# An Algorithm for Smarter Heating

MARLON ABELN
MARCUS SUNDELL
BACHELOR´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY |
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

# An Algorithm for Smarter Heating


LUND UNIVERSITY
Campus Helsingborg

LTH School of Engineering at Campus Helsingborg, Department of
Electrical and Information Technology

Bachelor Thesis:
Marlon Abeln
Marcus Sundell

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Lunds Universitet
Lund 2019

## Abstract

This report aims to examine the use and development of a more intelligent heating process in a house. By controlling switches integrated into a fuse, the techniques that can be used to allow an algorithm to predict how long the heating process would take are evaluated. Currently users have to, themselves, estimate how long the heating process will take. The algorithm developed aims to alleviate this problem and to allow the heating to be turned off as long as possible. With this kind of automation, the total efficiency of the heating process aims to be increased.

Using linear regression, an algorithm was developed based on ON/OFF regulation. The developed algorithm attempts to estimate how fast a house can be heated. The system was then tested in a simple test facility in order to determine its functionality. Several mathematical functions for handling the required matrices for a least squares analysis in conjunction with other operations were implemented.

It was found that the algorithm performs acceptably when the heating system is well set up. However, it is slow to adjust its prediction when the thermal properties of the house change. Nevertheless, it was discovered that the tests used in this project were biased. This was due to the fact that the testing facility used had no isolation and the radiator was too weak to heat the facility adequately.


**Keywords:** Heating, Automation, API, Least Squares, Linear Regression

## Sammanfattning

Examensarbetet ämnar att utvärdera användning samt utveckling av en ”intelligent uppvärmning” för ett hus. Genom att kontrollera strömbrytare, kopplade till en säkring, kan olika tekniker som kan användas för en självlärande algoritm utvärderas. För tillfället måste användarna själva försöka förutsäga hur lång tid en uppvärmning kommer att ta. Den utvecklade algoritmen ska hjälpa användarna att förutsäga detta och se till så att värmen endast är igång när den faktiskt behöver vara det. Genom att automatisera processen är det tänkt att energieffektiviteten ska öka.

Med hjälp av linjär regression har en algoritm, baserad på till/från reglering, utvecklats. Den utvecklade algoritmen försöker uppskatta hur snabbt ett hus kan värmas upp. Systemet testades i en enkel testomgivning för att upskatta dess funktionalitet. Det implementerades ett flertal matematiska funktioner, vilka bland annat inkluderar metoder för hantering av matriser samt minsta kvadrat-skattning.

Det visade sig att algoritmen presterade acceptabla resultat när värmesystemet är installerat på ett bra sätt. Det gick dock långsamt för algoritmen att justera in sig då värmespecifika detaljer i ett hus ändrades. Det visade sig även att testen som genomfördes under examensarbetet var mindre bra. Detta berodde på att testutrymmet saknade isolering samt att elementet inte klarade av att värma upp utrymmet tillräckligt.

**Nyckelord:** Uppvärmning, Automation, API, Minsta Kvadrat, Linjär Regression

# Preface

This project was done in cooperation with Tempiro AB for the Bachelor of Science degree in Engineering at the Faculty of Engineering at Lund university. To be able to fully grasp the contents of this report it would help to have knowledge about programming, automatic control, and understand the basics of linear algebra.

We would like to thank Tempiro for the opportunity to work with them. In particular Magnus Lindström who helped us with the system, supplied us with the hardware, and acted as our supervisor from Tempiro. Furthermore, we would like to thank Miljöbron and Malin Planander who established contact between us and Tempiro, and also was available as a supervisor for us when help was needed with our writing. We want to express our exceptionally large gratitude to Mats Lilja, who acted as our supervisor from the university and helped us with all aspects of the project from start to finish.

Lastly, we would like to thank our friends and family who have helped and supported us throughout the project. We would also like to thank them with providing us with a test facility where we could setup Tempiro's hardware, as well as read and review our report.

***Marcus Sundell*** "Bachelor of Science in Engineering, Electrical Engineering"

***Marlon Abeln*** "Bachelor of Science in Engineering, Computer Science and Engineering"

Helsingborg 2019

# Contents

# 1 Introduction

## 1.1 Background

Tempiro is a company that is developing a product that they refer to as ”smart fuses”. This entails that a circuit is located within the housing of the fuse. This circuit enables users to switch a phase on and off in their house through a server.

The main idea is that users are able to control the indoor temperature in their house through a mobile device, e.g. a cellphone or a tablet. Currently users have to manually decide when the server should begin heating.

Our assignment aims to relief the users from having to approximate when to start the heating process. To do this, we wanted to develop a self-learning process to accommodate the users, who only should need to set a specific time the heating should be finished.

Through the use of this algorithm, it should also be possible to develop some form of ”personal scheduling” for heating purposes. An example of this could be that a user know that they will not be home during the day, and have no need for the house to be heated during a specific time-period. The user could then schedule that the house should reach a certain temperature for when they are back home.

## 1.2 Current Status

The current solution requires the users to, themselves, approximate when to start the heating-process to reach a target temperature. The system will then start the process until the target temperature is reached, and then hold it there until the user actively shuts off the process.

The application also has an option to schedule events. For example, a user could schedule when to switch on or off certain lights or if they want to, schedule heating of a house. It is, of course, up to the user what they connect to which switches. The problem with a heating-schedule remains from before, a user have to approximate the time it takes for the house to be fully heated. This could lead to high inaccuracies and low energy efficiency.

## 1.3 Purpose

This Thesis project aims to develop an algorithm for a more intelligent control of heating in a house. For instance, the algorithm should determine when the heating in a house should be activated so that a given target temperature is reached when a given time period is approached.

This algorithm should act as a foundation for Tempiro to develop different platforms which they can market as different services. Furthermore, our algorithm aims to improve Tempiro's current product and through this be able to satisfy their current customers as well as attract new ones.

## 1.4 Goals

An algorithm, which should calculate when an electric phase should be active to reach a specific temperature a specific time shall be developed. Comparison between different mathematical models should be done in order to find the best suited model for this purpose. The input data to the algorithm should contain the current indoor- and outdoor temperatures as well as the historical data of these parameters from Tempiro's database.

From Tempiro, the clients will be given a thermostat for the indoor temperature while the outdoor temperature is collected through a database. This means that no modifications on the hardware are necessary to get access to the above mentioned parameters.

After a model has been developed, an implementation is created to help with the integration of the algorithm into Tempiro's current solution. Testing is performed when a new module has been developed. This is done with help of simulation and through a test facility. Before integration, the algorithm should be evaluated and tested through pilot customers. The feedback could then be used to fix and adjust problems with the algorithm.

## 1.5 Research Questions

The research questions of the project are separated into main- and secondary questions. The main questions were meant to guide the research and development of the project while the secondary questions were devised in order to assist in answering the main questions.

### 1.5.1 Main Questions

- How can an algorithm be designed to get a more energy efficient heating in a house?
- Which kind of system can be developed on top of such an algorithm to reduce the amount of time the heating is on without reducing the level of comfort?

### 1.5.2 Secondary Questions

- Which kind of measured data is relevant to be used?
- Which type of model structure should be used?
- How should the model parameters be estimated?
- Which type of controller should be used? Examples examined will include PWM-regulation (which in itself can include P, PI and PID-regulation) and ON/OFF based regulation.

## 1.6 Motivation

There are several motivations for this project. Firstly, this project could contribute with positive environmental impacts as heating a house can be seen as unnecessary in certain scenarios. In other words, the heating should only be active when necessary. This project also aims to minimize the loss of comfort to the users of the final product. Given the algorithm works well, the likelihood that the algorithm is adopted into daily usage is deemed high.

Secondly, as of now, the topic of machine learning is widespread in the technology sector. By creating an algorithm that uses simple machine learning or a self learning solution, this project can, perhaps, act as a reason to further integrate this technology into mainstream markets.

Finally, the decrease of energy consumption could not only be better for the environment, but also gain the individual user in other aspects. One motivation was that if users could save money using this product, it would become more attractive. This would help both Tempiro as a company, the individual users, and the environment in the long term.

## 1.7 Delimitations

Because the change in temperature is a slow process, the developed algorithm does not require frequent updates. The previously mentioned updates will instead occur every five to ten minutes. The main reason for this is that the calculations will update the algorithm with identical values intermittently. This could result in premature wear of the hardware. Additionally, it's not necessary to update the database with the same values repeatedly.

Furthermore, minor discrepancies in the time when the target temperature is reached can be tolerated. For instance, it can be tolerated that a room set to 21 degrees is around 20 degrees when the target time is reached. This facilitates simpler procedures for updating the model, in addition to less complicated adjustments to temperature changes throughout the year.

Moreover, the size of the project can limit what is able to be accomplished during the given time span. This implies that the PWM part of the project may be reduced to a mainly theoretical section.

The tests leading up to the finished algorithm will be made in various small facilities. This can potentially affect the accuracy of the algorithm negatively because of a faster heating-process. The data given from the tests will be essential for going forward testing on a more realistic scenario, where the results then may differ.

# 2 Theory

This chapter brings up various mathematical theories used in the development of the algorithm, as well as the various ways the algorithm could be implemented. Explanations of the interfaces used is also brought up in this section.

## 2.1 Linear regression

Linear regression is a method of estimating a linear relationship between a set of inputs to a single output. This is a form of supervised machine learning, which means that for the set of input values, the output values are known. The result of performing linear regression is going to result in a weighted sum of the inputs.

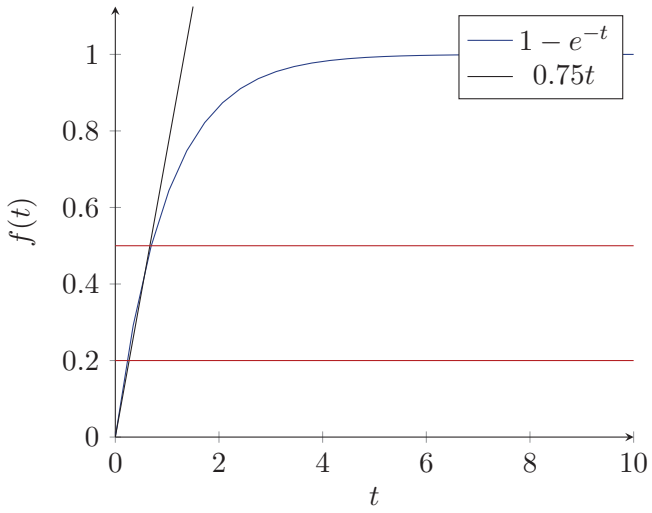



Figure 2.1: Heating curve and linear approximation

As the heating of a house isn't linear, to be able to use a linear model, an assumption must be made regarding the validity of linearity. In Figure 2.1 the linear function and the exponential function approximately follow each other in the beginning of the functions. After some time, the exponential function rather quickly begins to go towards its maximum. The assumption is, that as long as the heating process lay between the red lines of Figure 2.1, the heating of a house can be assumed linear.

To specify the argument of linearity, the time constant will be assumed to be very long. Also, the static gain is assumed to lay on a level high above the measured area. This is described in Figure 2.1.

### 2.1.1 Hypothesis function

When training, the hypothesis can be estimated. This means that we solve the weighted sum with our current weights. The hypothesis function can be expressed as [1, s. 79]:

$$h_w(x) = w_0x_0 + w_1x_1 + ... + w_nx_n + b \tag{2.1}$$

(2.1) can later also be used to predict values, however during training this is called the hypothesis.

### 2.1.2 Cost function

When training, the cost of the weights will need to be considered. The cost function expresses how inexact the estimated model is. This function will need to be minimized in order to get an as exact prediction as possible. The cost can be expressed as the sum of the difference between the known Y value and the output from the hypothesis function. This can be written as [1, s. 80]:

$$RSS(w) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - h_w(x_i))^2 \tag{2.2}$$

Where n is the number of weights, w are the weights, and x are the inputs.

### 2.1.3 Linear regression through least squares

Least squares is a method to perform linear regression. If one looks back upon the cost function, linear regression aims to determine the weights. Given a

linear model [2]:

$$y_i = m + kx_i \tag{2.3}$$

That instead could be written with matrices:

$$\begin{pmatrix} 1 & x(0) \\ 1 & x(1) \\ \vdots & \vdots \\ 1 & x(n) \end{pmatrix} \begin{pmatrix} m \\ k \end{pmatrix} = \begin{pmatrix} y(0) \\ y(1) \\ \vdots \\ y(n) \end{pmatrix} \tag{2.4}$$

And, for ease of use, the matrices could be redefined as:

$$\Phi\theta = Y \tag{2.5}$$

Because we want to solve for $\theta$, which contains the coefficients, a first step would to be multiplying both sides in (2.5) with $\Phi^{-1}$ from the right, which would result in:

$$\theta = \Phi^{-1}Y \tag{2.6}$$

The inverse of $\Phi$ might not be possible, because it isn't always a fact that $\Phi$ is a quadratic matrix. This method is therefore not preferable. Instead, we can create a guaranteed quadratic matrix, that is probable to have an inverse, and then repeat the process.

By instead multiplying both sides of (2.5) with $\Phi^T$ from the right, we can guarantee the quadratic matrix, which can be inverted as long as the determinant isn't zero.

$$\Phi^T\Phi\theta = \Phi^T Y \tag{2.7}$$

Solve for $\theta$ in (2.7) by multiplying both sides with $\left(\Phi^T\Phi\right)^{-1}$ from the right:

$$\theta = \left(\Phi^T\Phi\right)^{-1}\Phi^T Y \tag{2.8}$$

Because $\Phi$ and $Y$ have the same amount of rows, the $\Phi^T Y$ will create a matrix with one column and the same amount of rows as the $\left(\Phi^T \Phi\right)^{-1}$ matrix. When multiplied, the result of the $\theta$ matrix will contain one column and the same amount of rows, where each row will be the sought coefficient from the initial equation.

## 2.2 API

Tempiro has an API which can be used to interact with the company's server by e.g. sending PUSH and GET requests to the server. Through this, different values can be downloaded, such as the outdoor temperature and historic temperature values.

To protect unauthorized access, an access token is required. This token has to be sent with a request in order for the server to be able to authenticate a user and give them access to their information. By sending a series of requests to the servers API, it's possible to interact with it and therefore be able to send commands to it. This could, for example, be used to develop a micro service or set up a test environment.

### 2.2.1 JSON data format

The data returned from the server is in the JSON format. JSON stands for "Java Script Object Notation" and is a format that objects can be serialized into to be read by a receiver at a later point in time. The syntax gives that, for example, an object with an "int" variable that is called x and has the value "10" can be serialized to the form.

```
1 {
2     "x" : 10
3 }
```

This example is constructed using [3].

This can be performed using the "JavaScriptSerilizer" class using c#.
The previous JSON example could be generated when serializing the following example class.

**Listing 2.1: Example class**

```
1 class EX
2 {
3     int x {get; set;}
4     public EX(int xinit)
5     {
6         x = xinit;
7     }
8 }
```

Using the ”JavaScriptSerializer” class, this can be serialized to JSON and back into an object using:

**Listing 2.2: Serialising and deserializing the example class**

```
1 EX obj = new EX(10);
2 string json = new JavaScriptSerializer().Serialize(obj);
3 EX newObj = new JavaScriptSerializer().Deserialize<EX>(json
      );
```

This example was created using [4]. After execution of this code, ”obj” and ”newObj” should reference to identical objects.

## 2.3 Controllers

The controller is the system component which controls the heating. This component can control if the heating should be on or off as well as the power of the heating.

### 2.3.1 ON/OFF

ON/OFF based regulation implies that the heating is switched on or off to control the heating. When the heating is on, the power is set to a constant value. This is a simple type of control, where the measured output will oscillate around the desired output. The control signal of this model will only include zero and the maximum value.

### 2.3.2 PWM based

Through the use of PWM based regulation, proper values can be calculated. This allows the possibilities of for example PID controllers to facilitate a more exact curve than with ON/OFF regulation. This will additionally open up for the possibility for optimization such as minimizing the used power.

### Theoretical example

With PWM, ”Pulse Width Modulation”, a duty cycle is used to encode different digital values [5]. For example, if a light-emitting diode is connected to a 5 V source and a cycle of 10 milliseconds is assumed, the LED is turned on 5 milliseconds and then turned of for the remaining 5 milliseconds. Therefore a duty cycle of 50% is created. The LED would have a potential of 5 V over it for the first 5 milliseconds and a potential 0 V over it for the last 5 milliseconds. Through this, the LED appear to have a potential of 2.5 V over it for the entire cycle.

The same theory can be applied to a radiator that is setup to heat a room. A theoretical example is given where a duty cycle of 50% is used to hold a temperature at 23 degrees centigrade.

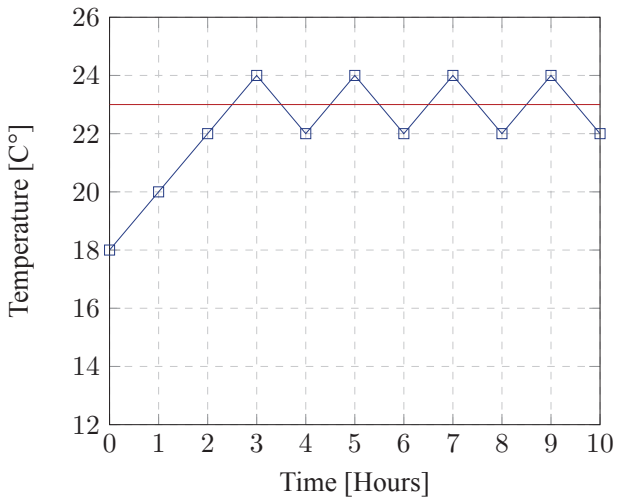



Figure 2.2: Temperature over time

The temperature in Figure 2.2 will oscillate between 24 and 22 degrees centigrade. Over the total cycle, the average temperature will be 23 degrees centigrade.

The voltage supplied to the radiator is illustrated in Figure 2.3.

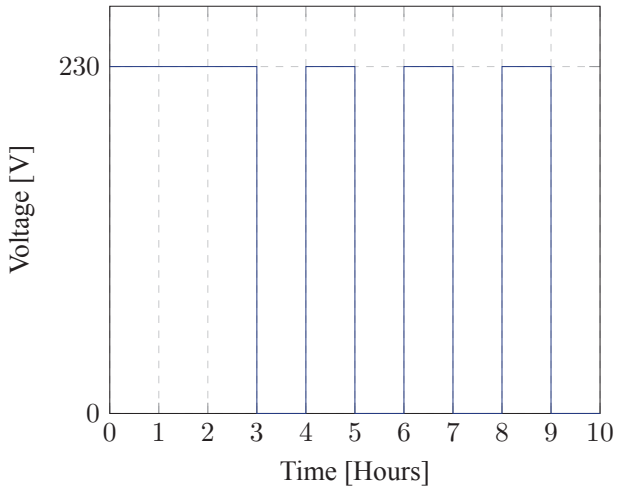



Figure 2.3: Output voltage over time

According to Figure 2.3, the radiator is connected to 230 V. While the radiator is running it is supplied with 230 V, else it is turned off.

Figure 2.4 illustrates a radiator running at 680 W.

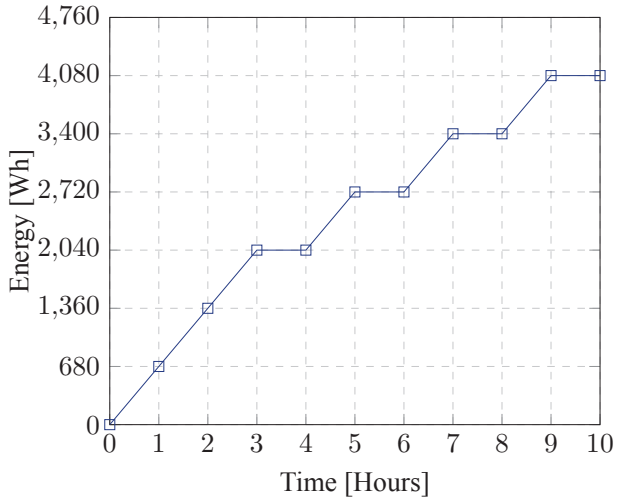



Figure 2.4: Energy consumption over time

While the radiator is turned off, it does not consume any power. Therefore the used power in Figure 2.4 is constant during this time.

# 3 Method

The development process will be divided into several stages. The first will be a discussion about the type of model to be used for the calculations. The data provided to the algorithm will consist in part of historic temperature data of the respective house in addition to the current indoor and outdoor temperatures.

This gives, in principle, two types of models to be considered:

- **Static energy based model:** Based on measured data, an estimate for how much energy is needed to raise the temperature of the house by one degree centigrade. This data can then be used to estimate the duration of the heating process.
- **Recursive dynamic model:** The algorithm attempts to detect a pattern in the presented data stored in the database for the respective houses. This data is then used to create a dynamic equation that can be used to estimate the thermal processes of the house.

After the models have been analyzed, an algorithm will be developed and tested through various simulations to verify its functionality. When the algorithm works as expected it will be tested both in a dedicated test facility and, if possible, with a pilot customer.

The algorithm will be implemented with the programming language C#, this is due to the fact that this language is estimated to be the simplest to be integrated into Tempiro's server. This is mainly as the back-end solution is developed with the same programming language on the .NET framework.

The algorithm will be developed outside of Tempiro's system and only use the server as a tool. To directly implement the system in the server could lead to unnecessary time consuming errors to fix. During the development phase, the algorithm will communicate with the server through an API. After the algorithm is completed it will be deployed on Tempiro's server as a back-end module.

## 3.1 Database analysis

The database was examined using Microsoft SQL Server. The data was analyzed to determine what values can be accessed and to see if there are any patterns. These patterns could be used to determine a model for the thermal properties of the house a user is living in. For this, access to the historic data of the users is required. This data can, for example, be graphed to determine how the house reacts when the heating is enabled. Additionally, the values present in the database can be compared to known models for heating in a house to determine which, if any, is suitable.

## 3.2 Algorithm development

The algorithm will be developed in Microsoft Visual Studio. The development should aim to be as independent from third party libraries as possible. Due to this, certain low level modules might have to be developed from scratch.

## 3.3 Testing

The developed system will be tested in several dedicated testing environments. Testing will be done outside Tempiro's system, and a separate program will be created for testing purposes only.

The system will be allowed to heat the testing facility to a target temperature, after which the room will be allowed to cool down again. The procedure is subsequently repeated for all tests. This is done in order to collect data regarding the performance of the algorithm.

## 3.4 Source criticism

[1] The author has published several books on this subject in addition to several articles in academic journals.

[2] Is deemed to be a valid source due to that the mathematical theory is the same as what we've learned in the courses "Mathematics, Linear Algebra and "Automation, Advanced Course". We've also tested the theory with mathematical problems where we know the answer, in which we get the same answer with our method.

[3] Is a document outlining the Json standard. The format has even been compared to the output from the "JavaScripSerializer" class that is included in Microsoft's C# standard library.

[4], [8] Is part of Microsoft's official documentation for their C# language. As they have developed and maintain the C# language and standard library, the sources are deemed to be valid.

[5] The author of this article has written several books and articles about embedded programming. The author has also acted as an adjunct professor of electrical and computer engineering.

[6] Is deemed to be a valid source due to that the creators of the website have bachelors and masters degrees in mathematics.

[7] The Wolfram|Alpha answer engine is developed by the company Wolfram Research where the CEO Stephen Wolfram is well known for his work in computer science, mathematics and theoretical physics.

# 4 Implementation

## 4.1 Deciding on a model

As heating is a slow process the assumption was, which was described in Figure 2.1, that it would be an almost linear curve for increasing the temperature of a house. Because access to a database with various variables was available, the discussion of self learning, or machine learning was also brought up.

Linear regression was an obvious choice, together with ordinary linear equations. However, since there could be more than one variable in the calculations, linear regression might not suffice. A natural step forward would be to implement the least squares method, which can be used with multiple variables.

## 4.2 Implementing least squares

The least squares method is to be implemented using matrix-multiplications as described in (2.8). This includes coding for both calculating the inverse of a matrix and the transpose of a matrix.

## 4.3 Matrix operations

To be able to perform a least square analysis, several matrix operations had to be implemented. The full implementation for the ”matrix class” is given in Appendix A, Listing: 9.1.

### 4.3.1 Transpose

The transpose of a matrix is done by switching its rows with its columns. For a matrix $A$, the transpose $A^T$ will result in:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \iff A^T = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \tag{4.1}$$

This can be achieved by iterating through a matrix and putting the value on the diagonal. This is expressed in the Listing 4.1. This code will create a new matrix, as most matrix operations do, which is then returned. Due to this, the program will iterate over the entire matrix and put the values in the correct spots instead of switching places between them, which would have fewer iterations through the for-loops.

**Listing 4.1: Transposing a matrix**

```
1  public Matrix Transpose()
2  {
3      double[,] new_matrix = new double[data.GetLength(1),
           data.GetLength(0)];
4      for (int i = 0; i < data.GetLength(0); ++i)
5      {
6          for (int j = 0; j < data.GetLength(1); ++j)
7          {
8              new_matrix[j, i] = data[i, j];
9          }
10     }
11     return new Matrix(new_matrix);
12 }
```

### 4.3.2 Inverse

When inverting a matrix two things must be calculated, the determinant and the indices of the inverse matrix.

The algorithm's matrix inverse calculations were developed following the guide in [6]. The guide and calculations were fact checked using the Wolfram|Alpha platform [7].

To demonstrate how the algorithm finds the matrix inverse, the $A$ matrix from (4.1) will be used:

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \tag{4.2}$$

**Step 1: Create a matrix of minors**

The values of the indices for the new matrix of minors are determined by the following calculations:

$$\bar{a_1} = \begin{bmatrix} \circ & & \\ & b_2 & b_3 \\ & c_2 & c_3 \end{bmatrix} = b_2 * c_3 - c_2 * b_3 \tag{4.3}$$

$$\bar{a_2} = \begin{bmatrix} & \circ & \\ b_1 & & b_3 \\ c_1 & & c_3 \end{bmatrix} = b_1 * c_3 - c_1 * b_3 \tag{4.4}$$

$$\bar{a_3} = \begin{bmatrix} & & \circ \\ b_1 & b_2 & \\ c_1 & c_2 & \end{bmatrix} = b_1 * c_2 - c_1 * b_2 \tag{4.5}$$

$$\bar{b_1} = \begin{bmatrix} & a_2 & a_3 \\ \circ & & \\ & c_2 & c_3 \end{bmatrix} = a_2 * c_3 - c_2 * a_3 \tag{4.6}$$

$$\bar{b_2} = \begin{bmatrix} a_1 & & a_3 \\ & \circ & \\ c_1 & & c_3 \end{bmatrix} = a_1 * c_3 - c_1 * a_3 \tag{4.7}$$

$$\bar{b_3} = \begin{bmatrix} a_1 & a_2 & \\ & & \circ \\ c_1 & c_2 & \end{bmatrix} = a_1 * c_2 - c_1 * a_2 \tag{4.8}$$

$$\bar{c_1} = \begin{bmatrix} & a_2 & a_3 \\ & b_2 & b_3 \\ \circ & & \end{bmatrix} = a_2 * b_3 - b_2 * a_3 \tag{4.9}$$

$$\bar{c_2} = \begin{bmatrix} a_1 & & a_3 \\ b_1 & & b_3 \\ & \circ & \end{bmatrix} = a_1 * b_3 - b_1 * a_3 \tag{4.10}$$

$$\bar{c_3} = \begin{bmatrix} a_1 & a_2 & \\ b_1 & b_2 & \\ & & \circ \end{bmatrix} = a_1 * b_2 - b_1 * a_2 \tag{4.11}$$

The white dot in the matrices represents the index value that is to be calculated. The row and the column where the new index value is calculated are blanked out. The index value is then calculated with the same method used to determine the determinant of a 2 x 2 matrix. The determinant calculations is written out for each matrix index.

For further reference, the matrix of minors will hereafter be represented as:

$$\begin{bmatrix} \bar{a_1} & \bar{a_2} & \bar{a_3} \\ \bar{b_1} & \bar{b_2} & \bar{b_3} \\ \bar{c_1} & \bar{c_2} & \bar{c_3} \end{bmatrix} \tag{4.12}$$

The described method can be expressed in C#.

**Listing 4.2: Creating a matrix of minors**

```
1  private Matrix Minors()
2  {
3      double[,] minors = new double[data.GetLength(0), data.
           GetLength(1)];
4      for (int row = 0; row < data.GetLength(0); ++row)
5          for (int col = 0; col < data.GetLength(1); ++col)
6          {
7              minors[row, col] = DeterminantRekur(Decompose(
                   data, row, col));
8          }
9          return new Matrix(minors);
10 }
```

As the operation to create a matrix of minors uses partly the same procedure as calculating the determinant of part matrices, the same recursive method can be used to implement this function. Here we iterate over the matrix and calculate the determinant for every index.

### Step 2: Create a matrix of cofactors

To create the matrix of cofactors, change every other of (4.12) indices to the negative value. This is done in a square pattern, where the diagonal is positive:

$$\begin{bmatrix} + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ + & - & + & - & \cdots \\ - & + & - & + & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{4.13}$$

The matrix of cofactors will be represented as:

$$\begin{bmatrix} \bar{a_1} & -\bar{a_2} & \bar{a_3} \\ -\bar{b_1} & \bar{b_2} & -\bar{b_3} \\ \bar{c_1} & -\bar{c_2} & \bar{c_3} \end{bmatrix} \tag{4.14}$$

This gives a checkerboard pattern that can be laid over the matrix of minors.

**Listing 4.3: Creating a matrix of cofactors**

```
1  private Matrix Cofactors()
2  {
3      double[,] cofactors = Minors().data;
4      for (int row = 0; row < data.GetLength(0); ++row)
5      {
6          for (int col = 0; col < data.GetLength(1); ++col)
7          {
8              //Om raden är udda OCH kolonnen är jämn ELLER
                   om raden är jämn OCH kolonnen är udda
9              if ((row % 2 != 0 && col % 2 == 0) || ((row % 2
                    == 0 && col % 2 != 0)))
10             {
11                 cofactors[row, col] = -cofactors[row, col];
12             }
13         }
14     }
15     return new Matrix(cofactors);
16 }
```

The method simply iterates over the matrix and based on certain parameters, the value is multiplied with -1. The parameters give the aforementioned checkerboard pattern. As multiplying anything with a positive one has no effect on the value, only the case in which the value is multiplied with -1 has to be handled.

### Step 3: Find the matrix adjoint

To find the matrix adjoint, transpose all the elements from (4.14) as explained in (4.1). The matrix adjoint will then be represented as:

$$\begin{bmatrix} \bar{a_1} & -\bar{b_1} & \bar{c_1} \\ -\bar{a_2} & \bar{b_2} & -\bar{c_2} \\ \bar{a_3} & -\bar{b_3} & \bar{c_3} \end{bmatrix} \tag{4.15}$$

This is simply implemented as a call to the transpose method of the cofactor matrix.

### Step 4: Find the determinant of the original matrix

For most matrix operations, such as the inverse, a matrix determinant is needed. This was implemented using a recursive method that breaks down the matrix

into minor matrices and calculates the determinant of each of them. Through this, the determinant of matrices of any size can be found. If it's not possible to find the determinant of a matrix, an exception is thrown.

For this example, the determinant is calculated by taking the original matrix index values of any row and multiply them by the cofactors of the same indices. For the top row, the determinant equation would be:

$$detA = a_1 * \bar{a_1} + a_2 * (-\bar{a_2}) + a_3 * \bar{a_3} \tag{4.16}$$

Using a recursive approach, the determinant of any matrix can be calculated. This is done by calculating the determinant of smaller "part matrices" and adding them together using (4.16).

**Listing 4.4: Recursive determinant calculation**

```
1  private double Determinant()
2  {
3      return DeterminantRekur(data);
4  }
5  private double DeterminantRekur(double[,] values)
6  {
7      if (values.GetLength(0) == values.GetLength(1))
8      {
9          if (values.GetLength(0) == 1)
10         {
11             return values[0, 0];
12         }
13         double det = 0;
14         if (values.GetLength(0) == 2)
15         {
16             return (values[0, 0] * values[1, 1]) - (values
                   [0, 1] * values[1, 0]);
17         }
18         for (int i = 0; i < values.GetLength(0); ++i)
19         {
20             if (i % 2 == 0)
21             {
22                 det += values[0, i] * DeterminantRekur(
                       Decompose(values, 0, i));
23             }
24             else
25             {
26                 det -= values[0, i] * DeterminantRekur(
                       Decompose(values, 0, i));
27             }
28         }
29         return det;
```

```
30      }
31      throw new MatrixDimensionException("Unable to find
            Determimant for non-square Matrix");
32  }
```

The matrix is decomposed and added to the determinant sum. This method will then recursively calculate the determinant for all the smaller matrices that are generated when the matrix is decomposed. For this, a decompose-method was needed which removes the row and column just as when the matrix of minors was calculated.

**Listing 4.5: Decomposing a matrix for finding the determinant**

```
1  private double[,] Decompose(double[,] values, int row, int
       col)
2  {
3      double[,] new_dat1 = new double[values.GetLength(0) -
           1, values.GetLength(1)];
4      double[,] new_dat = new double[values.GetLength(0) - 1,
            values.GetLength(1) - 1];
5
6      //remove row
7      int rowindex = 0;
8      int newrowindex = 0;
9      while (rowindex < values.GetLength(0) && newrowindex <
           new_dat1.GetLength(0))
10     {
11         if (rowindex == row)
12             if (++rowindex == values.GetLength(0))
13                 break;
14         for (int i = 0; i < values.GetLength(1); ++i)
15             new_dat1[newrowindex, i] = values[rowindex, i];
16         ++rowindex;
17         ++newrowindex;
18     }
19     //remove col
20     int colindex = 0;
21     int newcolindex = 0;
22     while (colindex < new_dat1.GetLength(1) && newcolindex
           < new_dat.GetLength(1))
23     {
24         if (colindex == col)
25             if (++colindex == new_dat1.GetLength(1))
26                 break;
27         for (int i = 0; i < new_dat1.GetLength(0); ++i)
28         {
29             new_dat[i, newcolindex] = new_dat1[i, colindex
                   ];
30         }
```

```
31          ++colindex;
32          ++newcolindex;
33      }
34      return new_dat;
35  }
```

The decompose-method removes the row and the column that a given index is in. This is exactly like the matrix of minors. A determinant can then be returned to the caller.

**Result**

When all of the sub calculations are done, the inverse of the matrix can be calculated using the determinant and the adjoint:

$$A^{-1} = \frac{1}{detA} \begin{bmatrix} \bar{a_1} & -\bar{b_1} & \bar{c_1} \\ -\bar{a_2} & \bar{b_2} & -\bar{c_2} \\ \bar{a_3} & -\bar{b_3} & \bar{c_3} \end{bmatrix} \tag{4.17}$$

### 4.3.3 Matrix multiplication

When multiplying two matrices, a new matrix is created.

$$AB = C \tag{4.18}$$

To be able to multiply two matrices, the amount of columns in the first matrix must be equal to the amount of rows in the second matrix.

For this example, the $A$ and $B$ matrices will be represented as:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \tag{4.19}$$

$$B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} \tag{4.20}$$

To complete the multiplications and create $C$, multiply the indices of the rows in $A$ with the column indices in $B$ and add them together. As an example, to calculate the first index in $C$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & \end{pmatrix} \tag{4.21}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & \end{pmatrix} \tag{4.22}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & \end{pmatrix} \tag{4.23}$$

The same calculations are done for the remaining indices, which will result in:

$$C = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{pmatrix} \tag{4.24}$$

Matrix multiplication implemented in C#:

**Listing 4.6: Multiplying two matrices**

```
1  public Matrix Multiply(Matrix other)
2  {
3      if (data.GetLength(1) == other.data.GetLength(0))
4      {
5          double[,] c = new double[data.GetLength(0), other.
               data.GetLength(1)];
6          for (int i = 0; i < c.GetLength(0); i++)
7          {
8              for (int j = 0; j < c.GetLength(1); j++)
9              {
10                 c[i, j] = 0;
11                 for (int k = 0; k < data.GetLength(1); k++)
                       // OR k<b.GetLength(0)
12                     c[i, j] = c[i, j] + data[i, k] * other.
                           data[k, j];
13             }
14         }
15         return new Matrix(c);
16     }
17     throw new MatrixDimensionException("Unable to multiply
           two Matrixes where the collumn count of first non
           equal to row count of second");
18 }
```

If the matrices have incorrect dimensions, an exception has to be cast as they cannot be multiplied.

## Least squares

After all the matrix operations were implemented, the mathematical operations to perform a least squares analysis, as shown in (2.8), were ready. However the structure for the input data had to be determined. This implies that some sort of interface for callers to the last squares function needed to be created.

### Structuring tables

In order to easily read and structure data, a table class was devised. The purpose of this class was to give an interface which the least squares operations could easily read. This would also allow to have a better overview of the data for debugging purposes. The full implementation for this class is given in Appendix A, Listing; 9.2. Using a one dimensional array of strings, each column of the table can receive a label. Using this label, the least squares method will not require to know which column the features and targets are located in. This

allows the least squares to take a table and find the correct columns given that they have the correct labels.

### Implementation

Using the created table class, the least squares method was implemented by using (2.8). Initially, the data from the table class is read and placed inside a matrix object. Subsequently, the operations for performing a least squares analysis are executed. The program then instantiates a model object which contains the calculated function in addition to the cost.

**Listing 4.7: Least squares calculation**

```
1  public static Model LeastSquare(Table data, int nbrFeatures
       )
2  {
3      Matrix features = new Matrix(data.numberRows(),
           nbrFeatures + 1);
4      Matrix targets = new Matrix(data.numberRows(), 1);
5      for (int i = 0; i < data.numberRows(); ++i)
6      {
7          targets.put(i, 0, data.get("Y", i));
8          for (int j = 0; j < nbrFeatures; ++j)
9          {
10             features.put(i, j, data.get("Feature" + j, i));
11         }
12         features.put(i, nbrFeatures, 1);
13     }
14     Matrix output = new Matrix(nbrFeatures, 1);
15     Matrix weights = (features.Transpose().Multiply(
           features)).Inverse().Multiply(features.Transpose())
           .Multiply(targets);
16     for (int i = 0; i < nbrFeatures; ++i)
17     {
18         output.put(i, 0, weights.get(i, 0));
19     }
20     double bias = weights.get(nbrFeatures, 0);
21     return new Model(new Functions.
           MultiVariableFirstOrderPolinomial(output, bias),
           cost(features, weights, targets));
22 }
```

The majority of the given code is to read the table class, as the actual calculations are implemented in only a single line. The general approach of a table relieves the user, or someone who later wants to use the method, from needing to know how the matrices required to perform the least squares calculations have to be setup. However, this requires the caller of this method to specify the number of features for the least squares. An additional requirement is, that

in the table that is given to the least squares method as an input, one column has the label ”Y” and the features have the labels ”Feature 0” until ”Feature n-1”, where n is the number of features. These columns are the targets and the features respectively.

## 4.4 Updating the model

When a model is estimated, there are several factors which may change the coefficient. One example of such noise might be an open window where heat may escape. Due to this, an estimated model will need to be moderated through several different estimates. By calculating the average between the different estimated coefficients, the influence of such noise can be reduced.

**Listing 4.8: Updating a model**

```
1 public void update(Model data)
2 {
3     double new_coeff = data.GetFunction().GetCoeff().get(0,
          0);
4     coeff = ((coeff * nbrVals) + new_coeff) / (nbrVals + 1)
          ;
5     ++nbrVals;
6 }
```

The above given function is part of the ”HeatingCoefficient” class. The full implementation for the class is given in Appendix A Listing 9.7. For this there are two attributes that are stored. These are the average coefficient, named ”coeff”, and the denominator, named ”nbrVals”. The average is then multiplied with the denominator to get a sum of the previous values. The new coefficient is then added to the sum and is divided by the incremented value for the denominator.

## 4.5 Handling different outdoor temperatures

The least squares estimate will give a coefficient that states how many degrees centigrade per hour a house can be heated. However, this may change depending on the outdoor temperature. In a well isolated house, this relationship may be less noticeable. However, in a house with less isolation, the relationship could be quite significant. Other factors that might play a role is if the outdoor temperature affects the indoor temperature, the size of the house, and how many rooms the house has.

Therefore, the estimated model from the least squares analysis will take into account how fast the house heats up under certain conditions, when the data was collected. To have a more general approach to this, the outdoor temperatures are divided into intervals of five degrees centigrade. The intervals are treated using one single equation. This means that the system will have to be set up for different outdoor temperatures. The different generated models are stored in a Hash-Map and can be accessed using a key, calculated from the outdoor temperature.

**Listing 4.9: Calculating the map index**

```
1  public int DictIndex(double OuterTemp)
2  {
3      int DictIndex = (int)System.Math.Round(OuterTemp);
4      if (OuterTemp < 0)
5      {
6          DictIndex -= 5;
7      }
8      return DictIndex /= 5;
9      }
10 }
```

For this, the outdoor temperature is first rounded into an integer. If it is negative it is shifted by -5, to avoid using zero twice as an index. The value is then divided by 5 using integer division. Because of this, everything within the interval is generalized into one index. However due to this, one interval will be between -1 and -4, which means that there will only be four temperatures in that interval. This allows for several models to be stored and used in the system.

Using the given calculation, an array or Hash-Map can be used to hold the models. An example of this is given in Appendix A, Listing: 9.7. The class ”HeatingAgent” contains the implementation through a Hash-Map.

## 4.6 Putting the algorithm together

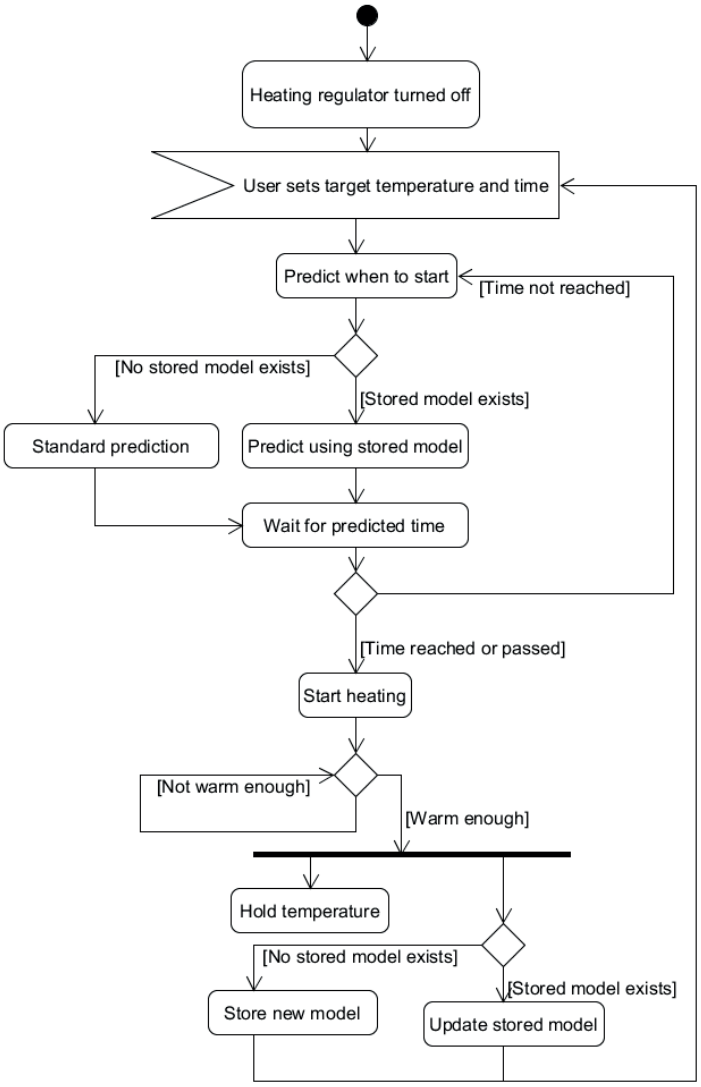



Figure 4.1: Example implementation of the algorithm

Figure 4.1 displays how the execution of the algorithm transpires. First, user input is required to setup the target temperature and at what time this target

should be reached. Subsequently, it will be checked whether there is a model stored for a given interval of outdoor temperature. Given there is a stored model, it is used. Otherwise the algorithm will assume heating with one degree centigrade per hour. After the given calculation, the system has to wait for the adequate time to start heating. As the environmental conditions are subject to change, there is the possibility that the heating would have to be started at a different time than the one that was given. Due to this, every time the temperature is updated, the time when to start heating would need to be re-evaluated. Once the time to start heating has been reached, the algorithm will proceed to the next stage.

The system will then simply wait for the target temperature to be reached. Once this condition has been met, the current temperature should be held and the model should either be updated or a new model should be created, depending on whether there is a previous model stored or not. Once new information is given to the system on the new target temperature, the system should aim to reach this temperature again.

When this is integrated into a system, several changes to the structure in Figure 4.1 can be made to better fit the target architecture. For instance, if a system updates from a thermometer every five minutes, the new prediction can be done when the temperature is updated.

## 4.7 Testing

In order to evaluate the method used in the algorithm, a testing scenario was devised. This was done in order to draw conclusions about the type of model used and to give improvement ideas for the system developed as well as the algorithm.

### 4.7.1 Test program

The test program that was developed utilized Tempiro's API to communicate with the server. The Implementation for the program and the API connection is given in Appendix A Listing 9.6. The program first creates an API object using the access token that the user needs from the server. As this is a program designed for testing, the token is hard-coded into the program. Using this, the program attempts to download a list of thermostats from the server. The program then needs to instantiate a ”heating agent”, which requires the API and the device that the user has chosen. This class handles both the models and

their respective updates.

The program later requests the user to enter at which time it should be warm and what target temperature should be reached. The system then attempts to predict how long it will take for the house to reach the target temperature. Initially, this will be one degree centigrade per hour. The system then waits for the predicted moment. However, to account for changes in the indoor and outdoor temperatures, the system will re-predict when the heating should start regularly. Once the predicted moment is reached, the system sets the servers target temperature and triggers the fuses to turn on and the starting time is recorded.

While the system is heating, the temperature is monitored and once the target temperature is reached the fuses are turned off, the servers target temperature is set to zero and the agent is told to update the model from the data between the two recorded time periods. This process then repeats until the user exits the program. This is designed so that the change of the model can be monitored to test whether the model will adjust after heating.

The test program is also aimed to act as an example application to show how such an algorithm can be implemented in a system. Therefore it follows Figure 4.1 closely with the exception that once the heating is complete, it does not hold the temperature but allows the room to cool down so that more tests can be performed.

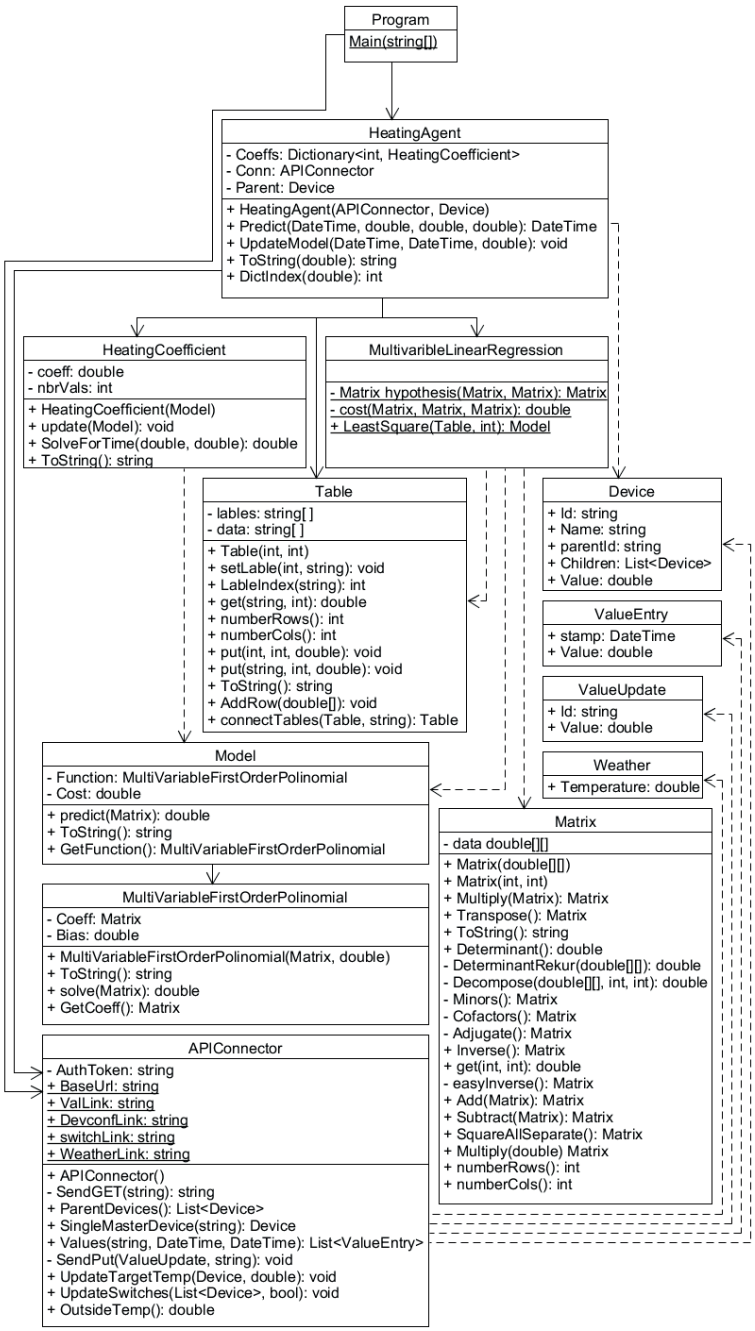



Figure 4.2: Class diagram over the test implementation

Figure 4.2 describes the test program. It can be seen that the ”Matrix” class is only used in the ”MultiVariableLinearRegression” class. This is because the ”Table” class acts as the interface between the ”LeastSquare” function and the caller. There are some classes that are only required for the ”APIConnector” class, which are used to deserialize the data returned from Tempiro’s server.

### 4.7.2 Contacting the API

The API object, mentioned in 4.7.1, communicates between the test program and Tempiro’s server. Also mentioned in 4.7.1 is that the access token is hard-coded into the application code. This was done due to that it was deemed impractical to ask for a username and password every time the program started. The application sends ”GET” requests to get information from the server and ”PUT” requests to send information to the server. The full implementation for the API class is given in Appendix A: Listing 9.6.

**Listing 4.10: Sending a GET request**

```
1  private string SendGET(string url)
2  {
3      HttpWebRequest request = (HttpWebRequest)WebRequest.
           Create(url);
4      request.Method = "GET";
5      request.ContentType = "application/json";
6      request.Headers.Add("Authorization","Bearer " +
           AuthToken);
7      request.AutomaticDecompression = DecompressionMethods.
           GZip | DecompressionMethods.Deflate;
8      using(HttpWebResponse response = (HttpWebResponse)
           request.GetResponse())
9          using(Stream stream = response.GetResponseStream())
10             using(StreamReader reader = new StreamReader(
                   stream))
11             {
12                 return reader.ReadToEnd();
13             }
14 }
```

A ”GET” request is sent to Tempiro’s server using a bearer token. This is the type of access token used. Using a ”StreamReader”, the servers response is read and returned.

For example, a possible ”GET” request for the server could be ”/api/Devices”. The server would then return an array of an object in the form of a JSON string that can be deserialized.

Using the API, ”PUT” requests can be sent, which requests the server to update its information.

**Listing 4.11: Sending a PUT request**

```
1  private void SendPut(ValueUpdate VU , string url)
2  {
3      var httpWebRequest = (HttpWebRequest)WebRequest.Create(
           url);
4      httpWebRequest.ContentType = "application/json";
5      httpWebRequest.Method = "PUT";
6      httpWebRequest.Headers.Add("Authorization", "Bearer " +
            AuthToken);
7
8      using (var streamWriter = new StreamWriter(
           httpWebRequest.GetRequestStream()))
9      {
10         string json = new JavaScriptSerializer().Serialize(
               VU);
11         streamWriter.Write(json);
12         streamWriter.Flush();
13         streamWriter.Close();
14     }
15     var httpResponse = (HttpWebResponse)httpWebRequest.
           GetResponse();
16     using (var streamReader = new StreamReader(httpResponse
           .GetResponseStream()))
17     {
18         var result = streamReader.ReadToEnd();
19     }
20 }
```

This sends a ”PUT” request that contains a ”ValueUpdate”, formatted to JSON, to the server. This method is meant to update the value of the switches and thermostats using the API endpoints ”api/Switch” and ”/api/DeviceConfigurations”.

The relevant documentation for the API functions can be found in Appendix B.

### 4.7.3 Test Environment

For testing purposes, a dedicated test environment that would not be used for anything else than testing during the project was set up. At first, this was a small basement unit in Helsingborg. During the development of the test program, this was also used to monitor whether the fuses could be turned on and off reliably.

The test environment used was a small basement area where an empty fuse-box was mounted. The fuse-box contained a single fuse mount where one of Tempiro's smart fuses was installed. The fuse-box itself was connected to a regular wall output. A cut of extension cord was attached to the fuse output. The radiator was subsequently plugged into this, allowing the fuse to turn the radiator on and off. The gateway for the fuse was connected to an ethernet port in a router. It was then verified that the test program and the application could reliably control the fuse. A trap door allowed access to the basement from a main building that was isolated and heated to around 22 degrees centigrade. The side walls and the floor of the basement had no isolation. This allowed the room to cool down quickly after a test. However, this also had an effect on the heating of the facility.

After some testing, this setup was moved to a storage unit in Lund that was a bit larger and had better isolation. It was hoped that this would be a more accurate representation of an actual house for the testing purposes. This environment had one drawback, there was no dedicated internet in the facility. This meant, unfortunately, that the connection to the setup was unreliable and many tests had to be canceled as the system was unable to control the fuses. After this, the setup was moved back to Helsingborg where it was possible to monitor, and easier fix potential problems that could occur.

# 5 Results

The test environment, described in 4.7.2, was used to run the developed test program ten times. The results for ”Model 2” where noted and can be processed to allow drawing conclusions about the performance of the algorithm.

Table 5.1: Unprocessed set time and temperature values for model stored at index 2

| Trial | Target time | Target temperature [C°] | Start temperature [C°] |
|---|---|---|---|
| 1 | 10/05/2019 20:00:00 | 21 | 17.06 |
| 2 | 11/05/2019 10:40:00 | 21 | 17.75 |
| 3 | 11/05/2019 13:50:00 | 23 | 18.81 |
| 4 | 11/05/2019 18:30:00 | 23 | 20.43 |
| 5 | 12/05/2019 11:00:00 | 22 | 18.18 |
| 6 | 12/05/2019 18:30:00 | 23 | 20.37 |
| 7 | 13/05/2019 10:50:00 | 21 | 18.68 |
| 8 | 13/05/2019 13:00:00 | 22 | 19.12 |
| 9 | 13/05/2019 18:30:00 | 23 | 20.31 |
| 10 | 13/05/2019 21:30:00 | 23 | 19.31 |

The data in Table 5.1, excluding the outdoor temperature, was set to gather data about the algorithms performance. The outdoor temperature was queried from Tempiro’s server, which in turn queried an external service.

Table 5.2: Unprocessed calculated data for the model saved at index 2

| Trial | Estimated start time | Actual start time | Actual end time | Coefficient |
|---|---|---|---|---|
| 1 | 10/05/2019 16:03:36 | 10/05/2019 19:28:53 | 10/05/2019 20:37:58 | 3.60 |
| 2 | 10/05/2019 09:45:50 | 11/05/2019 10:10:38 | 11/05/2019 10:57:41 | 4.38 |
| 3 | 11/05/2019 12:52:38 | 11/05/2019 13:06:05 | 11/05/2019 14:28:10 | 3.85 |
| 4 | 11/05/2019 17:49:55 | 11/05/2010 17:49:55 | 11/05/2019 18:57:55 | 3.54 |
| 5 | 12/05/2019 09:55:20 | 12/05/2019 10:01:06 | 11/05/2019 10:57:10 | 3.78 |
| 6 | 12/05/2019 17:48:15 | 12/05/2019 17:56:38 | 12/05/2019 18:47:46 | 3.74 |
| 7 | 13/05/2019 10:12:49 | 13/05/2019 10:13:39 | 13/05/2019 10:37:43 | 4.49 |
| 8 | 13/05/2019 12:21:33 | 13/05/2019 12:21:33 | 13/05/2019 13:07:01 | 4.52 |
| 9 | 13/05/2019 17:54:19 | 13/05/2019 17:54:19 | 13/05/2019 18:47:07 | 4.47 |
| 10 | 13/05/2019 20:40:29 | 13/05/2019 20:48:15 | 13/05/2019 22:17:30 | 4.29 |

It is important to note that some tests in Table 5.1 and Table 5.2 were started after the time that was predicted. Due to this, the heating will start and end after the set time. During the first trial, the system has no approximated model yet and therefore estimates one degree centigrade per hour. This gives the estimated duration of over 3 hours. In subsequent trials, the algorithm will know an approximated model for the heating.

The data that should be examined is the estimated- and the actual duration of the heating process and also the difference between the estimated- and actual duration. This is due to the fact that the algorithm estimates how many degrees centigrade per hour the facility is being heated.

Table 5.3: Estimated heating- and actual heating duration

| Trial | Target [C°] | Estimated duration | Actual duration | Difference (hh:mm:ss) | Difference (%) |
|---|---|---|---|---|---|
| 1 | 21 | 03:56:24 | 01:09:05 | -02:47:19 | -70.78 |
| 2 | 21 | 00:54:10 | 00:47:03 | -00:07:07 | -13.14 |
| 3 | 23 | 00:57:22 | 01:22:05 | 00:24:43 | 43.09 |
| 4 | 23 | 00:40:05 | 01:08:00 | 00:27:55 | 69.64 |
| 5 | 22 | 01:04:40 | 00:56:04 | -00:08:36 | 13.30 |
| 6 | 23 | 00:41:45 | 00:51:08 | 00:09:23 | 22.48 |
| 7 | 21 | 00:37:11 | 00:24:04 | -00:13:07 | -35.28 |
| 8 | 22 | 00:38:27 | 00:45:28 | 00:07:01 | 18.25 |
| 9 | 23 | 00:35:41 | 00:52:48 | 00:17:07 | 47.97 |
| 10 | 23 | 00:49:31 | 01:29:15 | 00:39:44 | 80.24 |

The Estimated duration was calculated as the difference of the target time and and the estimated start time. Through this, the duration that the algorithm

estimated to heat up the house is calculated. The actual duration was the difference between the actual start time and the actual end time. This gives the actual time that the heating process took, regardless when it was started. This was done as some tests predicted an earlier time than when the trial was started. This can be seen in Table 5.1 and Table 5.2. The percentage difference was calculated with (5.1).

$$\frac{Duration_{actual}}{Duration_{estimated}} - 1 \tag{5.1}$$

This gives an approximation on how exact the algorithm performed for the time it took to heat up the test environment. Furthermore, this can be compared to the thermal properties of the test environment to draw conclusions about the power of the heating element in a given facility. In order to evaluate this, the temperature was graphed over time in minutes.

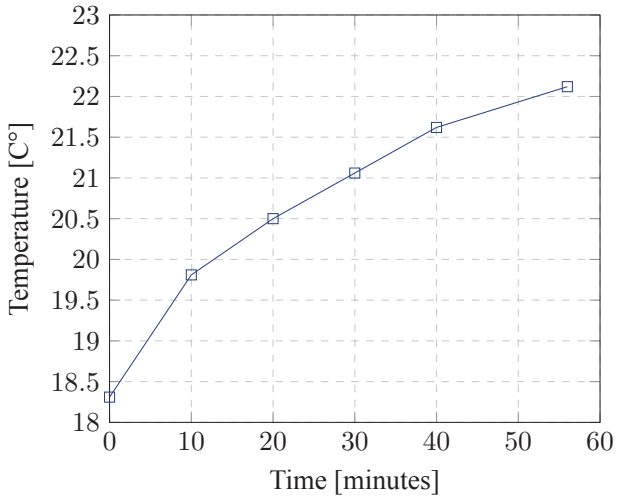



Figure 5.1: Temperature over time up to 22 degrees C° from Trial 5

The temperature curve in Figure 5.1 was found to have a part (between 10 and 40 minutes) which was rather linear. However another problem that was to be analyzed are the large spikes in error, which can be seen in Table 5.3 under the difference column, that were observed when heating to 23 degrees centigrade. In order to analyze this, the temperature over time for heating was

graphed around this temperature.

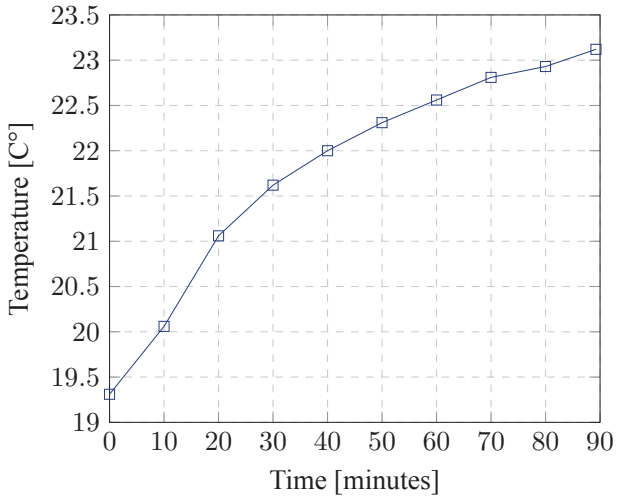



Figure 5.2: Heating over time up to 23 degrees C° from Trial 10

Figure 5.2 shows the decreasing slope in the temperature, which increases the time it takes to reach the specified temperature.

Another aspect to note is how the coefficient changes with each trial. This can be used in order to draw conclusion about how the model can handle errors and adjust based on new data.

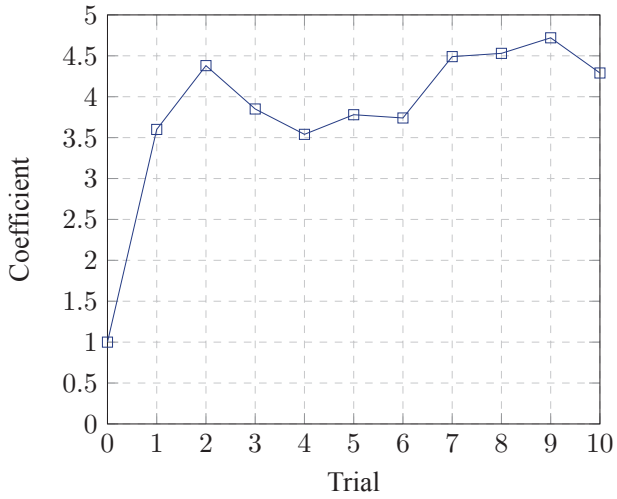



Figure 5.3: Coefficient average for each trial

Figure 5.3 shows how the coefficient average changes for each trial. The coefficient oscillates between approximately 4.5 and 3.5 in the given data.

# 6 Discussion

## 6.1 Evaluation of the linear model

As can be seen in Figure 5.1 and Figure 5.2, the model doesn't quite seem linear. In the beginning of the heating process, it seems to be rapid and in the end section of the graphs, it seems to slow down. However, in both graphs there is a linear relation between different points of time. For example, in Figure 5.1, a linear relation can be assumed between the span of 10 and 40 minutes.

In the beginning of a heating process, the radiator will use its maximum power to heat itself up, which is a rapid increase. When the radiator is at max capacity, a linear heating process will begin. Depending on the environment, the heating process will slow down once the loss of energy begins to slowly overtake the gain of energy.

The testing facility, explained more in dept in 4.7.2, was a basement in a house. This space was very small, so the radiator was set to as low power as possible to test temperature heating over time. This testing facility also lacked any isolation. It only takes a short amount of time to heat up a small space. However, due to the lack of isolation and loss of heat, the low power of the radiator made it impossible for the temperature to reach over a certain limit.

The result is, that the heating will be quicker in the beginning while the radiator is going towards maximum power. The heating will be linear while the radiator is at its maximum power, and the heating will slow down when the room begins to approach its temperature threshold.

In a house however, there is isolation and radiators which are set to increase temperature as fast as possible, with maximum power. There is less loss of heat, and if there is a loss of heat it will be however, due to the lack of isolation and loss of heat, the low power of the radiator made it impossible for the

temperature to reach over a certain limit. temperatures than 22-23 degrees centigrade, which seems to be the threshold of our testing facility.

### 6.1.1 Accuracy

The accuracy of the model was deemed acceptable as long as the apparent temperature threshold wasn't reached. In the tests, the temperature threshold seemed to lie somewhere above 22 degrees centigrade. If we study Table 5.2 it can be seen, that when the target temperature was below 23 degrees centigrade, the percentage difference was around 15-20% too early or too late. If we instead look at the target temperature set to 23 degrees centigrade, the model was highly inaccurate.

The reasons behind the lack of accuracy was discussed in 6.1. However, in a scenario where the heating process is longer and the coefficient lay around 1-2, this model should work fine with a somewhat linear behaviour.

## 6.2 Explaining the changes to the coefficient

It can be observed in Figure 5.3 that the coefficient oscillates between approximately 4.5 and 3.5 degrees centigrade per hour. In order to explain this, Table 5.2 has to be taken into account. Firstly, the large jump in the beginning occurs because, before the first trial is complete, the standard model of one degree centigrade per hour will be used. The model is then setup with the first learned coefficient. During the second trial, the new coefficient is used. According to Table 5.2, this gave a difference of approximately 7 minutes. Therefore, it can be argued that the model has successfully learned to estimate the time required in order to heat to 21 degrees centigrade.

During Trial 3 and Trial 4, the facility was heated to 23 degrees centigrade. Here it can be seen that the algorithm performs worse. Due to this, the coefficient decreases in order to counteract the fact that the algorithm performed too slow during the heating process. The facility is then heated to 23 degrees again. The coefficient then decreases again in order to compensate for the estimate that was not strict enough.

For the next trial when the facility is heated to 22 degrees centigrade, the estimate is too generous as the algorithm has reduced the coefficient for when it was wrong with heating to 23 degrees centigrade. Due to this, the algorithm

compensates with increasing the coefficient.

In Trial 7, the facility is heated to 21 degrees centigrade. The algorithm is quite a bit off for this trial, percentage wise. Therefore, the algorithm raises the coefficient. The curve seen in Figure 5.3 represents the algorithm attempting to compensate for a too strict or too generous estimation. If the system were heated to the same temperature repeatedly, the algorithm would aim to get a more exact estimation and reduce the effect of errors. Due to the properties discussed earlier in 6.1, the estimation might not be accurate. However, in a more accurate environment, this approach could perform better and have a better estimation range. Perhaps if the model would have been trained for the higher temperature from the beginning, the algorithm would have performed better during all the tests.

From this, it was obvious that the model had some trouble adjusting to untrained temperatures. One possible explanation for this is the reason discussed in 6.1 and that we operated at a range of temperatures where the linear section of the heating curve ended. Therefore, when we attempted to heat to 23 degrees, we operated at a temperature range that the algorithm was not used to. It can be inferred that the algorithm would have performed better if a radiator with a higher output power had been used in the tests. In theory, if the relationship is linear, and the coefficient has a similar magnitude, the algorithm should be able to adjust relatively quickly. If the algorithm has not yet learned a model for a given temperature range, the adjustment should be done after the facility has been heated once. The difference is that if the algorithm has already learned a model for a temperature range, new and different data will be treated as noise. When enough new data is gathered, the average calculation will treat the original data as noise.

## 6.3 Possible changes to the model

It can be observed, that in some cases, the linear model does not suffice. While for normal use, the model should suffice, in some cases the model can be modified to give a more accurate prediction.

### 6.3.1 Using a model of higher order

Due to the fact that the radiator was overwhelmed, the system is not completely linear. As this is not something that should become a problem in a practical application, a linear approximation should suffice. However, to

counteract the symptoms shown in the given results, a second order model, for example, could be used. If implemented, the input to the least squares method would need to be changed, as well as the ”HeatingCoefficient” class.

The system might more accurately adjust to the properties of the test system, and the model could instead be in the form of (6.1).

$$y_i = m + k_1 x_i + k_2 x_i^2 \tag{6.1}$$

It follows that, perhaps, this model would give a benefit in the case of a building with little to no isolation and a weak radiator. In the general case of good isolation and a powerful radiator, this should not become a problem as the temperature would probably not reach the point where the temperature planes out. It can be investigated how much of a practical benefit a model of higher order would give.

Perhaps another input parameter can be used. This other parameter could be the power of the radiators used. Through this, the amount of power that the room is heated with is taken into account. This could help to make the model more exact. However, this has to be further investigated.

### 6.3.2 Using an exponential model

It can be seen in Figure 2.1 as well as the presented results that the model can be written in the form of:

$$k(1 - e^{-\frac{t}{T}}) \tag{6.2}$$

Due to this, an exponential model could be introduced in order to, more exactly, estimate the time taken to heat the facility. This would allow the model to accurately predict the time it would take to heat the facility outside of the acceptable window for a linear approximation.

## 6.4 Potential problems that can cause failure

There are sources of error that can cause the algorithm to fail to work properly. One example of this is when a user installs other- or extra radiators in their house. The algorithm will in this case predict a too long heating time as it is trained for a lower total output effect. Given time, the model should be able to adjust as the average will adjust. This can however also be a weakness for a

model that is trained over a large number of trials. The model would take the same amount of training that it already has to even begin to properly adjust. This can be seen as a very inefficient effect.

For this, a remedy can be proposed. This is that the model checks whether a very large difference in coefficient exists when the model is updated. The model then sets the coefficient to the new one and sets the counter for the number of values in the average to ”1”. This effectively sets the sum to to the new value, and returns the ”HeatingCoefficient” class to a newly initialized state.

Alternatively, the user could be given an option to reset the stored models. This would imply that the stored models are simply removed, causing the algorithm to default to the standard one degree centigrade per hour when the next heating is started. However the reset is performed, the algorithm will need to perform a heating which will be wrong in order to estimate a new model to use.

Another possible error that can occur over time is due to the data types used to represent the amount of values in the average. In very extreme cases, this counter can overflow. The probability of this for normal use was evaluated.

According to Microsoft’s documentation for the .NET framework, an int is represented as a signed 32 bit integer [8]. From this definition i follows that:

$$\frac{2^{31}-1}{365.25*24*60^2} \approx 68.05 \tag{6.3}$$

If the algorithm updates once per second, it will take approximately 68 years in order for the counter to overflow. Therefore regarding regular use, it is very improbable that the algorithm will overflow. It is however worth noting that there exists a possibility for an overflow. However, in the rare case that this does occur, the value will become negative and eventually zero. While a negative value will generate a problematic result, dividing by zero will cause the system to become unstable or even crash depending on the implementation as in regular arithmetic a division by zero is undefined.

The previously mentioned problem can easily be handled using a data type that can handle larger numbers or by handling the overflow by setting the number of values to ”1” in time.

When the value of the counter is one, during the next run in the given implementation, it will take the previous average and use this as a value for the average calculations with the next estimated coefficients.

Given, for any reason, the determinant of the of the features matrix in the least squares method becomes zero, the matrix does not have an inverse. Therefore the least square method will not be performed.

When inverting a matrix, it's determinant can not be zero. As explained in (4.17), calculating the inverse requires division with the determinant. The features matrix is defined as the input variables for the least squares, which can be seen in Listing 6.1:

**Listing 6.1: Example class**

```
1 Matrix weights= (features.Transpose().Multiply(features)).
      Inverse().Multiply(features.Transpose()).Multiply(
      targets);
```

The system will throw an exception in the given implementation if this occurs. This will need to be handled through either another method to perform linear regression, such as a gradient descent analysis or by simply canceling the update to the model and skipping this part.

## 6.5 Possible applications of the algorithm

Several systems can be built using this algorithm. The first is the previously mentioned scheduling. However the algorithm can even be used to analyse the thermal properties of a house. This is due to the fact that the algorithm estimates how fast the house can be heated up given a set effect. Through this, conclusions can be drawn about how the heating in a given house should be setup. Furthermore, this can be used to determine if the heating in the house is above or below average.

This can also be used to collect data to perform further analysis on. This analysis can include the estimation about how the isolation in a house is. As this algorithm is intended for a more intelligent scheduling for the application of heating a house, this would be the most adequate use for this algorithm.

## 6.6 Evaluation of energy optimisation using the algorithm

One idea could be to minimize the energy consumption. As the current system uses ON/OFF control, the radiator will always use the maximum amount of energy possible. Therefore, to optimize the energy consumption, the time that the radiator is turned on would need to be minimized. This is the aim of this algorithm. When trained well, this will hopefully be achieved.

However, if one would use a different type of controller, more types of optimizations can be done. For example if PWM control would be used, the radiator would only be turned on a certain time during a duty cycle to reduce the effect. The practicality of this would however need to be examined as it would take time for the radiator to start cooling down. This could give that the duty cycle would need to be so long that it would not be practical to add PWM control. If this were an option, a PI controller could be used to control the heating.

If a PWM approach is used, the model will need to be updated to include the equation that the controller would use. Furthermore, the wear of components such as the relay, currently used to hold the power, must be considered. Perhaps, in this case, a solid state switch for the fuses would first need to be introduced.

## 6.7 Ethical aspects

During the project, a copy of Tempiro's database was analyzed. This presented one problem. The database contained customer information. In order to protect the customers privacy, the data was anonymized. All names, email-addresses, physical addresses, user ID- and locations where removed. The user ID was replaced with another unique number. This resulted in a anonymous data that could not be connected to Tempiro's customers in any fashion, while still retaining the relationships of the data. Through this, the privacy of the customers was ensured.

Furthermore, in order to contact the API, an IP address was used to connect to Tempiro's server. It was requested that this IP address remain a secret. Therefore this was removed from the source code in the implementation given in Appendix A. As the access tokens used by Tempiro's server are long-lived, it is also not included in the source code. This will protect access to Tempiros

server and the privacy of the authors.

# 7 Future work

## 7.1 Integration

As of now, the algorithm is a standalone software, running outside of Tempiro's system. This software is to be integrated into Tempiro's system and to be written into the Temprio mobile application. If integrated, the algorithm will be easier to use for the company's clients and, if needed, easier to manage if Tempiro someday wants to change or update the algorithm.

Currently, there are two constraints as to why integration is not possible in the scope of this project. The first is the time constraints on this project. This will prevent from an integration into Tempiros server to be possible at this point. Secondly, Tempiro's server is not setup to handle the algorithm in its current form. The manner in which their current back-end solution handles jobs will need to be extended before an integration of the algorithm can be done.

Furthermore, as integration of the algorithm into Tempiro's system wasn't possible at this time, it was decided, together with Tempiro, that testing through pilot customers also couldn't be done in the scope of the project. This, however, can be done by Tempiro at a later stage, when integration has been implemented into their current software.

## 7.2 Correlation

When integrated, Tempiro will be able to acquire alot of data from their users. Tempiro can look at that data to find some correlations between different outside temperatures and the algorithm. Instead of doing a test run every time a new index is triggered or a new client wants to use the algorithm, the coefficient can be approximated and the service can be used immediately. This cannot be done without data from many different situations.

For instance, Tempiro notices, that overall, the coefficient between two specific vector indices, which are described in section 4.5, is approximately 15 percent. A client could have one well trained vector index, where the algorithm triggers an index beside this one. For now the index must learn what coefficient should be there. With data however, Tempiro could use the working index with the 15 percent increase or decrease. The learning algorithm will correct the hopefully small error over time.

## 7.3 Using a neural network

The developed algorithm aims to tune its own coefficient in order to increase its own accuracy to predict the time it will take to heat a facility. However, the same result could be achieved with a neural network. The error would then be corrected using backward propagation in order to change its weights. The value of this approach has to be investigated. Furthermore, several extensive changes to the test implementation and the given algorithm needs to be done in order to allow this.

# 8 References

[1] A. Ghatak, Machine Learning with R, Singapore: Springer eBooks, 2017, s. 79-113, [E-book] (Last accessed 2019-04-04)

[2] A. Chamberlain, ”The Linear Algebra View of Least-Squares Regression”, medium.com, december 2016 [Online] Available: https://medium.com/@andrew.chamberlain/the-linear-algebra-view-of-least-squares-regression-f67044b7f39b (Last accessed 2019-05-19)

[3] “Standard ECMA-404,” Standard ECMA-404 The JSON Data Interchange Syntax. [Online]. Available: https://www.ecma-international.org/publications/standards/Ecma-404.htm. (Last accessed 2019-05-21).

[4] “JavaScriptSerializer Class (System.Web.Script.Serialization).” [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.web.script.serialization.javascriptserializer. (Last accessed 2019-05-21).

[5] M. Barr, “Introduction to Pulse Width Modulation (PWM),” Barr Group, 01-Sep-2001. [Online]. Available: https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation. (Last accessed 2019-05-21).

[6] ”Inverse of a Matrix using Minors, Cofactors and Adjugate”, MathIsFun, [Online] Available: https://www.mathsisfun.com/algebra/matrix-inverse-minors-cofactors-adjugate.html?fbclid=IwAR2TWgb38xwC9snI98FQMab4LXpQw9UnNdTyt0-RjpOu6ZfFWdViI0v2cqs (Last accessed 2019-03-27)

[7] ”Wolfram|Alpha” https://www.wolframalpha.com/ [Online] (Last accessed 2019-05-19)

[8] B. Wagner, “int - C# Reference”, int (C # Reference), [Online] Available: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/key

words/int (Last accessed 2019-05-11).

# 9 Appendix

## A Code

**Listing 9.1: Matrix.cs**

```
1  namespace Math
2  {
3      public class Matrix
4      {
5          private double[,] data;
6          public Matrix(double[,] init) => data = init;
7          public Matrix(int rows, int cols) => data = new
               double[rows, cols];
8          public void put(int row, int col, double val)
9          {
10             data[row, col] = val;
11         }
12         public Matrix Multiply(Matrix other)
13         {
14             if (data.GetLength(1) == other.data.GetLength
                   (0))
15             {
16                 double[,] c = new double[data.GetLength(0),
                       other.data.GetLength(1)];
17                 for (int i = 0; i < c.GetLength(0); i++)
18                 {
19                     for (int j = 0; j < c.GetLength(1); j
                           ++)
20                     {
21                         c[i, j] = 0;
22                         for (int k = 0; k < data.GetLength
                               (1); k++) // OR k<b.GetLength
                               (0)
23                             c[i, j] = c[i, j] + data[i, k]
                                   * other.data[k, j];
24                     }
25                 }
26                 return new Matrix(c);
27             }
```

```
28              throw new MatrixDimensionException("Unable to
                    multiply two Matrixes where the collumn
                    count of first non equal to row count of
                    second");
29          }
30          public Matrix Transpose()
31          {
32              double[,] new_matrix = new double[data.
                    GetLength(1), data.GetLength(0)];
33
34              for (int i = 0; i < data.GetLength(0); ++i)
35              {
36                  for (int j = 0; j < data.GetLength(1); ++j)
37                  {
38                      new_matrix[j, i] = data[i, j];
39                  }
40              }
41              return new Matrix(new_matrix);
42          }
43          public override string ToString()
44          {
45              string return_str = "";
46              for (int i = 0; i < data.GetLength(0); ++i)
47              {
48                  for (int j = 0; j < data.GetLength(1); ++j)
49                  {
50                      return_str += data[i, j];
51                      return_str += " ";
52                  }
53                  return_str += '\n';
54              }
55              return return_str;
56          }
57          private double Determinant()
58          {
59              return DeterminantRekur(data);
60          }
61          private double DeterminantRekur(double[,] values)
62          {
63              if (values.GetLength(0) == values.GetLength(1))
64              {
65                  if (values.GetLength(0) == 1)
66                  {
67                      return values[0, 0];
68                  }
69                  double det = 0;
70                  if (values.GetLength(0) == 2)
71                  {
72                      return (values[0, 0] * values[1, 1]) -
                            (values[0, 1] * values[1, 0]);
```

```
73                  }
74                  for (int i = 0; i < values.GetLength(0); ++
                        i)
75                  {
76                      if (i % 2 == 0)
77                      {
78                          det += values[0, i] *
                                DeterminantRekur(Decompose(
                                values, 0, i));
79                      }
80                      else
81                      {
82                          det -= values[0, i] *
                                DeterminantRekur(Decompose(
                                values, 0, i));
83                      }
84                  }
85                  return det;
86              }
87              throw new MatrixDimensionException("Unable to
                    find Determimant for non-square Matrix");
88          }
89          private double[,] Decompose(double[,] values, int
                row, int col)
90          {
91              double[,] new_dat1 = new double[values.
                    GetLength(0) - 1, values.GetLength(1)];
92              double[,] new_dat = new double[values.GetLength
                    (0) - 1, values.GetLength(1) - 1];
93
94              //remove row
95              int rowindex = 0;
96              int newrowindex = 0;
97              while (rowindex < values.GetLength(0) &&
                    newrowindex < new_dat1.GetLength(0))
98              {
99                  if (rowindex == row)
100                     if (++rowindex == values.GetLength(0))
101                         break;
102                 for (int i = 0; i < values.GetLength(1); ++
                        i)
103                     new_dat1[newrowindex, i] = values[
                            rowindex, i];
104                 ++rowindex;
105                 ++newrowindex;
106             }
107             //remove col
108             int colindex = 0;
109             int newcolindex = 0;
110             while (colindex < new_dat1.GetLength(1) &&
```

```
                newcolindex < new_dat.GetLength(1))
111             {
112                 if (colindex == col)
113                     if (++colindex == new_dat1.GetLength(1)
                            )
114                         break;
115                 for (int i = 0; i < new_dat1.GetLength(0);
                        ++i)
116                 {
117                     new_dat[i, newcolindex] = new_dat1[i,
                            colindex];
118                 }
119                 ++colindex;
120                 ++newcolindex;
121             }
122             return new_dat;
123         }
124         private Matrix Minors()
125         {
126             double[,] minors = new double[data.GetLength(0)
                    , data.GetLength(1)];
127             for (int row = 0; row < data.GetLength(0); ++
                    row)
128                 for (int col = 0; col < data.GetLength(1);
                        ++col)
129                 {
130                     minors[row, col] = DeterminantRekur(
                            Decompose(data, row, col));
131                 }
132             return new Matrix(minors);
133         }
134         private Matrix Cofactors()
135         {
136             double[,] cofactors = Minors().data;
137             for (int row = 0; row < data.GetLength(0); ++
                    row)
138             {
139                 for (int col = 0; col < data.GetLength(1);
                        ++col)
140                 {
141                     //Om raden är udda OCH kolonnen är jämn
                             ELLER om raden är jämn OCH
                            kolonnen är udda
142                     if ((row % 2 != 0 && col % 2 == 0) ||
                            ((row % 2 == 0 && col % 2 != 0)))
143                     {
144                         cofactors[row, col] = -cofactors[
                                row, col];
145                     }
146                 }
```

```
147             }
148             return new Matrix(cofactors);
149         }
150         private Matrix Adjugate()
151         {
152             return Cofactors().Transpose();
153         }
154         public Matrix Inverse()
155         {
156             if (data.GetLength(0) == data.GetLength(1))
157             {
158                 if (Determinant() != 0)
159                 {
160                     if (data.GetLength(0) == 2)
161                     {
162                         return easyInverse();
163                     }
164                     if (data.GetLength(0) == 1)
165                     {
166                         double[,] Res = new double[1, 1];
167                         Res[0, 0] = 1 / data[0, 0];
168                         return new Matrix(Res);
169                     }
170                     else
171                     {
172                         double[,] inverse = new double[data
                                .GetLength(0), data.GetLength
                                (1)];
173                         double coeff = 1 / Determinant();
174                         double[,] adjugate = Adjugate().
                                data;
175 
176                         for (int row = 0; row < data.
                                GetLength(0); ++row)
177                             for (int col = 0; col < data.
                                    GetLength(1); ++col)
178                                 inverse[row, col] = coeff *
                                        adjugate[row, col];
179                         return new Matrix(inverse);
180                     }
181                 }
182                 throw new MatrixLogicException("The
                        Determinant of the Matrix was 0, it is
                        not invertible");
183             }
184             throw new MatrixDimensionException("No inverse
                    for a non-sqaure Matrix");
185         }
186         public double get(int row, int col)
187         {
```

```
188             if (row < data.GetLength(0) && col < data.
                    GetLength(1))
189                 return data[row, col];
190             throw new MatrixDimensionException("The indexes
                     you wanted where out of bounds");
191         }
192         private Matrix easyInverse()
193         {
194             double a1 = data[0, 0];
195             double a2 = data[0, 1];
196             double b1 = data[1, 0];
197             double b2 = data[1, 1];
198 
199             double denominator = a1 * b2 - b1 * a2;
200             double coefficient = 1 / denominator;
201 
202             double[,] temp = new double[2, 2];
203             temp[0, 0] = b2 * coefficient;
204             temp[1, 1] = a1 * coefficient;
205             temp[1, 0] = -b1 * coefficient;
206             temp[0, 1] = -a2 * coefficient;
207 
208             return new Matrix(temp);
209         }
210 
211         public Matrix Add(Matrix other)
212         {
213             if ((data.GetLength(0) == other.data.GetLength
                    (0) && (data.GetLength(1) == other.data.
                    GetLength(1))))
214             {
215                 double[,] result = new double[data.
                        GetLength(0), data.GetLength(1)];
216                 for (int i = 0; i < data.GetLength(0); ++i)
217                 {
218                     for (int j = 0; j < data.GetLength(1);
                            ++j)
219                     {
220                         result[i, j] = data[i, j] + other.
                                data[i, j];
221                     }
222                 }
223                 return new Matrix(result);
224             }
225             throw new MatrixDimensionException("Unable to
                    add two matrixes of different sizes");
226         }
227         public Matrix Subtract(Matrix other)
228         {
229             if ((data.GetLength(0) == other.data.GetLength
```

```
                (0) && (data.GetLength(1) == other.data.
                GetLength(1))))
230             {
231                 double[,] result = new double[data.
                        GetLength(0), data.GetLength(1)];
232                 for (int i = 0; i < data.GetLength(0); ++i)
233                 {
234                     for (int j = 0; j < data.GetLength(1);
                            ++j)
235                     {
236                         result[i, j] = data[i, j] - other.
                                data[i, j];
237                     }
238                 }
239                 return new Matrix(result);
240             }
241             throw new MatrixDimensionException("Unable to
                    add two matrixes of different sizes");
242         }
243         public Matrix SquareAllSeparate()
244         {
245             double[,] result = new double[data.GetLength(0)
                    , data.GetLength(1)];
246             for (int i = 0; i < data.GetLength(0); ++i)
247             {
248                 for (int j = 0; j < data.GetLength(1); ++j)
249                 {
250                     result[i, j] = data[i, j] * data[i, j];
251                 }
252             }
253             return new Matrix(result);
254         }
255
256         public Matrix Multiply(double d)
257         {
258             double[,] result = new double[data.GetLength(0)
                    , data.GetLength(1)];
259             for (int i = 0; i < data.GetLength(0); ++i)
260             {
261                 for (int j = 0; j < data.GetLength(1); ++j)
262                 {
263                     result[i, j] = d * data[i, j];
264                 }
265             }
266             return new Matrix(result);
267         }
268         public int numberRows()
269         {
270             return data.GetLength(0);
271         }
```

```
272         public int numberCols()
273         {
274             return data.GetLength(1);
275         }
276     }
277 }
```

**Listing 9.2: Table.cs**

```
1  namespace Math
2  {
3      public class Table
4      {
5          private string [] lables;
6          private double [,] data;
7          public Table(int rows, int cols)
8          {
9              this.lables = new string[cols];
10             for(int i =0; i<cols; ++i)
11             {
12                 lables[i] = i+"";
13             }
14
15             this.data = new double [rows,cols];
16         }
17         public void setLable(int col, string value)
18         {
19             if(col<lables.GetLength(0)&&col>=0)
20             {
21                 lables[col] = value;
22             }
23             else
24             {
25                 throw new TableException("Unable to set the
                       Lable of Collumn: " + col + " It does
                      not exist");
26             }
27         }
28         public int LableIndex(string Lable)
29         {
30             for(int i = 0; i<lables.GetLength(0); ++i)
31             {
32                 if(lables[i].ToLower().Equals(Lable.ToLower
                       ()))
33                 {
34                     return i;
35                 }
36             }
37             throw new TableException("The lable: " + Lable
                   + " does not exits");
38         }
39         public double get (string colLable, int rowNumb)
40         {
41             return get(rowNumb, LableIndex(colLable));
42         }
43         public double get(int row, int col)
44         {
```

```
45              if(row<data.GetLength(0)&&col<data.GetLength(1)
                    &&row>=0&&col>=0)
46              {
47                  return data[row,col];
48              }
49              throw new TableException("Attempting to get
                    data outside of table size");
50          }
51          public int numberRows()
52          {
53              return data.GetLength(0);
54          }
55          public int numberCols()
56          {
57              return data.GetLength( 1);
58          }
59
60
61          public void put(int row, int col, double val)
62          {
63
64              if(row<data.GetLength(0)&&col<data.GetLength(1)
                    &&row>=0&&col>=0)
65              {
66                  data[row,col]= val;
67              }
68              else
69              {
70                  throw new TableException("Attempting to put
                        data outside of table size, row, col
                       was " + row + " , " + col);
71              }
72          }
73          public void put(string lable, int row, double val)
74          {
75              put(row,LableIndex(lable), val);
76          }
77          public override string ToString()
78          {
79              string result = "";
80              for(int i = 0;i<lables.GetLength(0); ++i)
81              {
82                  result += lables[i] + " | ";
83              }
84              result += "\n";
85
86              for(int i =0; i<data.GetLength(0); ++i)
87              {
88                  for(int j =0; j <data.GetLength(1); ++j)
89                  {
```

```
90                      result += data[i,j] + " | ";
91                  }
92                  result+="\n";
93              }
94              return result;
95          }
96          public void AddRow(double[] vals)
97          {
98              if(numberCols()==vals.GetLength(0))
99              {
100                 double[,] NewData = new double[numberRows()
                        +1,numberCols()];
101                 for(int i = 0; i<numberRows(); ++i)
102                 {
103                     for(int j = 0; j<numberCols(); ++j)
104                     {
105                         NewData[i, j] = data[i, j];
106                     }
107                 }
108                 for (int j = 0; j < numberCols(); ++j)
109                 {
110                     NewData[numberRows(), j] = vals[j];
111                 }
112                 data = NewData;
113                 return;
114             }
115             throw new TableException("Row to add has
                    different amount of collumns");
116         }
117         public Table connectTables( Table second, string
                col) {
118             int nbrCols = numberCols() + second.numberCols
                    () - 1;
119             int nbrRows = numberRows();
120             Table temp = new Table(nbrRows, nbrCols);
121             temp.setLable(0, col);
122
123             //Namnge kolonnerna
124             for(int i = 0; i <numberCols(); i++) {
125                 if(!lables[i].Equals(col)) {
126                     temp.setLable(i, lables[i]);
127                 }
128             }
129
130             int Track = numberCols();
131             for(int i = 0; i < second.numberCols(); i++) {
132                 if(!second.lables[i].Equals(col)) {
133                     temp.setLable(Track, second.lables[i]);
134                     ++Track;
```

```
135                 }
136             }
137
138             //Fyll raderna
139             for(int i = 0; i < temp.numberRows() ; i++) {
140                 for(int k = 0; k < temp.numberCols(); k++)
                        {
141                     int labelIndex = -1;
142                     bool firstFlag = false;
143                     bool secondFlag = false;
144                     for(int j = 0; j < numberCols(); j++) {
145                         if(lables[j].Equals(temp.lables[k])
                                ) {
146                             firstFlag = true;
147                             labelIndex = j;
148                         }
149                     }
150                     if(firstFlag) {
151                         temp.put(i, k, get(i, labelIndex));
152                     } else {
153                         for(int j = 0; j < second.
                                numberCols(); j++) {
154                             if(second.lables[j].Equals(temp
                                    .lables[k])) {
155                                 secondFlag = true;
156                                 labelIndex = j;
157                             }
158                         }
159                         if(secondFlag) {
160                             temp.put(i, k, second.get(i,
                                    labelIndex));
161                         }
162                     }
163                 }
164             }
165             return temp;
166         }
167     }
168 }
```

**Listing 9.3: MultiVariableLinnearRegression.cs**

```
1  namespace Math
2  {
3      namespace LinnearRegression
4      {
5          public class MultiVariableLinnearRegression
6          {
7              private static Matrix hypothesis(Matrix Weights
                   , Matrix features)
8              {
9                  /**
10                 weights -  1 collumn andx x rows for the
                       weights
11                 features - 1 row per measurement, 1 col per
                        feasture
12                 retrurn - 1 col and 1 row per prediction
13                  */
14                 return features.Multiply(Weights);
15             }
16             private static double cost(Matrix features,
                   Matrix Weights, Matrix targets)
17             {
18                 /**
19                 weights -  1 collumn andx x rows for the
                       weights
20                 features - 1 row per measurement, 1 col per
                        feasture
21                 target = 1 col, 1 row per target
22                 */
23                 double n = targets.numberRows();
24
25                 Matrix predictions = hypothesis(Weights,
                       features);
26
27                 Matrix A = (predictions.Subtract(targets));
28                 Matrix SquareError = A.SquareAllSeparate();
29
30                 double sum = 0;
31                 for (int i = 0; i < SquareError.numberRows
                       (); ++i)
32                 {
33                     sum += SquareError.get(i, 0);
34                 }
35
36                 return (1.0 / (2 * n)) * sum;
37             }
38             public static Model LeastSquare(Table data, int
                    nbrFeatures)
39             {
40                 Matrix features = new Matrix(data.
```

```
                    numberRows(), nbrFeatures + 1);
41                  Matrix targets = new Matrix(data.numberRows
                        (), 1);
42                  for (int i = 0; i < data.numberRows(); ++i)
43                  {
44                      targets.put(i, 0, data.get("Y", i));
45                      for (int j = 0; j < nbrFeatures; ++j)
46                      {
47                          features.put(i, j, data.get("
                                Feature" + j, i));
48                      }
49                      features.put(i, nbrFeatures, 1);
50
51                  }
52                  //System.Console.WriteLine(features.
                        ToString());
53                  //System.Console.WriteLine(targets.ToString
                        ());
54                  Matrix output = new Matrix(nbrFeatures, 1);
55                  Matrix weights = (features.Transpose().
                        Multiply(features)).Inverse().Multiply(
                        features.Transpose()).Multiply(targets)
                        ;
56                  for (int i = 0; i < nbrFeatures; ++i)
57                  {
58                      output.put(i, 0, weights.get(i, 0));
59                  }
60                  double bias = weights.get(nbrFeatures, 0);
61                  return new Model(new Functions.
                        MultiVariableFirstOrderPolinomial(
                        output, bias), cost(features, weights,
                        targets));
62              }
63          }
64      }
65  }
```

**Listing 9.4: Model.cs**

```
1  using Math.Functions;
2  using System;
3  namespace Math
4  {
5      public class Model
6      {
7          private MultiVariableFirstOrderPolinomial Function;
8          private double Cost;
9
10         public Model(MultiVariableFirstOrderPolinomial func
               , double Cost)
11         {
12             this.Function = func;
13             this.Cost = Cost;
14         }
15
16         public double predict(Matrix values)
17         {
18             return Function.solve(values);
19         }
20         public override string ToString()
21         {
22             return Function.ToString() + "  Cost: " + Cost;
23         }
24         public MultiVariableFirstOrderPolinomial
               GetFunction()
25         {
26             return Function;
27         }
28     }
29 }
```

**Listing 9.5: Function.cs**

```
1  namespace Math
2  {
3      namespace Functions
4      {
5
6          public class MultiVariableFirstOrderPolinomial
7          {
8              Matrix Coeff;
9              double Bias;
10             public MultiVariableFirstOrderPolinomial(Matrix
                   coeff, double bias)
11             {
12                 this.Bias = bias;
13                 this.Coeff = coeff;
14             }
15             public override string ToString()
16             {
17                 string result = "y=";
18                 for(int i =0; i<Coeff.numberRows(); ++i)
19                 {
20                     result += (Coeff.get(i,0) + "x" + i + "
                           +");
21                 }
22                 result +=Bias;
23                 return result;
24             }
25              public double solve(Matrix values)
26              {
27                  return values.Multiply(Coeff).get(0,0) +
                        Bias ;
28              }
29             public Matrix GetCoeff()
30             {
31                 return Coeff;
32             }
33         }
34     }
35 }
```

**Listing 9.6: Api.cs**

```
1  using Math;
2  using System;
3  using System.Net;
4  using System.IO;
5  using System.Collections.Generic;
6  using System.Web.Script.Serialization;
7
8  public class APIConnector
9  {
10     string AuthToken;
11     static string BaseUrl = @"URL";
12     static string DevLink = "/api/Devices";
13     static string ValLink = "/api/Values";
14     static string DevconfLink = "/api/DeviceConfigurations"
           ;
15     static string switchLink = "/api/Switch";
16     static string WeatherLink = "/api/Weathers";
17
18     public APIConnector(String AuthToken)
19     {
20         this.AuthToken = AuthToken;
21     }
22
23     private string SendGET(string url)
24     {
25         HttpWebRequest request = (HttpWebRequest)WebRequest
               .Create(url);
26         request.Method = "GET";
27         request.ContentType = "application/json";
28         request.Headers.Add("Authorization","Bearer " +
               AuthToken);
29         request.AutomaticDecompression =
               DecompressionMethods.GZip |
               DecompressionMethods.Deflate;
30         using(HttpWebResponse response = (HttpWebResponse)
               request.GetResponse())
31             using(Stream stream = response.
                   GetResponseStream())
32                 using(StreamReader reader = new
                       StreamReader(stream))
33                 {
34                     return reader.ReadToEnd();
35                 }
36     }
37     public List<Device> ParentDevices()
38     {
39         string Devices = SendGET(BaseUrl + DevLink);
40         List<Device> parents = new JavaScriptSerializer().
               Deserialize<List<Device>>(Devices);
```

```
41          return parents;
42      }
43      public Device SingleMasterDevice(string parentID)
44      {
45          string Devices = SendGET(BaseUrl + DevLink+"/"+
                parentID);
46          Device Master = new JavaScriptSerializer().
                Deserialize<Device>(Devices);
47          return Master;
48      }
49      public  List<ValueEntry> Values(string DeviceID,
            DateTime StartDate, DateTime EndDate )
50      {
51          List<ValueEntry> result = new List<ValueEntry>();
52          DateTime current = StartDate;
53          while (current<=EndDate)
54          {
55              string vals = SendGET(BaseUrl + ValLink + "/" +
                     DeviceID + "/" +current.Year+"-"+current.
                    Month+"-"+current.Day);
56              List<ValueEntry> Values = new
                    JavaScriptSerializer().Deserialize<List<
                    ValueEntry>>(vals);
57              foreach(ValueEntry VE in Values)
58              {
59                  if ((VE.stamp.CompareTo(StartDate) >0 || VE
                        .stamp.CompareTo(StartDate)==0) && (VE.
                        stamp.CompareTo(EndDate) < 0 || VE.
                        stamp.CompareTo(EndDate)==0))
60                  {
61                      result.Add(VE);
62                  }
63              }
64              current = current.AddDays(1);
65          }
66          return result;
67      }
68      private void SendPut(ValueUpdate VU , string url)
69      {
70          var httpWebRequest = (HttpWebRequest)WebRequest.
                Create(url);
71          httpWebRequest.ContentType = "application/json";
72          httpWebRequest.Method = "PUT";
73          httpWebRequest.Headers.Add("Authorization", "Bearer
                 " + AuthToken);
74
75          using (var streamWriter = new StreamWriter(
                httpWebRequest.GetRequestStream()))
76          {
77              string json = new JavaScriptSerializer().
```

```
                Serialize(VU);
78              streamWriter.Write(json);
79              streamWriter.Flush();
80              streamWriter.Close();
81          }
82          var httpResponse = (HttpWebResponse)httpWebRequest.
                GetResponse();
83          using (var streamReader = new StreamReader(
                httpResponse.GetResponseStream()))
84          {
85              var result = streamReader.ReadToEnd();
86          }
87      }
88      public void UpdateTargetTemp(Device d, double
            TargetTemp)
89      {
90          ValueUpdate VU = new ValueUpdate();
91          VU.Id = d.Id;
92          VU.Value = TargetTemp;
93
94          SendPut(VU, BaseUrl + DevconfLink + "/" + VU.Id);
95      }
96      public void UpdateSwitches(List<Device> switches, bool
            onState)
97      {
98          foreach(Device d in switches)
99          {
100             ValueUpdate VU = new ValueUpdate();
101             VU.Id = d.Id;
102             VU.Value = onState ? 1 : 0;
103             SendPut(VU, BaseUrl+switchLink+"/"+d.Id);
104         }
105     }
106     public double OutsideTemp()
107     {
108         string weather = SendGET(BaseUrl + WeatherLink);
109         return new JavaScriptSerializer().Deserialize<
                Weather>(weather).Temperature;
110     }
111 }
112
113 public class ValueUpdate
114 {
115     public string Id { get; set; }
116     public double Value { get; set; }
117 }
118     public class Device
119 {
120     public string Id { get; set; }
```

```
121     public string Name { get; set; }
122     public string parentID { get; set; }
123     public List<Device> Children { get; set; }
124     public double Value { get; set; }
125 }
126 public class ValueEntry
127 {
128     public DateTime stamp { get; set; }
129     public double Value { get; set; }
130 }
131 public class Weather
132 {
133     public double Temperature { get; set; }
134 }
```

**Listing 9.7: Program.cs**

```
using System.Collections.Concurrent;
using Math;
using Math.Functions;
using Math.LinnearRegression;
using System;
using System.Collections.Generic;

namespace Tempiro
{
    class Program
    {
        static void Main(string[] args)
        {
            //setup
            APIConnector Conn = new APIConnector("Access
                Token");
            Console.WriteLine("Coose a device (0-x)");
            Console.WriteLine(Conn.ParentDevices().ToString
                ());
            foreach (Device D in Conn.ParentDevices())
            {
                Console.WriteLine(D.Name);
            }
            int choice = Int32.Parse(Console.ReadLine());
            Device device = Conn.ParentDevices()[choice];
            Console.WriteLine(device.Name + " Selected");

            //setting up the agent for handling the model
            HeatingAgent agent = new HeatingAgent(Conn,
                device);

            //beginning the loop for the user to schedule
                the testing temperature
            while (true)
            {
                int Year = 0;
                int Month = 0;
                int Day = 0;
                int Hour = 0;
                int Minute = 0;
                int TargetTemp = 0;
                DateTime Target = DateTime.Now;

                Boolean setupDone = false;
                while (!setupDone)
                {
                    try
                    {
```

```
45                      //getting user parameters
46                      Console.WriteLine("Year when it
                            should be warm");
47                      Year = Int32.Parse(Console.ReadLine
                            ());
48                      Console.WriteLine("Month when it
                            should be warm");
49                      Month = Int32.Parse(Console.
                            ReadLine());
50                      Console.WriteLine("Day when it
                            should be warm");
51                      Day = Int32.Parse(Console.ReadLine
                            ());
52                      Console.WriteLine("Hour when it
                            should be warm");
53                      Hour = Int32.Parse(Console.ReadLine
                            ());
54                      Console.WriteLine("Minute when it
                            should be warm(This should be
                            an even multiple of 10)");
55                      Minute = Int32.Parse(Console.
                            ReadLine());
56                      Console.WriteLine("How Warmn should
                             it be");
57                      TargetTemp = Int32.Parse(Console.
                            ReadLine());
58
59                      Target = new DateTime(Year, Month,
                            Day, Hour, Minute, 0);
60                      setupDone = true;
61                  }
62                  catch (Exception e)
63                  {
64                      Console.WriteLine(e.Message);
65                      Console.WriteLine("Please try again
                            ");
66                  }
67              }
68              //setting up the different timers using the
                     models predicition
69              double StartingTemp = Conn.
                    SingleMasterDevice(device.Id).Value;
70              Console.WriteLine("Current inner
                    Temperature is: " + StartingTemp);
71              double OuterTemp = Conn.OutsideTemp();
72              Console.WriteLine("Current outer
                    Temperature is: " + OuterTemp);
73
74              DateTime prediction = agent.Predict(Target,
                     TargetTemp, StartingTemp, OuterTemp);
75              Console.WriteLine("Predicting to start
```

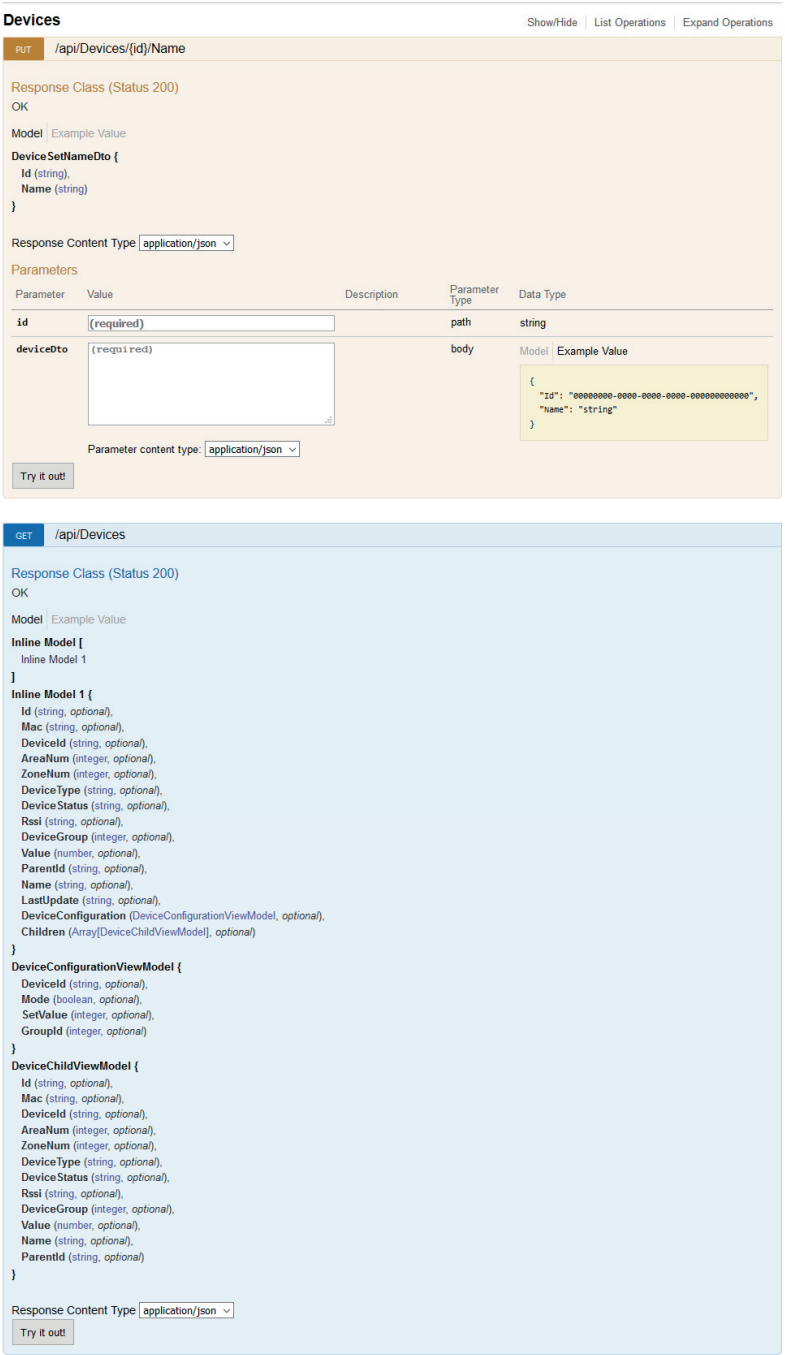
```
                heating: " + prediction.ToString());
76              Console.WriteLine("Now is: " + DateTime.Now
                );
77              Console.WriteLine("Using Model: " + agent.
                DictIndex(OuterTemp));
78              //waiting to start heating
79              Console.WriteLine("Waiting for predicted
                Moment");
80              while (prediction.CompareTo(DateTime.Now)
                >0)
81              {
82                  System.Threading.Thread.Sleep(60000);
83                  StartingTemp = Conn.SingleMasterDevice(
                    device.Id).Value;
84                  Console.WriteLine("Current Temperature
                    is: " + StartingTemp);
85                  OuterTemp = Conn.OutsideTemp();
86                  Console.WriteLine("Current outer
                    Temperature is: " + OuterTemp);
87                  Console.WriteLine("Using Model: " +
                    agent.DictIndex(OuterTemp));
88                  prediction = agent.Predict(Target,
                    TargetTemp, StartingTemp,OuterTemp)
                    ;
89                  Console.WriteLine("New predicting to
                    start heating: " + prediction.
                    ToString());
90              }
91              Console.WriteLine("Starting heating");
92              //StartHeating
93              Conn.UpdateSwitches(device.Children, true);
94              Console.WriteLine("Triggered Switches");
95              Conn.UpdateTargetTemp(device, TargetTemp +
                1);
96              Console.WriteLine("Setup Server Target");
97
98              DateTime Start = DateTime.Now;
99              Console.WriteLine("Waiting for Target
                Temperature");
100             //Waititng for it to be warm
101             double currentTemp = StartingTemp;
102             while ( currentTemp< TargetTemp)
103             {
104                 currentTemp = Conn.SingleMasterDevice(
                    device.Id).Value;
105                 Console.WriteLine("Current Tempertaure:
                     " + currentTemp);
106                 System.Threading.Thread.Sleep(60000);
107             }
108             Console.WriteLine("Stopping Heating");
109             //Stopping the heating
```

```
110                 Conn.UpdateSwitches(device.Children, false)
                        ;
111                 Conn.UpdateTargetTemp(device, 0);
112                 DateTime End = DateTime.Now;
113                 Console.WriteLine("Procces Finished: " +
                        End.ToString());
114
115                 //telling the agen to update the model
116                 agent.UpdateModel(Start, End, OuterTemp);
117                 Console.WriteLine("New Model: " + agent.
                        ToString(OuterTemp));
118                 Console.WriteLine("Model saved under: " +
                        agent.DictIndex(OuterTemp));
119                 Console.ReadKey();
120             }
121         }
122     }
123
124     class HeatingAgent
125     {
126         Dictionary<int, HeatingCoefficient> Coeffs;
127         APIConnector Conn;
128         Device Parent;
129
130         public HeatingAgent(APIConnector connector,Device
                Parent)
131         {
132             Coeffs = new Dictionary<int, HeatingCoefficient
                    >();
133             Conn = connector;
134             this.Parent = Parent;
135         }
136         public DateTime Predict(DateTime Target, double
                TargetTemp, double StartingTemp, double
                OuterTemp)
137         {
138             if(Coeffs.ContainsKey(DictIndex(OuterTemp)))
139             {
140                 //calculate backwards
141                 return Target.Subtract(TimeSpan.FromHours(
                        Coeffs[DictIndex(OuterTemp)].
                        SolveForTime(StartingTemp, TargetTemp))
                        );
142             }
143             else
144             {
145                 //divided by one is implied as 1 C/h
146                 return Target.Subtract(TimeSpan.FromHours(
                        TargetTemp - StartingTemp));
147             }
```

```
148
149         }
150         public void UpdateModel(DateTime StartTime,
                DateTime EndTime, double OuterTemp)
151         {
152             List<ValueEntry> Temps = Conn. Values(Parent.Id
                    , StartTime, EndTime);
153             Table TempTable = new Table(0, 2);
154             TempTable.setLable(0, "Date"); TempTable.
                    setLable(1, "Temp");
155             double j = 0;
156             foreach (ValueEntry VE in Temps)
157             {
158                 double[] NewRow = new double[2];
159                 NewRow[0] = j; NewRow[1] = VE.Value;
160                 TempTable.AddRow(NewRow);
161                 j += 1.0/6.0;
162             }
163
164             Console.WriteLine("Data that will be used for
                    Least Square: ");
165             Console.WriteLine(TempTable.ToString());
166
167             //Setting up table meta data for the linnear
                    regression
168             int ycol = TempTable.LableIndex("Temp");
169             int x0col = TempTable.LableIndex("Date");
170
171             TempTable.setLable(ycol, "y");
172             TempTable.setLable(x0col, "feature0");
173             if (Coeffs.ContainsKey(DictIndex(OuterTemp)))
174             {
175                 //Updating the stored model in the interval
176                 Coeffs[DictIndex(OuterTemp)].update(
                        MultiVariableLinnearRegression.
                        LeastSquare(TempTable, 1));
177             }
178             else
179             {
180                 //adding a nerw model for the interval
181                 Coeffs.Add(DictIndex(OuterTemp), new
                        HeatingCoefficient(
                        MultiVariableLinnearRegression.
                        LeastSquare(TempTable, 1)));
182             }
183         }
184         public string ToString(double OuterTemp)
185         {
186             return "Model Saved at: " + DictIndex(OuterTemp
                    ) + "is: " + Coeffs[DictIndex(OuterTemp)].
```

```
                ToString();
187         }
188         public int DictIndex(double OuterTemp)
189         {
190             int DictIndex = (int)System.Math.Round(
                    OuterTemp);
191             if (OuterTemp < 0)
192             {
193                 DictIndex -= 5;
194             }
195             return DictIndex /= 5;
196         }
197
198     }
199
200     class HeatingCoefficient
201     {
202         double coeff;
203         int nbrVals;
204         public HeatingCoefficient(Model data)
205         {
206             coeff = data.GetFunction().GetCoeff().get(0, 0)
                    ;
207             this.nbrVals = 1;
208         }
209         public void update(Model data)
210         {
211             double new_coeff = data.GetFunction().GetCoeff
                    ().get(0, 0);
212             coeff = ((coeff * nbrVals) + new_coeff) / (
                    nbrVals + 1);
213             ++nbrVals;
214         }
215         public  double SolveForTime(double StartTemp,
                double TargetTemp)
216         {
217             // Y=AX+B
218             // (Y-B)/A = X
219             return (TargetTemp - StartTemp) / coeff;
220
221         }
222         public override string ToString()
223         {
224             return "Coefficient is: " + coeff + " Number
                    Values is: " + nbrVals;
225         }
226     }
227 }
```

# B Tempiro API documentation



**Devices** Show/Hide | List Operations | Expand Operations

PUT /api/Devices/{id}/Name

Response Class (Status 200)

OK

Model | Example Value

**DeviceSetNameDto {**
**Id** (string),
**Name** (string)
**}**

Response Content Type application/json

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| **id** | (required) | | path | string |
| **deviceDto** | (required) | | body | Model \| Example Value |

```
{
  "Id": "00000000-0000-0000-0000-000000000000",
  "Name": "string"
}
```

Parameter content type: application/json

Try it out!

GET /api/Devices

Response Class (Status 200)

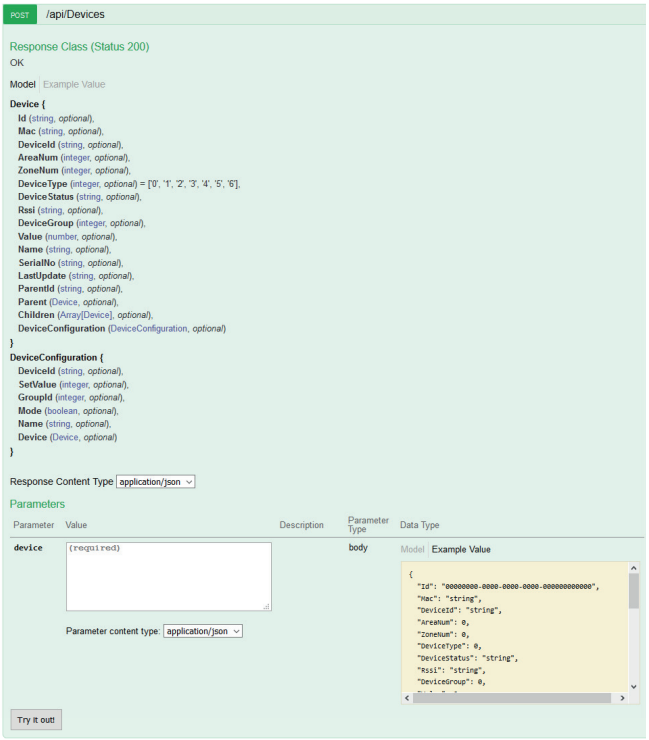
OK

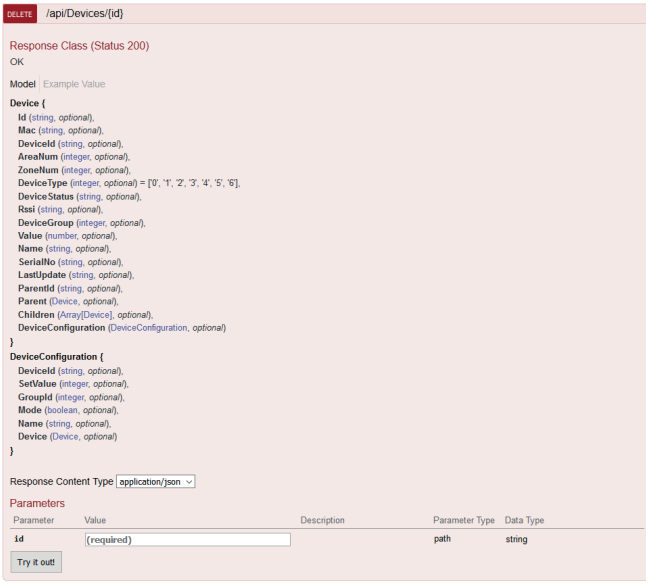
Model | Example Value

**Inline Model [**
Inline Model 1
**]**
**Inline Model 1 {**
**Id** (string, *optional*),
**Mac** (string, *optional*),
**DeviceId** (string, *optional*),
**AreaNum** (integer, *optional*),
**ZoneNum** (integer, *optional*),
**DeviceType** (string, *optional*),
**DeviceStatus** (string, *optional*),
**Rssi** (string, *optional*),
**DeviceGroup** (integer, *optional*),
**Value** (number, *optional*),
**ParentId** (string, *optional*),
**Name** (string, *optional*),
**LastUpdate** (string, *optional*),
**DeviceConfiguration** (DeviceConfigurationViewModel, *optional*),
**Children** (Array[DeviceChildViewModel], *optional*)
**}**
**DeviceConfigurationViewModel {**
**DeviceId** (string, *optional*),
**Mode** (boolean, *optional*),
**SetValue** (integer, *optional*),
**GroupId** (integer, *optional*)
**}**
**DeviceChildViewModel {**
**Id** (string, *optional*),
**Mac** (string, *optional*),
**DeviceId** (string, *optional*),
**AreaNum** (integer, *optional*),
**ZoneNum** (integer, *optional*),
**DeviceType** (string, *optional*),
**DeviceStatus** (string, *optional*),
**Rssi** (string, *optional*),
**DeviceGroup** (integer, *optional*),
**Value** (number, *optional*),
**Name** (string, *optional*),
**ParentId** (string, *optional*)
**}**

Response Content Type application/json

Try it out!


POST
/api/Devices
Response Class (Status 200)
OK
Model
Example Value
Device {
Id (string, optional),
Mac (string, optional),
DeviceId (string, optional),
AreaNum (integer, optional),
ZoneNum (integer, optional),
DeviceType (integer, optional) = ['0', '1', '2', '3', '4', '5', '6'],
DeviceStatus (string, optional),
Rssi (string, optional),
DeviceGroup (integer, optional),
Value (number, optional),
Name (string, optional),
SerialNo (string, optional),
LastUpdate (string, optional),
ParentId (string, optional),
Parent (Device, optional),
Children (Array[Device], optional),
DeviceConfiguration (DeviceConfiguration, optional)
}
DeviceConfiguration {
DeviceId (string, optional),
SetValue (integer, optional),
GroupId (integer, optional),
Mode (boolean, optional),
Name (string, optional),
Device (Device, optional)
}
Response Content Type application/json
Parameters
Parameter
Value
Description
Parameter Type
Data Type
device
(required)
body
Model
Example Value
Parameter content type: application/json
Try it out!


DELETE
/api/Devices/{id}
Response Class (Status 200)
OK
Model
Example Value
Device {
Id (string, optional),
Mac (string, optional),
DeviceId (string, optional),
AreaNum (integer, optional),
ZoneNum (integer, optional),
DeviceType (integer, optional) = ['0', '1', '2', '3', '4', '5', '6'],
DeviceStatus (string, optional),
Rssi (string, optional),
DeviceGroup (integer, optional),
Value (number, optional),
Name (string, optional),
SerialNo (string, optional),
LastUpdate (string, optional),
ParentId (string, optional),
Parent (Device, optional),
Children (Array[Device], optional),
DeviceConfiguration (DeviceConfiguration, optional)
}
DeviceConfiguration {
DeviceId (string, optional),
SetValue (integer, optional),
GroupId (integer, optional),
Mode (boolean, optional),
Name (string, optional),
Device (Device, optional)
}
Response Content Type application/json
Parameters
Parameter
Value
Description
Parameter Type
Data Type
id
(required)
path
string
Try it out!

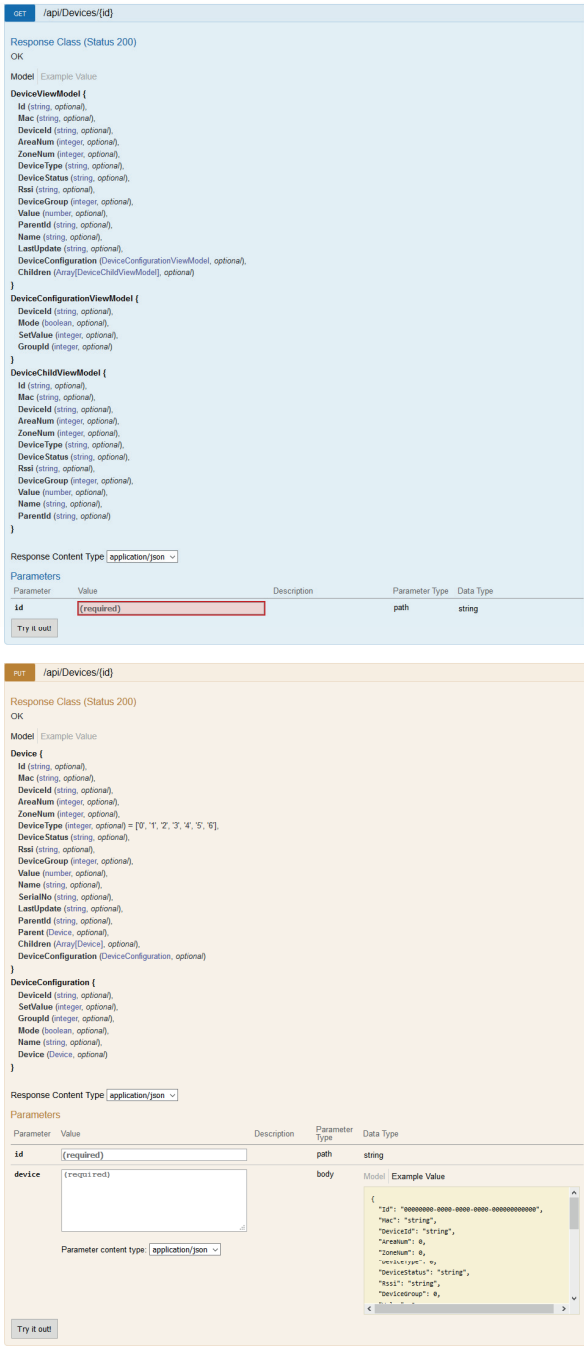



Figure 9.1: Device Documentation

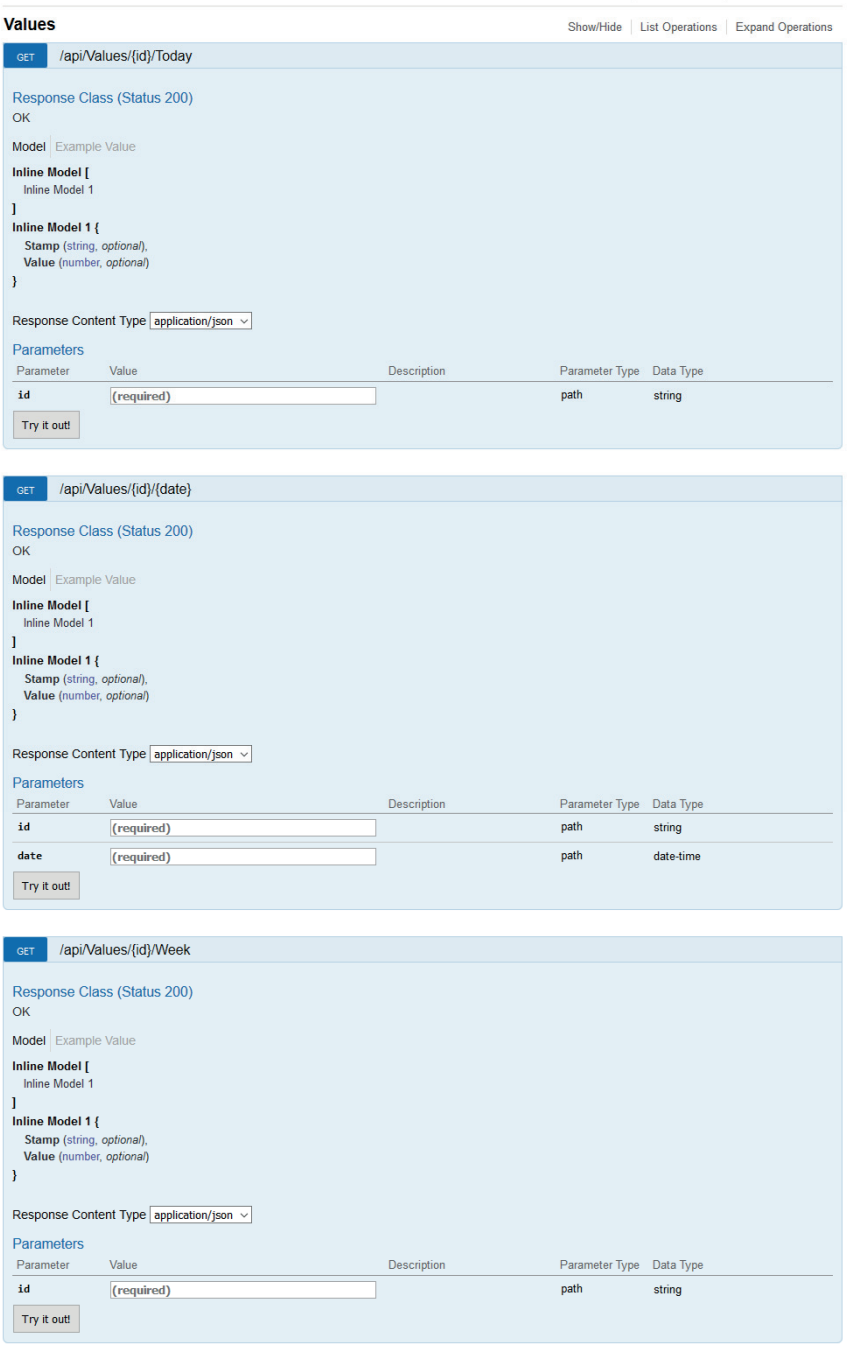



Figure 9.2: Value Documentation

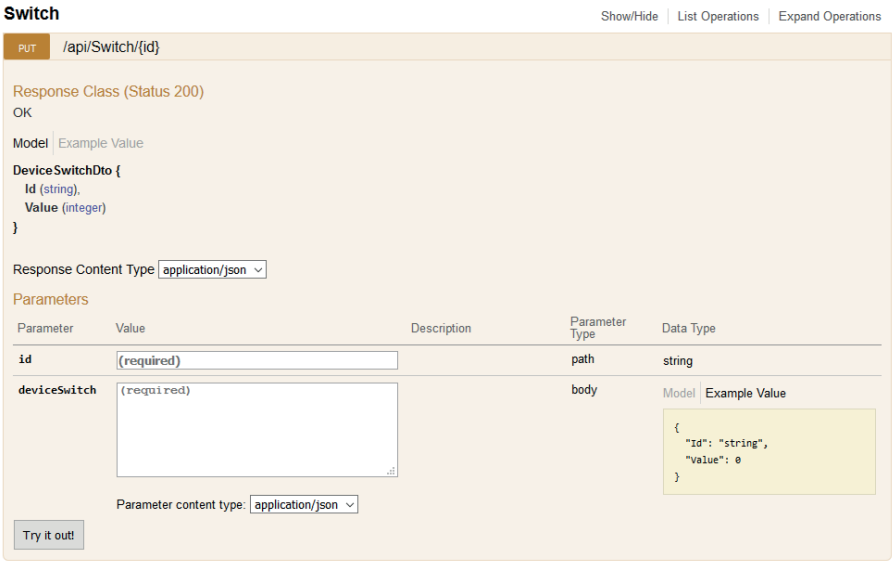



Figure 9.3: Switch Documentation

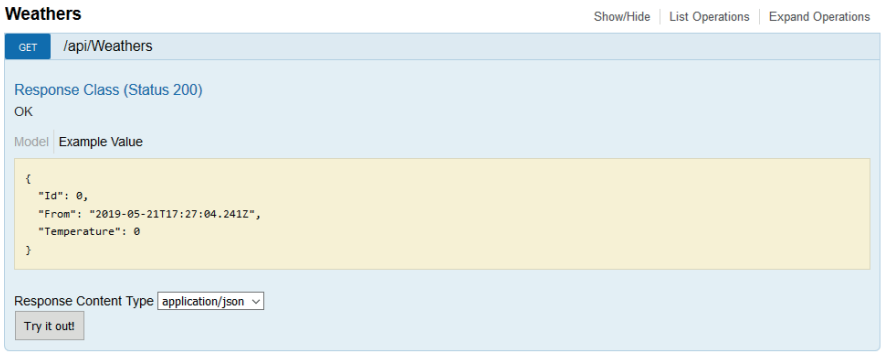



Figure 9.4: Weather Documentation

Printed by Tryckeriet i E-huset, Lund 2019

LUND
UNIVERSITY

Series of Bachelor's theses
Department of Electrical and Information Technology
LU/LTH-EIT 2019-705
http://www.eit.lth.se