O3as: Ozone assessment service

B. Esteban, M. Hardt, T. Kerzenmacher, V. Kozlov (KIT)

Feb 02, 2022

OZONE SCIENCE

O3as is a service within the framework of the European Open Science Cloud - Synergy (EOSC-Synergy) project, mainly for scientists working on the Chemistry-Climate Model Initiative (CCMI) for the quadrennial global assessment of ozone depletion due in 2022.

The O3as service shall provide an invaluable tool to extract ozone trends from large climate prediction model data to produce figures of stratospheric ozone trends in publication quality, in a coherent way.

CHAPTER

ONE

OZONE INTRODUCTION

This introduces you to ozone science: why ozone is important for our everyday life, what is ozone assessment. We present then our O3as service, how it can be used for simple analysis and a more advanced one. In the advanced section we show the underlying software technologies we are using.

1.1 Role of ozone for our life

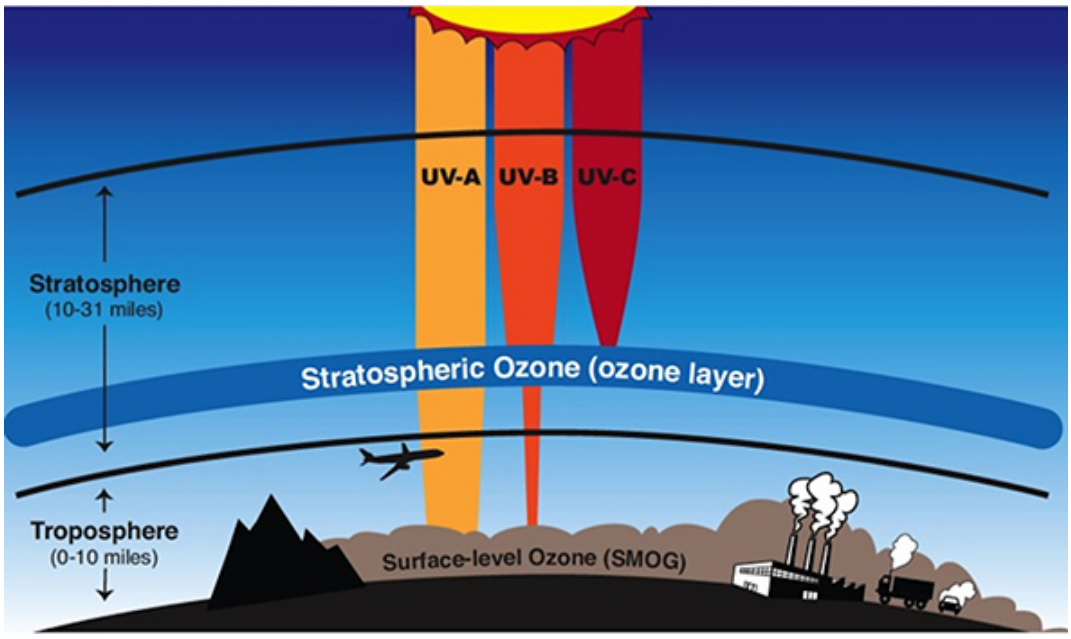


Fig. 1: The ozone layer in the stratosphere shields life on Earth from most UV-B and UV-C, the most harmful varieties of ultraviolet radiation. Credit: NASA

Ozone is a trace gas in the Earth's atmosphere, that means that it only occurs in very small quantities on Earth. Trace gases are gases that are not nitrogen (78.1%), oxygen (20.9%) or argon (0.934%), i.e. they make up only about 0.066% of the atmosphere (not including water which makes up about 4%). If all the ozone were compressed to the pressure at the Earth's surface (at 1013 hPa and 0°C) then the atmospheric content of ozone would make up a layer of 3 mm (which is equivalent to 300 Dobson Units (DU)). The highest content of ozone is found in the stratosphere at a height of about 25 to 30 km where the concentrations are about 10 parts per million parts of air (ppm), i.e. 0.001%.

The ozone present in the stratosphere at about 25 to 30 km is what makes up most ozone in the atmosphere. This ozone is important for life on planet Earth because it is responsible for removing most of the harmful ultraviolet (UV) radiation from the Sun. The longwave UV-A radiation (about 315–400 nm) is not absorbed, medium range UV-B

radiation (about 280–315 nm) is mostly absorbed and the most dangerous shortwave UV-C radiation (about 100–280 nm) is totally absorbed by the ozone layer. A decrease in the ozone layer increases the amount of UV-B radiation reaching the Earth's surface and the risk of skin cancer. Since UV radiation not only has an effect of the health of human beings it also has an effect on vegetation it is important to monitor stratospheric ozone.

1.2 Ozone hole and ozone assessment

With support of the World Meteorological Organization (WMO), the United Nations Environment Programme (UNEP), NASA, NOAA, and the European Commission periodic assessments of the state of ozone in the stratosphere are produced. Monitoring and the assessment of stratospheric ozone was started after the discovery of the ozone hole in the early eighties. Realizing that the decline of stratospheric ozone was due to harmful ozone destroying substances (ODSs) consisting of chlorofluorocarbons (CFCs) and hydrochlorofluorocarbons (HCFCs) the Montreal protocol (1987) and successor protocols were established to gradually curb the production of ODSs. The first assessment of the state of ozone was published in 1985, since 1994 they appear quadrennially.

CHAPTER

TWO

OZONE SIMULATIONS AND OBSERVATIONS (E.G. CCMI, SBUV ETC)

There is a huge effort to predict the future development of stratospheric ozone under different scenarios. These scenarios are modelled in different groups with different model setups/different boundary/initial conditions. In order to get a robust prediction many different model runs can be combined into ensembles. Model runs of total ozone often don't have same absolute values and are therefore normalized using satellite observations of ozone data. A date can then be picked at which the ozone of all the models agrees with the measured ozone. Only then they can be compared. For a more robust prediction of the behaviour of future ozone an average of all the model realizations can be produced to make a multi model mean so that a more robust prediction can be made.

CHAPTER

THREE

TOTAL COLUMN OZONE, ZONAL MEAN

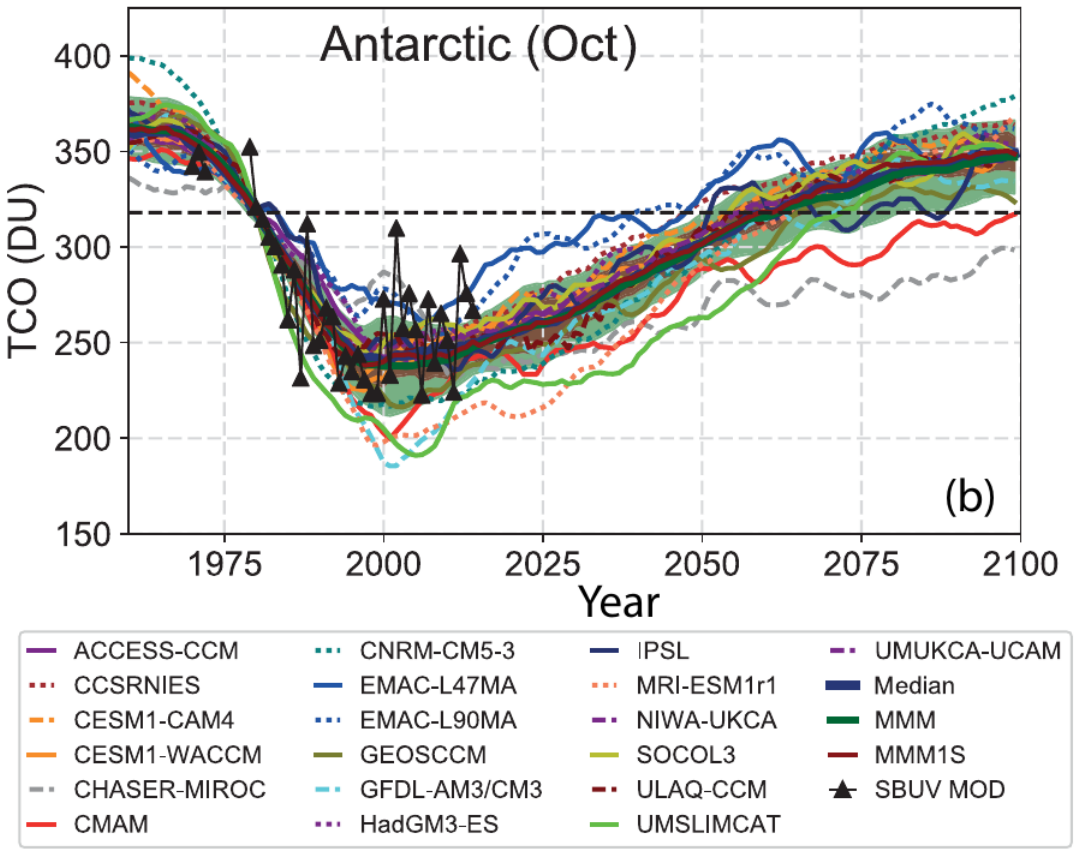


Fig. 1: Total column ozone time series for Antarctic in October from 19 individual CCMs for the REF-C2 simulations with respect to mean 1980–1984 observations. The dashed black line indicates 1980 reference value. Credit: S.S. Dhomse et al., Atmos. Chem. Phys., 18, 8409–8438, 2018

The model results for the ozone data can come in different units, like volume or mass mixing ratios, partial pressure, number densities or ozone density. Those units can be converted into each other:

Conversions of ozone volume mixing ratio (vmr) into common other units:

The volume mixing ratio is given in ppmv (parts per million, 10^{-6},) pressure in hPa and temperature in K, then:

- Ozone partial pressure pO_3 [nbar]:

  $pO_3 = vmrO_3 \times pressure$

- Ozone density O3[g/m^3]:

  O_3 = $vmrO_3$ × 577.3 × pressure / temperature

- Ozone mass mixing ratio mO_3 [g/g]:

  mO_3 = $vmrO_3$ × 1.66 (mol_m_O_3/mol_m_air = 48/28.9)

- Ozone number density NO_3 [molecules/cm^3]:

  NO_3 = $vmrO_3$ × pressure / (1.38 × 10^{-19} × temperature)

Also all these units can be converted to ozone volume mixing ratio (vmr) can be converted by inverting the above equations: The volume mixing ratio is given in ppmv, pressure in hPa and temperature in K.

- Ozone partial pressure pO_3 [nbar]:

  $vmrO_3$ = pO_3 / pressure

- Ozone density O_3 [g/m^3]:

  $vmrO_3$ = O_3 × 1.7322 × 10^{-3} × temperature / pressure

- Ozone mass mixing ratio mO_3 [g/g]:

  $vmrO_3$ = mO_3 × 0.602

- Ozone number density NO_3 [molecules/cm^3]:

  $vmrO_3$ = NO_3 × 1.38 × 10^{-19} × temperature / pressure

(http://www.meteo.mcgill.ca/hydroxyl/wiki/doku.php?id=ozone from Thilo Erbertseder, Frank Baier, last modified: October 2005)

Once ozone number density values are calculated, the values can be integrated over all heights to get a total column of ozone. Often the integration is done only partially either over the heights in the troposphere (from the ground to the tropopause) to get a tropospheric partial column of ozone or over the heights of the stratosphere (from the tropopause to the top of the atmosphere) to get a stratospheric partial column. The unit most often used for the column values of ozone is the Dobson unit:

- 1 Dobson Unit (DU) is:

  2.6867 × 10^{+20} molecules per square meter

  4.4615 × 10^{-04} mol per square meter

  2.1415 × 10^{-05} kilogram of ozone per square meter

Also, a Dobson Unit can also be seen as a measurement of thickness of the ozone column: A typical column amount of 300 DU of atmospheric ozone corresponds to a 3 mm layer of pure gas at the surface of the Earth.

It can be seen that the integration over altitude in m could be done over those other units (mol, kg) as well, instead of only the number densities.

Often we deal with large data sets. Since we are interested mainly in the latitudinal distribution of ozone zonally (E-W direction) averaged values of ozone can be used.

Now we have reduced the 4 dimensional dataset (time, latitude, longitude and altitude) into a two dimensional dataset (time, latitude) by first integrating over the altitudes and then taking the zonal mean across the longitudes. Of these values time series can be plotted to see the trend in future ozone over different latitude bands.

CHAPTER

FOUR

TOTAL COLUMN OZONE, RETURN YEARS

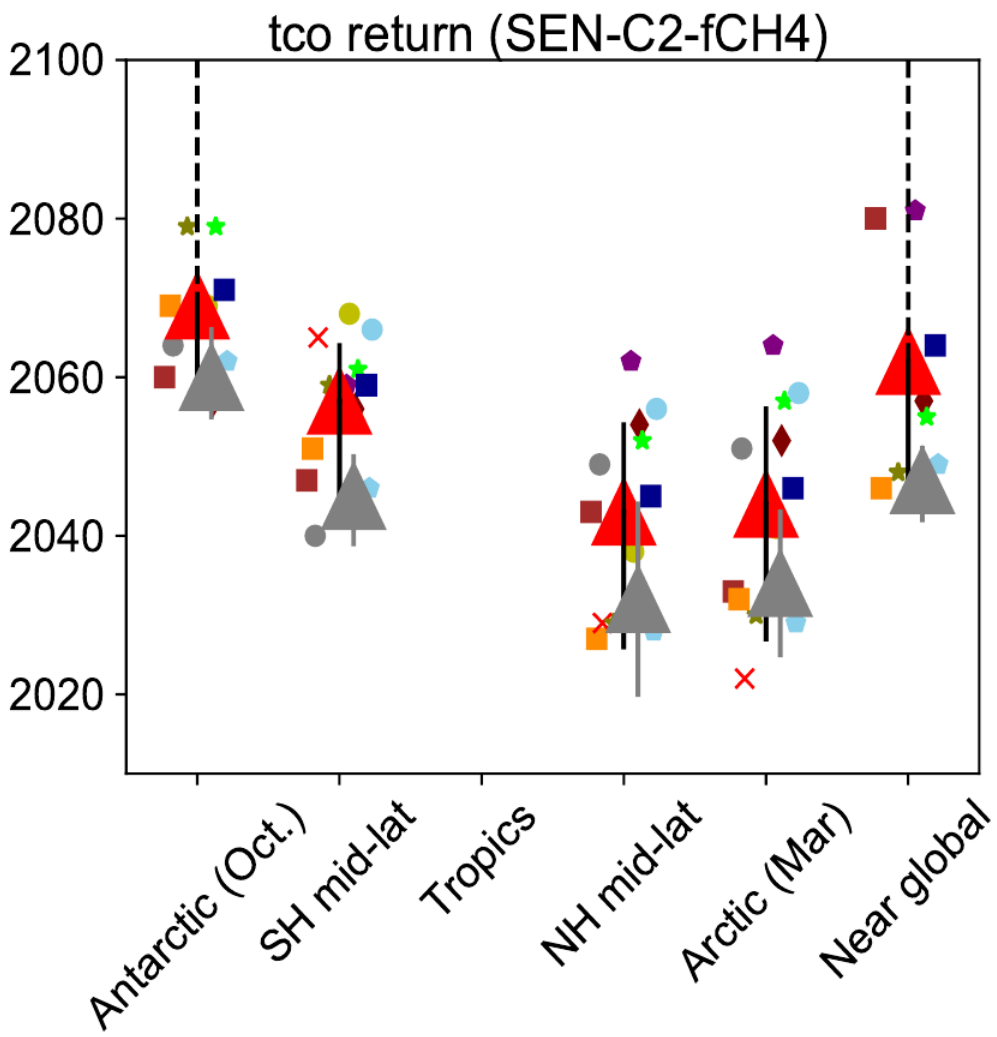


Fig. 1: Estimated MMM1S return dates (red triangles) of total column ozone from the SEN-C2-fCH4 simulations for different latitude bands. Credit: S.S. Dhomse et al., Supplement of Atmos. Chem. Phys., 18, 8409–8438, 2018

As mentioned in the previous section, the time series of total column ozone (zonally averaged) can be normalized using satellite observations of ozone data. Usually a year before the decline of ozone in the 1980s is used for the normalization. If the year 1980 is taken, then we use the measurements of the ozone column and move all the model time series up or down so that they pass through the point of measurement. Since ozone has been decreasing from that

point and has slowly recovered and is rising again, we are able to check when in the future ozone will be at the same level as in 1980 again. Having done this we can plot the time of ozone returning to 1980 level for each model and for different seasons/months/latitude bands.

CHAPTER

FIVE

QUIZZES ABOUT OZONE SCIENCE

5.1 Quiz: Ozone Introduction

1. Why is stratospheric ozone important for life on Earth?
2. How much ozone is present in the Earth's atmosphere?
3. What happened in the seventies that was discovered in the early eighties?
4. What was done to reduce the destruction of ozone?
5. What is done to publicize the state of ozone every four years?

5.2 Quiz: Total Column Ozone

1. What is the difference between a total column and a partial column of ozone?
2. What is a Dobson Unit?
3. How do you get from a data field of volume mixing ratios (latitude, longitude, height) into a total column? What steps are necessary?
4. What is a zonal mean?
5. Why would one use a multi model mean?
6. What is ozone return?
7. How can models be compared that have totally different values (offsets) from each other?

Wanna cross-check your answers? Have a look here

CHAPTER

SIX

O3AS PROJECT

6.1 Motivation

Monitoring and projecting stratospheric ozone is mandated by UN Environment to **safeguard a healthy planet**. Regularly many climate models project future climate and ozone change, producing **huge amounts of data** that have to be analysed for **key metrics**. Those key metrics help policy makers to judge if measures implemented to protect the stratospheric ozone layer are working.

6.2 Project goals

To provide a framework to efficiently explore ozone projections, including the calculation of key metrics:

- Improve the existing workflow and provide a **reliable tool** for scientists to perform analysis in a **more efficient** manner
- Ensure **reproducibility** of results
- **Simplify data access** and the use
- Publish high-level data to **citizens**

6.3 Problem to solve

- A typical workflow of today has many **manual** steps
- Full processing from raw data takes **hours**
- The code is **not always accessible** or well **maintained**

Consequences:
=> Plots are **not easy to rebuild** for various inputs
=> Possible **inconsistency** in results
=> **No-way to assess** the results by **non-specialists**

6.4 O3as solution

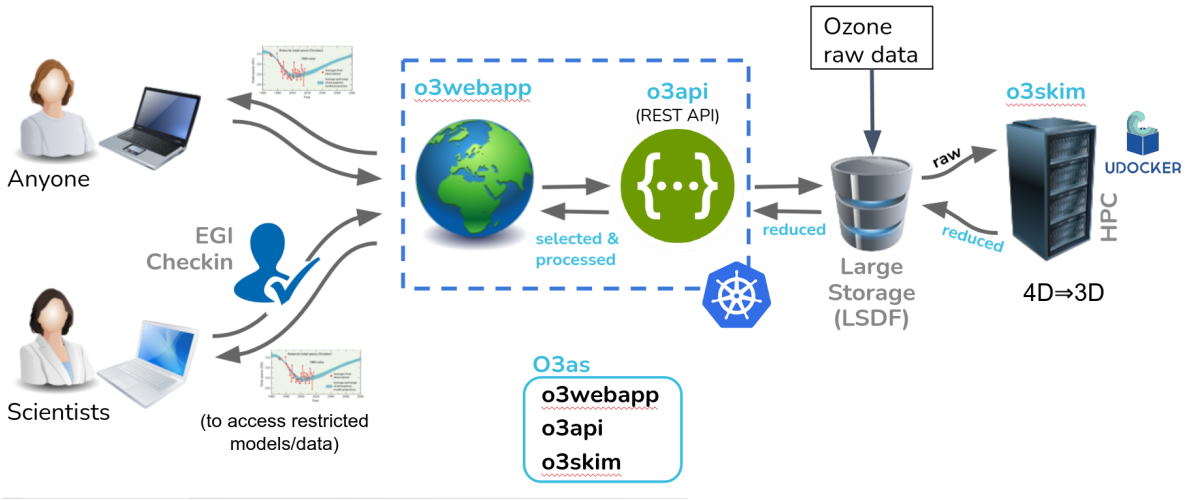


Fig. 1: O3as service general structure

- Climate Models data (10's TB) are collected in **one place** (Large Scale Data Facility, LSDF at KIT)
- The data are **reduced** to the parameters of interest and **homogenized** at HPC by the means of **o3skim** component
- The reduced data (100's MB) can be accessed with the **REST API** in seconds (**o3api** component)
- A user may do final processing and plotting by leveraging the **WebApp** (**o3webapp** *to come*)

All components are open source (GPLv3), documented, implemented with continuous integration and delivery (CI/CD) based on Jenkins (JePL), and dockerised: o3skim is run via udocker in HPC, o3api and o3webapp are in the cloud (Kubernetes cluster).

CHAPTER

SEVEN

GETTING STARTED

1. Main entry point to our service is https://o3as.data.kit.edu. There you find a general information about the service and links to corresponding resources like this documentation site, git repository, docker hub organization etc.

   Note: O3as Service is also offered via the EOSC Marketplace.

2. From the main website (https://o3as.data.kit.edu) you access our WebApp for a user-friendly communication with the service (to come) and O3as REST API for a programmatic usage of the service.
3. By using our service you agree to the Terms of Use.

   Important: Once your ozone plots are produced, please, **respect** data policies, references and acknowledgments for the O3as service and original data.

4. In the case of questions, suggestions, or for any feedback, please contact us.

CHAPTER

EIGHT

TUTORIALS

8.1 How to use API

8.1.1 How to use API (command line)

Introduction

This HowTo demonstrates basic communication with the O3as API by using cURL tool. Though cURL is a command-line tool, libcurl library is implemented in many programming languages (e.g. pycurl for Python). The examples below could also help understanding how the API calls can be implemented in your program using other means (e.g. urllib in Python).

Note: O3as API is based on the Open API 3 and Swagger, which provides Swagger UI, visual documentation allowing to interact with the API in a friendly way via web. The O3AS API Swagger UI webpage is accessible at https://api.o3as.fedcloud.eu/api/v1/ui/.

Tip: For the complete description of the REST API, please, check *O3API Endpoints* and https://api.o3as.fedcloud.eu/api/v1/ui/.

Info about available models

Get list of all available models

```
curl -X 'GET' 'https://api.o3as.fedcloud.eu/api/v1/models' -H 'accept: application/json'
```

List models with a certain pattern

e.g select models with 'refC2' in the name:

```
curl -X 'GET' \
'https://api.o3as.fedcloud.eu/api/v1/models?select=refc2' \
-H 'accept: application/json'
```

Get information about a particular model

e.g. about CCMI-1_ACCESS_ACCESS-CCM-refC2:

```
curl -X 'GET' \
'https://api.o3as.fedcloud.eu/api/v1/models/CCMI-1_ACCESS_ACCESS-CCM-refC2' \
-H 'accept: application/json'
```

Get default plot styles of selected models

e.g for `CCMI-1_ACCESS_ACCESS-CCM-refC2`, `CCMI-1_CCCma_CMAM-refC2`, `CCMI-1_CHASER-MIROC-ESM-refC2`

```
curl -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/models/plotstyle' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
```

(continues on next page)

(continued from previous page)

```
  "CCMI-1_ACCESS_ACCESS-CCM-refC2",
  "CCMI-1_CCCma_CMAM-refC2",
  "CCMI-1_CHASER-MIROC-ESM-refC2"
]'
```

Creating plots

Get list of possible plots

```
curl -X 'GET' \
  'https://api.o3as.fedcloud.eu/api/v1/plots' \
  -H 'accept: application/json'
```

Create tco3_zm plot

This example shows how to retrieve a *total column ozone (zonal mean)* figure either in PDF or JSON format. The query parameters used in the example are shown in the table below.

Tip: For the explanation of parameters, please, check O3API Endpoints

Table 1: Table: Query parameters to create a tco3_zm plot (in bold are required parameters)

Query parameter	Value
begin	1959
end	2100
month	[9,10,11]
lat_min	-90
lat_max	90
ref_meas	SBUV_GSFC_merged-SAT-ozone
ref_year	1980
Request body (for the list of models)	e.g. all refC2 models

Example for PDF (`tco3_zm.pdf` is created)

```
curl -o tco3_zm.pdf -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/plots/tco3_zm?begin=1959&end=2100&month=9,10,11&
↪lat_min=-90&lat_max=90&ref_meas=SBUV_GSFC_merged-SAT-ozone&ref_year=1980' \
  -H 'accept: application/pdf' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
```

(continues on next page)

(continued from previous page)

```
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

Example for JSON (`tco3_zm.json` is created)

```
curl -o tco3_zm.json -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/plots/tco3_zm?begin=1959&end=2100&month=9,10,11&
↪lat_min=-90&lat_max=90&ref_meas=SBUV_GSFC_merged-SAT-ozone&ref_year=1980' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

Create tco3_return plot

The example shows how to retrieve a *total column ozone (return years)* figure either in PDF or JSON format. The query parameters used in the example are shown in the table below.

Tip: For the explanation of parameters, please, check O3API Endpoints

Table 2: Table: Query parameters to create a tco3_return plot (in bold are required parameters)

Query parameter	Value
month	[9,10,11]
lat_min	-90
lat_max	90
ref_meas	SBUV_GSFC_merged-SAT-ozone
ref_year	1980
Request body (for the list of models)	e.g. all refC2 models

Example for PDF (`tco3_return.pdf` is created)

```
curl  -o tco3_return.pdf -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/plots/tco3_return?month=9,10,11&lat_min=-90&lat_
↪max=90&ref_meas=SBUV_GSFC_merged-SAT-ozone&ref_year=1980' \
  -H 'accept: application/pdf' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

Example for JSON (`tco3_return.json` is created)

```
curl  -o tco3_return.json -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/plots/tco3_return?month=9,10,11&lat_min=-90&lat_
↪max=90&ref_meas=SBUV_GSFC_merged-SAT-ozone&ref_year=1980' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

Info on skimmed data

o3skim component produces skimmed data which are further processed by the o3api to build plots for the Ozone assessment (e.g. *total column ozone (zonal mean)* and *total column ozone (return years)*).

Get list of plot types with the available skimmed data

```
curl -X 'GET' 'https://api.o3as.fedcloud.eu/api/v1/data' -H 'accept: application/json'
```

Get skimmed data used to build tco3_zm plot

Tip: For the explanation of parameters, please, check O3API Endpoints

Table 3: Table: Query parameters for retrieving skimmed data used to build a tco3_zm plot (in bold are required parameters)

Query parameter	Value
begin	1959
end	2100
month	[9,10,11]
lat_min	-90
lat_max	90
Request body (for the list of models)	e.g. all refC2 models

`tco3_zm_skimmed.json` is created:

```
curl -o tco3_zm_skimmed.json -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/data/tco3_zm?begin=1959&end=2100&month=9,10,11&
↪lat_min=-90&lat_max=90' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

Get skimmed data used to build tco3_return plot

Tip: For the explanation of parameters, please, check O3API Endpoints

Table 4: Table: Query parameters for retrieving skimmed data used to build a tco3_return plot (in bold are required parameters)

Query parameter	Value
begin	1959
end	2100
month	[9,10,11]
lat_min	-90
lat_max	90
Request body (for the list of models)	e.g. all refC2 models

`tco3_return_skimmed.json` is created:

```
curl -o tco3_return_skimmed.json -X 'POST' \
  'https://api.o3as.fedcloud.eu/api/v1/data/tco3_return?begin=1959&end=2100&month=9,10,
↪11&lat_min=-90&lat_max=90' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '[
"CCMI-1_ACCESS_ACCESS-CCM-refC2",
"CCMI-1_CCCma_CMAM-refC2",
"CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1",
"CCMI-1_CESM1-CAM4Chem_refC2_r3i1p1",
"CCMI-1_CESM1-WACCM_refC2_r1i1p1",
"CCMI-1_CESM1-WACCM_refC2_r2i1p1",
"CCMI-1_CESM1-WACCM_refC2_r3i1p1",
"CCMI-1_CHASER-MIROC-ESM-refC2",
"CCMI-1_CNRM-CERFACS_CNRM-CM5-3-refC2",
"CCMI-1_CNRM-CERFACS_MOCAGE-refC2",
"CCMI-1_ETH-PMOD_SOCOL3-refC2",
"CCMI-1_GSFC_GEOSCCM-refC2",
"CCMI-1_MESSy_EMAC-L90MA-refC2",
"CCMI-1_MOHC_HadGEM3-ES-refC2",
"CCMI-1_MRI_ESM1r1-refC2",
"CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2",
"CCMI-1_NIWA_NIWA-UKCA-refC2",
"CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2",
"CCMI-1_U-LAQUILA_CCM-refC2",
"CCMI-1_U-LEEDS_UMSLIMCAT-refC2"
]'
```

8.1.2 How to retrieve tco3 plots (jupyter)

This tutorial shows how to use a Jupyter notebook to retrieve data points of either **tco3_zm** plot or **tco3_return** plot in the JSON format and plot them. The Jupyter notebook can be downloaded from here.

System installations

If necessary, install required libraries or python packages

[1]:
```
### If needed, install additional modules:
#!pip3 install pandas
#!pip3 install matplotlib
### interactive plotting in jupyterlab requires node.js (pip does not install it!!):
#!apt update && apt install -y nodejs
#!pip3 install ipympl
#!jupyter labextension install @jupyter-widgets/jupyterlab-manager
#!jupyter labextension install jupyter-matplotlib
#!jupyter nbextension enable --py widgetsnbextension
### DON'T FORGET TO RESTART JupyterLab !!
# start jupyterlab:
# jupyter lab --ip=0.0.0.0
```

Import necessary packages

In this tutorial we need the following python packages:

json - to decode JSON data

matplotlib - to plot the data

numpy and pandas - to manipulate the data

os - to manipulate URL paths

requests - to communicate with the O3as API and retrieve data

[2]:
```
import json
import matplotlib.style as mplstyle
mplstyle.use('fast')
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import requests
```

Define global variables

we set a few variables: e.g. the base URL of O3AS API and the default figure size

[3]:
```
# for interactive plot, may change to widget, if installed
%matplotlib inline

# Set default size of plots
plt.rcParams['figure.figsize'] = [12, 8]

# A few variables needed for every REST API call:
url_o3api = "http://api.o3as.fedcloud.eu/api/v1/"  # base URL of the O3as API
headers = {'Content-Type': 'application/json',
           'Accept': 'application/json'}

# Define Reference Measurement and Reference Year
refMeasurement = 'SBUV_GSFC_merged-SAT-ozone'
refYear = 1980
```

Retrieve the list of models

We will analyse refC2 models, therefore we first request the list of corresponding models via the REST API

[4]:
```
# Use '/models' API Endpoint, i.e. append "models" to the base url_o3api:
url_o3api_tco3_zm_models = os.path.join(url_o3api, "models")

# request the list of modesl from O3as API, select "refC2" models:
tco3_zm_models = requests.request("GET",
                                  url=url_o3api_tco3_zm_models,
                                  params={'select': 'refc2'},
                                  headers=headers).json()
print(tco3_zm_models)
```

```
['CCMI-1_ACCESS_ACCESS-CCM-refC2', 'CCMI-1_CCCma_CMAM-refC2', 'CCMI-1_CESM1-CAM4Chem_
↪refC2_r1i1p1', 'CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1', 'CCMI-1_CESM1-CAM4Chem_refC2_
↪r3i1p1', 'CCMI-1_CESM1-WACCM_refC2_r1i1p1', 'CCMI-1_CESM1-WACCM_refC2_r2i1p1', 'CCMI-1_
↪CESM1-WACCM_refC2_r3i1p1', 'CCMI-1_CHASER-MIROC-ESM-refC2', 'CCMI-1_CNRM-CERFACS_CNRM-
↪CM5-3-refC2', 'CCMI-1_CNRM-CERFACS_MOCAGE-refC2', 'CCMI-1_ETH-PMOD_SOCOL3-refC2',
↪'CCMI-1_GSFC_GEOSCCM-refC2', 'CCMI-1_MESSy_EMAC-L90MA-refC2', 'CCMI-1_MOHC_HadGEM3-ES-
↪refC2', 'CCMI-1_MRI_ESM1r1-refC2', 'CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2', 'CCMI-1_NIWA_
↪NIWA-UKCA-refC2', 'CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2', 'CCMI-1_U-LAQUILA_CCM-refC2',
↪ 'CCMI-1_U-LEEDS_UMSLIMCAT-refC2']
```

We also add a reference measurement, **refMeasurement** (defined above in the Global Variables, e.g. SBUV_GSFC_merged-SAT-ozone), to the list of models to be plotted

[5]:
```
tco3_zm_models.append(refMeasurement)
```

Request points of the tco3_zm plot

1. we configure parameters of interest to get the plot. In the example we specify that we want a plot for:
   - Range of years: (1960, 2100)
   - month: September, October, November
   - South Hemisphere (SH, latitudes: -90, -60)
   - reference measurement: refMeasurement (e.g. SBUV_GSFC_merged-SAT-ozone)
   - reference year: refYear (e.g. 1980, see Global Variables)

[6]:
```
# initialize an empty dictionary
kwargs_tco3_zm = {}

# Build kwargs with tco3_zm parameters for O3as API.
# Keys have to correspond to expected by the API keys!

kwargs_tco3_zm = {
    'begin'   : 1960,
    'end'     : 2100,
    'month'   : '9,10,11',
    'lat_min' : -90,
    'lat_max' : -60,
    'ref_meas': refMeasurement,
    'ref_year': refYear
}
```

2. We want to retrieve "tco3_zm" plot: we use **'/plots'** endpoint, plot type 'tco3_zm', therefore we append 'plots/tco3_zm' to the base URL of the O3as API
3. Then we retrieve tco3_zm plot data for the parameters of interest and the list of refC2 models defined above

[7]:
```
# Build the API URL, use /plots/tco3_zm endpoint:
url_o3api_plot_tco3_zm = os.path.join(url_o3api, "plots/tco3_zm")

# Request data
response = requests.request("POST",
                            url=url_o3api_plot_tco3_zm,
                            params=kwargs_tco3_zm,
                            headers=headers,
                            data=json.dumps(tco3_zm_models))
# print(response.request.url)  # uncomment if you like to see the API call
# print(response.request.body) # uncomment if you like to see the API call Request Body

# Read the status_code. Normal response => 200
print(response.status_code)
tco3_zm_data = response.json()
print(tco3_zm_data[:5])  # print first five for cross-checking
```

```
200
[{'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html', 'model': 'CCMI-1_
↪ACCESS_ACCESS-CCM-refC2', 'plotstyle': {'color': 'purple', 'linestyle': 'solid',
↪'marker': ''}, 'x': ['1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967',
↪'1968', '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977', '1978',
↪ '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986', '1987', '1988', '1989
↪', '1990', '1991', '1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
↪'2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010',
↪ '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021
↪', '2022', '2023', '2024', '2025', '2026', '2027', '2028', '2029', '2030', '2031',
↪'2032', '2033', '2034', '2035', '2036', '2037', '2038', '2039', '2040', '2041', '2042',
↪ '2043', '2044', '2045', '2046', '2047', '2048', '2049', '2050', '2051', '2052', '2053
↪', '2054', '2055', '2056', '2057', '2058', '2059', '2060', '2061', '2062', '2063',
↪'2064', '2065', '2066', '2067', '2068', '2069', '2070', '2071', '2072', '2073', '2074',
↪ '2075', '2076', '2077', '2078', '2079', '2080', '2081', '2082', '2083', '2084', '2085
↪', '2086', '2087', '2088', '2089', '2090', '2091', '2092', '2093', '2094', '2095',
↪'2096', '2097', '2098', '2099', '2100'], 'y': [346.2776153564453, 346.2776153564453,␣
↪344.19089660644534, 345.2399810791016, 344.79651794433596, 343.2049926757813, 341.
↪64759521484376, 341.9790252685547, 339.6935241699219, 339.7173004150391, 336.
↪461865234375, 334.004965209961, 332.95016479492193, 330.1485961914063, 327.
↪3143676757813, 326.07980957031253, 323.82362060546876, 320.05503234863284, 317.
↪1924194335938, 314.53354187011723, 310.925, 307.60857543945315, 304.67973937988285,␣
↪302.41758117675784, 299.28678894042974, 295.69349060058596, 291.8769836425782, 287.
↪95793762207035, 286.21589965820317, 281.78271789550786, 277.63332672119145, 273.
↪80351562500005, 267.4904983520508, 261.4915878295899, 258.9700561523438, 255.
↪55068511962895, 251.66219329833987, 248.30885620117192, 244.19300231933596, 243.
↪56174621582034, 246.0753860473633, 243.7281616210938, 245.28644866943364, 247.
↪0736755371094, 245.87856597900395, 245.76735992431645, 246.93642578125002, 248.
↪34758758544925, 249.08704528808596, 247.95865173339848, 245.86597137451176, 247.
↪63198699951175, 247.2372985839844, 250.01163787841801, 250.87575683593752, 251.
↪80886383056645, 254.18242492675785, 253.67160949707034, 254.85982360839847, 256.
↪2275970458985, 255.62308349609378, 257.060238647461, 258.2514282226563, 256.
↪82307128906257, 257.3615539550782, 257.3642623901368, 256.9719482421875, 258.
↪62582092285163, 258.0639465332032, 259.03634338378913, 261.9813705444336, 261.
↪95432586669926, 263.4834625244141, 262.60323791503913, 264.1455886840821, 265.
↪7035308837891, 266.6639846801758, 268.2940048217774, 271.5132202148438, 275.
↪7468170166016, 279.5189178466797, 280.585482788086, 281.6898101806641, 284.
↪5627624511719, 288.2847290039063, 290.4588958740235, 291.752035522461, 293.
↪22461853027346, 297.2641906738282, 295.0239471435547, 296.0662506103516, 295.928125,␣
↪297.671859741211, 298.8583190917969, 298.3798034667969, 299.8877807617188, 301.
↪73552856445315, 302.51728515625, 301.0359985351563, 303.35169677734376, 302.
↪41975708007817, 305.3851501464844, 307.1610443115235, 308.31840515136724, 310.
↪8793548583985, 311.1203918457032, 312.7027984619141, 312.83945007324223, 314.
↪6934448242188, 317.3296325683594, 317.4012817382813, 318.46606750488286, 321.
↪8202758789063, 323.72616577148443, 322.7880340576172, 325.6069763183594, 326.
↪15849609375005, 330.8311218261719, 330.8529205322266, 328.95618896484376, 331.
↪4684204101563, 331.71318359375005, 329.0909851074219, 331.5364959716797, 331.
↪30587768554693, 330.81463317871095, 331.09461364746096, 329.43330078125, 330.
↪2022674560547, 331.39458923339845, 331.2634979248047, 332.8045166015625, 334.
↪02109680175784, 333.03064575195316, 334.6632781982422, 335.09021911621096, 334.
↪4844055175782, 334.8290222167969, 335.1350799560547, 336.1991668701172, 336.
↪1889068603516]}, {'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html',
↪'model': 'CCMI-1_CCCma_CMAM-refC2', 'plotstyle': {'color': 'red', 'linestyle': 'solid',
↪ 'marker': ''}, 'x': ['1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967',
↪'1968', '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977', '1978',
↪ '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986', '1987', '1988', '1989
↪', '1990', '1991', '1992', '1993', '1994', '1995', '1996', '1997', '1998', '1999',
↪'2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010',
↪ '2011', '2012', '2013', '2014', '2015', '2016', '2017', '2018', '2019', '2020', '2021
↪', '2022', '2023', '2024', '2025', '2026', '2027', '2028', '2029', '2030', '2031',
```

(continues on next page)

(continued from previous page)

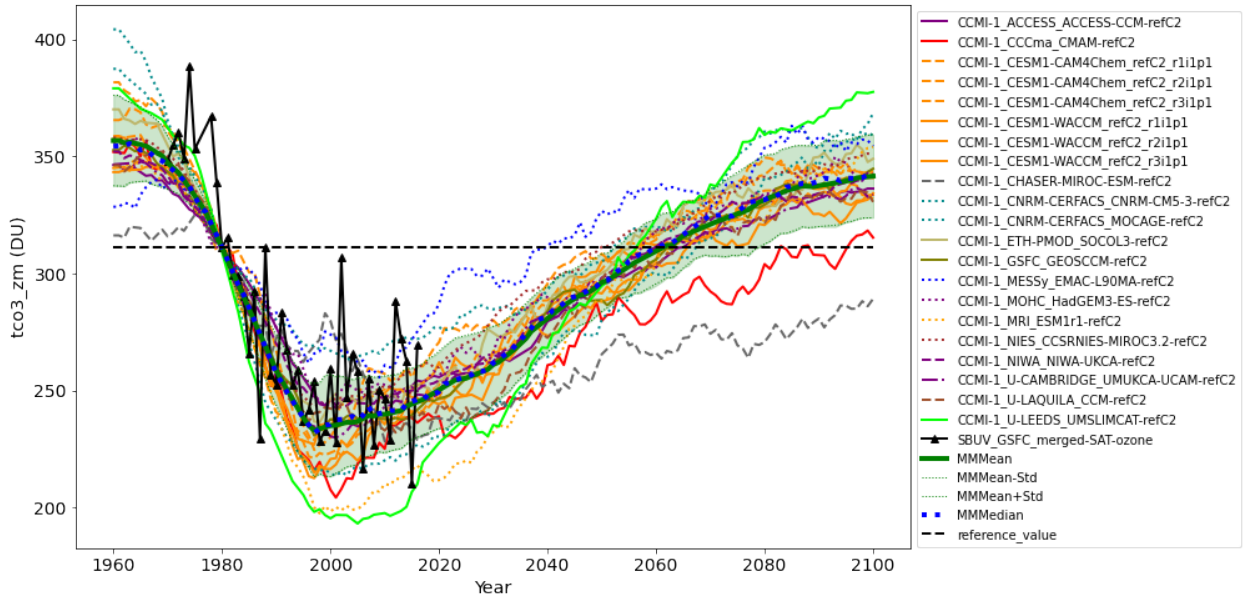
4. The received JSON data are converted into pandas Dataframe for an easier data manipulation

[8]:
```
tco3_zm_pd = pd.json_normalize(tco3_zm_data)
```

5. Finally, we plot the retrieved data:

[9]:
```
# set default linewidth:
linewidth_default = 2
tco3_zm_pd_plot = tco3_zm_pd.copy()

if not 'plotstyle.linewidth' in tco3_zm_pd_plot.columns:
  tco3_zm_pd_plot['plotstyle.linewidth'] = linewidth_default
else:
  tco3_zm_pd_plot['plotstyle.linewidth'] = tco3_zm_pd_plot['plotstyle.linewidth'].
↪fillna(linewidth_default)

# plot every model
for index, c in tco3_zm_pd_plot.iterrows():
  plt.plot(pd.to_datetime(c['x']), c['y'], label=c['model'],
           color=c['plotstyle.color'], linestyle=c['plotstyle.linestyle'],
           marker=c['plotstyle.marker'], linewidth=c['plotstyle.linewidth'])

# let's fill the area of (MMMean +/- 1 Std)
tco3_zm_mmmean_minus_std_pd = tco3_zm_pd_plot[tco3_zm_pd_plot['model']=='MMMean-Std']
tco3_zm_mmmean_plus_std_pd = tco3_zm_pd_plot[tco3_zm_pd_plot['model']=='MMMean+Std']

plt.fill_between(pd.to_datetime(tco3_zm_mmmean_minus_std_pd['x'].iloc[0]),
                 tco3_zm_mmmean_minus_std_pd['y'].iloc[0],
                 tco3_zm_mmmean_plus_std_pd['y'].iloc[0],
                 color='g', alpha=0.2);

# tune the plot
# show the legend
ax = plt.gca() # get axis instance
ax.legend(bbox_to_anchor=(1.0, 1.0))

plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# put titels for axes
plt.xlabel('Year', fontsize='x-large')
plt.ylabel('tco3_zm (DU)', fontsize='x-large')
```

[9]:
```
Text(0, 0.5, 'tco3_zm (DU)')
```



Request points of the tco3_return plot

1. Similar to the previous task, we configure parameters of interest to get the **tco3_return** plot. In the example we specify that we want a plot for:

- month: September, October, November
- South Hemisphere (SH, latitudes: -90, -60)
- reference measurement: refMeasurement (e.g. SBUV_GSFC_merged-SAT-ozone, see Global Variables)
- reference year: refYear (e.g. 1980, see Global Variables)
- we don't provide a specific range of years, like in the previous example, but use all available data

[10]:
```
# initialize an empty dictionary
kwargs_tco3_return = {}

# Build kwargs with tco3_zm parameters for O3as API.
# Keys have to correspond to expected by the API keys!

kwargs_tco3_return = {
    'month'   : '9,10,11',
    'lat_min' : -90,
    'lat_max' : -60,
    'ref_meas': refMeasurement,
    'ref_year': refYear
}
```

2. We want to retrieve "tco3_retrun" plot: we use '/plots' endpoint, plot type 'tco3_return', therefore we append 'plots/tco3_return' to the base URL of the O3as API
3. Then we retrieve tco3_retrun plot data for the parameters of interest and the list of refC2 models defined above (this time we exclude the reference measurement)

[11]:
```
# Build the API URL, use /plots/tco3_return endpoint:
url_o3api_plot_tco3_return = os.path.join(url_o3api, "plots/tco3_return")

# Request data
response = requests.request("POST",
                            url=url_o3api_plot_tco3_return,
                            params=kwargs_tco3_return,
                            headers=headers,
                            data=json.dumps(tco3_zm_models))
# Read the status_code. Normal response => 200
print(response.status_code)
tco3_return_data = response.json()
print(tco3_return_data[:5])  # print first five for cross-checking
```

```
200
[{'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html', 'model': 'CCMI-1_
↪ACCESS_ACCESS-CCM-refC2', 'plotstyle': {'color': 'purple', 'linestyle': 'none', 'marker
↪': 'o'}, 'x': ['Antarctic(Oct)', 'SH mid-lat', 'Tropics', 'NH mid-lat', 'Arctic(Mar)',
↪'Near global', 'Global', 'User region'], 'y': [2064, 2051, 2060, 2041, 2040, 2049,␣
↪2052, 2065]}, {'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html',
↪'model': 'CCMI-1_CCCma_CMAM-refC2', 'plotstyle': {'color': 'red', 'linestyle': 'none',
↪'marker': 'x'}, 'x': ['Antarctic(Oct)', 'SH mid-lat', 'NH mid-lat', 'Arctic(Mar)',
↪'Near global', 'Global', 'User region'], 'y': [2087.0, 2049.0, 1986.0, 1986.0, 2045.0,␣
↪2048.0, 2083.0]}, {'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html',
↪ 'model': 'CCMI-1_CESM1-CAM4Chem_refC2_r1i1p1', 'plotstyle': {'color': 'darkorange',
↪'linestyle': 'none', 'marker': 'd'}, 'x': ['Antarctic(Oct)', 'SH mid-lat', 'Tropics',
↪'NH mid-lat', 'Arctic(Mar)', 'Near global', 'Global', 'User region'], 'y': [2056, 2049,
↪ 2026, 1987, 2009, 2026, 2038, 2057]}, {'legalinfo': 'https://o3as.data.kit.edu/
↪policies/terms-of-use.html', 'model': 'CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1', 'plotstyle
↪': {'color': 'darkorange', 'linestyle': 'none', 'marker': 'd'}, 'x': ['Antarctic(Oct)',
↪ 'SH mid-lat', 'Tropics', 'NH mid-lat', 'Arctic(Mar)', 'Near global', 'Global', 'User␣
↪region'], 'y': [2057, 2038, 2022, 2006, 2017, 2026, 2036, 2058]}, {'legalinfo': 'https:
↪//o3as.data.kit.edu/policies/terms-of-use.html', 'model': 'CCMI-1_CESM1-CAM4Chem_refC2_
↪r3i1p1', 'plotstyle': {'color': 'darkorange', 'linestyle': 'none', 'marker': 'd'}, 'x':
↪ ['Antarctic(Oct)', 'SH mid-lat', 'Tropics', 'NH mid-lat', 'Arctic(Mar)', 'Near global
↪', 'Global', 'User region'], 'y': [2055, 2037, 2025, 2002, 2020, 2022, 2034, 2054]}]
```

4. The received JSON data can be converted into pandas Dataframe for an easier data manipulation

[12]:
```
tco3_return_pd = pd.json_normalize(tco3_return_data)
```

5. Finally, we plot the retrieved data:

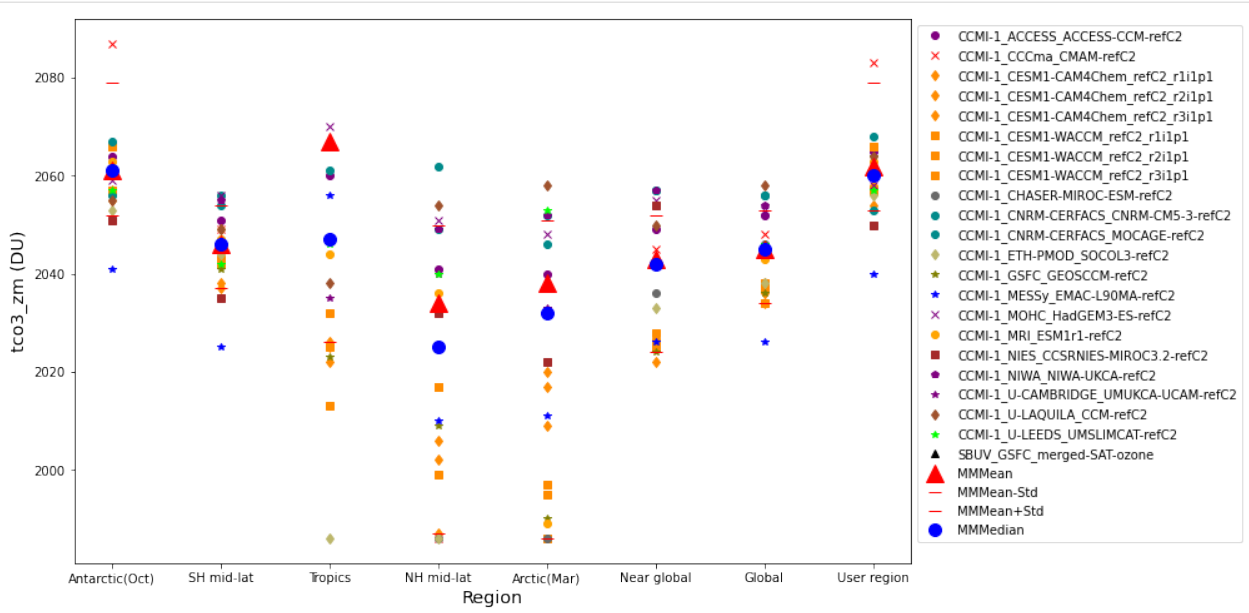
[13]:
```
# set default markersize:
markersize_default = 6
tco3_return_pd_plot = tco3_return_pd.copy()

if not 'plotstyle.markersize' in tco3_return_pd_plot.columns:
  tco3_return_pd_plot['plotstyle.markersize'] = markersize_default
else:
  tco3_return_pd_plot['plotstyle.markersize'] = tco3_return_pd_plot['plotstyle.markersize
↪'].fillna(markersize_default)

# plot every model
```

(continues on next page)

(continued from previous page)

```
for index, c in tco3_return_pd_plot.iterrows():
  plt.plot(c['x'], c['y'], label=c['model'],
           color=c['plotstyle.color'], linestyle=c['plotstyle.linestyle'],
           marker=c['plotstyle.marker'], markersize=c['plotstyle.markersize'])

# show the legend
ax = plt.gca() # get axis instance
ax.legend(bbox_to_anchor=(1.0, 1.0))

# put titels for axes
plt.xlabel('Region', fontsize='x-large')
plt.ylabel('tco3_zm (DU)', fontsize='x-large')
```

[13]:
```
Text(0, 0.5, 'tco3_zm (DU)')
```



The example is how to use a Jupyter notebook to retrieve data points of either *tco3_zm plot* or *tco3_return plot* in the JSON format and plot them. The corresponding Jupyter notebook can be downloaded here .

8.1.3 How to retrieve data, analyse, and plot tco3 plots (jupyter)

This tutorial shows how to use a Jupyter notebook to retrieve **skimmed data** points in the JSON format and analyse for either **tco3_zm** or **tco3_return** plot. The Jupyter notebook can be downloaded from here.

System installations

If necessary, install required libraries or python packages

[1]:
```
### If needed, install additional modules:
#!pip3 install pandas
#!pip3 install matplotlib
### interactive plotting in jupyterlab requires node.js (pip does not install it!!):
#!apt update && apt install -y nodejs
#!pip3 install ipympl
#!jupyter labextension install @jupyter-widgets/jupyterlab-manager
#!jupyter labextension install jupyter-matplotlib
#!jupyter nbextension enable --py widgetsnbextension
### DON'T FORGET TO RESTART JupyterLab !!
# start jupyterlab:
# jupyter lab --ip=0.0.0.0
```

Import necessary packages

In this tutorial we need the following python packages:

json - to decode JSON data

matplotlib - to plot the data

numpy and pandas - to manipulate the data

os - to manipulate URL paths

requests - to communicate with the O3as API and retrieve data

scipy - to apply boxcar smoothing on the skimmed data

[2]:
```
import json
import matplotlib.style as mplstyle
mplstyle.use('fast')
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import requests
from scipy import signal
```

Define global variables

we set a few variables: e.g. the base URL of O3AS API and the default figure size

[3]:
```
debug = False  # global debug flag, set to True to get more print-outs

pd.options.display.max_columns = None  # prints all columns of the pandas' DataFrame

# for interactive plot, may change to widget, if installed
%matplotlib inline
```

(continues on next page)

(continued from previous page)

```
# Set default size of plots
plt.rcParams['figure.figsize'] = [12, 8]

# A few variables needed for every REST API call:
url_o3api = "http://api.o3as.fedcloud.eu/api/v1/"  # base URL of the O3as API
headers = {'Content-Type': 'application/json',
           'Accept': 'application/json'}

# Define Reference Measurement and Reference Year
refMeasurement = 'SBUV_GSFC_merged-SAT-ozone'
refYear = 1980
```

Retrieve list of models

We will analyse refC2 models, therefore we first request the list of corresponding models via the REST API

[4]:
```
# Use '/models' API Endpoint, i.e. append "models" to the base url_o3api:
url_o3api_tco3_zm_models = os.path.join(url_o3api, "models")

# request the list of modesl from O3as API, select "refC2" models:
tco3_zm_models = requests.request("GET",
                                  url=url_o3api_tco3_zm_models,
                                  params={'select': 'refc2'},
                                  headers=headers).json()
print(tco3_zm_models)
```

```
['CCMI-1_ACCESS_ACCESS-CCM-refC2', 'CCMI-1_CCCma_CMAM-refC2', 'CCMI-1_CESM1-CAM4Chem_
↪refC2_r1i1p1', 'CCMI-1_CESM1-CAM4Chem_refC2_r2i1p1', 'CCMI-1_CESM1-CAM4Chem_refC2_
↪r3i1p1', 'CCMI-1_CESM1-WACCM_refC2_r1i1p1', 'CCMI-1_CESM1-WACCM_refC2_r2i1p1', 'CCMI-1_
↪CESM1-WACCM_refC2_r3i1p1', 'CCMI-1_CHASER-MIROC-ESM-refC2', 'CCMI-1_CNRM-CERFACS_CNRM-
↪CM5-3-refC2', 'CCMI-1_CNRM-CERFACS_MOCAGE-refC2', 'CCMI-1_ETH-PMOD_SOCOL3-refC2',
↪'CCMI-1_GSFC_GEOSCCM-refC2', 'CCMI-1_MESSy_EMAC-L90MA-refC2', 'CCMI-1_MOHC_HadGEM3-ES-
↪refC2', 'CCMI-1_MRI_ESM1r1-refC2', 'CCMI-1_NIES_CCSRNIES-MIROC3.2-refC2', 'CCMI-1_NIWA_
↪NIWA-UKCA-refC2', 'CCMI-1_U-CAMBRIDGE_UMUKCA-UCAM-refC2', 'CCMI-1_U-LAQUILA_CCM-refC2',
↪ 'CCMI-1_U-LEEDS_UMSLIMCAT-refC2']
```

We also add a reference measurement, **refMeasurement** (defined above, e.g. SBUV_GSFC_merged-SAT-ozone), to the list of models to be plotted

[5]:
```
tco3_zm_models.append(refMeasurement)
```

Request skimmed data to analyse and to build tco3 plots

1. we configure parameters of interest to get the data. In the example we specify that we want a plot for:

- Range of years: (1960, 2100)
- month: September, October, November
- South Hemisphere (SH, latitudes: -90, -60)

[6]:
```
# initialize an empty dictionary
kwargs_tco3_zm = {}

# Build kwargs with tco3_zm parameters for O3as API '/data' Endpoint.
# Keys have to correspond to expected by the API keys!

kwargs_tco3_zm = {
    'begin'   : 1960,
    'end'     : 2100,
    'month'   : '9,10,11',
    'lat_min' : -90,
    'lat_max' : -60
}
```

2. We want to retrieve skimmed data to analyse and build "tco3_zm" plot: we use **'/data'** endpoint, plot type **'tco3_zm'**, therefore we append **'data/tco3_zm'** to the base URL of the O3as API
3. Then we retrieve skimmed data for the tco3_zm plot for the parameters of interest and the list of refC2 models defined above

[7]:
```
# Build the API URL, use /data/tco3_zm endpoint:
url_o3api_data_tco3_zm = os.path.join(url_o3api, "data/tco3_zm")

# Request data
response = requests.request("POST",
                            url=url_o3api_data_tco3_zm,
                            params=kwargs_tco3_zm,
                            headers=headers,
                            data=json.dumps(tco3_zm_models))
if debug: print(response.request.url)  # debug: print the configured API call

# Read the status_code. Normal response => 200
print(response.status_code)
tco3_skim_data = response.json()
print(tco3_skim_data[:5])  # print first five for cross-checking
```

```
200
[{'legalinfo': 'https://o3as.data.kit.edu/policies/terms-of-use.html', 'model': 'CCMI-1_
↪ACCESS_ACCESS-CCM-refC2', 'plotstyle': {'color': 'purple', 'linestyle': 'solid',
↪'marker': 'o'}, 'x': ['1960-09-16 00:00:00', '1960-10-16 00:00:00', '1960-11-16 00:00:
↪00', '1961-09-16 00:00:00', '1961-10-16 00:00:00', '1961-11-16 00:00:00', '1962-09-16␣
↪00:00:00', '1962-10-16 00:00:00', '1962-11-16 00:00:00', '1963-09-16 00:00:00', '1963-
↪10-16 00:00:00', '1963-11-16 00:00:00', '1964-09-16 00:00:00', '1964-10-16 00:00:00',
↪'1964-11-16 00:00:00', '1965-09-16 00:00:00', '1965-10-16 00:00:00', '1965-11-16 00:00:
↪00', '1966-09-16 00:00:00', '1966-10-16 00:00:00', '1966-11-16 00:00:00', '1967-09-16␣
↪00:00:00', '1967-10-16 00:00:00', '1967-11-16 00:00:00', '1968-09-16 00:00:00', '1968-
↪10-16 00:00:00', '1968-11-16 00:00:00', '1969-09-16 00:00:00', '1969-10-16 00:00:00',
↪'1969-11-16 00:00:00', '1970-09-16 00:00:00', '1970-10-16 00:00:00', '1970-11-16 00:00:
↪00', '1971-09-16 00:00:00', '1971-10-16 00:00:00', '1971-11-16 00:00:00', '1972-09-16␣
↪00:00:00', '1972-10-16 00:00:00', '1972-11-16 00:00:00', '1973-09-16 00:00:00', '1973-
↪10-16 00:00:00', '1973-11-16 00:00:00', '1974-09-16 00:00:00', '1974-10-16 00:00:00',
↪'1974-11-16 00:00:00', '1975-09-16 00:00:00', '1975-10-16 00:00:00', '1975-11-16 00:00:
↪00', '1976-09-16 00:00:00', '1976-10-16 00:00:00', '1976-11-16 00:00:00', '1977-09-16␣
↪00:00:00', '1977-10-16 00:00:00', '1977-11-16 00:00:00', '1978-09-16 00:00:00', '1978-
↪10-16 00:00:00', '1978-11-16 00:00:00', '1979-09-16 00:00:00', '1979-10-16 00:00:00',
↪'1979-11-16 00:00:00', '1980-09-16 00:00:00', '1980-10-16 00:00:00', '1980-11-16 00:00:
↪00', '1981-09-16 00:00:00', '1981-10-16 00:00:00', '1981-11-16 00:00:00', '1982-09-16␣
↪00:00:00', '1982-10-16 00:00:00', '1982-11-16 00:00:00', '1983-09-16 00:00:00', '1983-
↪10-16 00:00:00', '1983-11-16 00:00:00', '1984-09-16 00:00:00', '1984-10-16 00:00:00',
↪'1984-11-16 00:00:00', '1985-09-16 00:00:00', '1985-10-16 00:00:00', '1985-11-16 00:00:
↪00', '1986-09-16 00:00:00', '1986-10-16 00:00:00', '1986-11-16 00:00:00', '1987-09-16␣
```

(continued on next page)

(continued from previous page)

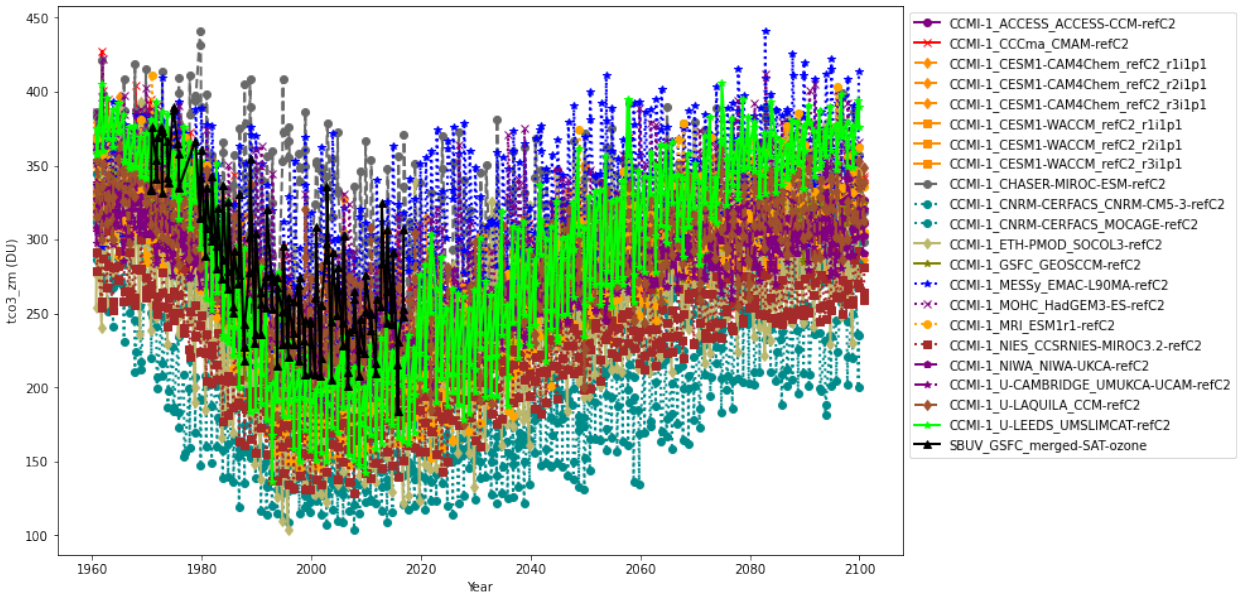
4. The received JSON data are converted into pandas Dataframe for an easier data manipulation

[8]:
```
tco3_skim_pd = pd.json_normalize(tco3_skim_data)
if debug: print(tco3_skim_pd.head())
```

5. Let's plot the retrieved skimmed data:

[9]:
```
# set default linewidth:
linewidth_default = 2
if not 'plotstyle.linewidth' in tco3_skim_pd.columns:
  tco3_skim_pd['plotstyle.linewidth'] = linewidth_default

# plot every model
for index, c in tco3_skim_pd.iterrows():
  plt.plot(pd.to_datetime(c['x']), c['y'], label=c['model'],
           color=c['plotstyle.color'], linestyle=c['plotstyle.linestyle'],
           marker=c['plotstyle.marker'], linewidth=c['plotstyle.linewidth'])

# show the legend
ax = plt.gca() # get axis instance
ax.legend(bbox_to_anchor=(1.0, 1.0))

# put titels for axes
plt.xlabel('Year')
plt.ylabel('tco3_zm (DU)')
```

[9]:
```
Text(0, 0.5, 'tco3_zm (DU)')
```



Analyse the skimmed data and plot tco3_zm

To analyse the data and produce tco3_zm plot, we do:

1. Define a help function for boxcar smoothing
2. Transform the Pandas Dataframe for easier data manipulation
3. Average skimmed data over one year
4. Apply boxcar smoothing, keep the reference year as-it-is, i.e. without smoothing
5. For the reference year, find the TCO3 value in the measured data (reference value)
6. Find biases in the simulation data in relation to the reference year
7. Shift all simulations to the reference value at the reference year
8. Plot shifted models together with the reference value.

Define a help function for boxcar smoothing

[10]:
```
def boxcar(data, bwin):
  """Function to apply boxcar, following
  https://scipy-cookbook.readthedocs.io/items/SignalSmooth.html
  N.B. 'valid' replaced with 'same' !

  :param data: input data
  :param bwin: width of the boxcar window
  :return boxcar_values: smoothed values
  """
  debug_boxcar = False

  boxcar = np.ones(bwin)

  if debug_boxcar: print("signal(raw) (len={}): {}".format(len(data),data))

  # mirror start and end of the original signal:
  sgnl = np.r_[data[bwin-1:0:-1],data,data[-2:-bwin-1:-1]]
  boxcar_values = signal.convolve(sgnl,
                                  boxcar,
                                  mode='same')/bwin
  if debug_boxcar: print("signal (len={}): {}".format(len(sgnl),sgnl))
  if debug_boxcar: print("signal+boxcar (len={}): {}".format(len(boxcar_values),
                                                            boxcar_values))
  return boxcar_values[bwin-1:-(bwin-1)]
```

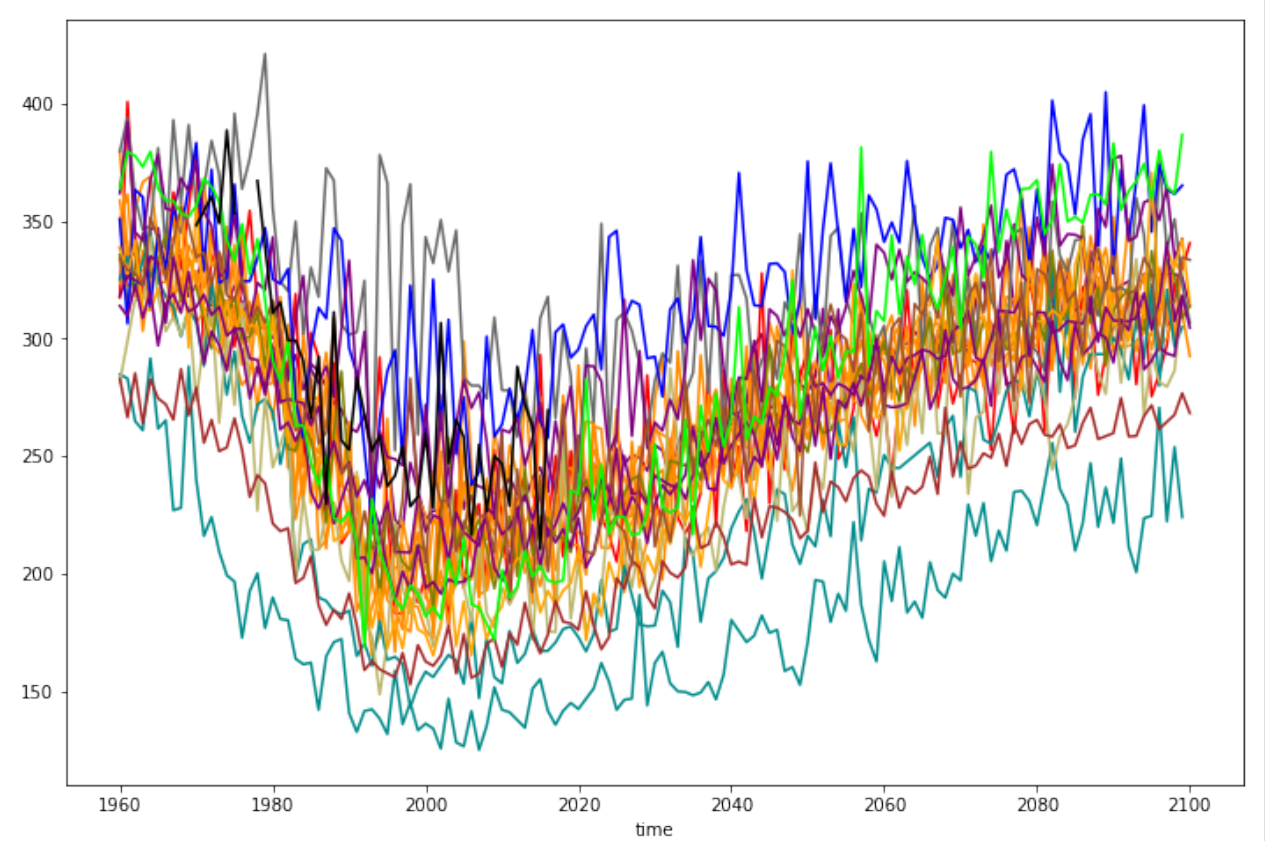
Transform the DataFrame

[11]:
```
tco3_zm_pd = pd.DataFrame({'time': pd.to_datetime(tco3_skim_pd.loc[0,'x']),
                           tco3_skim_pd.loc[0,'model']: tco3_skim_pd.loc[0,'y']})

for i in range(1, len(tco3_skim_pd)):
  curve = pd.DataFrame({'time': pd.to_datetime(tco3_skim_pd.loc[i,'x']),
                        tco3_skim_pd.loc[i,'model']: tco3_skim_pd.loc[i,'y']})
  tco3_zm_pd = tco3_zm_pd.merge(curve, how='outer', on=['time'], sort=True)

tco3_zm_pd = tco3_zm_pd.set_index('time')
tco3_zm_pd = tco3_zm_pd.replace({0: np.nan})

# print(tco3_zm_pd.head()) # uncomment if you like to see the created DataFrame
```

Average data over one year

[12]:
```
# Average model data per year, index -> year
tco3_zm_year_pd = tco3_zm_pd.groupby([tco3_zm_pd.index.year]).mean()

# we may use pandas.plot() immediately
# Optionally, we can set matplotlib colors to the default values specified by the API:
import matplotlib as mpl
from cycler import cycler
mpl.rcParams['axes.prop_cycle'] = cycler(color=tco3_skim_pd['plotstyle.color'].values)

tco3_zm_year_pd.plot(legend=False)
```

[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49e7fde350>
```

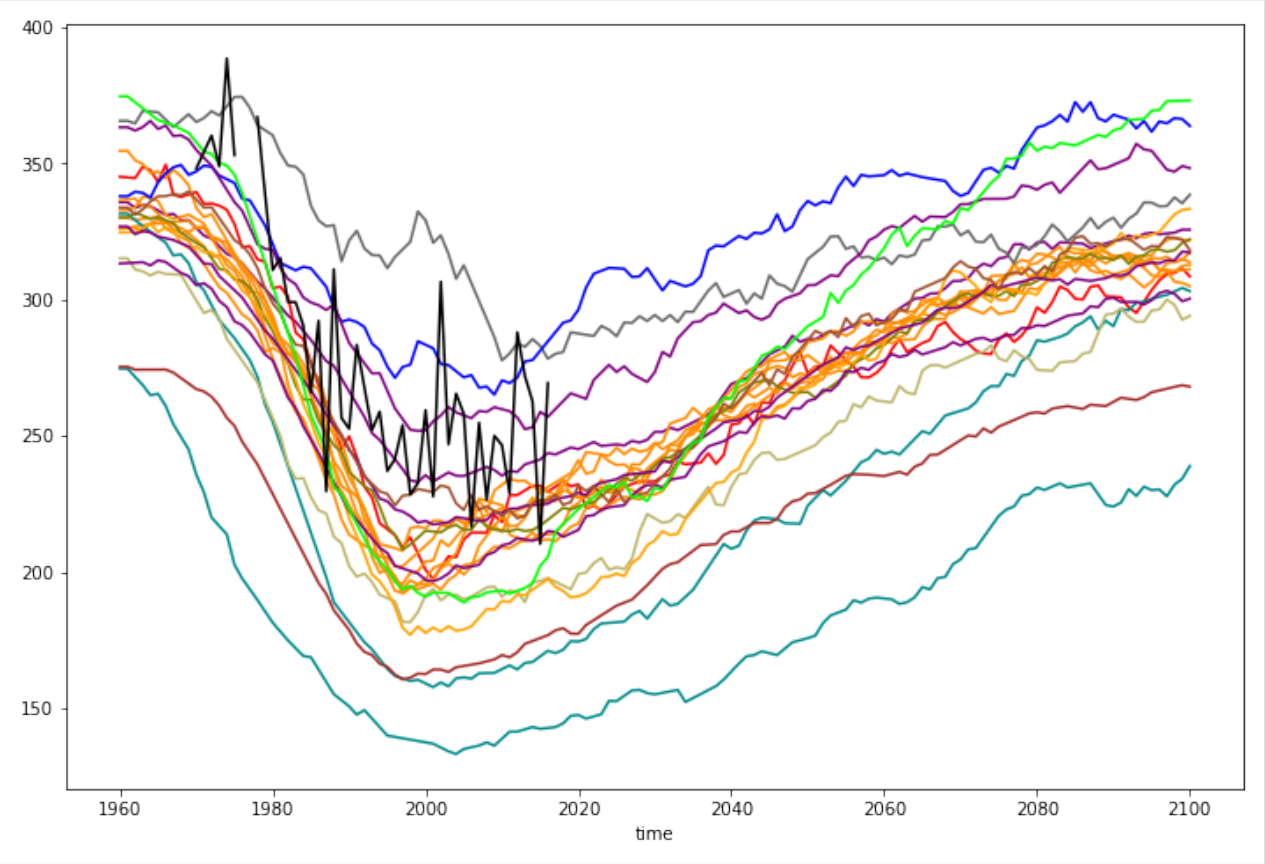


Apply boxcar smoothing

we use 10-point boxcar smoothing, when applied after averaging per year, it is an equivalent of 10-years

[13]:
```
boxcar_window = 10 # boxcar window for smoothing

# different models may have data point until a different year
# if last years have NaNs fill them with "before NaN values"
last_year = tco3_zm_year_pd.index.values[-1]
tco3_zm_year_pd[tco3_zm_year_pd.index > (last_year - boxcar_window)] = tco3_zm_year_
↪pd[tco3_zm_year_pd.index > (last_year - boxcar_window)].fillna(method='ffill')

# apply boxcar smoothing
tco3_zm_smooth_pd = tco3_zm_year_pd.apply(boxcar, args = [boxcar_window], axis = 0,␣
↪result_type = 'broadcast')

# inject the reference measurement (refMeasurement) as-it-is, i.e. **without** smoothing
if refMeasurement in tco3_zm_smooth_pd.columns:
  tco3_zm_smooth_pd[refMeasurement] = tco3_zm_year_pd[refMeasurement]

# plot smoothed data
tco3_zm_smooth_pd.plot(legend=False)
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49e8574990>
```



All models are quite apart. We can choose a reference value, i.e. TCO3 value for the reference year (see "refYear" in the "Global Variables") and the reference measurement ("refMeasurement") and shift all models to that value.

Find TCO3 value for the reference measurement (refMeasurement) and reference year (refYear)

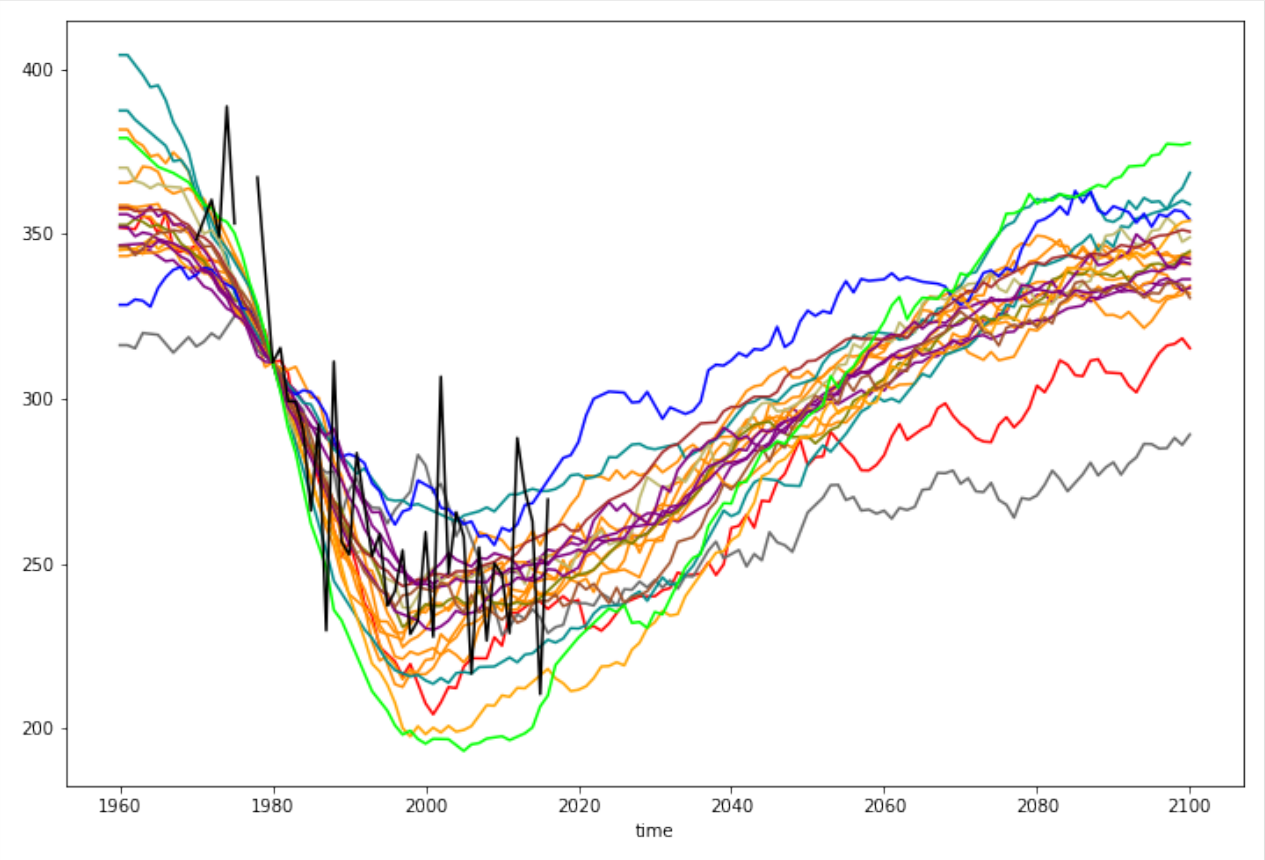
[14]:
```
refValue = tco3_zm_year_pd[refMeasurement][tco3_zm_year_pd.index == refYear].
↪interpolate(method='linear').values[0]
print(refValue)
```

```
310.925
```

Shift models to the reference value (refValue)

[15]:
```
# Calculate shift values for every model (refValue - modelValue for refYear)
tco3_zm_shift = refValue - tco3_zm_smooth_pd[tco3_zm_smooth_pd.index == refYear]

# Shift models by the shift values
tco3_zm_refYear = tco3_zm_smooth_pd + tco3_zm_shift.values

tco3_zm_refYear.plot(legend=False)
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f49e7dfc990>
```

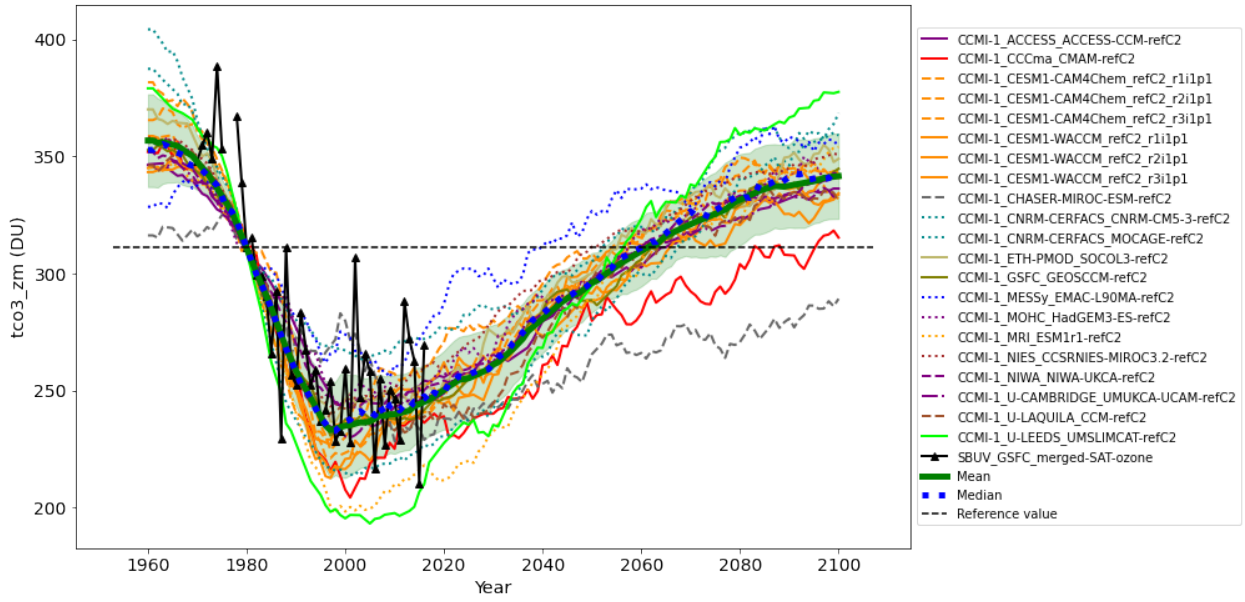


[16]:
```
# Create an empty DataFrame for statistics (also to have better control how to plot it␣
↪later)
tco3_zm_refYear_stat = pd.DataFrame(index=tco3_zm_refYear.index, columns=['Mean', 'Std',
↪'Median'])
# Fill 'Mean', 'Std' values, skip refMeasurement from the 'Mean' and 'Std' calculations
tco3_zm_refYear_stat["Mean"] = tco3_zm_refYear.drop(refMeasurement, axis=1,␣
↪inplace=False).mean(axis=1, skipna=True)
tco3_zm_refYear_stat["Std"] = tco3_zm_refYear.drop(refMeasurement, axis=1,␣
↪inplace=False).std(axis=1, skipna=True)
tco3_zm_refYear_stat["Mean-Std"] = tco3_zm_refYear_stat["Mean"] - tco3_zm_refYear_stat[
↪"Std"]
tco3_zm_refYear_stat["Mean+Std"] = tco3_zm_refYear_stat["Mean"] + tco3_zm_refYear_stat[
↪"Std"]
tco3_zm_refYear_stat["Median"] = tco3_zm_refYear.median(axis=1, skipna=True)
```

Finally, plot tco3_zm analysed data

[17]:
```
# one can use pandas.plot() but if we want to use default styles delivered by the API,␣
↪we better plot line-by-line
# for the tco3_zm plot we do not use "marker" except for the reference measurement

for col in tco3_zm_refYear.columns:
  style = tco3_skim_pd[tco3_skim_pd['model']==col]
  col_marker = '' if col != refMeasurement else style['plotstyle.marker'].values[0]
  tco3_zm_refYear[col].plot(color=style['plotstyle.color'].values[0],
                            linestyle=style['plotstyle.linestyle'].values[0],
                            marker=col_marker,
                            linewidth=style['plotstyle.linewidth'].values[0])

# draw horizonthal line for the reference value (refValue)
xmin, xmax = plt.xlim()
plt.hlines(refValue, xmin, xmax,
           colors='k', # 'dimgray'..?
           linestyles='dashed',
           label='Reference value',
           zorder=256)

# plot Mean, Median, and fill the area between Mean +/- Std
tco3_zm_refYear_stat['Mean'].plot(linewidth=5, color='g')
plt.fill_between(tco3_zm_refYear_stat.index,
                 tco3_zm_refYear_stat['Mean+Std'],
                 tco3_zm_refYear_stat['Mean-Std'],
                 color='g', alpha=0.2);
tco3_zm_refYear_stat['Median'].plot(linewidth=5, linestyle='dotted', color='b') # dashed

# tune the plot
ax = plt.gca() # get axis instance
handles, labels = ax.get_legend_handles_labels()
plt.legend(handles=handles,
           loc='center left',
           bbox_to_anchor=(1.0, 0.5))

plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

ax.set_xlabel('Year', fontsize='x-large')
ax.set_ylabel('tco3_zm (DU)', fontsize='x-large')
```

[17]:
```
Text(0, 0.5, 'tco3_zm (DU)')
```



Further analyse the data and plot tco3_return

Define a help function to find return year for every model

[18]:
```
def get_tco3_return(data):
  """Function to find return_years for the set of models

  :param data: input climate data after smoothing and shifting to the reference point
  :return tco3_return_year_pd: DataFrame of models with return years
  """

  refMargin = 5 # 'margin' years after refYear to avoid return years immediately after refYear
  # select data later than (refYear+refMargin) year and with values above refValue. Drop duplicates to remove raws with all False or True
  tco3_return = (data[data.index>(refYear+refMargin)]>refValue).drop_duplicates()
  # every model (column) with at least one return_year (True) add to the dictionary {'model': return_year}
  tco3_return_year = { col: [tco3_return[col][tco3_return[col]==True].index[0]] for col in tco3_return.columns if len(tco3_return[col][tco3_return[col]==True].index)>0 }
  # convert tco3_return_year dictionary to the pandas DataFrame
  tco3_return_year_pd = pd.DataFrame.from_dict(tco3_return_year)
  tco3_return_year_pd.index = ['user_region']

  return tco3_return_year_pd
```

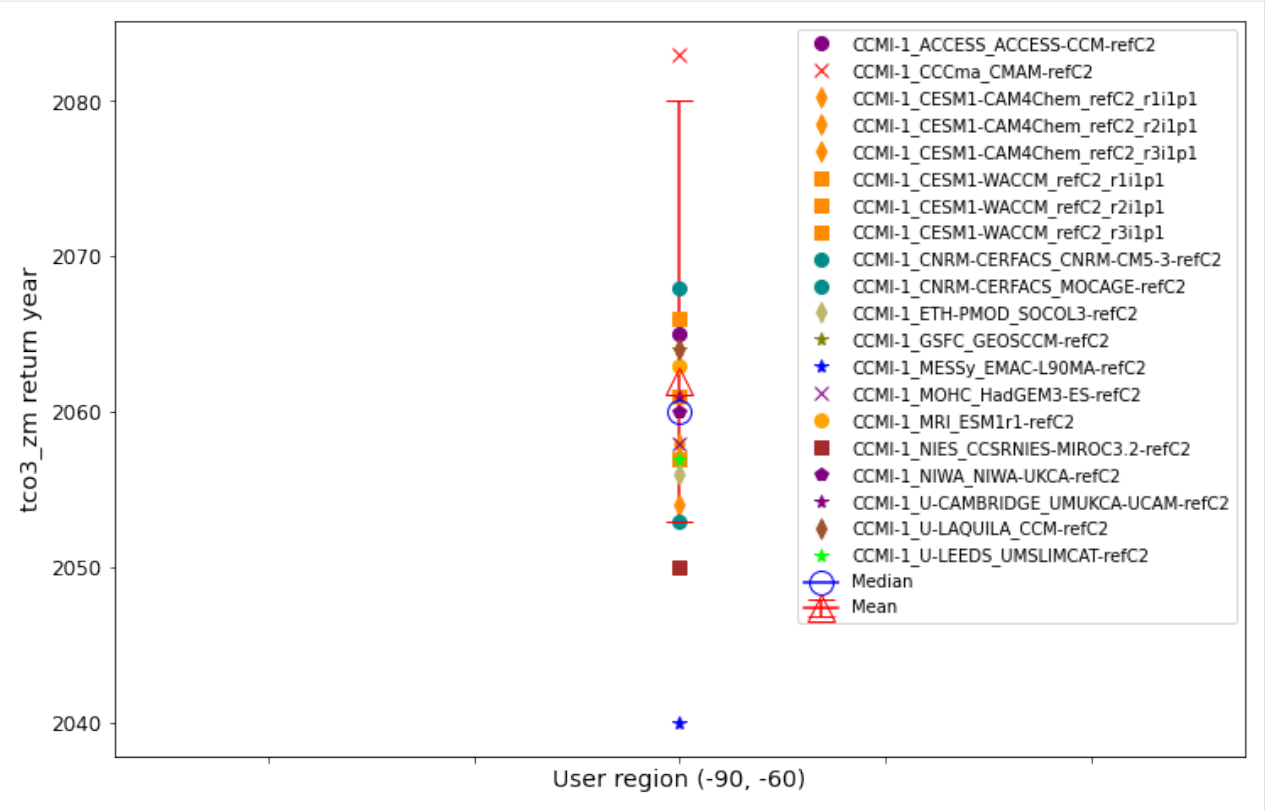
Find return years for specified models

[19]:
```
# find return years for the models, exclude the Reference Measurement (refMeasurement)
tco3_return_models = get_tco3_return(tco3_zm_refYear.drop(refMeasurement, axis=1))
if debug: print(tco3_return_models)

# find return years for statistical data points ('Mean', 'Mean-Std', 'Mean+Std')
tco3_return_stat = get_tco3_return(tco3_zm_refYear_stat)
if debug: print(tco3_return_stat)
```

Finally, plot the return years, tco3_return

[20]:
```
xlabel = F"User region ({kwargs_tco3_zm['lat_min']}, {kwargs_tco3_zm['lat_max']})"
tco3_return_models.index = [xlabel]

# to use default styles delivered by the API, we plot model-by-model
for col in tco3_return_models.columns:
  style = tco3_skim_pd[tco3_skim_pd['model']==col]
  tco3_return_models[col].plot(color=style['plotstyle.color'].values[0],
                               linestyle='none',
                               marker=style['plotstyle.marker'].values[0],
                               markersize=8)

mean_yerr = [ tco3_return_stat['Mean'] - tco3_return_stat['Mean+Std'],
              tco3_return_stat['Mean-Std'] - tco3_return_stat['Mean']]
tco3_return_stat.index = [xlabel]
tco3_return_stat['Mean'].plot(color='r', marker='^', markersize=16, yerr=mean_yerr,␣
↪capsize=8, legend='Mean', mfc='None')
tco3_return_stat['Median'].plot(color='b', marker='o', markersize=14, legend='Median',␣
↪mfc='None')

# tune the plot
# show the legend
ax = plt.gca() # get axis instance
ax.legend(bbox_to_anchor=(1.0, 1.0))

plt.xticks(fontsize=14)
plt.yticks(fontsize=12)

ax.set_ylabel('tco3_zm return year', fontsize='x-large')
```

[20]:
```
Text(0, 0.5, 'tco3_zm return year')
```



The example shows how to use a Jupyter notebook to retrieve skimmed data in the JSON format and analyse for *tco3_zm* and *tco3_return* plots. The corresponding Jupyter notebook can be downloaded here .

Tip: In order to run a Jupyter notebook one may use either Jupyter Hub of EOSC-Synergy or Colab free service from Google.

8.2 Public presentations, demos

- O3as YouTube channel

https://youtu.be/oBeQHge1iP4

CHAPTER

NINE

INTRODUCTION

O3as REST API provides:

- Access to O3as (Ozone assessment) skimmed data (produced by the `o3skim` component).
- Information about used Climate Models
- Ability to produce ozone plots (e.g. tco3_zm, tco3_return) in either PDF or JSON format

The API leverages Swagger, Flask, and Connexion.

The source code can be found in our o3api GitLab repository.

CHAPTER

TEN

O3API ENDPOINTS

10.1 Swagger API documentation

Tip: The full and actual Swagger documentation is found at https://api.o3as.fedcloud.eu/api/v1/ui/.

10.2 Endpoints description

The documentation below is automatically generated from the `swagger.yml`:

`GET /apiinfo`

Returns information about the API

Information about the API (o3api metadata)

Status Codes

- 200 OK – Successfully returned o3api info
- 404 Not Found – Requested resource not found

`GET /data`

Returns a list of plot types with the available raw data

List of plot types with the available raw data

Status Codes

- 200 OK – Successfully returned the list of data types
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`POST /data/tco3_zm`

Returns raw data for tco3_zm

Raw data to be processed for building tco3_zm plot

Query Parameters

- **`begin`** (*integer*) – Year to start data scanning from
- **`end`** (*integer*) – Year to finish data scanning
- **`month`** (*array*) – Month(s) to select, if not a whole year

- **lat_min** (*integer*) – Latitude (min) to define the range (-90..90)
- **lat_max** (*integer*) – Latitude (max) to define the range (-90..90)

Status Codes

- 200 OK – Successfully retrieved the data
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`POST /data/tco3_return`

Returns raw data for tco3_return

Raw data to be processed for building tco3_return plot

Query Parameters

- **begin** (*integer*) – Year to start data scanning from
- **end** (*integer*) – Year to finish data scanning
- **month** (*array*) – Month(s) to select, if not a whole year
- **lat_min** (*integer*) – Latitude (min) to define the range (-90..90)
- **lat_max** (*integer*) – Latitude (max) to define the range (-90..90)

Status Codes

- 200 OK – Successfully retrieved the data
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`GET /models`

Returns a list of available models

List of available models

Query Parameters

- **ptype** (*string*) – Plot type (tco3_return, tco3_zm, vmro3_zm)
- **select** (*string*) – Select models according to the {select} pattern

Status Codes

- 200 OK – Successfully returned list of models
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`GET /models/{model}`

Returns detailed information about a model

Detailed information about a model

Parameters

- **model** (*string*) – model name

Status Codes

- 200 OK – Successfully returned model information
- 404 Not Found – Requested resource not found

- **default** – Unexpected error

`POST /models/plotstyle`
Returns plot styles for models

Plot styles for selected models and plot type

Query Parameters

- **ptype** (*string*) – Plot type (tco3_return, tco3_zm, vmro3_zm)

Status Codes

- 200 OK – Successfully returned plot styles for models
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`GET /plots`
Returns a list of possible plots

List of possible plots

Status Codes

- 200 OK – Successfully returned the list of possible plots
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`POST /plots/tco3_zm`
Builds and returns tco3_zm plot

tco3_zm plot or corresponding data points

Query Parameters

- **begin** (*integer*) – Year to start data scanning from
- **end** (*integer*) – Year to finish data scanning
- **month** (*array*) – Month(s) to select, if not a whole year
- **lat_min** (*integer*) – Latitude (min) to define the range (-90..90)
- **lat_max** (*integer*) – Latitude (max) to define the range (-90..90)
- **ref_meas** (*string*) – Reference observational measurument (Required)
- **ref_year** (*integer*) – Reference year for the obsevational measurement (Required)

Status Codes

- 200 OK – Successfully created the plot
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

`POST /plots/tco3_return`
Builds and returns tco3_return plot

tco3_return plot or corresponding data points

Query Parameters

- **month** (*array*) – Month(s) to select, if not a whole year

- **lat_min** (*integer*) – Latitude (min) to define the range (-90..90)
- **lat_max** (*integer*) – Latitude (max) to define the range (-90..90)
- **ref_meas** (*string*) – Reference observational measurument (Required)
- **ref_year** (*integer*) – Reference year for the obsevational measurement (Required)

Status Codes

- 200 OK – Successfully created a plot
- 404 Not Found – Requested resource not found
- **default** – Unexpected error

CHAPTER

ELEVEN

O3API REFERENCE

If you are looking for information on a specific function, class or method, this part of the documentation is for you.

11.1 api

O3as REST API methods:

o3api.api.**get_api_info**()
Return information about the package

Returns The o3api package info

Return type dict

o3api.api.**get_data_tco3_return**(**args*, ***kwargs*)
Retrieve data to produce tco3_return plot

Parameters **kwargs** – provided in the API call parameters

Returns JSON document with data points

o3api.api.**get_data_tco3_zm**(**args*, ***kwargs*)
Retrieve data to produce tco3_zm plot

Parameters **kwargs** – provided in the API call parameters

Returns JSON document with data points

o3api.api.**get_data_types**()
Get list of plot types with available data

o3api.api.**get_data_vmro3_zm**(**args*, ***kwargs*)
Retrieve data to produce vmro3_zm plot

Parameters **kwargs** – provided in the API call parameters

Returns JSON document with data points

o3api.api.**get_model_detail**(**args*, ***kwargs*)
Return information about the Ozone model

Returns Info about the Ozone model

Return type dict

o3api.api.**get_models_info**()
Return dictionary of available models with the meta info

Returns The dictionary of available models

Return type dict

o3api.api.**get_models_list**(**args*, ***kwargs*)
Return the list of available Ozone models

Returns The list of available models

Return type list

o3api.api.**get_plot_style**(**args*, ***kwargs*)
Returning plot style for selected models and plot type

o3api.api.**get_plot_types**()
Get list of the provided plot methods

o3api.api.**plot**(*data*, *ckwargs*, ***kwargs*)
Main plotting routine

Parameters

- **data** – data to plot
- **ckwargs** – dictionary for curve plotting (e.g. color, style)
- **kwargs** – provided in the API call parameters

Returns PDF plot

o3api.api.**plot_json**(*data*, *ckwargs*, ***kwargs*)
Plotting routine returning JSON points

Parameters

- **data** – data ready for plotting
- **ckwargs** – dictionary for curve plotting (e.g. color, style)
- **kwargs** – provided in the API call parameters

Returns JSON document with data points and styles for plotting

o3api.api.**plot_tco3_return**(**args*, ***kwargs*)
Plot tco3_return

Parameters **kwargs** – provided in the API call parameters

Returns Either PDF plot or JSON document

o3api.api.**plot_tco3_zm**(**args*, ***kwargs*)
Plot tco3_zm

Parameters **kwargs** – provided in the API call parameters

Returns Either PDF plot or JSON document

o3api.api.**plot_vmro3_zm**(**args*, ***kwargs*)
Plot vmro3_zm

Parameters **kwargs** – provided in the API call parameters

Returns Either PDF plot or JSON document

11.2 plots

O3as helper classes to extact data for plotting:

class o3api.plots.**DataSelection**(*plot_type*, ***kwargs*)

Class to perform data selection, based on *Dataset*.

Parameters

- **begin** – Year to start data scanning from
- **end** – Year to finish data scanning
- **month** – Month(s) to select, if not a whole year
- **lat_min** – Minimum latitude to define the range (-90..90)
- **lat_max** – Maximum latitude to define the range (-90..90)

get_dataslice(*model*)

Function to select the slice of data according to the time and latitude requested

Parameters **model** – The model to process

Returns xarray dataset selected according to the time and latitude

Return type xarray

to_pd_dataframe(*ds*, *model*)

Convert xarray variable to pandas dataframe (faster method?)

Parameters

- **ds** – xarray dataset
- **model** – The model to process for self.plot_type

:return dataset as pandas dataframe :rtype: pandas dataframe

to_pd_series(*ds*, *model*)

Convert xarray variable to pandas series

Parameters

- **ds** – xarray dataset
- **model** – The model to process for self.plot_type

:return dataset as pandas series :rtype: pandas series

class o3api.plots.**Dataset**(*plot_type*, ***kwargs*)

Base Class to initialize the dataset

Parameters **plot_type** – The plot type (e.g. tco3_zm, vmro3_zm, ...)

get_dataset(*model*)

Load data from one datafile

Parameters **model** – The model to process

Returns xarray dataset

Return type xarray.Dataset

get_mfdataset(*model*)

Load data from the datafile list

Parameters **model** – The model to process

Returns xarray dataset

Return type xarray.Dataset

class o3api.plots.**ProcessForTCO3**(***kwargs*)
Subclass of *DataSelection* to calculate tco3_zm

get_ensemble_for_plot(*models*)
Build the ensemble of tco3_zm models for plotting, include reference

Parameters **models** – Models to process for tco3_zm

Returns ensemble of models, including the reference, as pd.DataFrame

Return type pd.DataFrame

get_ensemble_shifted(*data*)
Shift tco3_zm data to reference year

Parameters **data** – data to process as pd.DataFrame

Returns shifted data points for plotting

Return type pd.DataFrame

get_ensemble_smoothed(*models*, *smooth_win*)
Smooth tco3_zm data using boxcar

Parameters **models** – Models to process for tco3_return

Returns smoothed data points

Return type pd.DataFrame

get_ensemble_stats(*data*)
Calculate Mean, Std, Median for tco3_zm data

Parameters **data** – data to process as pd.DataFrame

Returns updated pd.DateFrame with stats columns

Return type pd.DataFrame

get_ensemble_yearly(*models*)
Rebin tco3_zm data for yearly entries

Parameters **models** – Models to process for tco3_return

Returns yearly data points

Return type pd.DataFrame

get_raw_data(*model*)
Process the model to get tco3_zm raw data

Parameters **model** – The model to process for tco3_zm

Returns raw data points in preparation for plotting

Return type pandas series (pd.Series)

get_raw_data_pd(*model*)
Process the model to get tco3_zm raw data

Parameters **model** – The model to process for tco3_zm

Returns raw data points in preparation for plotting

Return type pd.DataFrame

get_raw_ensemble_pd(*models*)
Build the ensemble of tco3_zm models

Parameters **models** – Models to process for tco3_zm

Returns ensemble of models as pd.DataFrame

Return type pd.DataFrame

get_ref_value()
Get reference value for the reference year

Returns reference value (tco3_zm at reference year)

class o3api.plots.**ProcessForTCO3Return**(***kwargs*)
Subclass of *ProcessForTCO3* to calculate tco3_return

get_ensemble_for_plot(*models*)
Build the ensemble of tco3_return points for plotting

Parameters **models** – Models to process for tco3_zm

Returns ensemble of models, including the mean, as pd.DataFrame

Return type pd.DataFrame

get_return_years(*data*)
Calculate return year for every model

Parameters **data** – data to process

Returns return years for models

Return type pd.DataFrame

class o3api.plots.**ProcessForVMRO3**(***kwargs*)
Subclass of *DataSelection* to calculate vmro3_zm

get_plot_data(*model*)
Process the model to get vmro3_zm data for plotting

Parameters **model** – The model to process for vmro3_zm

Returns xarray dataset for plotting

Return type xarray

11.3 plothelpers

O3as help functions to create figures:

o3api.plothelpers.**cleanse_models**(***kwargs*)
Cleansing models from empty entries, spaces, and quotes

Parameters **kwargs** – The provided in the API call parameters

Returns models cleansed from empty entries, spaces, quotes

Return type list

o3api.plothelpers.**get_date_range**(*ds*)
Return the range of dates in the provided dataset

Parameters **ds** – xarray dataset to check

Returns date_min, date_max

o3api.plothelpers.**get_periodicity**(*pd_time*)
Calculate periodicity in the provided data

Parameters **pd_time** (*pandas DatetimeIndex*) – The time period

Returns Calculated periodicity as the number of points per year

Return type int

o3api.plothelpers.**get_plot_info_html**(***kwargs*)
Generate info text (HTML) for the plot

Parameters **kwargs** – The provided in the API call parameters

Returns information text (HTML) with legal info, parameters etc

Return type string

o3api.plothelpers.**get_plot_info_txt**(***kwargs*)
Generate info text for the plot

Parameters **kwargs** – The provided in the API call parameters

Returns information text with legal info, parameters etc

Return type string

o3api.plothelpers.**get_plot_params**(***kwargs*)
Get plot parameters

Parameters **kwargs** – The provided in the API call parameters

Returns plot_params with added input parameters

Return type string

o3api.plothelpers.**set_figure_attr**(*fig*, ***kwargs*)
Configure the figure attributes

Parameters

- **fig** – Figure instance
- **kwargs** – The provided in the API call parameters

Returns none

o3api.plothelpers.**set_filename**(***kwargs*)
Set file name

Parameters **kwargs** – The provided in the API call parameters

Returns file_name with added input parameters (no extension given!)

Return type string

Data skimming for ozone assessment.

o3skim is an open source project and Python package that provides the tools to pre-process, standardize and reduce ozone data from netCDF models to simplify and speed up ozone data transfer and plot.

CHAPTER

TWELVE

GETTING STARTED

- Which *First steps* you need to start.
- How to use *o3norm* to standardize your models.
- How to use *o3skim* to reduce your models.

12.1 First steps

12.1.1 Prerequisites

This software is shipped as **python3** package, therefore you need to have python3 and pip installed. If not, please check pip documentation to find out how to install and run **pip** in your system with at least the following versions:

software	version
python	>= 3.6.12
pip	>= 21.0.1

Note: Non admin rights? Check how to run conda in your machine.

12.1.2 Installation

Once **python3** and **pip** are running in your system, download the package and install it using pip:

```
$ git clone https://git.scc.kit.edu/synergy.o3as/o3skim.git
Cloning into 'o3skim'...
...
$ cd o3skim
$ pip install .
...
Successfully installed o3skim-0.4.0
```

12.2 o3norm

You can standardize a dataset from any sources using the provided command **o3norm** when installing the package.

```
usage: o3norm [-h] [-v {DEBUG,INFO,WARNING,ERROR,CRITICAL}] [-t TARGET]
              (--tco3_zm | --vmro3_zm)
              {CCMI-1,ECMWF,ESACCI,SBUV} ...
```

This command loads the model from the specified sources and produces an standardized netCDF output with the following structure:

```
Dimensions:   (lat: _, lon: _, plev: _, time: _)
Coordinates:
  * time      (time) datetime64[ns] ____-__-__ ... ____-__-__T__:__:__
  * plev      (plev) float64 ___._ ... ___._
  * lon       (lon) float64 ___._ ... ___._
  * lat       (lat) float64 ___._ ... ___._
Data variables:
    tco3_zm   (time, lon, lat) float64 __
    vmro3_zm  (time, plev, lon, lat) float64 __
    ...
Attributes:
    <The original dataset attributes>
```

This can help you to work easier with multiple sources having a common data structure for your data.

The usage is very simple, call the **o3norm** command followed by the specific model type you would like to load as a *Sub-command*. Note that there are general optional arguments common to all source types (for example –target) and specific to each source type (for example –delimiter in the case of SBUV).

```
positional arguments (Sub-commands):
  {CCMI-1,ECMWF,ESACCI,SBUV}
                        Sub-commands
    CCMI-1              CCMI-1 Source input
    ECMWF               ECMWF Source input
    ESACCI              ESACCI Source input
    SBUV                SBUV Source input

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --verbosity {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Sets the logging level (default: INFO)
  -t TARGET, --target TARGET
                        Target netCDF file (default: o3data)
  --tco3_zm             Standardization for total column ozone
  --vmro3_zm            Standardization for volume mixing ratio ozone
```

You can use the *optional argument* **–help** to see the *Command* and *Sub-command* instructions. For example, to see the options available for the *CCMI-1* you can use:

```
$ o3norm CCMI-1 --help
usage: o3norm CCMI-1 [-h] [--time TIME] [--plev PLEV] [--lat LAT] [--lon LON]
                     variable paths [paths ...]
...
```

As last example, the following commands shows how to produce an output of standardized netCDF files at the file *mydata.nc* using the files provided from a CCMI-1 source:

```
$ o3norm --tco3_zm -t mydata.nc CCMI-1 toz Ccmi/some_path/*.nc
$ o3norm --vmro3_zm -t mydata.nc CCMI-1 vmro3 Ccmi/some_path/*.nc
...
```

See the first command loads the **toz** from the source as **tco3_zm** and the second the **vmro3** as **vmro3_zm**. Both commands are targeting the file *mydata.nc*, therefore that file will contain the information about the 2 variables.

12.3 o3skim

You can reduce an output dataset produced by **o3norm** using the provided command **o3skim** when installing the package.

```
usage: o3skim [-h] [-v {DEBUG,INFO,WARNING,ERROR,CRITICAL}] [-o OUTPUT]
              [--lon_mean] [--lat_mean] [--year_mean]
              paths [paths ...]
```

This command loads the model from the specified paths and produces a set of standardized netCDF files. The output data are splitted into one file per variable at the original dataset. For example, skimming a dataset with **tco3_zm** and **vmro3_zm** produces the following output at the specified folder:

```
$ tree output_folder
output_folder/
├── tco3_zm.nc
└── vmro3_zm.nc
```

The structure of each output file will depend on the variable it contains (for example, tco3_zm does not contain *plev* coordinates) and the skimming operations performed (for example, lon_mean reduces the *lon* coordinate). This reduces the transference size by transferring only the files with the intended variables.

The usage is quite simple, call the **o3skim** command followed by the operations you would like to performed as *optional arguments* and with the paths to the dataset you would like to reduce.

```
positional arguments:
  paths                 Paths to netCDF files with the data to skim

optional arguments:
  -h, --help            show this help message and exit
  -v {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --verbosity {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Sets the logging level (default: INFO)
  -o OUTPUT, --output OUTPUT
                        Folder for output files (default: .)

operations:
  --lon_mean            Longitudinal mean across the dataset
  --lat_mean            Latitudinal mean across the dataset
  --year_mean           Time average across the year
```

As an example, the following command shows how to reduce a dataset running an average mean over the latitude and longitude coordinates at the folder *skimmed_model*:

```
$ o3skim -o skimmed_model --lon_mean --lat_mean mydata.nc
...
$ tree skimmed_model
skimmed_model/
└── tco3_zm.nc
```

From this example, we can foreseen that the original dataset *mydata.nc* contained only **tco3_zm** information. The expected structure for the *tco3_zm.nc* dataset would be as follows:

```
Dimensions:   (time: _)
Coordinates:
  * time      (time) datetime64[ns] ____-__-__ ... ____-__-__T__:__:__
Data variables:
    tco3_zm   (time) float64 __
Attributes:
    <The original dataset attributes>
```

CHAPTER

THIRTEEN

DEVELOPER GUIDE

- Check the *First steps* before start.
- Check the *Package index* to learn about the functions.
- Check the *Test guidelines* to extend the functionality.

13.1 Package index

O3as package with utilities to handle ozone data skimming.

`o3skim.`**`process`**(*dataset*, *actions*)

Function in charge of processing the list of o3skim operations to the ozone variables dataset. The available list of operations to perform are:

lon_mean Longitudinal mean accross the dataset.

lat_mean Latitudinal mean accross the dataset.

Note that multiple operations can be concatenated. For example using the list ['lon_mean', 'lat_mean'] as actions parameter input would perform a longitudinal mean followed afterwards by a latitudinal mean before returning the result.

Parameters

- **`dataset`** (`xarray.Dataset`) – Original o3 dataset where to perform operations.
- **`actions`** (*`list`*) – List of operation names to perform.

Returns Dataset after processing listed operations.

Return type `xarray.Dataset`

`o3skim.`**`save`**(*dataset*, *target*, *split_by=None*)

Function in charge of saving the input dataset into the file system using an specified time range. The available type of output file formats are:

None Output file format is {target}.nc

year Output file format is {target}_{y}-{y+01}.nc

decade Output file format is {target}_{y}-{y+10}.nc

Parameters

- **`dataset`** (`xarray.DataSet`) – DataSet to save in the target.
- **`target`** (*`str`*) – Location where to save followed by the name prefix.

- **split_by** (*str, optional*) – Type of saving format to apply.

13.1.1 o3skim loads

Module in charge of model data loading.

o3skim.loads.**ccmi**(*variable*, *paths*)
Loads and returns a CCMI-1 DataArray model and the dataset attributes.

Parameters

- **variable** (*str*) – Variable to load from the dataset.
- **paths** (*str or [str]*) – Paths expression to the dataset netCDF files.

Returns Standardized DataArray.

Return type (xarray.DataArray, dict)

o3skim.loads.**ecmwf**(*variable*, *paths*)
Loads and returns a ECMWF DataArray model and the dataset attributes.

Parameters

- **variable** (*str*) – Variable to load from the dataset.
- **paths** (*str or [str]*) – Paths expression to the dataset netCDF files.

Returns Standardized DataArray.

Return type (xarray.DataArray, dict)

o3skim.loads.**esacci**(*variable*, *time_position*, *paths*)
Loads and returns a ESACCI DataArray model and the dataset attributes. Note the name structure is composed by sections: For example: ESACCI-OZONE-L3S-TC-MERGED-DLR_1M-20010302-fv0100. Therefore is needed to indicate the position in the string for the dataset time (7 or -2 for the case above).

Parameters

- **variable** (*str*) – Variable to load from the dataset.
- **time_position** (*int*) – Name position for the dataset time.
- **paths** (*str or [str]*) – Paths expression to the dataset netCDF files.

Returns Standardized DataArray.

Return type (xarray.DataArray, dict)

o3skim.loads.**sbuv**(*textfile*, *delimiter*)
Loads and returns a SBUV DataArray model and the dataset attributes. Note SBUV models do not have longitude coordinate.

Parameters

- **textfile** (*str*) – Location to the textfile with model information.
- **delimiter** (*str or [str]*) – Delimiter character for row values on the table.

Returns Standardized DataArray.

Return type (xarray.DataArray, {})

13.1.2 o3skim normalization

Module in charge of dataset normalization when loading models.

o3skim.normalization.**run**(*datarray*, *variable*, ***coords*)
Standardizes a DataArray.

Returns Standardized DataArray.

Return type xarray.DataArray

13.1.3 o3skim operations

Module in charge of implementing the o3skim operations.

o3skim.operations.**run**(*name*, *dataset*)
Main entry point for operation call on o3skimming functions:

lon_mean Longitudinal mean across the dataset.

lat_mean Latitudinal mean across the dataset.

year_mean Time coordinate averaged by year.

Parameters

- **name** (*str*) – Operation name to perform.
- **dataset** (xarray.Dataset) – Original o3 dataset where to perform operations.

Returns Dataset after processing the specified operation.

Return type xarray.Dataset

13.2 Test guidelines

Testing is based on sqa-baseline criteria, tox automation is used to simplify the testing process.

To install it with pip use:

```
$ pip install tox
```

To run unit and functional tests together with reports use:

```
$ tox
...
clean: commands succeeded
stylecheck: commands succeeded
bandit: commands succeeded
docs: commands succeeded
py36: commands succeeded
report: commands succeeded
...
```

The last coverage report output is produced at **htmlcov** which can be displayed in html format accessing to **index.html**.

The last Pep8 report produced by flake8 at the output file **flake8.log**.

However, you can also run different test configurations using different tox environments:

13.2.1 Code Style [QC.Sty]

Test pep8 maintenance style conventions based on pylint format. To run stylecheck use:

```
$ tox -e stylecheck
...
stylecheck: commands succeeded
...
```

13.2.2 Unit Testing [QC.Uni]

All unit tests are placed inside the package (./o3skim/test). This helps to test easily functions at low level and ensure the functions have the expected behavior. To run unit tests use:

```
$ tox -e unittesting
...
unittesting: commands succeeded
...
```

This environment also provide a coverage term report for the tests. The design of Unit Tests is based on the python unittest framework, a simple and extended test framework which ships by default together with python.

The usage is very simple and straight forward for simple tests, but the difficulty to parametrize and combine multiple test fixtures makes it not suitable for Black-Box testing without a very complex customization.

13.2.3 Functional Testing [QC.Fun]

Located inside tests package folder (./tests). Functional testing is used to test the system from a general overview of the application. To run functional tests use:

```
$ tox -e functional
...
functional: commands succeeded
...
```

This environment also provide a coverage term report for the tests. The framework used is pytest to provide a simple syntax to test all possible combinations from the user point of view.

Pytest detects directly all tests following the test_discovery naming conventions. Therefore all functional tests should be located on the **tests** folder at the package root and start with **test**. For example *test_sources.py*.

More than 500 test combinations are generated using which otherwise might not be feasible using other python test frameworks.

13.2.4 Security [QC.Sec]

Security checks are performed by bandit, a tool designed to find common security issues in Python code. To run security checks use:

```
$ tox -e functional
...
functional: commands succeeded
...
```

13.2.5 Documentation [QC.Doc]

Documentation is build using sphinx, a tool designed to create documentation based on code. To run documentation build checks use:

```
$ tox -e docs
...
docs: commands succeeded
...
```

The HTML pages are build inside in docs/_build.

CHAPTER

FOURTEEN

AUTHORS

- @T.Kerzenmacher - Data scientist at KIT-IMK
- @V.Kozlov - Code developer at KIT-SCC
- @B.Esteban - Code developer at KIT-SCC

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

O

HTTP ROUTING TABLE

/apiinfo

/data

/models

/plots

INDEX

P

R

S

T