Group project, MNXB01

Elias Nyholm, Anna Giacomozzi, Johannes Stubbe

november 2018

Introduction

In this report, we will briefly explain how we produced graphs and analyzed data for the final group project in the course MNXB01. All code is published on GitHub, see `https://github.com/el5036ny-s/MNXB01-project`.

Import of data from .csv to .root

To store and read data, we wanted to use the tree data structure introduced to us in this course during the last couple of weeks. Therefore, the first step was to create a data conversion macro, which takes a .csv file as an input and outputs a .root file containing a tree of the data. Since (almost) all of the data sets we were given have identical format, we could be fairly specific about how to read the data from the .csv file.

```
Datum;Tid (UTC);Lufttemperatur;Kvalitet;;Tidsutsnitt:
1951-01-01;06:00:00;-8.2;Y;;Kvalitetskontrollerade historiska data (uto
1951-01-01;12:00:00;-8.0;Y;;Tidsperiod (fr.o.m.) = 1951-01-01 00:00:00
1951-01-01;18:00:00;-7.1;Y;;Tidsperiod (t.o.m.) = 2015-09-01 06:00:00 (
1951-01-02;06:00:00;-4.6;Y;;Samplingstid = Ej angivet
1951-01-02;12:00:00;-2.4;Y;;
```

Figure 1: Example of data format (.csv file).

```
class MyClass : public TObject {
      private:

            Int_t year;
            Int_t month;
            Int_t day;

            Int_t time;

            Double_t temp;

            Int_t qCode;
```

Figure 2: The structure of member variables in MyClass. This will determine how the data is stored in the TTree.

Each data point (line) in the file was first read and the first 3 occasions of the ";" delimeter was used to separate the data into four strings. `Int_t year`, `Int_t month` and `Int_t day` was extracted by separating the first string. The first 2 characters of string 2 were extracted and casted as `Int_t time`, and `Double_t temp` was given by casting the third string. Finally, `Int_t QCode` was set to `1` if the first character of string 4 equals `'G'`, otherwise set to `0`. This was to be able to cast this value as a `bool` later.

Now, for each of the data points, we filled the tree from the `MyClass` object. The tree was saved to a .root file for later use.

Plot 1: Temperature difference between noon and midnight

For this part, we want to read the data from the .root file and for each day of the year, calculate the average temperature difference between noon and midnight.

First, we read event from the .root file. For each of the events, we check if the time (hour) is set to 12 or 0, which corresponds to noon and midnight respectively. If this is the case, we add the value to the corresponding entry in an array. The index in the array corresponds to one less than the number of the corresponding day in the year (e.g. 1 feb has index 32-1 = 31). In a separate array, we also keep track of how many times we have added values to a specific day. After looping through all entries in the tree, we calculate the arithmetic mean of each day, and then plot that value against the number of that particular day in the year. We have also added an option to either include or exclude the "bad" data points from the dataset. The following plot is an example of what's generated. In this particular one, the bad points are included.

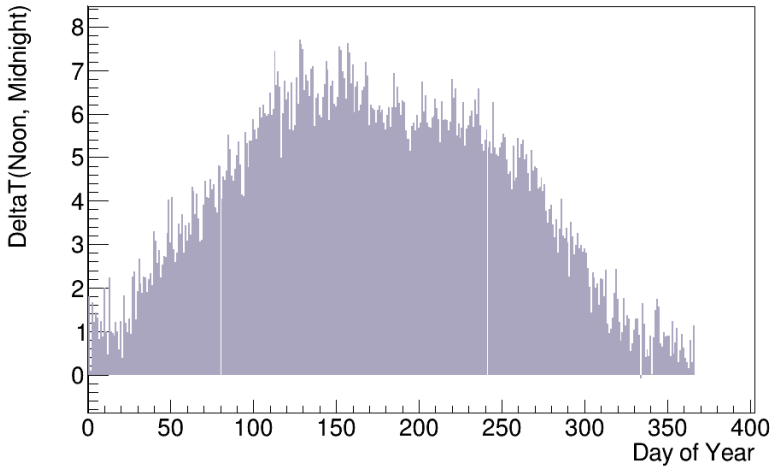

Figure 3: The structure of member variables in MyClass. This will determine how the data is stored in the TTree.

Plot 2: Mean temperature for each day of the year

In order to plot the mean temperature for each day of the year with the corresponding standard deviation, we proceed in the following way.

As for the previous function, we import the data and initialize the arrays.

The following two `for` loops are used to obtain the average temperature of each day of the year over all years.
In the first `for` loop (Figure 4), we loop over all the events in the dataset, and the temperatures for a given day are summed up in `tSum[currentDayOfYear-1]` while we keep track of the number of events summed for each day in the array `tCount`. The option to consider the confirmed data is provided here.

In the second `for` loop (Figure 5), we loop over the days and the average temperature `tAvg[i` is calculated for each day from `tSum[i]` and `tCount[i]`.

The standard deviation from the mean for each day is calculated with the two `for` loops.
The first loop (Figure 6) is looped over the number of events in the dataset and is used to sum the square of the deviation from the mean.

The second loop (Figure 7) is looped over the days and returns the standard

```
41    Int_t nEvents = eventBranch->GetEntries();
42
43    Int_t currentDayOfYear;
44
45    // Iterate through all events in TTree branch
46    for(Int_t i = 0; i < nEvents; i++) {
47
48      // Update event object
49      eventBranch->GetEntry(i);
50
51      // If excludeBadData = 1, then exclude deviating data points
52      if(excludeBadData && !(event->GetQCode())) { continue; }
53
54      currentDayOfYear = event->GetDayOfYear();
55      tSum[currentDayOfYear - 1] += event->GetTemp();
56      tCount[currentDayOfYear - 1] += 1;
57
58    }
```

Figure 4: `for` loop over the number of events that returns the arrays `tSum[currentDayOfYear-1]` with the sum of all temperatures for a given day and `tCount[currentDayOfYear-1]` with the number of summed events for each day.

deviation `tErr[i]` for each day.

Finally, `TGraphErrors()` is used to plot the average temperature for each day with the standard deviation as error bar (Figure 8.

```
82    // Graph the result
83    TGraphErrors *gr  = new TGraphErrors(n,tDay,tAvg,t0,tErr);
84
85    TCanvas *c1 = new TCanvas("c1","Temperature per Day");
```

Figure 8: `TGraphErrors()` is used to plot the mean temperature of each day with error bars representing the corresponding standard deviation.

Figure 9 and 10 show the plot of the mean temperature using all data and using only confirmed data respectively. The difference is not remarkable.

```
60    // For each day of the year, calculate the arithmetic mean over all years
61    for(Int_t i = 0; i < n; i++) {
62      tAvg[i] = tSum[i] / tCount[i];
63      tDay[i]=i+1;
64    }
```

Figure 5: `for` loop over the days that returns average temperature of each day `tAvg[i]` and fills the `tDay` array.

```
68    for(Int_t i = 0; i < nEvents; i++) {
69      // Update event object
70      eventBranch->GetEntry(i);
71      // If excludeBadData = 1, then exclude deviating data points
72      if(excludeBadData && !(event->GetQCode())) { continue; }
73
74      currentDayOfYear = event->GetDayOfYear();
75      tErr[currentDayOfYear-1] += pow((event->GetTemp()-tAvg[currentDayOfYear-1]),2);
76    }
```

Figure 6: `for` loop over number of events returning the squared deviation from the mean for each day.

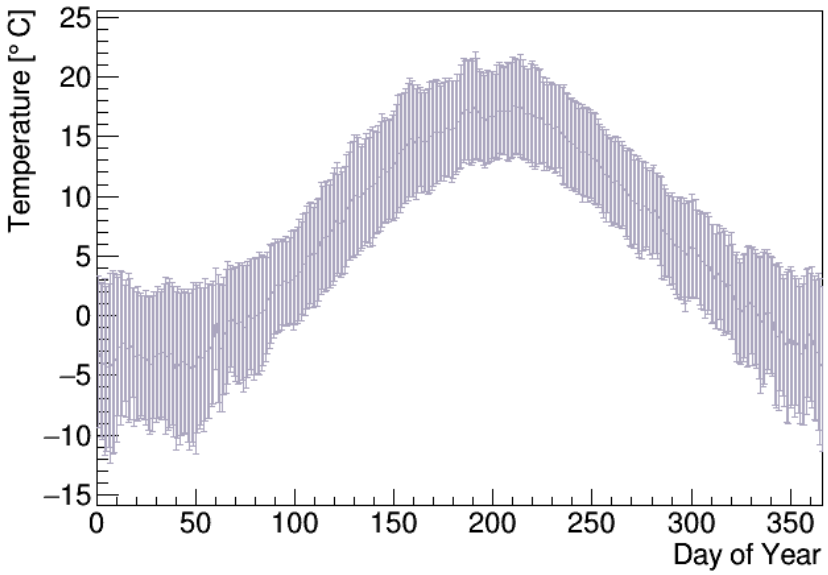

Figure 9: Histogram showing the mean temperature on each day of the year using all data.

```
78    for(Int_t i = 0; i < n; i++) {
79      tErr[i] = sqrt(tErr[i]/ tCount[i]);
80    }
```

Figure 7: `for` loop over the days returning the standard deviation `tErr[i]` for each day of the year.

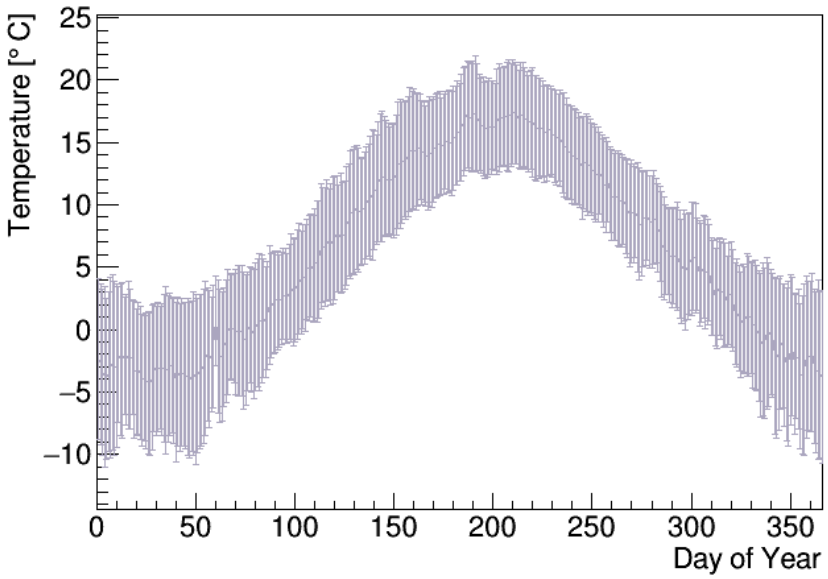

Figure 10: Plot showing the mean temperature on each day of the year considering only confirmed data.

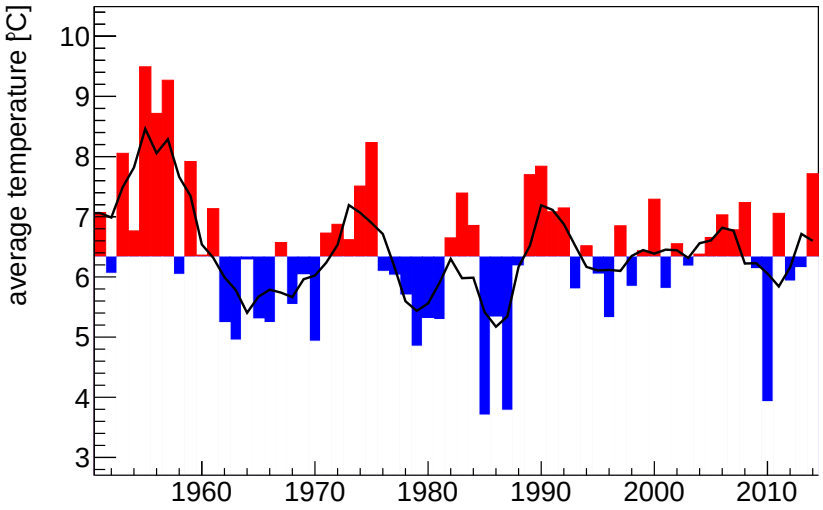

Figure 11: Mean temperatures per year around the average mean temperature per year.

Temperature development

Evaluation of the temperature development over long periods of time can indicate changes of the global climate. However, a single data set, or several data sets from the same region, neglecting other parts of the world, are not sufficient to give conclusive evidence for climate change. On the other hand it might be interesting to look at a single place's temperature development to evaluate whether and how the global climate change effects a region.
The bars in figure 11 depict the deviation from the average mean temperature per year, whilst the trend line is the moving average over five consecutive mean temperatures. Additionally, figures 12 and 13 are displaying similar plots for the extreme temperatures per year.
However, the mean temperature graph (figure 11) shows no significant changes in either direction, nor does figure 12 with the minima. Only the maximum graph (figure 13) adumbrates a change toward warmer temperatures. I assume that more visible changes in temperature had arisen if the temperature recordings had included preindustrial times.

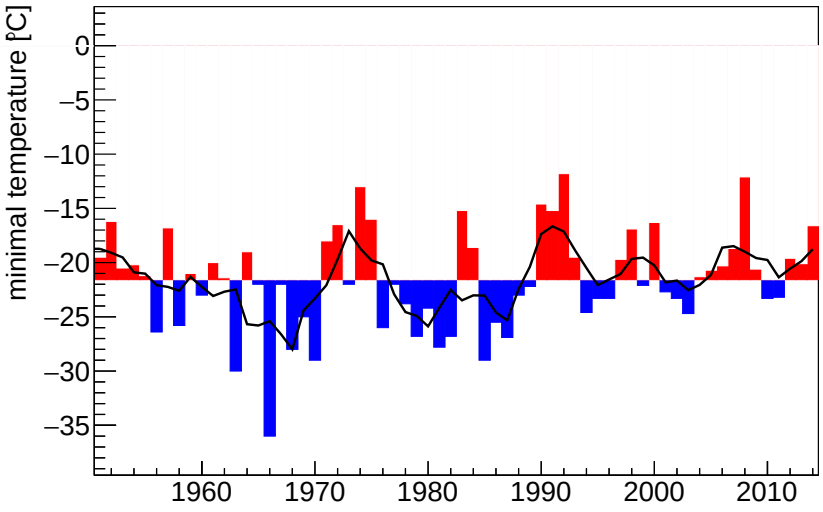

Figure 12: Lowest temperatures per year around the average lowest temperature per year.

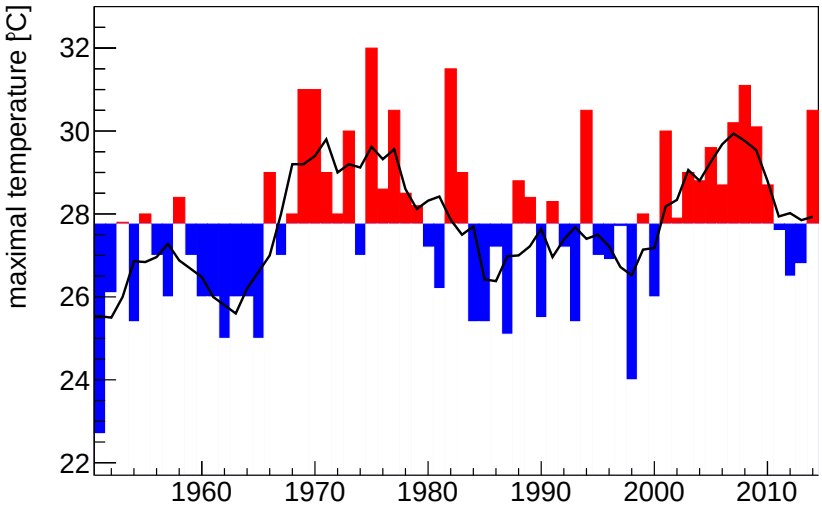

Figure 13: Highest temperatures per year around the average highest temperature per year.

Code snippets

Here I am presenting some interesting parts of the code: I am plotting four graphs on top of each other. (figure 14)

```
77   // create a red, blue and white graph
78   TGraph *rgr  = new TGraph(); //red
79   TGraph *bgr  = new TGraph(); //blue
80   TGraph *wgr  = new TGraph(); //white
81   TGraph *lgr  = new TGraph(); //trendline
82   //TMultiGraph *hs  = new TMultiGraph();
```

Figure 14: Initializing four graphs.

I am starting with the red bars, that depict the temperature of one year each. (figure 15)

```
84    // Calculate the arithmetic mean over the the days and then all years
85    Double_t sum = 0;
86    Double_t count = 0;
87    for(Int_t i = 0; i < totyears; i++) {
88      tAvg[i] = tSum[i] / tCount[i];
89      sum += tSum[i];
90      count += tCount[i];
91      rgr->SetPoint(y1+i, y1+i, tAvg[i]);
92    }
93    Double_t avg = sum/count;
```

Figure 15: Array for the red bars.

Then I add a blue box with height of the average of the displayed red bars. (figure 16)

```
98    for(Int_t i = 0; i < totyears; i++) {
99      bgr->SetPoint(y1+i, y1+i, avg);
100     wAvg[i]=tAvg[i];
101     if(min>tAvg[i]){min=tAvg[i];}
102     if(max<tAvg[i]){max=tAvg[i];}
103     if(type>=0){
104       if(tAvg[i]>avg) {wAvg[i]=avg;}
105     }
106     else{
107       if(tAvg[i]<avg) {wAvg[i]=avg;}
108     }
109     wgr->SetPoint(y1+i, y1+i, wAvg[i]);
```

Figure 16: Arrays for the blue and white graphs.

After that I take the values of the red bars and crop every one that is greater than the average. Those values are my third, white bars. In the end I insert the moving average. (figure 17)

```
111     if(i==0){ lAvg[i] = (tAvg[i]+tAvg[i+1]+tAvg[i+2])/3; }
112     else if(i==1){ lAvg[i] = (tAvg[i-1]+tAvg[i]+tAvg[i+1]+tAvg[i+2])/4; }
113     else if(i==(totyears-2)){ lAvg[i] = (tAvg[i]+tAvg[i-1]+tAvg[i-2]+tAvg[i+1])/4; }
114     else if(i==(totyears-1)){ lAvg[i] = (tAvg[i]+tAvg[i-1]+tAvg[i-2])/3; }
115     else{lAvg[i] = ( tAvg[i-2]+tAvg[i-1]+tAvg[i]+tAvg[i+1]+tAvg[i+2])/5; }
116     lgr->SetPoint(y1+i, y1+i, lAvg[i]);
117   }
```

Figure 17: Array for the moving average.

For negative values red and blue areas have to be switched. (figure 18)

```
121     if(type>=0){
122       rgr->SetFillColor(kRed);
123       bgr->SetFillColor(kBlue);
124     }
125     else{
126       rgr->SetFillColor(kBlue);
127       bgr->SetFillColor(kRed);
128     }
129       wgr->SetFillColor(kWhite);
```

Figure 18: Switching red and blue for minima.