CS2035 - Assignment 4 - 2018

Analyzing London Weather Data

Out: March 20th, 2018

In: April 6th, 2018 at 11:55 pm via Owl

Introduction

This MATLAB assignment requires you to write one MATLAB script file, a4.m, to perform a simple (and **primitive**) analysis on London weather data from 1883 to 2006.

You will learn about:

- inputing data into MATLAB from `csv` files;
- processing of a real data set;
- checking data for validity;
- reducing data from shorter (monthly) to longer (seasonal) time scales;
- basic statistical inference based on linear regression results.

This assignment is worth $11\frac{2}{3}\%$ of the course mark.

Introduction

Are you a climate change denier or believer? This assignment requires you to input, process and analyze London weather data from 1883 to 2006. Is the seasonal weather in London becoming warmer? Is there more or less seasonal snowfall, precipitation and rain? You are given the weather data, obtained from data measurements made at the weather stations Lambeth A (1883-1932), London South (1930-1941) and London International Airport (1941-2006). The data is contained in `csv` files obtained from the Government of Canada website http://climate.weather.gc.ca/. This data is contained in three `csv` files:

- `London_South_Monthly_1883-1932.csv`;
- `London_Lambeth_A_1930-1941.csv`; and
- `London_Intl_Airport_Monthly_1940-2006.csv`.

These files are available in the assignment folder.

In addition to a header specifying where the data was obtained from, these files contain the following data entry fields:

1. Year;
2. Month;
3. Mean Maximum Temp (°C) (average max daily value over month);
4. Mean Minimum Temp (°C) (average minimum daily value over month);
5. Mean Temp (°C) (average value over month);
6. Extreme Maximum Temp (°C) (max value over month);
7. Extreme Minimum Temp (°C) (min value over month);
8. Total Rain (mm) (over month);
9. Total Snow (cm) (over month); and
10. Total Precipitation (mm) (over month).

For most months there are measurements available, but in some cases where data was not taken (or was lost), we have `NaN` entires for those fields. Note that you cannot use `NaN` numbers in your calculations (any calculation with `NaN` is `NaN`). Thus, you will need to remove these elements from any calculations you do, which you will see how to do in the instructions below.

Instructions

Step I: Read in the Data

The first step is to read the data into MATLAB, which can be done using the command `csvread`, which reads in data stored in `csv` (comma separated values) format. For each of the three files, you will read them into a matrix of numerical values (`D1`, `D2` and `D3`) using a line of code like

```
D1 = csvread('London_South_Monthly_1883-1932.csv',19,1);
```

Here `19` is the row number where the data starts and `1` is the column number where the data starts (where counting starts from `0`). Use these numerical values for all three files.

Once you have read the data into the three matrices `D1`, `D2` and `D3`, you need to combine these into a single data matrix `D`. If you have read in the three data files correctly, you can merge the data into one set with the following line of code:

```
D = [D1(3:end-8,:); D2(29:end-17,:); D3(8:end-50,:)];
```

This will avoid missing data as much as possible, while providing monthly entries from March 1883 to October 2002. This is the data you will use for your analysis.[1]

Step II: Plot the Data

For each of the eight (8) data fields generate a plot of the data stored in `D`. Using the following procedure you can essentially automate the generation of the plots. First, you will find the following cell arrays useful for generating `ylabel`s and `title`s:

[1] Note that there are some valid values of the total precipitation quantities for years after 2002 in the London International Airport data, but we are ignoring these in our analysis. The only data set including these values would extend are the three total precipitation quantities for the Winter season.

```
ylabels = {'Year','Month','Mean Max Temp (°C)','Mean Min Temp (°C)',...
    'Mean Temp (°C)','Extr Max Temp (°C)','Extr Min Temp (°C)',...
    'Total Rain (mm)','Total Snow (cm)','Total Precip (mm)'};
titles = {'Year','Month','Mean Maximum Temperature (°C)',...
    'Mean Minimum Temperature (°C)','Mean Temperature (°C)',...
    'Extreme Maximum Temp (°C)','Extreme Minimum Temp (°C)',...
    'Total Rain (mm)','Total Snow (cm)','Total Precipitation (mm)'};
```

The `x`-values for the plots can be computed from the first two columns of `D`, which contain the year and the month, respectively. The `x`-values can thus be computed as `year+((month-1)/12)`. The corresponding `y`-values are then each of the remaining columns of `D`, one column for each plot. For column `3` (mean maximum temperature), for example, you could use the following code:

```
figure
plot(x,D(:,3))
xlabel('Year')
ylabel(ylabels(3))
title(strcat('London Ontario'," ", titles(3)))
axis tight
```

By replacing `3` with a variable `n`, then you can construct a loop to make all eight (8) plots. You should change the height and width of the plot so that the data is not too bunched up and difficult to read. Add something like the following line to change the shape of each plot:

```
set(gcf,'Position',[100 100 800 400])
```

Include an additional line to print out each plot, either in `jpg` or `png` format. Use the `ylabels` information in the output filename so that the filename indicates the data plotted in it. For example, for `png` format, the filename for $n = 3$ should read `London Mean Max Temp (°C).png`.

Your plots should look something like the Figure 1 below.

Step III: Look for Linear Trends

In this part you will look for linear trends in the data by season for the four seasons defined as Spring (March–May), Summer (June–August), Fall (September–November) and Winter (December–February). For each season, this task breaks down into the following subtasks:

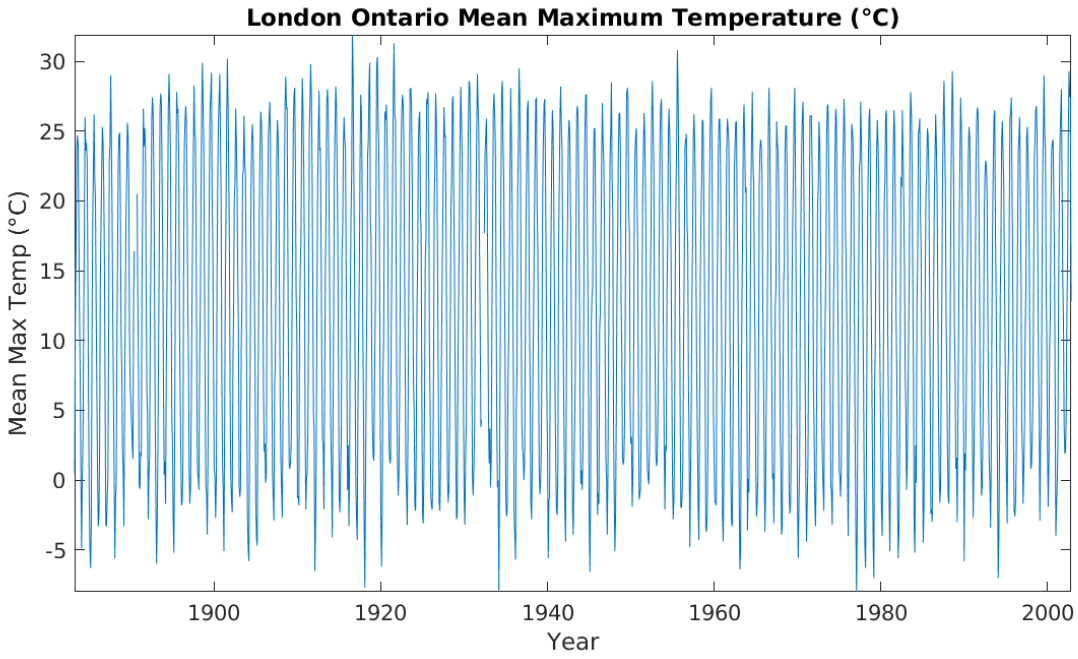

Figure 1: Mean maximum temperature data for London, ON from 1883 to 2002.

1. Compute seasonal values for four (4) of the eight (8) data fields, specifically:

   (a) one field from columns 3–5 (average temperatures);

   (b) column 6 (max temperature);

   (c) column 7 (min temperature);

   (d) one field from columns 8–10 (total precipitation amounts).

   Whichever four fields you choose, make sure to use the same fields for all of the four seasons. Note that you may process more fields if you are interested to see the results, but you do not receive additional credit for doing so.

   To compute the seasonal values you need to keep track of several things:

   (a) the number of years of data for the given season;

   (b) the number of days in the months of the given season (for averages only);

   (c) the kind of quantity measured (average, max, min, total);

   (d) whether the data contains `NaN` values.

To illustrate how this is done, consider the case of column `3` of the data `D` and the Spring season. We use the fact that the data starts in March of 1883, the beginning of Spring as we have defined it. First we need to determine the number of Spring seasons, which we can do in the following way. To find the number of full years of data we use

```
n_years = floor(size(D,1)/12);
```

and we add one more year if there are at least three more months of data (corresponding to March, April and May of the final year of data):

```
if (rem(size(D,1),12) >= 3)
    n_years = n_years+1
end
```

Note that for Summer, and Fall, the '`>= 3`' is replaced by '`>= 6`', '`>= 9`', respectively. The adjustment does not need to be done for Winter (Why?).

Continuing with the Spring example, the next step is to create a loop to compute the data values for the Spring season. The aim here, for each field, is to compute two vectors, `years`, containing the years that have viable data, and `data`, the corresponding data values for those years. We will assume that the first year always has valid data, which it does for our data set. Thus, we can extract the first year with the code

```
years = D(1,1);
```

and then compute the first data value. How we do this depends on whether we are computing an average, a maximum, a minimum or a total. In the case of fields `n=3`, `n=4` and `n=5`, where averages are computed, for the Spring season we would use the code

```
data = (D(1,n)*31 + D(2,n)*30 + D(3,n)*31)/(31+30+31);
```

to get a scaled average over March (31 days), April (30 days) and May (31 days). In the case of field `n=6`, where a maximum is computed, we would use the code

```
data = max(D(1:3,n))
```

to compute the maximum for the season, and similarly for `n=7`, where a minimum is computed (replacing `max` with `min`). In the case of fields `n=8`, `n=9` and `n=10`, where totals are computed, we would use the code

```
data = sum(D(1:3,n))
```

to compute the total for the season.

We then need a loop to compute the values for the other years that have valid data for all three months of the season. In the case of `n=3`, `n=4` and `n=5`, we can use the code

```
for i=1:n_years
    first_element = 1+i*12;
    last_element = 3+i*12;
    yrdata = D(first_element:last_element,n);
    if (~any(isnan(yrdata)))
        years(end+1,1) = D(first_element,1);
        data(end+1,1) = (yrdata(1)*31 + yrdata(2)*30 + yrdata(3)*31)/(31+30+31);
    end
end
```

You should add comments to indicate that you know what each step in this code does.

In this code `first_element` and `last_element` are the indices of the start of the Spring season and the end of the Spring season. The vector `yrdata` is therefore the list of data values for the current season and current year. We need to check that none of the data elements are `NaN`, which is done with the statement `if (~any(isnan(year_data)))`. If there are no `NaN` values, then we add that year to the `years` vector and add the corresponding data value to the `data` vector. Note that the expression setting the `data` value only works for `n=3`, `n=4` and `n=5`, where an average is computed. For other values of `n`, it needs to be replaced with the appropriate expression to compute the maximum, minimum or total of the `yrdata` vector.

2. Compute a linear regression for each of the four (4) data fields you have selected to analyze. Use the `regress` function to compute a linear regression on each `years` and `data` vectors. Do this as we did in lecture, by having `regress` compute a y-intercept and

slope and returning a vector `[b bint]`, where `b` contains the y-intercept as `b(1)` and the slope as `b(2)`, and `bint` contains the 68% confidence interval for the y-intercept (first row) and slope (second row) of the regression.

Once you have computed the regression, plot the regression line on top of the data you computed. You can do this with the code:

```
figure
plot(years,data)
axis tight
hold on
xlabel('Year')
ylabel(ylabels(n))
title(strcat('London Ontario Spring'," ", titles(n)))
plot(years,b(2)*years+b(1),'LineWidth',2)
hold off
```

Your plots should look something like the plot in Figure 2

3. Analyze the results of the regression for each of the four (4) fields you selected. For this you must find answers to the following questions about the results of the regression:

   (i) Does the regression indicate an increasing (decreasing) trend in the quantity, i.e., is the slope positive (negative)?

   (ii) Is this trend significant at a 68% confidence level (one standard deviation)?

   (iii) Is this trend significant at a 95% confidence level (two standard deviations)?

   (iv) If you find a significant trend, what is the slope in °C/century, mm/century or cm/century?

   You can answer these questions based on the content of the vectors `b` and `bint` computed in your regression.

   Whether the trend is increasing is just determined by the sign of the slope.

   Answers to questions (ii) and (iii) are determined by the width of the confidence interval, namely whether you can rule out a zero slope, at a given confidence level. Since `regress`

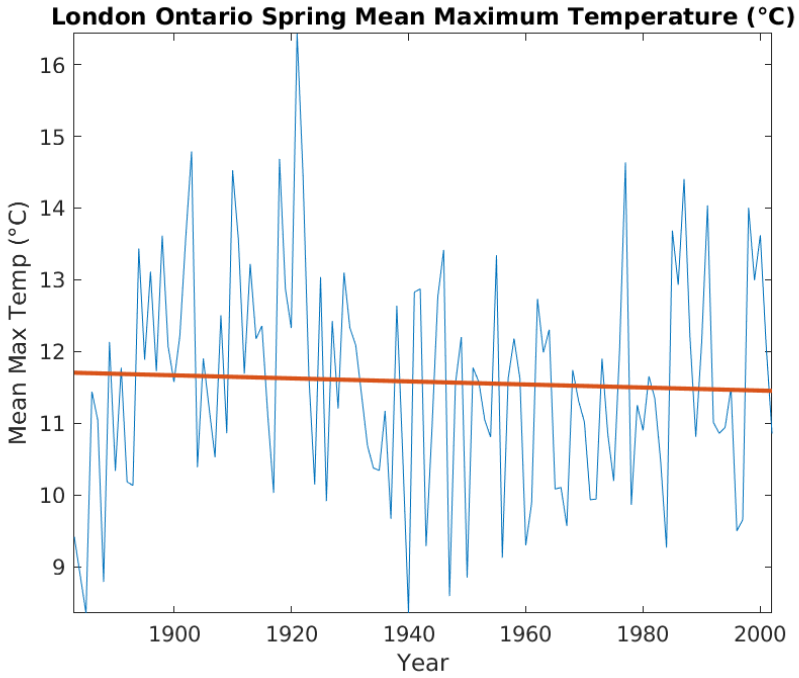

Figure 2: Mean maximum temperature Spring trend for London, ON from 1883 to 2002.

returns a 95% confidence interval by default, for question (ii) you need to make sure to change the confidence level `regress(y,X,alpha)` uses by modifying the third argument `alpha` appropriately. We used this third argument in lecture and it is described in the help file for `regress`. Once you have set the correct `alpha` for 68% confidence, you need to check whether the interval returned in `bint` crosses zero. If it does cross zero, the trend is not statistically significant at the 68% confidence level. If it doesn't cross zero, then you have a significant trend at the 68% confidence level.

If the trend is not significant at 68% confidence, it will not be at 95% either. If it is significant at 68%, however, you will need to compute the 95% confidence interval. This is done simply by widening the interval by another standard deviation. Since the standard deviation `sigma` is the absolute value of the difference between the ends of the 68% interval and the value of the slope, you can compute `sigma` as

```
sigma = abs(b(2) - bint(2,1))
```

For any significant trends you find, increasing or decreasing, note the number of °C/century,

mm/century or cm/century, as appropriate, which is just the slope multiplied by 100. You may wish to compute all of this information automatically by appropriately modifying the following code:

```
if (slope>0)
    disp(strcat('Potential increasing trend for'," ",titles(n),':'))
elseif (slope<0)
    disp(strcat('Potential decreasing trend for'," ",titles(n),':'))
end
if (sign(left_conf_bound) == sign(right_conf_bound))
    disp('  Significant trend at 68% confidence!')
    sigma = abs(slope-left_conf_bound);
    fprintf('  Trend is %f +/- %f ',b(2)*100,sigma*100);
    if 3<=n && n<=7
        fprintf('°C/century\n');
    elseif n==8 || n==10
        fprintf('mm/century\n');
    elseif n==9
        fprintf('cm/century\n');
    end
    if (sign(left_conf_bound-sigma) == sign(right_conf_bound+sigma))
        disp('  Significant trend at 95% confidence too!')
    end
else
    disp('  Trend is not statistically significant')
end
```

where `slope` is the slope, and `left_conf_bound` and `right_conf_bound` are the left and right confidence bounds for the slope, computed by `regress`. You should add comments to show that you understand what each significant line of this code does.

The output of this code for the example we have been considering throughout is

```
Potential decreasing trend for Mean Maximum Temperature (°C):
  Trend is not statistically significant
```

Things to keep in mind when extending your code to the other seasons:

- You need to adjust the number of days in each month for the fields that are computed as a mean value. For the remaining seasons:
  - Summer season: June 30 days, July 31 days, August 31 days;
  - Fall season: September 30 days, October 31 days, November 30 days
  - Winter season: December 31 days, January 31 days, February 28 or 29 days.

  To compute the Winter season averages correctly you need to account for leap year. If you get stuck on how to do this, use 28, and come back to it later. It may help to know that 1884 was a leap year, and leap years happen every four years. Thus, you may find it useful to compute the remainder of the given year for division by 4, using something like `rem(year,4)` to determine whether the given `year` is a leap year.
- Remember to recompute the number of years `n_years` with data entries for each season, since different seasons might have different numbers of years of data available.
- Remember to change the starting rows for each season (Spring starting at row 1, Summer starting at row 4, Fall starting at row 7 and Winter starting at row 10) and to adjust the computation of `first_element` and `last_element` accordingly.

Part IV: Discussion

Include a brief discussion of the results of your analysis as a new subsection in your script file a4.m. Thus, your discussion section should look something like the following:

```
%% Discussion of Results
% This is where the discussion will appear. This will be prose where you
% explain what the results say, and respond to any questions asked in the
% assignment sheet. Add as many lines, preceeded with % signs as you need.
```

This discussion should contain the following:

- For each season, and for each of the four fields you choose to analyze, indicate which show and increasing trend and which a decreasing trend. Considering the results for one season at a time, do the trends for that season taken together indicate a meaningful shift

in weather in the 100 or so year period, i.e. are the suggested trends consistent with one another (to your state of knowledge of meteorology)?

- Point out which, if any, of these trends are statistically significant at a 68% confidence level.
- After doing these first two things for each of the seasons, take note of which trends were significant at a 95% or higher confidence level. If you treat only these 95% trends as significant, with the other results being inconclusive, do they provide a consistent picture of a shift in weather?
- Make an overall brief assessment about whether you can conclude anything about a change in London's weather over the last 100 years or so given your analysis.

Submission

Finally, write all of this code in a script file a4.m, and publish the result of running your code to produce a4.pdf. Make sure that all of the plots you produce appear in your published output (this requires you to plot them in separate figures). Submit your source file, published pdf output and images all together as a zip file a4.zip to OWL when you are finished.

Make sure to start your file a4.m with an appropriate title and comment with your name and student number as you have for previous assignments! Also insert section headings (starting with %%) and comments at appropriate places in your code both to make your code easy to understand and to produce a nice sectioning of your code when you run `publish`.