

ELEN3007 Probabilistic System and Signal Assignment

October 26, 2017

1 ELEN3007 ICAO-Codes-Weather-analysis

This Script imports the weather data for the Chulman Airport (UELL) weather station, in Russia and then answers question outlined. Each question that has a discussion of results is analysed in the respective question markdown block before the code block.

The Git repo, storing this code, can be seen here: <https://github.com/Solidarity/ICAO-Weather-Analysis-Python>

This main file, showing all code execution, can be seen here: <https://github.com/Solidarity/ICAO-Weather-Analysis-Python/blob/master/MainFile.ipynb>

Both a PDF version and HTML version of this report are included in this submission. For an optimal viewing experience, please view the HTML version as the formatting is persistent from the Jupyter notebook.

Please note that this file was generated using the built in Jupyter notebook Latex export function. As a result, some graphics are not included and so to view all content presented in this report, please view the HTML version also included in the submission.

Import libraries and such needed for program execution.

```
In [1]: import numpy as np
import seaborn as sns

import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf

import scipy.stats
import matplotlib.pyplot as plt
from IPython.display import HTML, display

from io import BytesIO
from base64 import b64encode
import scipy.misc as smp

from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator
```

```
from scipy.stats import norm

plt.rcParams['figure.figsize'] = (24, 6)
```

First, define some useful functions. These are used in the printing of data later on.

```
In [2]: def printMatrix(data): #used to print matrices to HTML
        display(HTML(
            '<table><tr>{}</tr></table>'.format(
                '</tr><tr>'.join(
                    '<td>{}</td>'.format('</td><td>'.join(
                        str(_) for _ in row)) for row in data)
            )
        ))

def printText(text):
    display(HTML('<p>' + text + '<p>'))

def displayHTML(html):
    display(HTML(html))

def drawImg(img):
    b = BytesIO()
    img.save(b, format='png')
    displayHTML("<img src='data:image/png;base64,{0}' />"
                .format(b64encode(b.getvalue()).decode('utf-8')))
```

1.1 Question 1

Import data from text files. These are stored as csvs in the "Data" directory in the repo. This CSV is formatted as: {Unit Timestamp, max Temp, avg Temp, min Temp}. Each CSV is read into a matrix. These matrices are then added to a vector so they can be iterated through later on.

```
In [3]: w1995 = np.genfromtxt('Data/1995.csv', delimiter=',')
        w2000 = np.genfromtxt('Data/2000.csv', delimiter=',')
        w2005 = np.genfromtxt('Data/2005.csv', delimiter=',')
        w2010 = np.genfromtxt('Data/2010.csv', delimiter=',')
        w2015 = np.genfromtxt('Data/2015.csv', delimiter=',')

        weatherData = [w1995, w2000, w2005, w2010, w2015]
```

1.2 Question 2

Next, identify the minimum, maximum, mean and standard deviation for each year. As the matrices are in a vector, we can do this sequentially in a loop. These outputs are produced in a matrix, where the columns are the years (1995,2000,2005,2010,2015) and the rows are the minimum, maximum, mean and standard deviation for each year. This can be seen in the table below.

1.2.1 Comment on findings

The minimum, maximum, mean and standard deviations are very similar for each year. This is to be expected as they were taken from the same location. However, as time went on over the past 20 years, it seems that the averages became warmer with both the maximums and minimums increasing over time. There are outliers in this trend as seen in 2010's minimum temperature. This trend is also verified by looking at the mean where each year is shown to be getting warmer.

```
In [4]: dataValues = np.zeros((4, 5))
        counter = 0;
        for year in weatherData:
            dataValues[0, counter] = year[:, 3].min() #max of max values
            dataValues[1, counter] = year[:, 1].max() #min of min values
            dataValues[2, counter] = round(year[:, 2].mean(), 2) #average of average values
            dataValues[3, counter] = round(year[:, 2].std(), 2) #Standard deviation of average
            counter = counter + 1;
        # printMatrix(dataValues)

        displayHTML(pd.DataFrame(dataValues,
                                   columns=['1995', '2000', '2005', '2010', '2015'],
                                   index=['Minimum', 'Maximum', 'Mean', 'Standard Deviation']
                                   ).to_html())
```

<IPython.core.display.HTML object>

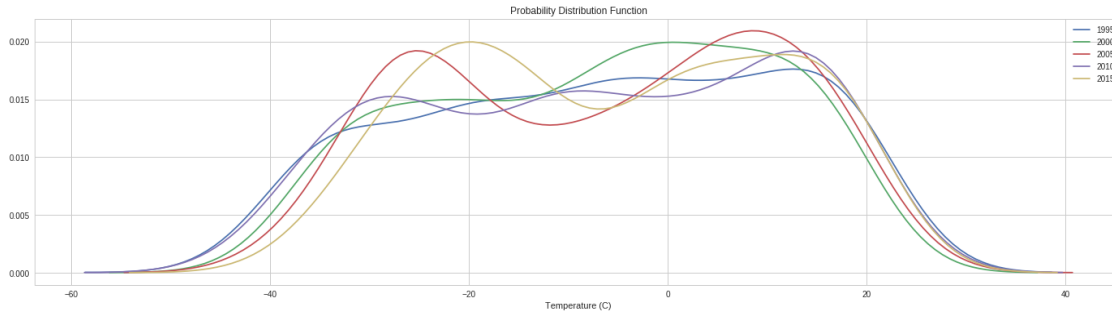
1.3 Question 3

Each Year's probability distribution functions are now plotted, for each year, on the same set of axes. To achieve this, a fixed plot of univariate distributions is generated. This is done with the Seaborn displot function.

1.3.1 Comment on findings

Each year has a similar general probability distribution function. They appear to be close to normal in shape. General average trends can also be seen through this graph such as 2005 had higher distribution of both highs and lows, with a lower distribution in the middle. This indicates more extreme weather during this year.

```
In [5]: sns.set_style('whitegrid')
        counter = 0;
        for year in weatherData:
            sns.distplot(year[:, 2], hist=False,
                          label=1995 + counter * 5,
                          axlabel="Temperature (C)")
            counter = counter + 1;
        plt.title('Probability Distribution Function')
        plt.show()
```



1.4 Question 4

The cross-correlation between each year's annual temperatures is now calculated. This is shown as a matrix output. Note here that we are slicing the data on each year to ignore the leap year in 2000. This is done as the correlation needs matrices of the same dimension.

Additionally note that we need to normalise the data. This is the same as using the numpy `corrcoef` function.

1.4.1 Comment on findings

Through the cross-correlation of the presented data, one can see the relationship between each year's weather. The closer these values are to 1, the more closely correlated the data is. The highest correlation is seen between 2010 and 1995, interestingly enough, with a 94% correlation. The main diagonal of 1's indicates the full correlation, with each year correlated against its self.

```
In [6]: autoCorrelation = np.zeros((5, 5))
        for x in range(0, 5):
            for y in range(0, 5):
                a = weatherData[x][0:365, 2]
                b = weatherData[y][0:365, 2]
                a = (a - np.mean(a)) / (np.std(a) * len(a))
                b = (b - np.mean(b)) / (np.std(b))
                autoCorrelation[x, y] = np.correlate(a, b)
                columns=['1995', '2000', '2005', '2010', '2015']
                index=['1995', '2000', '2005', '2010', '2015']
        displayHTML(pd.DataFrame(autoCorrelation,
                                columns,
                                index).to_html())
```

<IPython.core.display.HTML object>

1.5 Question 5

The autocorrelation function for each year's data, where τ ranges from 0 to 364 is now generated.

Confidence intervals are drawn as a cone. By default, this is set to a 95% confidence interval, suggesting that correlation values outside of this cone are very likely a correlation and not a statistical fluke.

First, each Autocorrelation is drawn on its own graph. After, They are overlayed.

1.5.1 Comment on findings

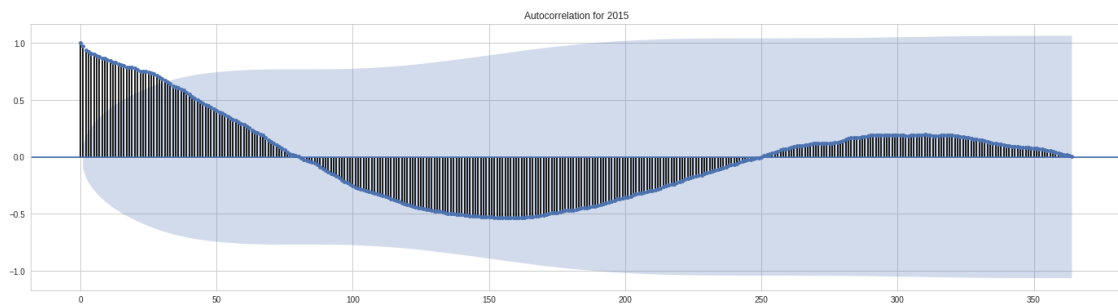
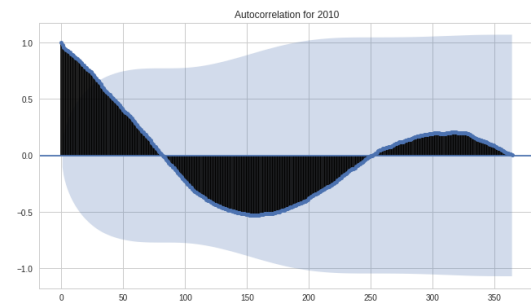
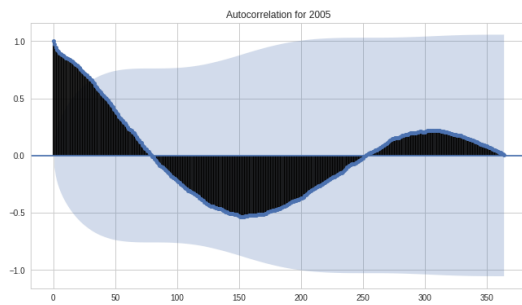
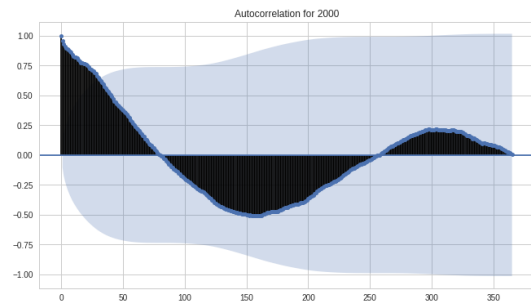
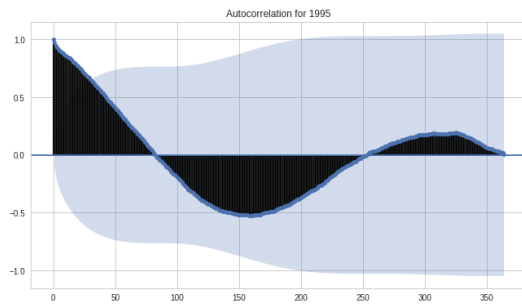
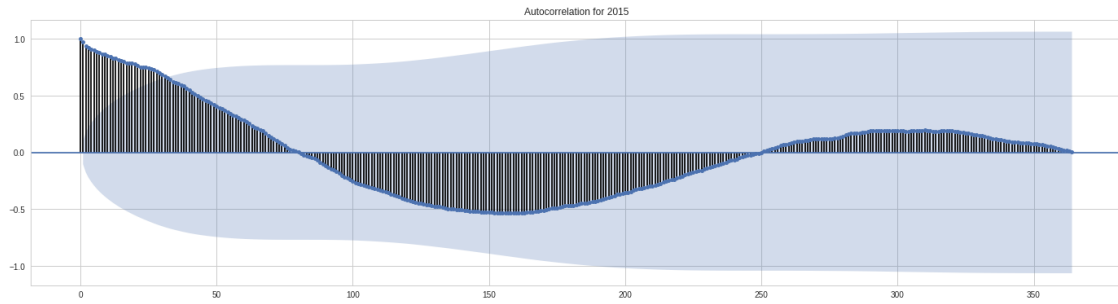
This autocorrelation shows how each year is related to a time-shifted version of itself. This can be seen as an effective convolution process. From this, one can see how each part of the year is related to the rest of the year for a given shift amount τ . For example, at $\tau=0$, the magnitude is 1. At this time, no shifting has occurred. At approximately $\tau=150$, there is the highest negative correlation seen. This is due to the correlation between the middle of summer and winter of the two data sets.

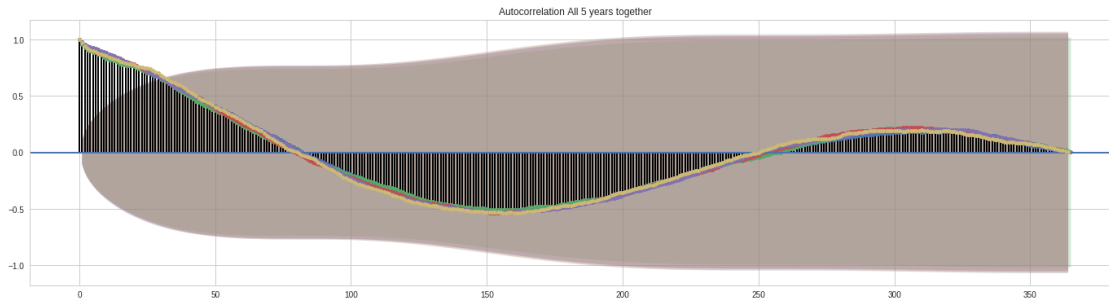
```
In [12]: counter = 0;
        for year in weatherData:

            if (counter == 0): #on the first loop, generate the subplots
                fig, axs = plt.subplots(1, 2)
            if counter == 4: # if we are at the last figure, we want it to be on its own line
                plot_acf(year[:, 2],
                        title="Autocorrelation for {}".format(1995 + 5 * counter))
                plt.show()
            else:
                plot_acf(year[:, 2],
                        title="Autocorrelation for {}".format(1995 + 5 * counter),
                        ax=axs[counter % 2])
            if counter % 2 == 1: # every two figures, we need to generate a new row
                plt.show()
                if (counter < 2): #a new sub plot is needed on second row
                    fig, axs = plt.subplots(1, 2)
            counter = counter + 1

        counter=0;
        fig, axs = plt.subplots(1, 1)
        for year in weatherData:

            plot_acf(year[:, 2],
                    title="Autocorrelation All 5 years together".format(1995 + 5 * counter),
                    ax=axs,
                    label="hello")
        plt.show()
```





1.6 Question 6.a & 6.b

Next, each year, temp is broken down into subdivisions in the range:

$$[minimum - 0.1 = t_0, maximum + 0.1 = t_{10}]$$

into ten equal intervals. This is then used to generate 10 intervals, as:

$$[[t_0, t_1], \dots, [t_9, t_{10}]]$$

1. binColours are predefined RGB values, given in the question
2. The linspace function below is used to generate a set of numbers with equal distributions between the min and max data for each year.
3. Digitisation is used to put the data into the respective buckets. This can then be used later on when the graphs are drawn
4. generate a matrix to store the image colour values
5. iterate over each day of the year*2. double as we need to have two rows for each day(high and low)
6. set the current iteration-1 row of pixels to the low digitised index value from colour matrix provided.
7. same as above, but for the current iterator for the high values provided
8. convert matrix to image
9. draw image to screen

The output below this code shows first the bins for each year, for question 6.a. Then, the blankets are drawn for question 6.b

```
In [8]: binColours = [[0.139681, 0.311666, 0.550652], [0.276518, 0.539432, 0.720771],
                      [0.475102, 0.695344, 0.802081], [0.670448, 0.803486, 0.824645],
                      [0.809791, 0.848259, 0.777550], [0.861927, 0.803423, 0.673050],
                      [0.830690, 0.667645, 0.546349], [0.742023, 0.475176, 0.424114],
                      [0.613033, 0.281826, 0.306352], [0.450385, 0.157961, 0.217975]]

counter = 0;
for year in weatherData:
    binsLow = np.linspace(year[:, 3].min()-0.1, year[:, 3].max()+0.1, num=10)
    binsHigh = np.linspace(year[:, 1].min()-0.1, year[:, 1].max()+0.1, num=10)
```

```

digitizedLow=np.digitize(year[:, 3],binsLow) #put the data into the bins
digitizedHigh=np.digitize(year[:, 3],binsHigh)

rows= 2*len(year[:,3])
image = np.zeros((rows, 451, 3)) #make a matrix to store the image

for x in range(0,rows,2): #iterate over each year's values from the above values and
    image[x-1,0:451]=binColours[int(digitizedLow[int(x/2)])]
    image[x,0:451]=binColours[int(digitizedHigh[int(x/2)])]
printText("<h2>Tempreture blanket for year: {}".format(1995+5*counter))

displayHTML(pd.DataFrame(np.column_stack((binsLow.reshape(10,1),
                                           (binsHigh.reshape(10,1)))),
                           columns=["Low bin","High bin"]).to_html())

outputImage = smp.toimage(image)
outputImage.save('WeatherBlanketsOutput/{}.png'.format(1995+5*counter))
drawImg(outputImage) #Draw image to screen, using custom draw function to put output
counter=counter+1;

```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

1.7 Question 7

Next, a surface plot of a stochastic process is generated. For this, averages are generated over all 5 sets of data for each year, as well as a standard deviation for each day. Next, a linear space is generated representing the possible temperature ranges. finally, the PDF is found then plotted in 3D.

```
In [9]: plt.rcParams['figure.figsize'] = (24, 6) #Make the figure for this question bigger
```

```
#define the variables to store the mean, std deviation, range for each day(temp), pdfs a
meanDay=np.zeros((365,1))
stdDay=np.zeros((365,1))
dayArray=np.zeros((5,1))
pdf=np.zeros((365,365))
dayRange=range(0,365,1)

for day in dayRange:
    for inx, year in enumerate(weatherData):
        dayArray[inx,0]=year[:, 2][day]
        meanDay[day]=dayArray.mean() #calculate the mean
        stdDay[day]=dayArray.std() # calculate the standard deviation

#generate a linear space of all days in the region, for the min to max temp
temperatureSpace=np.linspace(meanDay.min(),
                               meanDay.max(),
                               len(weatherData[0][:,2]))

#iterate over the days again, now generating the pdf
for day in dayRange:
    pdf[day,:]=norm.pdf(temperatureSpace,
                        meanDay[day],
```

```

stdDay[day])

#convert the values to a meshgrid (return coordinate matrices from coordinate vectors)
tempretureSpace, dayRange = np.meshgrid(tempretureSpace, dayRange)

#finally, plot it as a 3d surf
fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(tempretureSpace,
                        dayRange, pdf,
                        cmap=cm.coolwarm,
                        linewidth=0,
                        antialiased=False)

ax.set_xlabel('Temperature (C)')
ax.set_ylabel('Days')
ax.set_zlabel('f(x)')

#add a colour bar on the side to see magnitudes
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.title('Surface Plot of Stochastic Process')
plt.show()

```

