Testing the Python aLMI code

Author: Eric Lehmann, CSIRO
E-mail: eric.lehmann@csiro.au
Date: March 2020

Summary

This notebook was written to carry out initial testing of the aLMI Python code on the VDI.

The original Python code was cloned from https://bitbucket.csiro.au/projects/CMAR_RS/repos/ereefs-lmi-python/ (https://bitbucket.csiro.au/projects/CMAR_RS/repos/ereefs-lmi-python/). It has been converted from Python 2 to Python 3 using `2to3` functionality, and subsequently updated / debugged to enable its use on the NCI's high-performance computing facility.

This notebook was created / launched after executing the following commands in a VDI terminal:

```
module use /g/data/v10/public/modules/modulefiles
module load dea
export PYTHONPATH=/path/to/install/dir/lib/python3.6/site-packages/:$PYTHONPATH
jupyter lab    # or jupyter notebook
```

The command `module load dea` automatically loads up Python 3.6.7. See also below for further notes on exporting the Python path variable.

Preliminaries

The aLMI code below requires the Python package `pyhdf` , which is not installed by default on the VDI or NCI (Gadi).

This package thus requires a manual install, which can be performed as follows. First, open a new terminal (NCI or VDI) and type in the following commands:

```
module load python3/3.6.7      # VDI: use python3/3.6.7 -- Gadi: use python3/3.7.4
module load hdf4/4.2.13        # VDI: use hdf4/4.2.13 -- Gadi: use hdf4/4.2.14
pip3 install -v --no-binary :all: --prefix=/path/to/install/dir pyhdf     # replace prefix string with relevant path
```

The `pyhdf` package can be installed, for instance, in the user's home directory. For instance, in Eric's installation of `pyhdf` , the prefix string `/path/to/install/dir` was replaced with `/home/599/eal599/.local/` .

Once the `pyhdf` package is installed, the following command needs to be executed prior to running the aLMI code:

```
export PYTHONPATH=/path/to/install/dir/lib/python3.6/site-packages/:$PYTHONPATH      # VDI: use python3.6 -- Gadi: use python3.7
```

(Here again, replace the string `/path/to/install/dir` with the relevant path to the `pyhdf` install directory -- same as above.)

This allows Python to locate the relevant `pyhdf` files when needed during execution of the aLMI code.

Code testing

Code and parameter files

The main Python aLMI function (`aLMI_main.py`) relies on three input arguments:

- input file of reflectance data (.nc or .hdf file)
- .ini file of input parameters for the aLMI run, which, among others, contains the path to the SIOP sets file (.nc file) to use
- path and name of the output file (.nc or .hdf) where the aLMI results are to be saved.

Executing the main aLMI function also calls upon a series of subroutines provided in various .py files, as listed below.

In [1]:
```
SRC_DIR = ".."    # where the source code is located, one directory up

!ls -lh {SRC_DIR}/*.py
```

```
-rwxr--r-- 1 eal599 CSIRO  46K Mar 17 10:09 ../aLMI_main.py
-rwxr--r-- 1 eal599 CSIRO 1.9K Mar  4 09:05 ../calcCost.py
-rwxr--r-- 1 eal599 CSIRO 4.0K Mar  4 09:05 ../calc_kd_SD.py
-rwxr--r-- 1 eal599 CSIRO 1.3K Mar  4 09:05 ../calc_mu_d.py
-rwxr--r-- 1 eal599 CSIRO 1.1K Mar  4 09:05 ../calcReflForward.py
-rwxr--r-- 1 eal599 CSIRO 1.1K Mar  4 09:05 ../calcUBackward.py
-rwxr--r-- 1 eal599 CSIRO 3.8K Mar  4 09:05 ../calcUForward.py
-rwxr--r-- 1 eal599 CSIRO 1.8K Mar  4 09:05 ../calcY.py
-rwxr--r-- 1 eal599 CSIRO  20K Mar 17 14:01 ../chunkProcessLMI.py
-rwxr--r-- 1 eal599 CSIRO 7.0K Mar  4 16:25 ../configUtils.py
-rwxr--r-- 1 eal599 CSIRO  48K Mar 12 15:30 ../generic_io.py
-rwxr--r-- 1 eal599 CSIRO 5.6K Mar 12 13:21 ../lmi_io.py
-rwxr--r-- 1 eal599 CSIRO  878 Mar  4 09:05 ../RrsAboveToBelow.py
-rwxr--r-- 1 eal599 CSIRO 2.3K Mar  4 09:05 ../RrsAboveToBelow_test.py
-rwxr--r-- 1 eal599 CSIRO  880 Mar  4 09:05 ../RrsBelowToAbove.py
-rwxr--r-- 1 eal599 CSIRO 8.1K Mar  4 09:05 ../SIOP_sets_load.py
-rwxr--r-- 1 eal599 CSIRO 4.5K Mar 17 10:07 ../svd_LMI.py
-rwxr--r-- 1 eal599 CSIRO 5.5K Mar  4 09:05 ../var_dump.py
-rwxr--r-- 1 eal599 CSIRO 3.0K Mar 16 10:00 ../wavelengthsToVarNames.py
```

With the Eric's VDI setup, the .ini files can be found in a sub-directory `ini_files` here:

In [2]:
```
!ls -lh {SRC_DIR}/ini_files
```

```
total 16K
-rw-r--r-- 1 eal599 CSIRO 3.0K Mar  4 12:46 aLMI_config.ini
-rw-r--r-- 1 eal599 CSIRO 3.2K Mar 12 14:26 aLMI_config_VDItest.ini
-rw-r--r-- 1 eal599 CSIRO 3.1K Mar 13 16:45 aLMI_config_VDItest_nc.ini
-rw-r--r-- 1 eal599 CSIRO 2.8K Jan 20  2016 MODIS_below_config.ini
```

And some example input data files (file of remote sensing reflectances, and SIOP sets) can be accessed here:

In [3]:
```
!ls -lh /g/data/r78/eal599/iWQ_aLMI/data
```

```
total 1.2G
-rw-r--r-- 1 eal599 r78 163M Mar  4 15:22 A20120403_0410.20130805213444.L2.ANN_P134_V20140
704.hdf
-rw-r--r-- 1 eal599 r78  11K Mar  4 15:26 A20120403_0410.20130805213444.L2.ANN_P134_V20140
704.hdf.dump
-rw-r--r-- 1 eal599 r78 413M Mar  4 15:22 A20120403_0410.20130805213444.L2.ANN_P134_V20140
704.MIM_CLT4_gLee_412_748.hdf
-rw-r--r-- 1 eal599 r78  28K Mar  4 15:27 A20120403_0410.20130805213444.L2.ANN_P134_V20140
704.MIM_CLT4_gLee_412_748.hdf.dump
-rw-r--r-- 1 eal599 r78  96M Mar  4 15:25 A20120403_0410.20150928173107.L2OC_BASE.ANN_P134
_V20140704.nc
-rw-r--r-- 1 eal599 r78  11K Mar 13 16:52 A20120403_0410.20150928173107.L2OC_BASE.ANN_P134
_V20140704.nc.dump
-rw-r--r-- 1 eal599 r78 2.3K Mar  4 15:25 A20120403_0410.20150928173107.L2OC_BASE.ANN_P134
_V20140704.nc.log
-rw-r--r-- 1 eal599 r78 499M Mar  4 15:25 A20120403_0410.20150928173107.L2OC_BASE.ANN_P134
_V20140704.PyLMI.nc
-rw-r--r-- 1 eal599 r78  93K Mar  4 15:25 A20120403_0410.20150928173107.L2OC_BASE.ANN_P134
_V20140704.PyLMI.nc.log
-rw-r--r-- 1 eal599 r78  37K Jan 19  2016 siops_10nm_all_CLT4.nc
-rw-r--r-- 1 eal599 r78  28K Dec 17  2015 siops_MODIS_all_CLT4.nc
```

These files are stored in /g/data due to the limited space available (2GB limit) in home directories on the VDI. All of these example files were found and collected from various subdirectories within `/g/data/u83/code/ereefs/` .

A summary of what these files are is provided below:

- `A20120403_0410.20130805213444.L2.ANN_P134_V20140704.hdf` : file of input reflectances (.hdf file)
- `A20120403_0410.20130805213444.L2.ANN_P134_V20140704.MIM_CLT4_gLee_412_748.hdf` : aLMI output file produced by the IDL code (for comparison)
- `A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.nc` : another file of input reflectances (.nc file)
- `A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.nc.log` : log file of the (ANN-based) atmospheric correction process that created the previous dataset
- `A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.PyLMI.nc` : aLMI output file produced by the Python code (for comparison)
- `A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.PyLMI.nc.log` : log file generated by aLMI (Python) while processing the previous dataset
- `... .dump` : text files showing the variables included in the corresponding data files above
- `siops_... .nc` : datasets of SIOP sets

All the files listed above (.ini, .nc, .hdf, etc.) can be used for the purpose of testing the Python code, as shown below.

Running the tests

"Unit tests"

In this notebook, we'll run some basic tests on each of the .py functions above to ensure that everything is running smoothly. Each of the `.py` functions listed above includes a section with some basic test code, which can be executed as follows.

(Note that all the .py code needs to be made executable if not already the case: `chmod u+x *.py`).

(Note2: in the bash code below, the construct `$?` basically queries the exit flag from the previously run code).

In [4]:
```
CHECK_OUTPUT_STR = "if [[ $? == 0 ]]; then echo \"TEST OK\"; else echo \"TEST FAILED\"; fi"
# command string to execute after running each test

!{SRC_DIR}/RrsAboveToBelow.py > ./log_files/RrsAboveToBelow.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [5]:
```
!{SRC_DIR}/RrsBelowToAbove.py > ./log_files/RrsBelowToAbove.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [6]:
```
!{SRC_DIR}/calcCost.py > ./log_files/calcCost.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [7]:
```
!{SRC_DIR}/calcReflForward.py > ./log_files/calcReflForward.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [8]:
```
!{SRC_DIR}/calcUBackward.py > ./log_files/calcUBackward.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [9]:
```
!{SRC_DIR}/calcY.py > ./log_files/calcY.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

In [10]:
```
!{SRC_DIR}/calc_mu_d.py > ./log_files/calc_mu_d.log; $CHECK_OUTPUT_STR
```
```
TEST OK
```

```
In [11]: !{SRC_DIR}/svd_LMI.py > ./log_files/svd_LMI.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [12]: !{SRC_DIR}/var_dump.py > ./log_files/var_dump.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

Some of the code files require some input:

```
In [13]: CONFIG_FILE = SRC_DIR + "/ini_files/aLMI_config_VDItest.ini"    # example config file
INPUT_FILE = SRC_DIR + "/configUtils.py " + CONFIG_FILE + " optionalParameters test_ANN_file"    # get an example input file from the config .ini file
VAR_PREFIX = "Rrs_"    # reflectance bands prefix string for this input file -- only used for testing!

!echo "Config file is: \"$CONFIG_FILE\""
!echo "Input file is: \"`$INPUT_FILE`\""
```

```
Config file is: "../ini_files/aLMI_config_VDItest.ini"
Input file is: "/g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20130805213444.L2.ANN_P134_V20140704.hdf"
```

```
In [14]: !{SRC_DIR}/configUtils.py $CONFIG_FILE > ./log_files/configUtils.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [15]: !{SRC_DIR}/calc_kd_SD.py $CONFIG_FILE > ./log_files/calc_kd_SD.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [16]: !{SRC_DIR}/calcUForward.py $CONFIG_FILE > ./log_files/calcUForward.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [17]: !{SRC_DIR}/SIOP_sets_load.py $CONFIG_FILE get_attrs > ./log_files/SIOP_sets_load.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [18]: !{SRC_DIR}/wavelengthsToVarNames.py `$INPUT_FILE` $VAR_PREFIX 412 685 550 > ./log_files/wavelengthsToVarNames.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

```
In [19]: !{SRC_DIR}/chunkProcessLMI.py $CONFIG_FILE $VAR_PREFIX > ./log_files/chunkProcessLMI.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

All seems OK so far. The resulting log files can be found in the following directory, and inspected for potential issues:

In [20]:
```
!ls -lh ./log_files/
```

```
total 200K
-rw-r--r-- 1 eal599 jr4  349 Mar 17 14:29 calcCost.log
-rw-r--r-- 1 eal599 jr4 2.8K Mar 17 14:29 calc_kd_SD.log
-rw-r--r-- 1 eal599 jr4  608 Mar 17 14:29 calc_mu_d.log
-rw-r--r-- 1 eal599 jr4  264 Mar 17 14:29 calcReflForward.log
-rw-r--r-- 1 eal599 jr4  209 Mar 17 14:29 calcUBackward.log
-rw-r--r-- 1 eal599 jr4 3.3K Mar 17 14:29 calcUForward.log
-rw-r--r-- 1 eal599 jr4 1.1K Mar 17 14:29 calcY.log
-rw-r--r-- 1 eal599 jr4 144K Mar 17 14:29 chunkProcessLMI.log
-rw-r--r-- 1 eal599 jr4 3.4K Mar 17 14:29 configUtils.log
-rw-r--r-- 1 eal599 jr4  156 Mar 17 14:29 RrsAboveToBelow.log
-rw-r--r-- 1 eal599 jr4  165 Mar 17 14:29 RrsBelowToAbove.log
-rw-r--r-- 1 eal599 jr4 4.0K Mar 17 14:29 SIOP_sets_load.log
-rw-r--r-- 1 eal599 jr4 1.4K Mar 17 14:29 svd_LMI.log
-rw-r--r-- 1 eal599 jr4  759 Mar 17 14:29 var_dump.log
-rw-r--r-- 1 eal599 jr4  337 Mar 17 14:29 wavelengthsToVarNames.log
```

Main aLMI routine -- short test

So, all the "unit tests" implemented to check the basic functionality of the above Python code files appear to execute properly.

This leaves the main aLMI routine `aLMI_main.py`, which can be tested as follows (takes a few minutes):

In [21]:
```
OUTPUT_FILE = "/g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main.hdf"   # save to /g/data due to large output file size

!{SRC_DIR}/aLMI_main.py `$INPUT_FILE` $CONFIG_FILE $OUTPUT_FILE -v > ./log_files/aLMI_main.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

So (...and after some code debugging!), the main aLMI routine executes without a hitch. The processing status of the aLMI code has been written to a `.log` file during execution:

In [22]:
```
!head -25 ./log_files/aLMI_main.log
```

```
aLMI_main.py:  startTime = 2020-03-17 03:29:25.432278 UTC
Verbosity level =  1
using lmi_io from /home/599/eal599/_Eric_VDI_/iWQ_aLMI/lmi_io.py
using lmi_io.generic_io from /home/599/eal599/_Eric_VDI_/iWQ_aLMI/generic_io.py
**** INPUT FILES ****
The input file to be used is:  /g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20130805213444.L2.ANN_P134_V20140704.hdf
The config file to be used is:  ../ini_files/aLMI_config_VDItest.ini
The output file to be created is:  /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main.hdf
****
configLoad:  found config file: ['../ini_files/aLMI_config_VDItest.ini']
getConfigOption:  has section 'inputParameters'
getConfigOption:  inputParameters has option 'inputSpectrumVarNames'
getConfigOption:  has section 'inputParameters'
getConfigOption:  inputParameters has option 'useWavelengths'
getConfigOption:  has section 'outputParameters'
getConfigOption:  outputParameters has option 'outWavelengthLabels'
getConfigOption:  has section 'inputParameters'
getConfigOption:  inputParameters has option 'tolerance'
getConfigOption:  has section 'inputParameters'
getConfigOption:  inputParameters has option 'SIOP_SETS_FILE'
getConfigOption:  has section 'ggParameters'
getConfigOption:  ggParameters has option 'g0'
getConfigOption:  has section 'ggParameters'
getConfigOption:  ggParameters has option 'g1'
getConfigOption:  has section 'inputParameters'
```

The output file of aLMI data generated by the code has also been written to the desired output file:

```
In [23]: !ls -lh $OUTPUT_FILE
```

```
-rw-r--r-- 1 eal599 r78 483M Mar 17 14:33 /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLM
I_main.hdf
```

We can create a text file of the various fields and variables contained within that aLMI output `.hdf` file, which can be further inspected from disk:

```
In [24]: CMD = "hdp dumpsds -h " + OUTPUT_FILE + " > " + OUTPUT_FILE + ".dump"
         print(CMD)
         !$CMD

         print("\n==== (Some of the) contents of the aLMI output file ====")
         !head -25 {OUTPUT_FILE}.dump
```

```
hdp dumpsds -h /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main.hdf > /g/data/r78/ea
l599/iWQ_aLMI/code_testing_out/aLMI_main.hdf.dump

==== (Some of the) contents of the aLMI output file ====
File name: /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main.hdf

File attributes:
         Attr0: Name = ATMCORR_VERSION
                 Type = 8-bit signed char
                 Count= 12
                 Value =     20140704
         Attr1: Name = ANN_CREATOR
                 Type = 8-bit signed char
                 Count= 16
                 Value = Thomas_Schroeder
         Attr2: Name = ANN_CREATION_DATE
                 Type = 8-bit signed char
                 Count= 24
                 Value = Sun Jul 20 04:36:09 2014
         Attr3: Name = LMI_VERSION
                 Type = 8-bit signed char
                 Count= 28
                 Value = Python aLMI, Version unknown
         Attr4: Name = CREATOR
                 Type = 8-bit signed char
                 Count= 6
                 Value = eal599
         Attr5: Name = CREATION_DATE
                 Type = 8-bit signed char
```

We can also inspect the contents of this `.hdf` file from Python as follows:

In [25]:

```
import sys
sys.path.insert(0,SRC_DIR)   # to be able to import lmi_io as a package
import lmi_io
SD_handle = lmi_io.generic_io.open(OUTPUT_FILE)

### Or alternatively:
# from pyhdf.SD import SD, SDC
# SD_handle = SD(OUTPUT_FILE, SDC.READ)

datasets_dic = SD_handle.datasets()
for idx,sds in enumerate(datasets_dic.keys()):
    print(idx,sds)
```

```
0 longitude
1 latitude
2 l2_flags
3 nn_flags
4 Rrs_412
5 Rrs_443
6 Rrs_488
7 Rrs_531
8 Rrs_547
9 Rrs_667
10 Rrs_678
11 Rrs_748
12 CHL_MIM
13 CDOM_MIM
14 NAP_MIM
15 SIOPindex
16 cost_Rrs_below
17 a_phy_MIM_441
18 a_CDOM_MIM_441
19 a_NAP_MIM_441
20 a_P_MIM_441
21 a_CDM_MIM_441
22 a_tot_MIM_441
23 bb_phy_MIM_551
24 bb_NAP_MIM_551
25 bb_P_MIM_551
26 Kd_par_MIM
27 Kd_490_MIM
28 SD_MIM
29 Rrs_MIM_412
30 Rrs_MIM_441
31 Rrs_MIM_488
32 Rrs_MIM_531
33 Rrs_MIM_551
34 Rrs_MIM_667
35 Rrs_MIM_678
36 Rrs_MIM_748
37 Delta_Rrs_below_MIM_412
38 Delta_Rrs_below_MIM_441
39 Delta_Rrs_below_MIM_488
40 Delta_Rrs_below_MIM_531
41 Delta_Rrs_below_MIM_551
42 Delta_Rrs_below_MIM_667
43 Delta_Rrs_below_MIM_678
44 Delta_Rrs_below_MIM_748
```

For illustration, let's have a closer look at one of the above datasets:

In [26]:

```
sds_obj = SD_handle.select('SIOPindex')    # select the dataset of SIOP indices
data = sds_obj.get()    # get the data from this dataset
print("Dataset size is (lines x pixels):", data.shape)
print("\nSome values from the 'SIOP index' dataset:")
print(data[0:10,1000:1010])
```

```
Dataset size is (lines x pixels): (2100, 1354)

Some values from the 'SIOP index' dataset:
[[-999 -999    3    0    3    3    3    3    3    3]
 [-999    3    3    0    3    3    3    0    0    3]
 [   3    3    3    0    3    3    3    3    0 -999]
 [   3    3    3    3    3    3    3    3    3 -999]
 [   3    3    3    3    3    3    3    3    0 -999]
 [   3    3    3    3    3    3    3    3    0    0]
 [   1    3 -999 -999    5    3    3    0    1    1]
 [-999 -999 -999 -999 -999    0    1    1    1    1]
 [-999 -999 -999 -999 -999    0    6    6    6    1]
 [-999 -999 -999 -999 -999    0    6    6 -999 -999]]
```

For comparison, let's have a look at the contents of the corresponding (example) IDL-based aLMI output (which was generated prior to running this notebook):

In [27]:

```python
base_dir = "/g/data/r78/eal599/iWQ_aLMI/data/"
IDL_OUTPUT_FILE = base_dir + "A20120403_0410.20130805213444.L2.ANN_P134_V20140704.MIM_CLT4_gLee_412_748.hdf"    # prepared earlier...

SD_handle = lmi_io.generic_io.open(IDL_OUTPUT_FILE)
datasets_dic = SD_handle.datasets()
for idx,sds in enumerate(datasets_dic.keys()):
    print(idx,sds)
```

```
0 longitude
1 latitude
2 l2_flags
3 nn_flags
4 Chl_MIM
5 CDOM_MIM
6 Nap_MIM
7 siop_MIM
8 dR_MIM
9 n_bands
10 Kd_par_MIM
11 Kd_490_MIM
12 SD_MIM
13 a_phy_MIM_441
14 a_CDOM_MIM_441
15 a_NAP_MIM_441
16 a_P_MIM_441
17 a_CDM_MIM_441
18 a_tot_MIM_441
19 bb_phy_MIM_551
20 bb_NAP_MIM_551
21 bb_P_MIM_551
22 a_budget_MIM_441
23 Rrs_MIM_412
24 Rrs_MIM_441
25 Rrs_MIM_488
26 Rrs_MIM_531
27 Rrs_MIM_551
28 Rrs_MIM_667
29 Rrs_MIM_678
30 Rrs_MIM_748
31 Delta_Rrs_MIM_412
32 Delta_Rrs_MIM_441
33 Delta_Rrs_MIM_488
34 Delta_Rrs_MIM_531
35 Delta_Rrs_MIM_551
36 Delta_Rrs_MIM_667
37 Delta_Rrs_MIM_678
38 Delta_Rrs_MIM_748
```

The IDL outputs look similar / identical to those generated by the Python version of the aLMI code.

Main aLMI routine -- in-depth test

The shell script `aLMI_main.sh` (within the current `testing` sub-directory) carries out the same test of the main aLMI routine as done above, but in addition, it also creates the `.dump` file of the aLMI output file contents, does a profiling of the code, and provides a comparison with the corresponding IDL-based aLMI outputs (in case a dataset of IDL outputs is provided in the `.ini` configuration file, as parameter `test_IDL_LMI_file`).

Let's run this shell script (will take a few minutes):

```
In [28]: CONFIG_FILE = SRC_DIR + "/ini_files/aLMI_config_VDItest.ini"    # config file
         INPUT_FILE = SRC_DIR + "/configUtils.py " + CONFIG_FILE + " optionalParameters test_ANN_fil
         e"    # get the aLMI input file from the config file
         OUTPUT_FILE = "/g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main2.hdf"    # where the d
         ata will be written to

         # for illustration only:
         IDL_file = SRC_DIR + "/configUtils.py " + CONFIG_FILE + " optionalParameters test_IDL_LMI_fi
         le"    # IDL output file from the config file
         !echo "The IDL file of aLMI results is: \"`$IDL_file`\""

         !./aLMI_main.sh `$INPUT_FILE` $CONFIG_FILE $OUTPUT_FILE -v > ./log_files/aLMI_main2.log; $CH
         ECK_OUTPUT_STR
```

```
The IDL file of aLMI results is: "/g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20130805
213444.L2.ANN_P134_V20140704.MIM_CLT4_gLee_412_748.hdf"

real    6m33.317s
user    5m18.828s
sys     1m14.320s
TEST OK
```

The profiling information, as well the comparison between the Python-based aLMI outputs and the IDL results, can be found in the outputs from `almi_main.sh` , which were logged to a `.log` file. A selection of this `.log` file output (here showing some of the IDL vs. Python aLMI comparison results) is shown below:

```
In [29]: !tail -n32 ./log_files/aLMI_main2.log
```

```
v = ['bb_NAP_MIM_551', -0.2, 'bb_NAP_MIM_551', -999.0, -0.1]
number of elements = 2843400
        ok lengths differ: 271187 259727 using PY_ok
IDL_min = -0.1 IDL_max = 1.3649466 IDL_median = 0.00073116715 IDL_std = 0.023767171
 PY_min = 0.00023633623  PY_max = 2.8820338  PY_median = 0.0021008593  PY_std = 0.04557481
8
RMS error = 0.022637488644467022 RMS relative error = 30.960757141384338

v = ['Kd_par_MIM', -0.2, 'Kd_par_MIM', -999.0, -0.1]
number of elements = 2843400
        ok lengths differ: 2843400 259727 using PY_ok
IDL_min = -1.0 IDL_max = 7.855482 IDL_median = 0.48106474 IDL_std = 0.17036988
 PY_min = 0.5395853  PY_max = 14.654421  PY_median = 0.5686032  PY_std = 0.2657226
RMS error = 0.14466218074498735 RMS relative error = 0.3007125026097327

v = ['Kd_490_MIM', -0.2, 'Kd_490_MIM', -999.0, -0.1]
number of elements = 2843400
        ok lengths differ: 2843400 259727 using PY_ok
IDL_min = -1.0 IDL_max = 8.418976 IDL_median = 0.059856065 IDL_std = 0.19681817
 PY_min = 0.02605452  PY_max = 18.32756  PY_median = 0.050359108  PY_std = 0.3505907
RMS error = 0.16039725880341296 RMS relative error = 2.6797160795983115

v = ['SD_MIM', -0.2, 'SD_MIM', -999.0, -0.1]
number of elements = 2843400
        ok lengths differ: 2843400 259727 using PY_ok
IDL_min = -1.0 IDL_max = 16.012741 IDL_median = 7.414263 IDL_std = 3.3448825
 PY_min = 0.035769753  PY_max = 22.419275  PY_median = 7.2462034  PY_std = 3.2709904
RMS error = 0.6907553799654296 RMS relative error = 0.09316575379696354
Total RMS relative error = 224.42625288063834

./compare_LMIs.py done
./aLMI_main.sh:  done
```

"Unit tests" -- revisited

At the beginning of this notebook, we have executed all the Python code's "unit tests" individually, one after the other.

The shell script `main_test.sh` (also within the current `testing` sub-directory) automates the processing of running all these tests at once (will take a few minutes due to the `aLMI_main.py` test).

In [30]:

```
OUTPUT_FILE = "/g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main3.hdf"
!./main_test.sh `$INPUT_FILE` $CONFIG_FILE $OUTPUT_FILE $VAR_PREFIX > ./log_files/main_test.log; $CHECK_OUTPUT_STR
```

```
TEST OK
```

As expected, all tests have passed successfully. Here's confirmation of this:

In [31]:

```
!cat ./log_files/main_test.log
```

```
***
*** Running all tests, including aLMI_main.
***
*** Setting up ./main_test.sh
Testing the aLMI python software on vdi-n18
Tests ran at 17/03/20 02:40:13 PM AEDT by eal599
*** SETUP ***
Input file was: /g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20130805213444.L2.ANN_P134_V20140704.hdf
Config file was: ../ini_files/aLMI_config_VDItest.ini
Output file was: /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main3.hdf
*** Ready to go. Starting tests. ***
+++| PASSED RrsAboveToBelow
+++| PASSED RrsBelowToAbove
+++| PASSED calcCost
+++| PASSED calcReflForward
+++| PASSED calcUBackward
+++| PASSED calcY
+++| PASSED calc_mu_d
+++| PASSED svd_LMI
+++| PASSED var_dump
+++| PASSED configUtils
+++| PASSED calc_kd_SD
+++| PASSED calcUForward
+++| PASSED SIOP_sets_load
+++| PASSED wavelengthsToVarNames
+++| PASSED chunkProcessLMI
*** Testing the aLMI_main script - this will take a few minutes. ***
+++| PASSED aLMI_main
***
*** Tests complete; 16 passed, 0 failed ***
***
*** For individual log files see /home/599/eal599/_Eric_VDI_/iWQ_aLMI/testing/log_files
***
```

Testing `.nc` files

The Python aLMI code is also able to handle (read and write) NetCDF `.nc` files.

Let's test this functionality by again running all the basic "unit tests" (excluding `aLMI_main.py` for now, with the `-s` flag below) in `main_test.sh`, but now selecting `.nc` files as input and output parameters (note that many of these tests are not actually influenced by the type of input file, `.nc` or `.hdf`).

In [32]:

```
# Select new .nc files as input / output:
CONFIG_FILE = SRC_DIR + "/ini_files/aLMI_config_VDItest_nc.ini"    # .nc-specific config file
INPUT_FILE = SRC_DIR + "/configUtils.py " + CONFIG_FILE + " optionalParameters test_ANN_file"    # get aLMI input file (.nc) from config parameters
OUTPUT_FILE = "/g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main4.nc"    # save data as .nc file
VAR_PREFIX = "Rrs_ANN_"    # radiance bands prefix string for this .nc file -- only used for testing!

!echo "The aLMI input file is: \"`$INPUT_FILE`\""

# Run the tests:
!./main_test.sh `$INPUT_FILE` $CONFIG_FILE $OUTPUT_FILE $VAR_PREFIX -s > ./log_files/main_test_nc.log; $CHECK_OUTPUT_STR
```

```
The aLMI input file is: "/g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.nc"
TEST OK
```

In [33]:

```
!cat ./log_files/main_test_nc.log
```

```
***
*** Running the 'unit tests' only.
***
*** Setting up ./main_test.sh
Testing the aLMI python software on vdi-n18
Tests ran at 17/03/20 02:44:35 PM AEDT by eal599
*** SETUP ***
Input file was: /g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20150928173107.L2OC_BASE.ANN_P134_V20140704.nc
Config file was: ../ini_files/aLMI_config_VDItest_nc.ini
Output file was: /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main4.nc
*** Ready to go. Starting tests. ***
+++| PASSED RrsAboveToBelow
+++| PASSED RrsBelowToAbove
+++| PASSED calcCost
+++| PASSED calcReflForward
+++| PASSED calcUBackward
+++| PASSED calcY
+++| PASSED calc_mu_d
+++| PASSED svd_LMI
+++| PASSED var_dump
+++| PASSED configUtils
+++| PASSED calc_kd_SD
+++| PASSED calcUForward
+++| PASSED SIOP_sets_load
+++| PASSED wavelengthsToVarNames
+++| PASSED chunkProcessLMI
***
*** Tests complete; 15 passed, 0 failed ***
***
*** For individual log files see /home/599/eal599/_Eric_VDI_/iWQ_aLMI/testing/log_files
***
```

Let's now test `aLMI_main.py` individually, and time it at the same time:

In [34]:

```
### Individual timing of aLMI_main.py, if needed:
!time {SRC_DIR}/aLMI_main.py `$INPUT_FILE` $CONFIG_FILE $OUTPUT_FILE -v > ./log_files/aLMI_m
ain.log; $CHECK_OUTPUT_STR
```

```
../aLMI_main.py:558: RuntimeWarning: invalid value encountered in greater
  ok = np.all(input_WL_EL > 0.000001, axis=0)

real    11m48.852s
user    11m35.867s
sys     0m12.735s
TEST OK
```

Here again, we can create a `.dump` file showing the resulting contents of the output aLMI `.nc` file:

In [35]:

```
CMD = "ncdump -h " + OUTPUT_FILE + " > " + OUTPUT_FILE + ".dump"
print(CMD)
!$CMD

print("\n==== (Some of the) contents of the aLMI output file ====")
!head -25 {OUTPUT_FILE}.dump
```

```
ncdump -h /g/data/r78/eal599/iWQ_aLMI/code_testing_out/aLMI_main4.nc > /g/data/r78/eal599/
iWQ_aLMI/code_testing_out/aLMI_main4.nc.dump

==== (Some of the) contents of the aLMI output file ====
netcdf aLMI_main4 {
dimensions:
        numberOfLines = 2170 ;
        numberOfPixelsPerLine = 1354 ;
variables:
        float longitude(numberOfLines, numberOfPixelsPerLine) ;
                longitude:_FillValue = -32767.f ;
                longitude:long_name = "longitude at control points" ;
                longitude:units = "degree_east" ;
                longitude:standard_name = "longitude" ;
                longitude:valid_min = -180.f ;
                longitude:valid_max = 180.f ;
        float latitude(numberOfLines, numberOfPixelsPerLine) ;
                latitude:_FillValue = -32767.f ;
                latitude:long_name = "latitude at control points" ;
                latitude:units = "degree_north" ;
                latitude:standard_name = "latitude" ;
                latitude:valid_min = -90.f ;
                latitude:valid_max = 90.f ;
        int l2_flags(numberOfLines, numberOfPixelsPerLine) ;
                l2_flags:long_name = "Level-2 Processing Flags" ;
                l2_flags:valid_min = -2147483648 ;
                l2_flags:valid_max = 2147483647 ;
                l2_flags:flag_masks = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4
096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 838860
8, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, -2147483648
;
                l2_flags:flag_meanings = "ATMFAIL LAND PRODWARN HIGLINT HILT HISATZEN COAS
TZ SPARE STRAYLIGHT CLDICE COCCOLITH TURBIDW HISOLZEN SPARE LOWLW CHLFAIL NAVWARN ABSAER S
PARE MAXAERITER MODGLINT CHLWARN ATMWARN SPARE SEAICE NAVFAIL FILTER SPARE SPARE HIPOL PRO
DFAIL SPARE" ;
```

9 tests

Finally, the `testing` shell script `9_tests.sh` was written to test the main aLMI code with all nine combinations of `inputType` and `outputType` parameters in the config `.ini` file (i.e. one of `reflAboveSurf`, `reflBelowSurf`, `uIopRatio`). This code can be run as follows, for a `.hdf` input file (**note**: takes a long time to run):

In [36]:

```
CONFIG_FILE = SRC_DIR + "/ini_files/aLMI_config_VDItest.ini"    # config file
INPUT_FILE = SRC_DIR + "/configUtils.py " + CONFIG_FILE + " optionalParameters test_ANN_fil
e"    # get the aLMI input file from the config file
OUTPUT_DIR = "/g/data/r78/eal599/iWQ_aLMI/code_testing_out/9_tests"    # where the data will
be written

!./9_tests.sh `$INPUT_FILE` $CONFIG_FILE $OUTPUT_DIR -v > ./log_files_9_tests/9_tests.log;
$CHECK_OUTPUT_STR
```

```
TEST OK
```

In [37]:

```
!cat ./log_files_9_tests/9_tests.log
```

```
Testing the aLMI python software (9_tests) on vdi-n18
Tests ran at 17/03/20 02:56:34 PM AEDT by eal599
*** SETUP ***
Input file was: /g/data/r78/eal599/iWQ_aLMI/data/A20120403_0410.20130805213444.L2.ANN_P134
_V20140704.hdf
Config file was: ../ini_files/aLMI_config_VDItest.ini
Output dir was: /g/data/r78/eal599/iWQ_aLMI/code_testing_out/9_tests
*** Ready to go. Starting tests. ***
+++| PASSED reflAboveSurfIn_reflAboveSurfOut
+++| PASSED reflAboveSurfIn_reflBelowSurfOut
+++| PASSED reflAboveSurfIn_uIopRatioOut
+++| PASSED reflBelowSurfIn_reflAboveSurfOut
+++| PASSED reflBelowSurfIn_reflBelowSurfOut
+++| PASSED reflBelowSurfIn_uIopRatioOut
+++| PASSED uIopRatioIn_reflAboveSurfOut
+++| PASSED uIopRatioIn_reflBelowSurfOut
+++| PASSED uIopRatioIn_uIopRatioOut
***
*** 9_tests complete; 9 passed, 0 failed ***
***
*** For individual log files see ./log_files_9_tests
***
```