Final project in MNXB01

Jonathan Petersson
Andreas Stokkeland
Martin Korsfeldt
Jonathan Widov

2019-11-17

LUND
UNIVERSITY

Introduction

In this final project in MNXB01 the temperature in Lund from 1961 to 2015 have been analyzed. Four different plots have been made to visualize different results regarding the weather. The work has been done mainly through *github.com* where one member have been appointed release manager and the other group members have forked the repository and done pull requests to update the code.

Results

The temperature of a given day - tempOnDay

The goal for this function is to make a histogram of the temperature of a given day. The function has three arguments: month, day and temperature. The third argument is used for calculating the probability to observe that particular temperature on that day.

The code for the function begins with reading each line in the data file. This can be seen in figure 1.

```
    if(myFile.is_open()) {
        cout << "Reading datafile..." << endl;
        while(getline(myFile, helpString)) {
            vecStrings.push_back(helpString);
        }
    }
    else {
        cout << "Error: not reading the file!" << endl;
        return 1;
    }
```

Figure 1: Adding each line in the data file to a vector.

After each line has been added to a vector, some clean-up has to be done since the first lines only contains text and some lines in the beginning of the data also contains text. How this was done can be seen in figure 2.

```
    // removing the first elements in the vector since they contain text and not data
    vecStrings.erase(vecStrings.begin(), vecStrings.begin() + 12);

    // removing text in the first lines of data
    for(int n=0; n < 12; n++) {
        vecStrings[n] = vecStrings[n].substr(0, 25);
    }
```

Figure 2: Clean-up process of the data.

A new file will then be created that only contains month, day and temperature. This can be seen in figure 3.

After that the new file will get read and only temperatures for the given month and day that gets specified in the input arguments will be added to a new vector, see figure 4.

```
// adding only date and temperature to the new textfile
int vs = vecStrings.size();
int stringLength = 0;
int endofstring = 0;
string date1;
string date2;
string temp;
for(int i=0; i < vs; i++) {
    stringLength = vecStrings[i].length();
    endofstring = stringLength - 22;
    date1 = vecStrings[i].substr(5, 2);
    date2 = vecStrings[i].substr(8, 2);
    temp = vecStrings[i].substr(20, endofstring);
    tempFile << date1 << " " << date2 << " " << temp << endl;
}
```

Figure 3: Adding data to a new file.

```
// while-loop that finds correct data for month and day
int monthNo = 0;
int dayNo = 0;
double temperature = 0.;
vector <double> vecTemp;

while(file >> monthNo >> dayNo >> temperature) {
    if(monthNo == month) {
        if(dayNo == day) {
            vecTemp.push_back(temperature);
        }
    }
}
```

Figure 4: Going through the new file and only adding temperatures for correct month and day.

To represent the data a histogram can be made. With this histogram the mean temperature and the standard deviation can easily be calculated. The possibility to observe a particular temperature can also be calculated, see figure 5. An example is 24/12 (see figure 6) which has a mean temperature of approximately 1.293 degrees Celsius and a standard deviation close to 4.045 degrees Celsius. The possibility to observe -2 degrees Celsius on the 24 of December is approximately 1.935%.

```
// creating the histogram
TH1I* histogram = new TH1I("temperature", "Temperature;Temperature[#circC];Entries", 300, -20, 40);
histogram->SetFillColor(kRed +1);
int vecTempSize = vecTemp.size();
for(int m=0; m < vecTempSize; m=m+1) {
    histogram->Fill(vecTemp[m]);
}

double meanTemp = histogram->GetMean(); // mean value for the temperature of the day
double stdevTemp = histogram->GetRMS(); // standard deviation of the temperature
double integralOfHist = histogram->Integral();
double binx = histogram->FindBin(yourTemp);
double integralOfTemp = histogram->Integral(binx-0.5, binx+0.5);
double probTemp = integralOfTemp/integralOfHist;

TCanvas* canvas = new TCanvas("tempOnDay", "tempOnDay");
histogram->Draw();
```

Figure 5: Creating a histogram for the temperature of the given day.

The histogram which is the end product of this function can be seen in figure 6. Possible improvements for this function is the calculation of the possibility to observe a particular temperature. Since there are gaps in the histogram some particular temperatures will have zero percent possibility to be observed since the integral for that area of the histogram will be zero. This is the case for example -10 degrees Celsius which is of course not reasonable with reality. To avoid this perhaps some fit would have to be done since it will be continues and no gaps would occur.

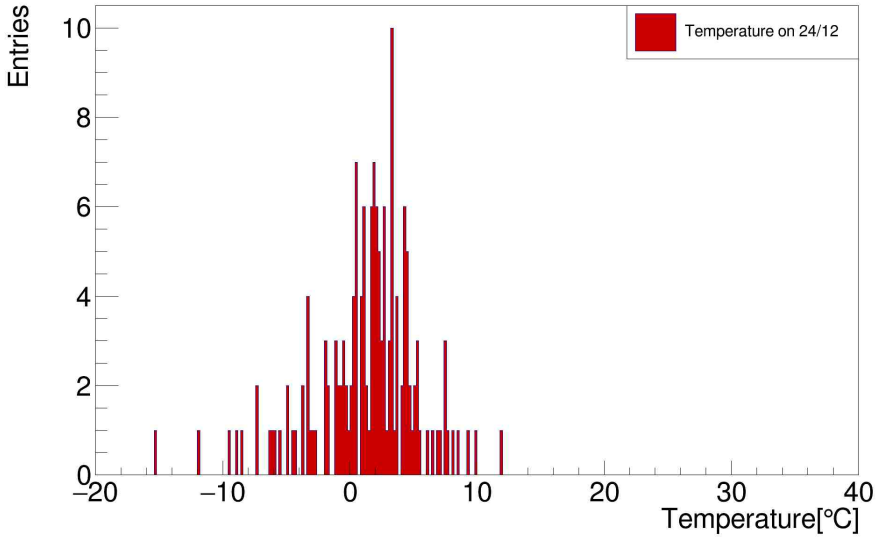

Figure 6: Histogram showing the temperature of the given day.

The mean temperature for a specific date

In this part a function called MeanTempAndTempOnDate was introduced. The goal for this function was to be able to give a year, month and day and get the mean temperature and a histogram of the values as output. In order to do this I used most of the code that was introduced in the previous section. In addition I introduced the year as a variable. This made it possible for the user to call for a specific date and see the mean temperature for that day. When I fixed with this part I also printed some other values as seen in the picture of the code below.

Below is also an example of the kind of histogram that the code would give. When doing a code there could always be some improvements. In this case the histogram could possibly be manipulated so that it showed time on the x-axis and temperature on the y-axis. There could also be a window in the histogram showing the mean value of the temperature. That would in the end give a more intuitive picture of how the temperature varied during the given date.

```
199
200
201       int L = 0;
202       L = vecTemp.size();
203
204       cout << "Length : " << L << endl;
205
206
207       double mean;
208       double tot = 0;
209
210
211       for(int i=0; i < L; i++) {
212
213           tot = tot + vecTemp[i];
214       }
215
216
217
218
219       cout << "Total : " << tot << endl;
220
221
222       mean = tot/L;
223
224       cout << "Mean : " << mean << endl;
225
226
227
228
229
230
231
232
233
234       file.close();
235
236       // creating the histogram
237       TH1I* histogram = new TH1I("temperature", "Temperature;Temperature[#circC];Entries", 300, -20, 40);
238       histogram->SetFillColor(kRed +1);
239       double vecTempSize = vecTemp.size();
240       for(int m=0; m < vecTempSize; m++) {
241           histogram->Fill(vecTemp[m]);
242       }
243       TCanvas* canvas = new TCanvas();
244       histogram->Draw();
245
246
247   }
248
```

Figure 7: Creating some output for the mean value of the temperature of a given date and then creating a histogram of the different temperatures for that date.

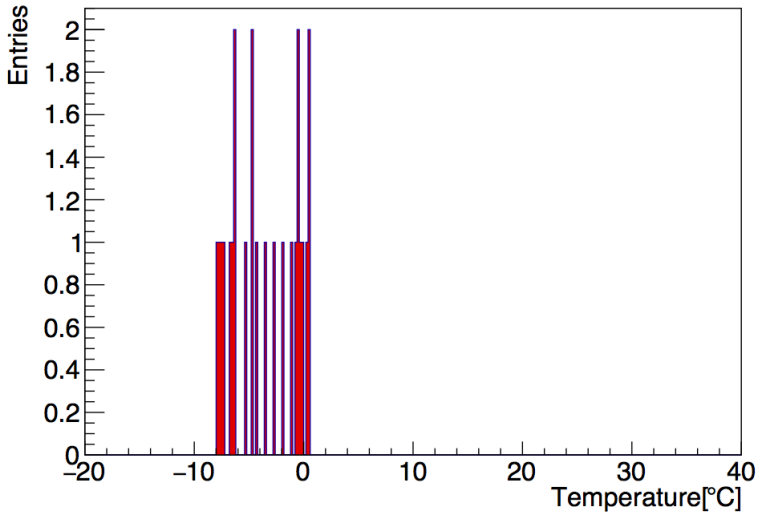

Figure 8: Histogram showing the different temperatures during a specific date.

The Yearly Averages

The main goal of this code is for it to be able to calculate an average temperature of a year and then add it to a graph for comparison. In order to to to this it first must modify the given text file which is what the segment in figure 9 does. The resulting file contains only the years and temperatures.

```
// removing the first elements in the vector since they contain text and not data
vecStrings.erase(vecStrings.begin(), vecStrings.begin() + 12);

// removing text in the first lines of data
for(int n=0; n < 12; n++) {
    vecStrings[n] = vecStrings[n].substr(0, 25);
}

// creating new file
ofstream tempFile("tempFiley.txt");
cout << "Creating new file with only year and temperature..." << endl;

// adding year and date and temperature  to the new textfile
int vs = vecStrings.size();
int stringLength = 0;
int endofstring = 0;
string date0;
string date1;
string date2;
string temp;
for(int i=0; i < vs; i++) {
    stringLength = vecStrings[i].length();
    endofstring = stringLength - 22;
    date0 = vecStrings[i].substr(0,4);
    date1 = vecStrings[i].substr(5, 2);
    date2 = vecStrings[i].substr(8, 2);
    temp = vecStrings[i].substr(20, endofstring);
    tempFile<<date0<<" "<<temp<<endl;
}
```

Figure 9: Modifying the text file to the format the rest of the will use.

Once the *txt* file is created the code runs over the created document summing the temperatures during a year and then dividing by the amount of entries in that year. It then resets the yearly sum and vector before it moves on to the next year and repeats the same process always adding the average to a vector.

```
    vector<double> YearlyTemps;// vector that will contain the average temp for each year
    vector<double>Years;
    double yearsum=0;
    double temper=0.0;
    int yearcounter=1961;
    int yearNo=0;
    int yearlength;
    double YearMean=0.0;
    int daycounter=0;

    //while loop for the entire document
    while(file1 >> yearNo >> temper)
    {
        if (yearNo==yearcounter)
        {
'/      cout<<yearNo << " "<< temper <<endl;  //This was used to verify the file being read in properly.
        yearsum+=temper;
        daycounter++;
        }

        //This segment first calculates the average temperature and the
        // resets the sums before moving on to the next year.

        else
        {
        yearlength=daycounter;
        YearMean=yearsum/yearlength;
        YearlyTemps.push_back(YearMean);
        daycounter=0;
        yearsum=0;
        Years.push_back(yearcounter);
        yearcounter++;
        }
```

Figure 10: Main part of the function where the yearly average temperatures are calculated.

The final bit of the code simply takes the vector created in the previous one and displays it in the graph seen in figure 12.

```
const int n=YearlyTempsize;
double x[n],y[n];

for(int i=0;i<YearlyTempsize;i++)
{
    x[i]= Years[i];
    y[i]= YearlyTemps[i];
}
TGraph* graph = new TGraph(n,x,y);
graph->SetLineColor(2);
graph->SetLineWidth(2);
graph->SetMarkerColor(4);
graph->SetMarkerStyle(20);
graph->SetTitle("Average temperature per year from 1961 to 2015");
graph->GetXaxis()->SetTitle("Year");
graph->GetYaxis()->SetTitle("Temperature( #circC) ");
graph->GetXaxis()->CenterTitle(true);
graph->GetYaxis()->CenterTitle(true);
TCanvas *c1 = new TCanvas("Other stuff"," Yearly Averages ",200,10,700,500);
c1->SetGrid();
graph->Draw();
```

Figure 11: Segment that creates the graph depicting the average temperatures.

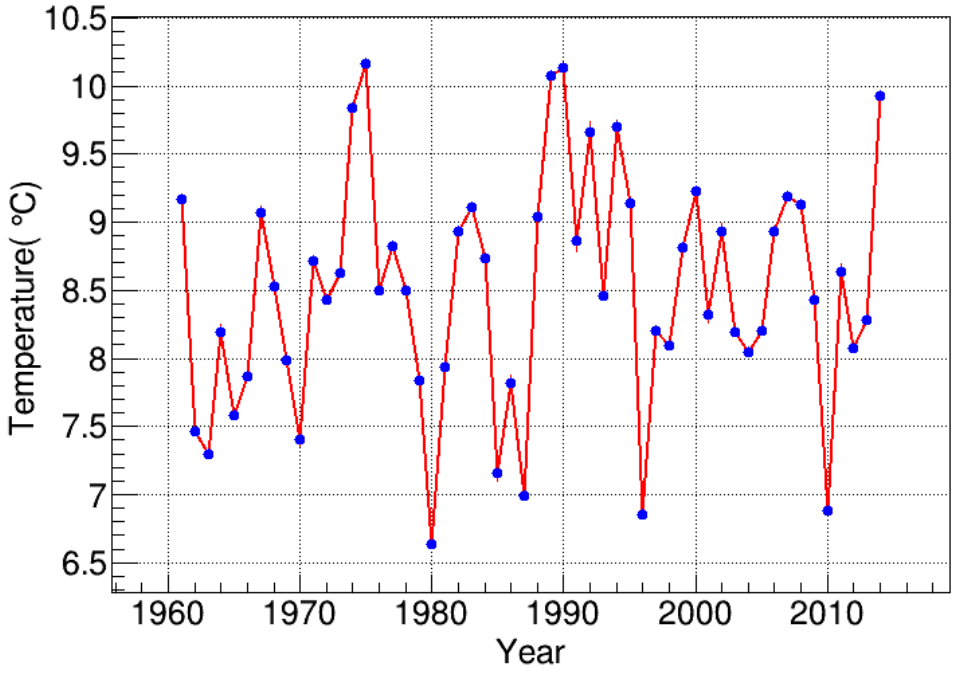

Figure 12: Final graph produced by the code.

The Minimum and Maximum Temperatures

This code is meant to produce a graph for the minimum and maximum temperatures for all the years in the data, as well as the minimum and maximum temperature for a specific year, as well on which day that minimum and maximum fell on. Firstly, all the necessary vectors, lists and variables are created, and the data file is read in. A specific years information is asked for, which is then entered by the user:

```
int tempTrender::MinMax(){
    //defining all the variable we need
    int YearCounter=1961;
    int DesiredYear;
    int YearNo;
    int MonthNo;
    int DayNo;
    int MinimumMonth;
    int MinimumDay;
    int MaximumMonth;
    int MaximumDay;
    double Temperature;
    double YearMin;
    double YearMax;
    //user enters specific year they wish to know min and max of
    cout << "Enter the year you wish to know the minimum and maximum temperature of between 1961 and 2015: "<< endl;
    cin >> DesiredYear;
    int InternalYear=DesiredYear-1961;
    ifstream File("tempFile.txt"); // open file
    //define all the vectors we need
    vector <double> Minvec;
    vector <double> Maxvec;
    vector <int> MinDay;
    vector <int> MinMonth;
    vector <int> MinYear;
    vector <int> MaxDay;
    vector <int> MaxMonth;
    vector <int> MaxYear;
```

Figure 13: All of the lists, vectors and variables that need to be defined.

The following code is what actually checks the data so that each year only gets one maximum and one minimum value before it moves onto the next year. Every time a new minimum or maximum is reached in the data, it overwrites the old maximum or minimum. When the code has reached the end of one year, it enters the minimum and maximum temperatures of that year into the list, as well as the date that the minimum and maximum temperatures occurred on, and then it moves on to the next year.

```
210
211     //adds all temperatures (min and max) and dates to vectors
212     while(File >> YearNo >> MonthNo >> DayNo >> Temperature){
213         if(YearNo == YearCounter){
214             if(YearMin > Temperature){
215                 YearMin = Temperature;
216                 MinimumMonth = MonthNo;
217                 MinimumDay = DayNo;
218             }
219             if(YearMax < Temperature){
220                 YearMax = Temperature;
221                 MaximumMonth = MonthNo;
222                 MaximumDay = DayNo;
223             }
224         }
225         else {
226             //resets when new year comes, and adds things to vectors
227             YearCounter++;
228
229             Minvec.push_back(YearMin);
230             Maxvec.push_back(YearMax);
231
232             MinMonth.push_back(MinimumMonth);
233             MinDay.push_back(MinimumDay);
234
235             MaxMonth.push_back(MaximumMonth);
236             MaxDay.push_back(MaximumDay);
237
238             YearMin=0;
239             YearMax=0;
240         }
241     }
```

Figure 14: The code that fills the vectors with the minimum and maximum temperatures.

The following code converts the vectors for the minimum and maximum temperatures into lists which are then graphed on two separate graphs, which are then joined into one graph with multigraph.

```
245     LF
246     const int NoYears=2015;LF
247     double MinList[NoYears];LF
248     double MaxList[NoYears];LF
249     double YearList[NoYears];LF
250     //converts vectors to lists for graphing purposesLF
251     for (int i=0; i<NoYears; i++){LF
252         MinList[i]=Minvec[i];LF
253         MaxList[i]=Maxvec[i];LF
254         YearList[i]=i+1961;LF
255     }LF
256     //graphing the data that was producedLF
257     TGraph* MinGraph = new TGraph(54, YearList, MinList);LF
258     MinGraph -> SetLineColor(4);LF
259     MinGraph -> SetLineWidth(2);LF
260     LF
261     TGraph* MaxGraph = new TGraph(54, YearList, MaxList);LF
262     MaxGraph -> SetLineColor(2);LF
263     MaxGraph -> SetLineWidth(2);LF
264 LF
265     TMultiGraph *MinMaxGraph = new TMultiGraph();LF
266     MinMaxGraph -> Add(MinGraph);LF
267     MinMaxGraph -> Add(MaxGraph);LF
268     MinMaxGraph -> SetTitle("Minimum and Maximum Temperature per year from 1961 to 2015; Year; Temperature( #circC)");LF
269     MinMaxGraph -> Draw("A");LF
270     LF
271     File.close(); // closing fileLF
272     return 0;LF
273 }LF
274
```

Figure 15: The code that converts the vectors into graphs.

The following graph is what is produced when all of the code above is run.

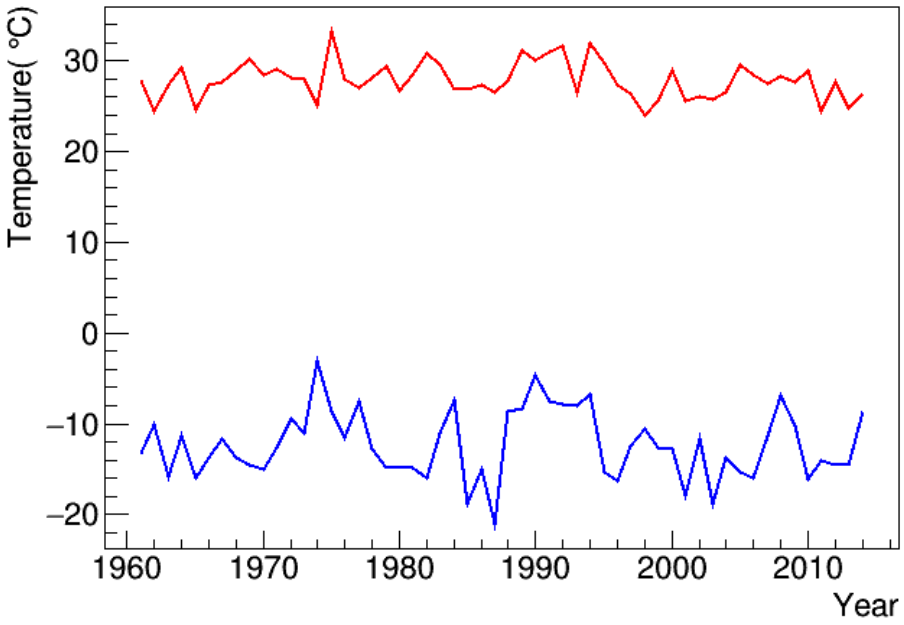

Figure 16: The results from analysing the data.