FTS Profile Retrieval Pre and Post Processing

Eric Nussbaumer[1], James Hannigan*[1] and Ivan Ortega[1]

[1]*National Center for Atmospheric Research, Boulder, CO, USA*

June 2021

Abstract

This document outlines the creation of the spectral database as well as the profiles for pressure, temperature, and water vapor.

*corresponding author: jamesw@ucar.edu

Contents

1 Introduction

This document describes the step-by-step procedures and computing tools, locations and repositories used by the NCAR OT group to process ground based high resolution spectral data from the sites located at Thule Greenland, Mauna Loa, Hawaii and Boulder Colorado. These processing tools can be applied to any similar data set.

The necessary python files described in this document are located in the sfit-processing-environment git repository. See section **??** for a detailed description of the supporting repositories.

The processing environment (PE) - python package - described here is used for pre and post-processing of sfit4. The PE is used for:

1. Pre-processing of spectra: building a spectral database, acquiring auxiliary data e.g. NCEP ZPTW (altitude, pressure, temperature, and water vapor)
2. Batch running of SFIT4 on large sets of spectra
3. Running the error analysis codes post retrieval
4. Plotting standard outputs
5. Using the database and retrievals to build GEOM compliant HDF files

2 Pre-Processing

The spectral database and ZPTW profiles are necessary pre-processing steps to retrievals. The spectral database holds information pertaining to each of the measurements. A spectral database is unique to each site

The majority of information in the spectral database comes from the OPUS file itself; however, we append in-situ meteorological data from local weather stations.

There are several steps in creating the spectral database:

1. Creating the initial spectral database
2. Re-formatting the house log data files
3. Re-formatting the external station weather data
4. Appending the initial spectral database with house an external station weather data

Note that not all sites have house or external station weather data.

Station	House Data	External Station Data
MLO	Yes	Yes (CMDL)
TAB	Yes	No
FL0	No	Yes (EOL)

2.1 Pulling Data

Both the ancillary data as well as the OPUS files need to be downloaded from various sources. The OPUS data is automatically downloaded from MLO and TAB by the program pullRemoteData2.py. This program is set on a cron tab to download data everyday.
The following table shows where the OPUS data is downloaded to.

Data	Local Storage
MLO	otserver:/ya4/id/mlo/
TAB	otserver:/ya4/id/tab/

The supporting data is pulled with a program using wget. The program is pullAncillary-Data.py and is located at: /data/bin/. This program has been setup in cron tab to pull data everyday. The program pullAncillaryData.py gets the following data: NCEP nmc, NCEP I re-analysis, EOL, and CMDL.

ERA-Interim data must be manually pulled through the server data-access.ucar.edu.

The following table shows the local storage of the ancillary data

Data	Local Storage
WACCM	otserver:/data/Campaign/TAB,MLO,FL0/waccm/
NCEP nmc Height	otserver:/data1/ancillary_data/NCEP_NMC/height/
NCEP nmc Temp	otserver:/data1/ancillary_data/NCEP_NMC/temp/
NCEP I Height	otserver:/data1/ancillary_data/NCEPdata/NCEP_hgt/
NCEP I Shum	otserver:/data1/ancillary_data/NCEPdata/NCEP_Shum/
NCEP I Temp	otserver:/data1/ancillary_data/NCEPdata/NCEP_Temp/
NCEP I Trpp	otserver:/data1/ancillary_data/NCEPdata/NCEP_trpp/
ERA-Interim	otserver:/data1/ancillary_data/ERAdata/
EOL	otserver:/data1/ancillary_data/fl0/eol/
CMDL Hourly	otserver:/data1/ancillary_data/mlo/cmdl/Hourly_Data/
CMDL Minute	otserver:/data1/ancillary_data/mlo/cmdl/Minute_Data/

Note: For security reasons the crontab is initialized in the otserver under Jim's account.

2.2 Initial Quality Check

An initial quality check on the spectrum is done using a GUI written in python (ckopPy.py). This python script uses a python Class to read opus format (nicely provided by Wolfgang Stremme, CCA-UNAM, Mexico). This GUI calculates a SNR based on out of band noise (or any other band) and maximal signal. Additionally, a proxy is created to integrate positive and negative values to create a ratio as a second quality check for each spectra. This program is available upon request. Note that this step needs to be performed through each individual spectra and look for good and bad spectra. Another python program (mvSpectra.py) can be used to sync folders from the /ya4/id/(mlo,tab,fl0) to the directory /data1/(mlo,tab,fl0).

Program	Description
ckopPy.py	Python program to check OPUS spectra
mvSpectra.py	Python program to sync folders from /ya4/id/(mlo,tab,fl0) to /data1/(mlo,tab,fl0)

3 Spectral Database

3.1 Initial Spectral Database

The initial spectral database file is created by running ckopus on the various raw OPUS files. A python program is created to manage the creation of the initial spectral database file (mkSpecDB.py). The program will create a new spectral database file or append an already existing file. Associated with mkSpecDB.py is an input file. The input file allows one to specify the starting and ending date to process, the station, and the various directories and files to use. In addition, one can specify additional ckopus flags to use in the ckopus call. There are logical flags which control the creating of a file which list the folders processed and whether bnr files are created. These files are located under the SpectralDatabase folder of the git repository.

The initial spectral databases should be made for individual years. The output files have the names spDB_loc_YYYY.dat. They are space delimited text files. You can either re-write the particular year database file or append it depending if you are processing an entire year or just a portion of it. If you want to append the spectral database file keep the original file in place and just adjust the time interval in the specDBInputFile.py input file. Make sure the start date in the input file is at least a day after the last entry in the spectral database file.

An input file for each site (MLO,FL0,TAB) already exists in each of the Campaign directories (/data/Campaign/(MLO,TAB,FL0)/Spectra_DB/.

Program	Description
mkSpecDB.py	Main program to create initial spectral database
specDBInputFile.py	Input file for mkSpecDB.py program

Note: Read through the individual input files. The configuration of the instruments, such as time, needs to be read properly and the flags are in the input files.

3.2 House Data

House data is data that is recorded by the FTS autonomous system, such as outside temperature, pressure, wind direction, etc. Up to now, this applies only to MLO and TAB. The format of this data has changed for each station over time as the instrument gets modified or upgraded. A python program (station_house_reader.py) is created to read the various formats and create a standardized file. There is one file for each year. There are no input files for the station_house_reader.py program. The time range, station identifier, and directories are specified directly in the program under the main function. The program creates the housekeeping files under /data/Campaign/MLO (or TAB). An excel spreadsheet describes the various formats for the house log files for MLO and TAB.

Program	Description
station_house_reader.py	Main program to read house data files
HouseReaderC.py	Supporting program with formats of previous house data files
HouseDataLog_2x̀lsx	Excel file with format of house log files

These programs are located in the ExternalData folder of the git repository.

3.3 External Station Data

There are currently two external station data sources used (EOL for FL0, and CMDL for MLO) only the EOL data needs to be pre-processed. The original format of this data is in netcdf files. The program read_FL0_EOL_data.py reads the daily netcdf files and creates a yearly text file. There are no input files for read_FL0_EOL_data.py program. The year of interest and directories of data are specified directly in the program under the main function. The program pullAncillaryData.py pulls the CMDL and EOL data from each individual ftp site.

Program	Description
pullAncillaryData.py	Program to automatically pull EOL and CMDL data
read_FL0_EOL_data.py	Main program to read EOL and CMDL data

These programs are located in the ExternalData folder of the git repository.

3.4 Append Spectral Database File

One can now appending the initial spectral database file with the house and external station weather data. A python program was created to accomplish this (appendSpecDB.py). The program appendSpecDB.py reads in the initial spectral database file. It then searches the house and external station files for weather data at the time of observation, plus a certain number of minutes specified by the user. The mean of the data collected is calculated and a

new spectral database file is created. If no data is present missing values are used. Associated with appendSpecDB.py is an input file. The input file allows one to specify directories and files, year to process, station, how many minutes to use for averaging, and whether to create a comma separated or pre-specified formatted new spectral database file.

One should remove previous spectral database files (HRspDB_loc_YYYY.dat) from the output location before creating new appended spectral database files.

Each site already has an input file located in their Campaign directory.

Program	Description
appendSpecDB.py	Program to create the append spectral database file
appndSpecDBInputFile.py	Input file for appendSpecDB.py

Note:A warning message will often appear when running this program originating from the python numpy module. This warning is a result of numpy taking the mean of an empty array. This is handled by the main program.

The sfit4Layer1 processing looks for a database file with all years. Once you create a year appended spectral database file you should copy or append this to the file which contains all years processed. For example say you have processed 2010 to 2014 and have a spectral database file called HRspDB_fl0_2010_2014.dat. You then create an and appended spectral database file for 2015 called HRspDB_fl0_2015.dat. You should append the contents of this file to the HRspDB_fl0_2010_2014.dat file and rename to HRspDB_fl0_2010_2015.dat. Take care not to include the header of the individual year file. There should only be one header at the top in the conglomerated file.

3.5 Coadd Spectral Database File

When the spectral data is to be co-added an additional step must be taken after the creation of the appended spectral database. The program mkCoadSpecDB.py co-adds two bnr files together with the appropriate forward and backward scans. A new coadded bnr file is created. The program mkCoadSpecDB.py calls the C program coadd.c to co-add the files. The program mkCoadSpecDB.py requires an input file.

The mkCoadSpecDB.py program reads in a HRspDB_loc_YYYY.dat data file and creates a CoaddspDB_loc_YYYY.dat output file. One can either run the co-add program on a conglomerated spectral database file (e.g. HRspDB_fl0_2010_2014.dat) or on individual year file (e.g. HRspDB_fl0_2014.dat) and then conglomerate all the years. When running the co-add program all previous output files with the same name should be removed.

Program	Description
mkCoadSpecDB.py	Main program to create coadded spectral database
CoadSpecDBInputFile.py	Input file for mkCoadSpecDB.py

Note: The co-adding spectra is normally needed at FL0 and TAB (before 2014).

4 ZPTW Profiles

The pressure, temperature, and water vapor profiles can be created from several outside sources. Temperature and pressure profiles are taken from NCEP nmc data; while currently only water profiles are taken from NCEP I and ERA-Interim re-analysis data. Both NCEP and ERA-Interim data are interpolated with WACCM data to reach 120km vertical height. The profiles are daily averages and they reside in the data directories (/data1/tab,mlo,fl0/).

The following is a table showing the various reference profiles, their sources, along with the associated file names.

Profile Type	**Source**	**File Name**
Temperature	NCEP nmc	ZPT.nmc.120
Pressure	NCEP nmc	ZPT.nmc.120
Water Vapor	WACCM	w-120.v1
Water Vapor	NCEP I	w-120.v3
Water Vapor	ERA-Interim	w-120.v4
Water Vapor	Retrieved	w-120.YYYYMMDD.HHMMSS.v99
Water Vapor	Retrieved Daily	w-120.v5

The following table shows the various sources for the data.

Data	**Source**
WACCM	Local (otserver:/data/Campaign/TAB,MLO,FL0/waccm/
NCEP nmc	ftp://ftp.cpc.ncep.noaa.gov/ndacc/ncep/
NCEP I re-analysis	ftp://ftp.cdc.noaa.gov/Datasets/ncep.reanalysis.dailyavgs/
ERA-Interim re-analysis	/glade/p/rda/data/ds627.0/ei.oper.an.pl/

4.1 Pressure & Temperature Profiles

Pressure and temperature profiles in the ZPT.nmc.120 files come from NCEP nmc data. The NCEP nmc data is vertically interpolated with WACCM data to reach 120km. In the event that the NCEP nmc data is not available for a particular day, the WACCM data is substituted.

The NCEP nmc data must first be formatted. This is done using the program NCEPnmcFormat.py.

After formating the NCEP nmc data one can create the altitude, pressure, and temperature profiles using the program MergPrf.py. This program also creates water profiles from WACCM data (v1).

Program	Description
NCEPnmcFormat.py	Program to format the NCEP nmc data
NCEPinputFile.py	Input file for NCEPnmcFormat.py program
MergPrf.py	Main program to create ZPT and water files from WACCM data
mergprfInput.py	Input file for MergPrf.py program

4.2 NCEP I & ERA Interim Water Profiles

The ERA-Interim daily profiles are calculated from 6 hourly data. Both the 6 hourly and daily data for profiles are created. The ERA-Interim data is housed locally at NCAR in the CISL Research Data Archive. There is a three month lag between the current date and when the data becomes available. The data is hosted on /glade/ and can be accessed through the data-access.ucar.edu server. The data can be found at: /glade/p/rda/data/ds627.0/ei.oper.an.pl/. The following steps should be used to pre-process the data:

1. Copy over the data from glade
2. Convert GRIB format files to NetCDF files using cnvrtNC.py
3. Create water profiles using ERAwaterPrf.py

The NCEP I re-analysis data are already daily averages. The grid resolution of NCEP I is less than ERA-interim. In addition ERA-Interim assimilates GPS occultation data. It is preferable to use ERA-Interim over NCEP I. The program to create water profiles from NCEP I data is NCEPwaterPrf.py.

Program	Description
cnvrtNC.py	Program to convert ERA-Interim GRIB files to NetCDF files
ERAwaterPrf.py	Program to extract daily averaged water profiles from ERA-Interim
NCEPwaterPrf.py	Program to create daily water profiles from NCEP I

4.3 Retrieved Water Profiles

For all sites (MLO,TAB, and FL0) water is retrieved when available. This water can be used as a prior for other retrievals. The program retWaterPrf.py creates w-120.YYYY MMDD.HHMMSS.v99 for each retrieval. These files are stored in the data directories (/data1/tab,mlo,fl0/). A daily average of these profiles can be created using the program retWaterPrfDaily.py. These daily averages are also stored in the main data directories (/data1/tab,mlo,fl0/).

Program	Description
retWaterPrf.py	Program to create water profiles from water retrieval
retWaterPrfDaily.py	Program to create daily average profiles from water retrievals

4.4 Steps for Pre-Processing

- Download OPUS and ancillary data (This is done automatically)
- Check OPUS spectra
- Copy spectra from /ya4/id/(mlo,tab,fl0) to /data1/(mlo,tab,fl0)
- Create initial database
- Format house data
- Format external station data
- Create appended spectral database
- Create co-added spectral database file if necessary
- Create Altitude, Pressure, and Temperature profiles (ZPT.nmc.120)
- Create water profiles (v1,v2,v3,v4,v5,v99)

5 Processing

5.1 Layer0

The purpose of Layer0 is to run a single retrieval. The program sfit4Layer0.py runs layer 0. This program is called with command line arguments. There is no input file.

Program	Description
sfit4Layer0.py	Program to run layer 0 using command line arguments

5.2 Layer1

The purpose of Layer1 is to batch process multiple or many retrievals. Layer1 requires an input file to specify retrieval options such as date range, input/output directory, etc. In addition, from Layer1 a user can create plots of an individual retrieval. The layer one processing environment serves to do the following:

- Create a directory structure to organize the output data
- Generate the necessary input files to run SFIT core code
- Execute the SFIT core code
- Conduct error analysis on output

The following figure shows the input/output flow control for layer 1 processing.

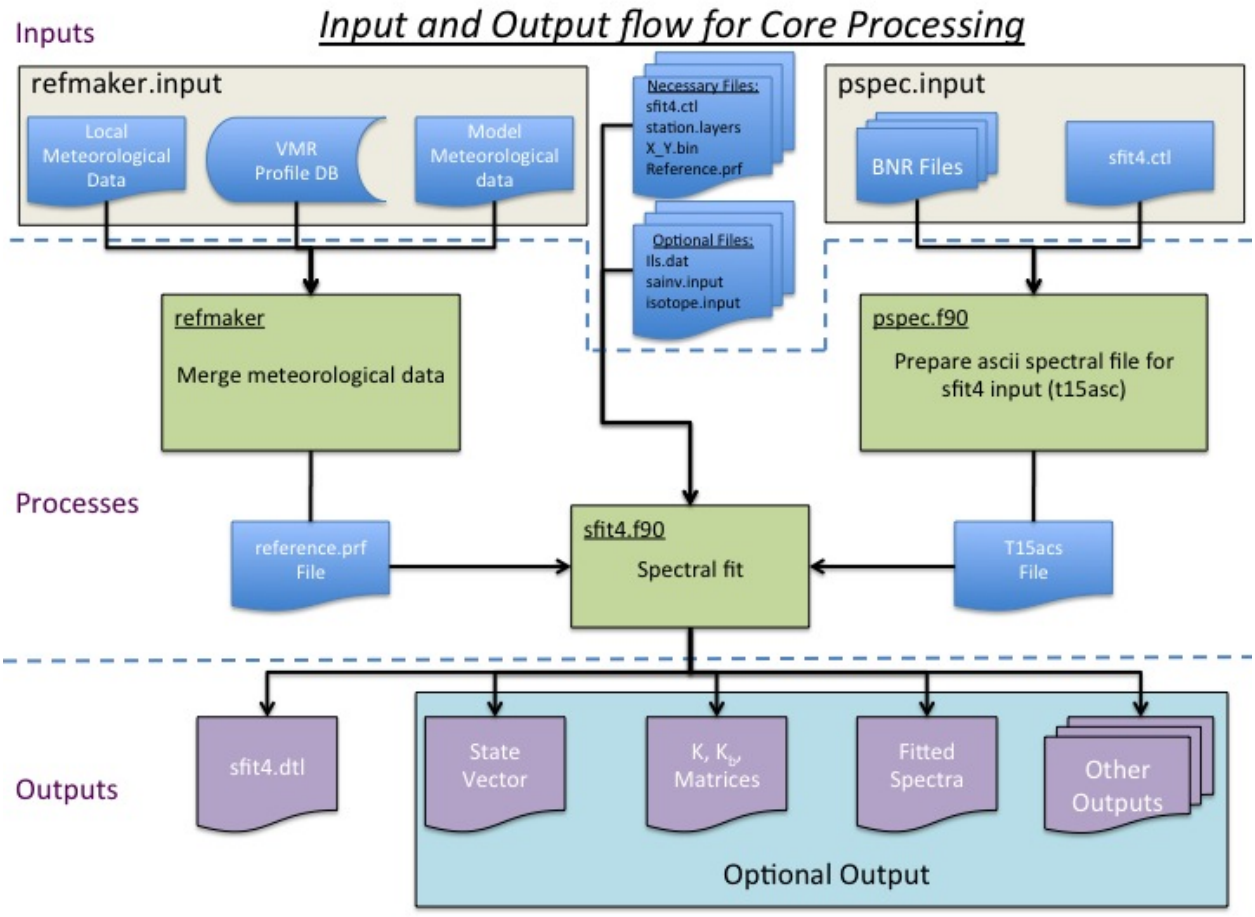

Figure 1: A visual representation of the processing flow.

Program	Description
sfit4Layer1.py	Program to run layer 1
stat_input.py	Input file for layer 1
mkListFile.py	Program to create list file from retrieval set (for IDL)

6 Post-Processing

6.1 Plotting

One can plot individual retrievals or an entire set of retrievals. There are no filtering options for a single retrieval; however, for a set of retrieval there are multiple parameters that one can filter on such as RMS, DOFs, date, etc. The program pltRet.py creates plots for a single retrieval and only requires command line arguments. The program pltSet.py plots an entire

set of retrievals and requires an input file (setInput.py).

Program	Description
pltRet.py	Program to plot individual retrieval using command line arguments
pltSet.py	Program to plot multiple retrievals using an input file
setInput.py	Input file for pltSet.py

6.2 HDF Creation

In order to archive retrievals of NDACC gases one must put the data in a GEOMS compliant HDF file (Link). There is a python routine that converts data to GEOMS HDF4 format. This package of code will also write HDF5 files. In order to run this code there are several software packages that must be install prior to use:

- The latest HDF4 libraries should be installed on your computer. This library and information on the install can be found at: HDF4
- If you wish to write HDF5 files, install the python package: h5py (IRWG / GEOMS files are only HDF4.)
- You will also need the python numpy package

The following are a list of files used in the creation of the HDF files:

- hdfBaseRetDat.py
- hdfCrtFile.py
- hdfInitData.py
- hdfsaveXXX.py
- HDFCreate.py
- input_HDFCreate.py

The only files that you will need to modify are hdfsaveXXX.py, input_HDFCreate.py, and hdfInitData.py. Here is a description of these files:

- HDFCreate.py – This is the main routine to create an object and create an HDF file and needs the input file input_HDFCreate.py. The input file needs to be modified accordingly.
- hdfsaveXXX.py – This file contains all the global and variable attributes or meta-data. The XXX is normally the a three letter ID for each site, e.g., hdfsaveMLO.py for Mauna Loa. You will need to modify the strings in this file to reflect the specifics of your group, instrument and retrieval process. Remember the formatting of the strings for GEOMS files is VERY specific (e.g. space and capitalization).

- hdfInitData.py – This file is the interface between your data and the HDF file. Everyone has data in a specific format so you will need to define a function that takes that data from that format and fills the appropriate class attributes. Currently there are three example interfaces in this file, although only the python interface is maintained:
  - initIDL – This interface takes data in from an IDL save file. Note the IDL save file has a specific structure (this idl program is available on request.)
  - initPy – This interface can take data using python functions. This interface has not been developed
  - initDummy – This is a dummy interface which will create dummy (FillValue) data to go into the HDF file.

To run the code create HDF4 type:
python HDFCreate.py -i input_HDFCreate.py

To run the code create h5 type:
python HDFCreate.py -i input_HDFCreate.py -h

Please don't hesitate to email us with questions or comments.

Note: An important input is the database file, the variables from the DB are: Lat, Lon, Duration, Instrument Altitude, RH, Wind, and Solar Azimuth. All other variables come from the input/output files in the sfit4 directory.

A note on writing data to HDF file:

First, a brief description of the difference between row-major (column is fastest running index) and column-major (row is the fastest running index):

Row-major and column-major are methods for storing multidimensional arrays in linear memory. For example, the C language follows row-major convention such that a 2x3 C matrix:

$$C[2,3] = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

will be written into linear memory such as: 1,2,3,4,5,6. The rows are written contiguously. The columns are the fastest running index.

In Fortran, a 2x3 matrix:

$$F[2,3] = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

will be written into linear memory such as: 1,4,2,5,3,6. The columns are written contiguously. The rows are the fastest running index.

How does this translate to higher level dimensions?

For column-major convention (Fortran) the fastest running index is furthest left index. For row-major convention (C) the fastest running index is the furthest right index.

What does this mean for writing to HDF?

HDF uses C storage conventions. It assumes row-major (or the column is the fastest running index). The HDF read and write codes also ensure that the fastest running index is consistent no matter which program (Fortran or C) reads/writes the data. This has implications if you are writing to an HDF file using the Fortran wrapper. From the HDF documentation:

"When a Fortran application describes a dataspace to store an array as A(20,100), it specifies the value of the first dimension to be 20 and the second to be 100. Since Fortran stores data by columns, the first-listed dimension with the value 20 is the fastest-changing dimension and the last-listed dimension with the value 100 is the slowest-changing. In order to adhere to the HDF5 storage convention, the HDF5 Fortran wrapper transposes dimensions, so the first dimension becomes the last. The dataspace dimensions stored in the file will be 100,20 instead of 20,100 in order to correctly describe the Fortran data that is stored in 100 columns, each containing 20 elements."

The Fortran wrapper transposes the matrix before it is written to HDF. So the Fortran matrix:

$$F[3,2] = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Is written to the HDF file as:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

If read using the C wrapper the matrix would look like:

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

This makes the values in the fastest running index consistent between Fortran and C. For Fortran the fastest running index are the row (1,2,3) (4,5,6) and for C the fastest running index is the column (1,2,3) (4,5,6). Transposing the matrix before writing and after reading in the Fortran wrapper ensures that the same values are in the fastest running index for Fortran as in C, even though these are different indices in terms of math matrix.

The higher-level scripting languages such as Python and IDL use the C set wrappers, so this is not an issue for them; however, if you are using Matlab to write the data this WILL be an issue. Matlab follows column-major convention (or the rows are the fastest running index). See Matlab documentation.

In terms of NDACC GEOMS HDF files this can be an issue for the averaging kernel (AVK) since this matrix is square. The standard for the GEOMS format is that the columns of the AVK (as described in Rodgers, 2000 pg) should be the fastest running index.

So, if you are using column-major (Fortran, Matlab) the dimensions of your AVK when you write to HDF should be:

$AVK[layer\_Index, Kernel_index, Datetime\_index]$

If you are using row-major (C, Python, IDL) the dimensions of your AVK when you write to the HDF should be:

$AVK[Datetime\_index, Kernel_index, layer\_index]$

If you have any doubt about how your data is written download either HDF-View or Panalopy or use HDP to view the HDF file you have written. These use C libraries to read the data. When you view the AVK with these programs it should have the following dimensions: $[datetime, kernel, altitude]$.

7 Program List

The following is a list of programs along with a description.

7.1 Pre-Processing

7.2 Spectral Database

- mkSpecDB.py
  - *Description:* This is the main program to create the spectral database. This program will append a spectral database if one already exists or create a new file. The spectral databases are text files space deliminated. The program looks for .bnr files in the date directories for a given input directory and runs ckopus. The output from ckopus is used to create the initial database. One must have a

working version of ckopus.c on their computer in order to create the database. A seperate database is created for each year. This program uses an input file.

- *Dependencies:* This program requires ckopus.c and an input file (specDBInputFile.py).
- *Invocation:* mkSpecDB.py -i specDBinputFile.py
- *Output:* Initial spectral database file. Current location of spectral databases: /data/Campaign/(MLO,TAB)/Spectral_DB/

- station_house_reader.py
  - *Description:* This program reformats the house data log files from MLO and TAB into a single format. The inputs are specified directly in the program file. Even though one can specify a start and stop year, this program should be run for only one year at a time.
  - *Dependencies:* This program requires HouseReaderC.py which tells the program how to read the various formats of the MLO and TAB house data.
  - *Invocation:* station_house_reader.py
  - *Output:* Reformated house log file data. Current directory of reformated house log data: /data/Campaign/(MLO,TAB)/House_Log_Files/

- HouseReaderC.py
  - *Description:* This program contains the formats for MLO and TAB house log files.
  - *Dependencies:* None
  - *Invocation:* None
  - *Output:* None

- HouseDataLog.xlsx
  - *Description:* This program contains a description of the house log files.
  - *Dependencies:* None
  - *Invocation:* None
  - *Output:* None

- pullAncillaryData.py
  - *Description:* This program uses the system call "wget" to get NCEP nmc, NCEP re-analysis, EOL, and CMDL data. The output data directories are specified directly in the program. Command line argument allows one to specify the particular date to get data for. This program is set on a cron tab to download data every day. If no date is specified current day is used.

  - *Dependencies:* None
  - *Invocation:* pullAncillaryData.py [-d YYYYMMDD]
  - *Output:* Various directories. See program for output directories.

- read_FL0_EOL_data.py
  - *Description:* This program re-formats the EOL data from netCDF data into simple text data to be used by the append spectral database program. All inputs are specified within the program. One year is processed at a time.
  - *Dependencies:* None
  - *Invocation:* read_FL0_EOL_data.py
  - *Input:* Input NetCDF files are currently located at: data1/ancillary_data/fl0/eol/flab.YYYYMMDD.cdf
  - *Output:* Output data is currently written to: data1/ancillary_data/fl0/eol/fl0_met_data_YYYY.txt

- appendSpecDB.py
  - *Description:* This program appends already created spectral database files with house and station outside pressure, temperature, and relative humidity values. The inputs are specified with an input file (appndSpecDBInputFile.py). The spectral databases for each year are appended. After the new appended spectral databases are created, the user should concatenate these together into a single database.
  - *Dependencies:* Requires an input file (appndSpecDBInputFile.py)
  - *Invocation:* appndSpecDBInputFile.py -i appndSpecDBInputFile.py
  - *Output:* Appended spectral database file. Current location of spectral databases: /data/Campaign/(MLO,TAB)/Spectral_DB/

- mkCoadSpecDB.py
  - *Description:* This program creates co-added bnr files (.bnrc) and a co-added spectral database file. The inputs to the program are specified through an input file (CoadSpecDBInputFile.py). The mkCoadSpecDB.py program reads in a HRspDB_loc_YYYY.dat data file and creates a CoaddspDB_loc_YYYY.dat output file. One can either run the co-add program on a conglomerated spectral database file (e.g. HRspDB_fl0_2010_2014.dat) or on individual year file (e.g. HRspDB_fl0_2014.dat) and then conglomerate all the years. When running the co-add program all previous output files with the same name should be removed.
  - *Dependencies:* Requires an input file (CoadSpecDBINputFile.py) and coad.c
  - *Invocation:* mkCoadSpecDB.py -i CoadSpecDBInputFile.py
  - *Input:* Appended spectral database file (e.g. HRspDB_fl0_2014.dat).
  - *Output:* Co-added spectral database file (e.g. CoaddspDB_fl0_2014.dat).

7.3 Reference Profiles

- NCEPnmnFormat.py
  - *Description:* For an individual station this program re-formats the NCEP nmc pressure and temperature data into a single file by year. This program takes an input file (NCEPinputFile.py)
  - *Dependencies:* Requires an input file (NCEPinputFile.py)
  - *Invocation:* NCEPnmnFormat.py -i NCEPinputFile.py
  - *Input:* The input NCEP nmc data is currently located at /data/ancillary_data/NCEP_NMC/
  - *Output:* The re-formated NCEP nmc data is currently located at /data/Campaign/(MLO,TAB,FL0)/NCEP_nmc/
- MergPrf.py
  - *Description:* This program creates ZPT files from NCEP nmc data and water files from WACCM data. It requires the use of an input file (mergprfInput.py)
  - *Dependencies:* Requires an input file (mergprfInput.py)
  - *Invocation:* MergPrf.py -i mergprfInput.py
  - *Input:* The input NCEP nmc data is currently located at /data/ancillary_data/NCEP_NMC/. The input WACCM data is currently at /data/Campaign/(MLO,TAB,FL0)/waccm/
  - *Output:* The profiles are stored with the bnr data in the daily directories /data1/(mlo,tab,fl0)/
- cnvrtNC.py
  - *Description:* This program converts the ERA-Interim GRIB files into NetCDF files. This program calls the linux program ncl_convert2nc to convert the GRIB file to NetCDF. The inputs are directly specified in the program. The NetCDF files are easily readable by python.
  - *Dependencies:* Requires installation of linux program ncl_convert2nc.
  - *Invocation:* cnvrtNC.py
  - *Input:* The input ERA-Interim (GRIB) data is currently located at /data1/ancillary_data/ERAdata/
  - *Output:* The output ERA-Interim (NetCDF) data is currently located at /data1/ancillary_data/ERAdata/
- ERAwaterPrf.py
  - *Description:* This program creates water profiles from the ERA-Interim data. The inputs are directly specified in the program. It interpolates a location and altitude profile to the ERA-Interim grid. It creates daily averages; however, it also stores the 6 hourly PV, Temperature, and Q.

  - *Dependencies:* None
  - *Invocation:* ERAwaterPrf.py
  - *Input:* The input ERA-Interim (NetCDF) data is currently located at /data1/ancillary_data/ERAdata/
  - *Output:* The output water profiles are stored with the bnr data in the daily directories /data1/(mlo,tab,fl0)

- NCEPwaterPrf.py
  - *Description:* This program creates water profiles from the NCEP re-analysis data. The inputs are directly specified in the program. It interpolates a location and altitude profile to the NCEP grid. Initial NCEP re-analysis data is already daily averaged.
  - *Dependencies:* None
  - *Invocation:* NCEPwaterPrf.py
  - *Input:* The input NCEP re-analysis (NetCDF) data is currently located at /data 1/ancillary_data/NCEPdata/
  - *Output:* The output water profiles are stored with the bnr data in the daily directories /data1/(mlo,tab,fl0)

- retWaterPrf.py
  - *Description:* This program creates water a priori profiles retrieved water profiles. The inputs are directly specified in the program. The a priori water profiles are time stamped (date and time) corresponding with measurement time.
  - *Dependencies:* None
  - *Invocation:* retWaterPrf.py
  - *Input:* The input is the location of the retrieved water data.
  - *Output:* The output water profiles are stored with the bnr data in the daily directories /data1/(mlo,tab,fl0)

- retWaterPrfDaily.py
  - *Description:* This program creates daily averaged water profiles from the a priori water files created from the program retWaterPrf.py. The inputs are specified directly in the program.
  - *Dependencies:* None
  - *Invocation:* retWaterPrfDaily.py
  - *Input:* The input water profiles are located with the bnr data in the daily directories /data1/(mlo,tab,fl0)
  - *Output:* The output water profiles are stored with the bnr data in the daily directories /data1/(mlo,tab,fl0)

7.4 Processing

- sfit4Layer0.py
  - *Description:* This program runs individual retrievals using sfit4. The inputs for this program are specified through command line arguments.
  - *Dependencies:* The program depends on the following files: sfitClasses.py and Layer1Mods.py
  - *Invocation:* They are several command line flags to use with this program: sfit4Layer0.py -f <str> [-i <dir> [-b <dir/str>]
    - -i Data directory. This is optional. The default directory is the current working directory
    - -f Run flags: h = hbin, p = psepc, s = sfit4, e = error analysis, c = clean directory
    - -b sfit binary directory: This is optional. The default is hard-coded in the main program. Also accepts v1, v2, etc
  - *Input:* The input is the specified data directory or the current working directory
  - *Output:* The output is also the specified data directory or the current working directory

- sfit4Layer1.py
  - *Description:* This program runs multiple retrievals using sfit4. The inputs are specified through an input file and some command line arguments. If one pauses during the processing using the -P command you can plot intermediate results or re-processes the last retrieval done.
  - *Dependencies:* This program depends on the following programs: sfitClasses.py, dataOutClass.py, Layer1Mods.py
  - *Invocation:* sfit4Layer1.py -i <InputFile> -l -L0 -P <int> -?
    - -i Filename and path of input file
    - -l Flag to create log file. Path for log files is specified in input file.
    - -L 0/1: Flag to create output list file.
    - -P <int>: Pause processing after retrieval number <int>. So -P1 would pause after the first retrieval.
    - -? Show all flags
  - *Input:* Inputs paths and directories are specified in the input file.
  - *Output:* Output paths and directories are specified in the input file

- mkListFile.py
  - *Description:* This program creates a list file of all the directories with retrievals and some meta-data for the IDL program gather.pro to create a sav file.

  - *Dependencies:* None
  - *Invocation:* mkListFile.py -i <file> -N <file> -d <dir> -?
    * -i Filename and path of layer1 input file
    * -N Filename and path to output list file
    * -d Base directory of data
    * -? Show all flags
  - *Input:* Layer1 input file, Base directory to data
  - *Output:* Name of list file

7.5 Post-Processing

- pltRet.py
  - *Description:* This program creates plots for an individual retrieval. The input, which is the directory of the retrieval, is given as a command line argument. If no command line argument is specified the current working directory is used.
  - *Dependencies:* dataOutClass.py
  - *Invocation:* pltRet.py -i <dir> -?
    * -i Directory of retrieval
    * -? Show all flags
  - *Input:* Directory of retrieval
  - *Output:* Various plots
- pltRet.py
  - *Description:* This program creates plots for a set of retrievals. There are multiple filters and flags that can be used to filter the data. These flags and filters are specified in the input file
  - *Dependencies:* dataOutClass.py
  - *Invocation:* pltSet.py -i setInput.py [-?]
    * -i Input file name
    * -? Show all flags
  - *Input:* Input file
  - *Output:* Various plots
- HDFCreate.py
  - *Description:* This is the main file for creating HDF files from data. In this program you call the HDFsave object and create an HDF file. From here you define the directory to write the HDF file, whether to write the file using single or double precision, and whether to write an HDF4 or HDF5 file (IRWG / GEOMS are single precision.). The inputs are directly specified in the file.

- *Dependencies:* hdfsave.py
- *Invocation:* HDFCreate.py -i inputHDFCreate.py
  - -i Input file name
  - -? Show all flags
- *Input:* Input file
- *Output:* HDF file(s)