FRM4GHG 2.0
Fiducial Reference Measurements for Greenhouse Gases

Create handling for CAMS a-priori for PROFFAST

Deliverable:	**D3.1.2**
Date:	**20/09/2023**
Lead authors:	**Benedikt Herkommer, Frank Hase**
Subject:	**ESA-Contract No. 4000136108/21/I-DT-lr**
Category:	**ESA Express Procurement (EXPRO)**
Our ref.:	**Proposal, FRM4GHG 2.0 – Expro, ESA RFP/3-17031/21/I-DT-lr, revision from 27 August 2021 and 7 September 2021**

Table of contents

1 DOCUMENT CHANGE RECORD

Issue	Date	Item	Comment
V1.0	2023-09-20	-	Initial version

2 ACCESS LIST

This document is the deliverable “D3.1.2: Create handling for CAMS a-priori for PROFFAST” created for the project FRM4GHG 2.0 and will be submitted to ESA. The document can be downloaded from the project webpage http://frm4ghg.aeronomie.be.

3 PURPOSE

This document reports the progress on the Task 2 of the WP 3 of the ESA project “Fiducial Reference Measurements For Ground-based IR Greenhouse Gas Observations 2.0”.
It focuses on the deliverable “D3.1.2” which describes the progress in using a priori trace gas profiles generated by the Copernicus Atmospheric Monitoring Service (CAMS) for the COCCON data analysis.

4 DOCUMENT STRUCTURE

See Table of Contents

5 APPLYING CAMS A PRIORI TO PROFFAST

For the processing of atmospheric ground-based remote sensing Fourier Transform Infrared (FTIR) measurements so-called a-priori profiles of atmospheric temperature and vertical mixing ratios of gases are required. Specifically, the PROFFAST data analysis used by COCCON also requires a-priori data.
The a-priori files provide pre-calculated height resolved profiles of the temperature and relevant trace gases. The required data are either based on climatological expectation or derived from atmospheric model runs.

As default the Collaborative Carbon Column Observing Network (COCCON) uses the same a-priori as the Total Carbon Column Observing Network (TCCON) which is generated centrally by the California Institute of Technology using data from the Goddard Earth Observing System Forward Product for Instrument Teams (GEOS FP-IT) (Laughner, et al., 2023). The files containing the a-prioris are called the “map-files” in the following. These data become available with some delay (in the order of several days), so this choice might hamper near-real time FTIR data processing in some future. Moreover, some features of the reported gas profiles are derived from climatological expectation, so it is expected that using profiles generated by a dedicated state-of-the-art atmospheric CAMS model run would allow for a superior description of the atmospheric state.

Note that the input format for the a-priori information required by PROFFAST carefully follows the format convention set by TCCON. We believe that this is an important aspect of COCCON data processing design philosophy, as this choice conserves the direct interchangeability of a-priori information between TCCON and COCCON, which obviously is a highly desirable feature. The files provided currently by BIRA still deviate from the format defined by TCCON. We created a workaround in order to compensate for this incompatibility by implementing specific adaptions in the PROFFAST version used for this exercise, but it should be kept in mind as important requirement that any further development of using CAMS profiles for operational COCCON data analysis needs to be realized in full accordance to the map-file formatting imposed by TCCON.

In the framework of the FRM4GHG-II project the project partner Royal Belgian Institute for Space Aeronomy (BIRA) generated map-files based on the data of the Copernicus Atmospheric Monitoring Service (CAMS) in a similar format needed for the PREPROCESS used by the COCCON. This is described in the status report for the deliverable D.3.1.1.

The CAMS map-file contains a-priori information for temperature, CO_2, CH_4 and H_2O. This document describes the conversion of the CAMS map-files to a PROFFAST compatible format and needed adaptations in PROFFAST and the PROFFASTpylot. Furthermore, the map-file is used to evaluate a test-day of COCCON data and compared to a standard COCCON evaluation.

5.1 Generation of a PROFFAST compatible a-priori and adaption to PROFFAST

The provided CAMS a-priori are not directly compatible with TCCON map files as digested by PROFFAST for two reasons. Firstly, they contain height resolved mixing ratios of only CO_2, CH_4 and H_2O. To get a retrieval working at least the height resolved mixing ratio of O_2 would be needed. Secondly, the format is not compatible with the TCCON map-file format. PROFFAST reads in the map-file format using a fixed FORTRAN format specifier.

Therefore, a program, named MapfileMerger, is written, which takes data of the standard TCCON a prioris (in the following called FPIT) to complement the missing columns in the CAMS a priori. Furthermore, this program writes out the data in a format which is compatible with PROFFAST as well as the PROFFASTpylot.

The code of this program is given in Appendix A. It is written as a standalone program, but designed such to be integrated as an Ad-On to the PROFFASTpylot with very minor changes.
For example, it reads in the standard PROFFASTpylot input file such that in case of an add on the input data can either be directly inherited by the PROFFASTpylot class or be passed from the PROFFASTpylot module “Preparation” to the “MapfileMerger”.

In addition, it is necessary to make three minor changes in the PROFFAST and PROFFASTpylot code:

1. In the class “Preparation” of the PROFFASTpylot in the module, it is necessary to implement a flexible time difference between two map-files. As the FPIT a priori are created in a 3-hour interval this time difference was hardcoded. Now it is read out by the following code in the

method "_interpolate_map_files":

```
t2 = int(mapfiles[i_noon][-7:-5])
t1 = int(mapfiles[i_noon - 1][-7:-5])
# interpolate between the files
   tdiff = abs(t2 - t1) * 60 * 60   # seconds
```

2. The module "pcxs" of PROFFAST reads in the a priori data. As the CAMS map-files have a finer grid than the FPIT map-files it is necessary to increase a hard-coded level limit in "pcxs". In the subroutine "read_pT" the variable "npT_max" must be changed from 120 to 140.
3. In the subroutine "read_vmr" the variable "nvmr_max" must be changed from 120 to 140.

5.2 Processing of One Day of Test Data

To test the CAMS a priori the data collected during 3rd of January 2023 in Karlsruhe using the COCCON reference spectrometer SN37 are processed twice using either the standard TCCON (FPIT) map-files or the newly generated hybrid CAMPS-FPIT map files. Figure 5-3 shows the a-priori files according to TCCON and CAMS.
The results for XCO_2 and XCH_4 are plotted in Figure 5-1 and Figure 5-2, respectively. The upper panel show in blue the time series of the XGas values retrieved using the FPIT a priori in blue and in the CAMS a priori in orange. The lower panels show the difference between both retrievals.
The maximum deviation for XCO_2 is approximately 0.08 % and 0.14 % for XCH_4.

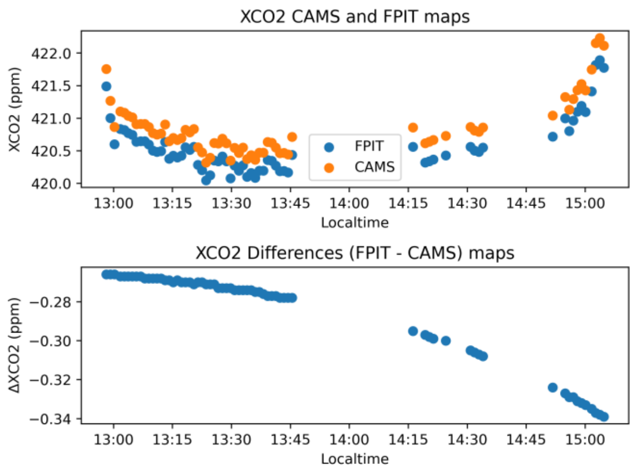

Figure 5-2: XCO_2 results of PROFFAST using the different a priori.

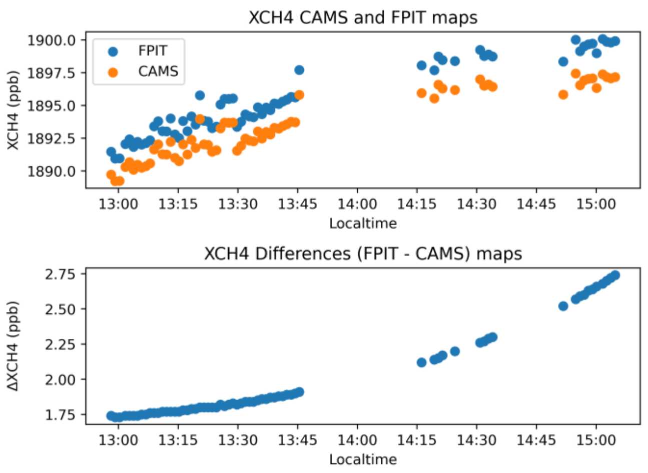

Figure 5-1: XCH_4 results of PROFFAST using the different a priori.

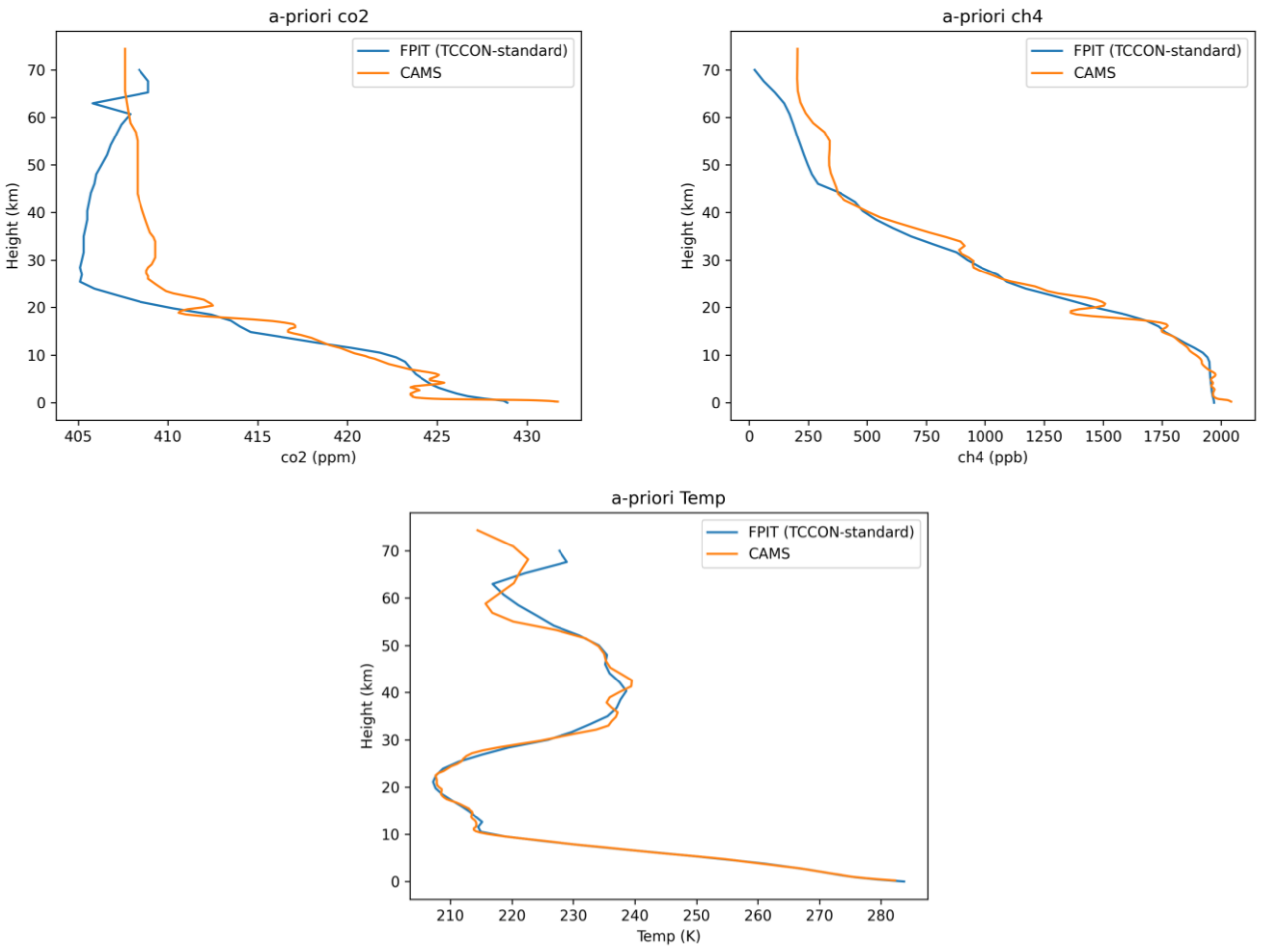

Figure 5-3: direct comparison of a-priori profiles of CO_2, CH_4 and the Temperature (TCCON and CAMS).

6 APENDIX A

```python
import yaml
import os
import pandas as pd
import numpy as np
import logging
import glob
import datetime as dt
import fortranformat as ff

class MapfileMerger:
    """Merge the mapfiles by BIRA and by Caltech to a single map-file"""
    def __init__(self, inputfile):
        """Init the MapfileMerger class"""
        # create a logger. Can be replaced later by pylots logger
        self.logger = self.create_logger()

        # read input file
        with open(inputfile, "r") as f:
            args = yaml.load(f, Loader=yaml.FullLoader)
        # and safe the content as `self` variables:
        for option, value in args.items():
            self.__dict__[option] = value

    def create_logger(self):
        """Create a logger. Can be replaced later by Pylots logger."""

        logger = logging.getLogger()
        # set logging to debug to record everything in the first place
        logger.setLevel(logging.DEBUG)
        StreamHandler = logging.StreamHandler()
        StreamHandler.setLevel(logging.DEBUG)
        logger.addHandler(StreamHandler)
        return logger

    def read_fpit_mapfiles(self, day:dt.date):
        """Read the caltech mapfile."""
        # We only need the hours 00, 06, 12, and 18:
        map_list = self._get_mapfile_list(day, self.fpit_map_path)

        self.fpit_mapfiles = {}
        for file, hour in zip(map_list, [0, 6, 12, 18]):
            tempdf = pd.read_csv(
                file,
                skipinitialspace=True,
                header=10)
            tempdf.drop(index=0, inplace=True)
            tempdf = tempdf.astype(np.float64)
            self.fpit_mapfiles[hour] = tempdf

    def read_cams_mapfiles(self, day:dt.date):
        """Read the bira mapfiles"""
        map_list = self._get_mapfile_list(day, self.cams_map_path)
        # remove potential `comma_separated` files
        for file in map_list.copy():
            if "comma_seperated" in file:
                map_list.remove(file)
        self.cams_mapfiles = {}
        for file, hour in zip(map_list, [0, 6, 12, 18]):
            # we first need to add commas to the data:
            comma_file = os.path.join(
                os.path.dirname(file),
                "comma_seperated_" + os.path.basename(file))
            with open(file, "r") as infile:
                with open(comma_file, "w") as outfile:
                    for i, line in enumerate(infile):
                        if i < 10: continue
                        elif i == 10:
                            outfile.write(line[2:]) # drop `#`
                        elif i == 11: continue
                        else:
                            line = line.split()
                            line = ",".join(line)
                            outfile.write(line + "\n")
            # now we have an easy-readble csv file:
            tempdf = pd.read_csv(comma_file)
            tempdf = tempdf.astype(np.float64)
            self.cams_mapfiles[hour] = tempdf

    def interpolate_caltech_to_cams_height(self):
        """Interpolate the Caltech data to the CAMS height grid"""
        self.interpolated_map = {}

        def replace_NaN(row:pd.Series):
            row = row.fillna(np.nan)
            if np.isnan(row["Height"]):
                row["Height"] = row["Height_fpit"]
            return row

        for h in [0, 6, 12, 18]:
            cams = self.cams_mapfiles[h]
            fpit = self.fpit_mapfiles[h]
            # fpit = pd.DataFrame()
```

```python
            fpit.rename(
                lambda x: x + "_fpit", inplace=True, axis="columns")
            fpit_cols = fpit.columns
            merged_df = pd.merge_ordered(
                cams, fpit, left_on="Height", right_on="Height_fpit")

            merged_df = merged_df.apply(
                replace_NaN, axis=1)

            merged_df.set_index("Height", inplace=True, drop=True)
            for column in fpit_cols:
                merged_df[column] =\
                    merged_df[column].interpolate(
                        method="slinear")

            merged_df.dropna(subset=["Temp"], inplace=True)
            merged_df.dropna(subset=["hf_fpit"], inplace=True)

            merged_df.reset_index(inplace=True, drop=False)

            merged_df.drop(columns=[
                "Height_fpit", "Temp_fpit", "Pressure_fpit",
                "Density_fpit", "h2o_fpit", "co2_fpit", "ch4_fpit",
                "gravity_fpit"], inplace=True)
            merged_df.rename(columns={
                    "hdo_fpit": "hdo",
                    "n2o_fpit": "n2o",
                    "co_fpit": "co",
                    "hf_fpit": "hf",
                    "o2_fpit": "o2"
                }, inplace=True)
            merged_df = \
                merged_df[[
                    "Height", "Temp", "Pressure", "Density", "h2o", "hdo",
                    "co2", "n2o", "co", "ch4", "hf" , "o2", "gravity"]]
            self.interpolated_map[h] = merged_df

    def convert_fpit_wet_to_dry(self):
        """Convert the columns of the cams map-file to dry air mole fractions
        """
        gas_cols = ["h2o", "co2", "ch4", "o2", "co", "hdo", "hf"]
        for h in [0, 6, 12, 18]:
            df = self.fpit_mapfiles[h]
            df["h2o_dry"] = 1 / (1 / df["h2o"] - 1)
            for gas in gas_cols:
                if gas == "h2o":
                    continue
                df[gas+"_dry"] = df[gas] * (1 + df["h2o_dry"])
            df["Density_dry"] = df["Density"] - df["Density"] * df["h2o"]
            for gas in gas_cols + ["Density"]:
                df.drop(columns=[gas], inplace=True)
                df.rename(columns={gas+"_dry": gas}, inplace=True)
            self.fpit_mapfiles[h] = df

    def write_final_mapfiles(self, day):
        """Write the final map-files in the format provided by gSetup"""
        header = \
"""12 13
ka_49N_008E_2023010100Z.map
This is the header of a custom map-file which is generated by a combining CAMS and FPIT data.
The header must be 12 lines long and needs the following statement in the second line:
ID_xxN_zzzE_.
The xx must be the latitude N stands for North or South. zzz is the longitude E ist East or West.
All columns are DRY-air mole fractions!
The CAMS data are provided by BIRA which comprises
Height, Temp, Pressure, Density, h2o, co2, ch4, gravity
The FPIT data are provided by Caltech which provide a standard map file.
Height,Temp,Pressure,Density,h2o,hdo,co2,n2o,co,ch4,hf,o2,gravity
km,K,hPa,molecules_cm3,parts,parts,ppm,ppb,ppb,ppb,ppt,parts,m_s2
"""
        for h in [0, 6, 12, 18]:
            output_mapfile = \
                os.path.join(
                    self.map_path,
                    f"{self.site_abbrev}_CustomCAMS_"
                    f"{day.strftime('%Y%m%d')}{h:02d}Z.map")

            with open(output_mapfile, "w") as f:
                f.writelines(header)
                # write the rest of the file

                frw = ff.FortranRecordWriter(
                    "(2(f8.3,','),4(e10.4,','),1x,(f7.3,','),1x,(f7.3,','),"
                    "(e10.3,','),1x,(f6.1,','),(f8.3,','),1x,(f6.4,','),1x,"
                    "f5.3)")
                data = self.interpolated_map[h]
                data = data.to_numpy()
                for line in data:
                    f.write(frw.write(line) + "\n")

    def _get_mapfile_list(self, day:dt.date, path:str):
        """Create a list of the needed map files and check if available."""
        # We only need the hours 00, 06, 12, and 18:
        map_list = []
        for hour in ["00", "06", "12", "18"]:
            temp_list = glob.glob(os.path.join(
                path, f"*_{day.strftime('%Y%m%d')}{hour}Z.map"
            ))
            if len(temp_list) == 0 or (temp_list is None):
                self.logger.error(
                    f"Could not find mapfile matching the scheme"
                    f"'*_{day.strftime('%Y%m%d')}{hour}Z.map' in {path}. "
                    "Quit the program!"
                    )
                exit()
            map_list.extend(temp_list)
        map_list.sort()
        return map_list




if __name__ == "__main__":
    MyMerger = MapfileMerger(r"PROFFASTpylot_example_input.yml")

    testday = dt.date(year=2023, month=1, day=3)
    MyMerger.read_fpit_mapfiles(testday)
    MyMerger.convert_fpit_wet_to_dry()

    MyMerger.read_cams_mapfiles(testday)

    MyMerger.interpolate_caltech_to_cams_height()
    MyMerger.write_final_mapfiles(testday)
```

7 LITERATURE

Laughner, J. L., Roche, S., Kiel, M., Toon, G., Wunch, D., Baier, B., . . . Wennberg, P. (2023). A new algorithm to generate a priori trace gas profiles for the GGG2020 retrieval algorithm. *Atmospheric Measurement Techniques*, S. 1121-1146. doi:10.5194/amt-16-1121-2023