CMIP6 Ozone forcing dataset: supporting information

R. Checa-Garcia,
Department of Meteorology, University of Reading, Reading, United Kingdom
email: r.checa-garcia@reading.ac.uk

Abstract

This document describes additional aspects of the publication: *Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database* to fulfil the recommendations of AGU's Data Policy as described in the Statement of Commitment of the Coalition on Publishing Data in the Earth and Space Sciences (COPDESS) and also in the Transparency and Openness Promotion Guidelines (TOP). Therefore, from the original *CMIP6 Ozone forcing dataset documentation*, it has been extracted specific validations in terms of total ozone column, partial ozone column, as well as zonal mean time series of ozone concentrations over several latitudinal bands. A second part of the document summarizes the code developed to create specific products like total column ozone or partial column ozone but also several steps of data analysis based on the estimation of time-series of concentrations, anomalies and seasonal cycles. Finally, a chapter includes an study of ozone radiative forcing based on CMAM model ozone concentrations dataset.

Important: This manuscript is not the reference document of CMIP6 ozone dataset. It has been uploaded to provide supporting information (and pre-print material) to other journal publications. In order to cite properly the CMIP6 ozone concentrations dataset and their the contributors, please contact with Michaela I. Hegglin: m.i.hegglin@reading.ac.uk.

Part of contents of this report might/will included on the official paper to be published on Geoscientific Model Development journal (https://www.geoscientific-model-development.net/index.html).

- **Version Initial v1**: Supporting information for *Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database*.
- **Version Updated v2**: It has been added an study of *ozone radiative forcing based on CMAM model ozone concentrations dataset*.

Contents

Part I.
Supporting Information

1 CMIP6 O3 database: comparison of total ozone column with observational reanalysis

1.1 Method to estimate the Total and Partial Ozone Column

The column of ozone between a level named l_0 and other upper level named l_n in a discrete vertical grid is estimated by the equation,

$$Column(DU) = 10\frac{RT_0}{g_0 P_0}\sum_{i=l_0}^{l_{n-1}}\frac{\chi_{O3}(i)+\chi_{O3}(i+1)}{2}(p(i)-p(i+1)) \tag{1.1}$$

with

$$\begin{aligned} R &= 287.3JKg^{-1}K^{-1} \\ T_0 &= 273.15K \\ P_0 &= 1.01325\cdot 10^5 Pa \\ g_0 &= 9.80665ms^{-2} \\ \chi_{O3}(i) &= \text{vmr of O3 at level i} \\ p(i) &= \text{pressure at level i} \end{aligned}$$

The equation above implies that the estimation of the total and partial ozone column requires the surface pressure, for which it is used the CESM1-WACCM surface pressure in the present study. The method above introduced is equivalent to the one described[1] by Dr. Jerry R Ziemke from NASA Goddard Space Flight Center. Given, for example, the CMIP5 and CMIP6 datasets, it is possible to create two different products: the first is simply the total ozone column in DU (TOC), from the surface pressure until the TOA. The other calculates the partial column of ozone (POC) until the level ith, so it represents the column of ozone between surface and the level ith.

1.2 Python code to calculate the partial and total ozone column

```python
def calculate_partial_column(alats, alons, p_lev, ao3_mmr, ps_surf, times,
                             case='TOTAL', loopin=False):
    """
    This function is designed to calculate the partial column of ozone in DU.
    The idea is transform the o3_vmr field to an partial column so

    o3_vmr(lat, lon, lev) => o3_DU(lat, lon, lev) and it is the partial column
                                    until level lev.

    """

    g0 = 9.80665
    T0 = 273.15
    p0 = 101325.
    R = 287.3

    factor = 10.0*R*T0*0.5/(g0*p0)

    kg_to_g      = 1.0e+03
    ppmv_to_ppv  = 1.0e-06
    mw_o3        = 47.9982
    mw_dryair    = 28.9648

    # 0.01 to calculate in hPa, 1e6*mw_dryair/mw_o3 to change mmr to ppmv
    f_mmr_to_vmr = 0.01*1.e6*mw_dryair/mw_o3

    f_units = 0.01*1.e6
    n_lev   = len(p_lev)

    ao3_DU = np.zeros_like(ao3_mmr)
    acc_DU = np.zeros_like(ao3_mmr[:,0,:,:])
    delta_pss = np.zeros_like(times)
```

[1] The number 10 here is a factor to change units therefore it has units.

```python
    if loopin:
        #for itim in tqdm(range(len(times)-1), ncols=80, desc='1st DIM'):
        for ilev in tqdm(range(n_lev-1), ncols=80, desc='2nd DIM'):
            for ilat in range(len(alats)):
                lati = alats[ilat]
                for ilon in range(len(alons)):
                    delta_mmr = ao3_mmr[:,ilev,ilat,ilon]+ao3_mmr[:,ilev+1,ilat,ilon]
                    delta_pss[:] = p_lev[ilev+1]-p_lev[ilev]


                    for itim,tm in enumerate(times):
                        if p_lev[ilev+1] < ps_surf[itim, ilat, ilon]:
                            delta_pss[itim] = 0.0
                        if p_lev[ilev] > ps_surf[itim,ilat, ilon] > p_lev[ilev+1]:
                            delta_pss[itim] = ps_surf[itim, ilat, ilon]-p_lev[ilev+1]

                    acc_DU[:,ilat,ilon] = acc_DU[:,ilat,ilon]+f_units*delta_mmr*delta_pss[:]
            ao3_DU[:,ilev+1,:,:] = factor*copy.deepcopy(acc_DU[:,:,:])

    else:
     for itim in tqdm(range(len(times)), ncols=80, desc='1st DIM'):
        for ilev in tqdm(range(n_lev-1), ncols=80, desc='2nd DIM'):
            for ilat in range(len(alats)):
                lati = alats[ilat]
                for ilon in range(len(alons)):
                    delta_mmr = ao3_mmr[itim,ilev,ilat,ilon]+ao3_mmr[itim,ilev+1,ilat,ilon]
                    delta_prs = p_lev[ilev]-p_lev[ilev+1]

                    if ilev<10:
                        if p_lev[ilev+1] < ps_surf[itim, ilat, ilon]:
                           delta_prs = 0.0
                        if p_lev[ilev] > ps_surf[itim,ilat, ilon] > p_lev[ilev+1]:
                           delta_prs = ps_surf[itim, ilat, ilon]-p_lev[ilev+1]
                    if (f_units*delta_mmr*delta_prs <0):
                        print(f_units*delta_mmr*delta_prs, f_units, delta_mmr, delta_prs)

                    acc_DU[itim,ilat,ilon] = acc_DU[itim,ilat,ilon]+f_units*delta_mmr*delta_prs
            ao3_DU[itim,ilev+1,:,:] = factor*copy.deepcopy(acc_DU[itim,:,:])

    return ao3_DU
```

1.3 Satellite Observations Reanalysis MSR-1

There are several merged datasets of ozone total column observations based on satellite measurements. In general create a robust dataset by using several satellite platforms is challenging, due to the different sampling issue, temporal drift or resolutions between the different sensors. Therefore, it is usual to choose one instrument as reference and apply correction factors to the other instruments and perform comparisons over zonal means [1]. An improvement was designed by [4] which relies on the use of a model output as a transfer function between satellite platforms to move them to a common unbiased time series. Other improvements on the literature aims to correct possible satellite bias with collocated ozone-sondes that periodically are overpassed by satellites. Finally, those ozone retrievals can be assimilated on a coupled chemistry-climate model. Here, first we will compare qualitatively our product of total ozone column with one of the datasets that belongs to the last category with the goal of ascertain if broad typical properties of the TOC: temporal and spatial patterns are reproduced by our dataset.

Table 1. The satellite datasets used in this study. The columns show the name of the dataset, the satellite instrument on which it is based, the satellite, the period(s) used, the maximum distance allowed in an overpass, the number of ground instruments (WSI) and the total number of overpasses for this dataset.

Name	Instrument	Satellite	From	To	Dist.	#WSI	Overpasses
TOMS2a	TOMS	Nimbus-7	1 Nov 1978	6 May 1993	0.75°	137	182 464
TOMS2b	TOMS	Earth probe	25 Jul 1996	31 Dec 2002	0.75°	146	129 839
SBUV07	SBUV	Nimbus-7	31 Oct 1978	21 Jun 1990	2.00°	112	24 345
SBUV9a	SBUV/2	NOAA-9	2 Feb 1985	31 Dec 1989	2.00°	099	11 705
SBUV9d	SBUV/2	NOAA-9	1 Jan 1992	19 Feb 1998	2.00°	135	22 706
SBUV11	SBUV/2	NOAA-11	1 Dec 1988	31 Mar 1995	2.00°	166	38 874
			15 Jul 1997	27 Mar 2001			
SBUV16	SBUV/2	NOAA-16	3 Oct 2000	31 Dec 2003	2.00°	131	16 384
GDP	GOME-1	ERS-2	27 Jun 1995	31 Dec 2008	1.80°	156	108 758
TOGOMI	GOME-1	ERS-2	1 Apr 1996	31 Dec 2008	1.80°	155	107 276
SGP	SCIAMACHY	Envisat	2 Aug 2002	31 Dec 2008	0.90°	139	50 017
TOSOMI	SCIAMACHY	Envisat	2 Aug 2002	31 Dec 2008	0.90°	139	47 532
OMDOAO3	OMI	Aura	1 Oct 2004	31 Dec 2008	0.90°	123	84 089
OMTO3	OMI	Aura	17 Aug 2004	31 Dec 2008	0.90°	125	83 405
GOME2	GOME-2	Metop-A	4 Jan 2007	31 Dec 2008	0.45°	105	28 538

Figure 1.1.: Extracted from reference [8]. Table 1 of doi:10.5194/acp-10-11277-2010. The table shows the several satellite platforms merged on the dataset MSR-1 used on this section

For this task we have selected the MSR-1 ozone reanalysis dataset [8] from 1980 to 2008. It is divided on three decades 80s, 90s and 2000s, and then compared its seasonal properties over both pole regions and global maps with CMIP6 (see figures on the next section). The figure 1.3 shows the typical mean values of total ozone column for the last three decades over the Southern Hemisphere, for the full year but also for each season (Spring is MAM, Summer is JJA, Autumn is SON, Winter is DJF). Key properties to look up are the ozone hole for each decade on autumn (its magnitude and its location/extension) and also the general local maxima at the south region below Australia which changes over the season and its related with the annular modes.

The figure 1.4 is the equivalent of the previous one, but located on the northern hemisphere, in this case it is interesting the location of the two maxima over the China region on Spring and Winter seasons and its changes with each decade: higher values of TOC on 80s, less in 90s and a slight recover other the last decade.

Same properties but on a global map are shown in the figures 1.5 and 1.6, but they also allow us to observe the contrast between both hemispheres for each season the evolution of the TOC over the tropics. The figure 1.2 on this section compares MSR-1 zonal means for three decades with ERA20C, to understand better general properties of the dataset.

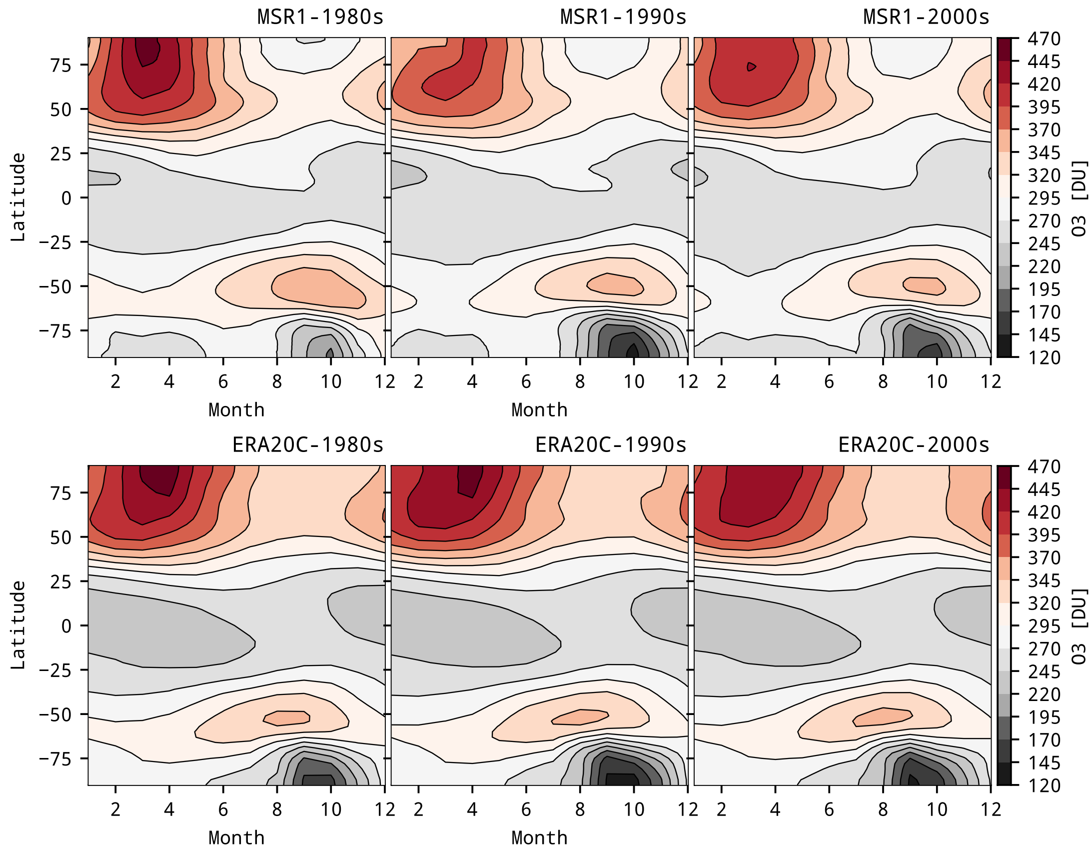

Figure 1.2.: Total Ozone Column for MSR-1 and ERA20C datasets.

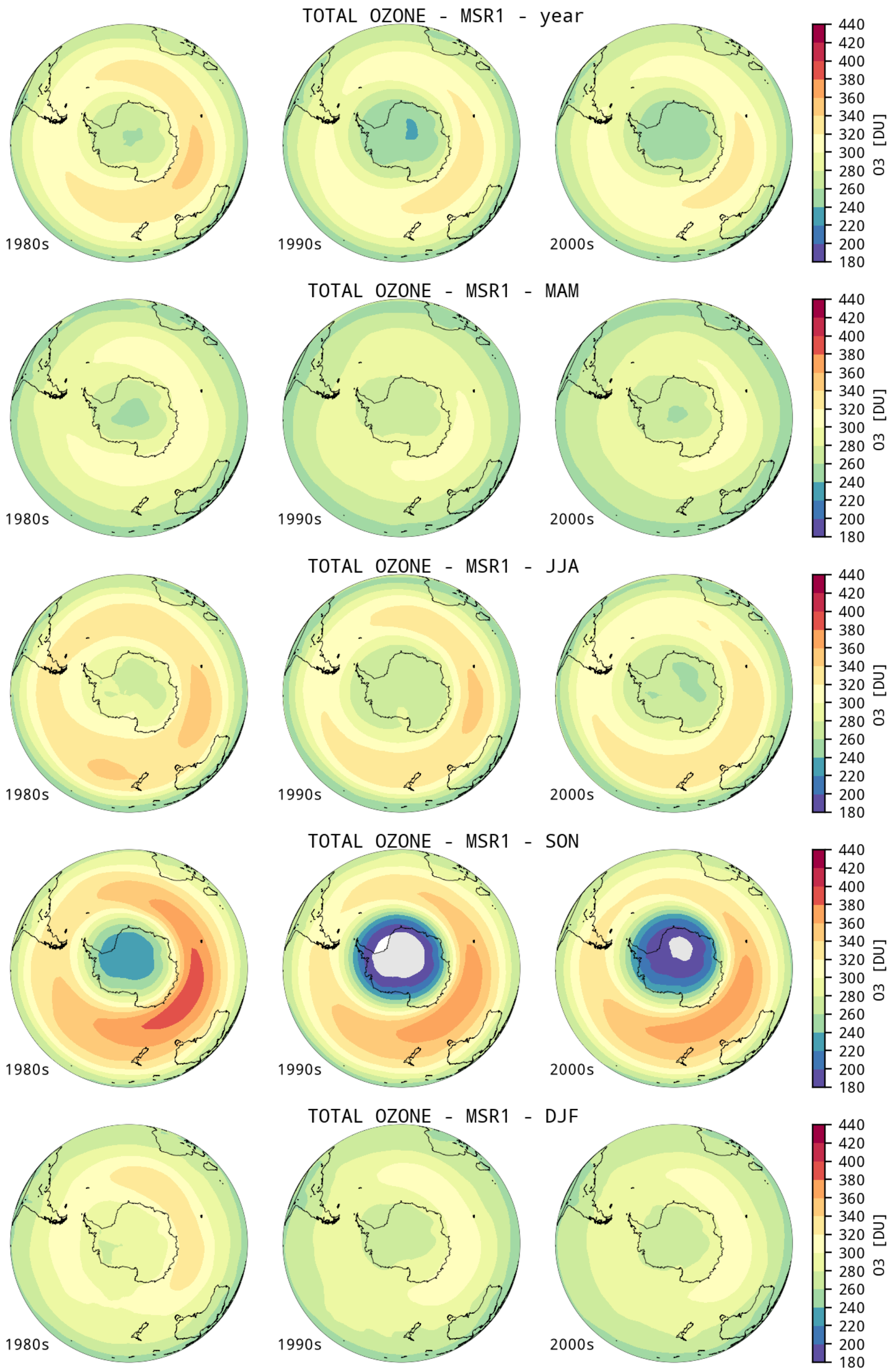

Figure 1.3.: Total Ozone Column on the Sourthen Hemisphere for MSR-1 dataset, for the mean of the full year and for each season: spring, summer, autumn and winter.

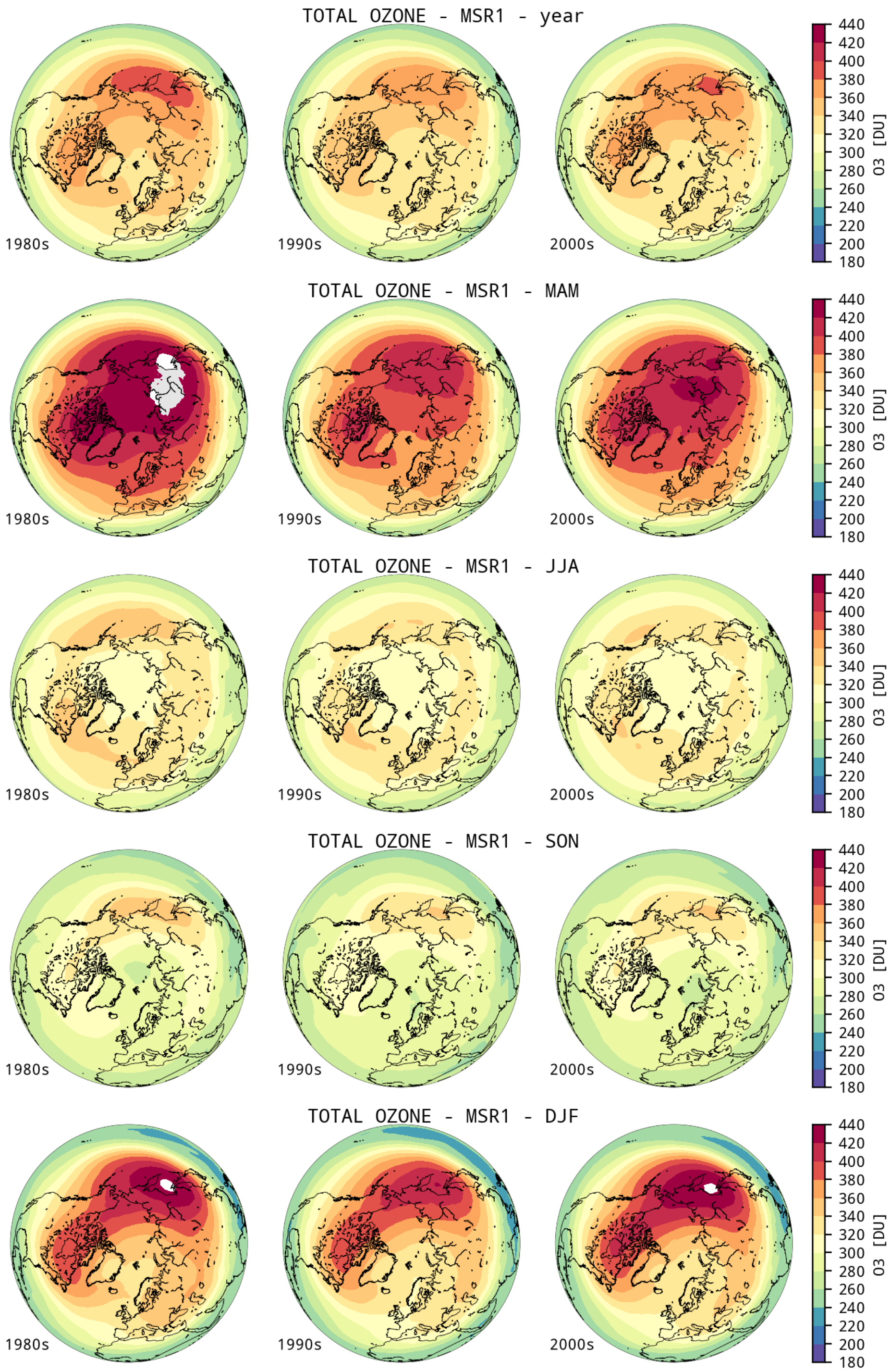

Figure 1.4.: Total Ozone Column on the Northern Hemisphere for MSR-1 dataset, for the mean of the full year and for each season: spring, summer, autumn and winter.

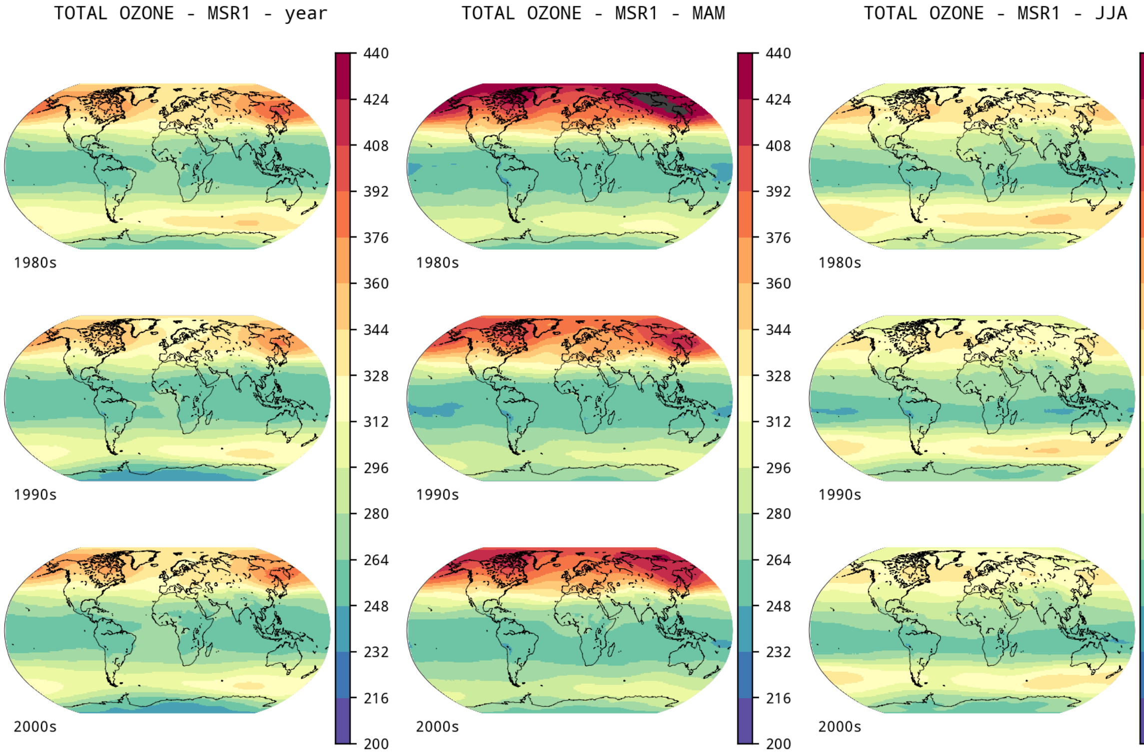

Figure 1.5.: Total Ozone Column for MSR-1 dataset, for the mean of the full year and for the seasons: spring and summer

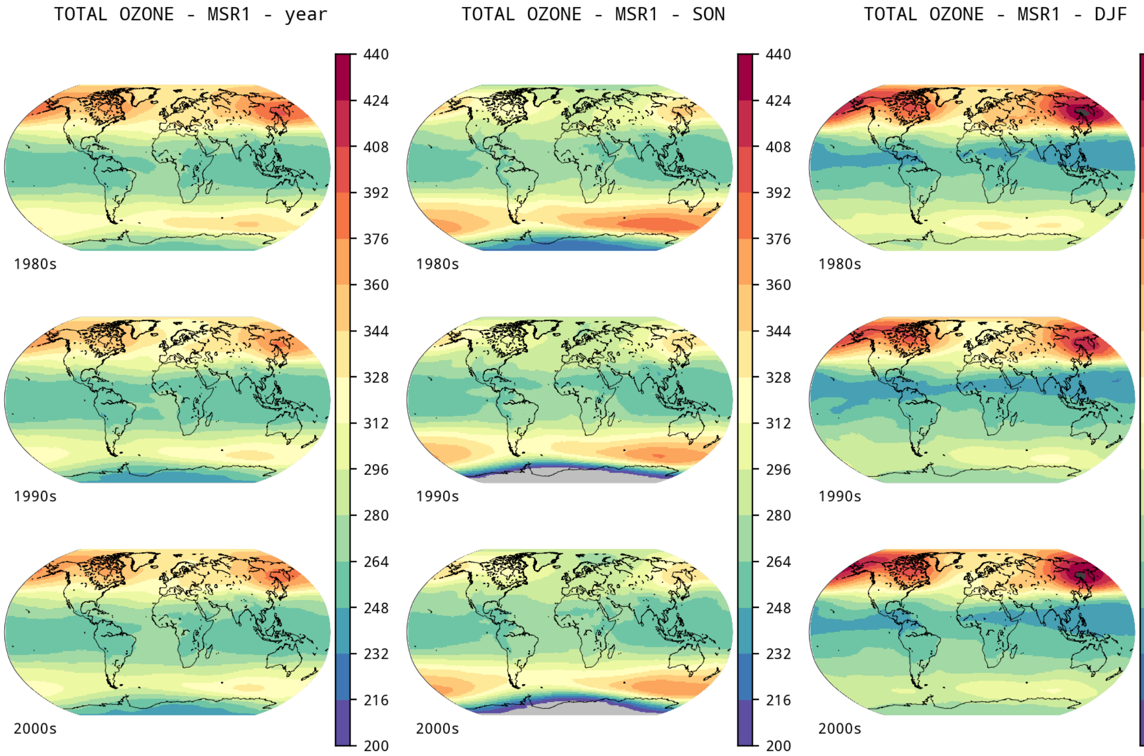

Figure 1.6.: Total Ozone Column for MSR-1 dataset, for the mean of the full year and for the seasons: autumn and winter

This section shows the same figures that those plotted for the MSR-1 dataset but from CMIP6 TOC product.

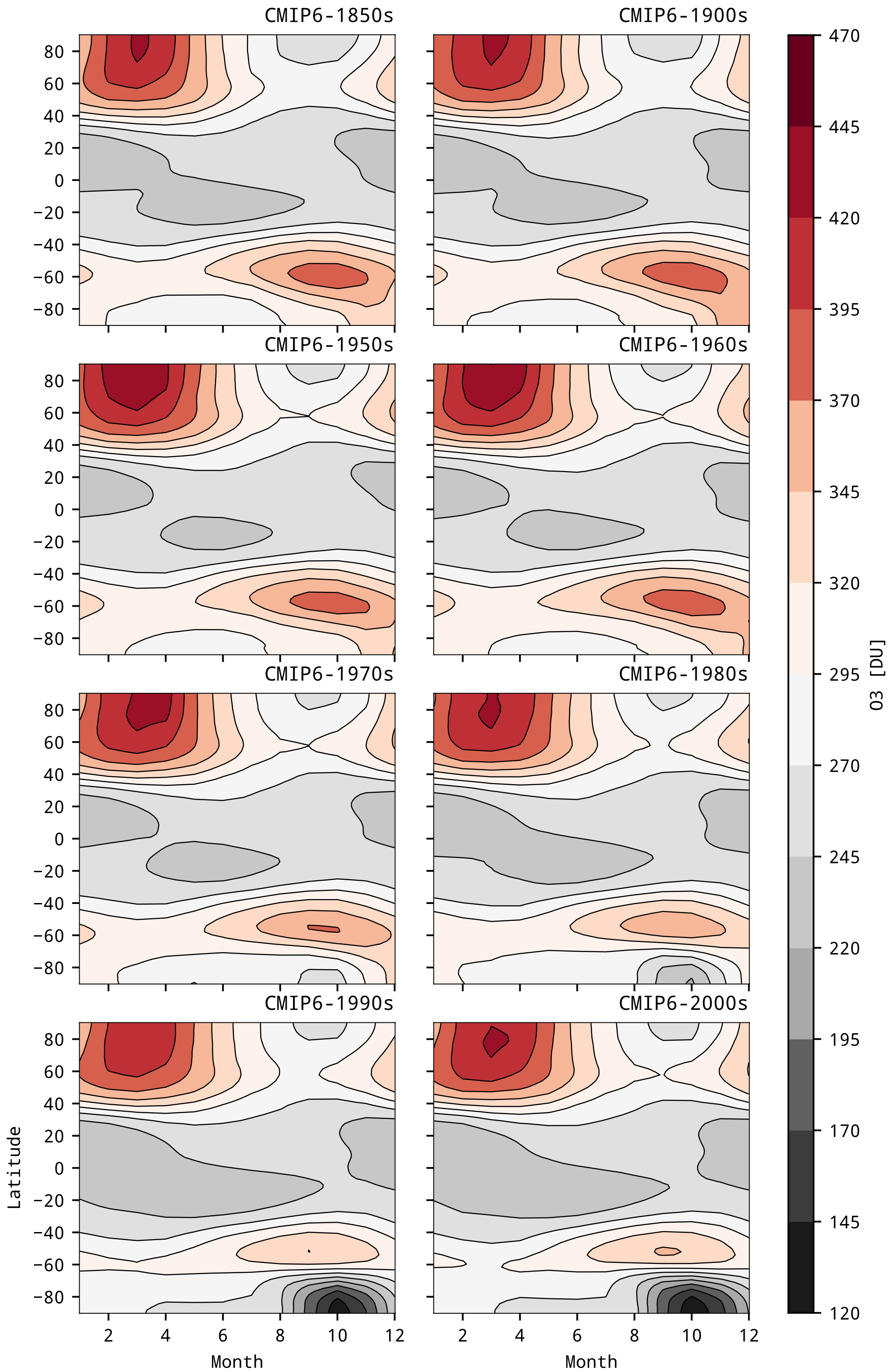

Figure 1.7.: Total Ozone Column for CMIP6 dataset

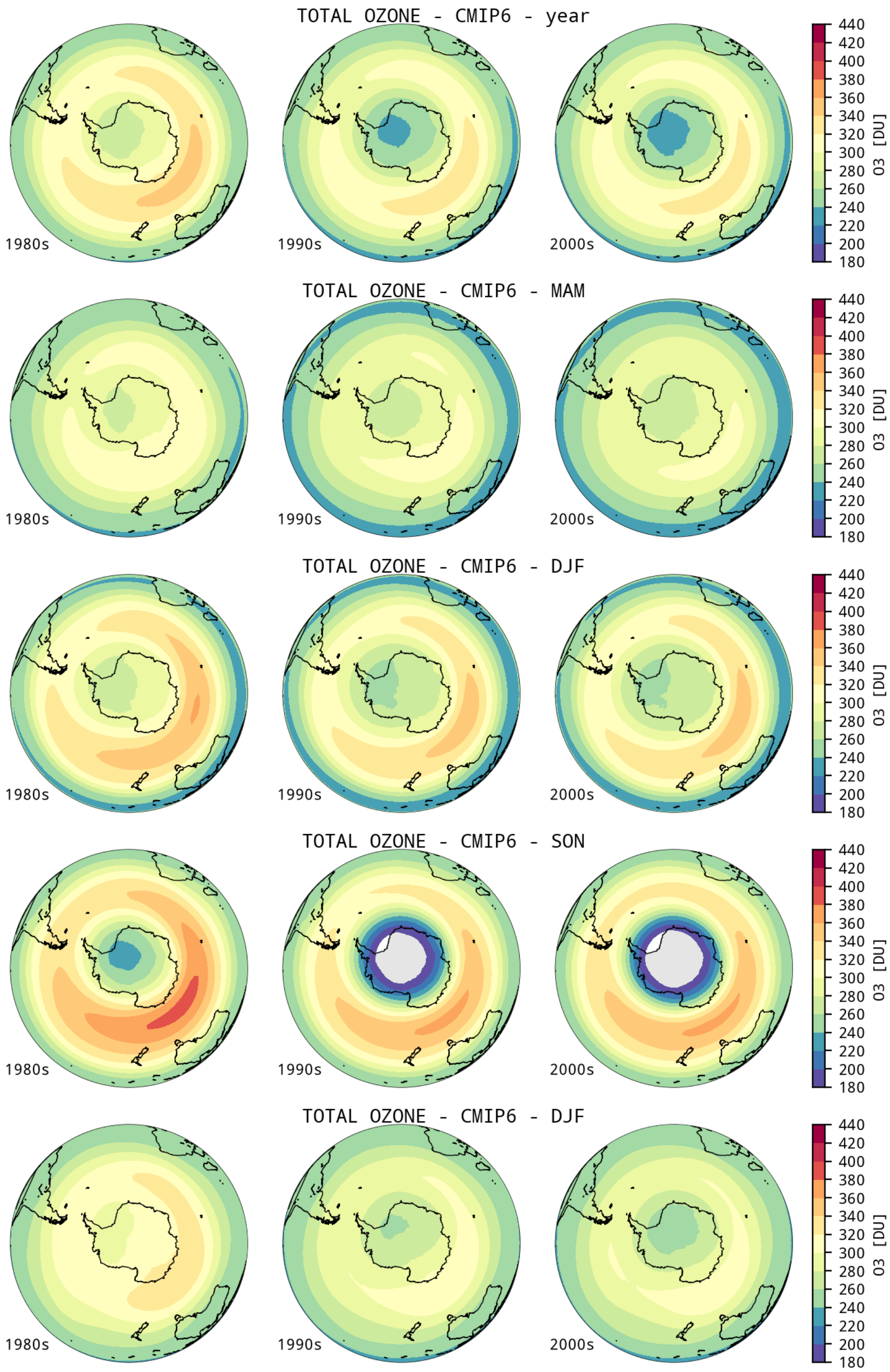

Figure 1.8.: Total Ozone Column on the Sourthen Hemisphere for CMIP6 dataset, for the mean of the full year and for each season: spring, summer, autumn and winter.

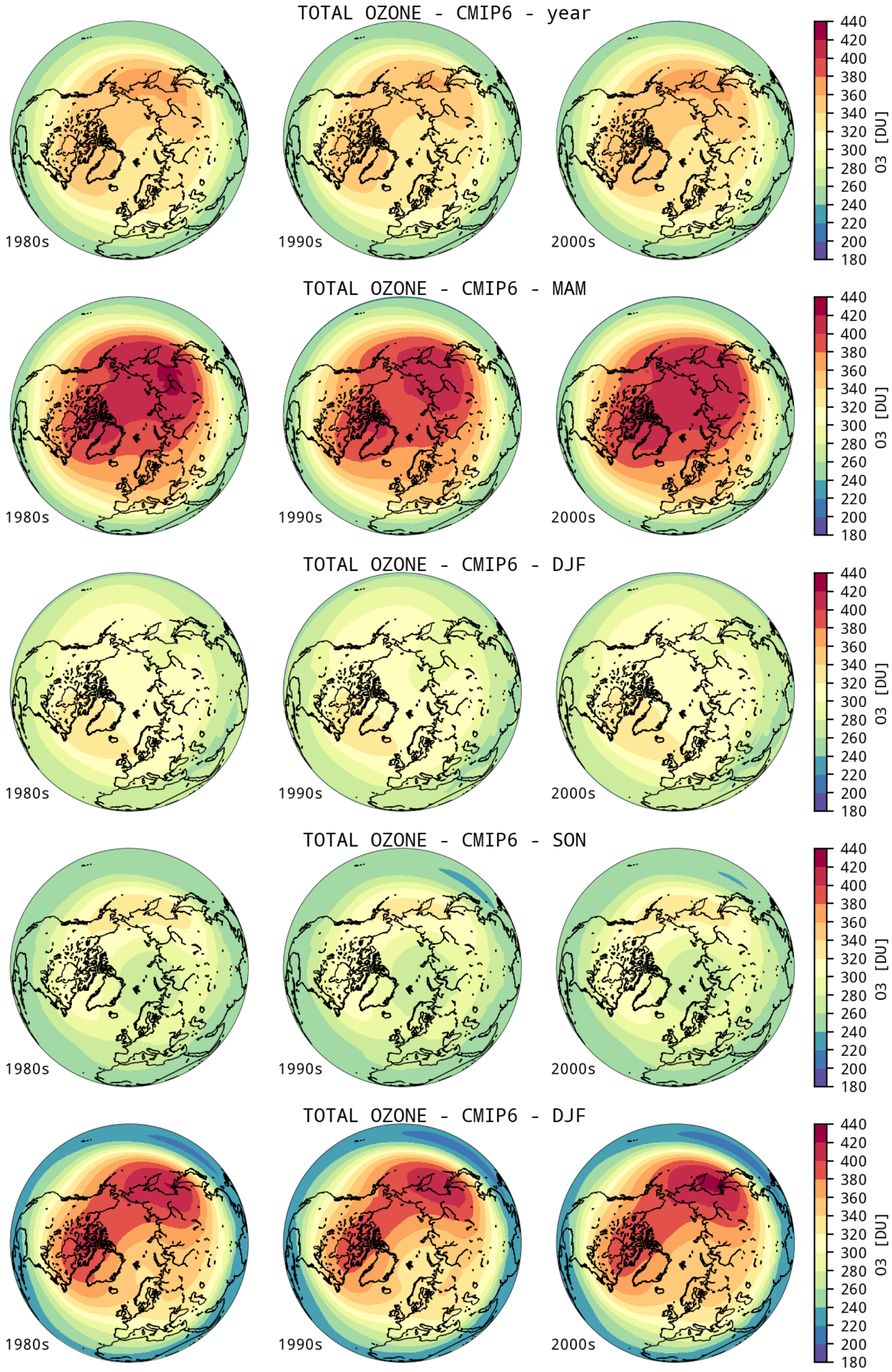

Figure 1.9.: Total Ozone Column on the Northern Hemisphere for CMIP6 dataset, for the mean of the full year and for each season: spring, summer, autumn and winter.

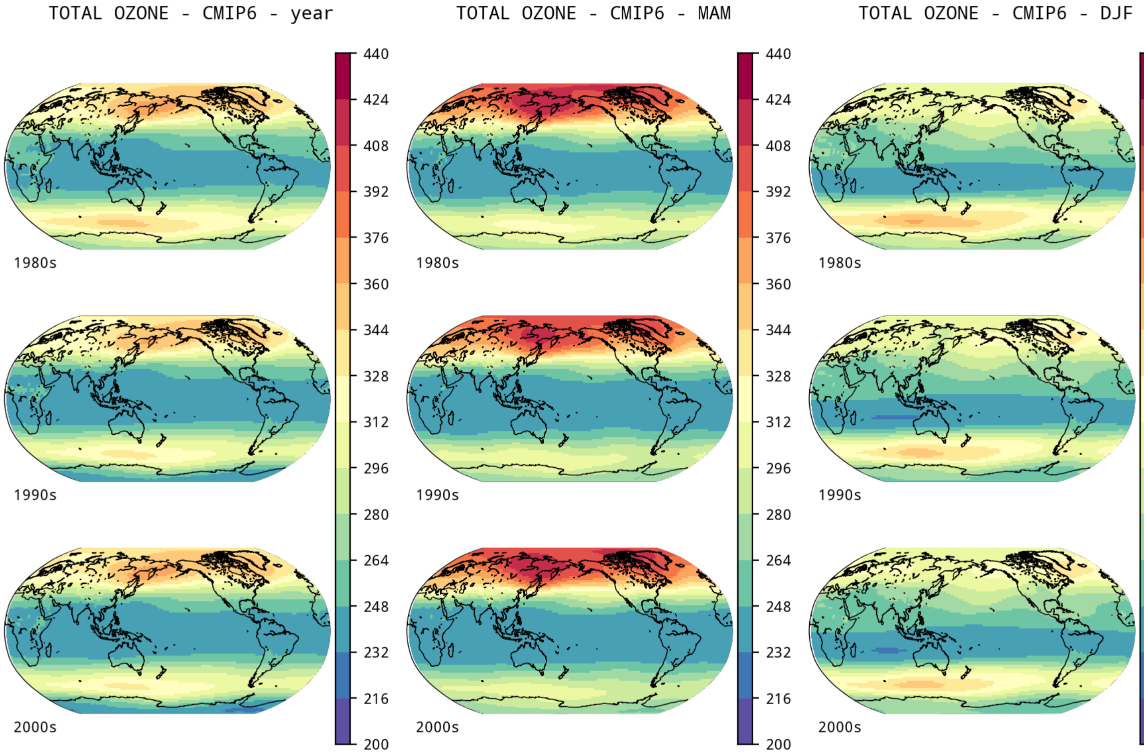

Figure 1.10.: Total Ozone Column for CMIP6 dataset, for the mean of the full year and for the seasons: spring and summer

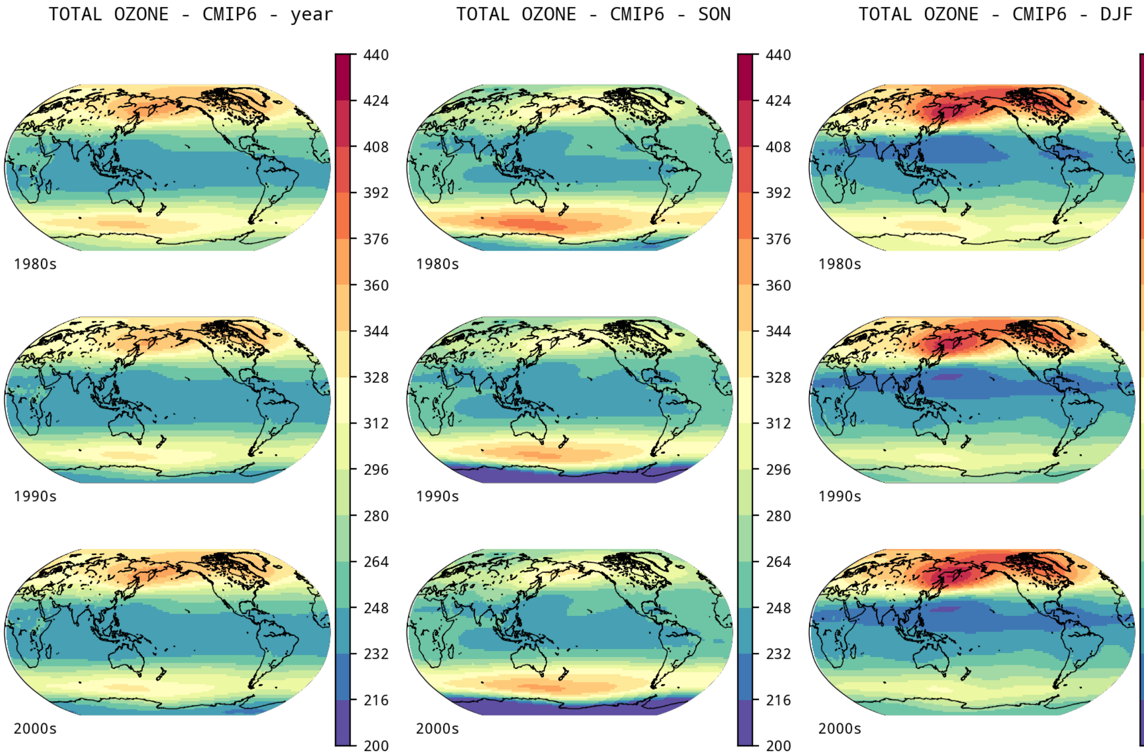

Figure 1.11.: Total Ozone Column for CMIP6 dataset, for the mean of the full year and for the seasons: autumn and winter

2 CMIP6 O3 database: time-series

2.1 Time series at latitudinal bands

In this section are included the time-series on a specific latitudinal band and at specific vertical levels. They are compared with several satellite platforms whose data has been prepared for these comparisons (and provided as zonal means by SPARC-DATA Initiative).

2.1.1 Band: 60N to 80N

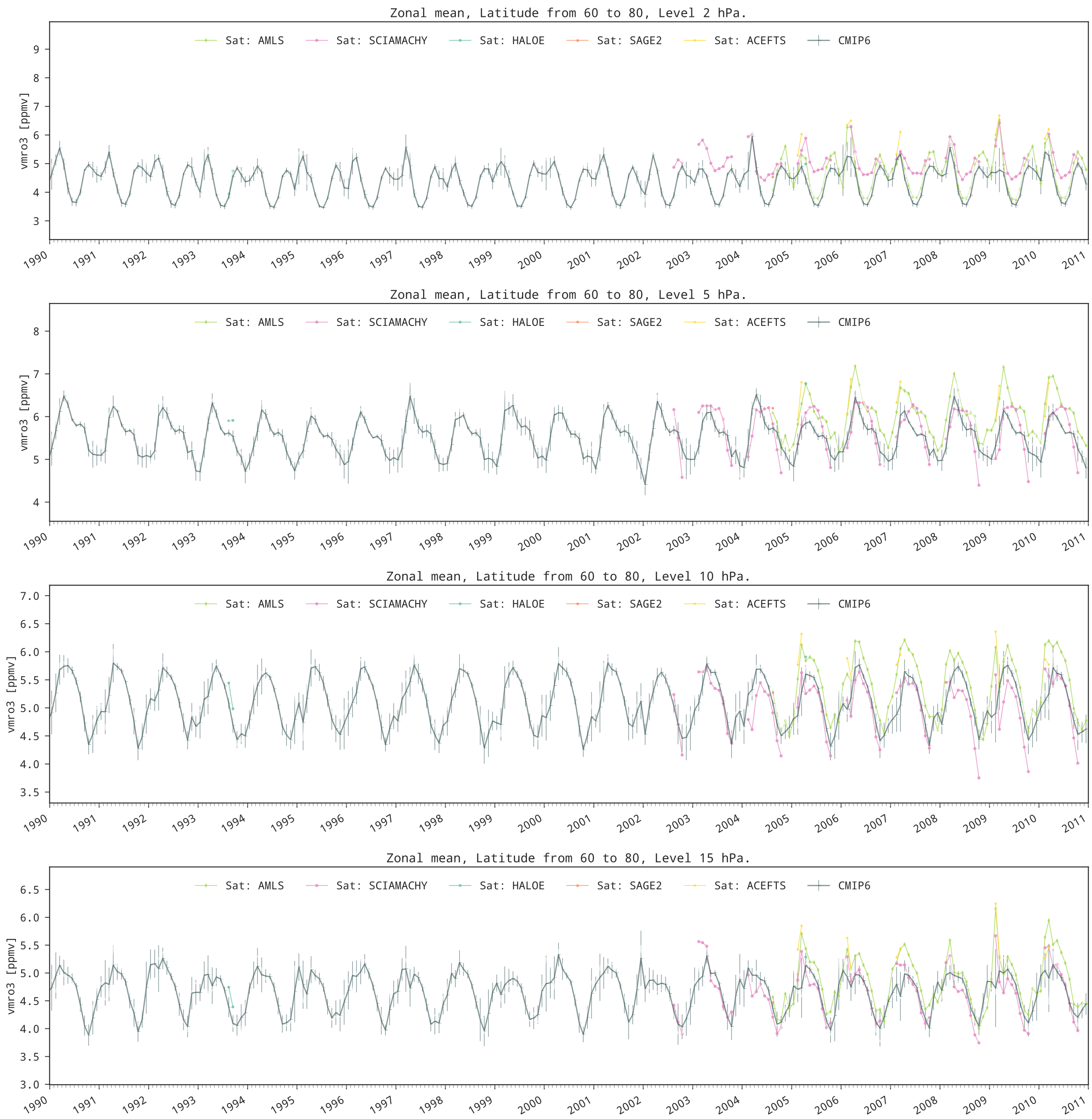

Figure 2.1.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 60N and 80N for levels from 1 to 15hPa.

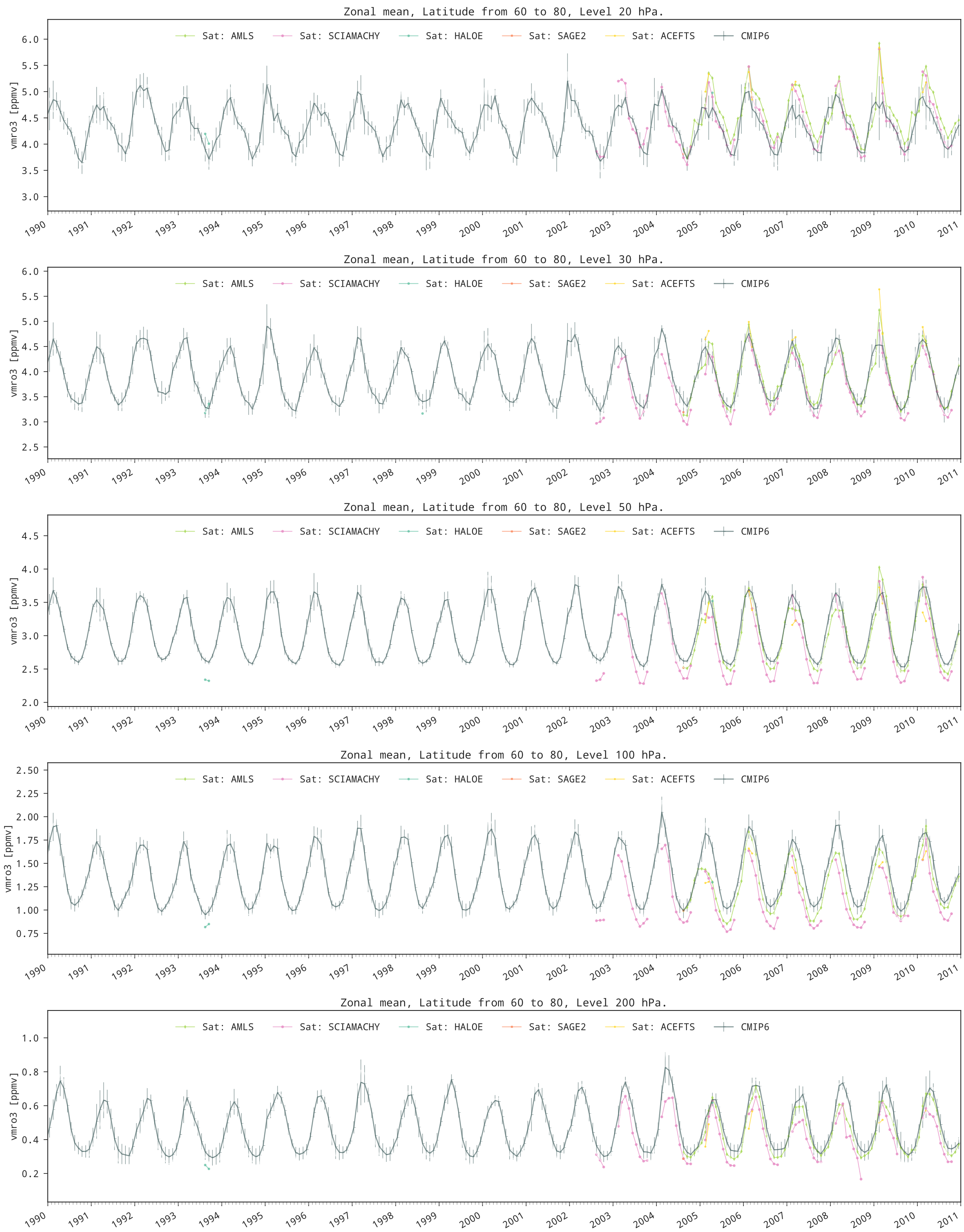

Figure 2.2.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 60N and 80N for levels from 20 to 200hPa.

2.1.2 Band: 40N to 60N

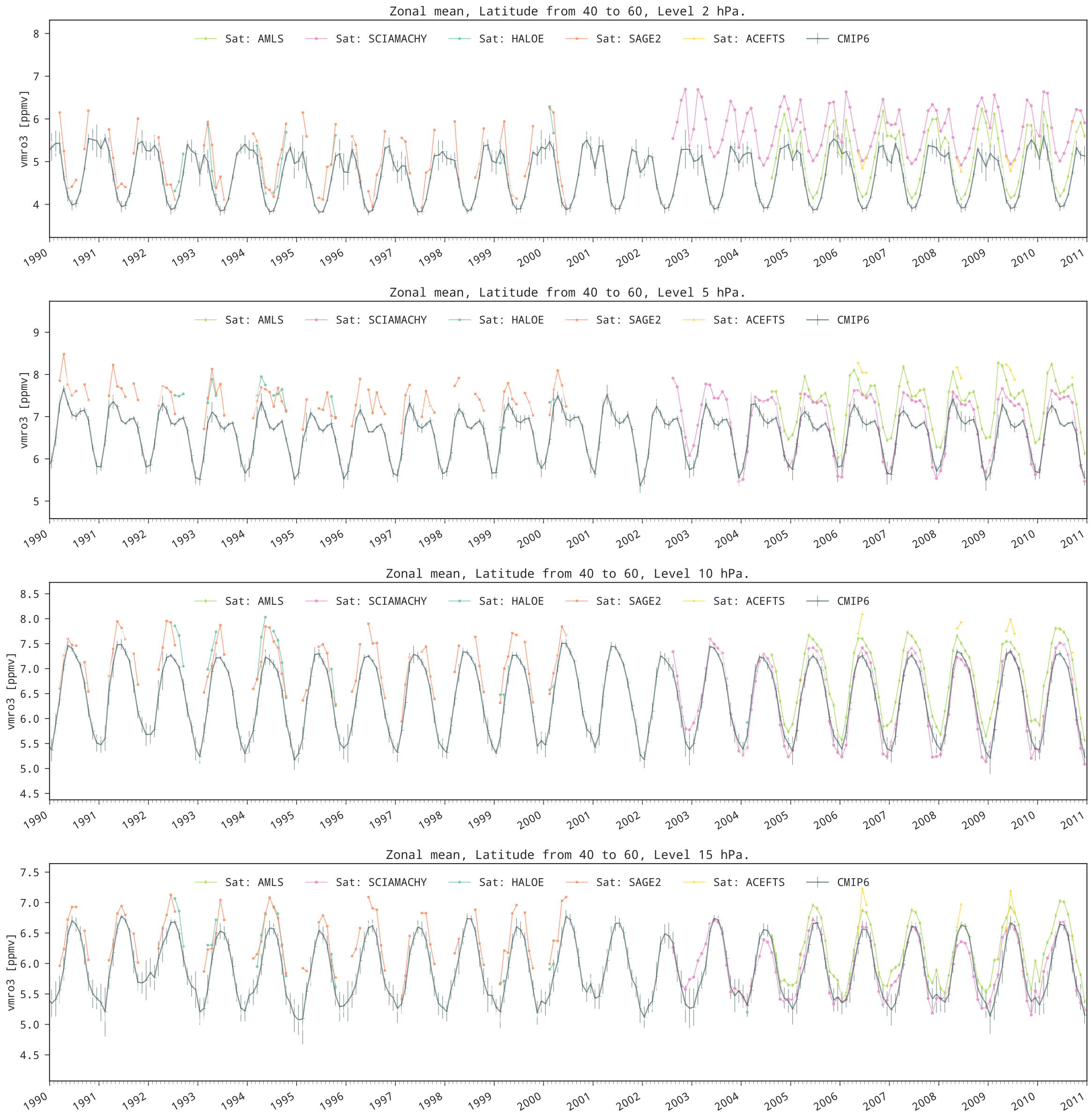

Figure 2.3.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 40N and 60N for levels from 2 to 15hPa.

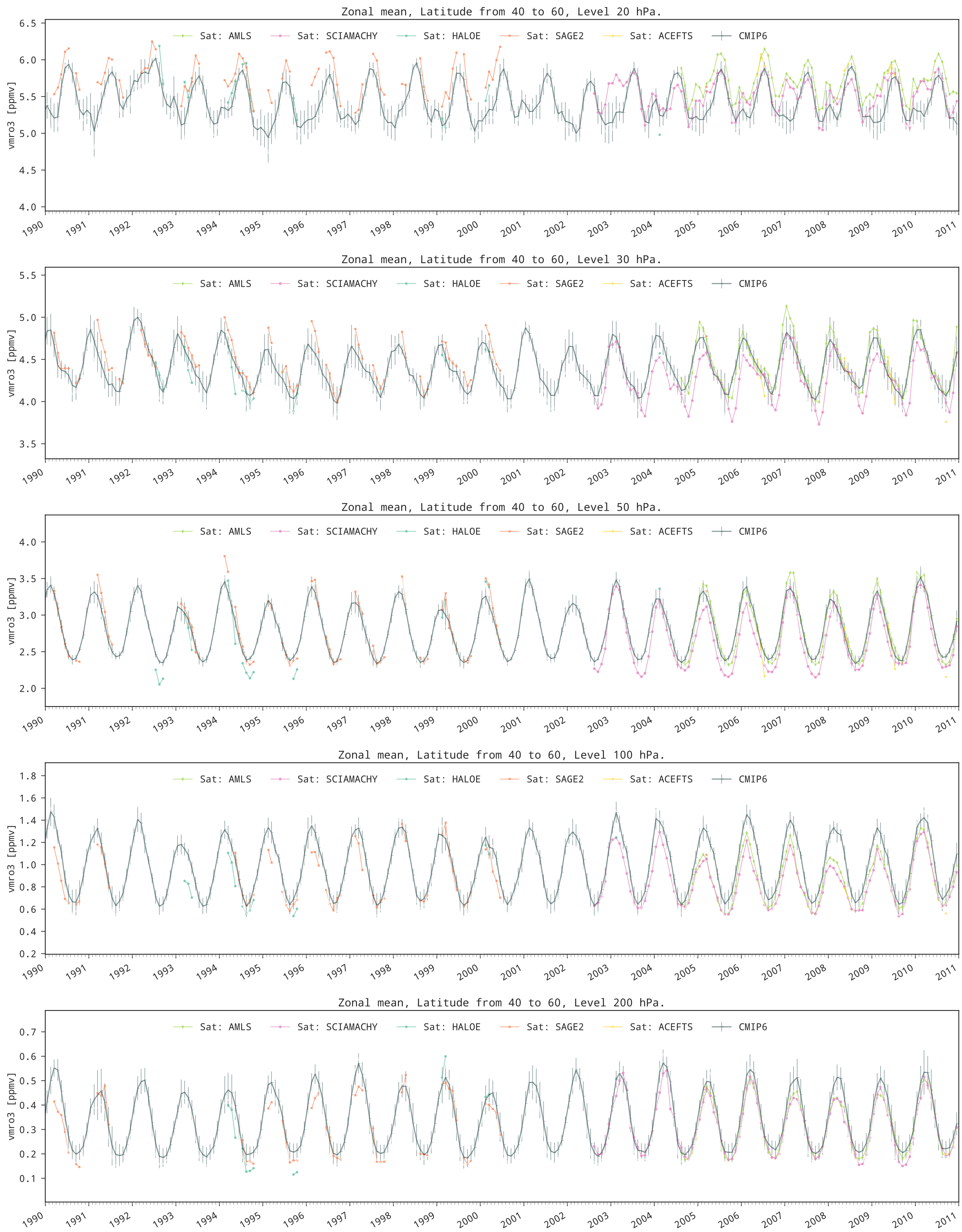

Figure 2.4.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 40N and 60N for levels from 20 to 200hPa.

2.1.3 Band: 20N to 40N

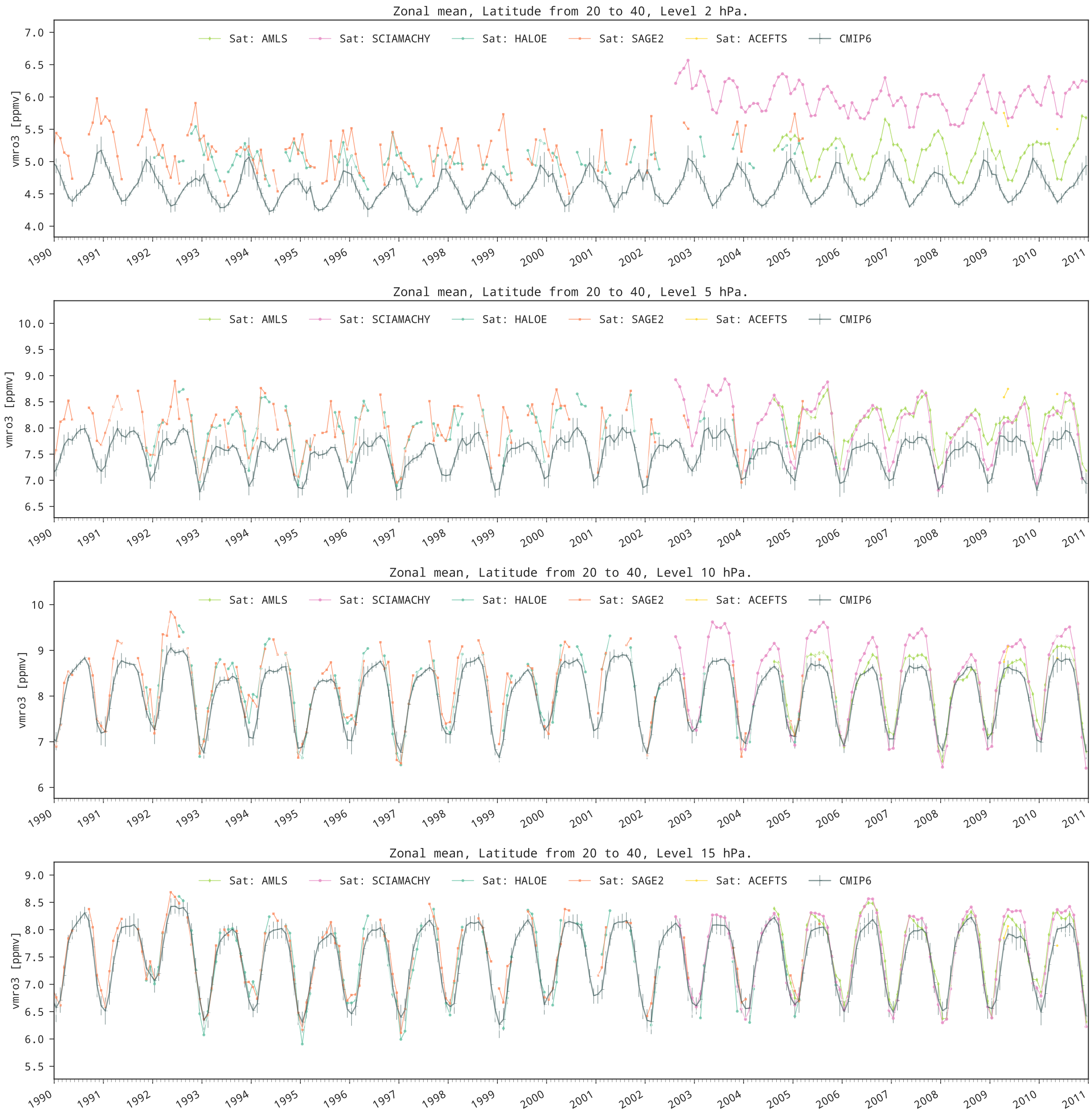

Figure 2.5.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 20N and 40N for levels from 2 to 15hPa.

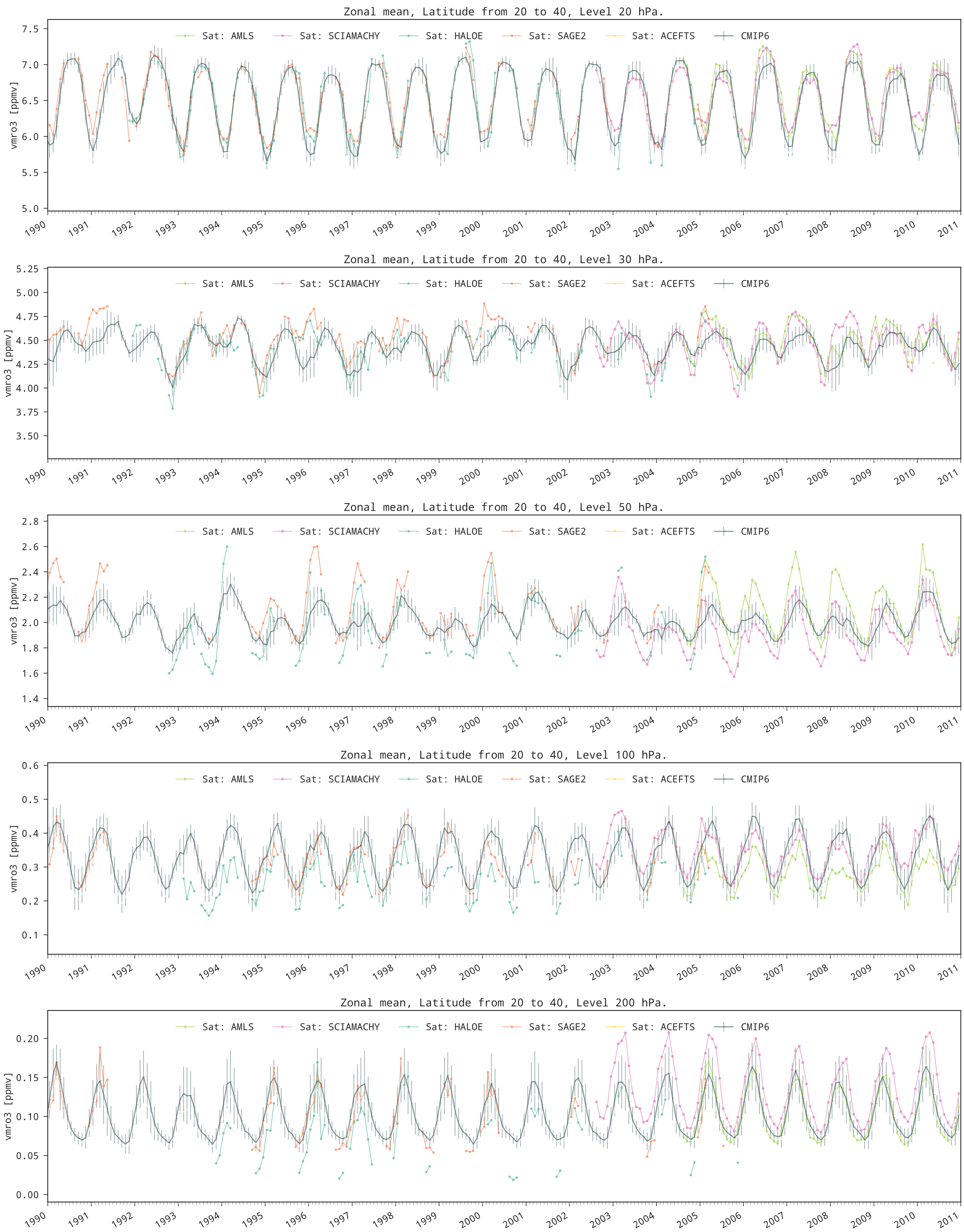

Figure 2.6.: Time series of CMIP6 ozone concentrations: zonal mean between latitudes 20N and 40N for levels from 20 to 200hPa.

2.1.4 Band: 20S to 20N

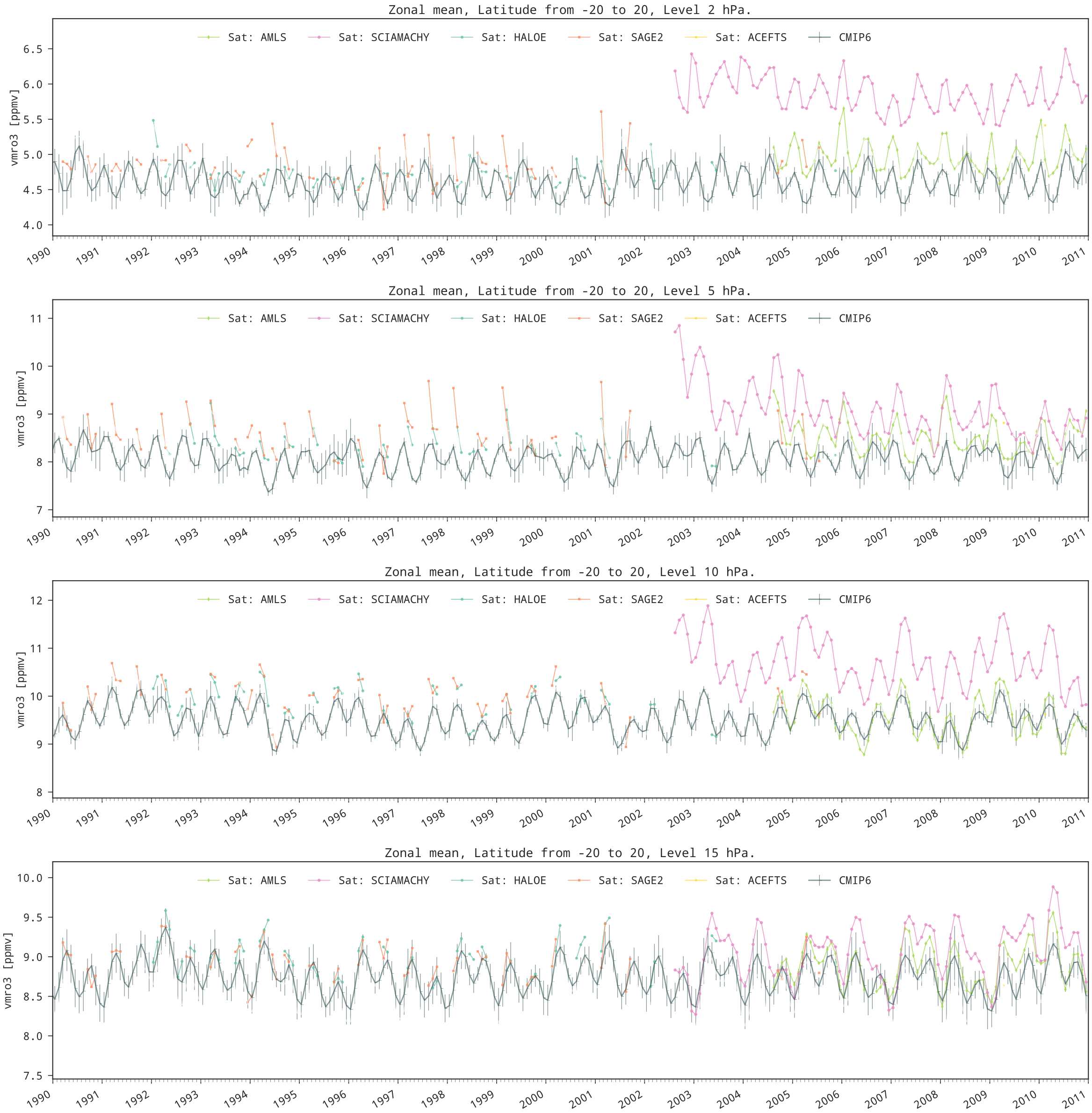

Figure 2.7.: Time series of CMIP6 ozone for the zonal mean between latitudes -20 and 20N for levels from 2 to 15hPa.

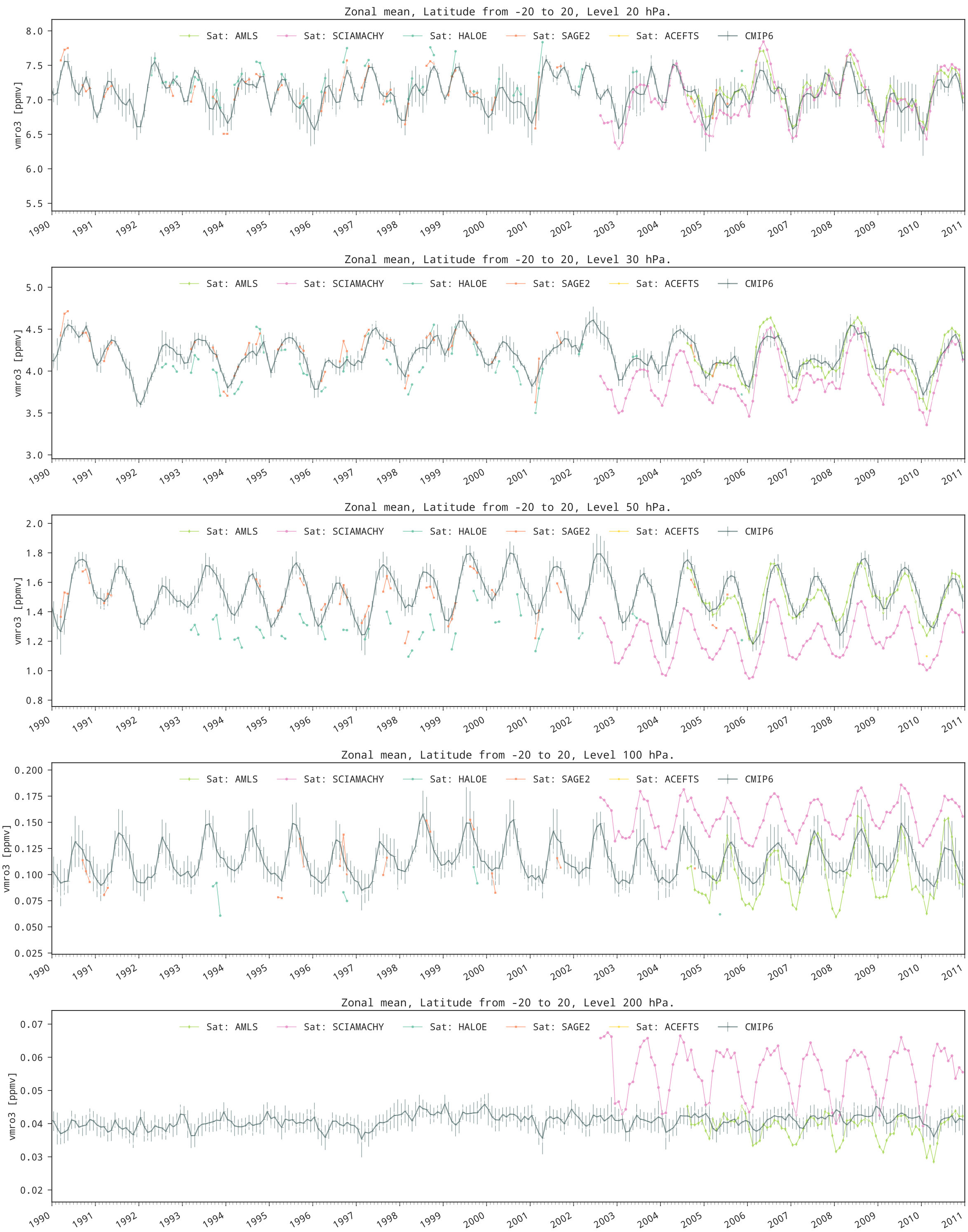

Figure 2.8.: Time series of CMIP6 ozone for the zonal mean between latitudes -20 and 20N for levels from 20 to 200hPa.

2.1.5 Band: 30S to 60S

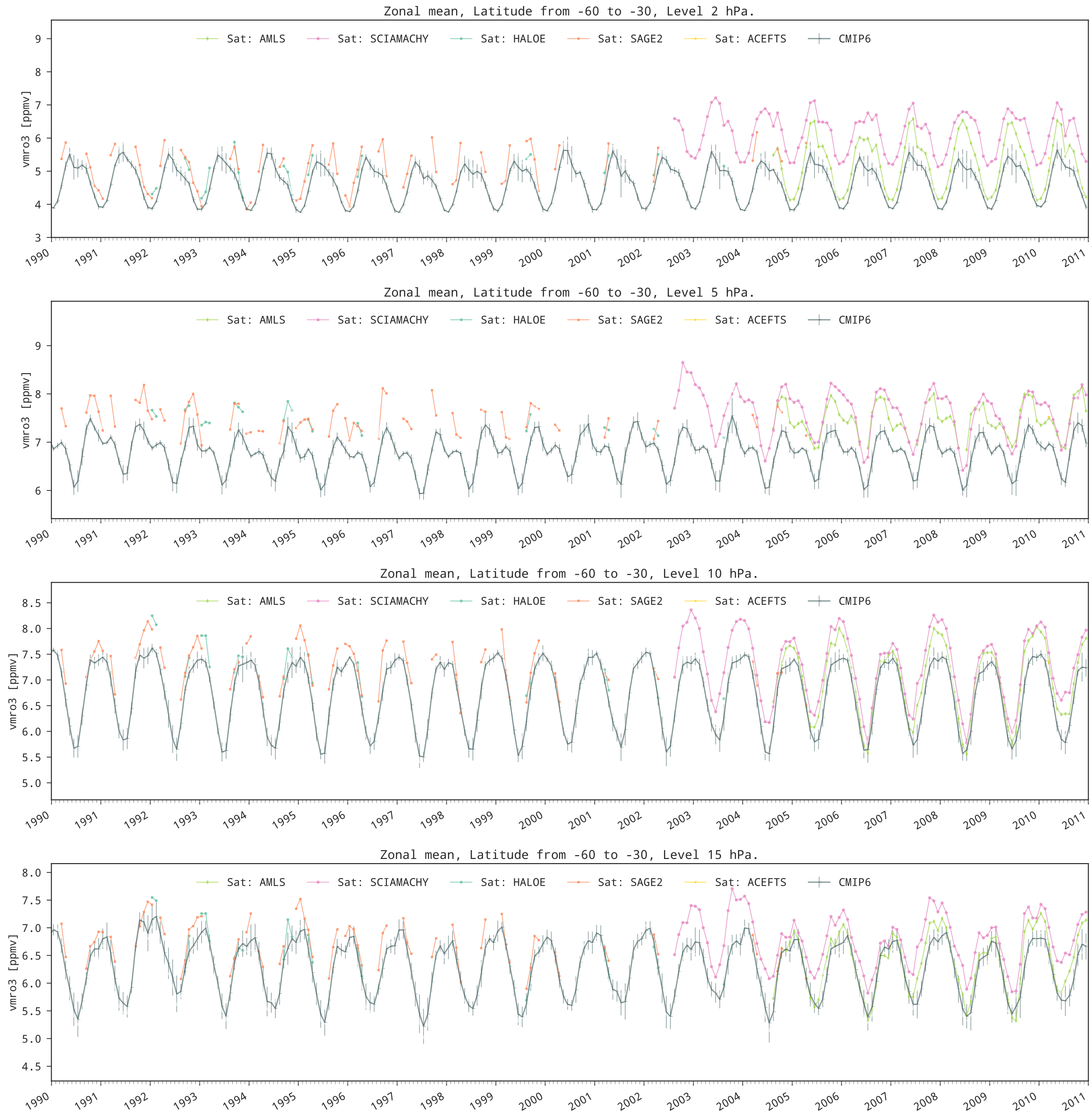

Figure 2.9.: Time series of CMIP6 ozone for the zonal mean between latitudes 30S and 60S for levels from 2 to 15hPa.

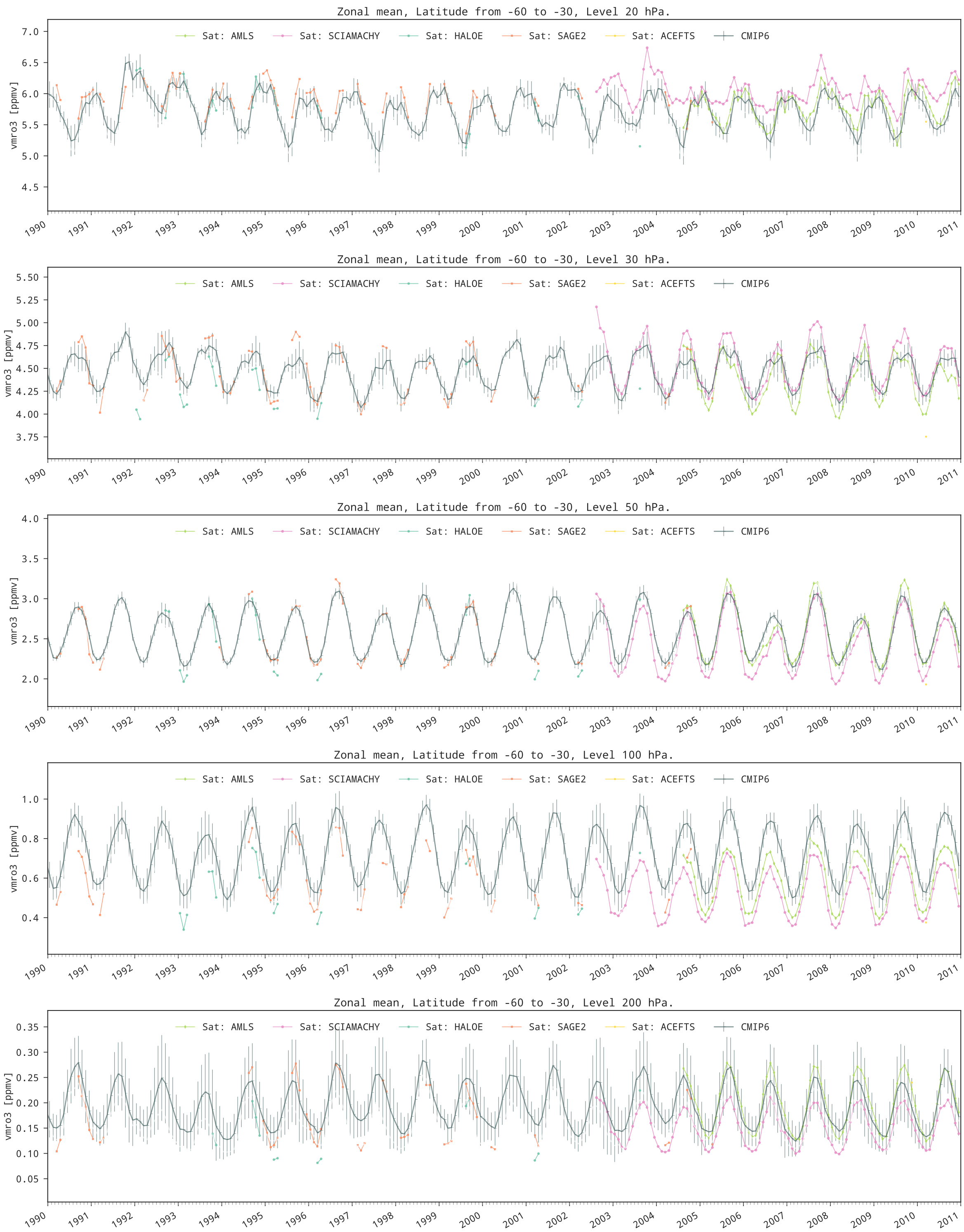

Figure 2.10.: Time series of CMIP6 ozone for the zonal mean between latitudes 30S and 60S for levels from 20 to 200hPa.

2.1.6 Band: 60S to 80S

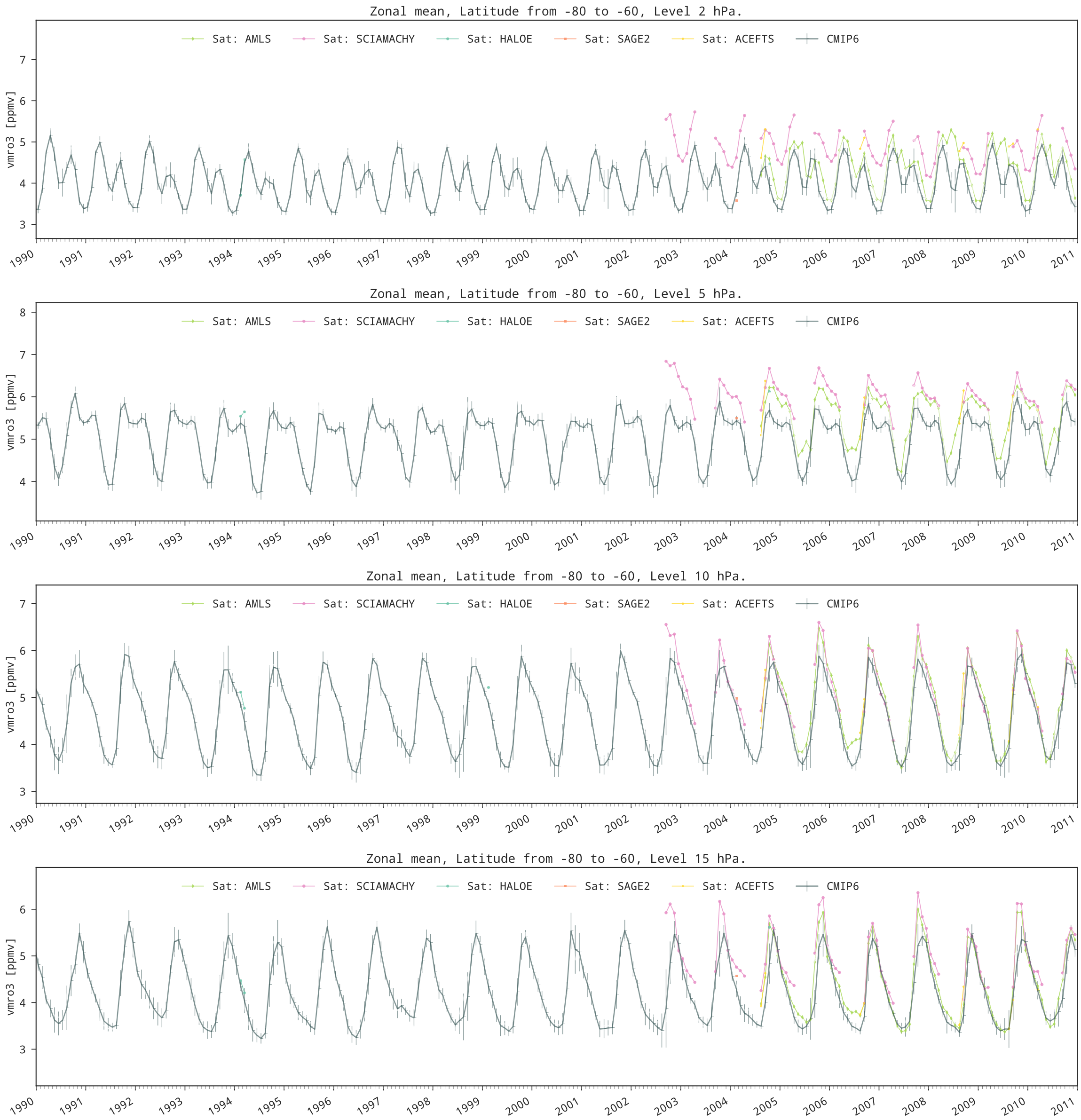

Figure 2.11.: Time series of CMIP6 ozone for the zonal mean between latitudes 60S and 80S for levels from 2 to 15hPa.

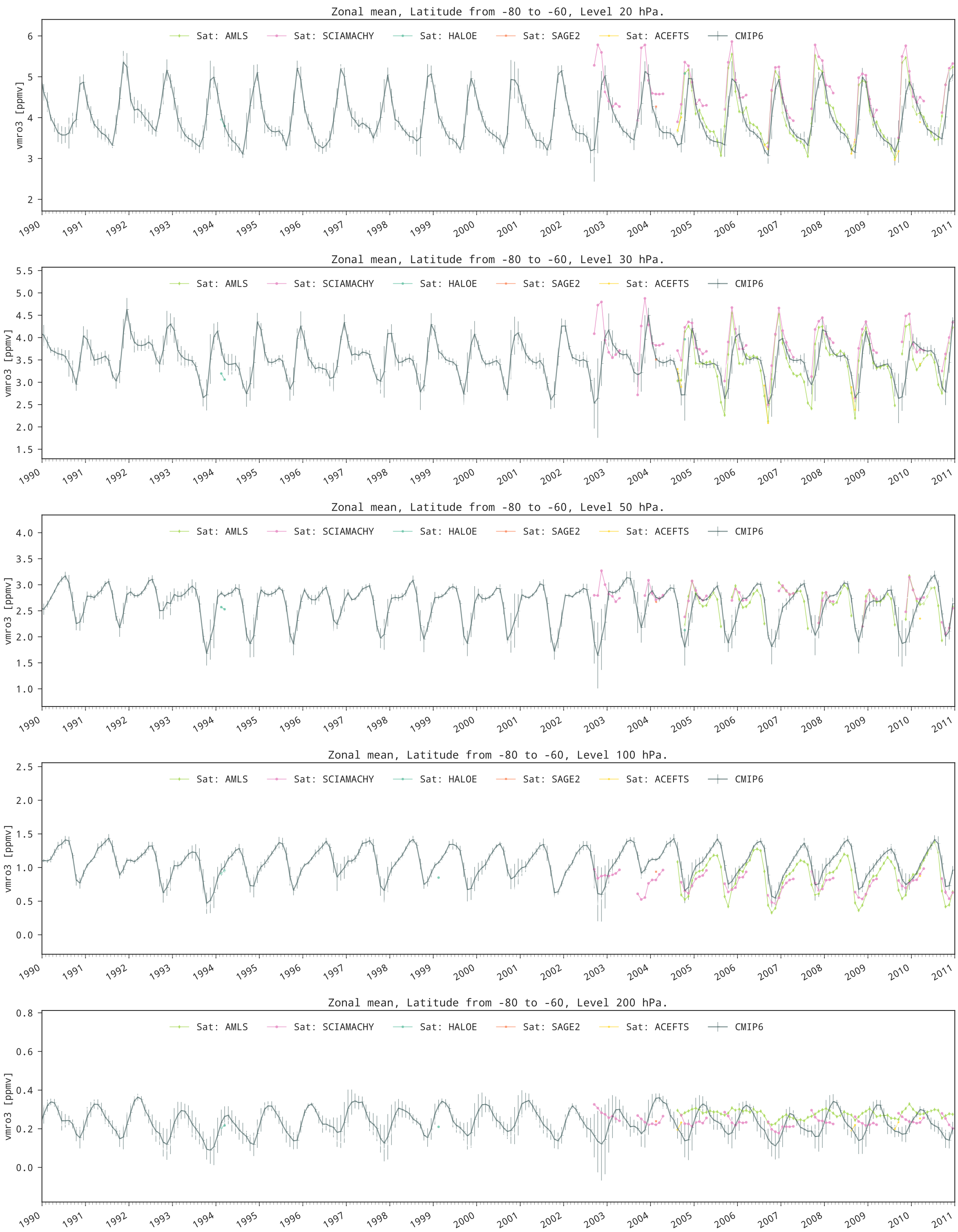

Figure 2.12.: Time series of CMIP6 ozone for the zonal mean between latitudes 60S and 80S for levels from 20 to 200hPa.

3 Information for CMIP6 Ozone concentration dataset

In this chapter I am including two figures that summarize the main methodological differences between CMIP5 and CMIP6 datasets, in support of the peer-review process of the publication "Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database" at Geophysical Research Letters. The figure 3.1 highlights properties of the CMIP5 ozone dataset regarding the sources of information and methods differentiated by time and vertical layers. The figure 3.2 shows the approach used on CMIP6 which aims to be more consistent on the vertical structure and in the historical times.

3.1 CMIP5 ozone dataset

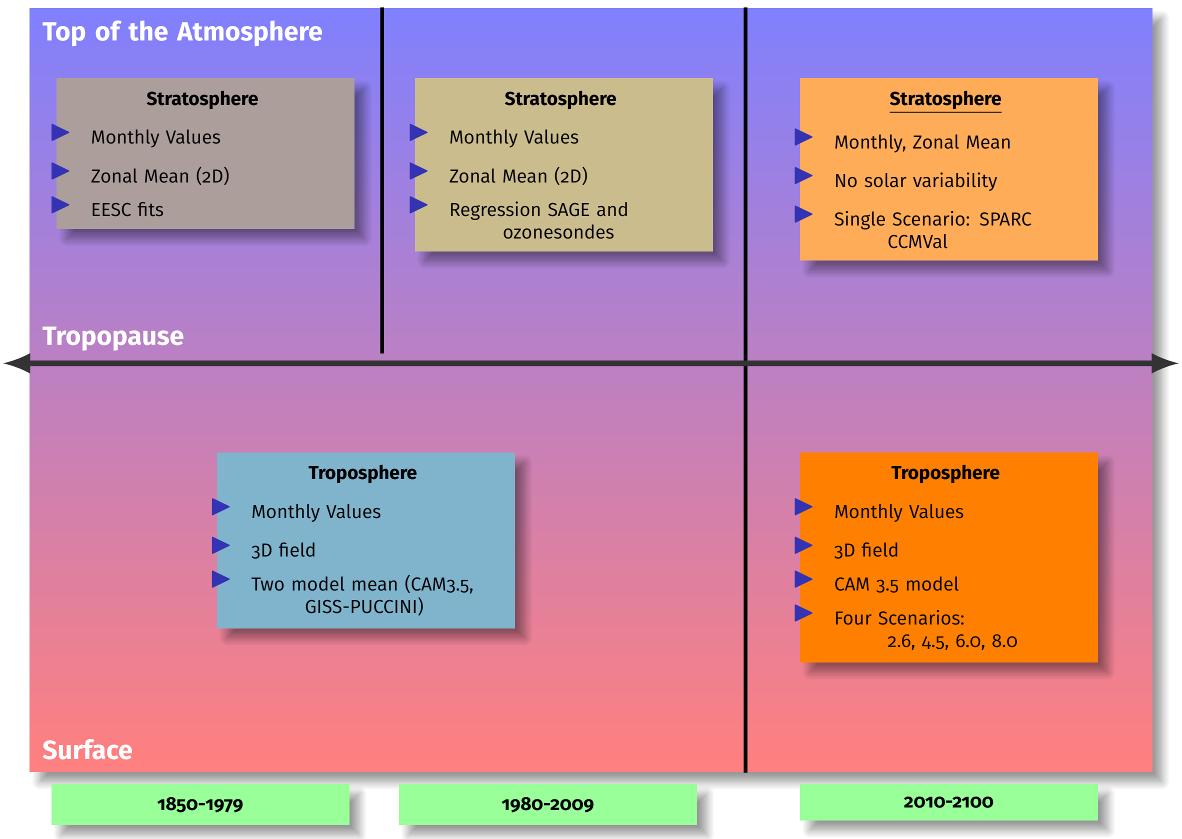

Figure 3.1.: Overview of the main properties of the CMIP5 Ozone dataset.

A main property of the CMIP5 dataset is that it is created from several sources of information added together on a single dataset. The figure 3.1 shows that the troposphere and the stratosphere are based on different approaches: the troposphere is given as a three dimensional field based on the arithmetic mean of two chemistry-climate models, on the other side the stratosphere values are zonal mean comming from regression models: ESSC for historical concentrations, and regression from SAGE satellite and two ozonesondes for the last decades. On the future scenarios there is not solar variability implemented and also relies on one single model. Note that the ESSC fits of CMIP5 are not including specific basis functions for ENSO or Volcano.

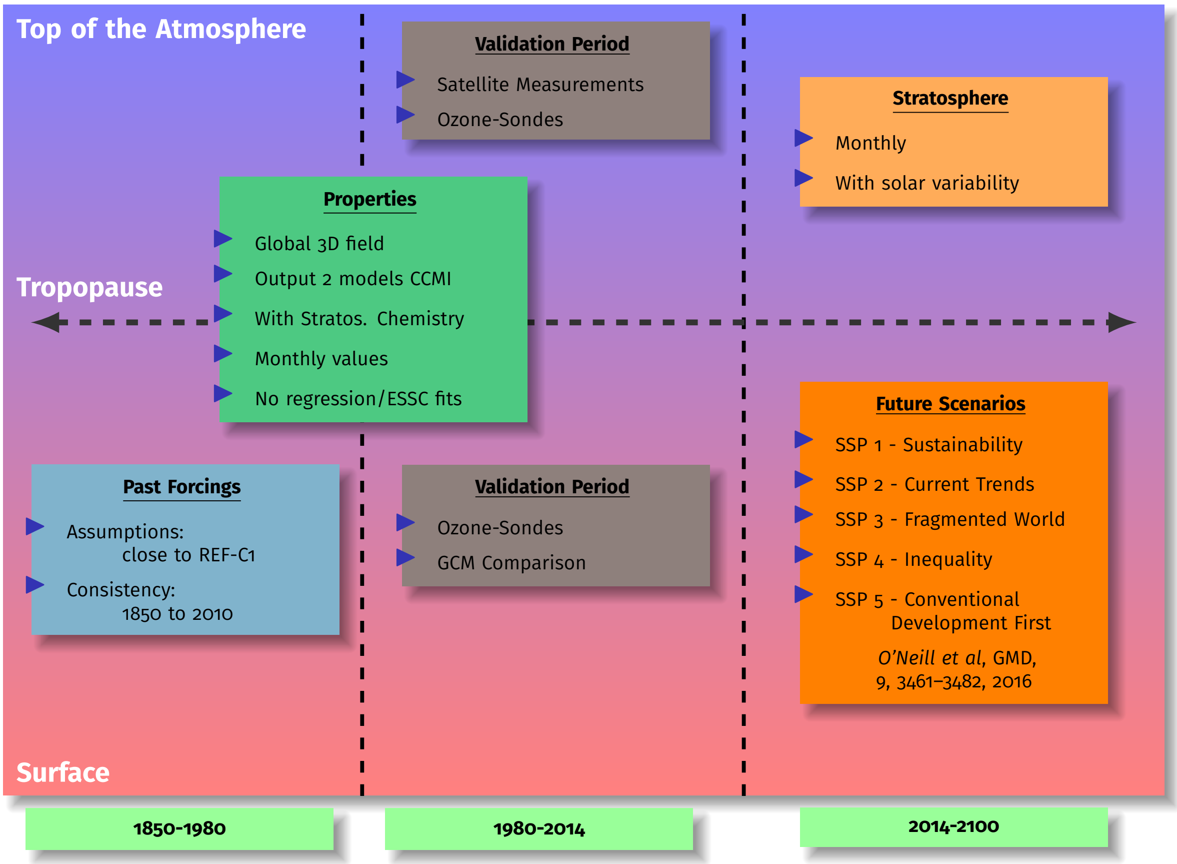

Figure 3.2.: Overview of the main properties of the CMIP6 Ozone dataset.

In the picture 3.2 there is not a different source of information between troposphere and stratosphere, this was achieved by using two climate models with tropospheric but also stratospheric chemistry. One of the two models used even has interactive chemistry up to 140 km height. In this situation now we expect more consistency in the transition between the troposphere and the straosphere values. Also it is worth to comment that given that in the last decades we are not relying on a regression over a satellite values, then we can compare the dataset with satellite measurements for validation instead of creation, having a kind of uncertainty that could be useful. The future scenarions will implement the solar variability and will reproduce the new SSP1 to SSP5 geo-political situations described in the figure.

	CESM1-WACCM	CMAM
Horizontal grid	1.9x2.5	T47
Model Top	0.00596Pa	0.08Pa
Number of Levels	L66/L88	L71
Tropospheric Chemistry	Yes	Yes (CH4)
Stratospheric Chemistry	Yes	Yes

Table 3.1.: Climate models with interactive chemistry used to create the CMIP6 ozone concentration dataset.

Tier-1 Scenarios	Similar RCP	RF by 2100
SSP5-8.5	RCP-8.5	8.5 Wm^{-2}
SSP3-7.0	-	7.0 Wm^{-2}
SSP2-4.5	RCP-4.5	4.5 Wm^{-2}
SSP1-2.6	RCP-2.6	2.6 Wm^{-2}
Tier-2 Scenarios		
SSP4-6.0	RCP-6.0	6.0 Wm^{-2}
SSP4-3.4	-	3.4 Wm^{-2}
SSP5-3.4-OS	-	3.4 Wm^{-2}

Table 3.2.: Brief description of the Future Scenarios mentioned on the figure 3.2. Please check the referece [6] for more details

4 Ozone radiative forcing for the CMAM model

In this chapter we show the estimation of ozone radiative forcing with an offline radiative transfer model by calculating the stratospherically-adjusted temperatures with the fixed dynamical heating methodology (as described for example in [5]). The code used relies on an extended version of the code *Community Radiative Transfer codes based on Edwards and Slingo* (SOCRATES) named SOCRATES-RF [2], also described in the publication *Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database*. In the case of the results presented here, the ozone concentrations are those coming from the CMAM simulation REF-C1 [7] but extended in time to cover the full period: 1850-2014. Like in the reference *Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database* we introduce the pre-industrial (PI) background state as monthly-means for the 1850-1859 decade. Then the RF calculations are calculated monthly for each decade from 1860-1869 to 2000-2009 named here, like in the original manuscript, perturbed state (PS). We show here, the geographical distribution of ozone radiative forcing for tropospheric and stratospheric ozone (Figure 4.1), the changes in the stratospheric temperature estimated (Figure 4.2) and a table with the global and the hemispheric values ozone radiative forcing (Table 4.1)

Important: This section shows the estimation of ozone radiative forcing with the same methodology that *Historical tropospheric and stratospheric ozone radiative forcing using the CMIP6 database* but based on CMAM model ozone concentrations dataset from a special model run to cover the full period 1850-2014. It was kindly provided by David Plummer, who would like to thank the Canadian Foundation for Climate and Atmospheric Sciences and the Canadian Space Agency for supporting the development of CMAM.

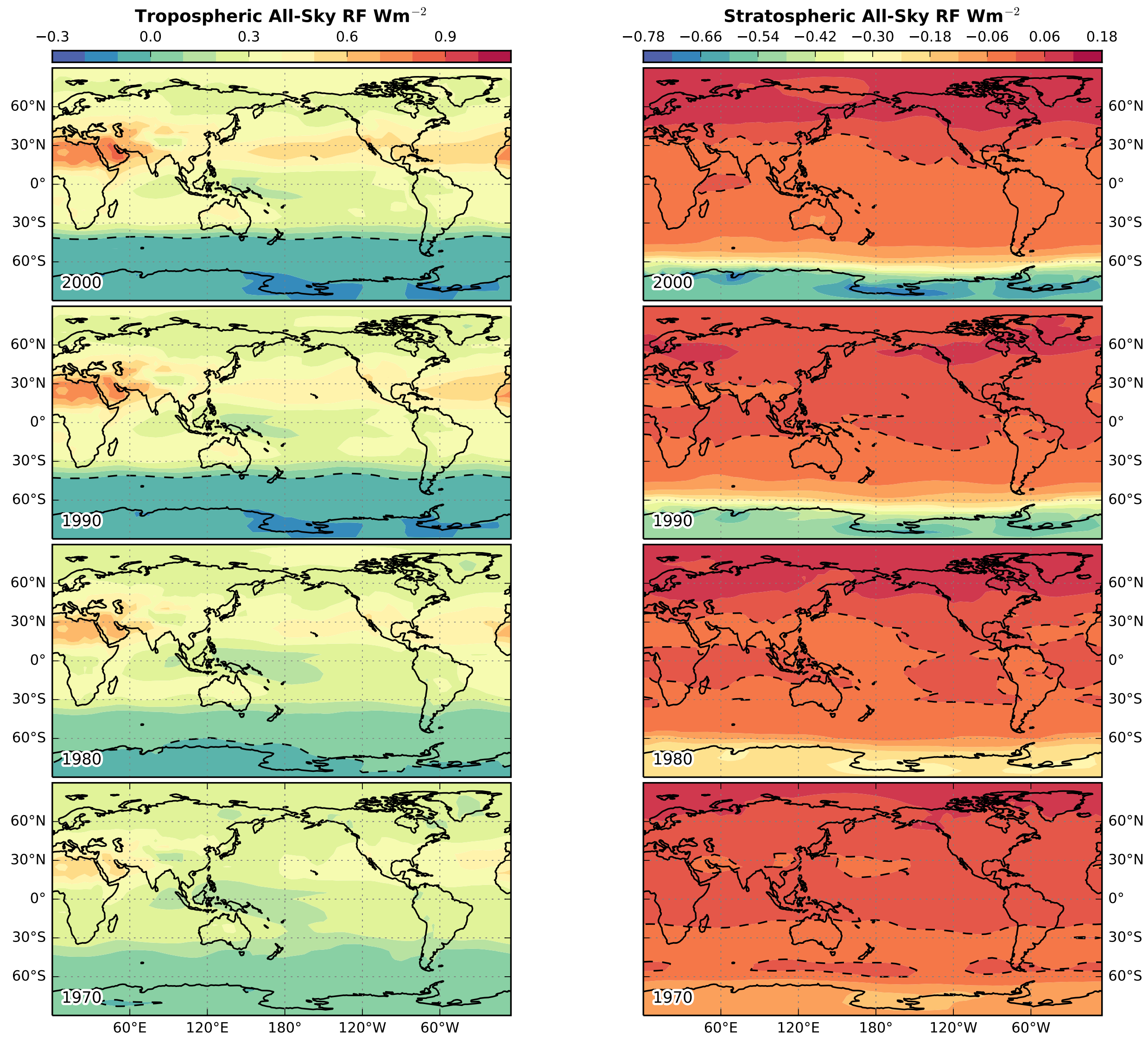

Figure 4.1.: Geographical distribution of radiative forcing (Wm^{-2}). Left: RF due to tropospheric ozone changes for the decades 2000s, 1990s, 1980s and 1970s (all with respect to 1850-1859). Right: Same as left column but for the stratospheric ozone change.

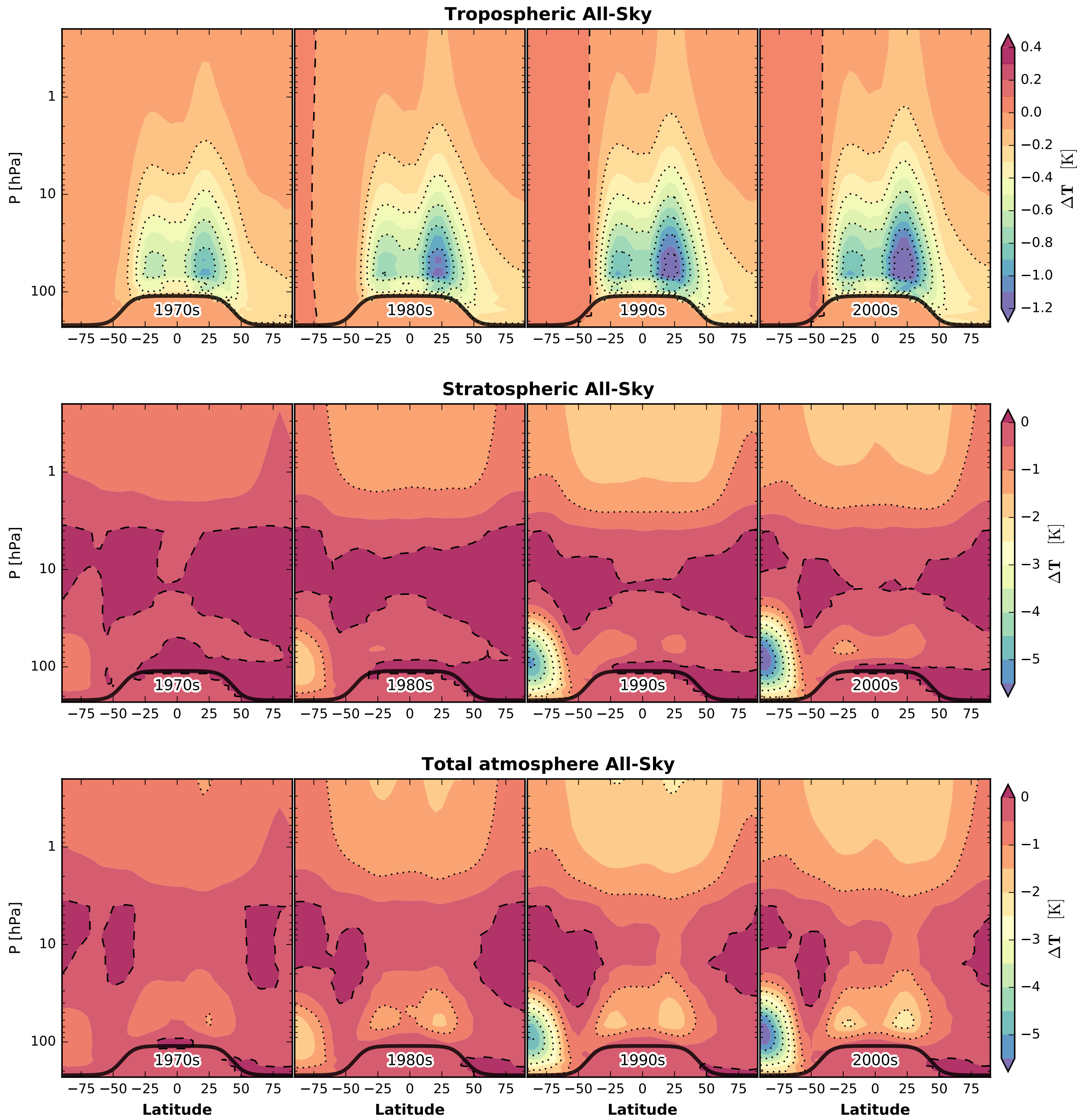

Figure 4.2.: Changes in stratospheric temperatures (ΔT in K) due to changes in CMAM ozone concentrations calculated using the fixed-dynamical heating approach. For tropospheric (top row), stratospheric (middle row), and total ozone changes (bottom row) for the four decades from 1970 to 2000s relative to the 1850s.

Table 4.1.: CMAM time series of stratospherically adjusted all-sky ozone radiative forcing for PI-1850s decade by decade since 1900.

Global Earth		**Tropospheric** [Wm^{-2}]			**Stratospheric** [Wm^{-2}]			**Total** [Wm^{-2}]		
Decade	PS/PI	SW	LW	net	SW	LW	net	SW	LW	net
1900-1909	CMAM/CMAM	0.009	0.025	0.034	-0.010	0.012	0.001	-0.001	0.036	0.035
1910-1919	CMAM/CMAM	0.013	0.037	0.050	-0.005	0.000	-0.005	0.009	0.037	0.046
1920-1929	CMAM/CMAM	0.016	0.048	0.064	-0.005	-0.000	-0.006	0.011	0.048	0.059
1930-1939	CMAM/CMAM	0.021	0.061	0.082	-0.009	0.004	-0.005	0.012	0.065	0.077
1940-1949	CMAM/CMAM	0.024	0.070	0.095	-0.013	0.009	-0.004	0.011	0.080	0.091
1950-1959	CMAM/CMAM	0.036	0.102	0.138	-0.023	0.026	0.003	0.013	0.128	0.142
1960-1969	CMAM/CMAM	0.050	0.138	0.189	-0.024	0.034	0.010	0.026	0.172	0.198
1970-1979	CMAM/CMAM	0.060	0.171	0.231	-0.099	0.006	0.005	0.059	0.177	0.236
1980-1989	CMAM/CMAM	0.068	0.197	0.265	0.024	-0.030	-0.006	0.092	0.167	0.259
1990-1999	CMAM/CMAM	0.068	0.208	0.276	0.074	-0.103	-0.029	0.142	0.106	0.247
2000-2009	CMAM/CMAM	0.071	0.218	0.290	0.084	-0.122	-0.037	0.156	0.096	0.252

Northern Hemisphere		**Tropospheric** [Wm^{-2}]			**Stratospheric** [Wm^{-2}]			**Total** [Wm^{-2}]		
Decade	PS/PI	SW	LW	net	SW	LW	net	SW	LW	net
1900-1909	CMAM/CMAM	0.011	0.030	0.041	-0.009	0.010	0.001	0.002	0.040	0.042
1910-1919	CMAM/CMAM	0.012	0.036	0.048	-0.003	-0.006	-0.009	0.009	0.030	0.039
1920-1929	CMAM/CMAM	0.019	0.055	0.074	-0.004	0.001	-0.003	0.015	0.056	0.071
1930-1939	CMAM/CMAM	0.024	0.070	0.094	-0.009	0.007	-0.002	0.015	0.077	0.092
1940-1949	CMAM/CMAM	0.031	0.089	0.120	-0.013	0.017	0.003	0.018	0.105	0.124
1950-1959	CMAM/CMAM	0.043	0.122	0.166	-0.024	0.033	0.008	0.019	0.155	0.174
1960-1969	CMAM/CMAM	0.057	0.159	0.216	-0.022	0.036	0.014	0.035	0.196	0.231
1970-1979	CMAM/CMAM	0.078	0.216	0.294	-0.006	0.029	0.023	0.072	0.245	0.317
1980-1989	CMAM/CMAM	0.092	0.257	0.349	0.012	0.009	0.022	0.105	0.266	0.371
1990-1999	CMAM/CMAM	0.101	0.284	0.385	0.051	-0.030	0.022	0.153	0.254	0.407
2000-2009	CMAM/CMAM	0.108	0.302	0.409	0.059	-0.040	0.020	0.167	0.262	0.429

Southern Hemisphere		**Tropospheric** [Wm^{-2}]			**Stratospheric** [Wm^{-2}]			**Total** [Wm^{-2}]		
Decade	PS/PI	SW	LW	net	SW	LW	net	SW	LW	net
1900-1909	CMAM/CMAM	0.007	0.02	0.028	-0.012	0.014	0.002	-0.005	0.034	0.029
1910-1919	CMAM/CMAM	0.014	0.037	0.051	-0.006	0.005	-0.001	0.009	0.042	0.051
1920-1929	CMAM/CMAM	0.014	0.041	0.055	-0.006	-0.002	-0.007	0.008	0.040	0.048
1930-1939	CMAM/CMAM	0.018	0.052	0.070	-0.008	-0.000	-0.008	0.010	0.052	0.062
1940-1949	CMAM/CMAM	0.017	0.051	0.067	-0.011	-0.001	-0.012	0.006	0.050	0.056
1950-1959	CMAM/CMAM	0.028	0.082	0.110	-0.018	0.016	-0.002	0.010	0.099	0.109
1960-1969	CMAM/CMAM	0.042	0.115	0.158	-0.019	0.023	0.004	0.024	0.138	0.162
1970-1979	CMAM/CMAM	0.040	0.125	0.166	0.011	-0.022	-0.011	0.051	0.103	0.154
1980-1989	CMAM/CMAM	0.041	0.136	0.178	0.041	-0.074	-0.033	0.083	0.063	0.145
1990-1999	CMAM/CMAM	0.034	0.133	0.167	0.096	-0.174	-0.078	0.130	-0.041	0.099
2000-2009	CMAM/CMAM	0.034	0.135	0.169	0.110	-0.203	-0.093	0.145	-0.068	0.076

A Bibliography

Bibliography

[1] M. Coldewey-Egbers, D. G. Loyola, M. Koukouli, D. Balis, J.-C. Lambert, T. Verhoelst, J. Granville, M. van Roozendael, C. Lerot, R. Spurr, S. M. Frith, and C. Zehner. The GOME-type total ozone essential climate variable (GTO-ECV) data record from the ESA climate change initiative. *Atmospheric Measurement Techniques*, 8(9):3923–3940, sep 2015.

[2] Edwards, J., and A. Slingo (1996), Studies with a flexible new radiation code. i: Choosing a configuration for a large-scale model, *Quarterly Journal of the Royal Meteorological Society*, *122*(531), 689–719, https://doi.org/10.1002/qj.49712253107.

[3] Checa-Garcia, R (2017), Pyozone: set of tools, *Zenodo*, https://doi.org/10.5281/zenodo.1118950.

[4] M. I. Hegglin, D. A. Plummer, T. G. Shepherd, J. F. Scinocca, J. Anderson, L. Froidevaux, B. Funke, D. Hurst, A. Rozanov, J. Urban, T. von Clarmann, K. A. Walker, H. J. Wang, S. Tegtmeier, and K. Weigel. Vertical structure of stratospheric water vapour trends derived from merged satellite data. *Nature Geoscience*, 7(10):768–776, aug 2014.

[5] Forster, P. M., R. S. Freckleton, and K. P. Shine (1997), On aspects of the concept of radiative forcing, *Climate Dynamics*, *13*(7), 547–560, https://doi.org/10.1007/s003820050182.

[6] B. C. O'Neill, C. Tevaldi, D. P. van Vuuren, V. Eyring, P. Friedlingstein, G. Hurtt, R. Knutti, E. Kriegler, J. F. Lamarque, J. Lowe, G. A. Meehl, R. Moss, K. Riahi, and B. M. Sanderson The Scenario Model Intercomparison Project (ScenarioMIP) for CMIP6 *Geosci. Model Dev.*,, 9, 3461âĂŞ3482, 2016 https://doi.org/10.5194/gmd-9-3461-2016.

[7] Morgenstern, O., M. I. Hegglin, E. Rozanov, F. M. O'Connor, N. L. Abraham, H. Akiyoshi, A. T. Archibald, S. Bekki, N. Butchart, M. P. Chipperfield, M. Deushi, S. S. Dhomse, R. R. Garcia, S. C. Hardiman, L. W. Horowitz, P. Jöckel, B. Josse, D. Kinnison, M. Lin, E. Mancini, M. E. Manyin, M. Marchand, V. Marécal, M. Michou, L. D. Oman, G. Pitari, D. A. Plummer, L. E. Revell, D. Saint-Martin, R. Schofield, A. Stenke, K. Stone, K. Sudo, T. Y. Tanaka, S. Tilmes, Y. Yamashita, K. Yoshida, and G. Zeng (2017), Review of the global models used within phase 1 of the Chemistry-Climate Model Initiative (CCMI), *Geoscientific Model Development*, *10*, 639–671, https://doi.org/10.5194/gmd-10-639-2017.

[8] R. J. van der A, M. A. F. Allaart, and H. J. Eskes. Multi sensor reanalysis of total ozone. *Atmospheric Chemistry and Physics*, 10(22):11277–11294, November 2010.

Part II.

Code released in support of the CMIP6 ozone dataset

Overview

The codes here described were created progressively since the spring of 2016. In this document are shown two Python modules named calculate.py and aux.py, where specific functions to create total and partial ozone column products, as well as, time-series data-analysis are included. Other functions are useful for harmonization of climate datasets and used on inital validation processes.

The code itself has a short description inside each function, please don't hesitate to contact with the author R.Checa-Garcia r.checagarcia at gmail.com if more information about any specific part of the code is needed. You can cite and download the last version of these codes at `https://doi.org/10.5281/zenodo.1118950`.

B calculate.py , module with functions to perform specific calculations

```
# (C) R. Checa-Garcia, Department of Meteorology, University of Reading.
#     email: r.checa-garcia@reading.ac.uk
#
# This file is part of a set of Python Modules to process and
# perform data-analysis of the CMIP6 ozone dataset. Some of the
# methods included here, were developed for SMURPHS project.
#
# This software is licensed with GPLv3. Please see <http://www.gnu.org/licenses/>.
"""
Python Module calculate.py

Purpose ---------------------------------------------------------------------
   calculate.py :
      module created to perform calculations on the support of CMIP6 ozone
      database products.

Author ----------------------------------------------------------------------
   R. Checa-Garcia, Department of Meteorology, University of Reading.
   email: r.checa-garcia@reading.ac.uk

CODE INFO -------------------------------------------------------------------

__author__       = "R. Checa-Garcia"
__organization__ = ["University of Reading"]
__license__      = "GPLv3"
__version__      = "First: 0.7 - April 2016, Current: July 2017"
__maintainer__   = "R. Checa-Garcia"
__project__      = "SMURPHS and CMIP6 ozone database"
__email__        = "r.checa-garcia@reading.ac.uk"
__status__       = "Consolidating"
-----------------------------------------------------------------------------

"""

# Load Libraries and Modules EXTERNAL
# -- note that not all these modules are actually needed by calculate.py

from optparse   import OptionParser     # Introduce options to code
from netCDF4    import Dataset          # netcdf4
from netCDF4    import date2num
from netcdftime import utime            # UTIME operations (local module)
from datetime   import datetime
from datetime   import timedelta

from scipy.interpolate import UnivariateSpline, interp1d
from tqdm import tnrange,  tqdm, tqdm_notebook
from os.path import isfile as exists_file

import numpy as np                     # numerical python
import os                              # Operating system operations
import glob                            # Find files
import pprint                          # Print arrays nice
import warnings                        # Manage warnings
import gc
import resource
import aux                             # This is a module of this software suite
import copy

# Functions --------------------------------------------------------------------
#

def pure_weighted_mean(val, ome):
    """
    Return the weighted average and standard deviation.
    values, weights -- Numpy arrays with the same shape.
    Also the function filters nan and gives as output the
    filtered arrays.

    :param val: for a given time, array mean values to merge
    :type val: numpy array with mean values
    :param ome: for a given time, array weights to merge
    :type ome: numpy array with weights for weigthing average
    :return mean, std, val_ok, ome_ok
    """
    val[np.argwhere(np.isnan(ome))] = np.nan
    ome[np.argwhere(np.isnan(val))] = np.nan

    val_ok = val[np.isfinite(val)]
    ome_ok = ome[np.isfinite(ome)]

    mean = np.sum(val_ok*ome_ok)/np.sum(ome_ok)
    std = 1.0/np.sqrt(np.sum(ome_ok))

    return mean, std, val_ok, ome_ok

```

```python
def weigthed_mean(arr_ubi, arr_err):
    """
    Given the array of unbiased values and
    the array with the expected errors (std)
    calculates the merged values and merged
    error.

    NOTE: arr_ubi and arr_err are bi-dimensional arrays,
    the number of platforms to merge and the second is
    the values of each platform to each time:

    [[v(p1,t1),v(p2,t1),...,v(pk,t1)],
     [v(p1,t2),v(p2,t2),...,v(pk,t2)],...

    therefore the loops inside are a loop in the values of
    the different platforms for a given time.

    :param arr_ubi: list of arrays of unbiased time values
    :param arr_err: list of arrays of each platform error.
    :return:  merge_val, merge_err
    """

    omegas = 1.0/np.power(arr_err, 2.0)
    merge_val = []
    merge_err = []
    ii = 0
    for ave, omega in zip(arr_ubi, omegas):
        val, err, val_ok, ome_ok = pure_weighted_mean(ave, omega)
        merge_val.append(val)
        merge_err.append(err)
        ii += 1

    return merge_val, merge_err


def mad_based_outlier(points, thresh=3.5):
    """
    This function is testing an array of values
    contained in points and try to identify outliers
    given a thershold (by default 3.5).


    :param points:
    :param thresh:
    :return: bolean array with same dimensions than points.
    """

    diff = np.zeros_like(points)
    modified_z_score = np.zeros_like(points)
    median = np.nanmedian(points)

    for ival, val in enumerate(points):
        if np.isnan(val) is True:
            diff[ival] = np.nan
        else:
            diff[ival] = np.sqrt((val - median)**2)

    med_abs_deviation = np.nanmedian(diff)

    for ival, val in enumerate(points):
        if np.isnan(val) is True:
            modified_z_score[ival] = True
        else:
            modified_z_score[ival] = 0.6745 * diff[ival] / med_abs_deviation

    return modified_z_score > thresh


def smooth_tropopause(lat):
    """
    Defines a reasonable tropopause consistent with Hansen et al 2005, but
    full symmetrical NH vs SH. The units are in hPa. Provides the pressure
    height of the tropopause for a given value of the latitude. For more,
    information about this tropopause definition check the supplementary
    information of the paper "Historical tropospheric and stratospheric
    ozone radiative forcing using the CMIP6 database", Checa-Garcia et al.

    :param lat:
    :return: tropopause
    """


    center  = 45.0
    smooth  = 10.0
    voffset = 165.0
    vwidth  = 55.0

    tropopause = np.tanh((abs(lat)-center)/smooth)*vwidth+voffset

    return tropopause


def zonal_regrid(plev_0, lat_0, lat_1, zonal_map):
    """
    This function re-grid from lat_0 to lat_1 the original zonal_map.
    Zonal map is a map with dimensions [time, plev_0, lat_0] and the
    output will be the same array at [time, plev_0, lat_1]

    :param plev_0:
    :param lat_0:
    :param lat_1:
```

```
    :param zonal_map:
    :return: new_zonal_map
    """

    from scipy import interpolate

    len_time = zonal_map.shape[0]
    new_zonal_map = np.zeros((len_time, len(plev_0), len(lat_1)))

    for itime in range(len_time):
        for ipres in range(len(plev_0)):
            if False in np.isfinite(zonal_map[itime, ipres, :]):
                print('PROBLEM')
                exit()

            interfunc = interpolate.interp1d(lat_0, zonal_map[itime, ipres, :],
                                             fill_value='extrapolate', kind='nearest')
            # A potential problem that actually happens that than for some negative
            # latitudes the values can be negative. This is more likely to happen with
            # a linear interpolation so a nearest neighbour is included.

            negatives = np.argwhere(interfunc(lat_1) < 0)
            if len(negatives) > 0:
                print(itime, ipres, negatives)
            new_zonal = interfunc(lat_1)
            new_zonal[negatives] = 0.0
            new_zonal_map[itime, ipres, :] = new_zonal

    return new_zonal_map


def new_plevs_grid(field, plev_old, plev_new, keep=False, extend=True):
    """
    This function re-grid a field only in the vertical level which is
    supposed to be on the axis=1. This function is created to minimize
    the dependence with external libraries (only depends on scipy) but
    users with cf-python or iris python modules may be good alternatives.
    Other is the use of cdo/nco.

    :param field:
    :param plev_old:
    :param plev_new:
    :param extrapolate:
    :param keep:
    :return: new_field
    """

    sizes = field.shape

    new_sizes = (sizes[0], len(plev_new), sizes[2], sizes[3])
    new_field = np.zeros(new_sizes)
    for idim in tqdm(range(sizes[0]), desc='Lev DIM'):
        gc.collect()
        for jdim in range(sizes[2]):
            for kdim in range(sizes[3]):
                if extend==True:
                    extrapolator = interp1d(plev_old, field[idim, :, jdim, kdim],
                                            kind='linear', bounds_error=False,
                                            fill_value=(field[idim, 0, jdim, kdim],
                                                        field[idim, -1, jdim, kdim]))
                else:
                    extrapolator = interp1d(plev_old, field[idim, :, jdim, kdim],
                                            kind='linear', bounds_error=False,
                                            fill_value=(field[idim, 0, jdim, kdim], 0.0))

                new_field[idim, :, jdim, kdim] = extrapolator(plev_new)

    return new_field


def new_lons_grid(field, lons_old, lons_new, keep=False):
    """
    This function re-grid a field only in the longitude which is
    supposed to be on the axis=-1. This function is created to minimize
    the dependence with external libraries (only depends on scipy) but
    users with cf-python or iris python modules may be good alternatives.
    Other is the use of cdo/nco.

    This function kept commented the memory analysis calls in case the
    user find them useful.

    :param field:
    :param plev_old:
    :param plev_new:
    :param extrapolate:
    :param keep:
    :return: new_field
    """

    # print('Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)

    sizes = field.shape
    if len(sizes)>3:
        new_sizes = (sizes[0], sizes[1], sizes[2], len(lons_new))
        #print('Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)

        print('...Loops LONS:',new_sizes)
        #gc.collect()
        #print('Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)
```

```python
        new_field = np.zeros(new_sizes)
        for idim in tqdm(range(sizes[0]), desc='Lon DIM'):
            gc.collect()
            for jdim in range(sizes[1]):
                for kdim in range(sizes[2]):
                    #extrapolator = UnivariateSpline(lons_old, field[idim, jdim, kdim, :], k=extrapolate)
                    extrapolator = interp1d(lons_old, field[idim, jdim, kdim, :],
                                            kind='linear', fill_value='extrapolate')
                    new_field[idim, jdim, kdim, :] = extrapolator(lons_new)

        if keep == True:
            new_field[:, :, :, 0:len(lons_old)] = field[:,:,:,:]

    if len(sizes)==3:
        new_sizes = (sizes[0], sizes[1], len(lons_new))
        #print('Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)
        #print('...Loops:',new_sizes)
        #gc.collect()
       # print('Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)


        new_field = np.zeros(new_sizes)
        for idim in tqdm(range(sizes[0])):
            gc.collect()
            for jdim in range(sizes[1]):
                    #extrapolator = UnivariateSpline(lons_old, field[idim, jdim, kdim, :], k=extrapolate)
                    extrapolator = interp1d(lons_old, field[idim, jdim, :],
                                            kind='linear', fill_value='extrapolate')
                    new_field[idim, jdim, :] = extrapolator(lons_new)


    return new_field


def new_lats_grid(field, lats_old, lats_new, keep=False):
    """
    This function re-grid a field only in the latitude which is
    supposed to be on the axis=2. This function is created to minimize
    the dependence with external libraries (only depends on scipy) but
    users with cf-python or iris python modules may be good alternatives.
    Other is the use of cdo/nco.


    :param field:
    :param plev_old:
    :param plev_new:
    :param extrapolate:
    :param keep:
    :return: new_field
    """

    sizes = field.shape
    if len(sizes)>3:
        new_sizes = (sizes[0], sizes[1], len(lats_new), sizes[3])

        print('...Loops LATS:',new_sizes)
        new_field = np.zeros(new_sizes)
        for idim in tqdm(range(sizes[0]), desc='Lat DIM'):
            gc.collect()
            for jdim in range(sizes[1]):
                for kdim in range(sizes[3]):
                    extrapolator =  interp1d(lats_old, field[idim, jdim, :, kdim],
                                             kind='linear', fill_value='extrapolate')
                    new_field[idim, jdim, :, kdim] = extrapolator(lats_new)

    if len(sizes)==3:
        new_sizes = (sizes[0], len(lats_new), sizes[2])

        print('...Loops:',new_sizes)
        new_field = np.zeros(new_sizes)
        for idim in tqdm(range(sizes[0])):
            gc.collect()
            for kdim in range(sizes[2]):
                    extrapolator =  interp1d(lats_old, field[idim, :, kdim],
                                             kind='linear', fill_value='extrapolate')
                    new_field[idim, :, kdim] = extrapolator(lats_new)

    return new_field



def hybrid_to_uniform_press_plev_model(model_tim, model_plev, model_lon, model_lat,
                                       model_var, plevs, extend=True, keep=False):
    """
    This function transform a variable given with hybrid coordinates to a simple
    fixed pressure level vertical grid.

    :param model_tim:
    :param model_plev:
    :param model_lon:
    :param model_lat:
    :param model_var:
    :param plevs:
    :param extend:
    :param keep:
    :return: new_field
    """

    print('Main Memory usage: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)

    new_sizes = (len(model_tim), len(plevs), len(model_lat), len(model_lon))
```

```python
    new_field = np.zeros(new_sizes)
    for ndim in tqdm(range(new_sizes[0]), ncols=80):
        for jdim in range(new_sizes[2]):
            for idim in range(new_sizes[3]):
                if extend==True:
                    fill_val = np.array([model_var[ndim, 0, jdim, idim], model_var[ndim, -1, jdim, idim]])
                    if False in np.isfinite(fill_val):
                        exit()

                    extrapolator = interp1d(model_plev[ndim, :, jdim, idim],
                                            model_var[ndim, :, jdim, idim],
                                            kind='linear', bounds_error=False,
                                            fill_value=(model_var[ndim, 0, jdim, idim],
                                            model_var[ndim, -1, jdim, idim]))
                else:
                    extrapolator = interp1d(model_plev[ndim, :, jdim, idim],
                                            model_var[ndim, :, jdim, idim],
                                            kind='linear', bounds_error=False,
                                            fill_value=(model_var[ndim, 0, jdim, idim], 0.0))

                new_field[ndim, :, jdim, idim] = extrapolator(plevs)

    return new_field


def input_hybrid_netcdf(file_name, varname):
    """
    Reads a netcdf file with hybrid coordinates.

    :param file_name:
    :param varname:
    :return: model_tim, model_plev, model_lon, model_lat,
             model_var, model_ps
    """


    nc_model = Dataset(file_name, mode='r')

    model_var = nc_model[varname][:]
    model_lon = nc_model['lon'][:]
    model_lat = nc_model['lat'][:]
    model_p0  = nc_model['P0'][:]
    model_ps  = nc_model['PS'][:]
    model_hyam= nc_model['hyam'][:]
    model_hybm= nc_model['hybm'][:]
    model_lev = nc_model['lev'][:]
    model_tim = nc_model['time'][:]

    model_plev = np.zeros_like(model_var)
    dim_model_plev = model_plev.shape
    for nval in tqdm(range(dim_model_plev[0]), ncols=80):
        for kval in range(dim_model_plev[1]):
            model_plev[nval,kval,:,:] =model_hyam[kval]*model_p0+model_hybm[kval]*model_ps[nval,:,:]

    return model_tim, model_plev, model_lon, model_lat, model_var, model_ps



def vertical_mean(vmr_A, vmr_B, plevs, goal='avg-fast'):
    """
    This subroutine estimate a kind of mean between vmr_A and vmr_B where wmr_A has a
    higher level of confidence in the trosphere. So the steps are:
    1. Identify the tropopause -> index in plevs -> i_tropo
    2. From surface to i_tropo we create a weighting function that gives 0.85 to the model
       in the surface and 0.50 in i_tropo. Between both a continuous monotonic decreasing
       function of the index is created and applied.
    3. From the tropopause ahead the weight is 0.5 for both models.

    This subroutine was created to test the vertical mean done to create the
    CMIP6 ozone dataset but also to test alternatives to be applied for the creation of
    SMURPHS datasets.
    :param vmr_A:
    :param vmr_B:
    :param plevs:
    :param goal:
    :return: new_mean
    """

    i_trop = np.argmin(np.abs(plevs - 15000.))

    w_A = np.array([0.70+i_plev*(-0.20/float(i_trop)) for i_plev in range(len(plevs))])
    w_A[i_trop::] = 0.5
    w_B = 1.0 - w_A
    new_size = vmr_A.shape
    new_mean = np.zeros_like(vmr_A)

    if goal=='avg':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for jd in range(new_size[2]):
                for id in range(new_size[3]):
                    for kd in range(new_size[1]):
                        new_mean[nd, kd, jd, id] = w_A[kd]*vmr_A[nd, kd, jd, id] + \
                                                   w_B[kd]*vmr_B[nd, kd, jd, id]
    if goal=='avg-fast':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for kd in range(new_size[1]):
                new_mean[nd, kd, :, :] = w_A[kd]*vmr_A[nd, kd, :, :] + \
                                         w_B[kd]*vmr_B[nd, kd, :, :]
    if goal=='err':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for jd in range(new_size[2]):
```

```
                for ld in range(new_size[3]):
                    for kd in range(new_size[1]):
                        new_mean[nd, kd, jd, ld] = np.abs(w_A[kd]*vmr_A[nd, kd, jd, ld]-\
                                                   w_B[kd]*vmr_B[nd, kd, jd, ld])*0.5
    return new_mean


def vertical_mean_cmip6(vmr_A, vmr_B, plevs, goal='avg-fast'):
    """
    This subroutine estimate a kind of mean between vmr_A and vmr_B where wmr_A has a
    higher level of confidence in the troposphere. So the steps are:
    1. Identify the tropopause -> index in plevs -> i_tropo
    2. From surface to i_tropo we create a weighting function that gives 0.85 to the model
       in the surface and 0.50 in i_tropo. Between both a continuous monotonic decreasing
       function of the index is created and applied.
    3. From the tropopause ahead the weight is 0.5 for both models.

    This subroutine was created to test the vertical mean done to create the
    CMIP6 ozone dataset but also to test alternatives to be applied for the creation of
    SMURPHS datasets.

    :param vmr_A:
    :param vmr_B:
    :param plevs:
    :param goal:
    :return:
    """

    i_trop = np.argmin(np.abs(plevs - 15000.))

    i1 = 21

    w_A = np.array([0.40+i_plev*(0.10/float(i1)) for i_plev in range(len(plevs))])
    w_A[i1:i1+18] = 0.5
    w_A_alpha = (0.9-0.5)/float(len(plevs)-39)
    w_A_beta  = 0.5-w_A_alpha*float(21+18)
    w_A[i1+18::] = np.array([(w_A_beta+i_plev*w_A_alpha) for i_plev in np.arange(i1+18,len(plevs))])

    w_B = 1.0 - w_A


    new_size = vmr_A.shape
    new_mean = np.zeros_like(vmr_A)
    if goal=='avg':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for jd in range(new_size[2]):
                for id in range(new_size[3]):
                    for kd in range(new_size[1]):
                        new_mean[nd, kd, jd, id] = w_A[kd]*vmr_A[nd, kd, jd, id] + \
                                                   w_B[kd]*vmr_B[nd, kd, jd, id]

    if goal=='avg-fast':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for kd in range(new_size[1]):
                new_mean[nd, kd, :, :] = w_A[kd]*vmr_A[nd, kd, :, :] + \
                                         w_B[kd]*vmr_B[nd, kd, :, :]
    if goal=='err':
        for nd in tqdm(range(new_size[0]), ncols=80, desc='1st DIM'):
            for jd in range(new_size[2]):
                for ld in range(new_size[3]):
                    for kd in range(new_size[1]):
                        new_mean[nd, kd, jd, ld] = np.abs(w_A[kd]*vmr_A[nd, kd, jd, ld]-\
                                                   w_B[kd]*vmr_B[nd, kd, jd, ld])*0.5
    return new_mean


def create_seasonal_zonal_du(model_base, var_name='vmro3', add_du=True,
                             add_seasonal=False, add_zonal=True):
    """
    This function add a seasonal and zonal estimation of the variable.

    In general it relies on the assumption that the first month of the
    dataset is January and the netcdf has a monthly resolution.

    model_base is a file-name
    :param model_base:
    :param var_name:
    :param add_du:
    :param add_seasonal:
    :param add_zonal:
    :return:
    """


    nc_base = Dataset(model_base, mode='a')

    nc_lev = nc_base['plev'][:]
    nc_lon = nc_base['lon'][:]
    nc_lat = nc_base['lat'][:]
    nc_tim = nc_base['time'][:]

    dat_time = aux.parse_datetime(nc_base)

    nc_var = nc_base[var_name][:]
    pr_surf = nc_base['ps'][:]

    var_sizes = nc_var.shape

    seasonal = np.zeros((12,var_sizes[1],var_sizes[2],var_sizes[3]))

    if add_seasonal:
```

```
        print('Adding seasonal mean')
        for imonth in range(12):
            i_count = 0
            for itime in np.arange(imonth,var_sizes[0],12):
                seasonal[imonth,:,:,:] = seasonal[imonth,:,:,:]+nc_var[itime,:,:,:]
                i_count = i_count + 1
            seasonal[imonth,:,:,:] = seasonal[imonth,:,:,:]/i_count

        nc_base.createDimension('month', 12)
        months = nc_base.createVariable('month', 'i4', ('month',))
        months[:] = [1,2,3,4,5,6,7,8,9,10,11,12]
        vmr = nc_base.createVariable(var_name+'_seasonal', 'f8',
                                     ('month', 'plev', 'lat', 'lon'),
                                     fill_value=-999.9)
        vmr.units = 'vmr'
        vmr.standard_name = 'Volume Mixing Ratio'
        vmr[:] = seasonal
        vmr.warning = 'Seasonal field estimated for whole netcdf file'

    if add_zonal:
        print('Adding zonal mean')
        zonal_model = np.nanmean(nc_var, axis=3)  # zonal mean, mean in longitude
        vmr_zonal = nc_base.createVariable(var_name+'_zonal', 'f8',
                                           ('time', 'plev', 'lat'),
                                           fill_value=-999.9)
        vmr_zonal[:] = zonal_model
        vmr_zonal.warning = 'Zonal field estimated for whole netcdf file'

    #try:
    #    nc_err = nc_base[var_name+'_err'][:]
    #    zonal_error = np.nanmean(nc_err, axis=3)  # zonal mean, mean in longitude
    #    vmr_zonal_err = nc_base.createVariable(var_name+'_err_zonal', 'f4',
    #                                   ('time', 'plev', 'lat'),
    #                                   fill_value=-999.9)
    #    vmr_zonal_err[:] = zonal_error
    #except:
    #    print('Zonal uncer. not added, input file did not have uncertainty...')

    if add_du:
        print('Adding Partial Column Field DU')

        partial_DU = nc_base.createVariable('O3_partialcolumn', 'f8',
                                 ('time', 'plev', 'lat', 'lon'),
                                 fill_value=-999.9)
        partial_DU.units = 'DU'
        o3_DU = calculate_partial_column(nc_lat, nc_lon, nc_lev, nc_var,
                                         pr_surf, nc_tim, case='TOTAL')
        partial_DU.standard_name = 'Column concentration in DU'
        partial_DU[:] = o3_DU
        partial_DU.warning = 'Partial column from surface until plev'

    nc_base.close()

    return


def add_anomalies_field(model_base, var_name='vmro3'):
    """
    This function allows add anomalies to an specific seasonal dataset.

    :param model_base:
    :param var_name:
    :return:
    """

    nc_base = Dataset(model_base, mode='a')

    nc_lev = nc_base['plev'][:]
    nc_lon = nc_base['lon'][:]
    nc_lat = nc_base['lat'][:]
    nc_tim = nc_base['time'][:]
    nc_mon = nc_base['months'][:]

    dat_time = parse_datetime(nc_base)

    nc_var = nc_base[var_name][:]
    seasonal = nc_base[var_name+'_seasonal'][:]

    var_sizes = nc_var.shape
    anomaly=np.zeros_like(nc_var)
    for itime in np.arange(var_sizes[0]):
        # we suppose that first time is January
        imonth = (itime + 1)%12 - 1
        if imonth == -1:
            imonth == 11
        anomaly[itime,:,:,:] = nc_var[itime,:,:,:] - seasonal[imonth, :,:,:]

    vmr = nc_base.createVariable(var_name+'_anomaly', 'f8',
                                 ('time', 'plev', 'lat', 'lon'),
                                 fill_value=-999.9, zlib=True)
    vmr.units = 'vmr'
    vmr.standard_name = 'Volume Mixing Ratio'
    vmr[:] = anomaly
    vmr.warning = 'Differ. between monthly field and seasonal value on each month'

    zonal_model = np.nanmean(anomaly, axis=3)  # zonal mean, mean in longitude
    vmr_zonal = nc_base.createVariable(var_name+'_anomalyzonal', 'f8',
                                       ('time', 'plev', 'lat'),
                                       fill_value=-999.9, zlib=True)
    vmr_zonal[:] = zonal_model
    vmr_zonal.warning = 'Estimated as the zonal mean of the anomaly field'
```

```
    nc_base.close()

    return


def calculate_partial_column_nc(namedata, case='TROPOS'):
    """
    This function is designed to calculate the partial column of ozone in DU.
    The idea is transform the o3_vmr field to an partial column so

    o3_vmr(lat, lon, lev) => o3_DU(lat, lon, lev) and it is the partial column
                               until level lev.

    :param namedata:
    :param case:
    :return:
    """

    dataset_o3 = Dataset(namedata)
    alats  = dataset_o3.variables['lat'][:]
    alons  = dataset_o3.variables['lon'][:]
    # Pressure level variable were renamed for RF calculations, the
    # original dataset has the name plev so it should be quite direct
    # use plev instead of pr_lay with appropiate changes.

    ps_surf = dataset_o3.variables['ps'][:]
    times   = dataset_o3.variables['time'][:]

    pr_lev = dataset_o3.variables['plev'][:] # begins at 1000 hPa until TOA
    ao3_mmr  = dataset_o3.variables['vmro3'][:][:,:,:]  #(plev, lat, lon)

    dataset_o3.close()

    ao3_DU = calculate_partial_column(alats, alons, pr_lev,
                                  ao3_mmr, ps_surf, times, case='TOTAL')

    return a3_DU


def calculate_partial_column(alats, alons, p_lev, ao3_mmr, ps_surf, times,
                             case='TOTAL', loopin=False):
    """
    This function is designed to calculate the partial column of ozone in DU.
    The idea is transform the o3_vmr field to an partial column so

    o3_vmr(lat, lon, lev) => o3_DU(lat, lon, lev) and it is the partial column
                               until level lev.

    :param alats:
    :param alons:
    :param p_lev:
    :param ao3_mmr:
    :param ps_surf:
    :param times:
    :param case:
    :param loopin:
    :return:
    """

    g0 = 9.80665
    T0 = 273.15
    p0 = 101325.
    R = 287.3

    factor = 10.0*R*T0*0.5/(g0*p0)

    kg_to_g      = 1.0e+03
    ppmv_to_ppv  = 1.0e-06
    mw_o3        = 47.9982
    mw_dryair    = 28.9648

    # 0.01 to calculate in hPa, 1e6*mw_dryair/mw_o3 to change mmr to ppmv
    f_mmr_to_vmr = 0.01*1.e6*mw_dryair/mw_o3

    f_units = 0.01*1.e6

    n_lev   = len(p_lev)

    ao3_DU = np.zeros_like(ao3_mmr)
    acc_DU = np.zeros_like(ao3_mmr[:,0,:,:])
    delta_pss = np.zeros_like(times)

    if loopin:
        for ilev in tqdm(range(n_lev-1), ncols=80, desc='2nd DIM'):
            for ilat in range(len(alats)):
                lati = alats[ilat]
                for ilon in range(len(alons)):
                    delta_mmr = ao3_mmr[:,ilev,ilat,ilon]+ao3_mmr[:,ilev+1,ilat,ilon]
                    delta_pss[:] = p_lev[ilev+1]-p_lev[ilev]


                    for itim,tm in enumerate(times):
                        if p_lev[ilev+1] < ps_surf[itim, ilat, ilon]:
                            delta_pss[itim] = 0.0
                        if p_lev[ilev] > ps_surf[itim,ilat, ilon] > p_lev[ilev+1]:
                            delta_pss[itim] = ps_surf[itim, ilat, ilon]-p_lev[ilev+1]

                    acc_DU[:,ilat,ilon] = acc_DU[:,ilat,ilon]+f_units*delta_mmr*delta_pss[:]
                    #if case=='TROPOS':
                    #    if (p_lev[ilev] >  100.0*smooth_tropopause(lati)):
```

```python
                    #        acc_DU[:,ilat,ilon] = acc_DU[:,ilat,ilon]+f_mmr_to_vmr*delta_mmr*delta_pss[:]
                    #if case=='TOTAL':
                    #
                    #if case=='STRATO':
                    #    if (p_lev[ilev] <  100.0*smooth_tropopause(lati)):
                    #        acc_DU[:,ilat,ilon] = acc_DU[:,ilat,ilon]+f_mmr_to_vmr*delta_mmr*delta_pss[:]
                    #if case=='SURFACE':
                    #    if (p_lev[ilev] >  70000.0):
                    #        acc_DU[:,ilat,ilon] = acc_DU[:,ilat,ilon]+f_mmr_to_vmr*delta_mmr*delta_pss[:]
            ao3_DU[:,ilev+1,:,:] = factor*copy.deepcopy(acc_DU[:,:,:])

    else:
     for itim in tqdm(range(len(times)), ncols=80, desc='1st DIM'):
        for ilev in tqdm(range(n_lev-1), ncols=80, desc='2nd DIM'):
            for ilat in range(len(alats)):
                lati = alats[ilat]
                for ilon in range(len(alons)):
                    delta_mmr = ao3_mmr[itim,ilev,ilat,ilon]+ao3_mmr[itim,ilev+1,ilat,ilon]
                    delta_prs = p_lev[ilev]-p_lev[ilev+1]

                    if ilev<10:
                        if p_lev[ilev+1] < ps_surf[itim, ilat, ilon]:
                           delta_prs = 0.0
                        if p_lev[ilev] > ps_surf[itim,ilat, ilon] > p_lev[ilev+1]:
                           delta_prs = ps_surf[itim, ilat, ilon]-p_lev[ilev+1]

                    acc_DU[itim,ilat,ilon] = acc_DU[itim,ilat,ilon]+f_units*delta_mmr*delta_prs
            ao3_DU[itim,ilev+1,:,:] = factor*copy.deepcopy(acc_DU[itim,:,:])

    return ao3_DU


def mean_netcdf_cmam_waccm(file_cmam, file_waccm,
                           varname='vmro3', vertical_w=False, extend=True):
    """
    This function shows how to perform a mean of two model datasets given
    the functions present in this module.

    :param file_cmam:
    :param file_waccm:
    :param varname:
    :param vertical_w:
    :param extend:
    :return:
    """


    info_waccm = file_waccm.split('_')
    period = info_waccm[3]
    print('Merging to files: ', period)
    print('                  ', file_cmam)
    print('                  ', file_waccm)


    if extend==True:
        strextend = ''
    else:
        strextend = '_noextend'

    dir_out = '../OUTPUT/'
    if vertical_w == False:
        surname = 'vmro3_CMIP6_v1.0_py_'+period+'_monthly_standard_weights05.nc'
        model_mean = dir_out+surname
    if vertical_w==True:
        surname = 'vmro3_CMIP6_v1.0_py_'+period+'_monthly_standard_weightsA.nc'
        model_mean = dir_out+surname
    if vertical_w=='cmip6':
        surname = 'vmro3_CMIP6_v1.0_py_'+period+'_monthly_standard_weightsCMIP6.nc'
        model_mean = dir_out+surname

    nc_CMAM = Dataset(file_cmam, mode='r')
    nc_WACC = Dataset(file_waccm, mode='r')

    # Variables

    cmam_lev = nc_CMAM['plev'][:]
    wacc_lev = nc_WACC['plev'][:]

    cmam_lon = nc_CMAM['lon'][:]
    wacc_lon = nc_WACC['lon'][:]

    cmam_lat = nc_CMAM['lat'][:]
    wacc_lat = nc_WACC['lat'][:]

    cmam_tim = nc_CMAM['time'][:][:]
    wacc_tim = nc_WACC['time'][:][:]

    # These arrays should be identical for both files, we could test:

    if not np.allclose(cmam_lev, wacc_lev):
        print('Problem with levels')
        exit()
    if not np.allclose(cmam_lat, wacc_lat):
        print('Problem with lats')
        exit()
    if not np.allclose(cmam_lon, wacc_lon):
        print('Problem with lons')
        exit()
    wacc_ozo = nc_WACC[varname][:][:,:,:,:]
    cmam_ozo = nc_CMAM[varname][:][:,:,:,:]
```

```
    nc_mean = Dataset(model_mean, mode='w', format='NETCDF4')

    # We create the main dimensions
    nc_mean.createDimension('time', 0)
    nc_mean.createDimension('plev', len(cmam_lev))
    nc_mean.createDimension('lat',  len(cmam_lat))
    nc_mean.createDimension('lon', len(cmam_lon))
    nc_mean.createDimension('bnds', 2)

    time = nc_mean.createVariable('time',      'f8', ('time',))
    lats = nc_mean.createVariable('lat',  'f4', ('lat',))
    levs = nc_mean.createVariable('plev',      'f4', ('plev',))
    lons = nc_mean.createVariable('lon', 'f4', ('lon',))


    lons.units = 'degrees east'
    lons.standard_name = "longitude"
    lons.long_name = "longitude"

    lats.units = 'degrees north'
    lats.standard_name = "latitude"
    lats.long_name = "latitude"
    levs.units = 'Pa'

    time.units = 'days since 1850-01-01 00:00:00'
    time.calendar = 'standard'

    time[:] = wacc_tim
    levs[:] = cmam_lev
    lats[:] = cmam_lat
    lons[:] = cmam_lon

    vmr = nc_mean.createVariable('vmro3', 'f8',
                                 ('time', 'plev', 'lat', 'lon'),
                                 fill_value=-999.9)
    vmr.units = 'mole mole -1'
    vmr.standard_name = 'mole_fraction_of_ozone_in_air'

    pr_surf = nc_mean.createVariable('ps', 'f8',
                                 ('time', 'lat', 'lon'),
                                 fill_value=-999.9)
    pr_surf.units = 'Pa'
    pr_surf.standard_name = 'Surface Pressure'
    pr_surf.warning = 'Surface Pressure from cesm1-waccm'

    pr_surf = nc_WACC['ps'][:]

    print(    '========== vertical mean calculation ===')
    if vertical_w== True:
        print('        --- own method                  >')

        vmr[:] = vertical_mean(cmam_ozo, wacc_ozo, cmam_lev)
        vmr.warning = 'merging method Ramiro'

    if vertical_w== False:
        print('        --- typical mean                >')
        vmr[:] =(cmam_ozo+wacc_ozo)*0.5
        vmr.warning = 'merging method 0.5 each'
        print('----------------------------          ok')

    if vertical_w== 'cmip6':
        print('        --- cmip6 mean                  >')
        vmr[:] = vertical_mean_cmip6(cmam_ozo, wacc_ozo, cmam_lev)
        vmr.warning = 'merging method cmip6'


    nc_CMAM.close()
    nc_WACC.close()
    nc_mean.close()
    gc.collect()

    create_seasonal_zonal_du(model_mean, var_name='vmro3', add_du=True)
    gc.collect()

    return model_mean
```

C aux.py , auxiliary functions module

```python
# (C) R. Checa-Garcia, Department of Meteorology, University of Reading.
#     email: r.checa-garcia@reading.ac.uk
#
# This file is part of a set of Python Modules to process and
# perform data-analysis of the CMIP6 ozone dataset. Some of the methods here
# included were developed for SMURPHS project.
#
# This software is licensed with GPLv3. Please see <http://www.gnu.org/licenses/>.
"""
Python Module aux.py

Purpose -------------------------------------------------------------------
    aux.py :
        auxiliary module created in the support of CMIP6 and SMURPHS
        database products.

Author --------------------------------------------------------------------
   R. Checa-Garcia, Department of Meteorology, University of Reading.
   email: r.checa-garcia@reading.ac.uk

CODE INFO -----------------------------------------------------------------

__author__       = "R. Checa-Garcia"
__organization__ = ["University of Reading"]
__license__      = "GPLv3"
__version__      = "First: 0.7 - April 2016, Current: July 2017"
__maintainer__   = "R. Checa-Garcia"
__project__      = "SMURPHS and CMIP6 ozone database"
__email__        = "r.checa-garcia@reading.ac.uk"
__status__       = "Consolidating"
---------------------------------------------------------------------------
"""

import numpy as np
import warnings
import logs
import cProfile
from netCDF4    import Dataset        # netcdf4
import resource
from netcdftime import utime       # UTIME operations
import sys
from netCDF4    import date2num
from netCDF4    import num2date
from datetime   import datetime
from datetime   import timedelta

def save_netcdf(new_var, all_time, val_plevs, val_lat, val_lon, model_ccmi,
                var_ps='none', varname='vmro3',
                tim_units='months since 1850-01-01 00:00:00.0', calendar='standard'):
    """
    This function saves a netCDF file on pressure levels. It is used mainly for
    temporal netCDF files during processing.

    TESTED OK.
    """

    print('Main Memory use: %s (kb)' % resource.getrusage(resource.RUSAGE_SELF).ru_maxrss)
    print('.... saving to : %s ' % model_ccmi)


    nc_ccmi = Dataset(model_ccmi, mode='w', format='NETCDF4')

    medat = nc_ccmi.createGroup('METADATA')
    medat.references   = 'http://www.met.reading.ac.uk/'
    medat.creator_name = "Ramiro Checa-Garcia (supervised by M.I. Hegglin)"
    medat.creator_mail = "r.checa-garcia@reading.ac.uk"
    medat.comment      = ('Created with pyMERGE_model done a University of Reading')
    medat.Conventions  = 'CF-1.0'
    medat.pyMerge_version = 'September 2016'

    # We create the main dimensions
    nc_ccmi.createDimension('time', None)
    nc_ccmi.createDimension('plev', len(val_plevs))
    nc_ccmi.createDimension('lat', len(val_lat))
    nc_ccmi.createDimension('lon', len(val_lon))

    time = nc_ccmi.createVariable('time', 'f8', ('time',))
    levs = nc_ccmi.createVariable('plev', 'f4', ('plev',))
    lats = nc_ccmi.createVariable('lat', 'f4', ('lat',))
    lons = nc_ccmi.createVariable('lon', 'f4', ('lon',))

    lons.units = 'degrees east'
    lons.standard_name = "longitude"
    lats.units = 'degrees north'
    lats.standard_name = "latitude"
    levs.units = 'Pa'
    time.units = tim_units
    time.calendar = calendar
    time[:] = all_time
    levs[:] = val_plevs
```

```
    lats[:] = val_lat
    lons[:] = val_lon

    if varname == 'vmro3':
        vmr = nc_ccmi.createVariable('vmro3', 'f8',
                                    ('time', 'plev', 'lat', 'lon'),
                                     fill_value=-999.9)
        vmr.units = 'vmr'
        vmr.standard_name = 'Volume Mixing Ratio O3'

    if varname == 'vmrh2o':
        vmr = nc_ccmi.createVariable('vmrh2o', 'f8',
                                    ('time', 'plev', 'lat', 'lon'),
                                    fill_value=-999.9)
        vmr.units = 'vmr'
        vmr.standard_name = 'Volume Mixing Ratio H2O'

    if var_ps != 'none':
        surf_press = nc_ccmi.createVariable('ps', 'f8',
                                    ('time', 'lat', 'lon'),
                                     fill_value=-999.9)
        surf_press.units = 'Pa'
        surf_press.standard_name = 'Surface Pressure in Pa'
        surf_press[:] = var_ps

    vmr[:] = new_var

    nc_ccmi.close()

    return

def concatenate_ordered(lvalue, ltimes):
    """
    First is estimate the interval with common times:

         From (list of arrays might not be

         (note:time value is on axis x)

         (1 array)        ------
         (2 array) -------
         (3 array)              -------
         We created a single array where first is 2 array then 1 array and
         then 3 array
         (new arr) --------------------

         For the arrays:
         ----------
                -----------
                         ------------
         We create:

         -------
                XXX
                   ------
                         XX
                           -----------

         X arrays are a mean (at this moment this is not implemented.)

    :param lvalue: list of arrays to concatenate
    :param ltimes: list of times of each array
    :return:
    """

    lminval = [np.min(a) for a in ltimes]
    lsorted = [i[0] for i in sorted(enumerate(lminval), key=lambda x: x[1])]

    l_ord_times = [ltimes[i] for i in lsorted]
    l_ord_value = [lvalue[i] for i in lsorted]

    new_value = np.concatenate(l_ord_value)
    new_time = np.concatenate(l_ord_times)

    return new_value, new_time


def do_cprofile(func):
    """
    With the decorator @do_cprofile the function is analyzed by cprofile module.

    :param func:
    :return:
    """
    def profiled_func(*args, **kwargs):
        profile = cProfile.Profile()
        try:
            profile.enable()
            result = func(*args, **kwargs)
            profile.disable()
            return result
        finally:
            profile.print_stats()
    return profiled_func


def filter_values(myarray, value):
    """
    Assign nan to those index where the array has value.

    :param myarray:
```

```
    :param value:
    :return:
    """
    my_array = myarray.astype('float') # to ensure next statement will work
    my_array[my_array == value] = np.nan

    return my_array


def create_set_TS(lat_val, dim_plev, target_array,
                  pmax=100.1, pmin=99.9, latmin=-20., latmax=20.):
    """
        This function returns an flatten array whose index represents time
    The input is target_array(time, plev, lat) so an average over lat range
    is done and an specific plev is selected.

    :param lat_val:
    :param dim_plev:
    :param target_array:
    :param pmax:
    :param pmin:
    :param latmin:
    :param latmax:
    :param iloop:
    :return:
    """
    warnings.filterwarnings('ignore')

    with warnings.catch_warnings():
        warnings.simplefilter("ignore", category=RuntimeWarning)

        # lat_val = full_nc.variables['lat'][:]
        plev_inds = np.where((dim_plev >= pmin) & (dim_plev<= pmax))
        lat_inds  = np.where((lat_val >= latmin) & (lat_val<= latmax))

        plev_fact = 0.1
        plev_iter = 0
        pmin_new = pmin
        pmax_new = pmax
        while len(plev_inds[0]) == 0:
            plev_iter += 1
            pext = plev_iter*plev_fact
            plev_inds = np.where((dim_plev >= pmin-pext) & (dim_plev <= pmax+pext))
            logs.INFO.warning(' --- IN aux.create_set_TS --- plev limit increased')
            pmin_new = pmin-pext
            pmax_new = pmax+pext

        lat_fact = 0.5
        lat_iter = 0
        latmin_new = latmin
        latmax_new = latmax

        while len(lat_inds[0]) == 0:
            lat_iter += 1
            lext = lat_iter*lat_fact
            lat_inds = np.where((lat_val >= latmin-lext) & (lat_val <= latmax+lext))
            logs.INFO.warning(' --- IN aux.create_set_TS --- lat limit increased')
            latmin_new = latmin-lext
            latmax_new = latmax+lext

        if lat_iter > 0:
            str_log = ' IN aux.create_set_TS --- lat limit set from'
            str_log += '(%3.3f,%3.3f) to %3.3f,%3.3f) ' %(latmin, latmax, latmin_new, latmax_new)
            logs.DEBUG.warning(str_log)
        if plev_iter > 0:
            str_log = ' IN aux.create_set_TS --- plev limit set from'
            str_log += '(%3.3f,%3.3f) to %3.3f,%3.3f) ' %(pmin, pmax, pmin_new, pmax_new)
            logs.DEBUG.warning(str_log)

        target_test = target_array[:, plev_inds[0], lat_inds]
        target_weig = np.ones_like(target_test)
        sizes = target_weig.shape
        ts_out = np.zeros((sizes[0],sizes[1]))


        fact = 0.0
        for lat in lat_val[lat_inds[0]]:
            fact = fact + np.abs(np.cos(lat*np.pi/180.))

        for i_val in range(sizes[0]):
            for j_val in range(sizes[1]):

                for k_val, lat_id in zip(range(sizes[2]), lat_inds[0]):
                    norm =  np.abs(np.cos(lat_val[lat_id]*np.pi/180.))/fact
                    ts_out[i_val, j_val] = ts_out[i_val, j_val] + target_test[i_val, j_val, k_val]*norm

    return ts_out.flatten(), plev_inds[0][0]


def create_set_TS_2D(lat_val, dim_plev, target_array,
                  pmax=100000.1, pmin=0.01, latmin=-5., latmax=5.):
    """
        This function returns an flatten array whose index represents time
    The input is target_array(time, plev, lat) so an average over lat range
    is done and an specific plev is selected.

    :param lat_val:
    :param dim_plev:
    :param target_array:
    :param pmax:
    :param pmin:
```

```python
    :param latmin:
    :param latmax:
    :param iloop:
    :return:
    """
    warnings.filterwarnings('ignore')

    with warnings.catch_warnings():
        warnings.simplefilter("ignore", category=RuntimeWarning)

        plev_inds = np.where((dim_plev >= pmin) & (dim_plev<= pmax))
        lat_inds  = np.where((lat_val >= latmin) & (lat_val<= latmax))

        plev_fact = 0.1
        plev_iter = 0
        pmin_new = pmin
        pmax_new = pmax
        while len(plev_inds[0]) == 0:
            plev_iter += 1
            pext = plev_iter*plev_fact
            plev_inds = np.where((dim_plev >= pmin-pext) & (dim_plev <= pmax+pext))
            logs.INFO.warning(' --- IN aux.create_set_TS --- plev limit increased')
            pmin_new = pmin-pext
            pmax_new = pmax+pext

        lat_fact = 0.5
        lat_iter = 0
        latmin_new = latmin
        latmax_new = latmax

        while len(lat_inds[0]) == 0:
            lat_iter += 1
            lext = lat_iter*lat_fact
            lat_inds = np.where((lat_val >= latmin-lext) & (lat_val <= latmax+lext))
            logs.INFO.warning(' --- IN aux.create_set_TS --- lat limit increased')
            latmin_new = latmin-lext
            latmax_new = latmax+lext

        if lat_iter > 0:
            str_log = ' IN aux.create_set_TS --- lat limit set from'
            str_log += '(%3.3f,%3.3f) to %3.3f,%3.3f) ' %(latmin, latmax, latmin_new, latmax_new)
            logs.DEBUG.warning(str_log)
        if plev_iter > 0:
            str_log = ' IN aux.create_set_TS --- plev limit set from'
            str_log += '(%3.3f,%3.3f) to %3.3f,%3.3f) ' %(pmin, pmax, pmin_new, pmax_new)
            logs.DEBUG.warning(str_log)


        new_levs = plev_inds[0].tolist()
        target_testA = target_array[:, :, lat_inds[0]]
        target_test = target_testA[:, new_levs,:]
        target_weig = np.ones_like(target_test)
        sizes = target_weig.shape
        ts_out = np.zeros((sizes[0],sizes[1]))


        fact = 0.0

        for lat in lat_val[lat_inds[0]]:
            fact = fact + np.abs(np.cos(lat*np.pi/180.))

        for i_val in range(sizes[0]):
            for j_val in range(sizes[1]):

                for k_val, lat_id in zip(range(sizes[2]), lat_inds[0]):
                    norm =  np.abs(np.cos(lat_val[lat_id]*np.pi/180.))/fact
                    ts_out[i_val, j_val] = ts_out[i_val, j_val] + target_test[i_val, j_val, k_val]*norm

    return ts_out, new_levs


def up():
    # My terminal breaks if we don't flush after the escape-code
    sys.stdout.write('\x1b[1A')
    sys.stdout.flush()


def down():
    # I could use '\x1b[1B' here, but newline is faster and easier
    sys.stdout.write('\n')
    sys.stdout.flush()

def movingaverage(interval, window_size):
    window = np.ones(int(window_size))/float(window_size)
    return np.convolve(interval, window, 'same')

def running_mean(x, N):
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / N


def parse_datetime(dataset_x):
    """
    This function just assign the day to 15. It might be useful for some data-analysis with other tools.

    :param dataset_x:
    :return: datetime_x
    """

    time_unit_model_x = utime(str(dataset_x.variables['time'].units).replace('"', ''))
    time_data_model_x = time_unit_model_x.num2date(dataset_x.variables['time'][:])
```

```
    dattime_x = [mytime.replace(day=15, hour=0, minute=0, second=0, microsecond=0)
                 for mytime in time_data_model_x]
    datetime_x = np.array(dattime_x)

    return datetime_x

def create_dataset_timeunits_other(file1, file2, newfile1):
    """
    This function take three arguments as file names of netcdf files
    file1: is a netcdf with time units to change
    file2: is a netcdf with the time units we want
    newfile1: is a name for a new netcdf file that will be the file1
              but with the file2 time units and calendar.
    """

    dataset_2 = Dataset(file2)
    time_units = dataset_2.variables['time'].units

    src = Dataset(file1,'r')
    dst = Dataset(newfile1, 'w')

    for namedimen in src.dimensions.keys():
        dimension = src.dimensions[namedimen]
        if not dimension.isunlimited():
            dst.createDimension(namedimen, size=len(dimension))
        else:
            dst.createDimension(namedimen, size=None)

    for varname in src.variables.keys():

        variable = src.variables[varname]

        if varname == 'some_variable':
            continue

        if varname == 'time':
                x = dst.createVariable(varname, variable.datatype,
                                       dimensions=variable.dimensions)
                x.units = time_units
                x.calendar = 'standard'

                time_x = num2date(src.variables['time'][:],
                                  src.variables['time'].units,
                                  calendar=src.variables['time'].calendar)

                dattime_x = time_x - timedelta(days=15)
                x[:] = date2num(dattime_x, time_units, calendar='standard')

        elif varname == 'time_bnds':
                x = dst.createVariable(varname, variable.datatype,
                                       dimensions=variable.dimensions)
                x.units = time_units
                time_bnd_0 = num2date(src.variables['time_bnds'][:][:,0],
                                      src.variables['time'].units,
                                      calendar=src.variables['time'].calendar)
                time_bnd_1 = num2date(src.variables['time_bnds'][:][:,1],
                                      src.variables['time'].units,
                                      calendar=src.variables['time'].calendar)

                x[:]=np.array([date2num(time_bnd_0, time_units, calendar='standard'),
                      date2num(time_bnd_1, time_units, calendar='standard')]).T

        else:
                x = dst.createVariable(varname,variable.datatype,
                                       dimensions=variable.dimensions)
                x[:] = src.variables[varname][:]

    src.close()
    dst.close()

    return


def create_dataset_timeunits_cmam(file1, newfile1):
    """
    This function take two arguments as file names of netcdf files
    file1: is a netcdf with time units to change
    newfile1: is a name for a new netcdf file that will be the file1
              but with the units 'days since 1850-01-01 00:00:00'
              and calendar standard.
    """

    new_time_units = 'days since 1850-01-01 00:00:00'

    src = Dataset(file1,'r')
    dst = Dataset(newfile1, 'w')

    cmam_time_units = src.variables['time'].units
    for namedimen in src.dimensions.keys():
        dimension = src.dimensions[namedimen]
        if not dimension.isunlimited():
            dst.createDimension(namedimen, size=len(dimension))
        else:
            dst.createDimension(namedimen, size=None)

    for varname in src.variables.keys():

        variable = src.variables[varname]

        if varname == 'some_variable':
```

```python
            continue
        if varname=='time':
                x = dst.createVariable(varname, variable.datatype,
                                       dimensions=variable.dimensions)
                x.units = new_time_units
                x.calendar = variable.calendar
                time_obj = utime(str(cmam_time_units.replace('00:00','00:00:00')).replace('"', ''))
                time_x = time_obj.num2date(variable[:])
                new_time_obj = utime(new_time_units)
                new_cmam_tim = new_time_obj.date2num(time_x)
                x[:] = new_cmam_tim
        else:
                x = dst.createVariable(varname,variable.datatype,
                                       dimensions=variable.dimensions)
                x[:] = src.variables[varname][:]
    src.close()
    dst.close()

    return
```

List of Figures