Report in MNXB01

Carl-Fredrik Lidgren
Daniel Falkowski
Isak Ellmer
Qian Liu

November 2021

Contents

1 Introduction

This is the final report for the project of the course MNXB01 with the topic about analysis the temperatures given by SMHI. In this project we will focus on the data of Lund.

2 Goals

We would like to achieve the following goals in this project.

1. Have a command-line interface that makes it easy to run the analysis commands on-demand.
2. Analyze temperature data over periods of time to conclude some things about the change in temperature over time. In particular,
   - for two given time periods, compute the average temperature distributions for those periods and compare them;
   - for a given (preferably long) time period, fit the temperature data to a function of the form $p_0 + p_1 x + p_2 \sin(2\pi x - p_3)$.
3. Analyse and process the temperature data in a specified month to find some interesting results.
   - Create a histogram of the average temperatures for the same month each year
   - Fit the histogram with Gaussian and find the 95% confidence interval(CI) for the fitting function
   - Plot a figure showing the highest and lowest temperature in that month each year
   - Plot the average temperatures by years in the above figure and predict the future temperature trend
4. Covid-19 broke out all over the world from the end of 2019 and deeply impressed people's daily life. With the data from Folkhälsomyndigheten (Fohm) about the number of people infected with Covid-19, discover the relationship between it and temperature between 2020.02.24(week 9) to 2021.4.4(week 13).

3 Approaches

3.1 Data extraction, storage & management

In general the data is foundationally handled by the class `WeatherDataVec` which provides an interface for the weather data, simultaneously being the object manages loading and processing of the raw information. Roughly, this can be split up into three steps.

3.1.1 Extraction

The data extraction is done automatically by one of the several featured options for constructing a `WeatherDataVec` object; a sketch of the procedure is demonstrated below.

```
#include "WeatherDataVec.h"
#include <string>

WeatherDataVec Wdata {"filepath"};
```

The actual processing of the file is then done in the initialization process, which is mostly handled by the functions in `csvregex.h`. Below is an (edited) excerpt of the code which reads the data:

```
//Get semi-raw data from csv
std::vector<std::string> datavec(std::string filename){
    std::vector<std::string> v;
    std::string line;
    std::ifstream file (filename);
    std::regex re ("...long regex string...");
    std::smatch m;
    while (std::getline(file,line)){
        std::regex_search(line,m,re);
        for(std::string x : m){
            if (x != ""){
                v.push_back(x);
            }
        }
    }
    file.close();
    return v;
}
```

3.1.2 Storage

The data, as alluded above, stored in the class `WeatherDataVec` which is in essence a wrapper/utility around a `std::vector<WeatherDataLine>` where the class `WeatherDataLine` is another class which stores a single datapoint of weather data. Below is an (edited) excerpt which demonstrates `WeatherDataLine` the member variables and the primary constructor for the class:

```
class WeatherDataLine{
    public:
    std::vector<int> date;
    std::vector<int> time;
    double temperature;

    WeatherDataLine(std::string s){
        //This will look like [1863-02-13,07:00:00,2.1,G]
        std::vector<std::string> first_split {string_split(s,";")};
        date = vstr_to_int(string_split(first_split[0],"-"));
        time = vstr_to_int(string_split(first_split[1],":"));
        temperature = std::stof(first_split[2]);
    }

```

```
15 //More code below
```

3.1.3 Management

A substantial portion of the data management is handled immediately by the class `WeatherDataVec` as there are a multitude of utility functions included for interfacing with the data; this include features such as: getting data by matching "regex", getting data between different dates, data slicing, getting max/min/average temperatures, etc.

Furthermore, to deal with the nuisance of interacting with calendar dates there exists another class `Gregorian` which provides calendar arithmetic and utility features.

3.2 Data analysis

We mainly use C++ and ROOT to analyse the temperature data and plot the figures.

3.2.1 Analysis of Temperatures Over Two Periods

This is done in the ROOT macro `avg_periodtemp.C`. The general premise is that we prepare two histograms and then plot them on top of each other to visually compare them.

Both histograms are produced by the function `temperature_over_period`, which takes two `int`s, along with temperature data stored in a `WeatherDataVec` as input, and returns a pointer to a histogram as an output (i.e. `TH1D*`). The histogram is generated by taking a vertical slice for each day in a year over the period and averaging the temperatures, then placing this average into a bin corresponding to that day. This could have, equivalently, been done with a `TGraph` with the y-axis being average temperature and x-axis being day number. Calculating the value for each day is done by combining various methods provided by the `WeatherDataVec` class.

The function `temperature_over_two_periods` is the function which calls `temperature_over_period` for two input periods and then combines the histograms on one canvas. It also computes the average difference in temperature between the two histograms. See the Results section for an example output.

3.2.2 Regression on Raw Temperature Data

This is done in the ROOT macro `global_temperature_regression.C`. The idea is simply to plot the weather data in a given period of years and fit a function to this data. The function, in particular, is of the form $p_0 + p_1 x + p_2 \sin(2\pi x - p_3)$, and this particular choice is made since this should, roughly speaking, model increasing but oscillating temperature over a number of years.

The plotting step is done by utilizing the `WeatherDataVec` class and the `Gregorian` class. In particular, the first lets us select the particular span of

years given as an input, and the latter lets us effectively compute the position a given day has in a year (i.e. a conversion from Y-M-D to fractions of years) using the Julian day number of a date. This is needed to make the x-axis of the plot useful. The plot is then generated using a `TGraph`.

The fitting step is done using ROOT's built-in fitting functionality. After this, it computes the predicted change in average temperature after 100 years. The prediction is quite rudimentary, and all it does is multiply the slope by the number of years, i.e. compute $100p_1$.

For an example output, see the Results section.

3.2.3 Analysis of monthly temperature

Relying on the data from the class `WeatherDataVec`, we plot the histograms and graphs with the help of the classes `TH1` and `TGraph` of ROOT to visualize our data.
The class `Analyse_Monthly` as in Figure 1a needs a temperature filename from the datasets and a exact month as the parameters. As the functionalities, the function `Temp_PerMonth()` plots a histogram of the distrubution for monthly average temperatures and `Month_Extreme()` plots a figure showing the highest, lowest and average temperatures of the specified month for each year.

```
class Analyse_Monthly {
    private:
    Int_t _month;
    WeatherDataVec _Wdata;
    public:
    Analyse_Monthly(Int_t month, std::string filename);
    void Temp_PerMonth(); //plot the histo of the average temperature of the month
    void Month_Extreme(); //plot the extreme temperatures & average temperature of the month each year
};
```

(a) the class `Analyse_Monthly`

```
class Analyse_Corona {
    private:
        std::string _filename;
        std::string _city;
        void Plot_Corona() const; //plot the figure of the relation between the temperature and number of people
infected with the COVID-19
    public:
        Analyse_Corona(std::string city, std::string filename);
};
```

(b) the class `Analyse_Corona`

Figure 1: classes `Analyse_Monthly` and `Analyse_Corona`

3.2.4 Covid-19 and the weekly temperature

The class `Analyse_Corona` as in Figure 1b needs a city name and a Covid-19 filename from datasets as parameters. As the functionalities, the function `Plot_Corona()` plots a figure containing the average temperatures and the number of infected people per week.

3.3 Command-line interface

We want to be able to run the analyzing functions from the terminal, from different files and with different arguments. Although we can do this by running root, it is more fun to program our own CLI. The CLI has basic commands like `clear` and `exit`, but we can also run the functions like `temperature_over_two_periods` etc. At the beginning you have to specify a filename, for example `"datasets/smhi-opendata_1_53430_20210926_101122_Lund"`, this is done with the command `"filename datasets/smhiopendata_1_53430_20210926_101122_Lund"`. Now we can run any of the functions we like on the chosen file.

The CLI works with a while loop that waits for input from `std::cin`. The input is first cleaned, if there by mistake are two consecutive spaces. Then it is separated by spaces into an array of strings. The array is then interpreted by nested if statements to see which command the input corresponds to.

4 Results

4.1 Analysis of Temperatures Over Two Periods

The purpose of the function `temperature_over_two_periods` is to demonstrate an increase in temperature when comparing "recent" weather with less recent weather. In particular, to make the result meaningful, the periods themselves have to be sufficiently large that they capture the "real" behavior around that time (as opposed to small-scale fluctuations in temperature), and the time periods have to be sufficiently separated for the difference to be visible. Hence, the periods 1890–1900 and 2005–2015 were chosen.

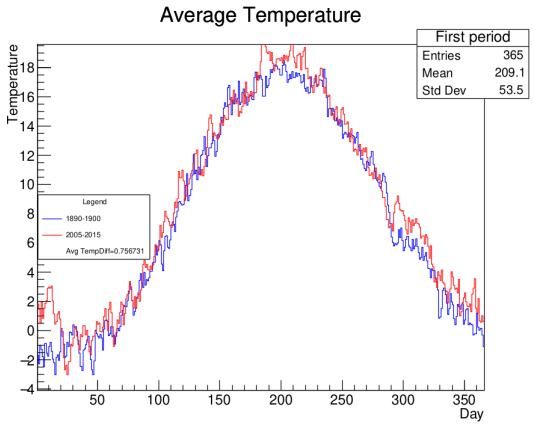


Figure 2: The output of `temperature_over_two_periods` for Lund's data using the periods 1890–1900 (blue) and 2005–2015 (red).

The two histograms in the plot shown in Figure 2 had an average difference of approximately 0.756 °C, which is indicative of average temperatures having

gotten warmer. One can also see in the plot that the extremes in the modern data are more prominent than in the historical data. Both of these things are in alignment with observations made in climate science.

4.2 Regression on Raw Temperature Data

This macro was, like the one above, made with the purpose of demonstrating an increase in average temperature over time. For this effect to be visible, a fairly large time period has to be used. Hence, the period chosen for demonstration was 1890–1990, i.e. a period of 100 years.

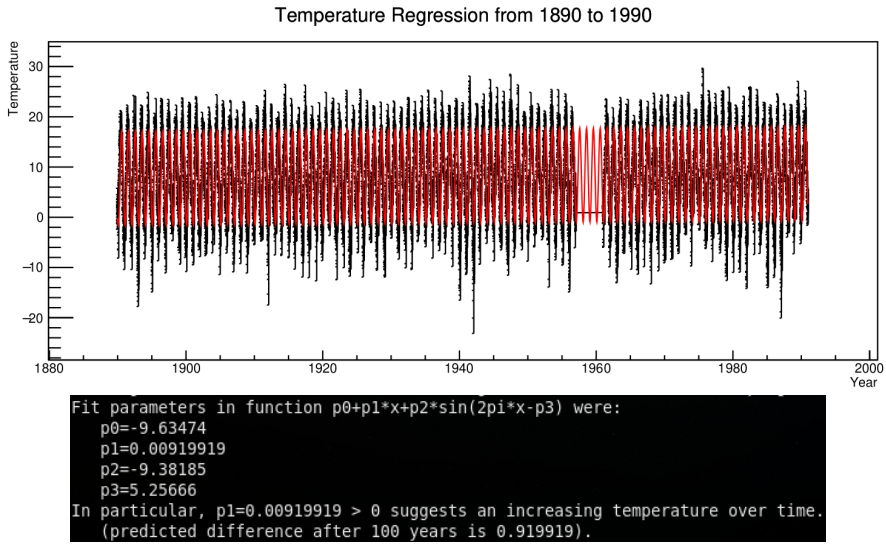


Figure 3: Plot for Lund's weather data 1890–1990, and the result of the fit.

In Figure 3 one can see the result of the macro. In particular, the produced function seems to fit the data fairly well, and the linear part has a slope of 0.009199 °C/year, yielding a predicted increase of a little under 0.92 °C per 100 years if the trend continues as it did there and the model is accurate. Regardless, the positive slope suggests an increasing temperature, which is what was also seen in Figure 2.

4.3 The temperature of a given month

In this part, all data will be based on some same months of each year.

4.3.1 Create a histogram of the average temperatures each year

Let's take a look at the January data first.

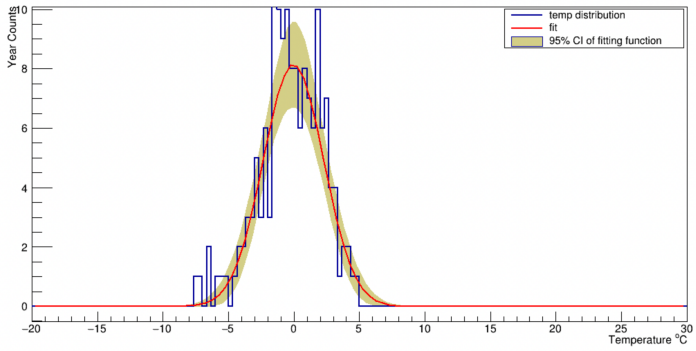


Figure 4: The average temperature in January

The blue line is the distrubution of the average temperature in January. It seems like Gaussian distribution, and it is indeed because of the central limit theorem (CLT). In next part, we will fit this histogram.

4.3.2 Fit the histogram

According to CLT, because we have enough data from the same month and they can been seem as independent random variables, their averages should tend towards a Gauss distribution. Thus we fitted the histogram with a Gaussian(red line).
Meanwhile, considering the fitted function is not 100% correct, the 95% CI which represents the error band for the fitted function is also plotted to see if the fit is good or not. As shown in the yellow part of Figure 4. The band indicates the range that we can be 95% sure contains the true fitting curve.
Figure 5 is the output of fit from the terminal. We can easily conclude that the fitted function for the January average temperatures distribution $f(x)$ is $f(x) = 8.15e^{-0.5(\frac{x+4.54}{2.31})^2}$.

EXT NO.	PARAMETER NAME	VALUE	ERROR	STEP SIZE	FIRST DERIVATIVE
1	Constant	8.15195e+00	9.50134e-01	1.47021e-03	4.00980e-06
2	Mean	-4.53680e-02	2.13280e-01	4.45695e-04	6.26417e-04
3	Sigma	2.30919e+00	2.09989e-01	4.56383e-05	1.84304e-03

Figure 5: The average temperature in January

4.3.3 Plot the extreme temperatures

The extreme situations in a month are worthy analysing as well.

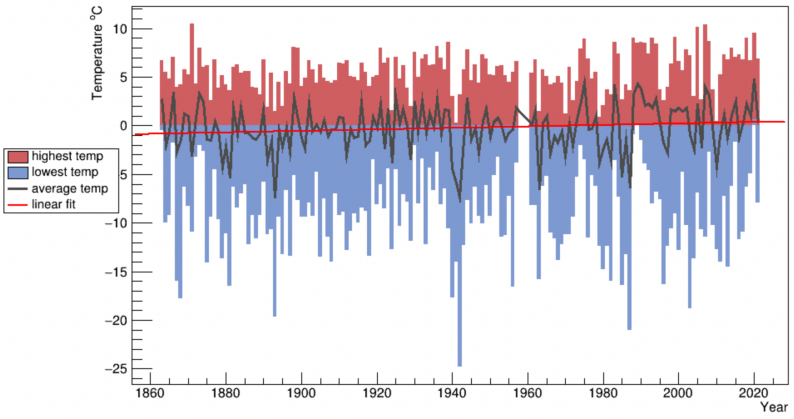


Figure 6: The average temperature in January

The red histogram part in Figure 6 shows the highest temperatures that happened in January each year. The blue part shows the lowest temperatures. It is not difficult to find that the overall extreme temperature is a random distribution, but we can obtain the year where the highest and the lowest temperature is.

```
The highest temperature of this month is 10.5 occuring in 1871
The lowest temperature of this month is -24.8 occuring in 1942

****************************************
Minimizer is Linear
Chi2                      =      876.561
NDf                       =          154
p0                        =     -15.0564   +/-   8.01694
p1                        =    0.0076378   +/-   0.00412771
```

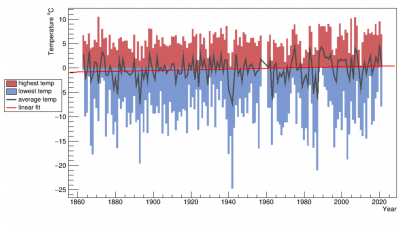
Figure 7: The average temperature in January

Thus the highest temperature of January in history, 10.5 ℃ occurred in 1871. The lowest one, -24.8℃ occurred in 1942.
January of 1942 was a really cold month, the highest temperature of January 1942 was only over 0 ℃. And the broken line also has a very large downward process here.

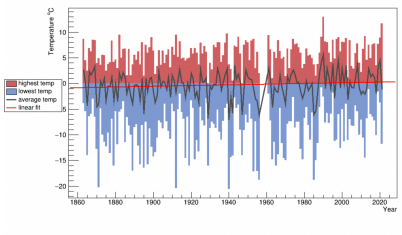
4.3.4 Plot and fit the average temperatures by years

The dark grey line in Figure 6 shows the average temperatures by years and the red line is its degree one polynomial (linear) fit. In this case, the slope of the fit function is 0.0076 read from the terminal, which implies the temperature is

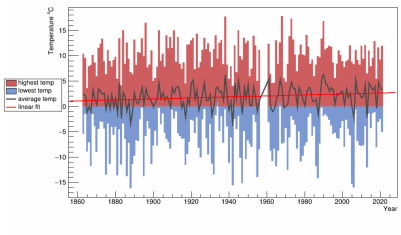
with a upward trend.
Let's observe all months with their fits.



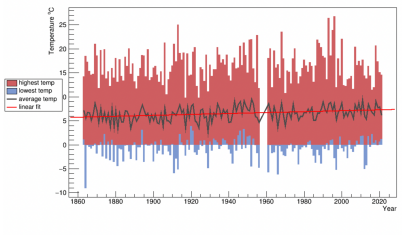
(a) January



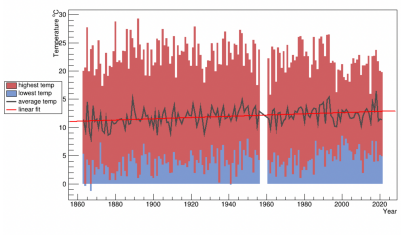
(b) February



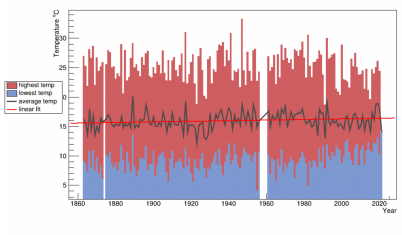
(c) March



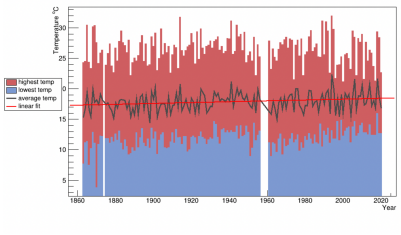
(d) April



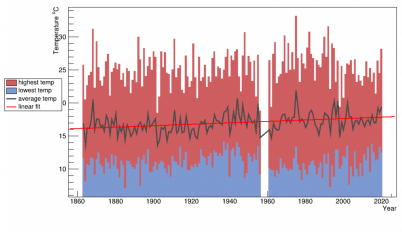
(e) May



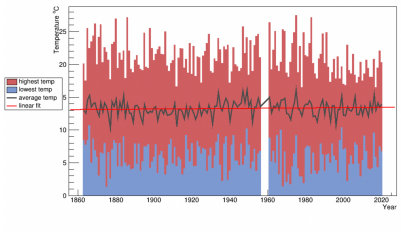
(f) June



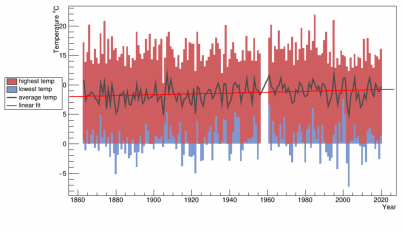
(g) July



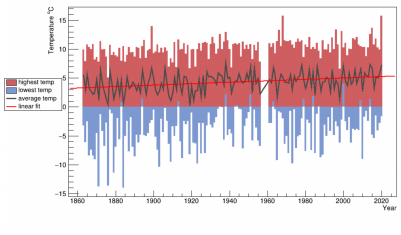
(h) August



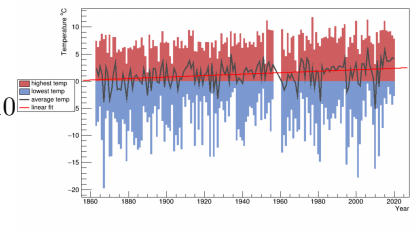
(i) September



(j) October



(k) November



(l) December

Figure 8: Trends of all months

By the Figure 8, we can boldly speculate that global warming seems to be happening in Lund too. That all red lines in each month are going up more or less implies the average temperature of every month trends higher.

4.4 Temperature and Covid-19

Finally, we will present Figure 9 with the average temperature and the number of people infected Covid-19 per week from 2020.02.24(week 9) to 2021.4.4(week 13).

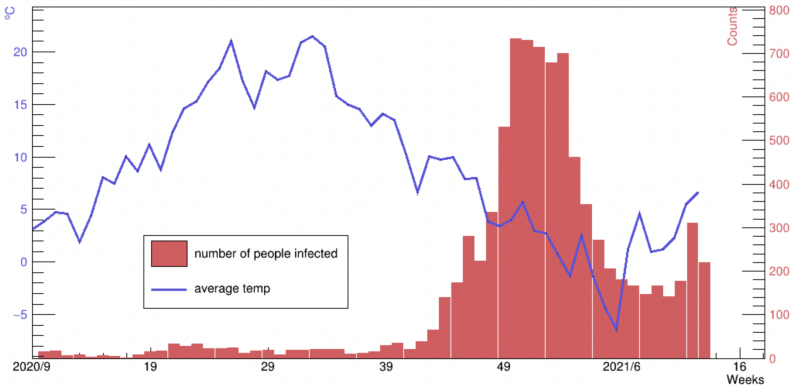


Figure 9: The average temperature and the number of infected people

Due to the effect of the vaccine, we did not analyse the data after week 13 of 2021, but we still found that the lower temperature may promote the spread of Covid-19. When 2020 entered about the 47th week, the temperature began to drop rapidly, and the virus also infected people on a large scale at this time. The peak of infection appears from the 50th week of 2020 to the second week of 2021. Then around the 6th week of 2021, the number of infections has dropped significantly.

Actually, the majority of currently available studies suggest that outdoor temperature is negatively correlated with Covid-19 transmission rates, although findings are not unanimous.[1]

References

[1] https://onlinelibrary.wiley.com/doi/10.1002/jmv.27042