Department of Informatics, University of Zürich

Final Report

Implementation of Approximation Functions for Time Series

Rojomon Nagaroor

Matrikelnummer: 08-707-804

Email: r.nagaroor@gmail.com

October 17, 2011

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

Contents

List of Figures

1. Introduction

This report is written in the context of a summer internship at the University of Zürich. The goal of this internship is the implementation of various approximation functions for time series. These approximation functions are important regarding the prediction of a course of a time series, for which individual values or a set of values are not observed, respectively missing. That is, with help of the approximations functions, one should be able to calculate the missing values. In order to achieve this goal, the main task of implementation is divided into five smaller subtasks, representing milestones whose finishing status can be well measured. These five subtasks consist of:

1. Proposing a relational schema for the hydrological data and creating an instance of it in an Oracle server.
2. Literature research on time series approximation techniques.
3. Implementation of continuous approximation functions (Linear interpolation, Cubic spline interpolation).
4. Implementation of piecewise approximation functions (Piecewise linear approximation, piecewise spline approximation, piecewise aggregate approximation).
5. Empirical precision comparison between the continuous and piecewise functions for approximating the hydrological time series.

The implementation of all the algorithms will be in PL SQL. Chapter 2 gives a basic definition of time series and missing values, since they are very important with respect to the algorithms presented in the following chapters. The result of task 3 and 4 are presented in chapter 3 and 4. Chapter 5 contains a comparison of the continuous and piecewise approximation technique addressing to task 5. Task 1 is handled in chapter 6. Chapter 7 contains the conclusions and a brief judgment of the accomplished tasks. The second task, which consists of literature research flows into each chapter, where it is necessary.

I thank everyone who helped me in achieving the goal of this internship, especially my supervisor Mr. M. Khayati for giving me very valuable inputs and for correcting me when it was necessary. I also thank Prof. Mr. M. Böhlen for giving me this chance to work in his DBTG group at the department of informatics, University of Zürich.

2. Time Series and Missing Values

A time series is a sequence of values, where the interval between the values are defined by time and where they are constant. We can imagine a time series as a set of points {(x, y)} in the Cartesian coordinate system, where the x-value represents the time and the y-value represents the corresponding value measured at that time. The x-values are also referred to as timestamps.

In this project, I worked with a set of time series containing hydrological data. The whole set contained 80 series of observations, where each series had approximately about 36 000 observations. But since not all of these observations could be measured since to various reasons like technical malfunctioning etc., some series had less observations than other series, meaning that some series had values, which were "missing". The aim of this project was to implement various approximating techniques in order to approximate the time series. The approximation should then be used to "guess" the missing values.

So first, I needed to identify the missing values before going on to writing approximating algorithms. For this project I used the following definition of missing value:

A missing value consists of the timestamp and its according value which the series, with which we are currently working, does not have, but is existent in another series.

Since the database of the hydrological data contains more than one series of observations, I used all given series to determine the missing values using the following method:

Take a series of observations T_i and check if all the timestamps of T_i are contained in another series T_j where $i \neq j$. If there exists any timestamps in T_j which are not contained in T_i, then these timestamps represents missing values for T_i. Figure 1 shows a snippet of the source code of the procedure which finds the missing value for a given series with the help of another series.

In this fashion, I was able to detect all missing values for each time series. I stored the missing values afterwards in a table called "MISSING_VALUES".

```
insert into missing_values(series_id, ts)
select series_to_find, ts
from (
  select series_to_find, ts
  from observations
  where series_id = series_lookup and ts not in(
    select ts
    from observations
    where series_id = series_to_find))
where ts not in(
  select ts
  from missing_values
  where series_id = series_to_find);
```

Figure 1: Snippet of the source code containing the procedure which checks for missing values

3. Continuous approximation functions

After defining the missing values, we can start to approximate the existing time series in order to interpolate the missing values of the given time series. In scope of this project, I used two classes of approximation functions, the continuous approximation functions and the piecewise approximation functions, which will be explained in chapter 4.

This chapter is dedicated to continuous approximation functions, that is, the approximation is done taking a global view of the time series, in contrast to the piecewise functions. The first section of this chapter is about linear interpolation. In the next section the cubic spline interpolation is discussed. Afterwards a small extension of both interpolation algorithms is presented. This chapter is concluded by a comparison of the results of the linear and cubic spline interpolation.

3.1. Continuous linear approximation

The continuous linear approximation is based on the formula for linear interpolation:

$$f(x) = f_0 + \frac{f_1 - f_0}{x_1 - x_0} * (x - x_0)$$

where

$$f_0: The\ value\ at\ the\ next\ lowest\ timestamp$$
$$f_1: The\ value\ at\ the\ next\ highest\ timestamp$$
$$x_0: The\ next\ lowest\ timestamp$$
$$x_1: The\ next\ highest\ timestamp$$

This formula takes two neighbor points of a given point *x* and interpolates the value at the point *x* between them. So the main goal of the algorithm with respect to predict missing values is to take the next highest and next lowest given value for a missing timestamp and to interpolate the value at the missing timestamp using the next highest and next lowest value with help of this formula.

The algorithm needs two tables to work: One table containing all observed timestamps and its values (table 'OBSERVATIONS') , as well as another table containing all the timestamps, whose values are missing corresponding to the definition of missing values given in chapter 2 (table MISSING_VALUES).

The algorithm works as follows: It loops over all tuples in MISSING_VALUES and takes the values of the next highest and next lowest timestamp from OBSERVATIONS and uses the linear interpolation formula to get the value at the timestamp of the missing value. Figure 2 shows a part of the source code which does this procedure.

```
for cur_irow in cur_to_interpolate loop
cur_cursor := cur_irow.ts;

--getting the next highest/lowest ts and values for interpolation
select ts, val into next_lowest_ts, next_lowest_value
from observations
where ts = (select max(ts) from observations where ts < cur_irow.ts
and series_id = cur_irow.series_id) and series_id =
cur_irow.series_id;

select ts, val into next_highest_ts, next_highest_value
from observations
where ts = (select min(ts) from observations where ts > cur_irow.ts
and series_id = cur_irow.series_id) and series_id =
cur_irow.series_id;

 --interpolating using the standard linear interpolation formula
update missing_values set
 lin_ipol_val = (next_lowest_value + ((next_highest_value -
next_lowest_value)/(next_highest_ts - next_lowest_ts)) *
(cur_irow.ts - next_lowest_ts))
where ts = cur_irow.ts and series_id = cur_irow.series_id;
end loop;
```

Figure 2: Code Snippet of the linear interpolation procedure

3.2. Continuous cubic spline approximation

The cubic spline approximation works with the usage of a third degree polynomial function between two points (so called "knot"-points). So, for our case of guessing missing values, the usage of the cubic spline approximation is the following: We define all timestamps and their values which are not missing as knot points. Afterwards, we try to define the third degree polynomial function (also called "spline" function) between every two knot points. If we now have a timestamp with the value missing, then we look up at which spline the timestamp is located, and we then use the spline function at that point to interpolate the missing value. So for *N* given points, we will need *N-1* spline functions $S_i(x)$ of the following form:

$$S_i(x) = a_i + b_i * (x - x_i) + c_i * (x - x_i)^2 + d_i * (x - x_i)^3$$

The characteristic of the cubic spline interpolation is that it uses a 'smooth' curve to interpolate the missing values, smooth meaning in this context that there are so less oscillations as possible. Since higher degree polynomial curves tend to have a lot of oscillations between two given points, the cubic spline interpolation uses a third degree polynomial for the interpolation, fulfilling following four attributes[1]:

$$(1)\ \ S_i(x_i) = y_i\ \ for\ i = 0, 1, \dots, n$$
$$(2)\ \ S_i(x_i) = S_{i-1}(x_i)\ \ for\ i = 1, 2, \dots, n$$
$$(3)\ \ S_i'(x_i) = S_{i-1}'(x_i)\ \ for\ i = 1, 2, \dots, n-1$$

[1] I will not explain in detail the derivation of the formulas, since it would be beyond the scope of this report. For a very detailed explanation see Rauch, S. & Stockie, J. , 2008, Cubic Splines from the website http://www.docstoc.com/docs/69786364/Cubic-Splines, 17.10.2011

$$(4)\ S_i''(x_i) = S_{i-1}''(x_i)\ for\ i = 1, 2, \dots, n-1$$

Taking these attributes into account, we get the following equations for the coefficients a_i, b_i,c_i and $d_i$for $i = 0, 1, \dots, n$:

$$a_i = y_i$$

$$b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{2} m_i - \frac{h_i}{6}(m_{i+1} - m_i)$$

$$c_i = \frac{m_i}{2}$$

$$d_i = \frac{m_{i+1} - m_i}{6h_i}$$

where $h_i = x_{i+1} - x_i$ and

$$m_i = S_i''(x_i)$$

Taking these formulas in account, and setting $m_0 = 0$ and $m_n = 0$ we will get the coefficients by solving the following $(n-1) \times (n-1)$ system:

$$\begin{bmatrix} h_0 & 2(h_0 + h_1) & h_1 & 0 & \cdots & 0 \\ 0 & h_1 & 2(h_1 + h_2) & h_2 & 0 & 0 \\ 0 & 0 & h_2 & 2(h_2 + h_3) & h_3 & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ m_{n-1} \end{bmatrix} = 6 \begin{bmatrix} \frac{y_2 - y_1}{h_1} - \frac{y_1 - y_0}{h_0} \\ \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\ \vdots \\ \frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \end{bmatrix}$$

Since this system consists of a tridiagonal matrix, we can use the so called “tridiagonal matrix algorithm”, also known as “Thomas algorithm” (named after Llewellyn Thomas), to solve this matrix[2].

The result of this matrix will be the values for $m_0, m_1, m_2, \dots, m_n$which we can use with the values for $h_0, h_1, h_2, \dots, h_n$ to calculate the coefficients a_i, b_i, c_i and d_i for each spline function $S_i(x)$.

[2] Also here I will not explain the Thomas algorithm, since it would go beyond the scope of this report. For a detailed explanation see: Depoutre, A., 2000, The Thomas Algorithm from website http://hmf.enseeiht.fr/travaux/CD0001/travaux/optmfn/hi/01pa/hyb74/node24.html, 17.10.2011

```
v_next_lowest_given_ts number := 0; --next lowest timestamp whose value is known
v_next_lowest_given_a number := 0; --coefficient a of the next lowest given timestamp
v_next_lowest_given_b number := 0; --coefficient b of the next lowest given timestamp
v_next_lowest_given_c number := 0; --coefficient c of the next lowest given timestamp
v_next_lowest_given_d number := 0; --coefficient d of the next lowest given timestamp
v_current_ts number := 0; --current timestamp
begin
for cur_m_ts_row in cur_m_ts loop
--assignments of values
v_current_ts := cur_m_ts_row.ts;
select max(ts) into v_next_lowest_given_ts from thomas_helper where ts <= v_current_ts and series_id = p_sid;
select c_a, c_b, c_c, c_d into v_next_lowest_given_a, v_next_lowest_given_b, v_next_lowest_given_c, v_next_lowest_given_d
from thomas_helper where ts = v_next_lowest_given_ts and series_id = p_sid;
 --interpolating the value
update missing_value
set my_c_spl_val = v_next_lowest_given_a + v_next_lowest_given_b * (v_current_ts - v_next_lowest_given_ts) + v_next_lowest_given_c *
(v_current_ts - v_next_lowest_given_ts) * (v_current_ts - v_next_lowest_given_ts) + v_next_lowest_given_d * (v_current_ts -
v_next_lowest_given_ts) * (v_current_ts - v_next_lowest_given_ts) * (v_current_ts - v_next_lowest_given_ts)
where ts = v_current_ts and series_id = p_sid;
```

Figure 3: Code Snippet of the cubic spline interpolation procedure

In summary, the continuous cubic spline interpolation algorithm needs two tables to work: One table MISSING_VALUES where the timestamps with the missing values are stored and another table called THOMAS_HELPER where the helping variables like m_i and $h_i$are stored with the data of the knot-points and coefficients a_i, b_i, c_i, d_i. The method to retrieve the missing value is following: First, we calculate with help of the Thomas algorithm, the coefficients for all spline functions. Afterwards, we look between which timestamps the missing value is located in order to find the correct spline function and its coefficients. Then we use the coefficients for the spline function to interpolate the missing value. Figure 3 shows and excerpt of the source code doing this procedure.

3.3. Extensions for the continuous approximation functions

In the last two sections, we looked at the linear and cubic spline interpolation algorithm. These two algorithms possess a major drawback: We can only interpolate the value, if the missing value is between two given values. For the case, that the very first or very last of the values were missing and are to be calculated, we cannot use these two algorithms. This sections explains a method, with which we’re able to solve this problem.

In order two solve this problem I extended the two approximation algorithms with an additional procedure: If the time series has a missing value which is located at the beginning or at the end of the series, then set the missing value equal to the next nearest neighbor. That is, if we cannot interpolate the missing value, because it is not between two values, then we simply assume that the missing value is the same as the next highest given value (if the value at the beginning is missing) or the next lowest given value (if the value at the end is missing). In this fashion, we can interpolate all missing values of a time series using the two approximation functions.

3.4. Comparison between linear and cubic spline approximation

This section briefly presents the results of the linear and cubic spline approximation function, which we discussed earlier. Figure 4 shows the measurements of the time series with ID = 0 from timestamps (ts) = 200 to ts = 400. Note that there is a missing segment from ts = 286 to ts = 342. This is the part of the time series, which has been interpolated. As we can see, the linear interpolation just interpolated the missing values with the help of a linear function, which connects the missing part with a straight line, whereas the cubic spline interpolation tries to follow the course, which the original curve had before. For this case, the cubic spline function seems to approximate the function better than the linear function.

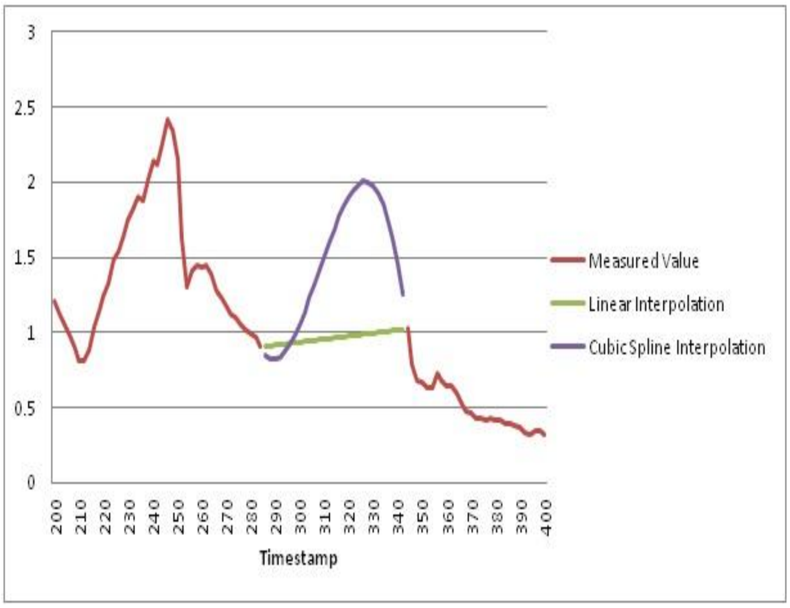

Figure 4: **Linear and cubic spline interpolation of a missing segment from timestamp 286 to 342 (Series ID = 0)**

Let us now look at Figure 5, which shows the same time series, but from ts = 400 to ts = 880. Here the segment of missing values is bigger, it goes from ts = 500 to ts = 800. Note that in the middle of this segment, the value at ts = 650 is given, and therefore not missing. We see the same behavior of the functions here, that is, the cubic spline interpolation tries to imitate the past behavior of the curve. However, in order to maintain the smoothness of the cubic spline curve, it goes into a negative range of values. This is a very dangerous behavior, because if we were working with values, which couldn't be negative (like a stock-prices), then this approximation would be useless. So for this case, I'd assume that the linear interpolation is better.

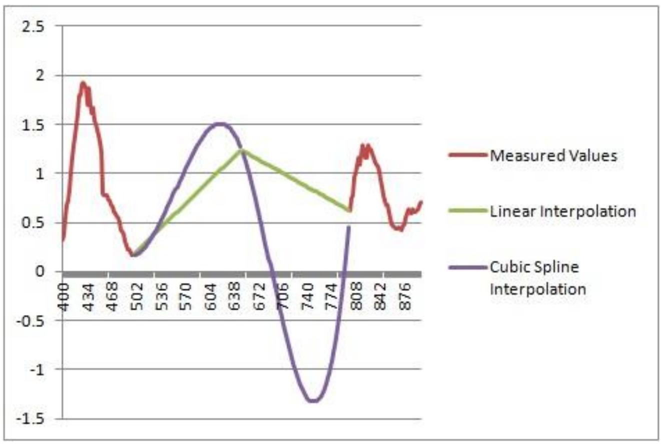

Figure 5: **Linear and cubic spline interpolation of a missing segment from timestamp 500 to 800 (Series ID = 0)**

My conclusion of these two algorithms is that, if the segment of missing values is not big, then the cubic spline approximation function would do better than the linear approximation since, the cubic spline function takes the history of the curve more or less in account which is not the case with the linear approximation function. However, if we have to deal with a big segment of missing values, then the results of the cubic spline function can perhaps be not as good as the linear interpolation function, depending on the kind of data we're working with.

4. Piecewise approximation functions

After discussing about the continuous approximations, we discuss in this chapter about piecewise approximation functions. The difference between the continuous and piecewise approximations is the fact that piecewise approximation functions divides the time series into segments and apply the approximation within the segments. Therefore, we do not need to find a global approximating function for the whole time series, which could be a difficulty when working with streaming time series. So we can use a piecewise approximation in order to reduce complexity of a time series.

The main challenge of piecewise approximation consists of finding algorithms which segments the time series in an efficient way regarding both performance and accuracy. Several segmenting techniques are proposed and analyzed regarding their performance and accuracy (Keogh et al., 2001). Due to reasons of simplicity, I didn't use any of those segmenting algorithm proposed in literature for this report, but I'm aware of the fact that I could have achieved better approximation results when I used those algorithms.

This chapter first introduces the piecewise linear approximation, followed by the piecewise cubic spline approximation. This chapter concludes with having a look on piecewise aggregate approximation. For each of the approximation techniques, I introduce the algorithm, with which I segment the time series. Afterwards, I explain how my approximation algorithm works. At the end, the results of these three piecewise approximation functions are presented.

4.1. Piecewise linear approximation

The piecewise linear approximation (PLA) works basically like the continuous linear approximation. The difference between the two types is that we do not use our linear approximation formula between every two points, instead, we use the formula between each starting and ending point of a segment. Like explained earlier, the segmentation is of much importance in consideration of accuracy and performance. The following section explains my segmentation.

4.1.1. Segmentation for piecewise linear approximation

To reduce the complexity of a time series in order to approximate it, I decided to segment the time series in such way, so that the trends are not lost. That is, I calculated all extremas (global and local) of the time series and stored them in the table 'SEGMENTING_POINTS_PLA'. The first row of that table would represent the starting point of the first segment . The next row would represent the ending point of the first segment, and at the same time, it would represent the starting point of the second segment. Using this method, I had the starting and ending point for each segment, so that I was able to interpolate between these two points. The reason I decided for this type of segmentation was that if I used any other segmentation, the local attributes like trends of the time series could have been lost.

4.1.2. Algorithm for the piecewise linear approximation

In the last section we discussed how to get the starting and ending points of a segment. After these points were stored, my task was to linearly interpolate the values between the starting and ending points of the section. For this, I just reused the linear interpolation formula presented in chapter 3.1. Instead of only interpolating the missing values, this algorithm interpolates all values between the segment starting and ending points (which are extremas). Figure 6 shows a snippet from the procedure of the PLA. We can see that the code only differs from the continuous linear interpolation in the fact that PLA takes the stored extremas as next highest and next lowest values.

```
for cur_ts_row in cur_ts loop
current_ts := cur_ts_row.ts;
 --assigning all the variables
select max(ts) into next_lowest_ts from segmenting_points_pla where ts < current_ts and
series_id = p_sid;
select val into next_lowest_val from segmenting_points_pla where ts = next_lowest_ts and
series_id = p_sid;
select min(ts) into next_highest_ts from segmenting_points_pla where ts > current_ts and
series_id = p_sid;
select val into next_highest_val from segmenting_points_pla where ts = next_highest_ts and
series_id = p_sid;
 --interpolating using the interpolation formula
update missing_value
set i_val = next_lowest_val + ((next_highest_val - next_lowest_val) / (next_highest_ts -
next_lowest_ts)) * (current_ts - next_lowest_ts)
where ts = current_ts and series_id = p_sid;
```

Figure 6: Code snippet of the PLA procedure

4.2. Piecewise cubic spline approximation

The piecewise cubic spline approximation (PSA) works the same way like the continuous one, with the difference that the PSA does not use all given points as knot points. The first subsection handles the segmentation of the time series. Afterwards we will discuss briefly the algorithm for the PSA.

4.2.1. Segmentation for piecewise cubic spline approximation

The PSA uses another segmenting algorithm than the one presented with the PLA in chapter 4.1.. Since we are trying to approximate the original values, we need to find a segmenting algorithm, which keeps more or less the main characteristics of the curve. So, we have to reduce the complexity of the curve without losing all too much information. I tried to define a segment so, that a segment contains exactly one (local) extrema, that means the segment has to contain at least one peak or valley. Since we need to reduce the complexity of the curve, it is not a good idea to define each small "hump" as a peak. Instead, I used a method proposed by Palshikar(2009) , which is based on the so called "Chebyshev inequality". Palshikar states that based on the Chebyshev inequality, a point x_i is a peak, if:

$$(1)\ x_i > m$$

and

$$(2)\ |x_i - m| \geq h * s$$

where m is the mean value of the distribution of points and s is the standard deviation of the distribution (2009). h can be any positive number which states how much standard deviations away from the mean the given value x_i is(Palshikar, 2009).

With these two given conditions, we are able to check if a given point x_i is whether a peak or not. However, we must be careful while choosing the value for h. When we use a high value for h, we will only get very few peaks, whereas we will get each small hump as a peak for a very low value for h. I tried some different values for h and implemented this extrema-finding procedure in a way, that the user is able to enter his value for h as a parameter while invocating the procedure. In my opinion, a value between 1.2 and 1.5 gives the best results. Besides, when we change the direction of the inequality-operators in the two conditions above, we are able to define the valleys.

In summary, the extrema-finding procedure works as follows: It takes a parameter for h and checks for each point $x_i$in the time series whether the two conditions above are fulfilled or not (To get the valleys, we change the direction of the inequality-operators). If they are fulfilled, the point x_i is considered as a peak, respectively valley, and is stored in the table 'PEAKS_VALLEYS'.

Since we have now all the extremas of the time series, we must deal now with the segmentation. The problem here was that if we had too many peaks or valleys which were very close to another, then we wouldn't get any complexity reduction for our curve, since we would then have too many segments. In order to solve this problem, I decided to introduce a variable called *window_size*, which sets the length of a window. Inside this window, there can only be one peak or valley, that is, if two very close peaks (or valleys) happens to be in the same window, only one of them, namely the higher one (in case of a valley, the lower one), will be regarded as a peak (or valley), the other one will be discarded from the table 'PEAKS_VALLEYS'. Choosing the window size also seemed to have a great impact on the result, since when we use a large window size, a lot of information about the curve will be lost, and on the other side, choosing a small window size would not reduce any complexity. Like for the value of h, I decided that the value for the window size can be given by the user as a parameter for the procedure. After testing some different values for the window size, in my opinion, a window size of 10 produced the best results.

After ensuring that there exists only one peak or valley inside of a window, I took the median timestamp between two neighboring peaks and/or valleys and declared the median as a segmenting point. I repeated this procedure for every two neighboring peaks and valleys and so I got the segmenting points, which I stored in the table 'SEGMENTING_POINTS_PSA_PAA'.

Summing up this subsection, the segmentation procedure of the PSA works as follows: First we use the two conditions of Palshikar to identify the peaks and valleys and store them into the table PEAKS_VALLEYS. Afterwards, we choose a window size and we make sure that there is only one peak or valley inside this window. Then we take the timestamp which lies in the middle of two neighboring peaks or valleys and store them as a segmenting point into the table SEGMENTING_POINTS_PSA_PAA.

4.2.2. Algorithm for the piecewise cubic spline approximation

After storing the segmenting points, we can now use the same algorithm which we used for the continuous cubic spline interpolation. The only difference is that, while we used all given points as knot points in the continuous cubic spline function, we now only use the segmenting points as knot points. I will not go in detail about the algorithm, since it is explained in chapter 3.2.. However, after doing some tests with this algorithm, I found that the results were too weak. So I decided to add the extrema-points, which we calculated in the subsection above, as additional knot points, to get better results. The results were much better.

4.3. Piecewise aggregate approximation

The piecewise aggregate approximation is an approximation technique, which simply calculates the average value of a given segment and replaces all the existing values of that segment with the average value. This approximation could be used for various situations, where the complexity of a series should be reduced strongly, without losing too much information.

This section first explains the segmentation which I used for the PAA, which is followed by the description of the algorithm which does the piecewise aggregate approximation.

4.3.1. Segmentation for the piecewise aggregate approximation

Like for all the other piecewise approximation functions, the segmentation is one the most important issues to think about. I used the same segmentation algorithm for the piecewise aggregate approximation (PAA), which I've used for the PSA discussed in chapter 4.2.1.. Inside the segment, the calculation for the mean value will be done with the algorithm presented in the next subsection.

4.3.2. Algorithm for piecewise aggregate approximation

The algorithm for computing the piecewise aggregate approximation is fairly simple. We take all the values from the table OBSERVATIONS which lies between two segmenting points from the table SEGMENTING_POINTS_PSA_PAA and we take the mean of them. After taking the average, we replace the old values with the new calculated average. Figure 7 shows a snippet of the code, which calculates the mean for each segment.

```
for cur_seg_row in cur_seg loop
  current_ts := cur_seg_row.ts;
  select min(ts) into next_highest_ts
  from segmenting_points_psa_paa
  where ts > current_ts and series_id = s_id;

  select avg(val) into avg_val
  from observations
  where ts >= current_ts and ts < next_highest_ts and series_id =
s_id;

  update missing_values
  set paa_val = avg_val
  where ts >= current_ts and ts < next_highest_ts and series_id =
s_id;
end loop;
```

Figure 7: Code snippet of the PAA procedure

4.4. Comparison of the piecewise approximating functions

The last three sections explained three kinds of piecewise approximation functions, namely PLA, PSA and PAA. In this section, I want to compare these three piecewise approximating techniques and discuss my

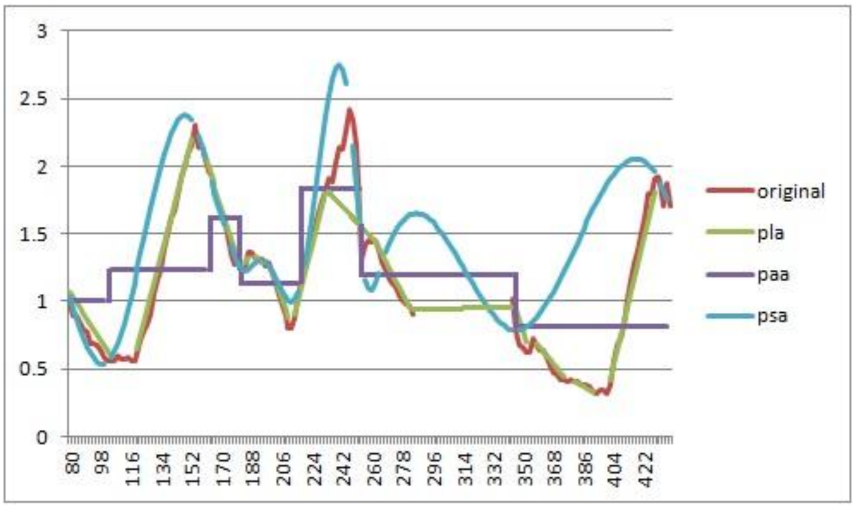

Figure 8: PLA, PSA and PAA on a missing segment from timestamp 284 to 342 (Series ID = 0)

conclusion of the piecewise approximating techniques.

Figure 8 represents the time serie with the ID = 0 from ts = 80 to ts = 440. Inside this scope, the values from ts = 284 to ts = 342 are missing. I used all the three approximation techniques described in this chapter to approximate this part of the time series. As we can see, the PLA approximates the time series, in a very good way, but the guessing of the missing values is unfortunately not so good, since it only connects two given points with a straight line. On the other hand, one can ask if it is good when the original curve is approximated so exactly. Since the PLA approximates too good, one cannot speak about an approximation, since here is no complexity reduction visible. However, the opposite is the case by the PAA. The PAA is just represented by the mean value of a segment. In my view, the PAA is not suitable to guess missing values, since important informations about peaks and valleys can get lost when using the PAA to approximate a time series, but I certainly think that there are other domains where the PAA could be useful.

The best performance in guessing the missing values is shown by the PSA, which more or less represents a simplified version of the original timestamp, and which assumes that the time series has a peak at the ts = 288. This cannot be correct, since the missing segment starts at 284 and when we look at the original value, there is no chance that there could be a peak. However, when we would shift the peak of the PSA at ts = 288 a little more to the right, then chances are high that there could be a peak. I conclude from this fact that the PSA could be a good technique for interpolating missing values, but my implementation is not yet 100% accurate.

5. Comparison of continuous and piecewise approximation functions

We have now discussed various continuous and piecewise approximation techniques in the last two chapters. We found out, that the piecewise cubic spline approximation is superior than the other piecewise approximation functions in terms of interpolating the missing values. Also for the continuous functions, we concluded that the cubic spline interpolation delivers better results than the continuous linear interpolation. In this chapter, we compare the piecewise and continuous cubic spline approximation function in order to find out, which one is better than the other one.

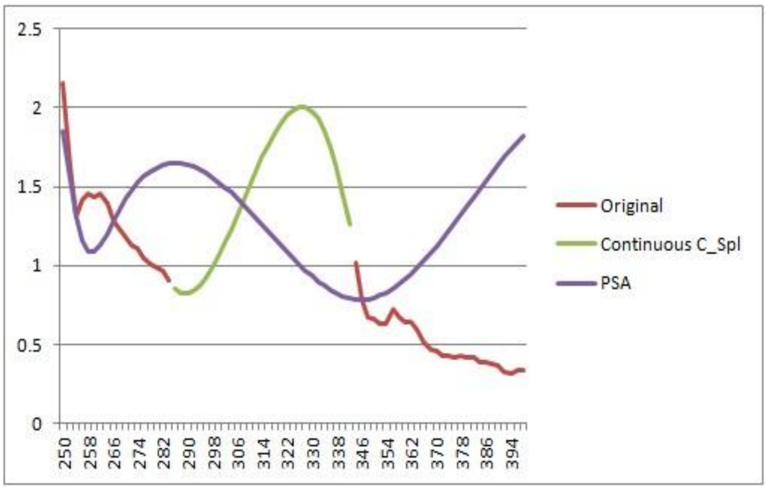

Figure 9: **Continuous and piecewise cubic spline interpolation of a missing segment from timestamp 250 to 398 (Series ID = 0)**

In figure 9 we see a section of the time series with the series ID = 0 starting from ts = 250 and ending at ts = 398. As we can see from the graphic, the continuous cubic spline approximation guesses the missing value (from ts = 286 to ts = 342) much better than the piecewise cubic spline approximation. The same behavior is not shown in figure 10, where we have the same time series, but the section starts at ts = 400 and ends at ts = 880 (missing values from ts

= 500 to ts = 800). Here the the PSA seems to have a better approximation at the first glance, because the PSA approximation does not go far into the negative range of values, as the continuous cubic spline does. But still, if we look at the positioning of the peak, the continuous cubic spline approximations has a more realistical peak-positioning than the PSA. These two figures favor the continuous cubic spline approximation over the PSA, but I think, we can only judge them after doing some more comparing-tests. In the scope of my internship, I didn't had the time to do that, but I think this would be an interesting question to answer in a future research, since the PSA is dependent on the value for *h* and the window size and because my implementation of the PSA can be optimized.

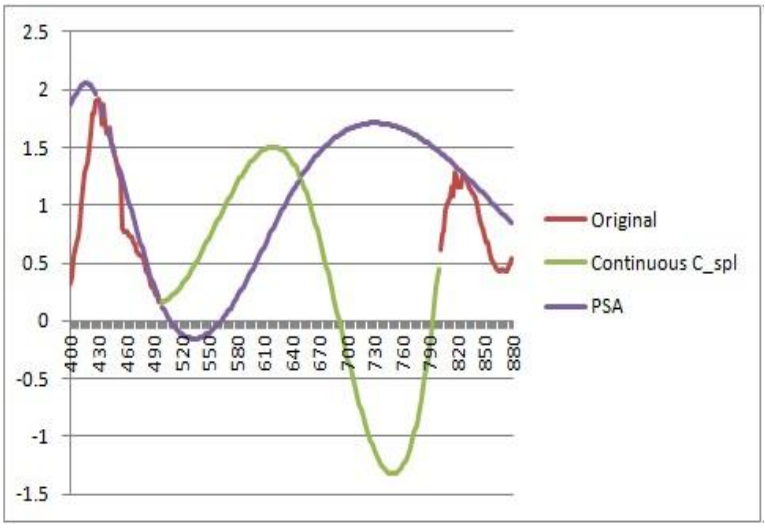

Figure 10: **Continuous and piecewise cubic spline interpolation of a missing segment from timestamp 400 to 880 (Series ID = 0)**

6. Proposition of the database schema

After discussing the various approximation functions, we now know which tables we need. The final relation schema, which I used for my project consists of the following relations and attributes (Primary keys are underlined):

OBSERVATION(<u>SERIES_ID</u>, <u>TS</u>, VAL, ADD_DT, UPD_DT)
Relation to store all observations; with TS for timestamp, VAL for value, ADD_DT for adding date and UPD_DT for updating date.

MISSING_VALUE(<u>SERIES_ID</u>, <u>TS</u>, LIN_IPOL_VAL, C_SPL_VAL)
Relation, where the timestamps with missing values and the (continuously) interpolated values are stored; with LIN_IPOL_VAL for the continuously linear interpolated value and C_SPL_VAL for continuously cubic spline interpolated value.

THOMAS_HELPER(<u>SERIES_ID</u>, <u>TS</u>, VAL, HELP_VAL_M, F_A, F_B, F_C, F_D, H_H, H_B, F_C1, F_D1, C_A, C_B, C_D, C_D)
Relation where all the helping variables are stored in order to execute the Thomas algorithm and to calculate the spline interpolation; with: HELP_VAL_M, F_A, F_B, F_C, F_C, H_H, H_B and F_C1, F_D1 as helping values for the Thomas algorithm, and C_A, C_B, C_C, C_D as coefficients for the spline function.

PEAKS_VALLEYS(<u>SERIES_ID</u>, <u>TS</u>, VAL)
Relation to store the peaks and valleys for segmentation

MISSING_SEGMENTS(<u>SERIES_ID</u>, <u>TS</u>, VAL, PLA_VAL, PSA_VAL, PAA_VAL)
Relation to store the piecewise calculated segments and interpolated missing values; with VAL as the

original value, PLA_VAL as the PLA-interpolated value, PSA_VAL and PAA_VAL as the PSA-respectively PAA-interpolated value.

SEGMENTING_POINTS_PLA(<u>SERIES_ID</u>, <u>TS</u>, VAL)
Relation to store the segmenting points needed for the PLA-interpolation.

SEGMENTING_POINTS_PSA_PAA(<u>SERIES_ID</u>, <u>TS</u>, VAL, HELP_VAL_M, F_A, F_B, F_C, F_D, H_H, H_B, F_C1, F_D1, C_A, C_B, C_D, C_D)
Relation to store the segmenting points needed for the PSA- and PAA-interpolation. The relations contains additionally all helping variables needed to calculate the spline functions for each segment.

7. Conclusions

This report is based on my informatics intership, done at the department of informatics at the university of Zürich. The aim of my project was to implement approximation functions, with the goal of approximating missing values of a give time series. This task was divided into five subtasks, as explained in the introduction. In chapter 2, we defined the term ‘missing value’ and in chapter 3, we discussed one class of approximation functions, namely the continuous ones. After comparing the continuous linear interpolation with the continuous cubic spline interpolation, we concluded that the continuous cubic spline interpolation was generally better than the linear interpolation, but depending on the data the linear interpolation could give better results. In chapter 4, we looked closely at the piecewise approximation functions, which segments the whole time series before approximating it. Also here, we found out that the piecewise cubic spline approximation was better than the piecewise linear approximation function and the piecewise aggregate approximation function in terms of interpolating the missing value. Chapter 5 compared the continuous cubic spline with the piecewise cubic spline approximation. The comparison favored the continuous cubic spline over the PSA, but a definitive answer can not be given in scope of this project, since this project was limited to 12 weeks and more tests were necessary. In chapter 6 we proposed the database schema, with the help of our knowledge gained through the previous chapters.

All in all, the implemented approximation functions worked more or less, but there is still room for optimization, not only in terms of coding, but also in terms of the structure of the algorithm, especially the algorithm for segmenting time series. In literature, there are a variety of possibilities which could provide better results than the results here (for example, see Keogh et al, 2001). I think this issue is something, where I can work on in the future, since it shows a great potential for future research. I am also aware of the fact that, techniques like SVD (Singular value decomposition) provide a much better way to retrieve the missing values. Besides, the comparisons could be done much better when I had more time to test the implemented functions with all available time series. Despite of these drawbacks, I can look at my work with pride since this internship was my first step into a whole new matter for me, and I successfully managed to get a basic understanding this matter, so that I can imagine of working at this topic in the future.

8. References

Depoutre, A., 2000, The Thomas Algorithm from website
http://hmf.enseeiht.fr/travaux/CD0001/travaux/optmfn/hi/01pa/hyb74/node24.html, 17.10.2011

Keogh, E., Chu, S., Hard, D. & Pazzani, M. (2001). An Online Algorithm for Segmenting Time Series," *ICDM*, pp.289

Rauch, S. & Stockie, J. , 2008, Cubic Splines from the website
http://www.docstoc.com/docs/69786364/Cubic-Splines, 17.10.2011