Post-Processing

Description of SFIT pos-Processing Environment: sfit4-procEnvPost_Ortega_v1.pdf

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. Plotting routines are distributed within the sfit-processing-environment. The program pltRet.py creates plots for a single retrieval and only requires command line arguments. Using the option pltRet.py -S would save plots into a pdf file. Type the following to see all flags:

```
$ python3 pltRet.py -?
```

The program pltSet.py plots an entire set of retrievals and requires an input file (setInput.py).

$ python3 pltSet.py -i setInput.py

The setInput.py needs to be modified accordingly.

Conversion to HDF

We now have 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.)
- For writing HDF4 files you need to install the python pyhdf package. We have had to 'tweak' this package for use with GEOMS file format. Installation instructions for this package are in the INSTALL text file
- You will also need pythons numpy package

Once you have those packages installed you can download and run the python codes to create GEOMS HDF files. This packages is located in the sfit-processing-environment and is called HDFsave. This package contains five files:

1. hdfBaseRetDat.py
2. hdfCrtFile.py
3. hdfInitData.py
4. hdfsaveXXX.py
5. input_HDFCreate.py
6. HDFCreate.py

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

- HDFCreate.py – This is the main 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 a three letter id for the 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).* Also, HDFCreate.py will try to to find this file automatically according with the three letter id.
- 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:
    - 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.) - Not further supported.
    - initPy -- This interface can take data using python functions. This interface has not been developed. **This is recommended.**
    - initDummy -- This is a dummy interface which will create dummy (FillValue) data to go into the HDF file.

To run the code type:

```
$ python HDFCreate.py -i input_HDFCreate.py
```

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

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] =

1	2	3
4	5	6

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) =

1	2	3
4	5	6

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)=

1	4
2	5
3	6

Is written to the HDF file as:

1	2	3
4	5	6

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

1	2	3
4	5	6

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].

Plotting HDF

One can plot a set of HDF files. The needed python files to plot HDF are in the HDFread folder. The program pltHDF.py creates plots for a single or multiple years as specified in the input file. The program pltHDF.py requires an input file (input_HDFRead.py).

The inputs and flags needed in input_HDFRead.py are self explanatory. To create plots using HDF files use the following syntax:

```
$ python3 pltHDF.py -i input_HDFRead.py
```

Link	Version	Release Date	Description
sfit4_procEnvPost_Ortega_v1	3.0	May 12th, 2020	Description of post analysis