MNXB01 - Project report

Viktor Drugge, Cameron Robertson, Louise Villander

November 9, 2017

1 Calculating the start of spring

In order to properly calculate at which date spring starts, a definition for the season is necessary. On the webpage of SMHI (https://www.smhi.se/kunskapsbanken/meteorologi/var-1.1080) a meteorological definition of spring is given. Below is the definition of the start of spring from SMHI.

> If the daily average temperature is above 0 °C but below 10 °C, we call this for a day with spring temperature. If this occurs seven days in a row, we say that spring arrived the first of these days. Even if there is a return to lower temperatures then it is still counting as spring.
> ...
> The start of spring can not occur before the 15th of febuary.
> ...
> Spring can, at latest, occur the 31th of July.

Using this definition, the temperature for spring arrival was extracted from the data. The method for reading, extracting, plotting, and, fitting the data is performed in a single method. There exist several ways to implement a solution of the posed question of spring arrival. The choice of using a single method is one of them. The inital set of lines of "springArrive()" initialize variables that will be used when reading the data file "uppsala_tm_1722-2013.dat".

The exact date for when each season start and end are not fixt and may vary a lot between each year. Using only the first paragraph of the above definition dates as late as november could be classified as the beginning of spring. For this reason, we use the last of July as a cutoff point. The beginning of spring may change a lot depending on location. The definition used here was found most promising for the data set of Uppsala.

The main part of the springArrive() method occur in the while loop. It can be summarized in two parts. First check if the spring of the current year is found using the foundSpring variable. If foundSpring is true, the current year is extracted and the file pointer is moved down the file till it finds the next year. Note that when this loop is complete the the next line read will represent the earliest date of next year given in the data file.

```
if(foundSpring == true)
       {
              dayCount = 0;
              foundSpring = false;
              Int_t nextYear = year+1;
              while(nextYear != year)
              {
                     if(getline(file,line, '\n'))
                     {
                            stringstream ssNextYear(line);
                            ssNextYear >> year >> month >> day >> temp >>
                               temp_urban >> id;
                     }
                     else
                            break;
              }
       }
```

If foundSpring is false, the file pointer is pointing at a year which has not yet been registered. In this case the second loop is used. Here, the program try performing a for loop representing the 7 day span required by the definition above. The id of the data is checked with the provided "dataset" variable so to ensure only data from the correct data set is stored. Next, the temperature is examined, it has to be between 0 °C and 10°C. dayCount represent the total number of dates iterated of the year, regardless of id number. This ensures that spring will be identified even if data points might be missing. As a final requirement, the month can not exceed July, in line with the definition of SMHI. At the final iteration of the for loop, the first registered date and temperature is stored. Note the method does not divide the year into 52 weeks but instead look for the first 7 succeeding iterations which satisfy the definition we use.

```
if(foundSpring == false)
       for(Int_t i=0; i < daysWeek; i++)
              {
                     Double_t tmp;
                     getline(file,line, '\n');
                     stringstream ss(line);
                     if(ss >> year >> month >> day >> temp >> temp_urban >> id)
                         //check output can is eligible
                     {
                            dayCount++;
                            if(id==dataset)//Take data from dataset
                            {
                                   //dayCount>=46 represent 15 feb (minimum
```

```
                date for spring), month<8
            remove missing data (otherwise autumn is
                classified as spring)
            if(temp_urban >= 0 && temp_urban <= 10 &&
                dayCount >= 46 && month
            < 8) //Check if temperature fulfill
                definition of spring
            {
                if(i==0)
                {
                    sYear=year; sMonth=month;
                        sDay=day;
                    sTemp=temp_urban;
                }
                if(i == daysWeek-1) //Save temp of
                    first day
                {
                    foundSpring = true;
                    hDays->Fill(dayCount - (
                        daysWeek+1)); //(daysWeek
                        +1),
                    +1 because dayCount
                        incremented by 1
                        previously
                    hTemp->Fill(sTemp);
                    //Print date of spring
                    cout << "Spring found:\t" <<
                        sYear << "\t" << sMonth <<
                         "\t" << sDay << "\t" <<
                        sTemp << endl;
                    springDate << sYear << "\t"
                        << sMonth << "\t" << sDay
                        << "\t-\t" << sTemp <<
                        endl;
                }
            }
            else //If temperature is not in interval,
                start new iteration
                break;
        }
    }
}
```

The date of each spring found is stored in a file "found_spring_date.dat". The registered days are saved in a histogram that span one year. The resulting histogram can be seen in figure 1. Note the sharp start at day 46, representing the 15th of february. The mean of the histogram, day 80, represent the 21 of March if its not a leap year and the 20 of March otherwise. A total of 365 bins are used. One for each day, as seen on the x-axis, and, the y-axis represent the number of entries. For each date registered, the temperature is saved in a histogram, see figure 2 for more details. Here, the x-axis represent the temperature interval and the y-axis the number of entries. The resulting histogram is fitted to a exponential function,

$$f(x) = \alpha e^{-\lambda x}. \tag{1}$$

The curve fit relatively well with the histogram. This is expected as the gradual increase of the temperature each spring should result in the majority of temperatures ending up in the low temperature region.

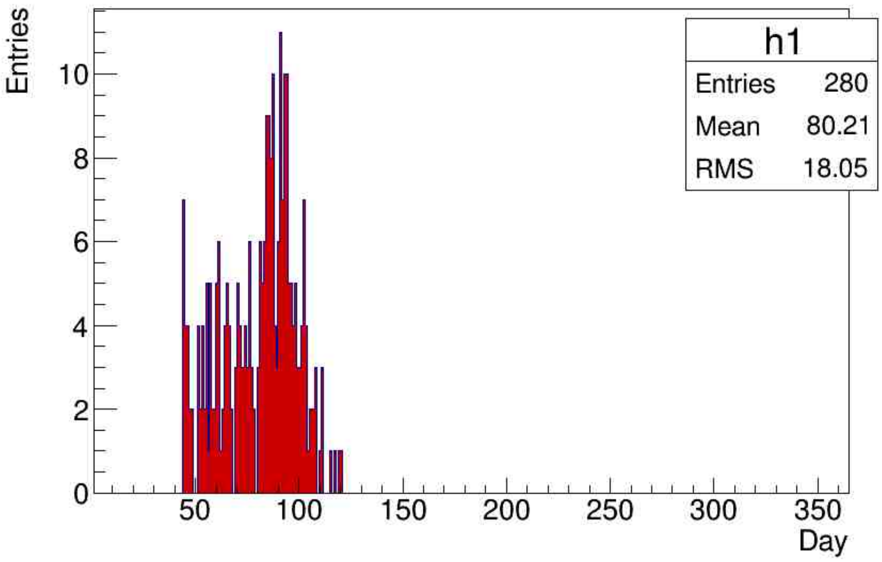

Figure 1: Number of times spring has arrived since the 18th century in Uppsala using the meteorological definition of SMHI. Day 80, the mean of this histogram, represent either March 20 if its a leap year or March 21 if its not a leap year. Note the sharp peak at day 46, representing the 15th of febuary. Dates after the last of July are not included. The histogram consist of 365 bins, one for each day of the year, seen on the x-axis, and, the number of entries on the y-axis. This binning shifts leap years by one day, however the effect is neglectable.

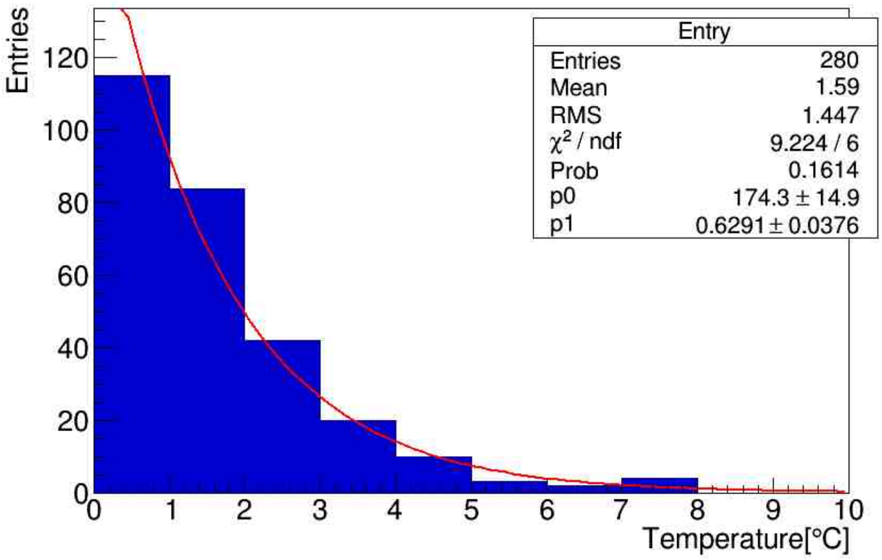

Figure 2: Temperature histogram for all spring arrival dates (blue) and fitted exponential function (red line), see equation (1). The histogram fit relatively well with a exponential function, as expected since the gradual increase of the temperature each year should yield a decaying distribution. Here, $p0$ represent α, and, $p1$ represent λ of equation (1). The x-axis represent the temperature in $^{\circ}$C and the y-axis the number of entries.