Alain Gasco 10114980

Mphys lab book

Sem 1 project: IAGOS flight dust

Tuesday 06/10/2020

Sent email to Gary and Martin to introduce ourselves for mphys project

Wednesday 07/10/2020

Tasks:

1. Read up on Beswick et al. that describes bcp backscatter cloud probe

bcp.pdf

- It only detects particles greater than 5 microns but less than 75 microns

2. Read up on Petzold et al. that combines bcp and water vapour data to look at presence and characteristics of high altitude ice clouds

tropospheric water vapour interaction wit

3. register on IAGOS data portal!!!

https://www.iagos.org/

information on iagos data sets:

http://www.iagos-data.fr/portal.html

4. read up on previous mphys report

Analysis of a 3D Global Cloud Data Se

Zoom meeting at Friday 3pm 09/10/2020

https://zoom.us/j/95346142817

Friday 09/10/2020

Zoom meeting with project supervisors – introduced project and ourselves + ideas to do with project

Meeting notes:

-do literature review

-understand how bcp probe works

-research about dust events

Friday 16/10/2020

Next zoom meeting Friday 3pm 23/10/2020

Wednesday 21/10/2020

Paper providing background on vertical structure of dust in atmosphere following dust events

acp-18-17655-2018.pdf

Papers from merren:

Three papers that discuss characteristics of deposited dust in Doha and one in Kuwait City

Dust fallout Kuwait.pdf

Structural and physical properties of

Characterization of dust accumulated on

Investigation on deposited dust fallou

Can maybe discuss mineralogy with merren and properties

Friday 23/10/2020

Zoom meeting:

https://zoom.us/j/502489451

meeting notes:

Download BCP data in NASAames format

- ➔ Time
- ➔ Longitude and latitude
- ➔ Altitude
- ➔ Particle concentration
- ➔ Tail number

-Time is in seconds from midnight!

-Cloud presence = -9999 means the probe is not turned on

-Equipment not turned on till ~ 300m

-From tail number – can tell rolls Royce engine and retrieve performance data

-Can compare data of different clouds and engine performance/ health

-identify cities and profiles

-make histogram frequency vs time of year (max exposure vs min exposure?)

-focus analysis below ~5km peak

-peak ingestion rate can be dependent on day/dust storm

-ideally has dust event in file -> collect data from 2017-2018 of about 50-100 profiles

-focus on cloud particle data -> to confirm cloud or not check relative humidity

- how frequent, month/year, country/region -> pick out example and use hysplit, location, time -> give air mass, plane altitude

-focus middle east + north Africa -> can extend to SE asia and china

-maybe NE America, Canada (Vancouver)

-interesting comparison (mineralogy)-> merren looks into minerals?

-Can make time history of dust events over region! -> for region

- Identify curious flights

-Demonstrate how our report can be used for future

Email:

1. Collect as many ascent/descent profiles as possible starting 2018 and work backwards. Say 50-100.
2. Plot all the ascent/descent concentrations on a single graph! Colour code by city.
3. Plot a time series of "dust" event flag versus time, to see the annual change. Colour code by city.
4. Do the same as 3 but colour code by aircraft tail number (for Merren).
5. Take the plot in 2 and average up the particle concentrations over different height intervals - say 300-400m, 400-500m etc. Plot the average concentrations for the whole year (for dust events) as a function of altitude along with the median, standard deviations etc.
6. If there is sufficient data do the same as in 5 but for different parts of the year - you may have to split the year into 3 monthly periods - find out when the dust event frequency is supposed to be highest at what time of year to help with this. This will allow you to compare extreme dust events with average inside and outside the seasonal dust periods.
7. Then start to compare the altitude vs concentration plots for the different cities and then possibly the different aircraft (tail numbers).
8. Keep adding data from as far back as you can say back to 2010-12.There may be year to year variations which is also important to try and cover.

9. Then we can talk about more detailed analysis for specific events/cities which folds into Merren's project. This will allow you to look at average exposures to dust by aircraft in different cities.
10. Then suggest you go to the NOAA HYSPLIT web site:https://www.ready.noaa.gov/READYVolcAsh.php
11. You can run HYSPLIT online or download to your computer (PC or Mac). There is a basic tutorial here: https://www.ready.noaa.gov/documents/Tutorial/html/index.html
12. Note you can find a description of the HYSPLIT model here: https://www.arl.noaa.gov/hysplit/hysplit/
    HYPLSIT is not a simple gaussian plume model - that is considered passe these days! - it is a hybrid of the "Lagrangian approach, using a moving frame of reference for the advection and diffusion calculations as the trajectories or air parcels move from their initial location, and the Eulerian methodology, which uses a fixed three-dimensional grid as a frame of reference to compute pollutant air concentrations" (The model name, no longer meant as an acronym, originally reflected this hybrid computational approach).
13. One of the HYSPLIT versions contains an automated source-receptor matrix computation which if you complete the above work in time you might have a go at so as to identify the sources of some of the larger dust events. It has been used to look at wind blown dust events in the scientific literature cited on their site e.g. summarised in this paper:
    https://doi.org/10.1175/BAMS-D-14-00110.1

hysplit.pdf

Saturday 24/10/2020

-Started downloading all data points

Tuesday 27/10/2020

Started working on dust codeing

Wednesday 28/10/2020

Sent email:

- asking about cloud p1 val and cloud p1 stat

-which altitude to use, barometric altitude, radar altitude or GPS altitude

-aircraft tail number?

Reply:

-Identify dust events manually based on location, altitude and concentration.

-You can find a complete description of the data file format at:

http://iagos-data.fr/#DataFormatPlace:

For example line 7 in the data file (NASA AMES) should be

- The file reference date indicates the start point of the time axis in the file. The time axis is always stated in days and begins at 00 UTC on the file reference date.
- For better data management the file reference data will always be the date of the start of the IAGOS project.
- The revision date is the date when the file was created or last updated.
- Both dates are space separated, and stated in the format YYYY MM DD.

The file number includes the departure time e.g. departureUTC_time: 2011-12-23T11:45:20 for the example file below
Aircraft Identification:

For example: Data file for

Flight name : 20111223114520003

Departure airport : Frankfurt (FRA)

Departure date : 2011-12-23 11:45:20

Arrival airport : Addis Ababa (ADD)

Arrival date : 2011-12-23 17:56:20

The downloaded file contains the line (4)

"Instrument-03, IAGOS-01, A340-313, D-AIGT, Lufthansa, 4 seconds resolution"

A340-313 (This should be aircraft type and model), D-AIGT is the plane registration number. This is in two parts. The D is the country of origin, i.e. the nationality, in this case "D" refers to Germany. There then follows a suffix containing 1 to 5 characters which refers to the specific aircraft - linking this to a registration number ..Gary may be able to help with this.

The BCP data is reported as the sum of all particles counted over a **4 second period** converted to per cm-3.

Remember - the time resolution is 4 seconds, which corresponds to a horizontal resolution of approx 1 km and a vertical resolution of approx 20 m. The full files include ascent, descent and cruise phases.

The **cloud_P1_err** is the standard error based on all the counts over this period, again in per cm-3.

The Val and Stat flag are two different flags. The status flag is the condition of the instrument.

If these two numbers are not -9999 the cloud_P1 coulumn data cm-3 is good to use. Focus on that column for now.

cloud_P1, number_concentration_of_cloud_liquid_water_particles_in_air, **no cm-3**, Measured by IAGOS BCP

cloud_P1_err, number_concentration_of_cloud_liquid_water_particles_in_air standard_error, no cm-3, Measured by IAGOS BCP

cloud_P1_val, number_concentration_of_cloud_liquid_water_particles_in_air status_flag, 1, Measured by IAGOS BCP

cloud_P1_stat, number_concentration_of_cloud_liquid_water_particles_in_air status_flag, 1, Measured by IAGOS BCP

USE PRESSURE ALTITUDE i.e. barometric altitude

Saturday 31/10/2020

-code to extract data from nasa ames files

```
"""
Importing python modules
"""
import os
import matplotlib.pyplot as plt
import matplotlib.patches as mptch
import numpy as np

#path directory
path = r"C:\Users\alain\Documents\University files\4th year\Dust Masters Project\Data sets\\"
#-----------------------------------------------------------------------------------------------------------
"""
Defining functions to make code nicer
"""
#function to find the number of .txt files in the directory
def num_files():
    filelist = os.listdir(path)
    m = 0
    for i in filelist:
        if i.endswith('.txt'):
            m += 1
    return m
```

```
"""
Extracting the flight data from the profiles
"""
#declaring the arrays
#date of departure/ arrival
year = []
month = []
day = []

#aircraft tail number
tailnum = []

#airport location
city = []
country = []

#ascent or descent
profile = []

#relevant measurements
#creates number of lists equal to the number of files
UTCsec = [[] for i in range(1, num_files() + 1)]
long = [[] for i in range(1, num_files() + 1)]
lat = [[] for i in range(1, num_files() + 1)]
alt = [[] for i in range(1, num_files() + 1)]
cloudconc = [[] for i in range(1, num_files() + 1)]
cloudconcerr = [[] for i in range(1, num_files() + 1)]
h2o = [[] for i in range (1, num_files() + 1)]

#finds the list of files in the path
filelist = os.listdir(path)

#counter for kth .txt files
k = 0
for i in filelist:
    #only opens the .txt files
    if i.endswith('.txt'):
        with open(path + i, 'r') as f:
            #reads the lines
            lines = f.readlines()

            #tail number
            tailnum.append(lines[3].split(', ')[3])
            #date of flight
            year.append(lines[6].split(' ')[0])
            month.append(lines[6].split(' ')[1])
            day.append(lines[6].split(' ')[2])
```

```
            #ascent or descent profile
            prof = lines[46].split(' ')[1].rstrip('_profile\n')

            #files with no extra H2O and CO2 lines
            try:
                #this is first to trigger index error in case of extra H2O and CO2 files
                #airport city and country
                city.append(lines[57].split(', ')[1])
                country.append(lines[57].split(', ')[2].rstrip('\n'))
                #now append profile
                profile.append(prof)
                n = 0
                #relevant measurements
                for i in range(68, len(lines)):
                    UTCsec[k].append(int(lines[i].split(' ')[0]))
                    long[k].append(float(lines[i].split(' ')[1]))
                    lat[k].append(float(lines[i].split(' ')[2]))
                    alt[k].append(float(lines[i].split(' ')[3]))
                    cloudconc[k].append(float(lines[i].split(' ')[28]))
                    cloudconcerr[k].append(float(lines[i].split(' ')[29]))
            #files with extra H2O and CO2 lines
            except:
                #profile
                prof = lines[54].split(' ')[1].rstrip('_profile\n')
                profile.append(prof)
                #airport city and country
                city.append(lines[65].split(', ')[1])
                country.append(lines[65].split(', ')[2].rstrip('\n'))


                #relevant measurements
                for i in range(76, len(lines)):
                    UTCsec[k].append(int(lines[i].split(' ')[0]))
                    long[k].append(float(lines[i].split(' ')[1]))
                    lat[k].append(float(lines[i].split(' ')[2]))
                    alt[k].append(float(lines[i].split(' ')[3]))
                    cloudconc[k].append(float(lines[i].split(' ')[28]))
                    cloudconcerr[k].append(float(lines[i].split(' ')[29]))
                    h2o[k].append(float(lines[i].split(' ')[36]))
            k += 1
```

```
"""
Plotting the data points
"""
#When the value is -9999 it screws up the graph
#the instrument is turned off, so converting the number to 0
for p in range(0,num_files()):
    for s in range(0,len(cloudconc[p])):
        if cloudconc[p][s] == -9999:
            cloudconc[p][s] = 0
    for s in range(0,len(cloudconcerr[p])):
        if cloudconcerr[p][s] == -9999:
            cloudconcerr[p][s] = 0
    for s in range(0,len(alt[p])):
        if alt[p][s] == -9999:
            alt[p][s] = 0
    for s in range(0,len(h2o[p])):
        if h2o[p][s] == -9999:
            h2o[p][s] = 0

#declare colour array for colour coding
colour = [[] for i in range(1, num_files() + 1)]

#finds unique cities
cities = np.unique(city)

#colour coding by city
for i in range(0,num_files()):
    for j in range(0,len(cloudconc[i])):
        try:
            if city[i] == cities[0]:
                colour[i].append('blue')
            elif city[i] == cities[1]:
                colour[i].append('red')
            elif city[i] == cities[2]:
                colour[i].append('green')
            elif city[i] == cities[3]:
                colour[i].append('black')
            elif city[i] == cities[4]:
                colour[i].append('purple')
            elif city[i] == cities[5]:
                colour[i].append('lime')
            elif city[i] == cities[6]:
                colour[i].append('orange')
        except:
            pass
```

```
#array for the legend
key = []

#colours
blue = 'blue'
red = 'red'
green = 'green'
black = 'black'
purple = 'purple'
lime = 'lime'
orange = 'orange'
cyan = 'cyan'
silver = 'silver'

#Creating a legend
colcode(cities, blue, 0)
colcode(cities, red, 1)
colcode(cities, green, 2)
colcode(cities, black, 3)
colcode(cities, purple, 4)
colcode(cities, lime, 5)
colcode(cities, orange, 6)
colcode(cities, cyan,7)
colcode(cities, silver, 8)
plt.legend(handles = key, loc = 'upper left', bbox_to_anchor = (1,1), fancybox = True)

#setting the minimum and maximum for the axes
xmax = 5
xmin = 1.5*(10**-3)
ymax = max(max(alt))
ymin = min(min(alt))
axes = plt.gca()
axes.set_xlim([xmin, xmax])
axes.set_ylim([ymin, ymax])
axes.set_xscale('log')

#Plotting cloud concetration against altitude
plt.xlabel('Concentration of Cloud Particles/ number cm\u207B\u00b3')
plt.ylabel('Altitude/ m')
plt.title(f'Altitude against Cloud Particle Concentration for {num_files()} ascents/ descents')

#plots the scatter graph
for j in range(0,num_files()):
    plt.scatter(cloudconc[j], alt[j], s = 30, c = colour[j])
plt.grid()
plt.show()
```

-aiman will do dust frequency graphs!

Wednesday 04/11/2020

Sent email:

-Asking about next meeting

-lack of ascent/ descent profiles for latter part of 2016, very few in 2017 and none in 2018

-only two tail numbers found

Friday 06/11/2020

Zoom meeting: 2pm

https://zoom.us/j/99525618632

meeting notes:

-look at linked paper

geosciences-09-00162.pdf

-keep data in context

-get a map and check RH to see if dust particles or cloud particles-> high RH means cloud, low RH likely dust

-first graph I made might have clouds -> check RH

Looking at >5micron size particles

- Check if avg per altitude is equally ascent/ descent
- Log x axis

4 seconds time resolution

- Break down avg graph by area and time
- Not 0 value profiles

At ending can:

-note key dust events -> determine particle size with gary?

Period of interest: doha paper on solar panel -> shows monthly variation

Compare with our data!

-make short objective list

Email:

Action: Produce a list of objectives based on today's discussion and circulate them for feedback.
Attached is the publication I showed - in case you could not download it from Zoom.
My main takeaway is
1. Check the dust event data using RH data as discussed.
2. Look at regional variation of dust events if possible - you may have to extend the data period you are looking at.
3. Check the data for the latter half of the year - cf the regional/seasonal dust event frequency based on ground based observations in the attached paper.
4. Check Merren's paper and accumulate the total particle number over the different ascent/descent phases - the total accumulated particle counts is important obviously for the engine ingestion issue.
5. End objective - identify a goo Dohar dust event for comparison with surface samples- Merren to provide typical season/air mass source region to correlate better and Gary to "perhaps" provide mean particle size distributions.

Monday 09/11/2020

Sent email checking obj:

1. Collate more profiles from 2016 - 2018 to analyse seasonal variation noting 0-value profiles, referencing the attached paper

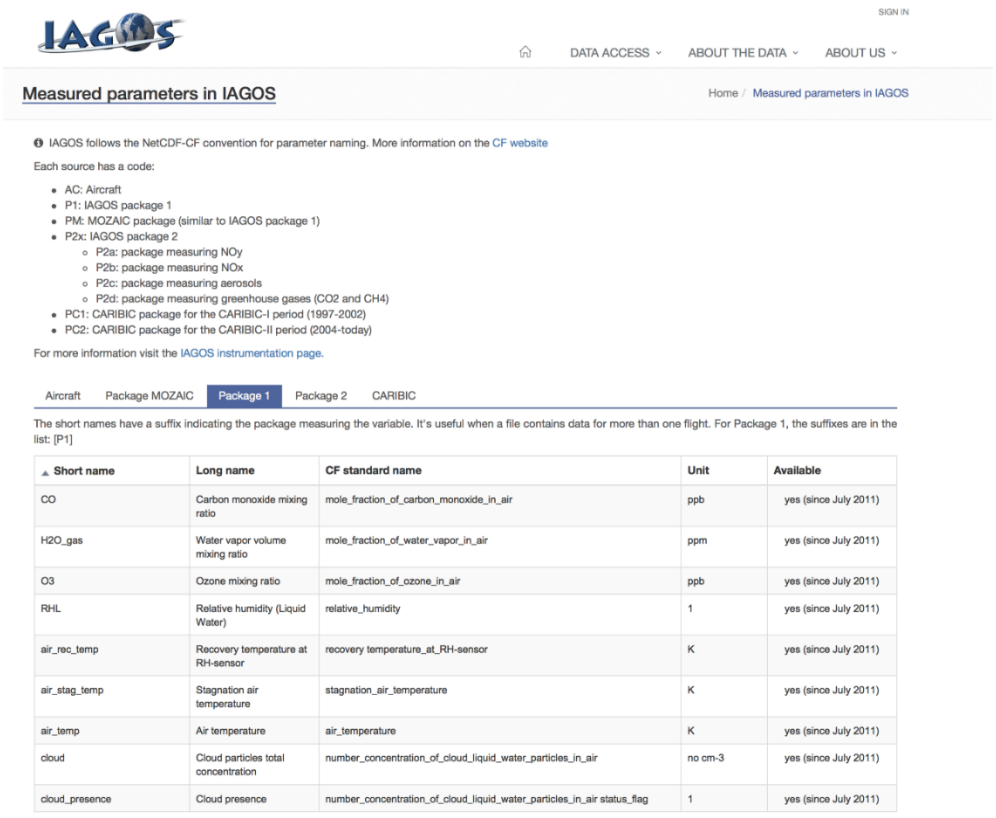
2. Extend this to pre-2016
3. Note key Doha dust event referencing ground sample paper on solar panels
4. Check RH data to verify cloud/dust content
5. Fix axis of the avg conc vs altitude graph - y-axis and log the x-axis
6. Use Merren's paper to work out the total particle number over different profiles.

Thursday 19/11/2020

-cant find relative humidity RH

Emailed asking where to find RH

Reply:



SIGN IN

DATA ACCESS ABOUT THE DATA ABOUT US

Measured parameters in IAGOS

Home / Measured parameters in IAGOS

IAGOS follows the NetCDF-CF convention for parameter naming. More information on the CF website

Each source has a code:

- AC: Aircraft
- P1: IAGOS package 1
- PM: MOZAIC package (similar to IAGOS package 1)
- P2x: IAGOS package 2
  - P2a: package measuring NOy
  - P2b: package measuring NOx
  - P2c: package measuring aerosols
  - P2d: package measuring greenhouse gases (CO2 and CH4)
- PC1: CARIBIC package for the CARIBIC-I period (1997-2002)
- PC2: CARIBIC package for the CARIBIC-II period (2004-today)

For more information visit the IAGOS instrumentation page.

Aircraft | Package MOZAIC | Package 1 | Package 2 | CARIBIC

The short names have a suffix indicating the package measuring the variable. It's useful when a file contains data for more than one flight. For Package 1, the suffixes are in the list: [P1]

Short name	Long name	CF standard name	Unit	Available
CO	Carbon monoxide mixing ratio	mole_fraction_of_carbon_monoxide_in_air	ppb	yes (since July 2011)
H2O_gas	Water vapor volume mixing ratio	mole_fraction_of_water_vapor_in_air	ppm	yes (since July 2011)
O3	Ozone mixing ratio	mole_fraction_of_ozone_in_air	ppb	yes (since July 2011)
RHL	Relative humidity (Liquid Water)	relative_humidity	1	yes (since July 2011)
air_rec_temp	Recovery temperature at RH-sensor	recovery temperature_at_RH-sensor	K	yes (since July 2011)
air_stag_temp	Stagnation air temperature	stagnation_air_temperature	K	yes (since July 2011)
air_temp	Air temperature	air_temperature	K	yes (since July 2011)
cloud	Cloud particles total concentration	number_concentration_of_cloud_liquid_water_particles_in_air	no cm-3	yes (since July 2011)
cloud_presence	Cloud presence	number_concentration_of_cloud_liquid_water_particles_in_air status_flag	1	yes (since July 2011)

-H2O gas data!

Water vapour mixing ratio is the density of water vapour divided by the density of dry air without water vapour

e.g. see this explanation if you are unfamiliar with the terms - https://www.e-education.psu.edu/meteo300/node/519

PDF

3-s2.0-B97804446329
68000020-main.pdf

Sunday 22/11/2020

-cloud concentration averaged over 100 m intervals code

```
131
132     #rounds up the maximum altitude to the nearest 100
133     maxalt = math.ceil(max(max(alt))/ 100) * 100
134
135     #finds the unique cities
136     cities = np.unique(city)
137
138     #declare average cloud concentration arrays
139     avg_cloud = [[[] for y in range(0, maxalt, 100)] for x in range(0,len(cities))]
140
141     #declare altitude intervals
142     alt_int = []
143     actavg_cloud = [[] for y in range(0,len(cities))]
144     actavg_clouderr = [[] for y in range(0,len(cities))]
145     altaxis = []
146
147     #appends all particle concentration in an altitude interval into one array, based on cities
148     for a in range(0, num_files()):
149         for b in range(0, len(alt[a])):
150             for i in range(0, len(cities)):
151                 x = 0
152                 if str(city[a]) == str(cities[i]):
153                     for c in range(0, maxalt, 100):
154                         if c < alt[a][b] <= c + 100:
155                             avg_cloud[i][x].append(cloudconc[a][b])
156                         else:
157                             pass
158                         x += 1
159                 else:
160                     pass
161
162     #finds the average concentration of each altitude interval
163     for i in range(0,len(cities)):
164         for d in range(0, maxalt, 100):
165
166             z = int(d/100)
167             #mean average
168             try:
169                 avg = sum(avg_cloud[i][z])/len(avg_cloud[i][z])
170             except:
171                 avg = 0
172             #standard deviation
173             std = np.std(avg_cloud[i][z])
174             #append data to array
175             actavg_clouderr[i].append(std)
176             actavg_cloud[i].append(avg)
177
178             if i == 1:
179                 alt_int.append(str(d) + '-' + str(d + 100))
180                 altaxis.append(d)
```

```
"""
Plotting the data
"""
#setting y ticks to every 500m
ypos = []
ylabels = []

for a in range(0, z):
    if a % 5 == 0:
        ypos.append(altaxis[a])
        ylabels.append(alt_int[a])

#setting the minimum and maximum for the axes
xmax = max(max(actavg_cloud))
xmin = 1.5*(10**(-5))
ymin = 0
ymax = max(max(alt))
axes = plt.gca()
axes.set_xlim([xmin, xmax])
axes.set_ylim([ymin,ymax])
#logs the graph
axes.set_xscale('log')

a1 = mptch.Patch(color = 'blue', label = str(cities[0]))
a2 = mptch.Patch(color = 'red', label = str(cities[1]))
a3 = mptch.Patch(color = 'green', label = str(cities[2]))
a4 = mptch.Patch(color = 'black', label = str(cities[3]))
a5 = mptch.Patch(color = 'purple', label = str(cities[4]))
a6 = mptch.Patch(color = 'lime', label = str(cities[5]))
a7 = mptch.Patch(color = 'orange', label = str(cities[6]))
plt.legend(handles = [a1, a2, a3, a4, a5, a6, a7], loc = 'upper left', bbox_to_anchor = (1, 1), fancybox = True)

#data point coloura
colour = []
colour.append('blue')
colour.append('red')
colour.append('green')
colour.append('black')
colour.append('purple')
colour.append('lime')
colour.append('orange')

#Plotting cloud concetration against altitude
plt.xlabel('Average Concentration of Cloud Particles in Interval/ number cm\u207B\u00b3')
plt.ylabel('Altitude Interval/ m')
plt.title('Altitude against Cloud Particle Concentration for ' + str(num_files()) + ' ascents/ descents')
plt.yticks(ypos, ylabels)

for i in range(0, len(cities)):
    plt.scatter(actavg_cloud[i], altaxis, s = 60, c = colour[i], edgecolors = 'black')
plt.grid()
plt.show()
```

Friday 04/12/2020

ZOOM MEETING: https://zoom.us/j/99525618632

Meeting notes:

Gary will ask for record of flights (service data) - > correlation of maintenance and end of a specific line?

-> can talk about this in report

Alt vs cloud conc – 6km demarcation? -> cirrus cloud regime?

->colour code for time of year, show seasonal trend

-> wind direction/ speed in data set!

-> use hysplit to show windspeed/ direction profile -> check temp data

-> apply hysplit analysis to different levels until cruise alt

-> show difference in location in air masses

Use merren's graph

Link profile to changes in mineral sources!

Log scale avg graph

Read Baumgardner paper about size distributions -> uncertainty large for large particles 20-40% -> look at stuff we could have done with more time

Seasonal data e.g. doha

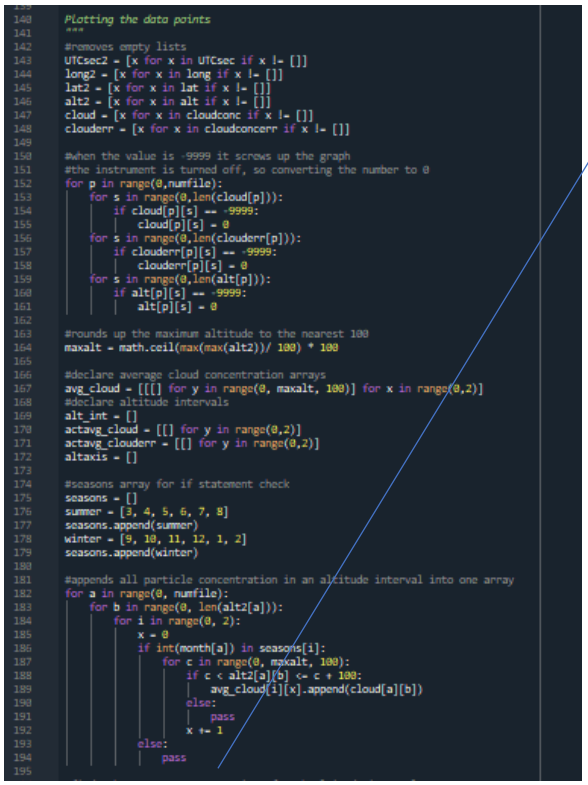
Recommendation for future research!

Saturday 19/12/2020

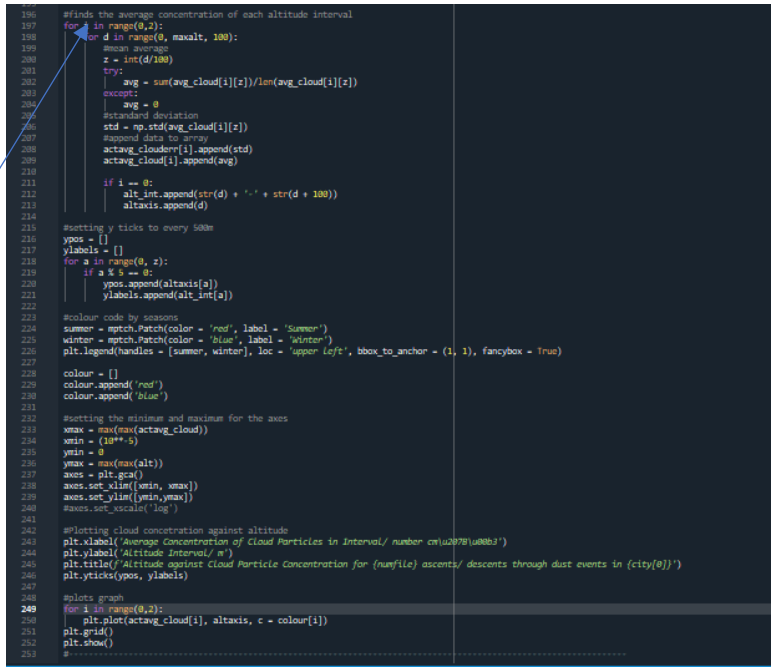
-Doha only flight profiles code

-compare summer and winter data

```
CITY = 'Doha'

#counter for kth .txt files
k = 0
for i in filelist:
    #only opens the .txt files
    if i.endswith('.txt'):
        with open(path + i, 'r') as f:
            #reads the lines
            lines = f.readlines()

            #files with no extra H2O and CO2 lines
            try:
                #this is first to trigger index error in case of extra H2O and CO2 files
                #airport city
                cty = lines[57].split(', ')[1]
                #only doha flights
                if cty == CITY:
                    city.append(cty)
                    country.append(lines[57].split(', ')[2].rstrip('\n'))

                    #tail number
                    tailnum.append(lines[3].split(', ')[3])

                    #date of flight
                    year.append(lines[6].split(' ')[0])
                    month.append(lines[6].split(' ')[1])
                    day.append(lines[6].split(' ')[2])

                    #ascent or descent profile
                    profile.append(lines[46].split(' ')[1].rstrip('_profile\n'))

                    #relevant measurements
                    for i in range(68, len(lines)):
                        #relevant measurements
                        UTCsec[k].append(int(lines[i].split(' ')[0]))
                        long[k].append(float(lines[i].split(' ')[1]))
                        lat[k].append(float(lines[i].split(' ')[2]))
                        alt[k].append(float(lines[i].split(' ')[3]))
                        cloudconc[k].append(float(lines[i].split(' ')[28]))
                        cloudconcerr[k].append(float(lines[i].split(' ')[29]))
                    k += 1
            #files with extra H2O and CO2 lines
            except:
                #airport city and country
                cty = lines[65].split(', ')[1]
                #only doha flights
                if cty == CITY:
                    country.append(lines[65].split(', ')[2].rstrip('\n'))
                    #profile
                    profile.append(lines[54].split(' ')[1].rstrip('_profile\n'))

                    #tail number
                    tailnum.append(lines[3].split(', ')[3])

                    #date of flight
                    year.append(lines[6].split(' ')[0])
                    month.append(lines[6].split(' ')[1])
                    day.append(lines[6].split(' ')[2])
```



```
"""
PLotting the data points
"""
#removes empty lists
UTCsec2 = [x for x in UTCsec if x != []]
long2 = [x for x in long if x != []]
lat2 = [x for x in lat if x != []]
alt2 = [x for x in alt if x != []]
cloud = [x for x in cloudconc if x != []]
clouderr = [x for x in cloudconcerr if x != []]

#when the value is -9999 it screws up the graph
#the instrument is turned off, so converting the number to 0
for p in range(0,numfile):
    for s in range(0,len(cloud[p])):
        if cloud[p][s] == -9999:
            cloud[p][s] = 0
    for s in range(0,len(clouderr[p])):
        if clouderr[p][s] == -9999:
            clouderr[p][s] = 0
    for s in range(0,len(alt[p])):
        if alt[p][s] == -9999:
            alt[p][s] = 0

#rounds up the maximum altitude to the nearest 100
maxalt = math.ceil(max(max(alt2))/ 100) * 100

#declare average cloud concentration arrays
avg_cloud = [[[] for y in range(0, maxalt, 100)] for x in range(0,2)]
#declare altitude intervals
alt_int = []
actavg_cloud = [[] for y in range(0,2)]
actavg_clouderr = [[] for y in range(0,2)]
altaxis = []

#seasons array for if statement check
seasons = []
summer = [3, 4, 5, 6, 7, 8]
seasons.append(summer)
winter = [9, 10, 11, 12, 1, 2]
seasons.append(winter)

#appends all particle concentration in an altitude interval into one array
for a in range(0, numfile):
    for b in range(0, len(alt2[a])):
        for i in range(0, 2):
            x = 0
            if int(month[a]) in seasons[i]:
                for c in range(0, maxalt, 100):
                    if c < alt2[a][b] <= c + 100:
                        avg_cloud[i][x].append(cloud[a][b])
                    else:
                        pass
                    x += 1
            else:
                pass
```



```
#finds the average concentration of each altitude interval
for i in range(0,2):
    for d in range(0, maxalt, 100):
        #mean average
        z = int(d/100)
        try:
            avg = sum(avg_cloud[i][z])/len(avg_cloud[i][z])
        except:
            avg = 0
        #standard deviation
        std = np.std(avg_cloud[i][z])
        #append data to array
        actavg_clouderr[i].append(std)
        actavg_cloud[i].append(avg)

        if i == 0:
            alt_int.append(str(d) + '-' + str(d + 100))
            altaxis.append(d)

#setting y ticks to every 500m
ypos = []
ylabels = []
for a in range(0, z):
    if a % 5 == 0:
        ypos.append(altaxis[a])
        ylabels.append(alt_int[a])

#colour code by seasons
summer = mptch.Patch(color = 'red', label = 'Summer')
winter = mptch.Patch(color = 'blue', label = 'Winter')
plt.legend(handles = [summer, winter], loc = 'upper left', bbox_to_anchor = (1, 1), fancybox = True)

colour = []
colour.append('red')
colour.append('blue')

#setting the minimum and maximum for the axes
xmax = max(max(actavg_cloud))
xmin = (10**-5)
ymin = 0
ymax = max(max(alt))
axes = plt.gca()
axes.set_xlim([xmin, xmax])
axes.set_ylim([ymin,ymax])
#axes.set_xscale('log')

#Plotting cloud concetration against altitude
plt.xlabel('Average Concentration of Cloud Particles in Interval/ number cm\u2078\u00b3')
plt.ylabel('Altitude Interval/ m')
plt.title(f'Altitude against Cloud Particle Concentration for {numfile} ascents/ descents through dust events in {city[0]}')
plt.yticks(ypos, ylabels)

#plots graph
for i in range(0,2):
    plt.plot(actavg_cloud[i], altaxis, c = colour[i])
plt.grid()
plt.show()
#-------------------------------------------------------------------------------------------
```