MNXB01
Analysis of weather data from Uppsala

Daniel Magdalinski (da1162ma-s@student.lu.se)
Dilan Chroscik (di3371ch-s@student.lu.se)
Jim Klintrup (ji0687kl-s@student.lu.se)

November 7th, 2019

Lund University

Department of Physics

Introduction

This report covers the process of analysing weather data from SMHI (Sweden's meteorological and hydrological institut) and the results of that analysis. The data is mostly taken from the city of Uppsala and covers from 1722 up until 2013. The coding language used is that of C++ and ROOT. Three different results is presented, the mean temperature over the span of all years with measurements, the beginning of spring for the span of all years and finally the hottest and coldest days over the years.

Work Process

The process started with a meeting where all members of the project were present. Before looking at the data in the data sets it was decided to focus on one data set, get the analysis working and then, if time permitted, expand the code to work for other data sets. Three questions were devised in order to have a direction of the method to be programmed. The questions are: What is the mean temperature over the years? When does spring start on different years? What is the hottest and coldest day for a specific year? Depending on the results from the analysis of the data, it remained to be seen if anything could be inferred. After looking at the different data sets, it was decided to start with the data pertaining to Uppsala. The data structure was compatible with that of a tree structure and so attempt were made to get the data into such a structure. After several attempts, without success, it was determined that a more crude method would suffice as time was running short. The requirement that specific months and days had to be attainable meant complexity in the programming of the tree structure.
A 3D array structure, as seen in read3Darray.C, meant that indices could match the specific year, month and day. A consequence of this method is that roughly 17/20 of the array is filled with data that is not used and just takes up space. This has been considered and is regarded as acceptable, as the data set is small enough and thus the computing time is small. Another factor is again time, all team members feel that this method is unsatisfactory but is adequate for our needs. Something that also could have been improved is the fact that the array is declared globally. Initializing a pointer and returning it from the function would have been a better way of constructing this method.

The process for the func2mean.C function was started with just computing the data. Plotting the histograms on top of each other with THStack took some time to fix but eventually it worked. A moving average was coded and eventually several attempts were made at fitting a function to the moving average and the original histogram. Summation of cosines and gaussians were tried with or without an addition of a polynomial. The cosines never gave a good result but the gaussian method gave a smooth curve following the moving average. This fit was done with many parameters which lowers the contribution of a fit because of over-fitting. The choice of function also very much determines the outcome of the fit so a reasonable prediction of the future development was not found.

Before writing the code for finding the start of spring, the definition of spring start had to be looked up. According to SMHI, the start of spring is the first day out of seven consecutive days, after 15 February, that has an average temperature between 0°C and 10°C. Although, since the code gave false positives due to the requirement of being under 10°C, this requirement was ignored. The 3D array from read3Darray.C was used, to loop over and find spring start. This is done by FindSpringStart.C. Once found, the dates for the spring starts were used by GraphSpring.C to graph the results as well as two polynomial fits, to find the average start of spring as well as the average change of start of spring.

From the beginning it was only intended for the checkday function to prompt the user to input a specific year and the macro would output the hottest and coldest day of that specific year. However, once the coding was complete, it was seen that the code could easily be expanded into checking all years with data and plotting a histogram of the hottest and coldest day for all years with data. And so that is implemented in the macro checkday.C.

Code description

read3Darray.C

The macro starts of with the creation of a 3D array (AvgTempCor) outside the scope of the function "read3Darray()" as the array is needed globally by the functions that do the analysis. The first dimension has 2014 indices which is used to represent the year. The second dimension has a length of 13, representing the months and the last dimension is of length 32, representing the days of the month. All index start at zero which is why the dimensions of the array are one larger in each direction, this to insure that the indices represent the actual year, month or day.
Next, the function read3Darray() is declared and implemented with no arguments.
The data file is loaded through ifstream with its file path and a check is done to ensure that the file was indeed loaded. In order to be able to check if there is data loaded into a element, all element are filled in a nested for-loop with the temperature -273.0 as this can easily be identified as missing data and ignored in the analysis.
The actual data from the data set is then transferred to the 3D array via a while-loop that reads each line and for each line sets the specific element given be the year, month and day with the corresponding corrected temperature.

func2mean.C

The code starts with imports and the inclusion of read3Darray.C so the code has access to the temperature array. Next, definitions of variables from the first and last year as well as the number of years is done. The variable nAvgPoints determines how many points are averaged over for every point in the moving average graph.
For-loops are then used to sum all temperatures over one year, that value is then set to the corresponding bins in a histogram. Another sum is also done over all years to get an average temperature for all years and this value is set to all bins in another histogram. A third histogram is filled with the average temperature when a year has a mean temperature over the overall average temperature and the yearly mean temperature in the other case. These three plots are then drawn on top of each other with the class THStack and the nostack option to create a deviation from average effect. For every year that has nAvgPoints/2 on both sides the average of these points were added to a array that was then plotted over the histograms.

FindSpringStart.C

The macro starts by including the 3D array from read3Darray.C, since we need to loop through the array to retrieve the temperatures for each date. Two arrays (*SpringDay* and *SpringMonth*) that will be filled with the day and month of the spring start are first initialized to have 291 elements, which corresponds to the total number of years in the data for Uppsala. They are declared globally for easy access by other macros.
The main function in the file, FindSpringStart(), starts by declaring an integer, *daycount*, that keeps track of the consecutive days with average temperature above 0°C, as well as a Boolean, *spring*, to keep track of when to break out of the loops.
Three for loops, loop through the years, month and days, respectively. In the first loop, *daycount* is initialized to 0 and *spring* to false. When looping through the days, the first if statement checks if the current day has a temperature above 0°C and if the month is between February and May. The second statement was used to avoid false positives. Then we check if the current month is February, since spring cannot start until after 15 February. Once the current day passes through these if statements, another for loop is used to check for consecutive days that meet the spring start criteria.
Since the 3D array has values set to -273.0°C, for days that do not exist, such as 30 or 31 February, the loop must ignore these values and continue with the loop to the next month, without giving a false negative. This is controlled by if statements.
For each consecutive day matching the criteria, *daycount* is incremented by 1. If a day breaks the cycle of being above 0°C, the*daycount* is set to 0 and we break out of the day loop. Once *daycount* ≥ 7, we set *spring* to true and store the day and month of the first day that triggered the loop checking for consecutive days. The dates are stored as strings first, but are then converted to integers with Atoi(), and are then added to the two arrays *SpringDay* and *SpringMonth*. Once the date for the spring start is found for the current year, we use the Boolean *spring* to break out of the loops to get to the next year in the loop.

GraphSpring.C

This macro includes FindSpringStart.C to get the arrays containing the dates of the start of spring for all the years. First a datime object is created, since we want to plot dates. Two arrays x and y are created. x will contain the integers 1722-2013 representing the years in our data. y will be filled wit the months and days of spring start, for all years. The years in y is set to 2000 for all entries, since we want to plot them all on the same year, and it is a leap year. Otherwise the graph would be hard to analyze.
Once the data for both axis have been filled, a pointer to a TCanvas and TGraph are created. Once the properties of the TGraph and the axis have been set to properly display the data, two polynomial fits were applied. The first fit checks the average date of spring start. The second fit tries to find a trend in the change of spring start dates, i.e. if overtime, the spring starts sooner or later in the year. A legend is initialized and filled with the polynomial fits and the values for the important coefficients.

checkday.C

In the macro for the checkday function, read3Darray.C is included as the macro needs to know what the array AvgTempCor is. Three pointers to histograms and the individual histograms are created. One for the hottest day of the year, one for the coldest and one

to be used as a divider.
The function then goes through the array, checking for the hottest and coldest day of the year that the for-loop is on and at the start of each year loop, the variable to check against is reset to zero. When passing the values found, the SetBinContent function is used which has the first bin number as 1 and so a workaround is implemented in the arguments for that function.
A function is named fitFunc is created to be used as a fit to the histograms. It is a linear function as it was desired to see if a trend could be inferred. The fit function is implemented on both the histogram for the hottest days and the coldest days. Then the aesthetics of the different histograms are defined.

Results

Mean temperature of each year

The figure below shows the mean temperature per year with a graph of the moving average. The graph is done with nAvgPoints = 6. Unfortunately no good fit could be made to the graph so no future prediction could be made.

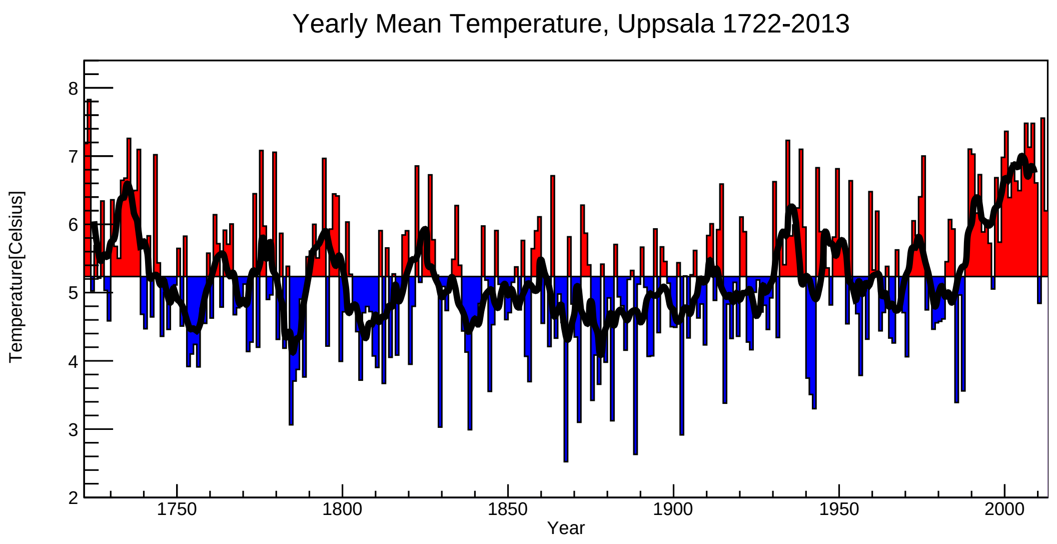

Figure 1: The figure shows the mean temperature per year, the color indicates deviation from average over all years. The graph is a moving average with 7 points used in the average.

Start of spring

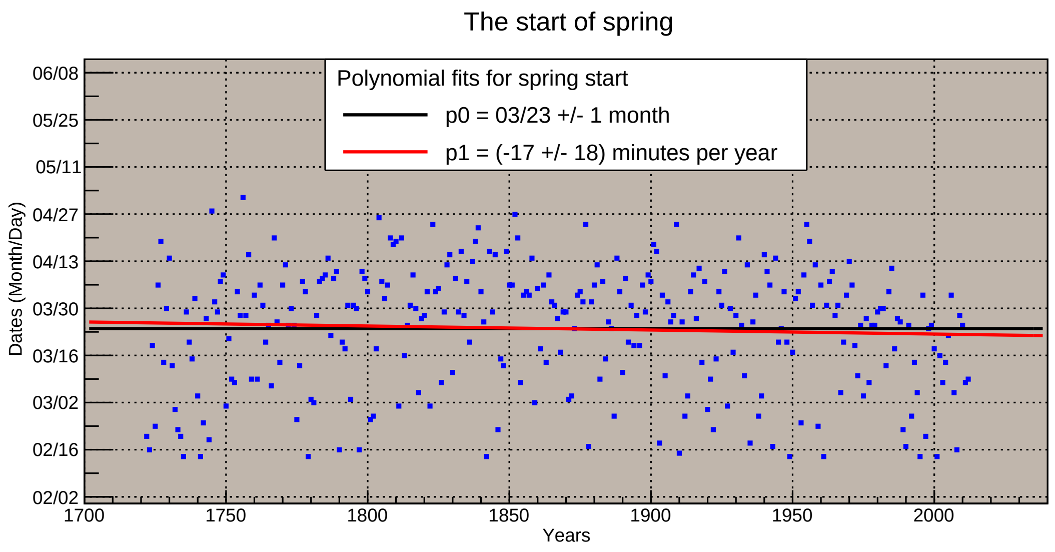

Figure 2: The graph of spring start from 1722 to 2013, with the two polynomial fits.

Figure 2, shows all the spring start dates with blue markers, as well as the average spring start in black and the average change in spring start in red. We first see that, overall the spring start oscillates from early in the year (February) to later in the year (May), this oscillation takes just a couple of years. The average day for spring start is around the 23 March $\pm$ 1 month. On average the spring start comes earlier for each year, with an average change of (-17 ± 18) minutes per year.

Trend in hottest and coldest day

In Figure 3, which displays the hottest and coldest days over the years, there is no apparent trend. The uncertainty in the slope of the fit for hHot is bigger than the slope. However, the χ^2/ndf is smaller than one, so the fit is actually somewhat fair. The fit to hCold is not good, the uncertainty in the slope is almost as big as the slope and χ^2/ndf is larger than one. It would seem that nothing can be deduced from this result.

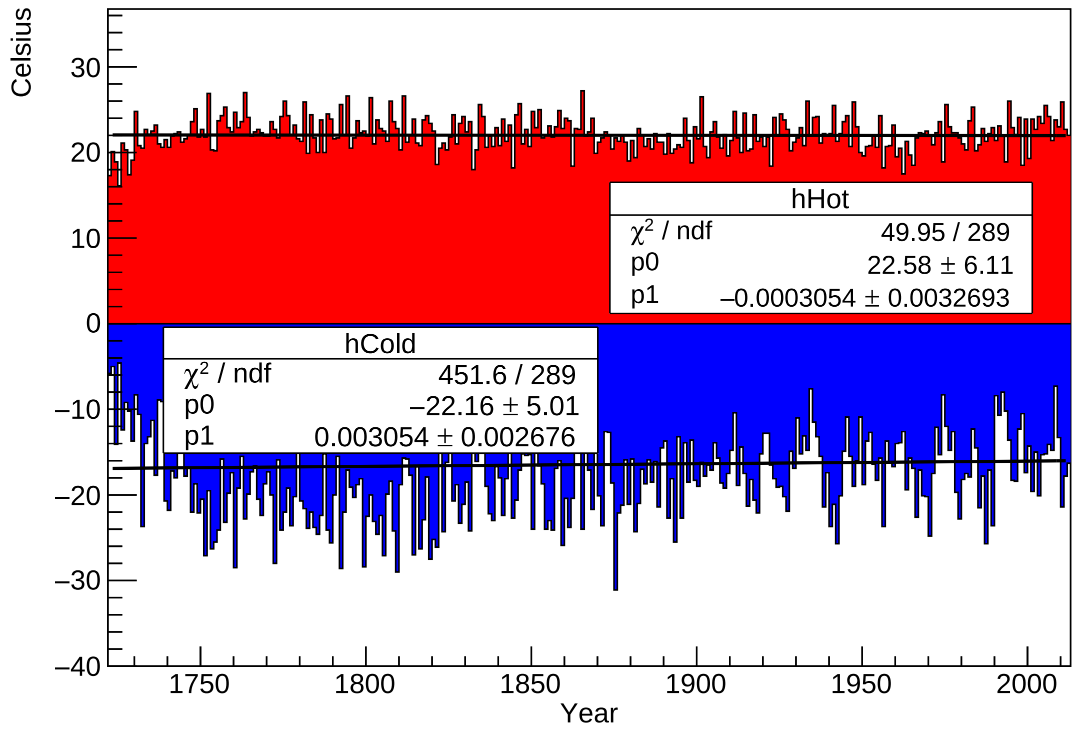

Figure 3: Histogram displaying the hottest (red) and coldest (blue) day of each year (x-axis). hHot, fit to the hottest days over the years. hCold is the fit to the coldest days over the years. Chi squared over ndf (number of dregrees of freedom), p0 is the coefficient, p1 is the slope of the fit.

Discussion/Conclusion

Overall, the task were completed successfully. Although, there is room for improvement in the code. The code works fine for our purposes, but does not scale well. Therefore, a few changes should be done to the code if one wishes to include more data sets. Instead of having globally accessible arrays, we could have returned pointers to them instead. This would decrease the memory load. While reading in the data, we chose to fill a 3D array in such a way that the indices could represent the dates. This resulted in an array of mostly empty elements, that were filled with the value -273.0°C. This can be avoided, and the array could be replaced by a tree. But the time did not allow us to solve these problems and the focus was to be able to complete the tasks first, before we could try to make the code more effective.