Studying Climate Trends using C++ and ROOT

MNXB01
Philip J. Fredholm
Lee Chun Hin

November 13, 2022

Contents

Introduction

In order to be able to draw conclusions about some physical phenomena, such as the climate of Earth, it is sometimes necessary to be able to process large amounts of data. In order to save a considerable amount of time doing so, it is often the case that such processing may be done with the help of a computer programming language. The purpose of the subject of this report has been to try to use the programming language C++ in order to to process data from the SMHI OpenData initiative and draw conclusion about climate and weather trends from this data. In particular, attempts have been made in order to analyse the data and find the distribution of coldest and hottest day of each year, the first day of summer as well as to analyse the temperature of a given day of the year (August 23rd) over many years.

Theory

In order to be able make the analysis of what day is the first day of summer, a definition of when summer starts must first be laid out. The definition chosen for this project was 5 consecutive days of an average temperature greater or equal than 10.0 °C. This is also in accordance with SMHI's definition [5].

In order to be able to read in the data to be analysed, a class called `readData` , in C++ code according to the C++17 standard, was created and used by placing a source file in a folder called `src` and a header file in a folder called `include` inside the folder called `PhilipCode` [1]. The class `readData` would take a file name for a file placed in the same folder as the final program and how many header lines of a text file with relevant (and properly formatted) data to read in. It would then have three getter-functions, one to get the first year of measurements of the read in dataset, one to get the last year where the data had measurements, one to get the file name of the read in file and finally one to get the data that was read in. The read in data was stored in in a vector from the C++ standard library, which in turn contained one vector for each year that there had been measurements. Each vector for a year in turn contained twelve vectors, one for each year. Finally, each 'month-vector' contained 31 vectors, one for each possible day, within which (that is to say within the 'day-vectors') the average of the measured values for that day was stored. If no measurements were available for a given day, that 'day-vector' was left empty.

In order to actually find the warmest and coldest day of each year, as well as the start of summer, a file called `main.cpp` was placed in the folder `PhilipCode` and when compiled had access to the `dataReader` class by being compiled with a translation unit. This `main.cpp` file contained a function for finding either the hottest or coldest day of a given year by taking a pointer to a 'year-vector' of the type that was stored in the vector given by the getter function for the data of the `readData`-class. Similarly, there was also a function for finding the first day of summer, according to the definition presented earlier, by taking a pointer to a 'year-vector'. It should be noted that the way these functions operated was by creating a 'flattened' vector, which created a vector of as many elements as there were (nested) day vectors in the 'year-vector' given to the function as an argument. If a day had no measured values, it was still added to the 'flattened' vector and given a 'garbage' value, either extremely big or extremely small, as to not interfere with the logic used (in the implementation of the function) to find the desired day. It is important to note that since the year-vectors' month vectors always had 31 'day-vectors', the number of days in the flattened vector was always greater than 366, as even days which do not exist, such as a February 31st, were still added as a 'garbage' value. Finally, the `main.cpp` created

output files storing the hottest and coldest days found of each year by the index of the hottest day in the flattened year vector. It also outputted a datafile with the index of the first day (in the flattened year vector) of the first day to meet the previously stated requirement for the start of summer.

In order to plot the results, three ROOT macros, called `hot.C`, `cold.C` and `summer.C`, and placed in the `PhilipCode` folder, were used. These macros read in the relevant data file created from the `main.cpp` file and created histograms with 366 bins from one to 366. It then added the number of counts from the read in processed data to the relevant bins. Note that for the macro `cold.C`, there was instead 400 bins between -200 and 199, and all counts for days with day number greater than 180 were given to the bin which had day number of 365 less than the relevant day. Note also that these 'days' give the day out of a 'year' where each of the twelve months have 31 days. The macros then also saved the histograms as graphs and printed the mean value of the day from the read in data and printed it to the standard output. Note that in [1] the macros, along with the processed data, have been moved to a folder called `rootmacros` in the folder `PhilipCode`.

In order to make the histogram for the specific date,23 of August, I have use of the excel and root. Firstly, I use the finding function in excel to locate the string or the data I need. However, it is in a string format, and i need to separate the temperature data from the string. Hence, I make use of the data analysis function in excel. It can separate the string by detecting each of the symbol(such as ;,) in the string. As a result, I can separate the temperature data I need from the original excel file.After that, I use Win Zip to transfer the revised data to Auora. For the coding part, I create a new canvas named as c1 and with the title of Temperature of Umeå FLyplats. Since I want to show the two graphs in the same page and in parllel, so I make use of the function of Divide(2,1).After that, I use the public member function of TH1 which can be used to create a 1-d histogram with variable bins type of numbers.I create a histogram named as Tgraph1 with the x,y axis be [°c], and Entries respectively. Besides, the pixels of the histogram is set to 700. Apart from that, the range of x axis is from -30 to 40. Then, I try to operate and open the data file which named as 8-23Temp.txt into the histogram by using the command ifstream and open. The data should be store in a ifstream called infiles. After that, I create a new variable named as x by using the command double. In order to make sure the data keep on filling untill all data are input to the histogram, I make a while statment. While there is unused data in the infiles, it will always run the loop, and fill into the histogram.Lastly, I try to use the command SetFillColor to control the color of the graph. Besides, I also create variable of mean and stdv to store the standard deviation and the mean of the graph.The similar strategy was used to create another graph. There are two datasets applied into the code.The first dataset named as "8-23Temp.txt" contain the noon temerature of Umeå Airport in the each 23 August from 1962 to 2020. The second data set which named as "tempfordate.txt" contain the noon temperature of Umeå Airport in the each twenty-third of each month since from 1962 to 2020. To execute the code, a command of .x Final Project should be enter in root, and two graphs should be drawn based on that two datafiles.

Method

The `main.cpp` file was compiled using `g++` with an object file for the translation unit for the `readData`-class and the flags `-Wall`, `-Wextra` and `-Werror` and the version flag `-std=c++17`. The compilation was done with the help of a program called `Make`. The outputted file was then run, with the relevant data file to read from being placed in the same directory as the file which was

run, and data files containing the processed data were outputted. Following this, ROOT version 6.26/06 for Ubuntu 20 was run on Ubuntu 20, running through Windows subsystem for Linux (WSL2). The macros `hot.C`, `cold.C` and `summer.C` were compiled with the `.L <filename>+`-syntax in ROOT. The defined functions in these macros were then ran and the outputted values of the means were read off from the the standard output and the generated histograms were saved. In order to run the code for getting the graph, a command of .x Final Project should be input in root, and two graphs should be drawn based on two datafiles.

Results

The mean day as for being the coldest day of the year was found to b, rounded to the nearest integer) day 32, the mean day for being the coldest day of the year was found to be day 192 and the mean day to be the start of summer was found to be day 139. All of these numbers were also rounded to the nearest integer. The respective histograms for coldest and hottest days, as well as for the first day of summer, may be seen in Figure 1, Figure 2 and Figure 3 respectively.The mean temperature of Umeå Airport in 23Agust from 1962 to 2020 is 16.3 degree celsius with a standard deviation of 3.2.

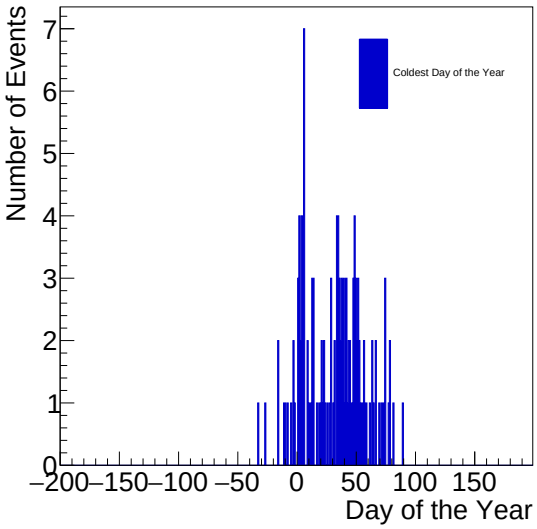

Figure 1: Shows the number of occurrences of a specific day being the coldest day of that year in Borås. Negative days indicate that the day was in the end of the previous year and its value for the day was shifted back 365 days. Note that the 'days' given here is the day out of a year where each of the twelve months are taken to have 31 days.

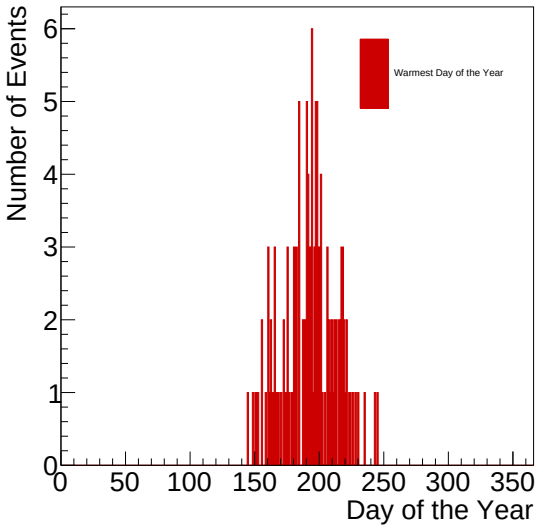

Figure 2: Shows the number of occurrences of a specific day being the warmest day of that year in Borås. Note that the 'days' given here is the day out of a year where each of the twelve months are taken to have 31 days.

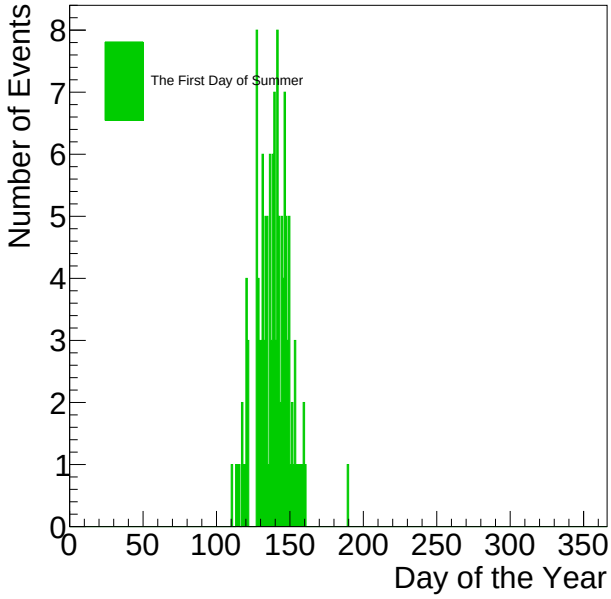

Figure 3: Shows the number of occurrences of a specific day being the first day of summer that year in Borås. Note that the 'days' given here is the day out of a year where each of the twelve months are taken to have 31 days.

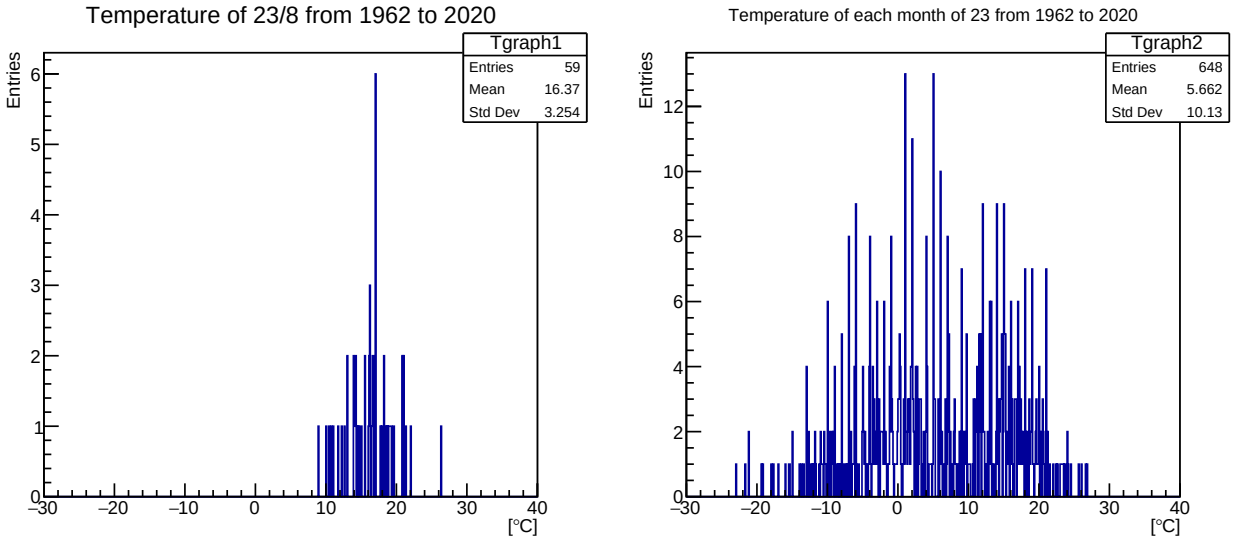

Figure 4: The left graph shows temperature of Umeå Airport in 23/8 from 1962 to 2020 while the right graph shows the temperature of Umeå Airport in each of the date of 23 since 1962 to 2020.

Discussion

It may be concluded that it was possible to produce histograms for the coldest and hottest days of the year, as well as for first day of summer. As for the calculated mean values for the relevant days, the coldest day was found to be, rounded upwards, to the 32nd day which would place it somewhere close to February 2nd, which seems reasonable. The first day of summer was found to occur around day 139. Considering that the start of June, the start of the six month where 5 months of about 30 days have passed, should be around day 150, this does not seem unreasonable. As for the hottest day of the year, the mean was around day 192. If the beginning of August occurs after seven months of approximately 30 days, that would place it at around day 210. Thus, from every day experience, a calculated value of 192 seems to be within reasonable bounds.

There are however some things which should be clearly remarked upon. That each month was assumed to have 31 days clearly means that the listed means do not corresponds to the actual day of an actual day of the Gregorian calendar. Secondly, as the days occurring after day 180 were shifted back by 365 days, while the used 'years' had 31 days each month, it is likely that many days for Figure 1 were not shifted properly. Since Figure 1 seems to mostly follow the normal distribution of a Gaussian, the large peak seemingly occurring 'off-center' of the thought of Gaussian could likely be explained by this. As for Figure 2 and Figure 3, these seem to, at least by eye, look somewhat like normal distribution. Furthermore, a significant improvement to taking the mean day for each respective quantity would probably had been to fit Gaussians to the histograms in Figure 1, Figure 2 and Figure 3 and look for what value these were centred around. This would also had provided other interesting results such as the standard deviations.

As for other things that may be improved upon, using a separate library for both reading in the data and for storing the dates, as to make sure that the processed data actually refers to the Gregorian calendar, would probably have been a big improvement to the code and thus the produced results.

Another thing to discuss is how the calculation of the average temperature of a given day was done. This was done by taking the average of all available measurements for a given day. Considering that the data [3] was not evenly spread throughout the day, this would clearly have biased the mean temperature of a day upwards considering there were not many temperature readings done during night-time. For the coldest and hottest day, more detailed data would probably not had made much of a difference since the coldest calendar day (24 hours) would be likely to have both the coldest day and the coldest night. Similarly, the hottest calendar day would likely correspond to both the hottest day and hottest night most years. However, for the data regarding the start of summer, the complication of not having the true mean temperature probably moved the calculated start of summer to a somewhat earlier day as the data was biased towards daytime. One way to improve upon this would had been to gather more data of how the temperature varies throughout the entire calendar day in Borås, perhaps by performing some Fourier decomposition, and then interpolating the data. This would then had let us compute a more accurate mean temperature for a given day.

To conclude; it is clear that the programming tools were successfully able to identify climate trends and give qualitative conclusions. However, the conclusions would have benefited from a more thorough implementation in order to be able to draw more quantitative conclusions, especially regarding properly implementing the Gregorian calendar.

The left graph of Fig4 shows the temperature for each month of 23/8 from 1962 to 2020.The mean temperature and the standard deviation is 16.37 and3.254 respectively.It is possible to predict the probability of particular tmeperature by using the mathematical formula below:

Z=(X-μ)/σ where Z=Standard score,,σ=standard deviation,μ=population mean

For example if we want to predict what is the probability for the coming 23/8 to be in the temperature of 12℃.By applying the above equation,

Z=(12-16.37)/3.254, Z=-1.34

By looking for the standard score table which can be found in the internet, we can observe the probability.In this case, the probabilty will be in0.09.

The right graph of figure4 show each of the date 23 in every month since 1962 to 2020. The mean and the standard deviation are 5.662 and 10.43 respectively.As discussed above, the same formula can be used to predcit the probability of specific temperature for the coming date 23.This graph may be important for the temperature forecasting in the observatory which can act as the statistic reference or database.Besides the left graph having a lower standard deviation than the right, which means that the data in the left are less disperse and more closing to the mean, which lead to a more focusing graph.

References

[1] The code in the folder "PhilipCode", available on a remote Git repository on GitHub at https://github.com/fredholmP/MNXB01-FinalProject .

[2] The code in the folder "ChrisCode" on a remote Git repository on GitHub at https://github.com/fredholmP/MNXB01-FinalProject .

[3] The data available from January 1st 1884 at 07.00 up until June 1st 2021 at 06.00 for the city Borås, provided by Lund Univeristy in association with the course MNXB01 during the autumn semester of 2022. At the time of writing, this data may be downloaded from the website https://www.smhi.se/data/meteorologi/ladda-ner-meteorologiska-observationer .

[4] The data available for the "Umeå Flygplats" measuring station, provided by Lund Univeristy in association with the course MNXB01 during the autumn semester of 2022. At the time of writing, this data may be downloaded from the website https://www.smhi.se/data/meteorologi/ladda-ner-meteorologiska-observationer .

[5] The website https://www.smhi.se/en/weather/observations/season-arrival/som/2022, last viewed on November 4th 2022.