README.TXT

SAMAC introduction:

SAMAC: Software for Airborne Measurements of Aerosol and Clouds

SAMAC was created between 2011 and 2014 to facilitate processing a large number of aircraft-measured clouds and compare them with each other.

SAMAC is a pre-defined structure, fit to host airborne cloud measurements and a set of methods to help analyzing the data, and compare clouds with each other. SAMAC is entirely writen with Python. Python is a free open-source programming language that can be used on any major Operating System. SAMAC is open source and users are encouraged to write new methods and pieces of code, and make improvements to the code.

Before using SAMAC:

Download the suite of code files at https://github.com/StephGagne/SAMAC using the "download zip" button and save them in the directory of your choice.

If not already installed, install Python [from http://www.python.org/download/]. Linux users can install using the Synaptic Software Manager.

The use of IPython is recommended and the software has only been tested under IPython. It can be downloaded here: http://ipython.org/

Some more Python modules must also be installed:

- Scipy and NumPy: http://www.scipy.org/install.html
- Matplotlib: http://matplotlib.org/users/installing.html
- Basemap: http://matplotlib.org/basemap/users/installing.html
- h5py: http://www.h5py.org/

SAMAC cannot be used without a populated cloud object. To know how to create a cloud object,

read Cloud_Searcher_and_Creator.txt and SAMAC_structure.txt. An example cloud object ExampleCloud.p can be downloaded with this package and loaded using the following commands:

```
import pickle
ExCloud=pickle.load(open("/PATH/ExampleCloud.p","rb"))
```

Running SAMAC:

Open an IPython shell in the same directory the suite of code file has been downloaded to.

Note that although IPython runs in the directory of the software program, the data itself does not need to be in this directory.

Load the example cloud using pickle, or some of your own saved cloud instances. [import pickle; ExCloud=pickle.load(open("/PATH/ExampleCloud.p","rb"))]

To call on a method on cloud 'ExCloud', type ExCloud.METHODNAME(options). The final parentheses, even if empty, must be present after the method name, e.g. ExCloud.overview().

To view the description of a method and its options, type help(ExCloud.METHODNAME). Type 'q' to quit the help mode and return to the IPython shell.

To call on a property on cloud 'ExCloud', type ExCloud.PROPERTYNAME. There are no parentheses after properties. Properties return answers (as do some methods) and it is sometimes useful to have this answer stored in a variable rather than to have it appear on the IPython shell. To store the answer in a variable called 'AnswerVariable', type AnswerVariable=ExCloud.PROPERTYNAME.

How to help improving SAMAC:

You can report bugs using the Issues section (a circled exclamation mark) and open an issue. The same section can be used to suggest improvements, new methods, and so on.

To actively contribute to the project (code new methods, modify existing ones, etc.), fork the repository and make your changes. Once your changes are made and tested, you may send a pull request to have your code merged with the original/main branch of SAMAC for the benefit of all users. SAMAC's page on GitHub is: https://github.com/StephGagne/SAMAC.

CLOUD SEARCHER AND CREATOR

INITIALIZATION

To create cloud instances, users must use the __init__ function by calling:

```
import Cloud
c=Cloud.Cloud(data,title,units,times,sizedist)
```

In this case 'c' is our cloud instance (any other name for c would also work), and the input data are:

data: format of c.data
title: format of c.dttl
units: format of c.dunit
times: format of c.times
sizedist: format of c.sd
See SAMAC_structure.txt for detail on formats

data, title and units must be passed for the initilization to be successful.

'times' can be a dictionary with empty arrays:

```
times={
```

```
        "abovecloud": np.zeros((0,2)),
        "belowcloud": np.zeros((0,2)),
        "cloud": np.zeros((0,2)),
        "horicloud": np.zeros((0,2)),
        "verticloud": np.zeros((0,2))
          }
'sizedist' can be an empty list sizedist=list()
```

After initialization or any changes made to the cloud instance, it is recommended to save the instance to be able to reload it in a later session.

OPTIONS FOR POPULATING THE STRUCTURE

There are several possible ways to populate the cloud instance's structure. Most likely, a user will use a combination of the options below:

1- Using SAMAC's companion loading software (creates the cloud instance).
2- Filling in the structure manually (once the cloud instance is created).
3- Using methods (once the cloud instance is created).

1- Using SAMAC's companion loading software.

A data loading software is published along with SAMAC (Read_CreateClouds.py with CloudDataSorter.py) that guides the user through locating the most important data and conforming their titles to SAMAC's requirements, preparing the basic aircraft data and size distributions to fit SAMAC's structure, initializing the cloud instance (there is no need to call Cloud.Cloud), fill the c.desc structure, and save the source files' path and name.

The software is optimized to handle the programmer's data sets and it is very likely that users will have to adapt the software to be able to handle their own data formats.

To try the software, simply type this line in the IPython shell: %run Read_CreateClouds.py

Modifications can be made in files Read_CreateClouds.py and CloudDataSorter.py. It is recommended that users do not modify the contents of plotcloud.py because routines in this file are also used by SAMAC. If modifications need to be made, it is rather recommended to make a copy of the code and use the modified copy with a different method name.
Example: def plotcloud2_copy
        Modified code starts here

2- Filling in the structure manually.

A user can also fill the structure manually following the formats described in SAMAC_structure.txt. This means preparing the data in the right format, initializing the cloud, and populating the rest of the structure by assigning data to the structure.

3- Using methods.

Some methods in SAMAC have for purpose to alter the cloud instance's content. These methods are listed here:

add2basic: adding general data (to c.data or to c.extradata depending on the time stamp).
describe: giving qualitative descriptives to the cloud
addsizedistxl: adding a size distribution from an excel file
addsizedist: adding a size distribution from a text file
defheight: lower and upper estimates for cloud base and top height
defBGheight: best guesses for cloud base and top height
lwcflag: quality flags of LWC profiles compared to adiabatic LWC
timechange: add, delete a manoeuvre, or change the times of an existing one
MaskData: quality control, removing unwanted data

SAMAC STRUCTURE

INTRODUCTION:

SAMAC is based on object-oriented programming. The methods (description available in SAMAC_methods.txt) are linked to these objects. As for any object-oriented program, objects must be created. In this case, objects are airborne measurements of clouds. Each measured cloud in stored in a cloud instance (an instance of the cloud object). For methods and properties to work on the object, the data must be organized following a known structure. In this file, the structure of SAMAC's cloud objects is described.

CREATING A CLOUD INSTANCE:

Creating a cloud instance is done by initializing a cloud. To initialize, one must first import the Cloud object program in SAMAC:

```
import Cloud
```

The function __init__ to initialize a cloud instance is called with this line of code:

```
MyFirstCloud=Cloud.Cloud(data,title,units,times,sizedists)
```

The initial input data in the parenteses must follow strict structures which are described in the structure section below:

data: format of c.data
title: format of c.dttl
units: format of c.dunit
times: format of c.times
sizedist: format of c.sd

For more information on how to create cloud instances can be found in Cloud_Searcher_and_Creator.txt

CLOUD OBJECT STRUCTURE:

To be able to use SAMAC, the data must be placed in predefined structures. The expression 'populating the structure' used in SAMAC's documentation means putting data in the right format and the right place in this predefined structure.

STRUCTURE:
---
c.data (2d numpy array or masked array).
Main or basic data. Table with time and corresponding data. The position of the aircraft (lon, lat, altitude, speed, etc.). All the data in this section should share the same time stamp. Any other data that shares the same time stamp should be in this section, in the same table. Each quantity is a row vector.
c.dttl (1d list of strings).
Corresponding titles or descriptions of the basic data found in c.data (ex. ['time','altitude',...]).
c.dunit (1d list of string).
Corresponding units to the basic data found in c.data (ex. ["g/m3","seconds",...]).
c.orient (string).
Orientation of the main data. Populated upon initialization. SAMAC has only been tested with the 'row' orientation. If the data is oriented in columns, c.orient='col'. In the case methods fail, it may be quicker to transpose c.data than modifying methods. Both are possible.
c.extradatanum (integer).
Number of times the methods add2basic was used to add data with a different time stamp from the basic data in c.data after initialization.
c.extradata (list of 2d numpy arrays or masked arrays).
Extra time series data with a different time stamp from the basic data. The format is the same as for the basic data, and there also must be a time stamp row in the data set. Each data set (a 2d numpy array) is an item of the list.
c.extrattl (list of 1d lists of strings).
List of the corresponding titles or descriptions of the data found in c.extradata (ex. ['time','lwc',...]). The list's position in the general list must correpond to the position of the data set in the extradata list.
c.extraunit (list of 1D lists of strings).
List of the corresponding units of the data found in c.extradata (ex. ['days','g/m3',...]). The list's position in the general list must correpond to the position of the data set in the extradata list and extrattl list.
c.sd (List of dictionaries).
Size distribution section. Each instrument measuring a size distribution gets a position in the list. All size distributions are stored in this section. Each individual size distribution can be accessed using c.sd[n] where n is a number representing the position of a distribution in the list (integers from 0 to the number of size distributions -1).
If there is a size distribution instrument but no measurements were made during the cloud measurements, the size distributions can be initialized with nan or empty arrays. Note that when using the method 'filler' if nans are used, nans are returned; if empty arrays are used, zero concentrations are returned.
c.sd[n] (dictionary).
Size distribution unit including concentration data, size bins, time stamps, distribution name, units, mean and median diameter, total concentration, source file and size distribution type.
c.sd[n]["data"] (2d numpy array).
Size distribution concentration as a function of time and size (size bins as rows and time as columns). The concentration must be in dN/dlogDp format for methods to work properly on it. Method to convert size distributions into dN/dlogDp and back are available in CloudDataSorter.py.
c.sd[n]["bins"] (1d numpy array).
Mean particle diameter sizes (1 per bin) of the distribution. The array should be as long as the number of rows in c.sd[n]["data"].
c.sd[n]["time"] (1d numpy array).
Times at which the size distribution was recorded. The array should be as long as the number of

columns in c.sd[n][“data”].

c.sd[n][“Distname”] (string).

Name of the size distribution, either an instrument name or a more descriptive name, to the user's discretion.

c.sd[n][“units”] (string).

Units of the size distribution's concentration.

c.sd[n][“medp”] (1d numpy array).

Median particle diameter for this size distribution. The array should be as long as the number of columns in c.sd[n][“data”].

c.sd[n][“mdp”] (1d numpy array).

Mean particle diameter for this size distribution. The array should be as long as the number of columns in c.sd[n][“data”].

c.sd[n][“total”] (1d numpy array).

Total particle number concentration for this size distribution. The array should be as long as the number of columns in c.sd[n][“data”].

c.sd[n][“sourcefile”] (string).

Source file for the size distribution's data.

c.sd[n][“sdtype”] (string).

Type of particles measured in this size distribution (Aerosol, Cloud droplet, or Precipitation). SAMAC supports, A,C and P, followed by a 1 to denote priority of an instrument (for example A1 for the primary aerosol size distribution). Users can improvise around it, but should be careful and verify how the methods behave with their new system.

c.times (dictionary)

Manoeuvre times - Times at which the cloud measurement and manoeuvres start and end. All times in the structure should be in days since 1 January of year 1 (1 Jan of year 1 at noon would be written 1.5). This can be calculated using the python datetime module and method "toordinal". The dictionary contains the overall cloud times, and times for manoeuvres above cloud, below cloud, horizontally in-cloud and vertically through clouds.

c.times[“cloud”] (2d numpy array 1X2)

Times at which the measurements of the cloud started and ended. Start time at [0,0] and end time at [0,1] (they would be reached by typing c.times[“cloud”][0,0] and c.times[“cloud”][0,1] respectively).

c.times[“abovecloud”] (2d numpy array NX2)

Times at which the measurements above the cloud started and ended. Shape is N per 2. Start time at [N,0] and end time at [N,1] where N is the (N-1)th measurement leg or scan number above cloud.

c.times[“belowcloud”] (2d numpy array NX2)

Times at which the measurements below the cloud started and ended. Shape is N per 2. Start time at [N,0] and end time at [N,1] where N is the (N-1)th measurement leg or scan number below cloud.

c.times[“horicloud”] (2d numpy array NX2)

Times at which the measurements horizontally in the cloud started and ended. Shape is N per 2. Start time at [N,0] and end time at [N,1] where N is the (N-1)th measurement leg or scan number horizontally in cloud.

c.times[“verticloud”] (2d numpy array NX2)

Times at which the measurements vertically in the cloud started and ended. Shape is N per 2. Start time at [N,0] and end time at [N,1] where N is the (N-1)th measurement leg or scan number vertically in cloud.

c.descedit (integer)

Number of times the method 'describe' has been used to either create or edit the Descriptive Data section (used to know whether to go through the basic description or allow editing).

c.desc (dictionary).

Section where qualitative or descriptive data is stored. Can be filled or edited using method 'describe', or using this kind of line: c.describe["newentry"]='newentryvalue'

c.desc[“campaign”] (string).
Name of the campaign or project where the data was measured or created

c.desc[“date”] (string).
Date on which the cloud was measured, YYYYMMDD.

c.desc[“time”] (string).
The time at which the cloud measurement started, HH:MM.

c.desc[“cloudtype”] (string).
Type of cloud observed (e.g. Stratiform, convective, etc.)

c.desc[“environment”] (string).
Environment in which the cloud was situated (e.g. Maritime, desert, tropical forest, etc.)

c.desc[“flight#”] (string).
Flight or simulation number or name

c.desc[“humanplace”] (string).
Place above which the cloud was measured. The place should be recognizable by humans (e.g. a city name)

c.desc[“warmcold”] (string).
Warm, cold (ice), or mixed clouds

c.desc[“landwater”] (string).
Surface type below the cloud: land, water or both.

c.desc[“layered”] (string).
Describes if there is a knowledge of other layers of cloud above or below the measured cloud

**These are only suggested descriptives and more can be added. The keywords used as entries should be consistent for users to be able to use the full potential of this section.

c.props (dictionary).
Section in which properties needing human input are stored after initialization. New properties can be added at will by adding a key to the dictionary: c.props["newproperty"]='newpropertyvalue'. The properties in the dictionary can be of any type e.g. string, array, list.

c.props[“height”] (list of numpy arrays).
Base and cloud top height for all vertical profiles (lower and upper estimates). Generate this entry using method 'defheight'. For each vertical profile there is a height entry (in the same order). Each entry is an array of shape 4 by 1 that includes the lowest and highest estimate for cloud base and lowest and highest estimate for cloud top.

c.props["BGheight"] (list of numpy arrays).
Base and cloud top height for all vertical profiles (best guess). Generate this entry using method 'defBGheight'.For each vertical profile there is a best guess height entry (in the same order). Each entry is an array of shape 2 by 1 with the best guess for cloud base and cloud top.

c.props[“lwcflags”] (list of numpy arrays).
Flags reflecting the quality of the LWC profile compared to the adiabatic LWC. Generate this entry using method 'lwcflag'. For each vertical profile there is a LWC profile quality entry (in the same order). Each entry is an array containing all the flags pertinent to the profile.

ESSENTIAL TITLES FOR SAMAC TO WORK:

Some titles in c.dttl and c.extrattl must conform to a predefined naming for some of the methods to work properly. Here is a list of predefined titles that must be used for the primary measurements of the types below:

Data measured	predefined name	note
- time stamps	time	must be in ordinal format (number of days since jan 1 of year 1)

- altitude	altitude	meters are the recommended units
- latitude	latitude	keep the sign according to international standards
- longitude	longitude	keep the sign according to international standards
- theta D	theta-d	
- theta Q	theta-q	
- theta E	theta-e	
- temperature	temperature	
- relative humidity	RH	Python is a case-sensitive language
- pressure	Pressure	
- liquid water content	LWC	
- wind speed	windspeed	
- wind direction	winddir	
- updraft velocity	udvel	
- true air speed	*TAS*	must have the sequence TAS in its name

---

SAMAC METHODS

INTRODUCTION

Methods and properties are the core of SAMAC. Methods and properties are routines designed to handle cloud objects as defined in SAMAC_structure.txt. Here we describe what each method does, and how to use the options. We use 'c' as a generic cloud instance to simplify the text.

CALLING & OPTIONS

Calling a method or a property means that a user is commanding the program to use a routine, either a method or a property. To call a method, a user must first write the cloud instance's name (in our case, we call it 'c') followed by a dot and the method or property name. To call a property on an instance, a user would type 'c.propertyname'. To call a method on an instance, a user would type 'c.methodname()'. There may be a need to define options inside the parentheses (see next paragraph), but even if options are not needed, calls on methods always must be followed by parentheses. To store the answers of a method or property in a variable, say 'R', a user would type 'R=c.methodname()' or 'R=c.propertyname'.

Methods often have options. Options are input variables that can be changed by the user upon calling the method [c.methodname(option1,option2)].
When options have defaults, there is no need to define them in the method call if the default suits the user. For example, if option1 has a default that is okay for the user and option2 doesn't have a default, or does not suit the user, the call can be: c.methodname(option2=something).
When an option does not have a default, its value must be specified. When it does have a default, users can either accept the default by not including the option in the call (as in the example above), or change the options:
c.methodname(option1=somethingelse, option2=something).

In some methods, input variables are required, these have no default and the input should be in the specified format.

METHODS

add2basic(newdata,newttl,newunit):
This method is used to add data to the basic aircraft data if the new data comes in after initialization, or if it has a different time stamp from the basic data (c.data). If the new data has the same time vector as the original basic aircraft data, it will be added to c.data, c.dttl, c.dunit. If the new data time stamp is different from the basic aircraft data, the data, titles and units will be added in lists under c.extradata, c.extrattl and c.extraunit respectively.
Inputs:
newdata: 2d numpy array containing the data to be added. In the case the time stamp is different from the basic data time stamp the time stamp must be part of the array. Each measured quantities should be in rows. If your data is in columns, you can use the transpose function.
newttl: list of titles for the data in newdata (length = number of rows in newdata). Note that the title for time in c.extrattl should be 'time'; in c.dttl it should be 'time1' or something different from 'time'!
newunit: list of units for the data in newdata (length = number of rows in newdata). Time must be in ordinal format (number of days since Jan 1st of year 1 [1 January of year 1 at noon is 1.5]).

addsizedist()
This method is used to add a size distribution to a cloud instance 'c'. The added size distribution will be found appended to c.sd. The data source must be a text format file (e.g. txt or dat). Use addsizedistxl for Excel spreadsheets handling. This guides the user in loading the source file and goes through the process of defining every dictionary entry for a size distribution (see SAMAC_structure for details).

addsizedistxl()
This method is used to add a size distribution to a cloud instance 'c'. The added size distribution will be found appended to c.sd. The data source must come from an Excel spreadsheet. Use addsizedist for text formats handling. This guides the user in loading the source file and goes through the process of defining every dictionary entry for a size distribution (see SAMAC_structure for details).

avCDNC(abovesize, uppersize, inst, scan)
This method returns the average cloud droplet concentration in a vertical profile weighted according to the LWC. The total concentration is integrated as a function of altitude, weighted with LWC and then divided by the liquid water path. [int(total conc * lwc * dalt) / int(lwc*dalt)] The units are the same as the units in the size distribution.
Options:
abovesize: size above which the total concentration is calculated. Default is NaN, which means the lowest size is the smallest size bin available for the instrument.
uppersize: size below which the total concentration us calculated. Default is NaN, which means the highest size is the biggest size bin available for the instrument.
inst: name of the instrument that should be used as found in c.sd[N]["Distname"]. Default is 'FSSP96'.
scan: scan number or nth vertical profile. Default is 0 because that is the first profile.

avsizedist(prof, scan, inst, filler)
This method returns the average size distribution for a given instrument, manoeuvre type and number. The returned average (mean) size distribution is an array with the mean concentration of particles in the first row and the size of the particles in the second. Units are the same as in the size distribution.
Options:

prof: profile, or manoeuvre type (cloud,abovecloud,belowcloud,verticloud,horicloud and belowvert (special profile, see method belowprof)). Default is 'belowcloud'.

scan: scan number or nth manoeuvre of type specified in option 'prof'. Default is 0.

inst: name of the instrument that should be used as found in c.sd[N]["Distname"]. Default is 'PCASP'.

filler: filling with a given time step (useful for 2d type data) =1 ON, =0 OFF. The time step can be adjusted in the call of method filler. Default is off (0).

belowprof()

This method returns a list of dictionaries (one per vertical manoeuvre) that includes a) index of what belongs to below the vertical profile, b) area index (column, binary), and c) times. This method is used to access the special profile 'belowvert'.

Returned list item format R=c.belowprof():

- R[0]["index"]: index of what is below cloud and within 0.0075 degrees radius (BELOW the cloud base (lowest estimate)).
- R[0]["areaindex"]: nonzero if the aircraft is in the area of the 0th (first) vertical profile and within 0.0075 degrees radius (includes all manoeuvres). Same length as the time stamp.
- R[0]["times"]: times when the aircraft was below the 0th (first) vertical profile.

defBGheight()

Interactive method. Sets up or clears and resets the best guess for cloud base and top. The user is asked to click twice per vertical profile: at the bottom and at the top. The answers is stored in a list under c.props["BGheight"] with one 2X1 array per vertical profile/list item. If you need to add a profile, we recommend you insert a space in the list, which can then be overwritten without touching the other entries.

defheight()

Interactive method. Sets up or clears and resets the lower and upper estimates for cloud base and top. The user is asked to click 4 times per vertical profile: twice at the bottom and twice at the top. The answers are stored in a list under c.props["height"] with one 4X1 array per vertical profile. If you need to add a profile, we recommend you insert a space in the list, which can then be overwritten without touching the other entries.

describe()

This method is used to enter the description of a cloud if it is used for the first time, otherwise to change previous entries or add a new one. Entries can also be added or modified manually by simply (re)defining c.desc["keyname"].

effrad(inst, bindist)

This method returns the effective diameter or radius (depending on what the size bins are) for a given instrument for the entire cloud period. The diameter/radius has the same units as the instrument's units (usually micrometers).

Options:

inst: name of the instrument that should be used as found in c.sd[N]["Distname"].

bindist: 'lin' if the difference between bins is linearly distributed and 'log' if they are logarythmically distributed. Default is 'lin'.

filler(ts, timestep, inst)

This method returns a filled size distribution (type dict), filled with zeros when no measurements

were returned. This was designed for 2d data: this instrument only reported data when it detected levels above zero.

Options:

ts: start and end time over which the data should be filled. Normally, times from c.times['manoeuvre'][scan#].

timestep: in seconds (must be a float), is the time step with which the instrument returns data, and the time step at which filling will take place. Default is 10.0.

inst: name of the instrument that should be used as found in c.sd[N]["Distname"]. Default is '2dc'.

hrzplot(interact, Rtime)

This method plots the LWC & Altitude, Na (aerosols) & Nd (droplets), Temperature & Theta-Q, updraft velocity & its running standard deviation (turbulence) as a function of time. One figure for each horizontal scan is produced. Fully handles the extradata module.

Options:

interact: interactive mode ON (=1) or OFF (=0). Default is OFF (=None)

Rtime: makes the plots display readable times (HH:MM). Default is OFF (=None), everything else activates Rtime.

lwcflag(base)

Interactive method. This method asks the users the rate the LWC profiles according to their quality and features. All vertical scans are evaluated the first time, after which one can edit a particular scan. The answers are stored under c.props["lwcflags"] in a list (one 1d array with flags per scan).

Option:

base: controls the method to evaluate the cloud base. Default is base='bg' for best guess (defBGheight) cloud base. Other possibility is base='4point': to use the lowest estimate of the 4-point method (defheihgt).

adlwc(base)

Calculates the adiabatic LWC based on the temperature and pressure for each vertical profile. It returns the altitude and the adiabatic LWC in a list of 2d arrays, one array for each vertical profile. The 0th profile has the first position in the list c.adlwc[0]. Altitude is in the first row c.adlwc[0][0], and adiabatic lwc in the second c.adlwc[0][1].

Option:

base: controls the method to evaluate the cloud base. Default is base='bg' for best guess (defBGheight) cloud base; base='4point' to use the lowest estimate of the 4-point method (defheight).

mapcloud(interact)

This method plots the trajectory of the aircraft during the whole cloud period over a map of the region. Time is colour-coded and the thickness of the line reflects the LWC. Note: extradata module is handled for lwc only, as latitude and longitude are expected to be found in the basic aircraft data.

Options:

interact: interactive mode. When the interactive mode is off, users must close the figure before they can use the IPython shell again. The figure can be kept open when the interactive mode is on. Default is ON (=1), 0 for OFF mode.

MaskData()

Interactive method. This method is used to mask points that are measurement errors or that should be removed from the analysis for any reason. The extradata module is fully handled.

overview(interact, Rtime)

This method plots the cloud altitude, latitude, longitude, particle concentration, droplet concentration, liquid water content and precipitation drop concentration as a function of time for the whole cloud period. When using the interactive mode, the user can magnify or offset the data, and use zooms. The extradata module is fully handled.

Options:

interact: controls whether adjustments to the figure can be made or not. Default is adjustments OFF (=0), for ON mode interact=1.

Rtime: makes the plots display readable times (HH:MM). Default is OFF (=None), everything else activates Rtime.

path3d()

This method plot the flight path of the aircraft during the whole cloud period. The colour follows the colour time tracking in overview and mapcloud.

plotallavsd(prof, num, interact, exc)

This method plots the average size distribution of all size distribution instruments (by default: except 2d-type measurements) for a given manoeuvre and scan number. The maximum number of instruments that can be displayed at once is 8.

Options:

prof: profile, or manoeuvre type (cloud, abovecloud, belowcloud, verticloud, horicloud and belowvert (special profile, see method belowprof)). Default is 'belowcloud'.

num: scan number or nth manoeuvre of type specified in option 'prof'. Default is 0.

interact: interactive mode. When the interactive mode is off, users must close the figure before they can use the IPython shell again. The figure can be kept open when the interactive mode is on. Default is OFF (=None,0), 1 for ON mode.

exc: exceptions not to be plotted. Exceptions should be stated in a single string, with instrument names (as found in c.sd[n]["Distname"]) separated by commas (e.g. exc='2dc,PCASP,FSSP100'). Default excepts all instruments that contain '2d' in their name.

plotavsd(prof, scan, inst, filler)

This method plots the average size distribution for a given instrument and manoeuvre type and number.

Options:

prof: profile, or manoeuvre type (cloud, abovecloud, belowcloud, verticloud, horicloud and belowvert (special profile, see method belowprof)). Default is 'belowcloud'.

scan: scan number or nth manoeuvre. Default is 0.

inst: name of the instrument that should be used as found in c.sd[N]["Distname"]. Default is PCASP.

filler: filling with a given time step (useful for 2d type data) =1 ON, =0 OFF. The time step can be adjusted in the call of method filler. Default is off (0).

plotsd(Rtime)

Interactive method. This method is used to plot the size distribution for a selection of instruments, and manoeuvres, as a function of time (surface plot). The selection of instruments and manoeuvres is made by answering to prompt questions.

Options:

Rtime: makes the plots display readable times (HH:MM). Default is OFF (=None), everything else activates Rtime.

precipconc(abovesize, scan, minlwc, filler)

This method returns the integrated amount of precipitation drops (number, with respect to altitude) in a vertical profile in-cloud divided by the depth of the cloud. A 2d instrument is used. Units are the same as the instrument's. Handles the extradata module for LWC.

Options:

abovesize: size above which the number of drop is integrated. Default is 100 microns (or whatever units are used in the size distribution).

scan: scan number or nth vertical profile. Default is 0.

minlwc: minimum lwc for the instruments to be considered in-cloud. Default is 0.01.

filler: filling with a given time step (useful for 2d type data) =1 ON, =0 OFF. The time step can be adjusted in the call of method filler. Default is off (0).

precipincloudTF(abovesize)

This method tells whether precipitation in cloud (horicloud + verticloud) was observed during the entire cloud period: a boolean (true or false) is returned. Uses a 2d instrument ('2d' must be in sd[n]["Distname"]).

Options:

abovesize: size above which a droplet counts as precipitations. Default is 100.

timechange(Rtime)

This method is used to change one of the manoeuvre time frames in c.times. The user will be asked to choose the manoeuvre type and number that need to be modified, added or deleted.

Options:

Rtime: makes the plots display readable times (HH:MM). Default is OFF (=None), everything else activates Rtime.

totalprecip(abovesize, scan, filler)

This method returns the integrated (with respect to altitude) amount of precipitation drops (number) in a vertical profile. A 2d instrument is used. Units are the same as the instrument's concentration X units of altitude.

Options:

abovesize: size above which a droplet counts as precipitations. Default is 100.

scan: scan number or nth vertical profile. Default is 0.

filler: filling with a given time step (useful for 2d type data) =1 ON, =0 OFF. The time step can be adjusted in the call of method filler. Default is off (0).

totalvolprecip(abovesize, scan, filler)

This method returns the integrated (with respect to altitude) amount of precipitation drops (volume) in a vertical profile. A 2d instrument is used. NOTE: Units are in meters if the diameter is in micrometers, the concentrations in L-1, and the altitude in meters. Otherwise, they are in the same units as the instrument's concentration X units of altitude X units of diameter^3 X 1e-15.

Options:

abovesize: size above which a droplet counts as precipitations. Default is 100.

scan: scan number or nth vertical profile. Default is 0.

filler: filling with a given time step (useful for 2d type data) =1 ON, =0 OFF. The time step can be adjusted in the call of method filler. Default is off (0).

vpinfo(param)

This method returns information on the chosen parameter from c.dttl or c.extrattl in the cloud for all vertical scans. The averaging of the parameter is done over each vertical profile. Returns a dictionary 'H' containing H["bottom"]: parameter at the cloud base; H["top"]: parameter at the cloud top; H["mean"]: mean parameter through the cloud; H["median"]: median parameter through the cloud; H["minimum"]: minimum parameter through the cloud; H["maximum"]: maximum parameter through the cloud; H["stdev"]: standard deviation of the parameter through the cloud; H["delta"]: difference of parameter between the bottom and the top; H["slope"]: delta divided by the mean thickness; H["units"]: units of the parameter. Fully handles the extradata module (note that altitude must be in the basic data at c.data).

Options:

param: title of the investigated quantity as found in c.dttl or c.extrattl.

vprof(interact, axtype1, axtype2)

This method plots the Temperature, Theta-Q, LWC, RH, Particle, Cloud Droplet and Precipitation drop concentrations, in the cloud as a function of Altitude. One plot for each vertical scan is produced. The extradata module is fully handled in this method.

Options:

interact: interactive mode. When the interactive mode is off, users must close the figure before they can use the IPython shell again. The figure can be kept open when the interactive mode is on. Default is OFF (=None,0), 1 for ON mode.

axtype1: axis scale type. 'log' if the bottom axis of the left hand plot should be logarithmic, 'lin' if it should be linear. Default is log scale.

axtype2: axis scale type. 'log' if the bottom axis of the right hand plot should be logarithmic, 'lin' if it should be linear. Default is log scale.

wholeprof(interact, axtype1, axtype2)

This method plots the Temperature, Theta-Q, LWC, RH, Particle, Cloud Droplet and Precipitation drop concentrations, in the cloud as a function of Altitude. All the available data, regardless of manoeuvres are plotted. The extradata module is fully handled in this method.

Options:

interact: interactive mode. When the interactive mode is off, users must close the figure before they can use the IPython shell again. The figure can be kept open when the interactive mode is on. Default is OFF (=None,0), 1 for ON mode.

axtype1: axis scale type. 'log' if the bottom axis of the left hand plot should be logarithmic, 'lin' if it should be linear. Default is log scale.

axtype2: axis scale type. 'log' if the bottom axis of the right hand plot should be logarithmic, 'lin' if it should be linear. Default is log scale.

writeouthdf5(fname,fdir)

This method writes out the content of the cloud in a string representation. This method can be used if one wants to export the cloud's data to a different programming language or make a language-independent backup of their cloud-object's data. This method is designed to handle the current official structure of the object, if the structure has been customized by users, these custom structures will not be saved. This method will need to be modified to accommodate the new structure members. Also note that all masked data will become unmasked.

Options:

fName: Name of the file where the cloud will be saved. Default saves with name SAMAC_Cloud_date_time.

fdir: directory in which the file will be saved.

PROPERTIES

lwp:
Calculates the liquid water path from a cloud's vertical profiles. Returns an array with the liquid water paths. One LWP is returned for each vertical profile in a 2d array (number of profiles X 1). The 0th (first) profile is accessed with c.lwp[0][0], 1st (second) with c.lwp[1][0], and so on. This property handles the LWC being in the extradata module.

thickness:
Calculates the max and min cloud thickness (in meters) based on c.props["height"] (lower and upper estimate for cloud base and cloud top). The method defheight must be run at least once on the object for this property to work. Returns a dictionary c.thickness["min"] and c.thickness["max"]. This returns estimates based on the 4-point method only (defheight).

tempinfo:
Returns information on the temperature in the cloud (same units as in data). Note that the temperature is averaged over the entire cloud time at the altitude required (bottom, top or in-cloud). This is different from using CloudObj.vpinfo(temperature) which gives information on vertical profiles only. Returns a dictionary: c.tempinfo["bottom"]: temperature at the cloud base; c.tempinfo["top"]: temperature at the cloud top; c.tempinfo["mean"]: mean temperature through the cloud (in cloud); c.tempinfo["median"]: median temperature through the cloud (in cloud); c.tempinfo["stdev"]: standard deviation of the temperature through the cloud (in cloud); c.tempinfo["delta"]: difference of temperature between the bottom and the top; c.tempinfo["slope"]: delta divided by the mean thickness.

presinfo:
Returns information on the pressure (same units as in data) in the cloud. Note that the temperature is averaged over the entire cloud time at the altitude required (bottom, top or in-cloud). This is different from using CloudObj.vpinfo(pressure) which gives information on vertical profiles only. Returns a dictionary: c.presinfo["bottom"]: pressure at the cloud base; c.presinfo["top"]: pressure at the cloud top; c.presinfo["mean"]: mean pressure through the cloud; c.presinfo["median"]: median pressure through the cloud; c.presinfo["stdev"]: standard deviation of the pressure through the cloud; c.presinfo["delta"]: difference of pressure between the bottom and the top; c.presinfo["slope"]: delta divided by the mean thickness.

precipblwTF:
Tells whether precipitation was observed below cloud during the entire cloud measurement: true or false for >100 & >200 microns. Returns a dictionary: c.precipblwTF["100"] for precipitation >100 microns, and c.precipblwTF["200"] for precipitation >200 microns.

precipblwvpTF:
Tells whether precipitation was observed below a given vertical profile: true or false for >100 & >200 microns. Returns a dictionary: c.precipblwvpTF["100"] for precipitation >100 microns, and c.precipblwvpTF["200"] for precipitation >200 microns.

turbhrz:
Calculates the turbulence average over 20 km or more in a horizontal enough leg (the plane cannot move more than by a combined angle (phi+theta) of 2 degrees using a running standard deviation (of

25 points) to calculate the turbulence. Returns a dictionary: R["InCloud"]: List with one item for each stretch level enough for at least 20 km in cloud (N, first=0). Each item contains an array with the turbulence value c.turbhrz["InCloud"][N][0] and the distance over which it was calculated c.turbhrz["InCloud"][N][1]; c.turbhrz["BlwCloud"]: List with one item for each stretch level enough for at least 20 km below cloud (N, first=0). Each item contains an array with the turbulence value c.turbhrz["InCloud"][N][0] and the distance on which it was calculated c.turbhrz["InCloud"][N][1]; c.turbhrz["InCloudStretches"]: List with one item for each stretch level enough during at least 20 km in cloud (N, first=0). Each item contains an array with the indexes corresponding to the stretch used to calculate the corresponding turbulence value. c.turbhrz["BlwCloudStretches"]: List with one item for each stretch level enough during at least 20 km below cloud (N, first=0). Each item contains an array with the indexes corresponding to the stretch used to calculate the corresponding turbulence value. turbhrz handles the extradata module for updraft velocity.

angles:

Calculates the vertical angle of the aircraft (or other real or hypothetical aerial vehicle) based on TAS (true air speed) and altitude change. It also calculates the horizontal angle of the aircraft based on the longitude and latitude change. Returns a list with: time, vertical angle, horizontal angle (with one point less than the original time stamp in c.data). Note that altitude and TAS are both expected to be found with the basic aircraft data (c.data).