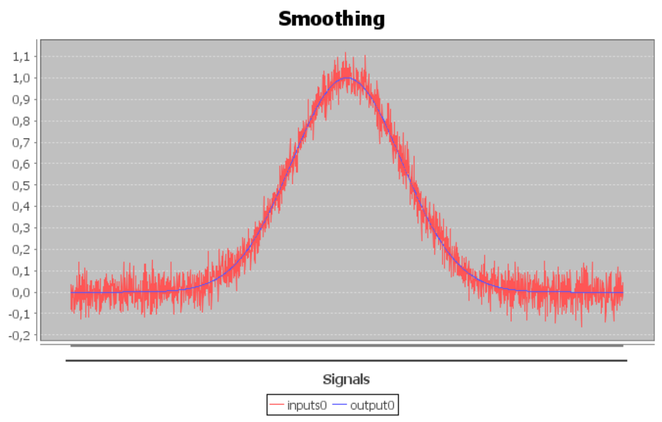

FIGURE 6.1: A Gaussian function with Gaussian noise.

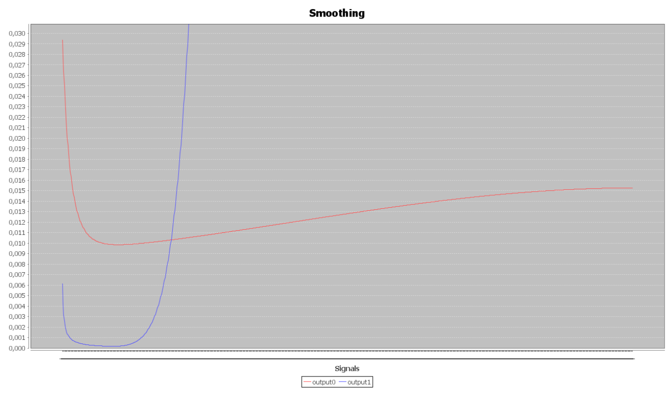

FIGURE 6.2: The results, with on the x-axis the different parameter values for the different methods and on the y-axis the SSE. The blue line is the quadratic smoothing result and the red one shows the result for the CEMA. Furthermore the CEMA is tested for parameter values ranging from 0.05 to 0.99 and the quadratic smoothing for values ranging from 5 to 500 (only the uneven values).

As suspected the quadratic smoothing reduced the noise more while leaving the original bell shape mostly intact with a window of 157. Therefore in the experiments we used this method. We estimate a good window for other input lengths with:

$$R_N = \begin{cases} round(\frac{157*N}{2048}) + 1 & if \quad round(\frac{157*N}{2048}) \; is \; even \\ round(\frac{157*N}{2048}) & if \quad round(\frac{157*N}{2048}) \; is \; uneven \end{cases}, \qquad W = max(5, R_N) \tag{6.1}$$

Where $round()$ is a rounding function to an integer and W the window. The window needs to be 5 at least.

On the dimension reduction we make use of the popular PCA method. It has the advantage of low computing cost and practical usability because we can explicitly choose the variance, i.e. amount of amplitude information, we want to keep. For experiments we choose that at least 90

We test our method against the SVR (section 4.3.5) and a simple MLP (section 4.3.1). These methods are implemented in de popular WEKA environment[1] with the time series package[2] .

We compare with the metrics Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) and Normalized Root Mean Squared Error (NRMSE):

$$RMSE = \sqrt{\frac{\sum_{t=0}^{N}(x_t - \hat{x}_t)^2}{N}} \tag{6.2}$$

$$MAPE = \frac{100\%}{N}\sum_{t=0}^{N}\left|\frac{x_t - \hat{x}_t}{x_t}\right| \tag{6.3}$$

$$NRMSE = \frac{RMSE}{o_{max} - o_{min}} \tag{6.4}$$

Where x_t is the actual value $\hat{x}_t$ is the estimated (predicted) value and o_{max} and o_{min} are the highest observed and lowest observed values respectively. We choose multiple error metrics in order to make a fairer and clearer comparison between methods on different data sets.

6.2 Results

The first results are from testing the method from [7] with our adjusted version on three data sets. One with temperatures from the metro station with very little trend, one with just two linear functions that are trend only and one with production values from [32], see Figures 6.3, 6.4 and 6.5. For the temperatures we try to predict 245 steps. This is an arbitrary choice, except that we want to make sure that we can get a good comparison on multi-step predictions for which the number of steps need to be large enough. For the linear functions we predict 452 steps, because more steps do not make a difference here on the chosen error metrics. Results are in Table 6.1.

[1]http://www.cs.waikato.ac.nz/ml/weka

[2]http://wiki.pentaho.com/display/DATAMINING/Time+Series+Analysis+and+Forecasting+with+Weka

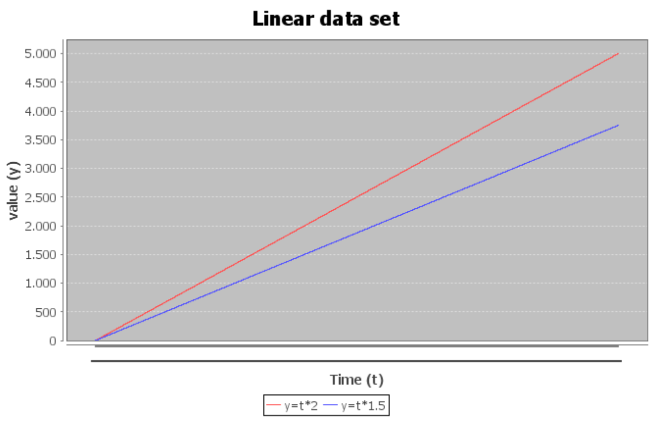

FIGURE 6.3: Two linear functions.

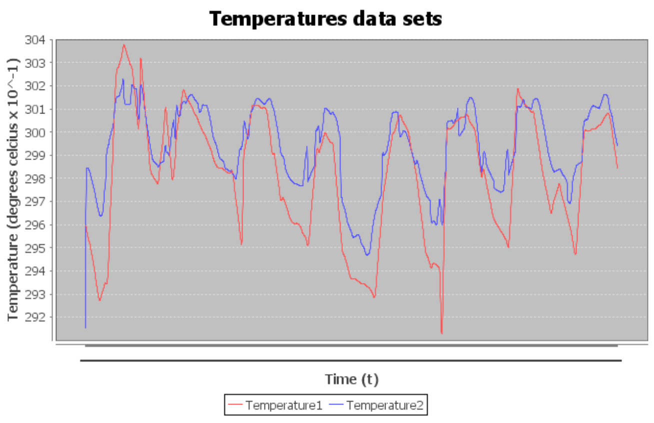

FIGURE 6.4: Two temperatures from the metro station.

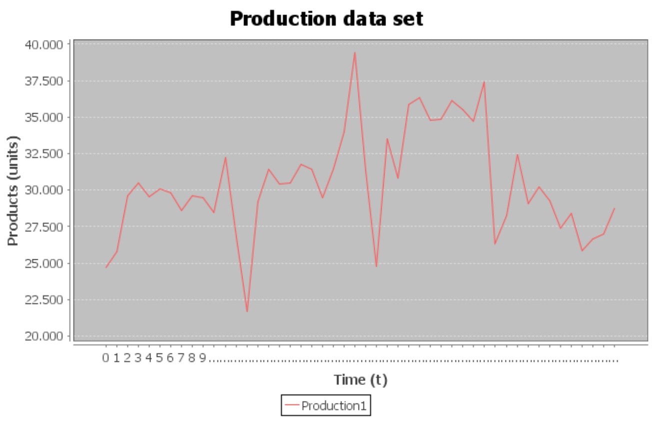

FIGURE 6.5: Production values see [32].

Table 6.1: Results from the Fourier prediction, the proposed method, SVR and a MLP on three error metrics. We use 5 data sets as described in the text. The bold values are the values of the methods that are the best on the given data set on the given error metric.

		RMSE:	**MAPE (%):**	**NRMSE:**
FP without compensation	Production1	**3158,098**	**8,985**	**0,178**
	Linear1	2514,676	54,853	0,503
	Linear2	1886,000	54,853	0,503
	Temperature1	**0,994**	0,290	**0,080**
	Temperature2	**0,508**	**0,140**	**0,047**
	sunspots1	**34,22**	infinite	**0,13**
Proposed method	Production1	**3158,098**	**8,985**	**0,178**
	Linear1	5,538	0,100	0,001
	Linear2	4,154	0,100	0,001
	Temperature1	1,000	**0,275**	**0,080**
	Temperature2	0,602	0,167	0,056
	sunspots1	**34,22**	infinite	**0,13**
SVR	Production1	12209,859	24,052	0,419
	Linear1	**4,003**	**0,073**	**0,004**
	Linear2	**3,002**	**0,073**	**0,004**
	Temperature1	1,622	0,488	0,739
	Temperature2	2,142	0,603	0,454
	sunspots1	57,58	infinite	0,23
MLP	Production1	11742,078	25,492	1,144
	Linear1	15,036	0,242	0,017
	Linear2	11,277	0,242	0,017
	Temperature1	3,680	1,038	0,268
	Temperature2	3,493	1,005	0,740
	sunspots1	508,63	infinite	2,00

Chapter 7

Discussion

7.1 Discussion of quality

Table 6.1 shows that the Fourier prediction and proposed method work better than the comparing methods on data with periodical patterns. The prediction accuracy of the proposed method even comes close on the state of the art method from [32] on the Production1 data set. This is not surprising since the proposed method focusses on finding periodical patterns that are very usable for multi-step prediction. So for the temperatures data it works relatively well and. The minor difference in the Fourier prediction method and the proposed method is possible due to a overcompensation for the trend. The Mann-Kendall test only test for a monotonic trend, which is not necessarily a linear one. We use however linear regression which leaves us with a very small over compensation. This error is only clear from the RMSE metric which does not take the total variation of the prediction into account.

On the linear data sets, which have no periodical patterns and are trend only, the Fourier prediction for the trend predicts relatively bad. This we also expected since there is only a trend. The proposed however predicts quite well. The RMSEs of 5,538 and 4,154 are due to the number of frequencies we keep. If we add more frequencies the RMSE of the proposed method with compensation will approach 0. The sinusoids have to cancel each other out, that is need to have the same frequency and amplitude but are in anti-phase. With only 8 available frequencies this is not yet happened and thus leaves us with an sinusoidal pattern around the regression line and therefore an error. However it is clear from Table 6.1 that the compensation increases the accuracy which makes the proposed method more generic than the Fourier prediction.

The most surprising result is the MLP multi-step prediction on the linear data is worse than the proposed method, i.e. the accumulation of errors over more time steps is greater

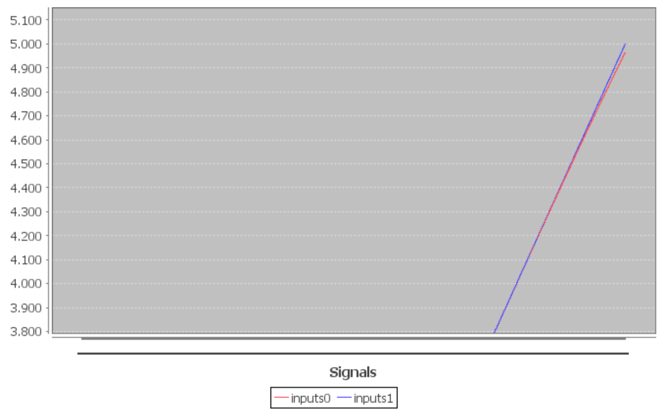

FIGURE 7.1: Two linear functions.

than we expected. The output of the prediction is shown in Figure 7.1. The slope of the predicted curve is a little off which leads to a larger RMSE at later time steps.

The RMSE of the SVR is due to the width of the tube for which error are still considered as good prediction. Although it is very small it can be even smaller if we use extra an assumption, i.e. that the data is linear. The prediction on the other datasets however will be worse probably, because this loss of generality.

7.2 Discussion of practical value

All methods from Table 6.1 are not well suited for online prediction. Therefore it is imperative that we use batch methods in practise. However for the MLP the trainings complexity is NP-hard as explained by [33] which makes it not very practical to use. The SVR training has a complexity of $O(N^4)$ which is the N for the number of data points times $O(N^3)$ for the quadratic optimization problem [34]. For very large data sets or batch sizes this can become a problem. Finally the complexity for the proposed method is $O(NlogN)$ due to the FFT. The naïve implementation will yield a algorithm with complexity $O(N^2)$. So if we take the batch size a power of 2, this will be more practical in use than the MLP or the SVR for online prediction.

The other parts of the prediction scheme have also low computational costs, the smoothing part is only $O(N)$ (see Section 3.1.3) and the PCA can be done in $O(d^2h+d^2N)$ [35], where d is the number of raw dimensions and h the number of reduced dimensions. Since N is large compared to h, for the metro station data, the complexity can be estimated with $O(d^2N)$.

Chapter 8

Conclusions and future work

8.1 Conclusions

In this thesis project we proposed a prediction method that is well suited for periodical data that also works with trends in the data. We compared the method against the popular prediction methods the MLP and SVR. For the metro data the proposed method outperformed the other two on accuracy. Furthermore the proposed method has relatively low complexity which makes it practical for projects like the SEAM4US. To make this method applicable for the metro station data we proposed a prediction scheme which includes data smoothing and dimension reduction. We used quadratic smoothing and PCA but the scheme is modular enough to incorporate all types of smoothing and dimension reduction techniques.

8.2 Future work

Future research can be extrapolation with wavelets. Although we expect a small increase in accuracy on single-step prediction because of the better representation, the multi-step prediction will probably not benefit from the localized patterns. In the Fourier prediction we only use a combination of waves.

An extension to the proposed technique could be an technique for other trend detections than linear trends. Linear trends (or near linear trends) are most common in natural and economical time series but the method can be more generic if we also can detect quadratic or logarithmic trends.

And an extension on the Fourier extrapolation is Chirp extrapolation.

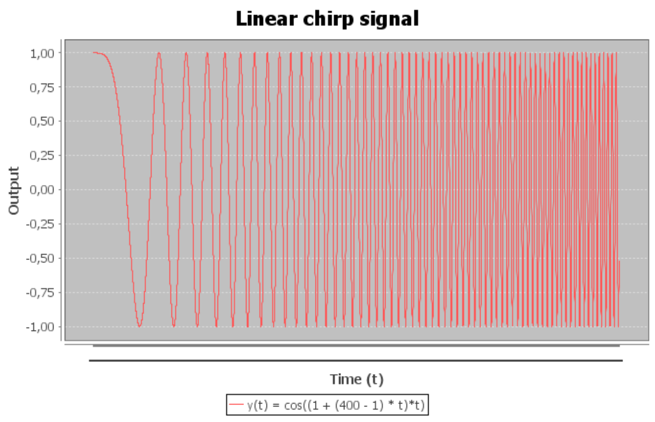

FIGURE 8.1: A chirping data set.

8.3 Chirp extrapolation

Instead of using the Fourier transform we can take a more general form as extension, namely the Chirp-Z transform. Prediction with this transform allows to take chirping signals into account. An example of a chirping signal is in 8.1. In other words a signal that has an increasing or decreasing frequency over time. This transform is a little more complex to compute and unnecessary for most natural and economical time series, but it makes the complete method more general in use.

The formal definition of a chirp is given by:

$$y(t) = \sin(\Phi_0 + 2\pi(f_0 t + \frac{k}{2}t^2)) \tag{8.1}$$

Where k is the chirp rate, i.e. the rate at which the frequency increases or decreases.

Thus instead of transforming a complex signal in simple sinusoids we transform it into chirps. The transformation is done with the algorithm of [31]. And an inverse transform based on the same algorithm. This algorithm in turn is based on:

$$Y_k = \sum_{t=0}^{N-1} x_t A^{-t} W^{tk} \tag{8.2}$$

Where A and W are of the form:

$$A = A_0 e^{i2\pi\theta_0} \tag{8.3}$$

and

$$W = W_0 e^{i2\pi\phi_0} \tag{8.4}$$

The algorithm can be described in the following steps:

1. Create for every time value (or frequency value since in our use their the same) a complex number w_k with:

$$w_k = \cos k^2 \frac{\phi_0}{2} + \sin k^2 \frac{\phi_0}{2} \tag{8.5}$$

   where $\phi_0 = \frac{-2\pi}{nz}$ and z is related to the chirp rate. Multiply these values with the corresponding values of the input array to obtain an array y_n. If the values of the input array are real valued then first make a complex number with the imaginary part set to zero.

2. Add as many zeros to y_n as the length of the input.
3. Take the Fourier transformation of y_n, i.e. $FT(y_n)$.
4. Create an array v_n of w_k with:

$$w_k = \begin{cases} \cos -k^2 \frac{\phi_0}{2} + \sin -k^2 \frac{\phi_0}{2} & if \quad 0 \le k \le n-1 \\ \cos -(2n-k)\frac{(2n-k)\phi_0}{2} + \sin -(2n-k)\frac{(2n-k)\phi_0}{2} & if \quad n \le k \le 2n-1 \end{cases} \tag{8.6}$$

5. Take the Fourier transformation of v_n, i.e. $FT(v_n)$.
6. Multiply $FT(y_n)$ by $FT(v_n)$ point by point and save the output in g_n.
7. Take the inverse Fourier transform of g_n, i.e. $IFT(g_n)$.
8. The output is given by multiplying $IFT(g_n)$ by w_k the same as in step 1.

And the in inverse is basically the same algorithm backwards:

1. Create for every time value (or frequency value since in our use their the same) a complex number w_k with:

$$w_k = \cos k^2 \frac{\phi_0}{2} + \sin k^2 \frac{\phi_0}{2} \tag{8.7}$$

   where $\phi_0 = \frac{-2\pi}{nz}$ and z is related to the chirp rate. Divide the input values with the corresponding w_k to obtain g_k.

2. Take the Fourier transformation of g_n, i.e. $FT(g_n)$.
3. Create an array v_n of w_k with:

$$w_k = \begin{cases} \cos -k^2 \frac{\phi_0}{2} + \sin -k^2 \frac{\phi_0}{2} & if \quad 0 \leq k \leq n-1 \\ \cos -(2n-k) \frac{(2n-k)\phi_0}{2} + \sin -(2n-k) \frac{(2n-k)\phi_0}{2} & if \quad n \leq k \leq 2n-1 \end{cases} \tag{8.8}$$

4. Take the Fourier transformation of v_n, i.e. $FT(v_n)$.
5. Divide $FT(g_n)$ by $FT(g_n)$ point by point and save the output in y_n.
6. Take the inverse Fourier transform of y_n, i.e. $IFT(y_n)$.
7. The output is given by dividing $IFT(y_n)$ by w_k the same as in step 1.

For certain data sets we expect it to enhance prediction accuracy. For $A = 1$ and $W = e^{-i2\pi/N}$ this transformation is the same as the DFT, which is why we think we will never be worse than the method described in 5.1.

Appendix A

Smoothing examples

Figures A.1 A.2 A.3 are the smoothing examples for the Moving average. The data comes from temperature measurements of a part of the Barcalona metro station and contains 2293 data points. These examples show clearly that a larger filter width means a smoother signal. The horizontal parts at the beginning and the end of the smoothed signal will be the lag in a prediction system. All graphs below are made with help of JFreeChart[1].

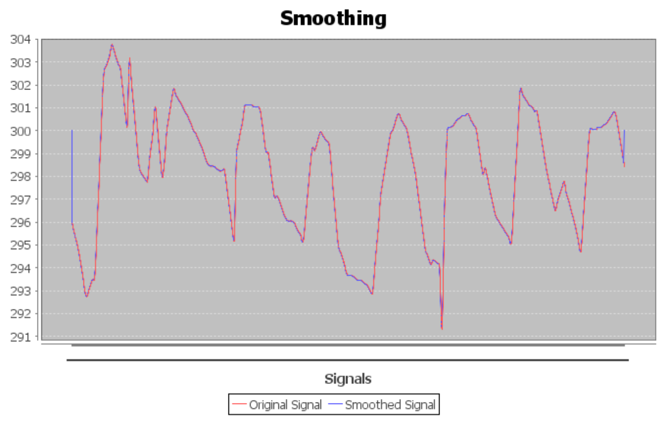

FIGURE A.1: Moving average with filter width 5. The smoothed signal will probably still contain a lot of noise.

[1]http://www.jfree.org/jfreechart/t

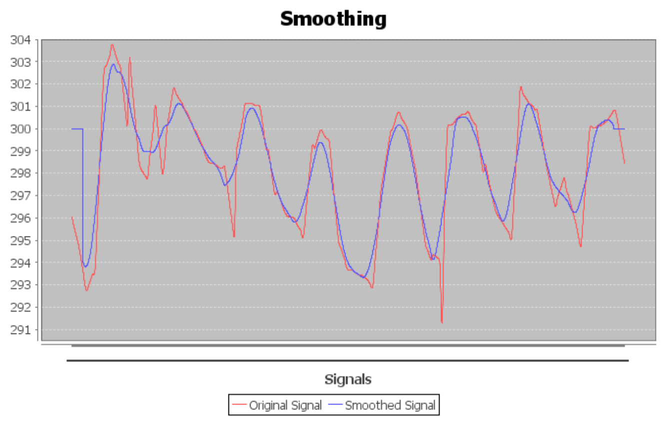

FIGURE A.2: Moving average with filter width 91. The smoothed signal filters out more fluctuations in the original signal.

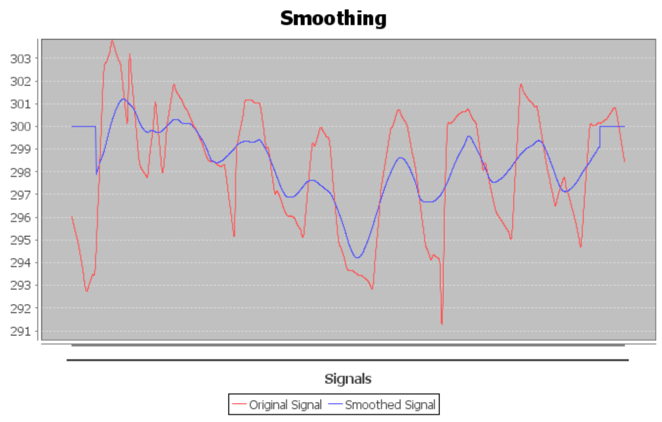

FIGURE A.3: Moving average with filter width 201. The smoothed signal filters out a lot of fluctuations in the original signal and the original signal will probably distorted a lot.

Figures A.4 A.5 A.6 are the smoothing examples for the Exponential smoothing. The examples show that the more smoothing means more lag.

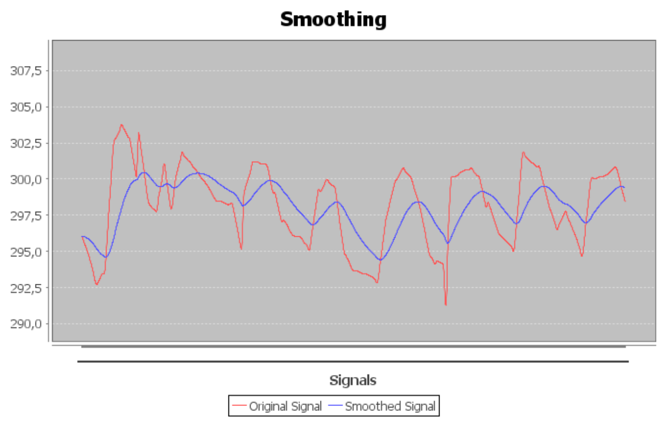

FIGURE A.4: Exponential smoothing with smoothing factor 0.01. Probably filters out a lot of noise but distorts the signal and introduces a great deal of lag.

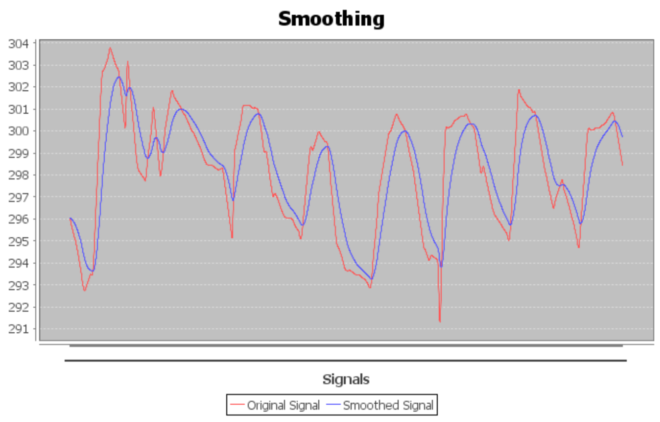

FIGURE A.5: Exponential smoothing with smoothing factor 0.03. Looks to be a good smoothing factor but the signal has lag.

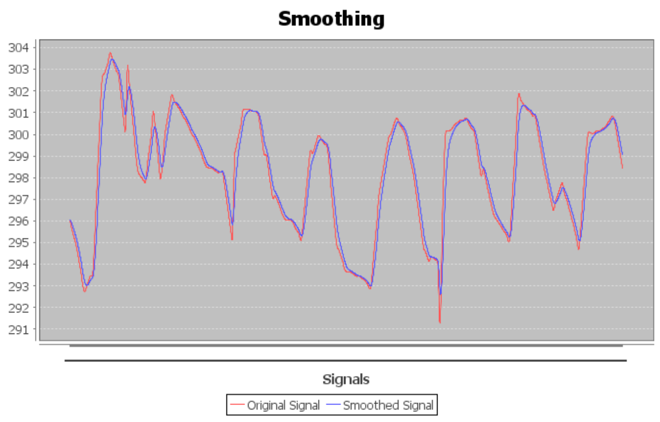

FIGURE A.6: Exponential smoothing with smoothing factor 0.09. Probably reduces the noise not that much but introduces less lag.

Figures A.7 A.8 A.9 are the smoothing examples for the CEMA. It works as well as Exponential smoothing only the lag is gone.

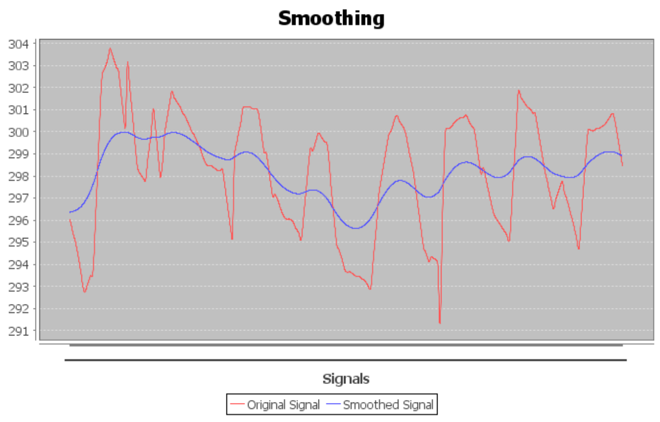

FIGURE A.7: CEMA with smoothing factor 0.01.

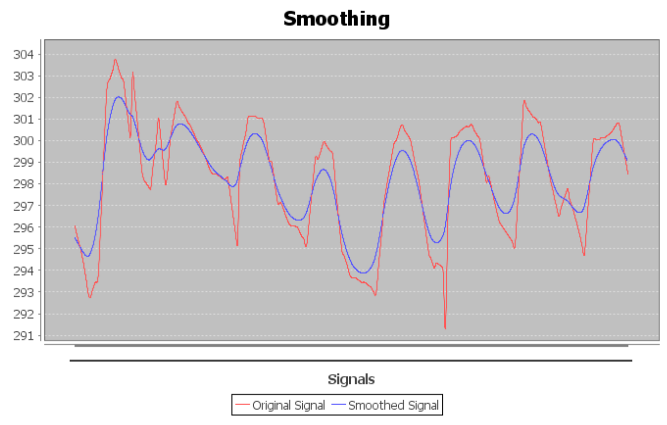

FIGURE A.8: CEMA with smoothing factor 0.03.

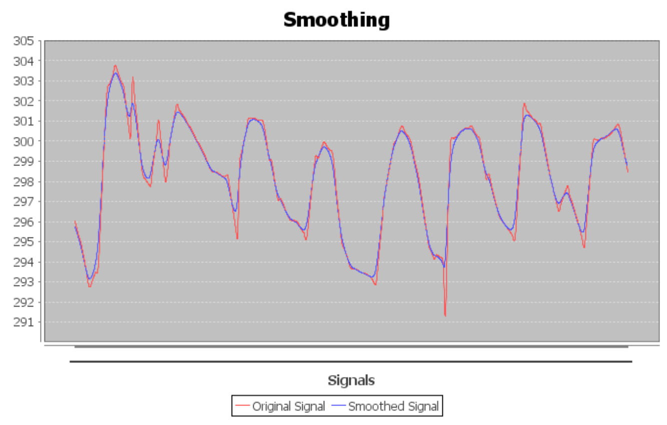

FIGURE A.9: CEMA with smoothing factor 0.09.

Figures A.10 A.11 A.12 are the smoothing examples for the SG-smoothing. No lag and probaly less distorted than with the Moving average.

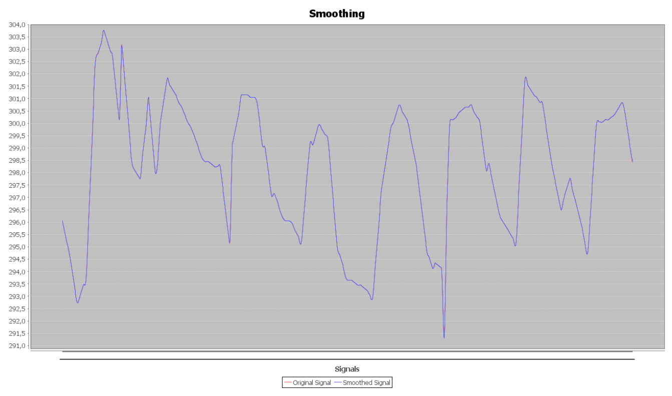

FIGURE A.10: SG with filter width 5. Almost a perfect fit so probably too less smoothing.

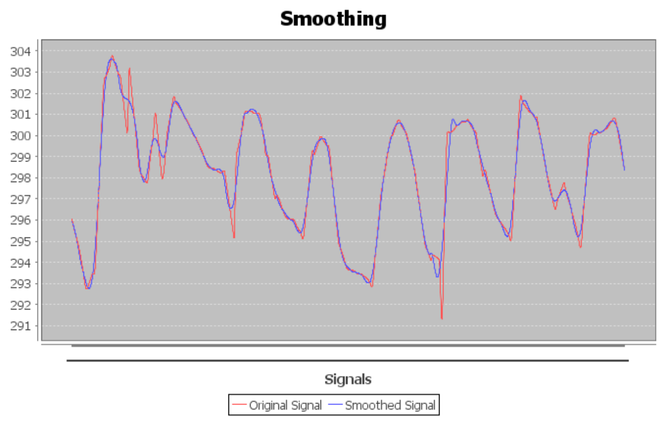

FIGURE A.11: SG with filter width 91.

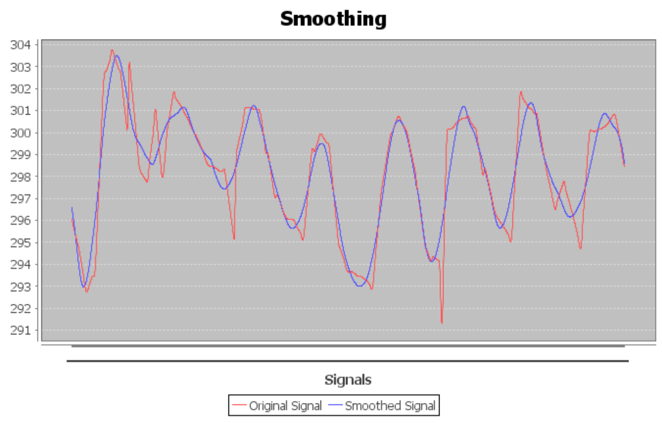

FIGURE A.12: SG with filter width 201.

Figures A.13 A.14 A.15 are the smoothing examples for the TVR. It ”cuts off” variations that are too rapid. An higher λ means more smoothing.

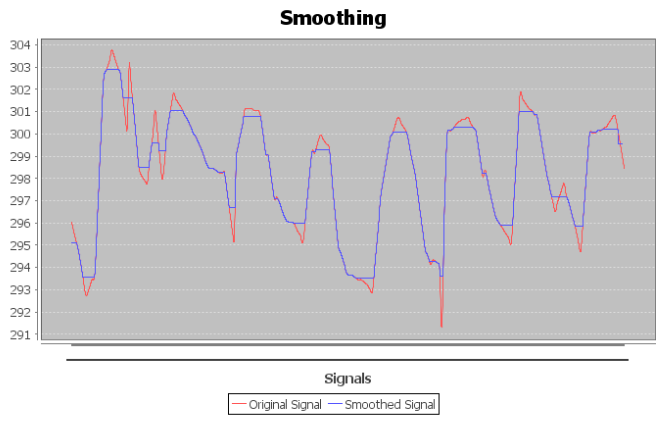

FIGURE A.13: TVR with $\lambda = 10$.

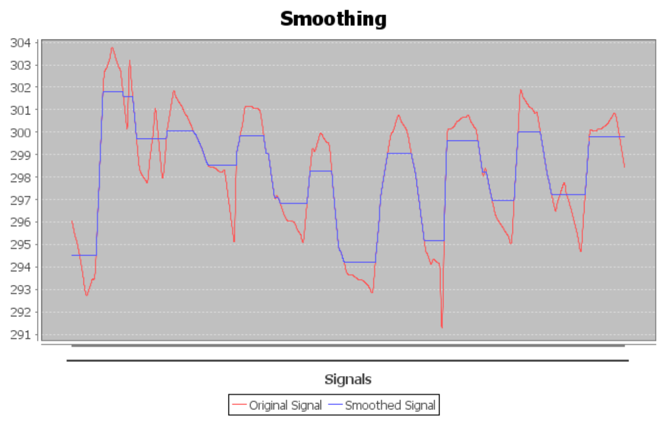

FIGURE A.14: TVR with $\lambda = 50$.

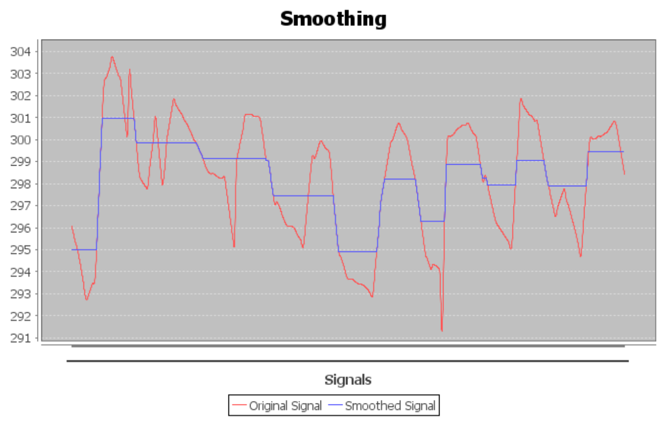

FIGURE A.15: TVR with $\lambda = 100$.

Appendix B

Filtering examples

This are the examples of smoothing with filters. We use the same example data as in Appendix A.

Figures B.1 B.2 B.3 are the filtering examples for the Fourier filter. As more coefficients are set to zero, i.e. the higher the threshold values, there is more smoothing.

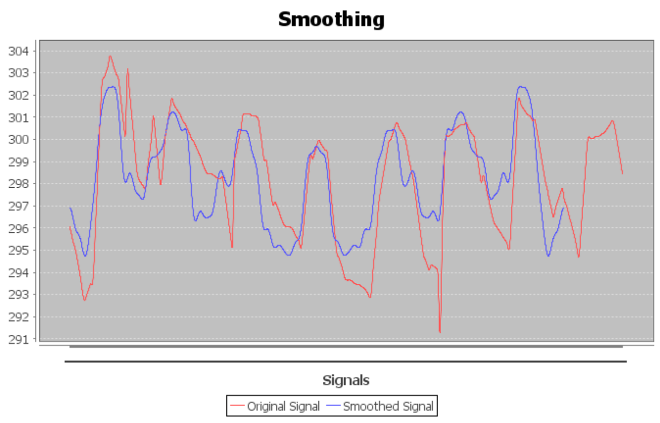

FIGURE B.1: Fourier filter with threshold 100 and -100.

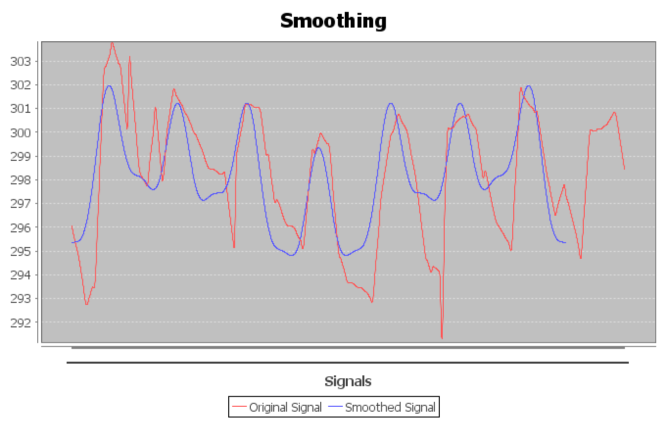

FIGURE B.2: Fourier filter with threshold 500 and -500.

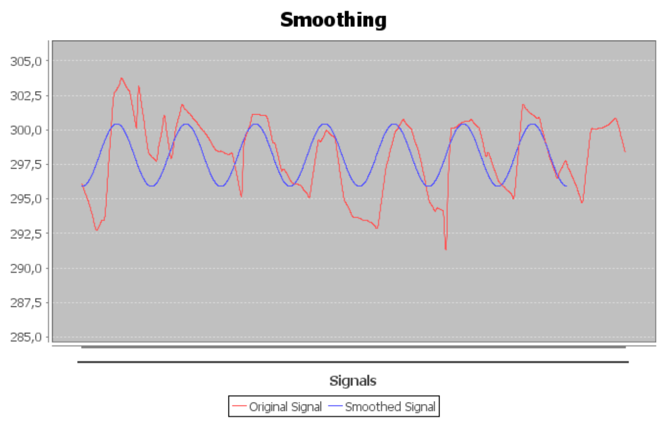

FIGURE B.3: Fourier filter with threshold 1000 and -1000. Here every coefficient except one is filtered and we are left with a single sinusiodal wave.

Figures B.4 B.5 B.6 are the filtering examples for the Haar Wavelet filter. As more coefficients are set to zero, i.e. the higher the threshold values, there is more smoothing.

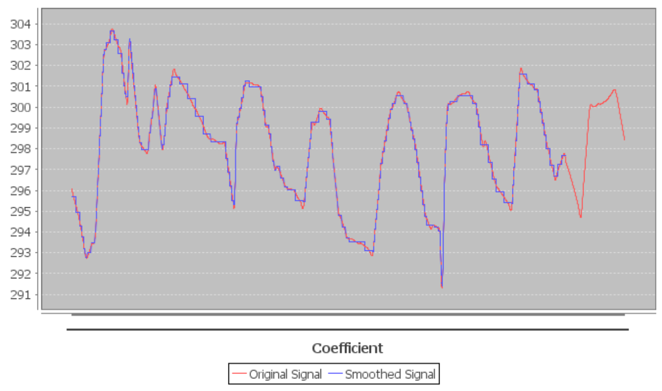

FIGURE B.4: Wavelet filter with threshold 0.2 and -0.2.

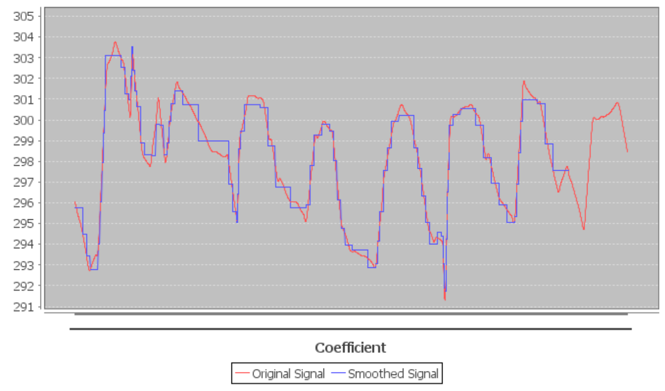

FIGURE B.5: Wavelet filter with threshold 0.5 and -0.5.

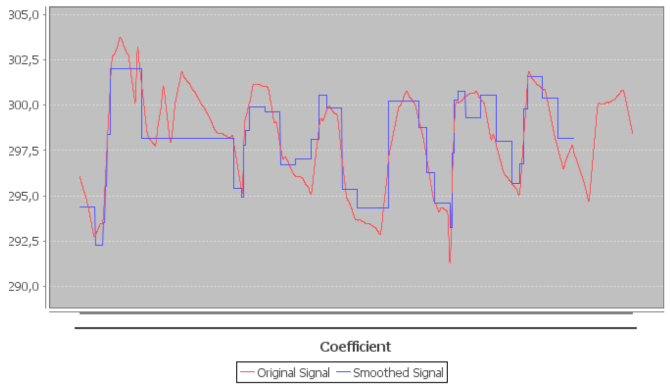

FIGURE B.6: Wavelet filter with threshold 1.0 and -1.0.

Figure B.7 is an example of the Kalman Filter. The result is dependent on the initial estimates.

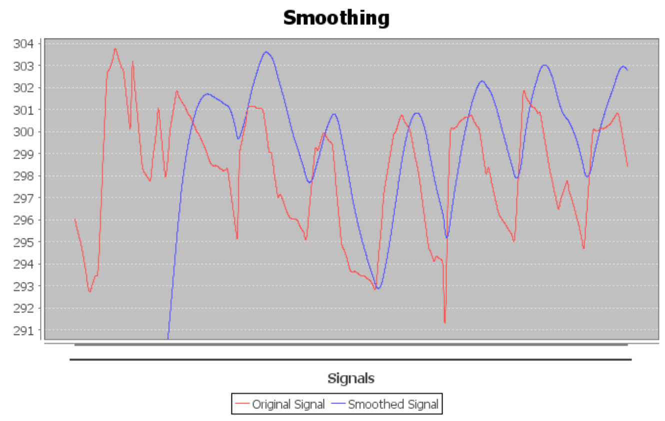

FIGURE B.7: Kalman Filter.

Figure B.8 and B.9 is an example of the Particle filter. The result is dependent on the sample size, i.e. number of particles. More precision takes more run time.

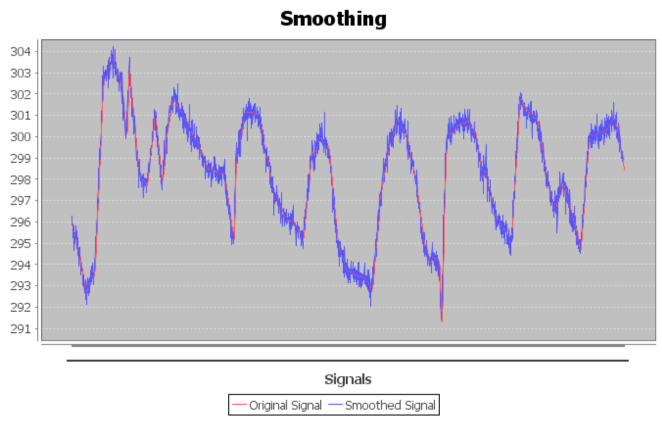

FIGURE B.8: Particle filter with 30 particles. This example took 2.07 seconds to make on a modern laptop.

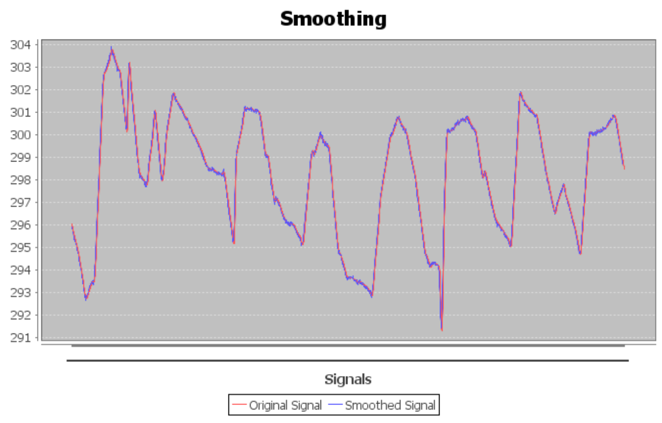

FIGURE B.9: Particle filter with 400 particles. This example took 251.56 seconds to make on a modern laptop.

Bibliography

[1] Yaacov Schul, Ruth Mayo, Eugene Burnstein, and Naomi Yahalom. How people cope with uncertainty due to chance or deception. *Journal of Experimental Social Psychology*, In Press, Corrected Proof:91–103, 2006. doi: 10.1016/j.jesp.2006.02.015. URL http://dx.doi.org/10.1016/j.jesp.2006.02.015.

[2] Guy Madison, Lea Forsman, rjan Blom, Anke Karabanov, and Fredrik Ulln. Correlations between intelligence and components of serial timing variability. *Intelligence*, 37(1):68 – 75, 2009. ISSN 0160-2896. doi: 10.1016/j.intell.2008.07.006. URL http://www.sciencedirect.com/science/article/pii/S0160289608000937.

[3] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall International Editions Series. Prentice Hall, 1999. ISBN 9780139083853. URL http://books.google.nl/books?id=M5abQgAACAAJ.

[4] W. Maass and C.M. Bishop. *Pulsed Neural Networks*. Bradford Books. Mit Press, 2001. ISBN 9780262632218. URL http://books.google.nl/books?id=jEug7sJXP2MC.

[5] Alexander Waibel, Toshiyuki Hanazawa, Geofrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Readings in speech recognition. chapter Phoneme recognition using time-delay neural networks, pages 393–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-124-4. URL http://dl.acm.org/citation.cfm?id=108235.108263.

[6] T.C. Mills. *Time Series Techniques for Economists*. Cambridge University Press, 1991. ISBN 9780521405744. URL http://books.google.nl/books?id=cNe3xrFg3PcC.

[7] Hans Butler. Feedforward signal prediction for accurate motion systems using digital filters. *Mechatronics*, June 2012. ISSN 09574158. doi: 10.1016/j.mechatronics.2012.05.002. URL http://dx.doi.org/10.1016/j.mechatronics.2012.05.002.

[8] Carlos E. Garca, David M. Prett, and Manfred Morari. Model predictive control: Theory and practicea survey. *Automatica*, 25(3):335 – 348, 1989. ISSN 0005-1098. doi: 10.1016/0005-1098(89)90002-2. URL http://www.sciencedirect.com/science/article/pii/0005109889900022.

[9] R. G. Brown. Exponential smoothing for predicting demand. *Proceedings of the Tenth National Meeting of the Operations and Research Society of America in San Francisco*, November 1956. URL http://legacy.library.ucsf.edu/tid/dae94e00/pdf;jsessionid=31DC9059F6100FD3E3184C130D2EB85B.tobacco03.

[10] D. H. Chen, P. Lo, and W. P. Swan. Zero-lag exponential moving average for real-time control and noisy data processing. *Hydrocarbon Processing*, 86(10):61 – 70, 2007. ISSN 00188190. URL http://search.ebscohost.com/login.aspx?direct=true&db=afh&AN=27159433&site=ehost-live.

[11] P. A. Gorry. General least-squares smoothing and differentiation by the convolution (savitzky-golay) method. *Analytical Chemistry*, 62(6):570–573, 1990. doi: 10.1021/ac00205a007. URL http://pubs.acs.org/doi/abs/10.1021/ac00205a007.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C The Art of Scientific Computing*, pages 650–655. Cambridge University Press, second edition, 1992.

[13] Åke B. *Numerical methods for least squares problems.* SIAM, 1996. ISBN 0898713609. URL http://www.worldcat.org/isbn/0898713609.

[14] L. Condat. A direct algorithm for 1d total variation denoising. 2012. URL http://hal.archives-ouvertes.fr/docs/00/68/16/48/PDF/condat_fast_tv.pdf.

[15] Leonid I. Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(14):259 – 268, 1992. ISSN 0167-2789. doi: 10.1016/0167-2789(92)90242-F. URL http://www.sciencedirect.com/science/article/pii/016727899290242F.

[16] W. L. Briggs and H. Van Emden. *The DFT: an owner's manual for the discrete Fourier transform.* Society for Industrial and Applied Mathematics, Philadelphia, 1995. ISBN 0-89871-342-0. URL http://opac.inria.fr/record=b1093820.

[17] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. ISSN 00255718. doi: 10.2307/2003354. URL http://dx.doi.org/10.2307/2003354.

[18] C. K. Chui. *An introduction to wavelets.* Academic Press Professional, Inc., San Diego, CA, USA, 1992. ISBN 0-12-174584-8. URL http://dl.acm.org/citation.cfm?id=163196.

[19] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45, 1960. URL http://www.cs.unc.edu/~{}welch/kalman/media/pdf/Kalman1960.pdf.

[20] Arnaud, Doucet, and Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. Technical report, 2008. URL http://www.cs.ubc.ca/~{}arnaud/doucet_johansen_tutorialPF.pdf.

[21] I. T. Jolliffe. *Principal Component Analysis*. Springer, second edition, October 2002. ISBN 0387954422. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0387954422.

[22] H. von Storch and F.W. Zwiers. *Statistical Analysis in Climate Research*. Cambridge University Press, 2002. ISBN 9780521012300. URL http://books.google.nl/books?id=_VHxE26QvXgC.

[23] T. Masters. *Signal and image processing with neural networks: a C++ sourcebook*. J. Wiley, 1994. ISBN 9780471049630. URL http://books.google.nl/books?id=F8RQAAAAMAAJ.

[24] N.G. Pavlidis, O.K. Tasoulis, V.P. Plagianakos, G. Nikiforidis, and M.N. Vrahatis. Spiking neural network training using evolutionary algorithms. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 4, pages 2190 –2194 vol. 4, 31 2005-aug. 4 2005. doi: 10.1109/IJCNN.2005.1556240.

[25] Yoonsuck Choe, Risto Miikkulainen, and Lawrence K. Cormack. Effects of presynaptic and postsynaptic resource redistribution in hebbian weight adaptation. *Neurocomputing*, 32–33:77–82, 2000. URL http://nn.cs.utexas.edu/?choe:cns99.

[26] K. P. Murphy. Dynamic bayesian networks: representation, inference and learning. *PhD thesis, University of California, Berkeley*, 2002. URL http://www.cs.ubc.ca/~murphyk/Thesis/thesis.html.

[27] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125. doi: 10.1007/BF00994018. URL http://dx.doi.org/10.1007/BF00994018.

[28] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000. ISBN 9780521780193. URL http://books.google.nl/books?id=B-Y88GdO1yYC.

[29] R. Chandler and M. Scott. *Statistical Methods for Trend Detection and Analysis in the Environmental Sciences*. Statistics in Practice. Wiley, 2011. ISBN 9781119991960. URL http://books.google.nl/books?id=zLh3OkKWJ7QC.

[30] P.D. Valz and A.I. McLoad. A simplified derivation of the variance of kendall's tau. *The American Statistician*, pages 39–40, 1990. doi: 10.1016/j.patrec.2007.01.012. URL http://www.stats.uwo.ca/faculty/aim/vita/ps/kendall.pdf.

[31] L. Rabiner, R.W. Schafer, and C.M. Rader. The chirp z-transform algorithm. *Audio and Electroacoustics, IEEE Transactions on*, 17(2):86–92, 1969. ISSN 0018-9278. doi: 10.1109/TAU.1969.1162034.

[32] Hao-Tien Liu and Mao-Len Wei. An improved fuzzy forecasting method for seasonal time series. *Expert Syst. Appl.*, 37(9):6310–6318, September 2010. ISSN 0957-4174. doi: 10.1016/j.eswa.2010.02.090. URL http://dx.doi.org/10.1016/j.eswa.2010.02.090.

[33] J.S. Judd. *Neural network design and the complexity of learning*. Neural Network Modeling and Connectionism Series. Massachusetts Institute Technol, 1990. ISBN 9780262100458. URL http://books.google.nl/books?id=DOGQ_EHRX1kC.

[34] L. Bottou. *Large-scale Kernel Machines*. Neural Information Processing Series. Mit Press, 2007. ISBN 9780262026253. URL http://books.google.nl/books?id=MDup2gE3BwgC.

[35] Alok Sharma and Kuldip K. Paliwal. Fast principal component analysis using fixed-point algorithm. *Pattern Recogn. Lett.*, 28(10):1151–1155, July 2007. ISSN 0167-8655. doi: 10.1016/j.patrec.2007.01.012. URL http://dx.doi.org/10.1016/j.patrec.2007.01.012.