Community Inter-comparrison Suite Documentation

Release 0.6.5

Centre of Environmental Data Archival

August 31, 2015

Contents

Contents:

CHAPTER 1

Installing CIS

First, clone the git repository: `$ git clone http://proj.badc.rl.ac.uk/git/jasmin_cis.git`

If you have admin rights, simply run the *setup.py install* script. It will check your system for dependencies but won't install them for you (installations scripts specifically for ubuntu 12 and fedora 17 are provided under the 'scripts' directory, but these come with no warranty).

If you haven't got admin rights, all is not lost! you can still install CIS by first creating a virtual environment:

```
$ virtualenv CISENV -p /usr/bin/python2.7 --system-site-packages
$ source CISENV/bin/activate
```

To deactivate the virtual environment, simply type `$ deactivate`

Note that, on the Jasmin cluster, some ports needs to be to open to make virtualenv works with:

```
$ export http_proxy=wwwcache.rl.ac.uk:8080
$ export https_proxy=wwwcache.rl.ac.uk:8080
```

Then simply run the setup.py script as you normally would. Now you're ready to rock and roll!

To check that CIS is installed correctly, simply type `cis` from the command line and should see something like:

```
(CISENV)[user@computer ~]$ cis
usage: CIS [-h] {plot,info,col,version} ...
CIS: error: too few arguments
```

1.1 Checking the version

Typing `cis version` displays the version number, for example:

```
(CISENV)[user@computer ~]$ cis version
Using CIS version: V0R6M0 (Development)
```

1.2 Dependencies

Use the following command to check the dependencies that CIS requries to run:

```
$ python setup.py checkdep
```

CHAPTER 2

What kind of data can CIS deal with?

2.1 Writing

When creating files from a CIS command, CIS uses the NetCDF 4 classic format. Ungridded output files are always prefixed with `cis-`, and both ungridded and gridded output are always suffixed with `.nc`.

2.2 Reading

CIS has built-in support for NetCDF and HDF4 file formats. That said, most data requires some sort of pre-processing before being ready to be plotted or analysed (this could be scale factors or offsets needing to applied, or even just knowing what the dependencies between variables are). For that reason, the way CIS deals with reading in data files is via the concept of "data products". Each product has its own very specific way of reading and interpreting the data in order for it to be ready to be plotted, analysed, etc.

So far, CIS can read the following ungridded data files:

Dataset	Product name	Type	File Signature
AERONET	Aeronet	Ground-stations	*.lev20
Aerosol CCI	Aerosol_CCI	Satellite	*ESACCI*AEROSOL*
CALIOP L1	Caliop_L1	Satellite	CAL_LID_L1-ValStage1-V3*.hdf
CALIOP L2	Caliop_L2	Satellite	CAL_LID_L2_05kmAPro-Prov-V3*.hdf
Cloud-Sat	CloudSat	Satellite	*_CS_*GRANULE*.hdf
Flight cam-paigns	NCAR_NetCDF_RAF	Aircraft	RF*.nc
MODIS L2	MODIS_L2	Satellite	*MYD06_L2*.hdf, *MOD06_L2*.hdf, *MYD04_L2*.hdf, *MOD04_L2*.hdf, *MYDATML2.*.hdf, *MODATML2*.hdf
Cloud CCI	Cloud_CCI	Satellite	*ESACCI*CLOUD*
CSV data-points	ASCII_Hyperpoints	N/A	*.txt
CIS un-gridded	cis	CIS output	cis-*.nc
NCAR-RAF	NCAR_NetCDF_RAF	Aircraft	*.nc containing the attribute Conventions with the value NCAR-RAF/nimbus
GASSP	NCAR_NetCDF_RAF	Aircraft	*.nc containing the attribute GASSP_Version
GASSP	NCAR_NetCDF_RAF	Ship	*.nc containing the attribute GASSP_Version, with no altitude
GASSP	NCAR_NetCDF_RAF	Ground-station	*.nc containing the attribute GASSP_Version, with attributes Station_Lat, Station_Lon and Station_Altitude

It can also read the following gridded data types:

Dataset	Product name	Type	File Signature
MODIS L3 daily	MODIS_L3	Satellite	*MYD08_D3*.hdf, *MOD08_D3*.hdf, *MOD08_E3*.hdf
Net_CDF Gridded Data	NetCDF_Gridded	Gridded Model Data	*.nc (this is the default for NetCDF Files that do not match any other signature)

The file signature is used to automatically recognise which product definition to use. Note the product can overridden easily by being specified at the command line.

This is of course far from being an exhaustive list of what's out there. To cope with this, a "plugin" architecture has been designed so that the user can readily use their own data product reading routines, without even having to change the code - see Design Maintenance Guide for more information.

the plugins are always read first, so one can also overwrite default behaviour if the built-in products listed above do not achieve a very specific purpose.

2.3 Datagroups

Most CIS commands operate on a 'datagroup', which is a unit of data containing one or more similar variables and one or more files from which those variables should be taken. A datagroup represents closely related data from a specific

instrument or model and as such is associated with only one data product.

A datagroup is specified with the syntax:

`<variable>...:<filename>[:product=<productname>]` where:

- `<variable>` is a mandatory argument specifying the variable or variable names to use. This should be the name of the variable as described in the file, e.g. the NetCDF variable name or HDF SDS/VDATA variable name. Multiple variables may be specified by commas, and variables may be wildcarded using any wildcards compatible with the python module glob, so that *, ? and [] can all be used

> **Attention:** When specifying multiple variables, it is essential that they be on the same grid (i.e. use the same coordinates).

- `<filenames>` is a mandatory argument used to specify the files to read the variable from. These can be specified as a comma seperated list of the following possibilities:

  1. a single filename - this should be the full path to the file
  2. a single directory - all files in this directory will be read
  3. a wildcarded filename - A filename with any wildcards compatible with the python module glob, so that *, ? and [] can all be used. E.g., `/path/to/my/test*file_[0-9]`.

> **Attention:** When multiple files are specified (whether through use of commas, pointing at a directory, or wildcarding), then all those files must contain all of the specified variables, and the files should be 'compatible' - it should be possible to aggregate them together using a shared dimension - typically time (in a NetCDF file this is usually the unlimited dimension). So selecting multiple monthly files for a model run would be OK, but selecting files from two different datatypes would not be OK.

- `<productname>` is an optional argument used to specify the type of files being read. If omitted, the program will attempt to figure out which product to use based on the filename. See *Reading* to see a list of available products and their file signatures.

For example:

```
illum:20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc
Cloud_Fraction_*:MOD*,MODIS_dir/:product=MODIS_L2
```

2.4 Reading NetCDF4 Hierarchical Groups

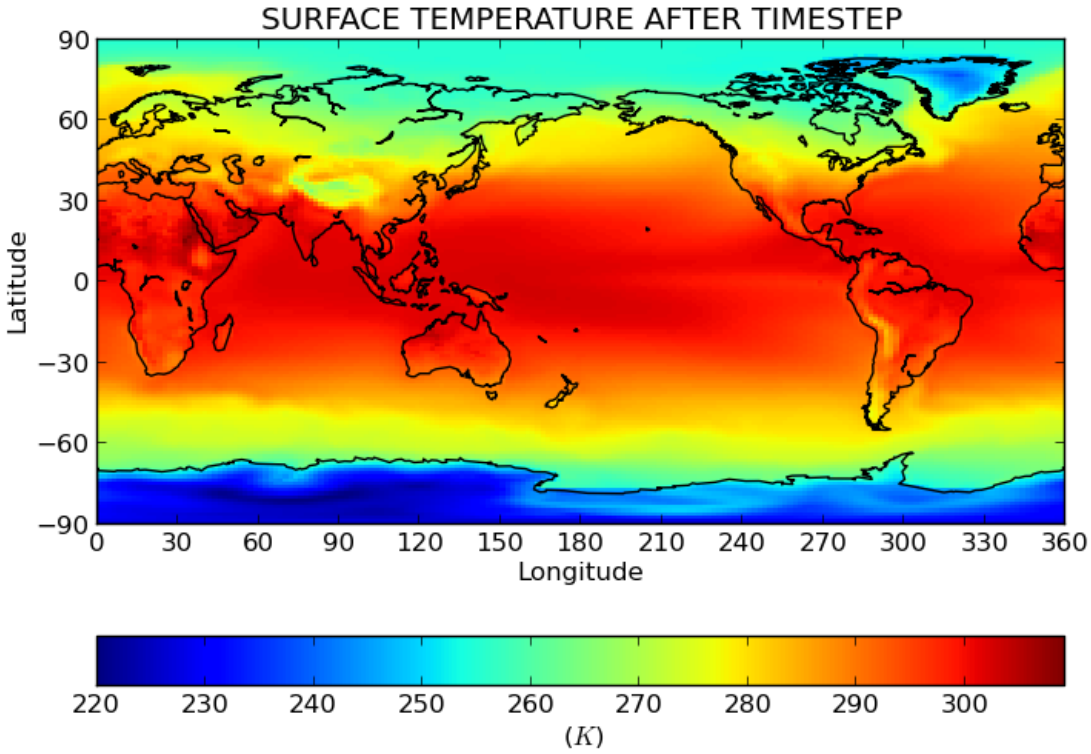
CIS supports the reading of NetCDF4 hierarchical groups. These can be specified on the command line in the format `<group>.<variable_name>`, e.g. `AVHRR.Ch4CentralWavenumber`. Groups can be nested to any required depth like `<group1>.<group2...>.<variable_name>`.

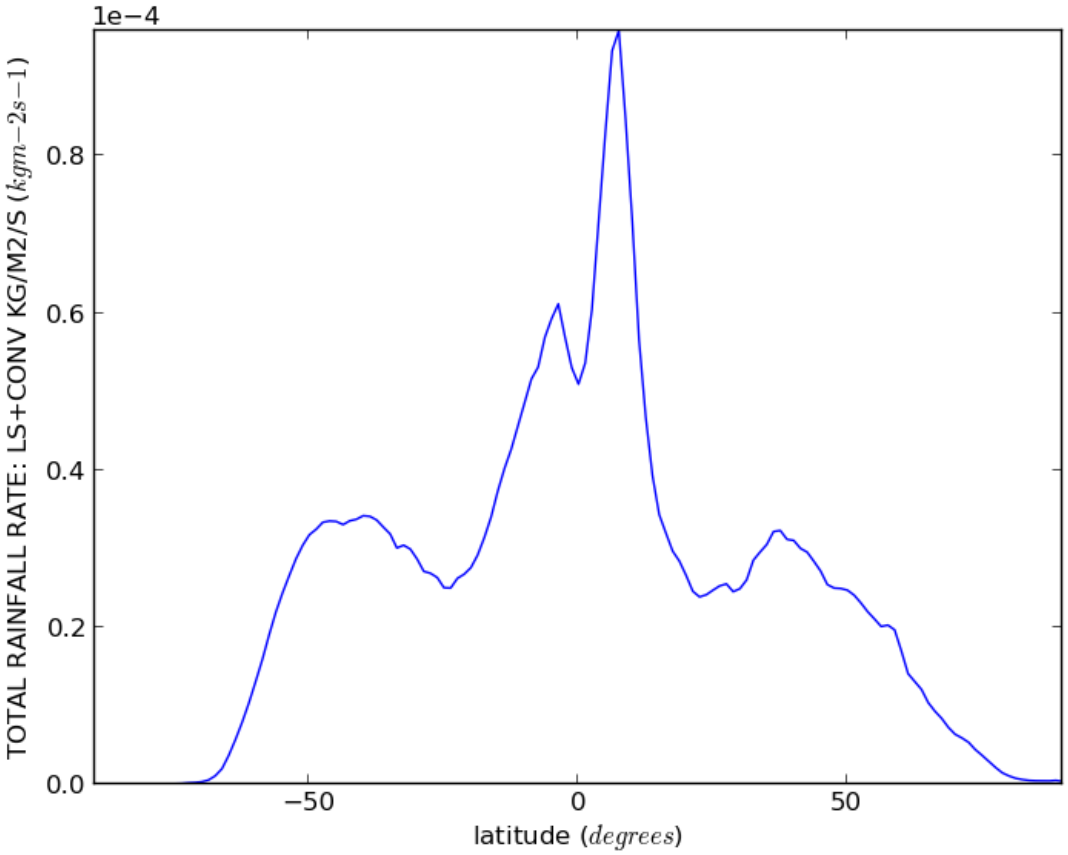
CIS currently does not support writing out of NetCDF4 groups, so any groups read in will be output 'flat'.

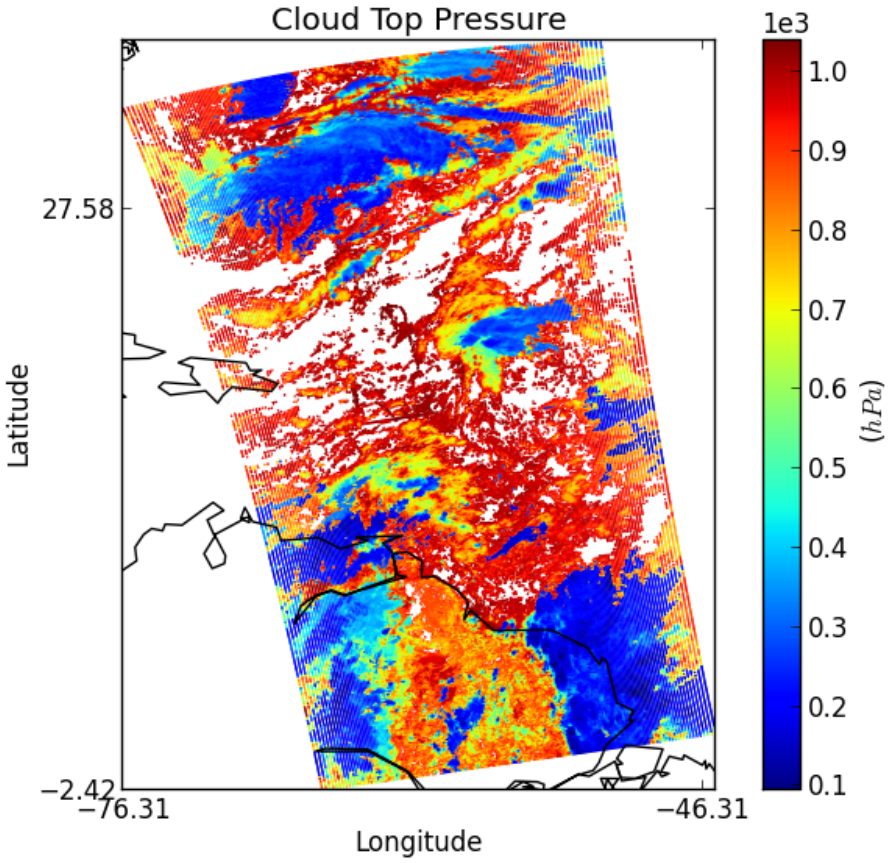
2.4.1 Reading groups in user-developed product plugins

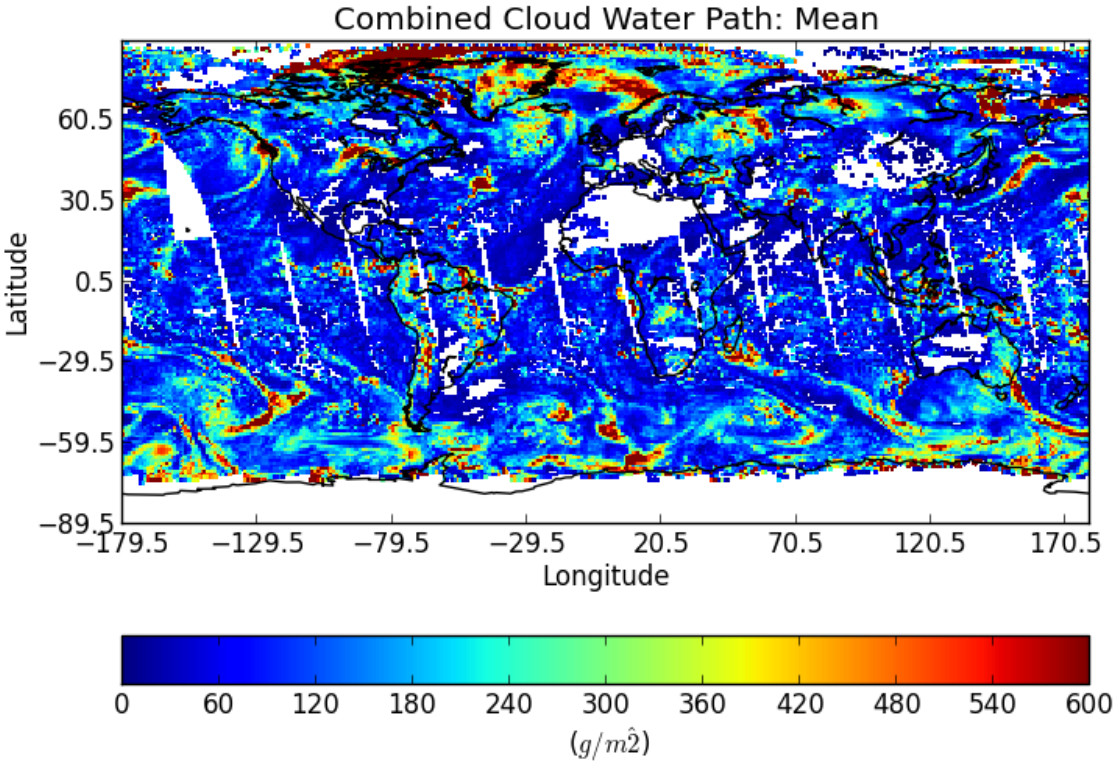
Most of the methods in the cis.data_io.netcdf module support netCDF4 groups using the syntax described above - users should use this module when designing their own plugins to ensure support for groups.

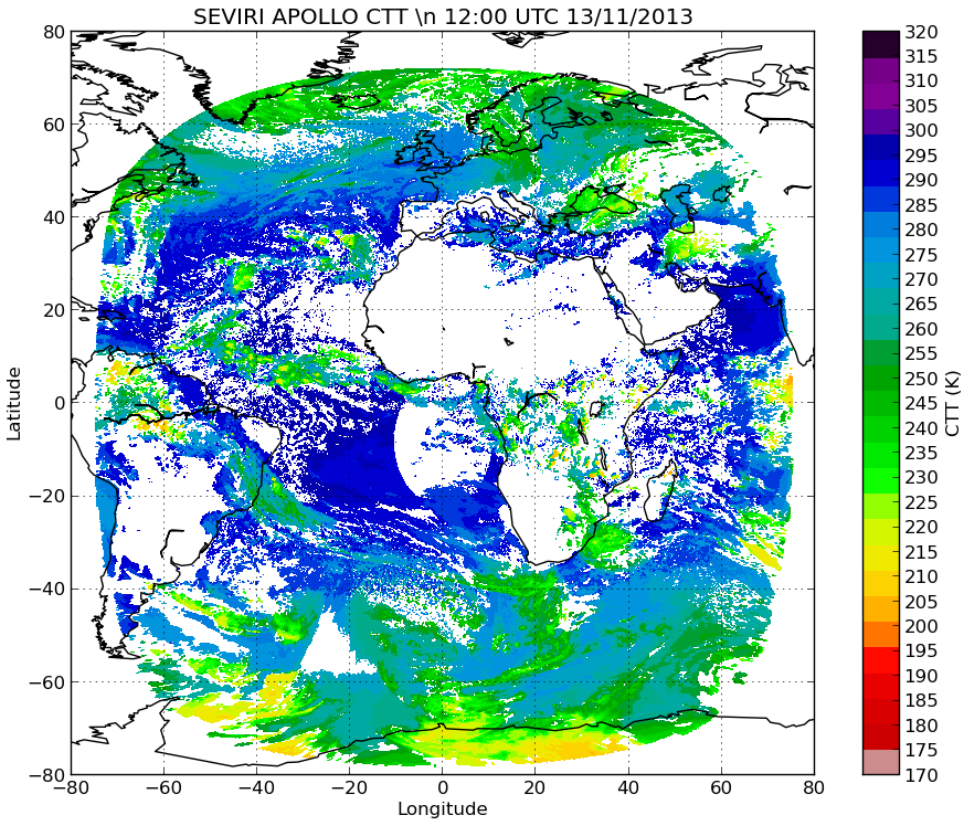
2.5 Example plots

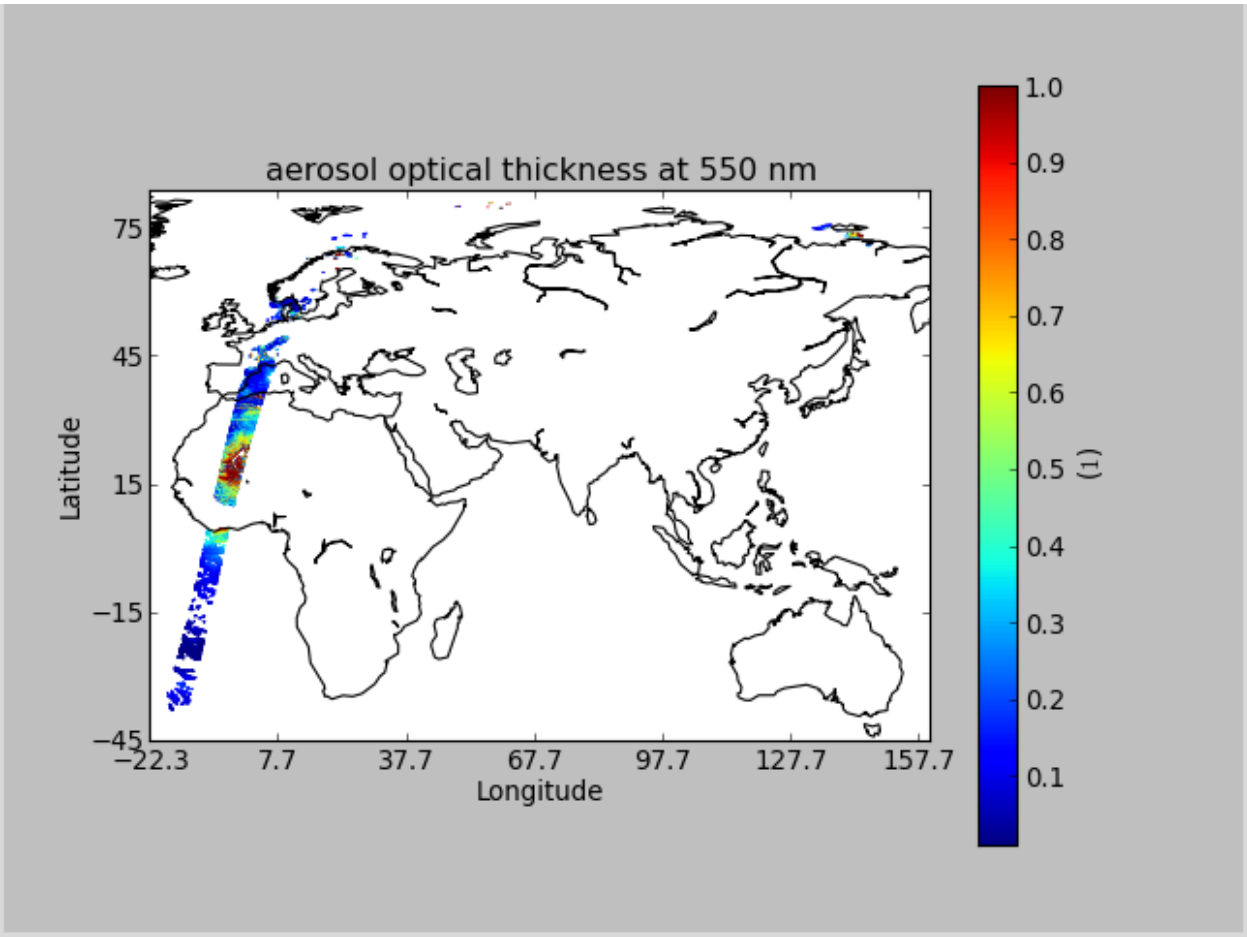


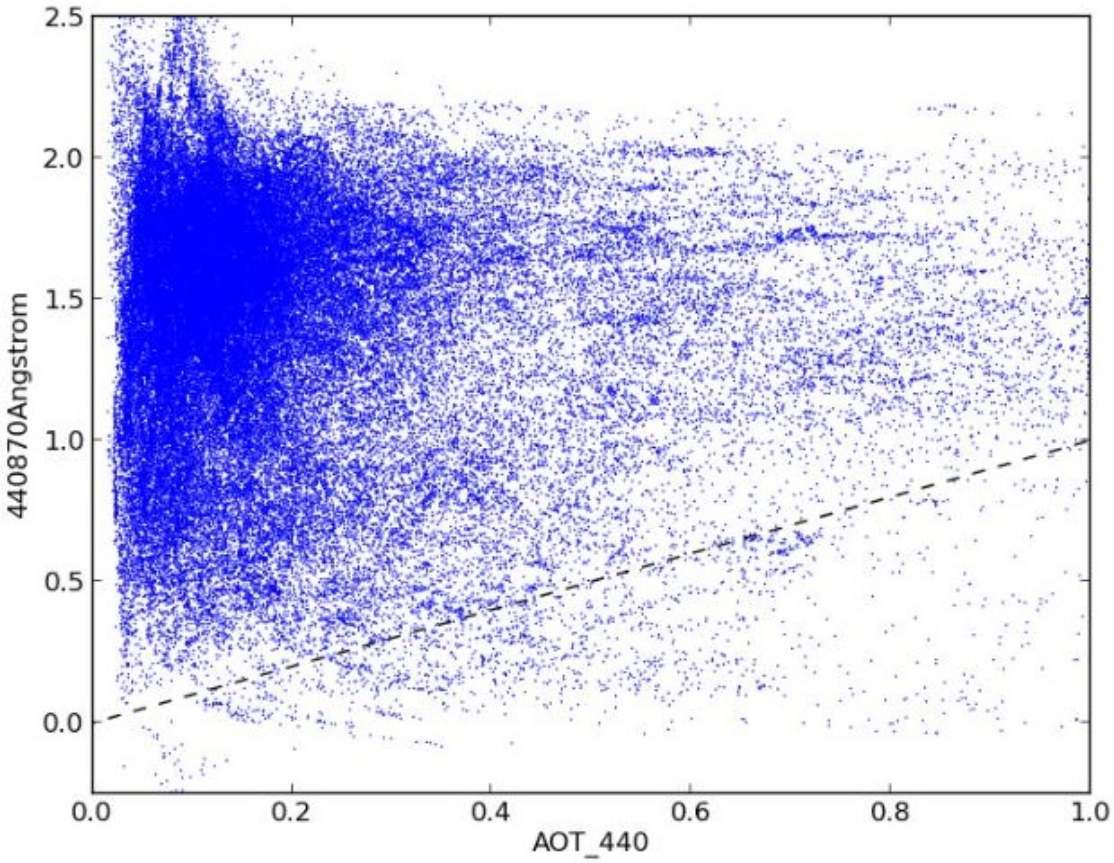

1e−4
0.8
0.6
0.4
0.2
0.0
TOTAL RAINFALL RATE: LS+CONV KG/M2/S ($kgm^{-2}s^{-1}$)
−50
0
50
latitude ($degrees$)

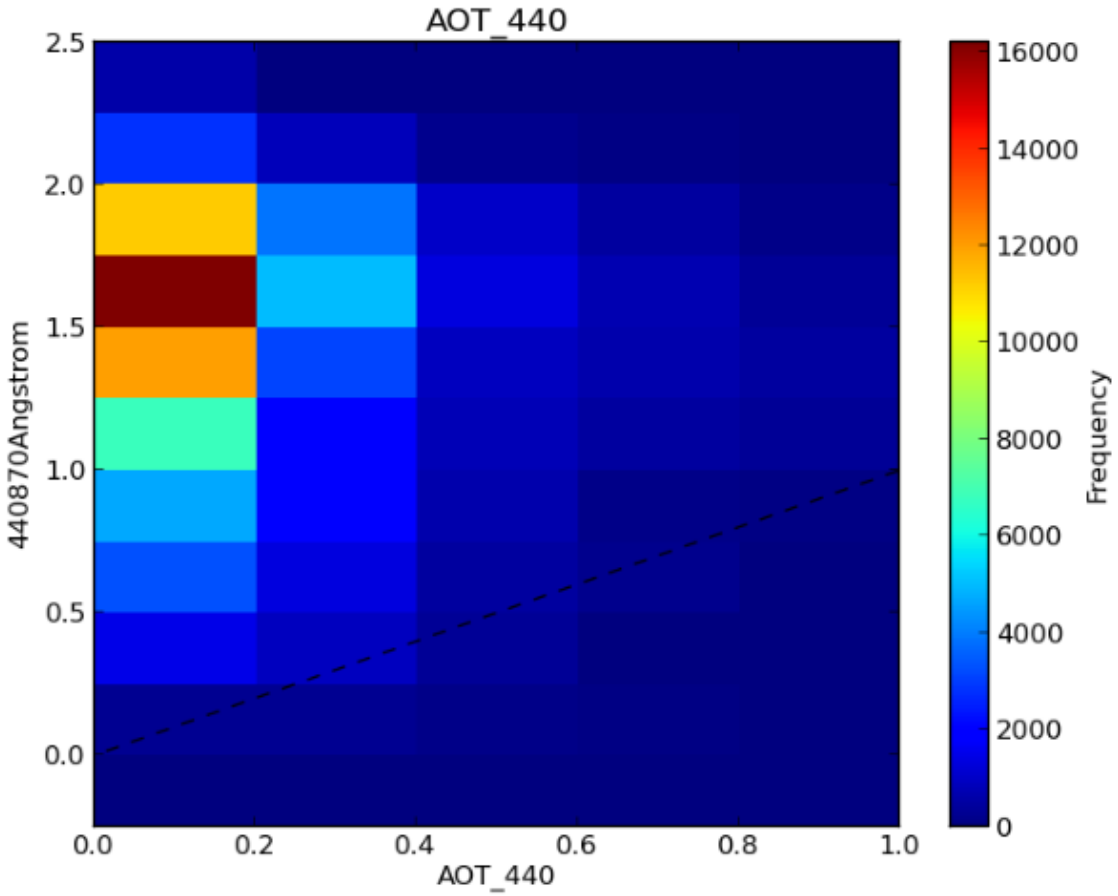

Cloud Top Pressure
1e3
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
(hPa)
27.58
−2.42
Latitude
−76.31
−46.31
Longitude

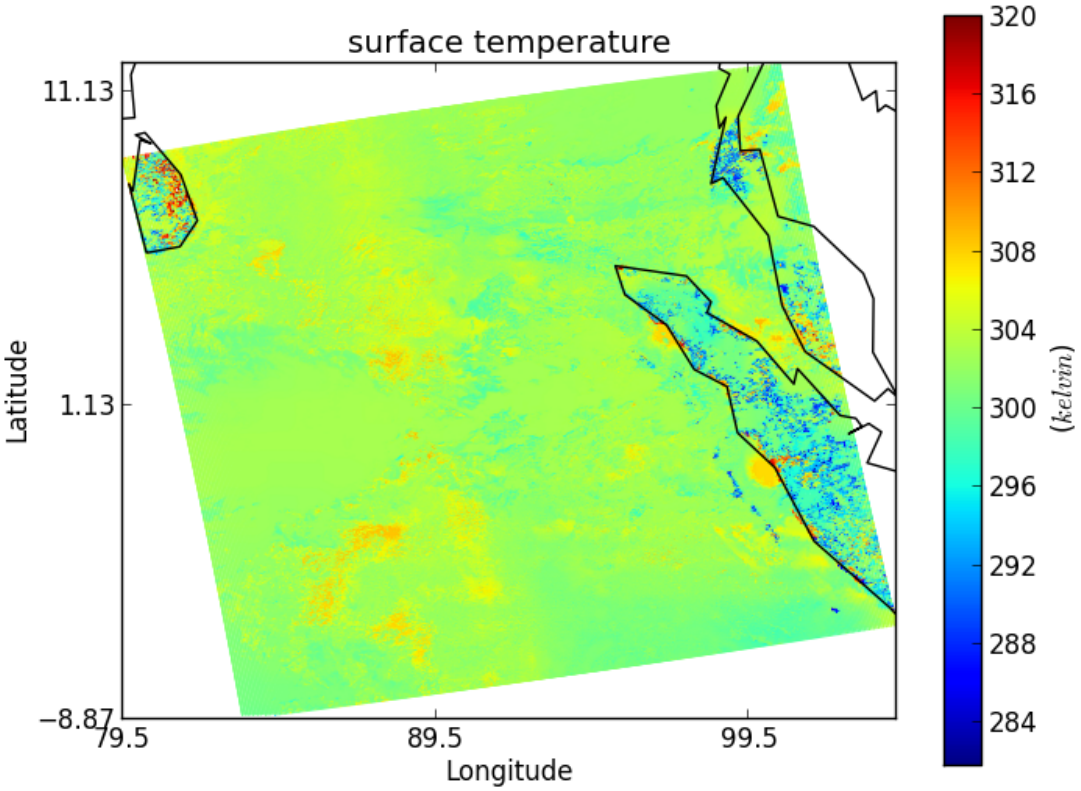

Combined Cloud Water Path: Mean
60.5
30.5
0.5
−29.5
−59.5
−89.5
Latitude
−179.5
−129.5
−79.5
−29.5
20.5
70.5
120.5
170.5
Longitude
0
60
120
180
240
300
360
420
480
540
600
$(g/m\hat{2})$

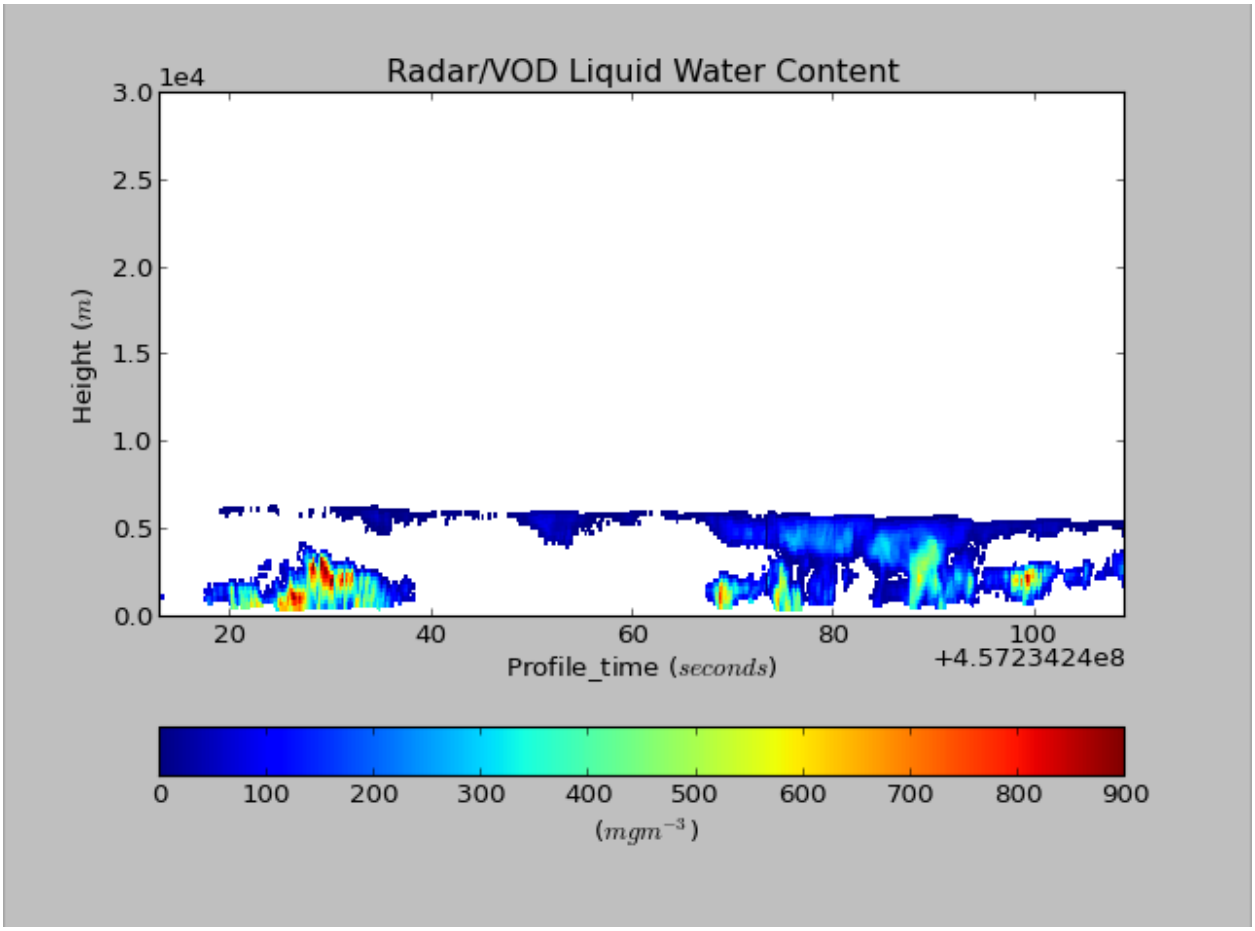

SEVIRI APOLLO CTT \n 12:00 UTC 13/11/2013
Latitude
Longitude
CTT (K)
80
60
40
20
0
−20
−40
−60
−80
320
315
310
305
300
295
290
285
280
275
270
265
260
255
250
245
240
235
230
225
220
215
210
205
200
195
190
185
180
175
170

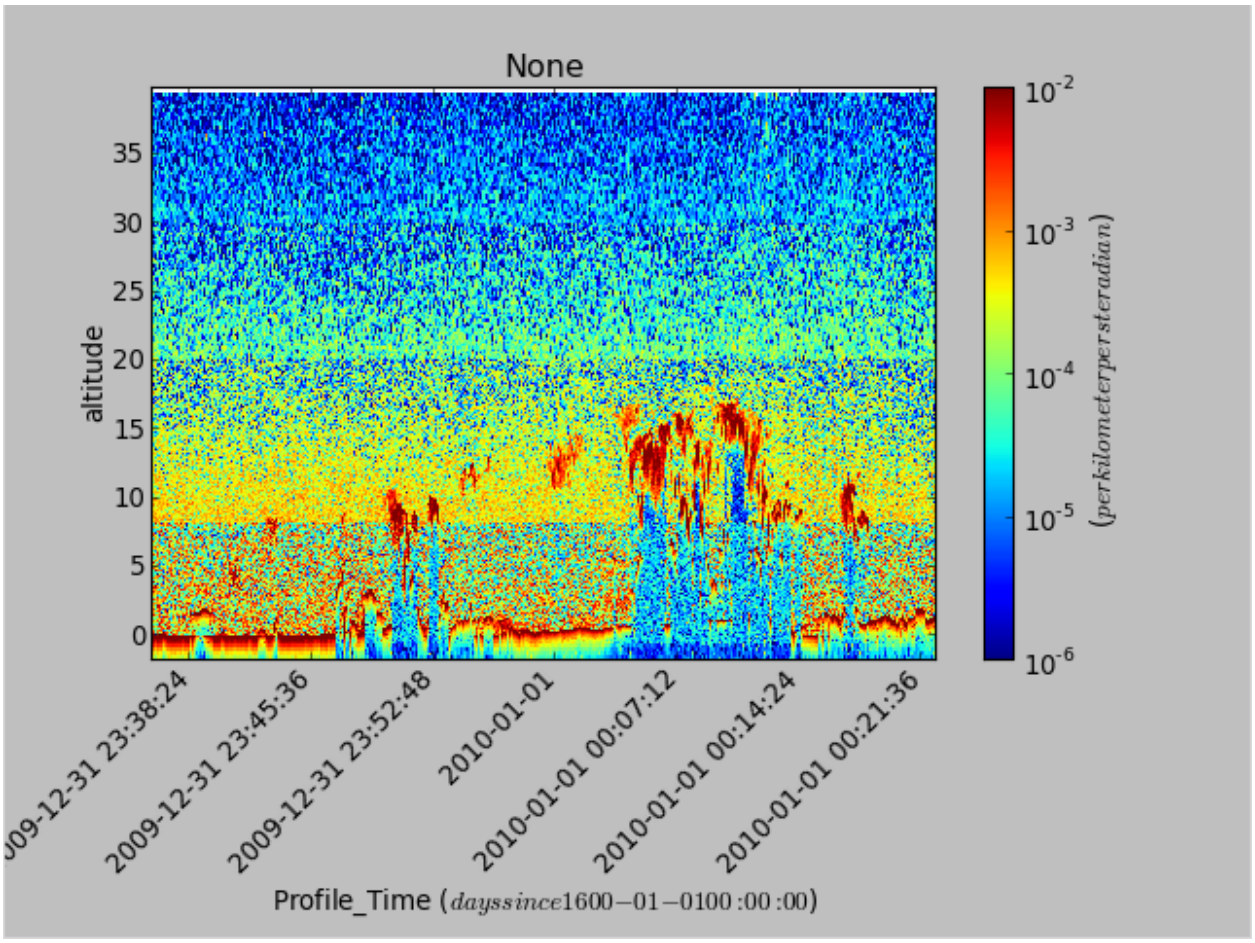

aerosol optical thickness at 550 nm
Latitude
75
45
15
−15
−45
−22.3
7.7
37.7
67.7
97.7
127.7
157.7
Longitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
(1)

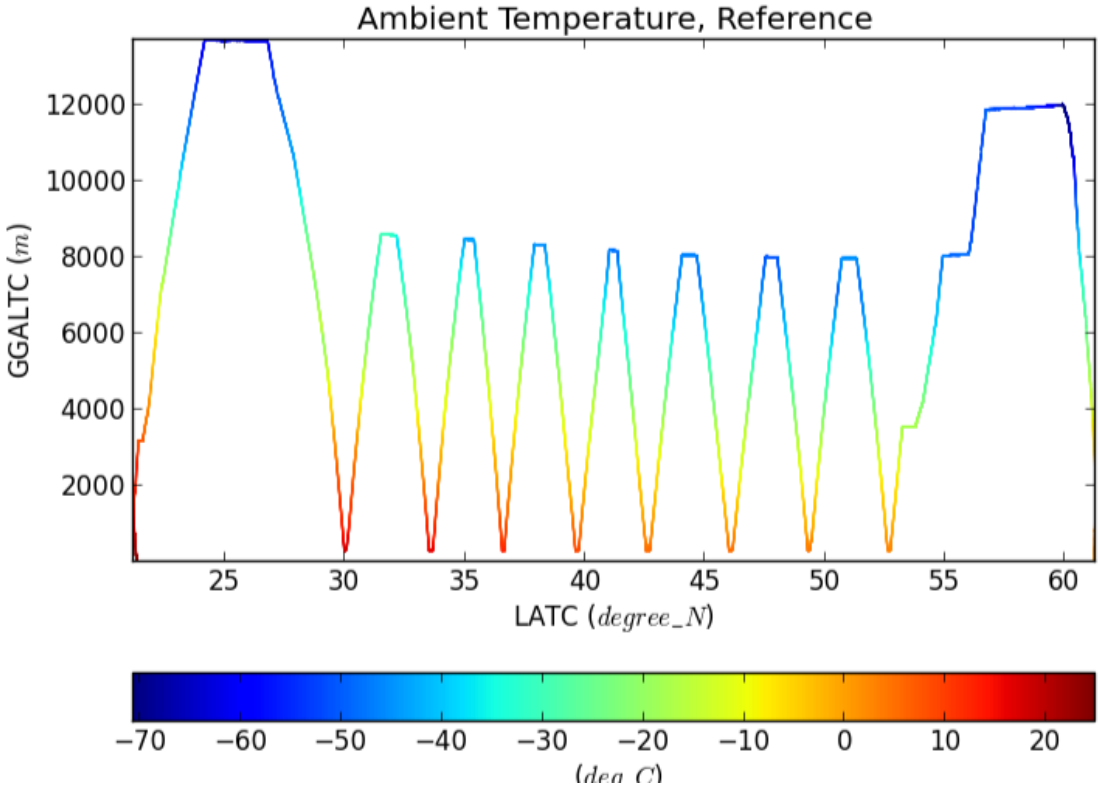

2.5
2.0
1.5
1.0
0.5
0.0
440870Angstrom
0.0
0.2
0.4
0.6
0.8
1.0
AOT_440


AOT_440
2.5
2.0
1.5
1.0
0.5
0.0
440870Angstrom
0.0
0.2
0.4
0.6
0.8
1.0
AOT_440
16000
14000
12000
10000
8000
6000
4000
2000
0
Frequency


surface temperature
11.13
1.13
−8.87
Latitude
79.5
89.5
99.5
Longitude
320
316
312
308
304
300
296
292
288
284
(kelvin)


Radar/VOD Liquid Water Content
1e4
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Height (m)
20
40
60
80
100
Profile_time ($seconds$)
+4.5723424e8
0
100
200
300
400
500
600
700
800
900
(mgm^{-3})


None
altitude
35
30
25
20
15
10
5
0
2009-12-31 23:38:24
2009-12-31 23:45:36
2009-12-31 23:52:48
2010-01-01
2010-01-01 00:07:12
2010-01-01 00:14:24
2010-01-01 00:21:36
Profile_Time (daysince1600−01−0100 :00 :00)
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
(perkilometerpersteradian)


Ambient Temperature, Reference
GGALTC (m)
12000
10000
8000
6000
4000
2000
25
30
35
40
45
50
55
60
LATC (degree_N)
−70
−60
−50
−40
−30
−20
−10
0
10
20
(deg C)

CHAPTER 3

Using the command line

Run the following command to print help and check that it runs: `cis --help`

The following should be displayed:

```
usage: cis [-h] {plot,info,col,aggregate,subset,version} ...

positional arguments:
  {plot,info,col,aggregate,subset,version}
    plot            Create plots
    info            Get information about a file
    col             Perform colocation
    aggregate       Perform aggregation
    subset          Perform subsetting
    eval            Evaluate a numeric expression
    stats           Perform statistical comparison of two datasets
    version         Display the CIS version number

optional arguments:
  -h, --help   show this help message and exit
```

There are 8 commands the program can execute:

- `plot` which is used to plot the data
- `info` which prints information about a given input file
- `col` which is used to perform colocation on data
- `aggregate` which is used to perform aggregation along coordinates in the data
- `subset` which is used to perform subsetting of the data
- `eval` which is used to evaluate a numeric expression on data
- `stats` which is used to perform a statistical comparison of two datasets
- `version` which is used to display the version number of CIS

If an error occurs while running any of these commands, you may wish to check the log file 'cis.log'; the default location for this is the current user's home directory.

3.1 LSF Batch Job Submission

CIS jobs may be submitted to an LSF type batch submission system (e.g. the JASMIN environment) by using the command `cis.lsf` instead of cis. In this case the job will be sent to the batch system and any output will be written

to the log file.

CHAPTER 4

Getting file information

Running ‘‘ ./cis.py info $filename ‘‘ will print a list of the variables available in that input file such as:

```
Trop
latitude
longitude_1
surface
unspecified_1
level6
ht
msl
latitude_1
```

To get more specific information about a given variable, simply run:

```
$ ./cis.py info $filename --variable $var1 $var2 $var3
```

where `$var1`, `$var2` and `$var3` are the names of the variables to get the information for.

Here is an example:

```
<type 'netCDF4.Variable'>
float32 mass(t, unspecified_1, latitude, longitude_1)
    _FillValue: 2e+20
    date: 01/09/08
    long_name: TOTAL COLUMN DRY MASS  RHO GRID
    missing_value: 2e+20
    name: mass
    source: Unified Model Output (Vn 7.3):
    time: 00:00
    title: TOTAL COLUMN DRY MASS  RHO GRID
    units: kg m-2
unlimited dimensions: t
current shape = (1, 1, 145, 192)
```

CHAPTER 5

Subsetting

Subsetting allows the reduction of data by extracting variables and restricting them to ranges of one or more coordinates.

To perform subsetting, run a command of the format:

```
$ cis subset <datagroup> <limits> [-o <outputfile>]
```

where:

<datagroup> is a *CIS datagroup* specifying the variables and files to read and is of the format `<variable>...:<filename>[:product=<productname>]` where:

- `variable` is a mandatory variable or list of variables to use.
- `filenames` is a mandatory file or list of files to read from.
- `product` is an optional CIS data product to use (see *Data Products*):

See *Datagroups* for a more detailed explanation of datagroups.

<limits> is a comma separated sequence of one or more coordinate range assignments of the form `variable=[start,end]` or `variable=[value]` in which

- `variable` is the name of the variable to be subsetted, or one of x, y, z or t, which refer to longitude, latitude, altitude or time, respectively.
- `start` is the value at the start of the coordinate range to be included
- `end` is the value at the end of the coordinate range to be included
- `value` is taken as the start and end value.

Note: Longitude coordinates are considered to be circular, so that -10 is equivalent to 350. The start and end must describe a monotonically increasing coordinate range, so `x=[90,-90]` is invalid, but could be specified using `x=[90,270]`. The range between the start and end must not be greater than 360 degrees. The output coordinates will be on the requested grid, not the grid of the source data.

Note: Date/times are specified in the format: `YYYY-MM-DDThh:mm:ss` in which `YYYY-MM-DD` is a date and `hh:mm:ss` is a time. A colon or space can be used instead of the 'T' separator (but if a space is used, the argument must be quoted). Any trailing components of the date/time may be omitted. When a date/time is used as a range start, the earliest date/time compatible with the supplied components is used (e.g., `2010-04` is treated as `2010-04-01T00:00:00`) and when used as a range end, the latest compatible date/time is used. Including optional and alternative components, the syntax is `YYYY[-MM[-DD[{T|:| }hh[:mm[:ss]]]]]`. When the `t=[value]` form is

used, value is interpreted as both the start and end value, as described above, giving a range spanning the specified date/time, e.g., `t=[2010]` gives a range spanning the whole of the year 2010.

outputfile is an optional argument to specify the name to use for the file output. This is automatically given a `.nc` extension and prepended with `cis-`, if it contains ungridded data, to make it distinguishable as a colocated file. The default filename is ``cis-out.nc` for ungridded data, and `out.nc` for gridded data.

A full example would be:

```
$ cis subset solar_3:xglnwa.pm.k8dec-k9nov.col.tm.nc x=[0,180],y=[0,90] -o Xglnwa-solar_3
```

Gridded netCDF data is output as gridded data, while ungridded and non-netCDF gridded data is output as ungridded data.

5.1 Examples

Below are examples of subsetting using each of the supported products (together with a command to plot the output):

```
$ cis subset AO2CO2:RF04.20090114.192600_035100.PNI.nc t=[2009-01-14:19:26:00,2009-01-14:19:36:00] -o
$ cis plot AO2CO2:cis-RF04-AO2CO2-out.nc

$ cis subset IO_RVOD_ice_water_content:2007180125457_06221_CS_2B-CWC-RVOD_GRANULE_P_R04_E02.hdf t=[20
$ cis plot IO_RVOD_ice_water_content:cis-CloudSAT-out.nc --xaxis=time --yaxis=altitude

$ cis subset Cloud_Top_Temperature:MYD06_L2.A2011100.1720.051.2011102130126.hdf x=[-50,-40],y=[0,10]
$ cis plot Cloud_Top_Temperature:cis-MODIS_L2-out.nc

$ cis subset cwp:20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc x=[85,90],y=[-3,3] -
$ cis plot atmosphere_mass_content_of_cloud_liquid_water:cis-Cloud_CCI-out.nc

$ cis subset AOD870:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc x=[-5,2
$ cis plot atmosphere_optical_thickness_due_to_aerosol:cis-Aerosol_CCI-out.nc

$ cis subset 440675Angstrom:920801_121229_Abracos_Hill.lev20 t=[2002] -o Aeronet-out
$ cis plot 440675Angstrom:cis-Aeronet-out.nc --xaxis=time --yaxis=440675Angstrom

$ cis subset solar_3:xglnwa.pm.k8dec-k9nov.vprof.tm.nc y=[0,90] -o Xglnwa_vprof-out
$ cis plot solar_3:Xglnwa_vprof-out.nc

$ cis subset solar_3:xglnwa.pm.k8dec-k9nov.col.tm.nc x=[0,180],y=[0,90] -o Xglnwa-out
$ cis plot solar_3:Xglnwa-out.nc

$ cis subset Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf x=[0,179.9],y=[0
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MODIS_L3-out.nc
```

The files used above can be found at:

```
/group_workspaces/jasmin/cis/jasmin_cis_repo_test_files/
  2007180125457_06221_CS_2B-CWC-RVOD_GRANULE_P_R04_E02.hdf
  20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc
  20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc
  MOD08_E3.A2010009.005.2010026072315.hdf
  MYD06_L2.A2011100.1720.051.2011102130126.hdf
  RF04.20090114.192600_035100.PNI.nc
  xglnwa.pm.k8dec-k9nov.col.tm.nc
  xglnwa.pm.k8dec-k9nov.vprof.tm.nc
```

```
/group_workspaces/jasmin/cis/data/aeoronet/AOT/LEV20/ALL_POINTS/
  920801_121229_Abracos_Hill.lev20
```

CHAPTER 6

Aggregation

The Community Intercomparison Suite (CIS) has the ability to aggregate both gridded and ungridded data along one or more coordinates. For example, you might aggregate a dataset over the longitude coordinate to produce an averaged measurement of variation over latitude.

CIS supports 'complete collapse' of a coordinate - where all values in that dimension are aggregated so that the coordinate no longer exists - and 'partial collapse' - where a coordinate is aggregated into bins of fixed size, so that the coordinate still exists but is on a coarser grid. Partial collapse is currently only supported for ungridded data. The output of an aggregation is always a CF compliant gridded NetCDF file.

The aggregation command has the following syntax:

```
$ cis aggregate <datagroup>[:options] <grid> [-o <outputfile>]
```

where:

<datagroup> is a *CIS datagroup* specifying the variables and files to read and is of the format `<variable>...:<filename>[:product=<productname>]` where:

- `<variable>` is a mandatory variable or list of variables to use.
- `<filenames>` is a mandatory file or list of files to read from.
- `<productname>` is an optional CIS data product to use (see *Data Products*):

See *Datagroups* for a more detailed explanation of datagroups.

<options> Optional arguments given as `keyword=value` in a comma separated list. Options are:

- `kernel=<kernel>` - the method by which the value in each aggregation cell is determined. `<kernel>` should be one of:
  - `mean` - use the mean value of all the data points in that aggregation cell. For gridded data, this mean is weighted to take into account differing cell areas due to the projection of lat/lon lines on the Earth.
  - `min` - use the lowest valid value of all the data points in that aggregate cell.
  - `max` - use the highest valid value of all the data points in that aggregate cell.
  - `moments` - In addition to returning the mean value of each cell (weighted where applicable), this kernel also outputs the number of points used to calculate that mean and the standard deviation of those values, each as a separate variable in the output file.

  If not specified the default is `moments`.
- `product=<productname>` is an optional argument used to specify the type of files being read. If omitted, CIS will attempt to figure out which product to use based on the filename. See *Reading* to see a list of available product names and their file signatures.

<grid> This mandatory argument specifies the coordinates to aggregate over and whether they should be completely collapsed or aggregated into bins. Multiple coordinates can be aggregated over, in which case they should be separated by commas. Coordinates may be identified using their variable names (e.g. `latitude`) or by choosing from `x`, `y`, `t`, `z`, `p` which refer to longitude, latitude, time, altitude and pressure respectively.

- *Complete collapse* - To perform a complete collapse of a coordinate, simply provide the name of the coordinate(s) as a comma separated list - e.g. `x,y` will aggregate data completely over both latitude and longitude. For ungridded data this will result in length one coordinates with bounds reflecting the maximum and minimum values of the collapsed coordinate.
- *Partial collapse* - To aggregate a coordinate into bins, specify the start, end and step size of those bins in the form `coordinate=[start,end,step]`. The step may be missed out, in which case the bin will span the whole range given. Partial collapse is currently only supported for ungridded data.

Note: Longitude coordinates are considered to be circular, so that -10 is equivalent to 350. The start and end must describe a monotonically increasing coordinate range, so `x=[90,-90,10]` is invalid, but could be specified using `x=[90,270,10]`. The range between the start and end must not be greater than 360 degrees.

Complete and partial collapses may be mixed where applicable - for example, to completely collapse time and to aggregate latitude on a grid from -45 degrees to 45 degrees, using a step size of 10 degrees:

```
t,y=[-45,45,10]
```

Note: For ungridded data, if a coordinate is left unspecified it is collapsed completely. This is in contrast to gridded data where a coordinate left unspecified is not used in the aggregation at all.

Note: The range specified is the very start and end of the grid, the actual midpoints of the aggregation cells will start at `start + delta/2`.

Date/times:

Date/times are specified in the format: `YYYY-MM-DDThh:mm:ss` in which `YYYY-MM-DD` is a date and `hh:mm:ss` is a time. A colon or space can be used instead of the 'T' separator (but if a space is used, the argument must be quoted). Any trailing components of the date/time may be omitted. When a date/time is used as a range start, the earliest date/time compatible with the supplied components is used (e.g., `2010-04` is treated as `2010-04-01T00:00:00`) and when used as a range end, the latest compatible date/time is used. Including optional and alternative components, the syntax is `YYYY[-MM[-DD[{T|:| }hh[:mm[:ss]]]]]`.

Date/time steps are specified in the ISO 8061 format `PnYnMnDTnHnMnS`, where any particular time period is optional, for example `P1MT30M` would specify a time interval of 1 month and 30 minutes. Years and months are treated as calendar years and months, meaning they are not necessarily fixed in length. For example a date interval of 1 year and 1 month would mean going from 12:00 15th April 2013 to 12:00 15th May 2013. The are two exceptions to this, in rare cases such as starting at 30th January and going forward 1 month, the month is instead treated as a period of 28 days. Also, for the purposes of finding midpoints for the start in a month the month is always treated as 30 days. For example, to start on the 3rd November 2011 at 12:00 and aggregate over each month up to 3rd January 2013 at 12:00:

- `t=[2011-11-03T12:00,2013-01,P1M]`

<outputfile> is an optional argument to specify the name to use for the file output. This is automatically given a `.nc` extension if not present. This must not be the same file path as any of the input files. If not supplied, the default filename is `out.nc`.

A full example would be:

```
$ cis aggregate rsutcs:rsutcs_Amon_HadGEM2-A_sstClim_r1i1p1_*.nc:product=NetCDF_Gridded,kernel=mean t
```

6.1 Conditional Aggregation

Sometimes you may want to perform an aggregation over all the points that meet a certain criteria - for example, aggregating satellite data only where the cloud cover fraction is below a certain threshold. This is possible by performing a CIS evaluation on your data first - see *Using Evaluation for Conditional Aggregation*

6.2 Aggregation Examples

6.2.1 Ungridded aggregation

Aircraft Track

Original data:

```
$ cis plot TT_A:RF04.20090114.192600_035100.PNI.nc --xmin -180 --xmax -120 --ymin 0 --ymax 90
```

Aggregating onto a coarse grid:

```
$ cis aggregate TT_A:RF04.20090114.192600_035100.PNI.nc x=[-180,-120,3],y=[0,90,3] -o NCAR_RAF-1
$ cis plot TT_A:NCAR_RAF-1.nc
```

Aggregating onto a fine grid:

```
$ cis aggregate TT_A:RF04.20090114.192600_035100.PNI.nc x=[180,240,0.3],y=[0,90,0.3] -o NCAR_RAF-2
$ cis plot TT_A:NCAR_RAF-2.nc
```

Aggregating with altitude and time:

```
$ cis aggregate TT_A:RF04.20090114.192600_035100.PNI.nc t=[2009-01-14T19:30,2009-01-15T03:45,30M],z=
$ cis plot TT_A:NCAR_RAF-3.nc --xaxis time --yaxis altitude
```

Aggregating with altitude and pressure:

```
$ cis aggregate TT_A:RF04.20090114.192600_035100.PNI.nc p=[100,1100,20],z=[0,15000,500] -o NCAR_RAF-4
$ cis plot TT_A:NCAR_RAF-4.nc --xaxis altitude --yaxis air_pressure --logy
```

MODIS L3 Data

Original data:

```
$ cis plot Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf
```

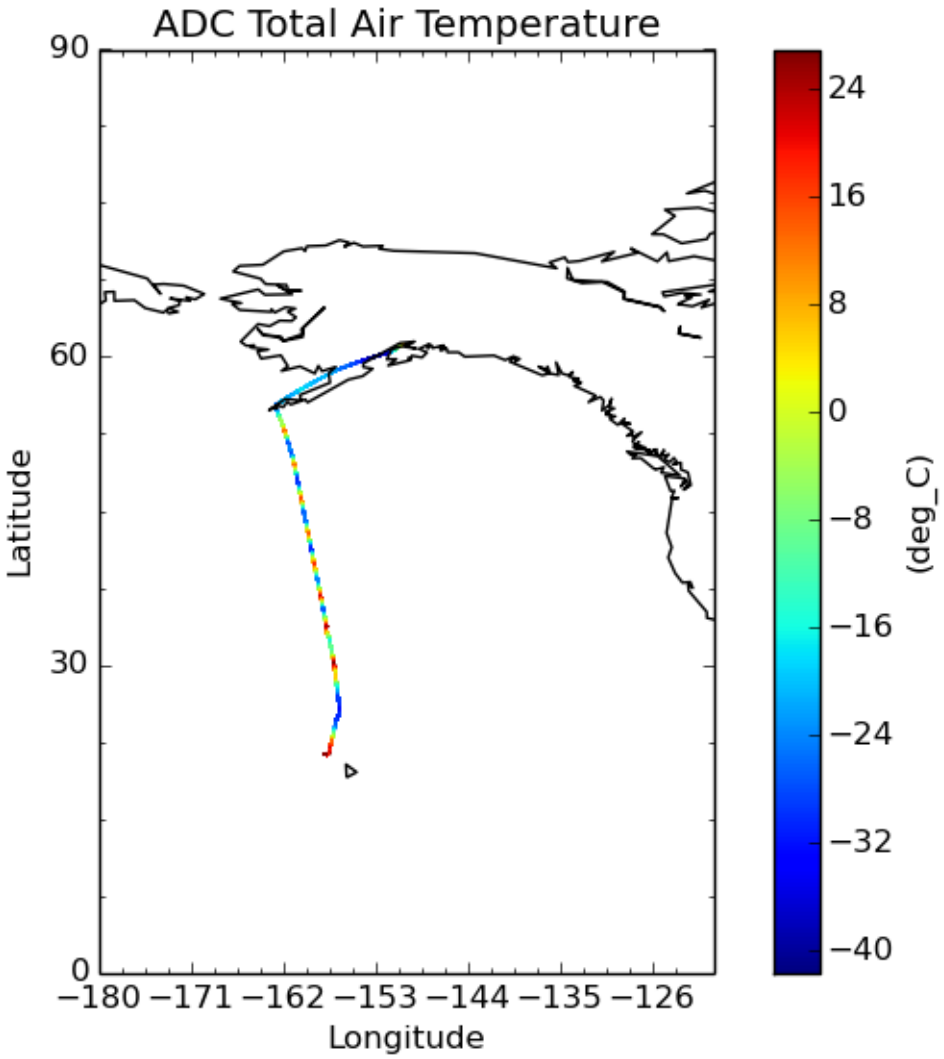
Aggregating with a mean kernel:

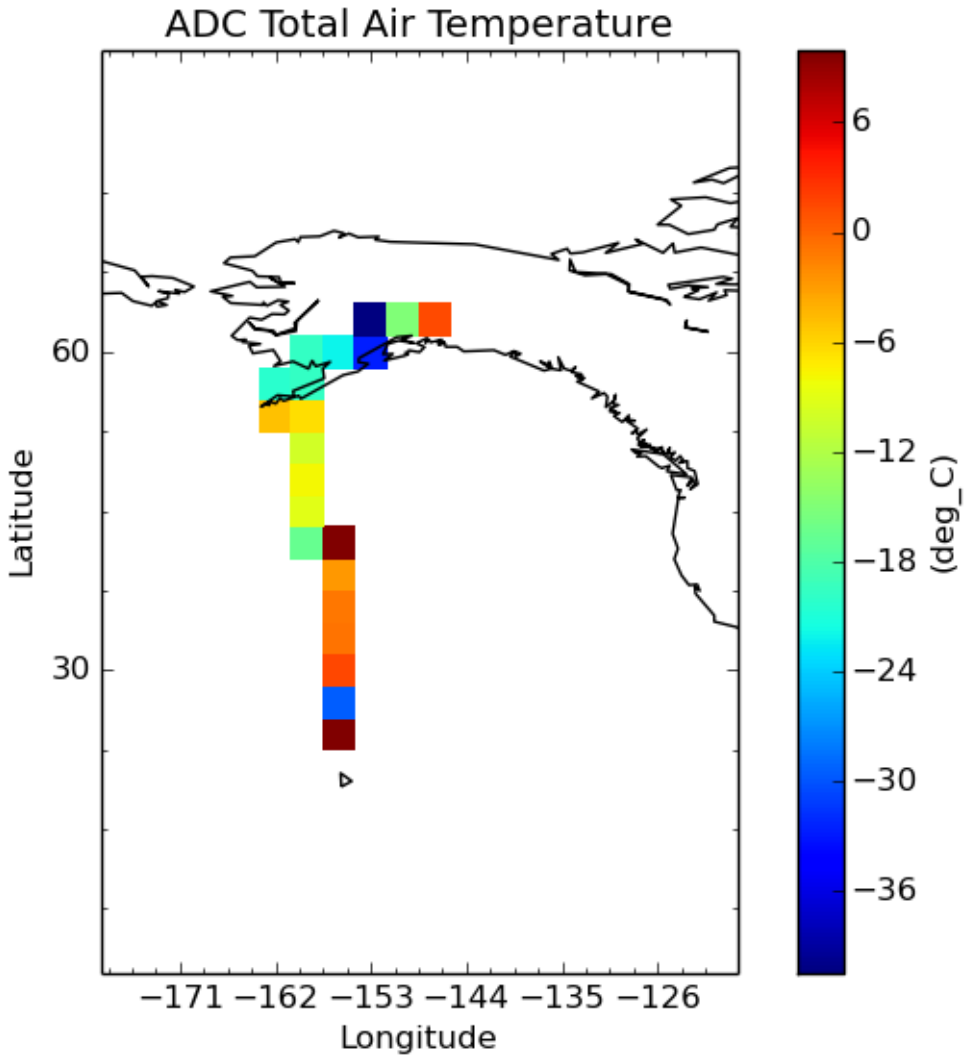
```
$ cis aggregate Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf x=[-180,180,1
$ cis plot Cloud_Top_Temperature_Mean_Mean:cloud-mean.nc
```

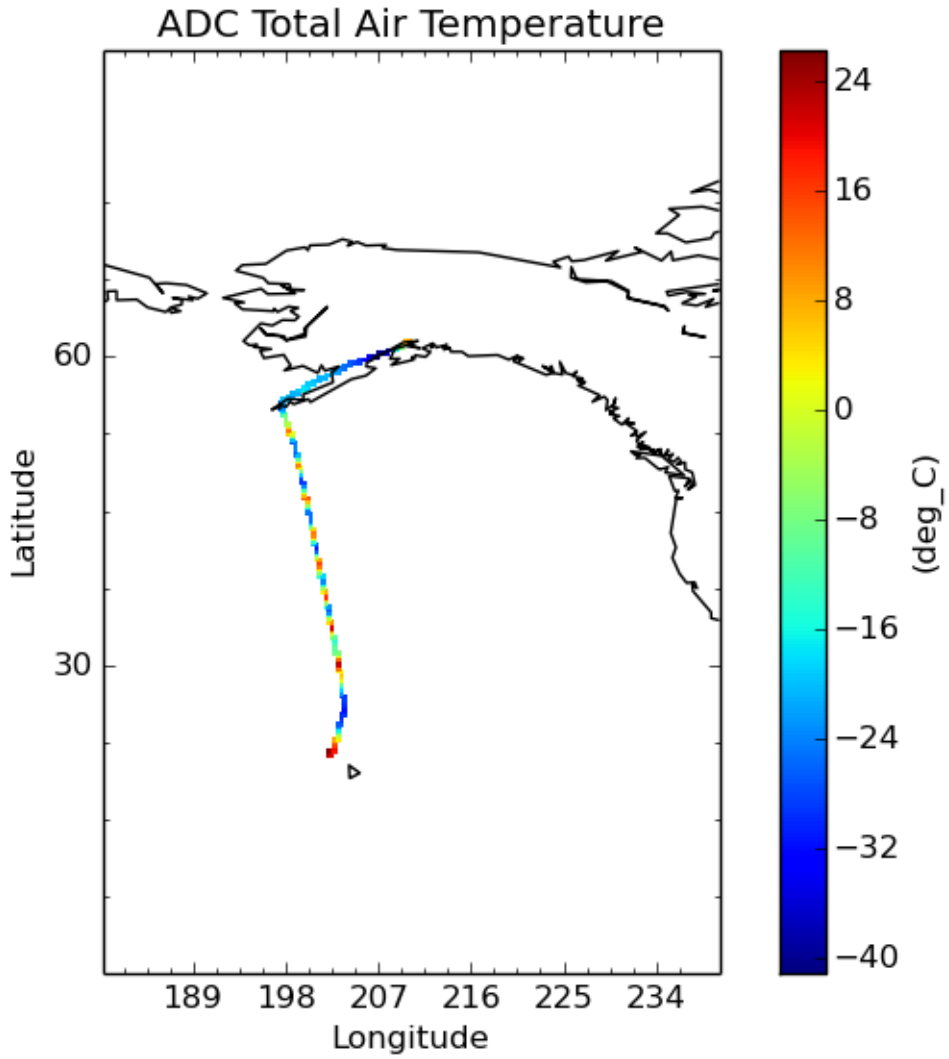
Aggregating with the standard deviation kernel:

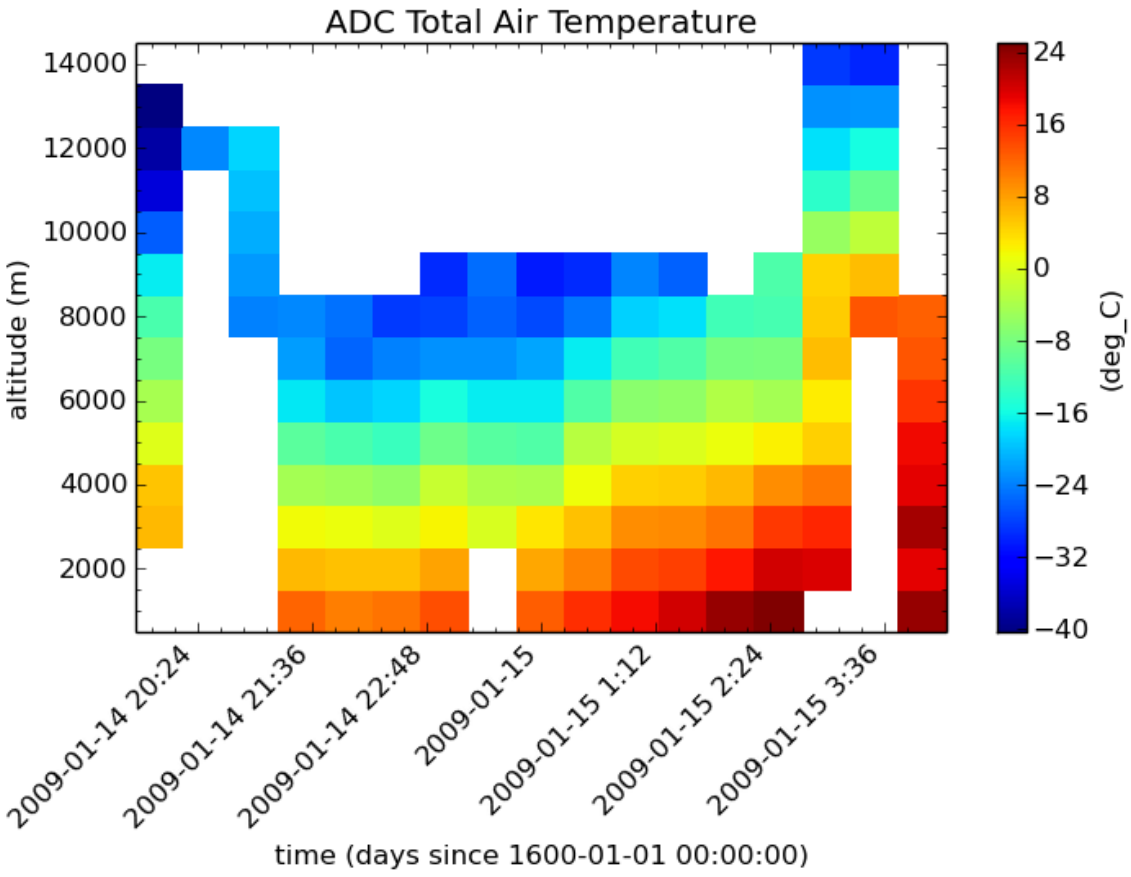
```
$ cis aggregate Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf:kernel=stddev
$ cis plot Cloud_Top_Temperature_Mean_Mean:cloud-stddev.nc &
```

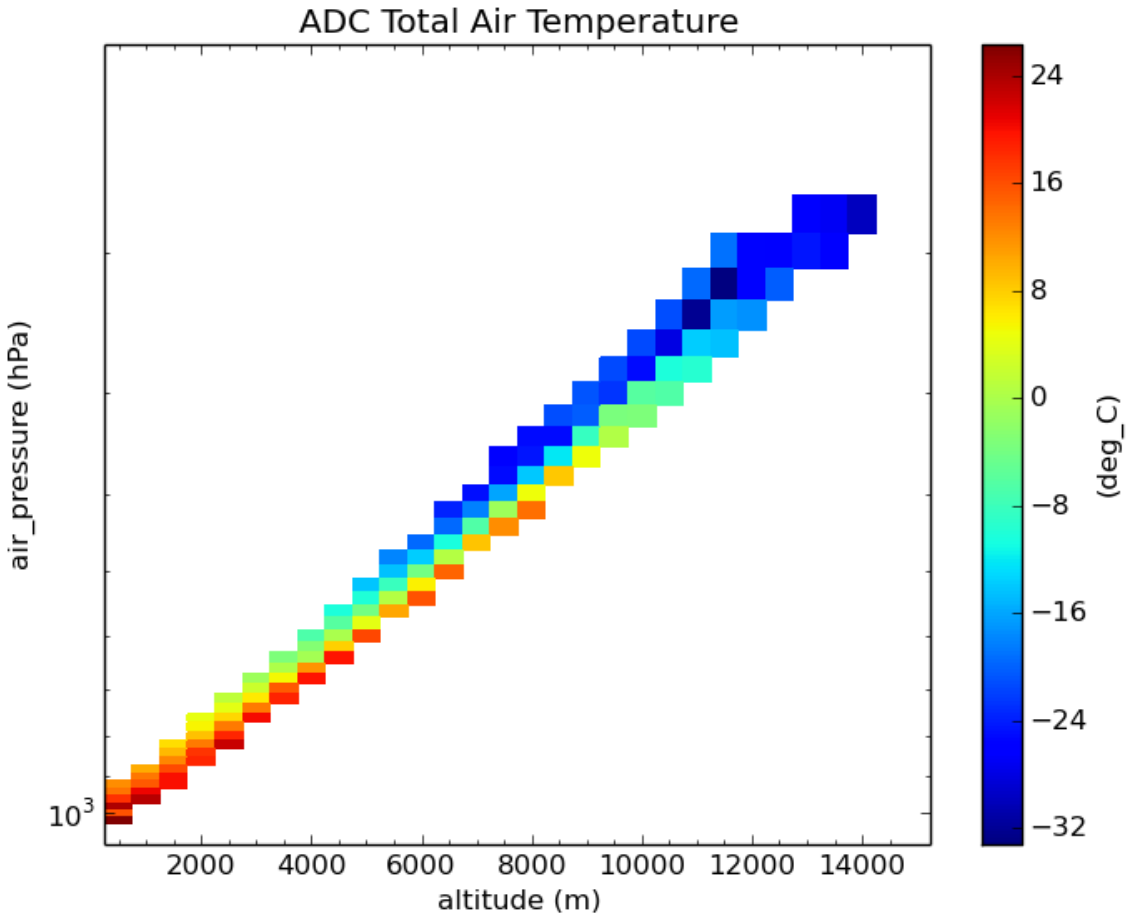
Aggregating with the maximum kernel:

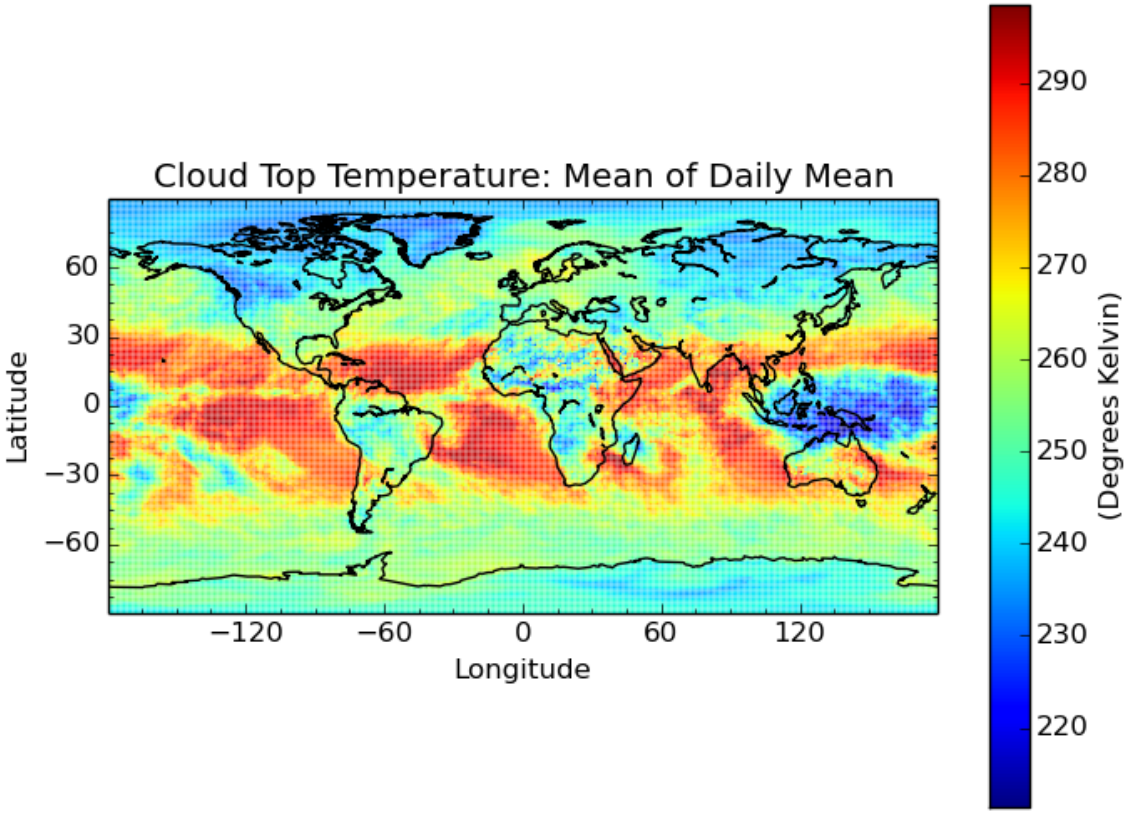

ADC Total Air Temperature
90
60
30
0
Latitude
−180 −171 −162 −153 −144 −135 −126
Longitude
24
16
8
0
−8
−16
−24
−32
−40
(deg_C)

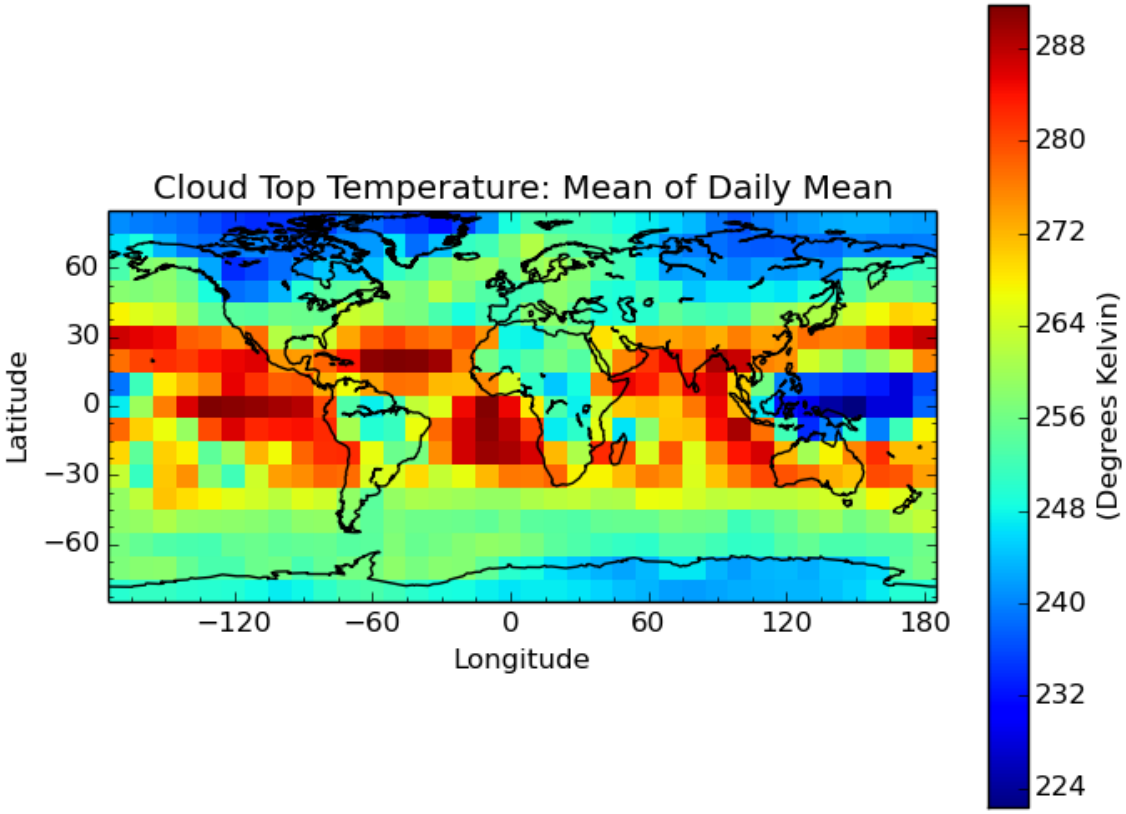

ADC Total Air Temperature
Latitude
60
30
−171 −162 −153 −144 −135 −126
Longitude
6
0
−6
−12
−18
−24
−30
−36
(deg_C)

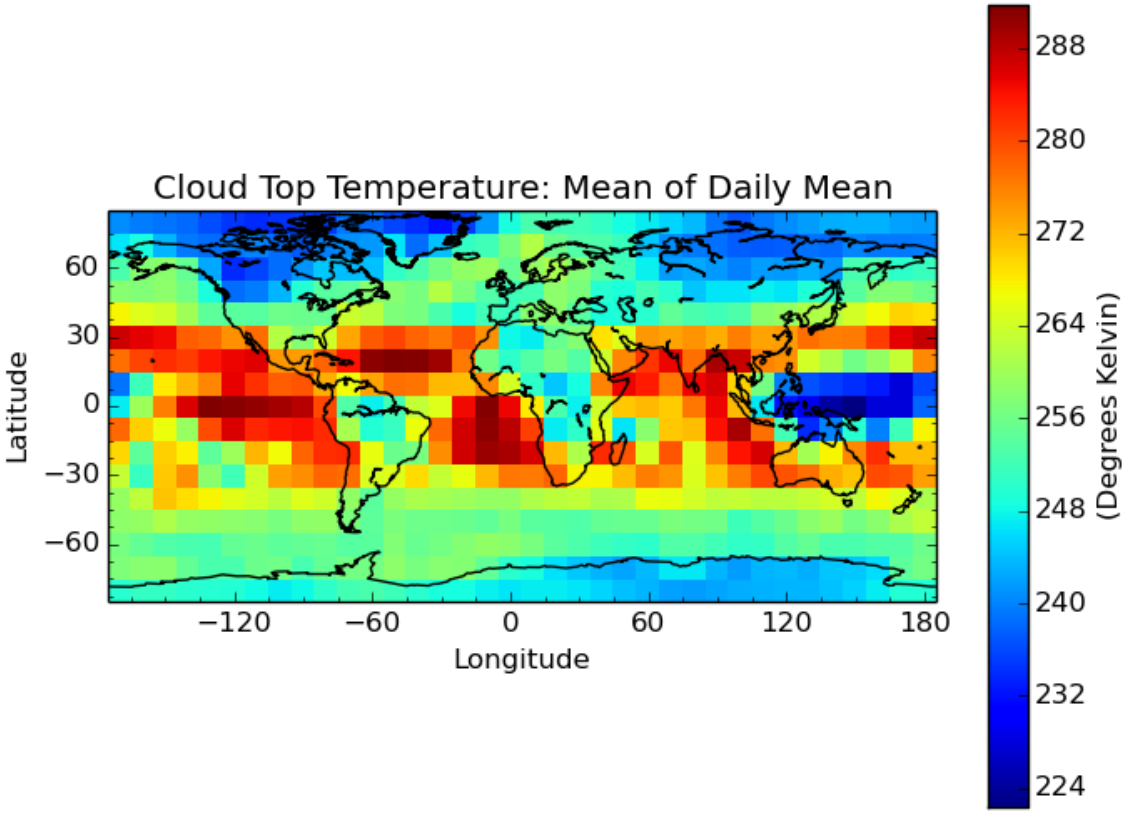

ADC Total Air Temperature
Latitude
60
30
189
198
207
216
225
234
Longitude
24
16
8
0
−8
−16
−24
−32
−40
(deg_C)


ADC Total Air Temperature
altitude (m)
14000
12000
10000
8000
6000
4000
2000
2009-01-14 20:24
2009-01-14 21:36
2009-01-14 22:48
2009-01-15
2009-01-15 1:12
2009-01-15 2:24
2009-01-15 3:36
time (days since 1600-01-01 00:00:00)
24
16
8
0
−8
−16
−24
−32
−40
(deg_C)

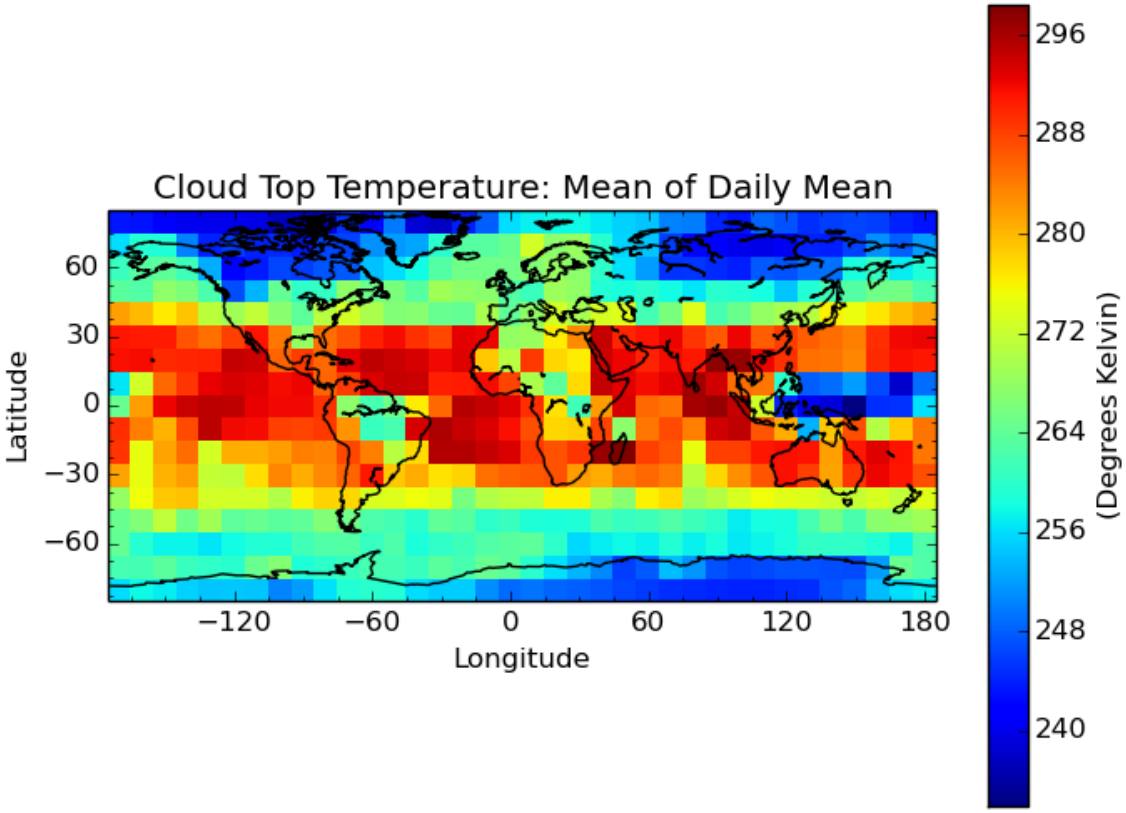

ADC Total Air Temperature
air_pressure (hPa)
10^3
2000
4000
6000
8000
10000
12000
14000
altitude (m)
24
16
8
0
−8
−16
−24
−32
(deg_C)


Cloud Top Temperature: Mean of Daily Mean
Latitude
60
30
0
−30
−60
−120
−60
0
60
120
Longitude
290
280
270
260
250
240
230
220
(Degrees Kelvin)


Cloud Top Temperature: Mean of Daily Mean
Latitude
60
30
0
−30
−60
−120
−60
0
60
120
180
Longitude
288
280
272
264
256
248
240
232
224
(Degrees Kelvin)


Cloud Top Temperature: Mean of Daily Mean
Latitude
60
30
0
−30
−60
−120
−60
0
60
120
180
Longitude
288
280
272
264
256
248
240
232
224
(Degrees Kelvin)

```
$ cis aggregate Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf:kernel=max x=
$ cis plot Cloud_Top_Temperature_Mean_Mean:cloud-max.nc
```



Aggregating with the minimum kernel:

```
$ cis aggregate Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf:kernel=min x=
$ cis plot Cloud_Top_Temperature_Mean_Mean:cloud-min.nc
```

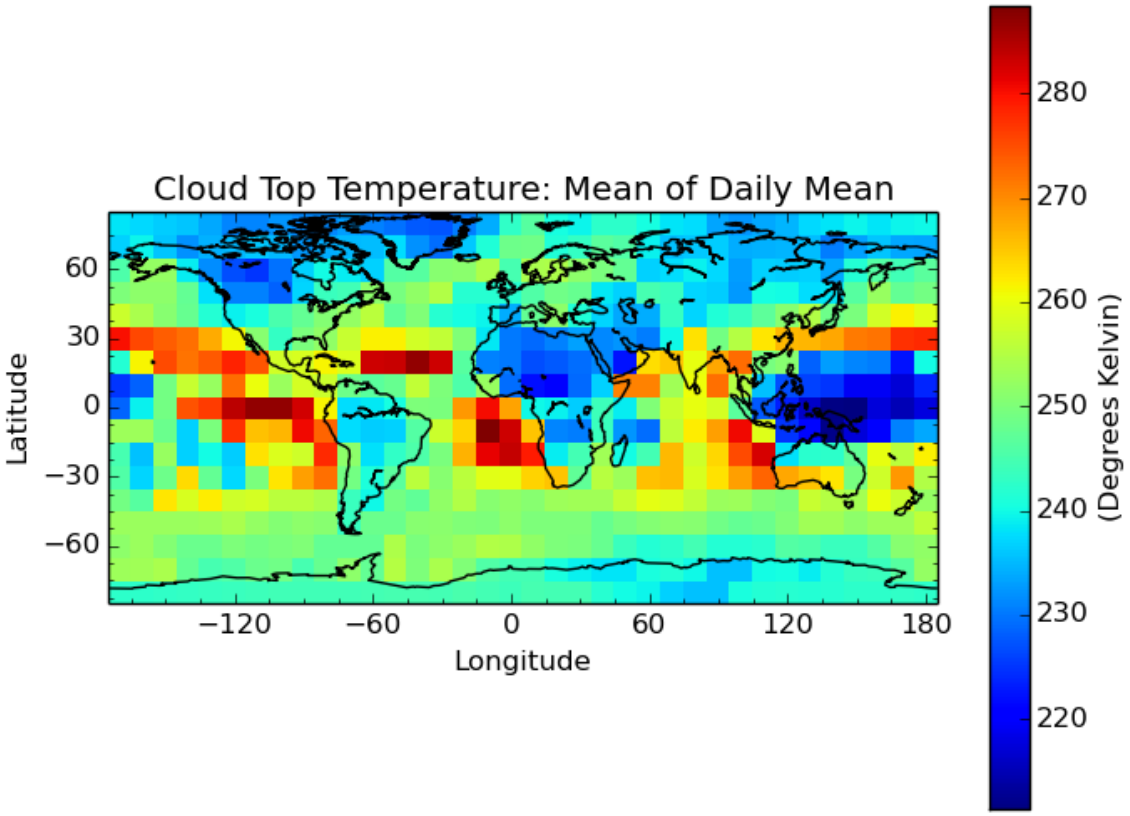
6.2.2 Gridded aggregation

Aggregating 3D model data over time and longitude to produce an averaged measure of variation with latitude:

```
$ cis aggregate rsutcs:rsutcs_Amon_HadGEM2-A_sstClim_r1i1p1_185912-188911.nc:kernel=mean t,x
$ cis plot rsutcs:/home/users/matken/agg-out.nc --xaxis latitude --yaxis rsutcs
```

This file can be found in:

```
/group_workspaces/jasmin/cis/gridded-test-data/cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1
```


Cloud Top Temperature: Mean of Daily Mean
Latitude
60
30
0
−30
−60
−120
−60
0
60
120
180
Longitude
280
270
260
250
240
230
220
(Degrees Kelvin)


img/aggregation/gridded_collapse.png

CHAPTER 7

Co-location

One of the key features of the Community Intercomparison Suite (CIS) is the ability to co-locate one or more arbitrary data sets onto a common set of coordinates. This page briefly describes how to perform co-location in a number of scenarios.

To perform co-location, run a command of the format:

```
$ cis col <datagroup> <samplegroup> -o <outputfile>
```

where:

<datagroup> is a *CIS datagroup* specifying the variables and files to read and is of the format `<variable>...:<filename>[:product=<productname>]` where:

- `<variable>` is a mandatory variable or list of variables to use.
- `<filenames>` is a mandatory file or list of files to read from.
- `<productname>` is an optional CIS data product to use (see *Data Products*):

See *Datagroups* for a more detailed explanation of datagroups.

<samplegroup> is of the format `<filename>[:<options>]` The available options are described in more detail below. They are entered in a comma separated list, such as `variable=Temperature,colocator=bin,kernel=mean`. Not all combinations of colocator and data are available; see *Available Colocators*.

- `<filename>` is a single filename with the points to colocate onto.
- `variable` is an optional argument used to specify which variable's coordinates to use for colocation. If a variable is specified, a missing value will be set in the output file at every point for which the sample variable has a missing value. If a variable is not specified, non-missing values will be set at all sample points unless colocation at a point does not result in a valid value.
- `colocator` is an optional argument that specifies the colocation method. Parameters for the colocator, if any, are placed in square brackets after the colocator name, for example, `colocator=box[fill_value=-999,h_sep=1km]`. If not specified, a *Default Colocator* is identified for your data / sample combination. The colocators available are:
  - `bin` For use only with ungridded data and gridded sample points. Data points are placed in bins corresponding to the cell bounds surrounding each grid point. The bounds are taken from the gridded data if they are defined, otherwise the mid-points between grid points are used. The binned points should then be processed by one of the kernels to give a numeric value for each bin.
  - `box` For use with gridded and ungridded sample points and data. A search region is defined by the parameters and points within the defined separation of each sample point are associated with the point.

The points should then be processed by one of the kernels to give a numeric value for each bin. The parameters defining the search box are:

* `h_sep` - the horizontal separation. The units can be specified as km or m (for example `h_sep=1.5km`); if none are specified then the default is km.
* `a_sep` - the altitude separation. The units can be specified as km or m, as for h_sep; if none are specified then the default is m.
* `p_sep` - the pressure separation. This is not an absolute separation as for h_sep and a_sep, but a relative one, so is specified as a ratio. For example a constraint of p_sep = 2, for a point at 10 hPa, would cover the range 5 hPa < points < 20 hPa. Note that p_sep >= 1.
* `t_sep` - the time separation. This can be specified in years, months, days, hours, minutes or seconds using `PnYnMnDTnHnMnS` (the T separator can be replaced with a colon or a space, but if using a space quotes are required). For example to specify a time separation of one and a half months and thirty minutes you could use `t_sep=P1M15DT30M`. It is worth noting that the units for time comparison are fractional days, so that years are converted to the number of days in a Gregorian year, and months are 1/12th of a Gregorian year.

If `h_sep` is specified, a k-d tree index based on longitudes and latitudes of data points is used to speed up the search for points. It h_sep is not specified, an exhaustive search is performed for points satisfying the other separation constraints.

- `lin` For use with gridded source data only. A value is calculated by linear interpolation for each sample point.
- `nn` For use with gridded source data only. The data point closest to each sample point is found, and the data value is set at the sample point.
- `dummy` For use with ungridded data only. Returns the source data as the colocated data irrespective of the sample points. This might be useful if variables from the original sample file are wanted in the output file but are already on the correct sample points.

Colocators have the following general optional parameters, which can be used in addition to any specific ones listed above:

- `fill_value` - The numerical value to apply to the colocated point if there are no points which satisfy the constraint.
- `var_name` - Specifies the name of the variable in the resulting NetCDF file.
- `var_long_name` - Specifies the variable's long name.
- `var_units` - Specifies the variable's units.

- `kernel` is used to specify the kernel to use for colocation methods that create an intermediate set of points for further processing, that is box and bin. The default kernel for box and bin is *moments*. The built-in kernel methods currently available are:
  - `moments` - **Default**. This is an averaging kernel that returns the mean, standard deviation and the number of points remaining after the specified constraint has been applied. This can be used for gridded or ungridded sample points where the colocator is one of 'bin' or 'box'. The names of the variables in the output file are the name of the input variable with a suffix to identify which quantity they represent:
    * *Mean* - no suffix - the mean value of all data points which were mapped to that sample grid point (data points with missing values are excluded)
    * *Standard Deviation* - suffix: `_std_dev` - The corrected sample standard deviation (i.e. 1 degree of freedom) of all the data points mapped to that sample grid point (data points with missing values are excluded)

        * *Number of points* - suffix: `_num_points` - The number of data points mapped to that sample grid point (data points with missing values are excluded)

    - `mean` - an averaging kernel that returns the mean values of any points found by the colocation method
    - `nn_t` (or `nn_time`) - nearest neighbour in time algorithm
    - `nn_h` (or `nn_horizontal`) - nearest neighbour in horizontal distance
    - `nn_a` (or `nn_altitude`) - nearest neighbour in altitude
    - `nn_p` (or `nn_pressure`) - nearest neighbour in pressure (as in a vertical coordinate). Note that similarly to the `p_sep` constraint that this works on the ratio of pressure, so the nearest neighbour to a point with a value of 10 hPa, out of a choice of 5 hPa and 19 hPa, would be 19 hPa, as 19/10 < 10/5.

- `product` is an optional argument used to specify the type of files being read. If omitted, the program will attempt to determine which product to use based on the filename, as listed at *Reading*.

`<outputfile>` is an optional argument specifying the file to output to. This will be automatically given a `.nc` extension if not present and if the output is ungridded, will be prepended with `cis-` to identify it as a CIS output file. This must not be the same file path as any of the input files. If not provided, the default output filename is *out.nc*

A full example would be:

```
$ cis col rain:"my_data_??.*" my_sample_file:colocator=box[h_sep=50km,t_sep=6000S],kernel=nn_t -o my_
```

Warning: When colocating two data sets with different spatio-temporal domains, the sampling points should be within the spatio-temporal domain of the source data. Otherwise, depending on the co-location options selected, strange artefacts can occur, particularly with linear interpolation. Spatio-temporal domains can be reducded in CIS with *Aggregation* or *Subsetting*.

7.1 Available Colocators and Kernels

Colocation type (data -> sample)	Available Colocators	Default Colocator	Default Kernel
Gridded -> gridded	`lin`, `nn`, `box`	`lin`	*None*
Ungridded -> gridded	`bin`, `box`	`bin`	`moments`
Gridded -> ungridded	`nn`, `lin`	`nn`	*None*
Ungridded -> ungridded	`box`	`box`	`moments`

7.2 Colocation output files

All ungridded co-location output files are prefixed with `cis-` and both ungridded and gridded data files are suffixed with `.nc` (so there is no need to specify the extension in the output parameter). This is to ensure the cis data product is always used to read co-located ungridded data.

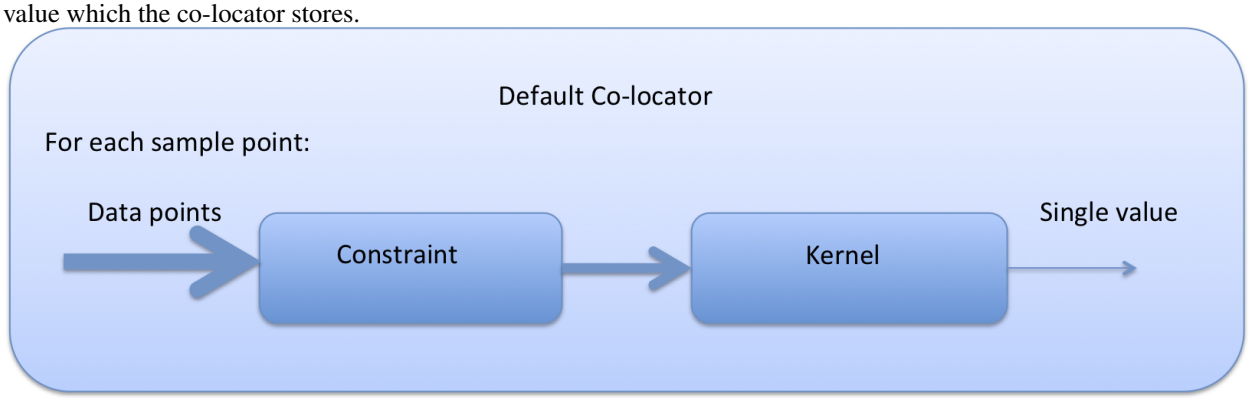
It is worth noting that in the process of colocation all of the data and sample points are represented as 1-d lists, so any structural information about the input files is lost. This is done to ensure consistency in the colocation output. This means, however, that input files which may have been plotable as, for example, a heatmap may not be after co-location. In this situation plotting the data as a scatter plot will yield the required results.

Each co-located output variable has a history attributed created (or appended to) which contains all of the parameters and file names which went into creating it. An example might be:

```
double mass_fraction_of_cloud_liquid_water_in_air(pixel_number) ;
    ...
    mass_fraction_of_cloud_liquid_water_in_air:history = "Colocated onto sampling from:   [\'/test/te
        "variable: mass_fraction_of_cloud_liquid_water_in_air\n",
        "with files: [\'/test/test_files/xenida.pah9440.nc\']\n",
        "using colocator: DifferenceColocator\n",
        "colocator parameters: {}\n",
        "constraint method: None\n",
        "constraint parameters: None\n",
        "kernel: None\n",
        "kernel parameters: None" ;
    mass_fraction_of_cloud_liquid_water_in_air:shape = 30301 ;
double difference(pixel_number) ;
    ...
```

7.3 Basic colocation design

The diagram below demonstrates the basic design of the co-location system, and the roles of each of the components. In the simple case of the default co-locator (which returns only one value) the Colocator loops over each of the sample points, calls the relevant constraint to reduce the number of data points, and then the kernel which returns a single value which the co-locator stores.



It is useful to understand that when a sample variable is specified that contains masked values (those with a fill_value) this is not taken into account when creating the list of sample points. E.g. the full list of coordinates is used from the file, regardless of the values of the sample variable.

On the contrary when a data variable is read in (which is to be co-located onto the sample) any masked values are ignored. That is, any value in the data variable which is equal to the fill_value is not considered for colocation, as it is treated as an empty value.

On their own each of these statements seem sensible, but together may lead to unexpected results if, for example, a variable from a file is co-located onto itself using the DefaultColocator. In this situation, the sampling from the file is used to determine the sample points regardless of fill_value, and the variable is co-located on to this (ignoring any fill_values). This results in an output file where the masked (or missing) values are 'filled-in' by the co-locator using whichever kernel was specified - see Figure 2a below. Using the DummyColocator simply returns the original masked values as no filling in is done (see 2b), and similarly for the difference co-locator when co-located onto itself the difference variable retains the mask as a non-value minus any other number is still a non-value (see 2c).

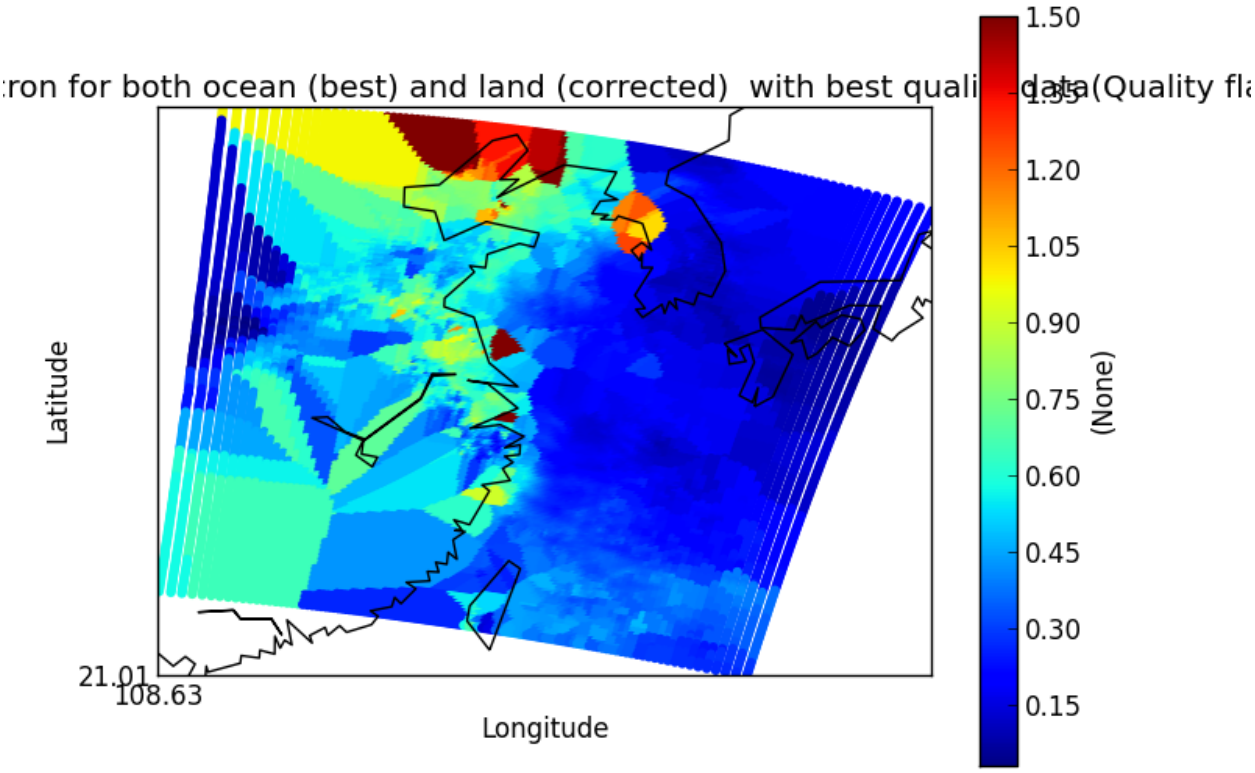


Fig. 7.1: Figure 2a

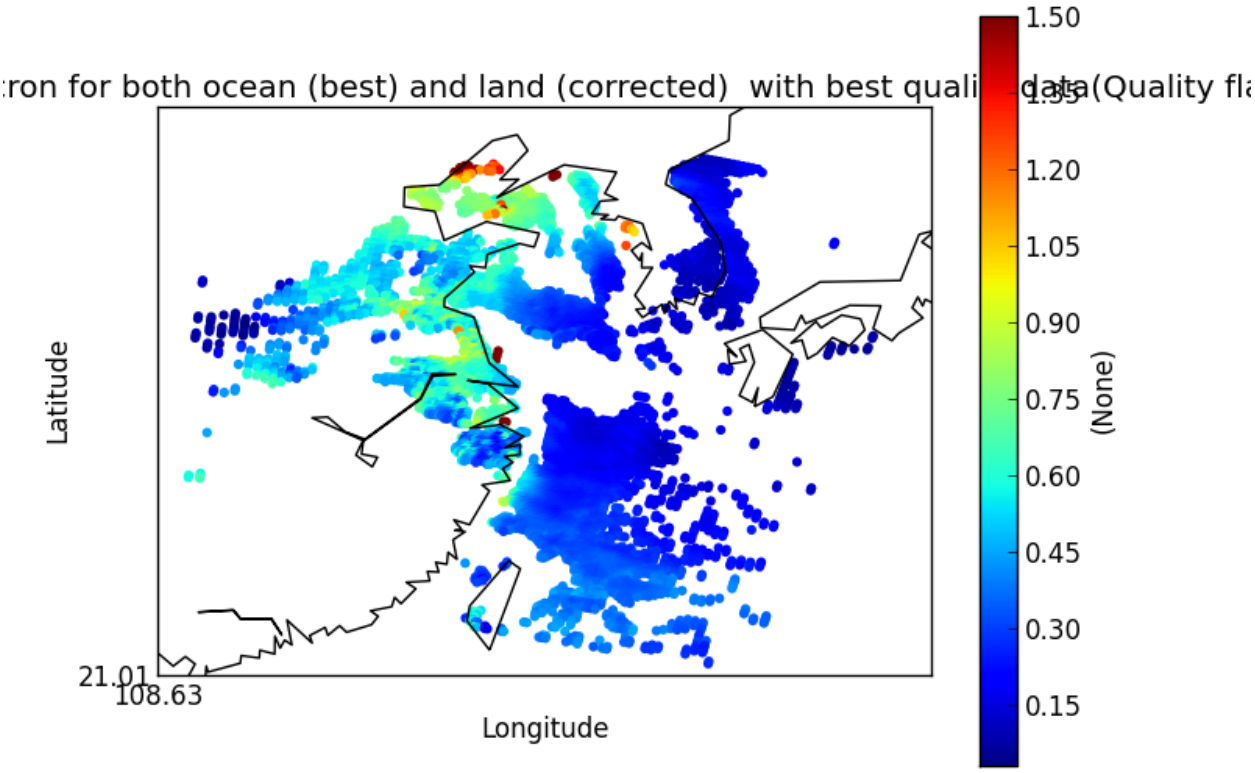


Fig. 7.2: Figure 2b

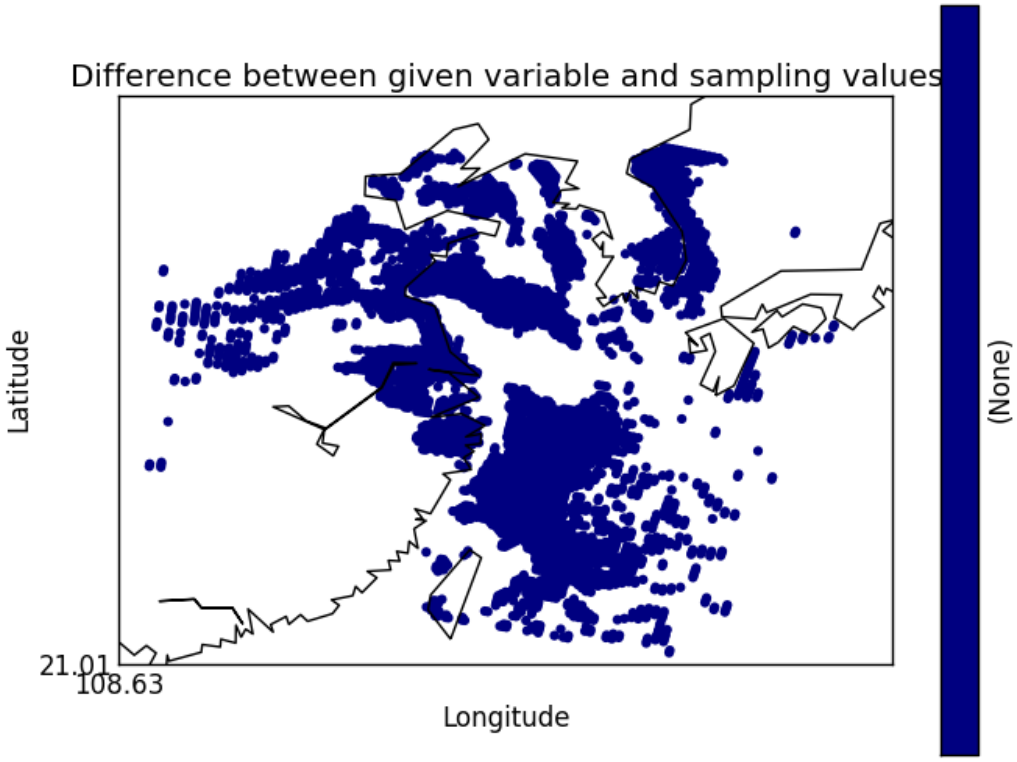


Fig. 7.3: Figure 2c

7.4 Writing your own plugins

The colocation framework was designed to make it easy to write your own plugins. Plugins can be written to create new kernels, new constraint methods and even whole colocation methods. See *Colocation Design* for more details

CHAPTER 8

Colocation Examples

8.1 Ungridded to Ungridded Colocation Examples

8.1.1 Ungridded data with vertical component

First subset two Caliop data files:

```
$ cis subset Temperature:CAL_LID_L2_05kmAPro-Prov-V3-01.2009-12-31T23-36-08ZN.hdf x=[170,180],y=[60,8
$ cis subset Temperature:CAL_LID_L2_05kmAPro-Prov-V3-01.2010-01-01T00-22-28ZD.hdf x=[170,180],y=[60,8
```

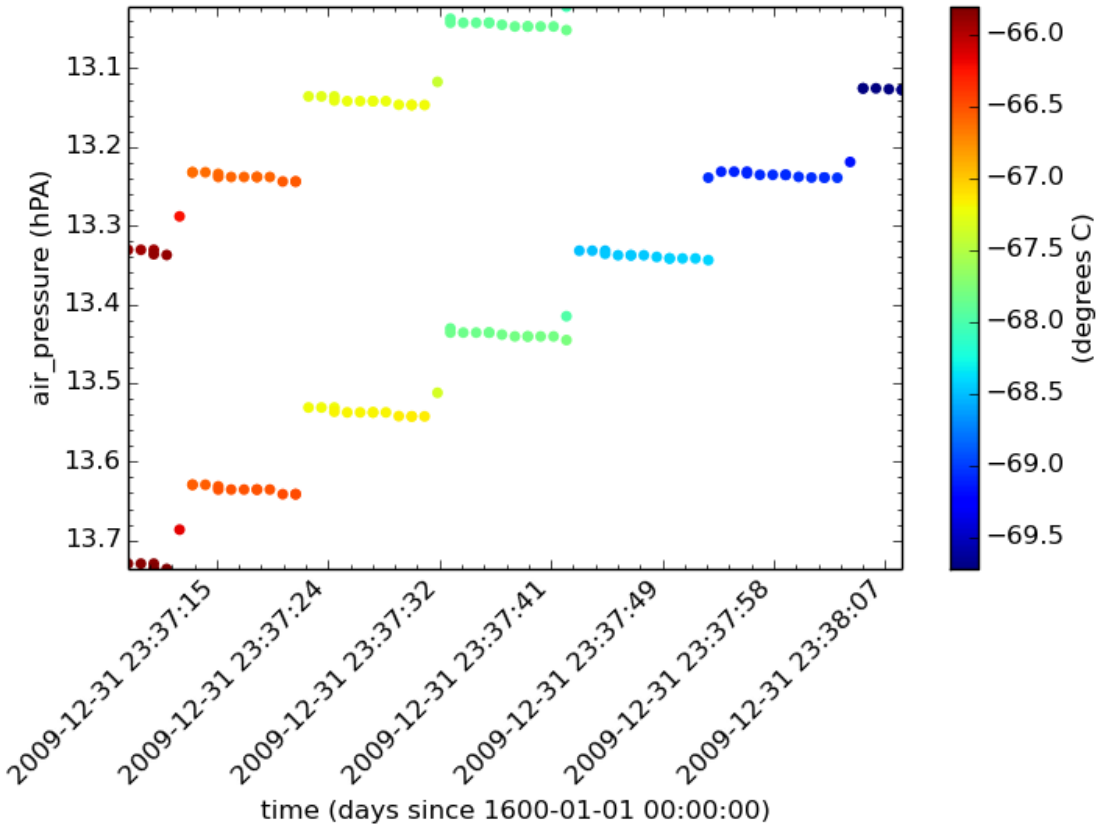
Results of subset can be plotted with:

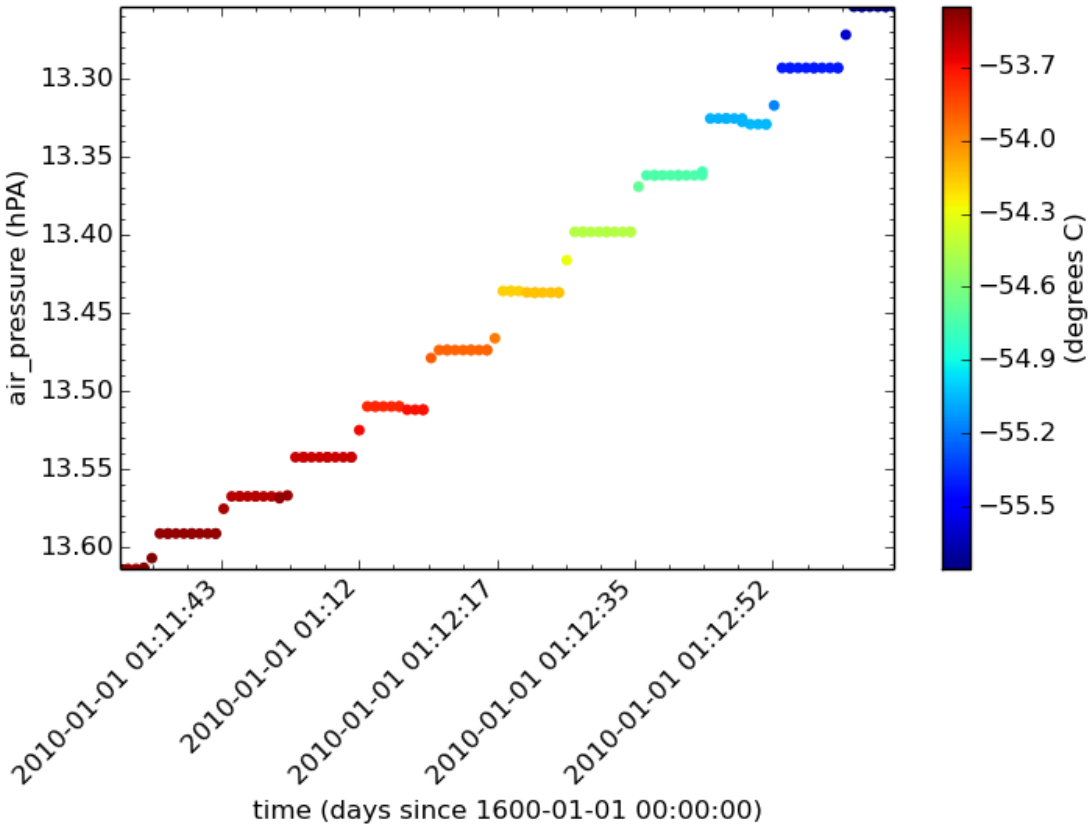
```
$ cis plot Temperature:cis-2009.nc --itemwidth 25 --xaxis time --yaxis air_pressure
$ cis plot Temperature:cis-2010.nc --itemwidth 25 --xaxis time --yaxis air_pressure
```

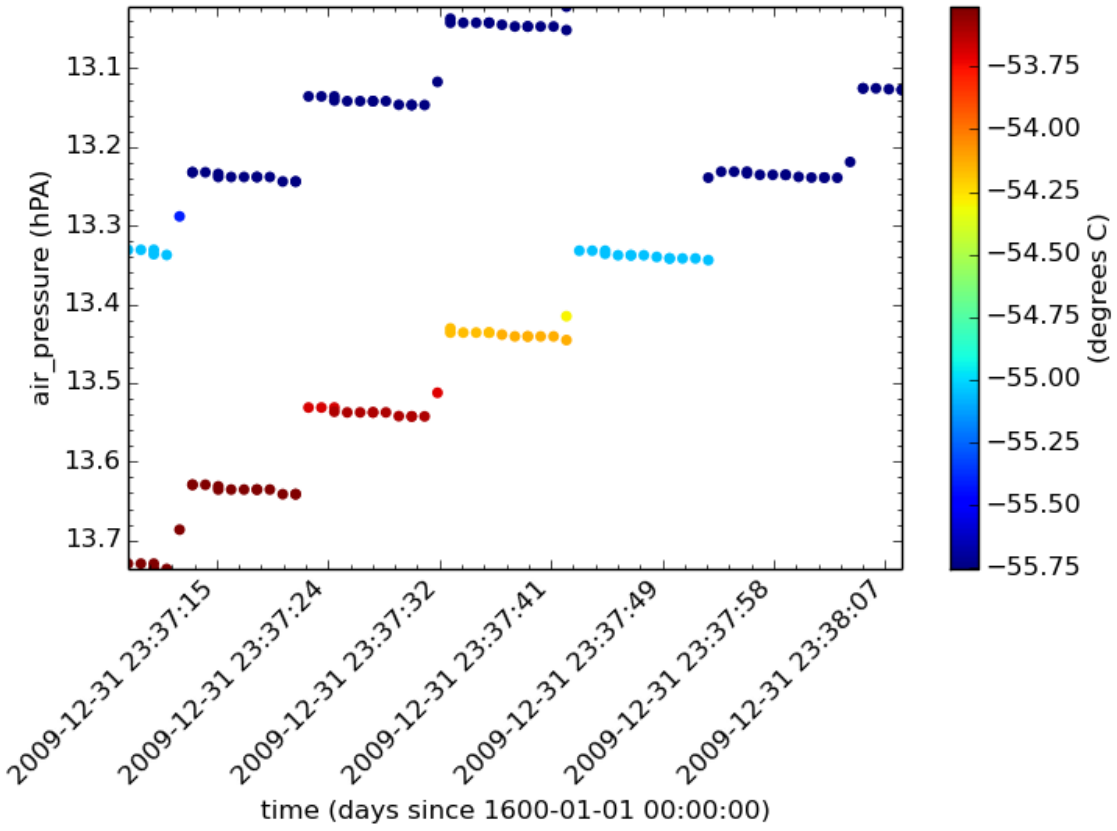
Then colocate data, and plot output:

```
$ cis col Temperature:cis-2010.nc cis-2009.nc:colocator=box[p_sep=1.1],kernel=nn_p
$ cis plot Temperature:cis-out.nc --itemwidth 25 --xaxis time --yaxis air_pressure
```

The output for the two subset data files, and the colocated data should look like:


air_pressure (hPA)
13.1
13.2
13.3
13.4
13.5
13.6
13.7
2009-12-31 23:37:15
2009-12-31 23:37:24
2009-12-31 23:37:32
2009-12-31 23:37:41
2009-12-31 23:37:49
2009-12-31 23:37:58
2009-12-31 23:38:07
time (days since 1600-01-01 00:00:00)
−66.0
−66.5
−67.0
−67.5
−68.0
−68.5
−69.0
−69.5
(degrees C)


13.30
13.35
13.40
13.45
13.50
13.55
13.60
air_pressure (hPA)
2010-01-01 01:11:43
2010-01-01 01:12
2010-01-01 01:12:17
2010-01-01 01:12:35
2010-01-01 01:12:52
time (days since 1600-01-01 00:00:00)
−53.7
−54.0
−54.3
−54.6
−54.9
−55.2
−55.5
(degrees C)



File Locations

The files used above can be found at:

```
/group_workspaces/jasmin/cis/data/caliop/CAL-LID-L2-05km-APro
```

8.1.2 Ungridded data colocation using k-D tree indexing

These examples show the syntax for using the k-D tree optimisation of the separation constraint. The indexing is only by horizontal position.

Nearest-Neighbour Kernel

The first example is of Aerosol CCI data on to the points of a MODIS L3 file (which is an ungridded data file but with points lying on a grid).

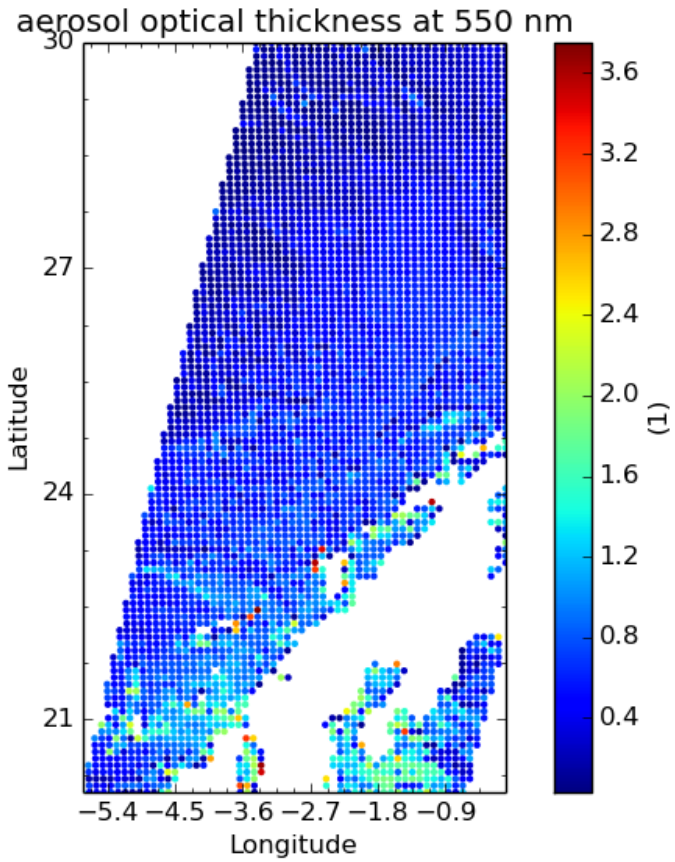
Subset to a relevant region:

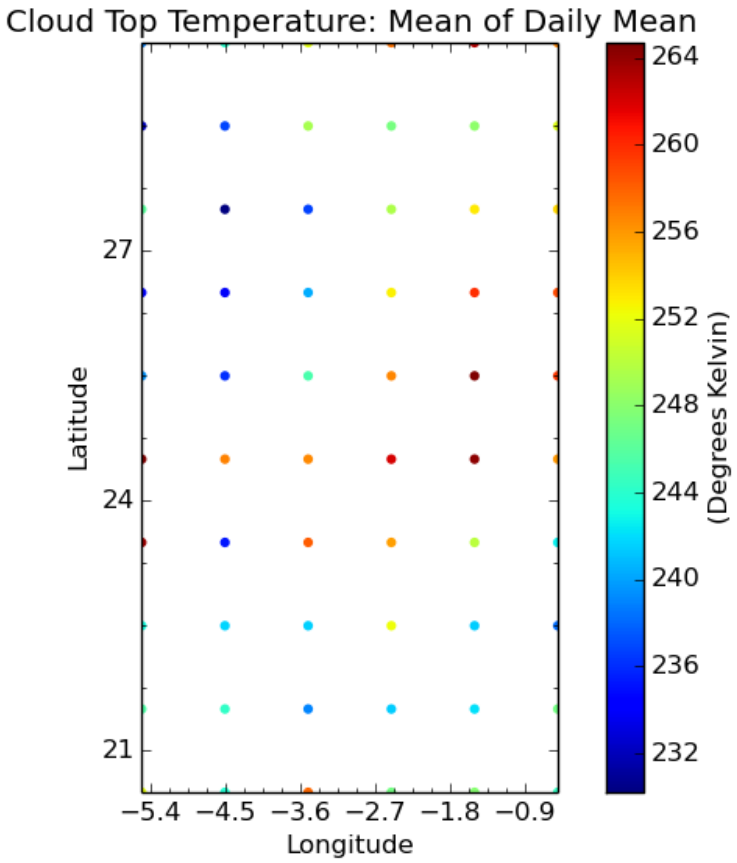
```
$ cis subset AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc x=[-6,0
$ cis subset Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf x=[-6,0],y=[20,3
```

The results of subsetting can be plotted with:

```
$ cis plot AOD550:cis-AOD550n_3.nc --itemwidth 10
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MOD08n_3.nc --itemwidth 20
```

These should look like:





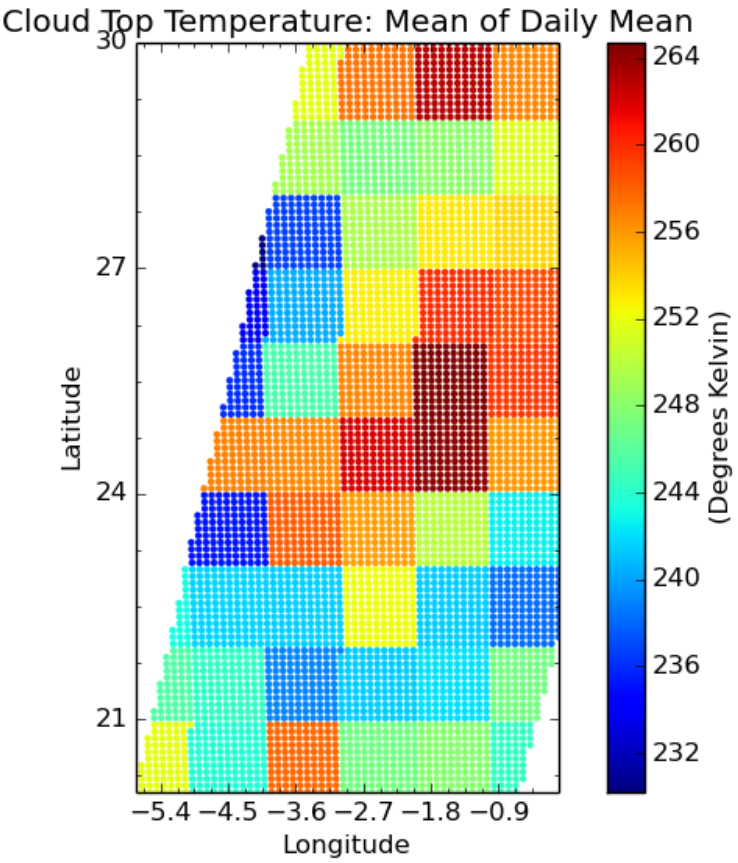
To colocate with the nearest-neighbour kernel use:

```
$ cis col Cloud_Top_Temperature_Mean_Mean:cis-MOD08n_3.nc cis-AOD550n_3.nc:colocator=box[h_sep=150],
```

This can be plotted with:

```
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MOD08_on_AOD550_nn_kdt.nc --itemwidth 10
```

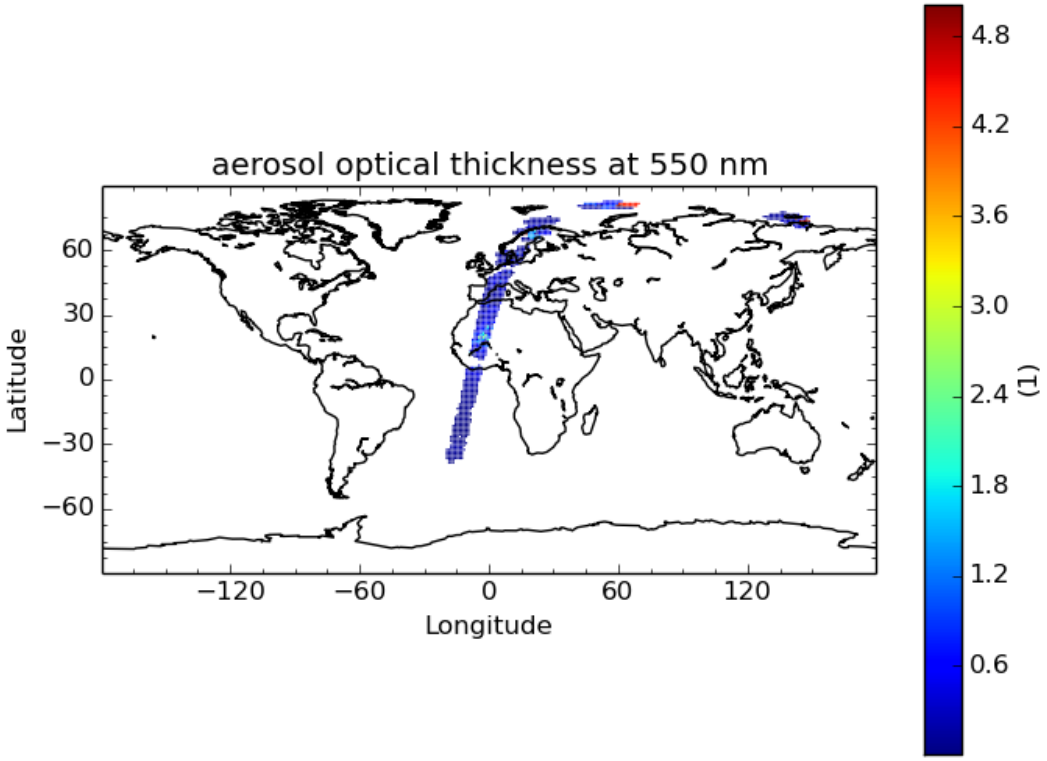
The sample points are more closely spaced than the data points, hence a patchwork effect is produced.



Colocating the full Aerosol CCI file on to the MODIS L3 with:

```
$ cis col AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc MOD08_E3.A
```

gives the following result



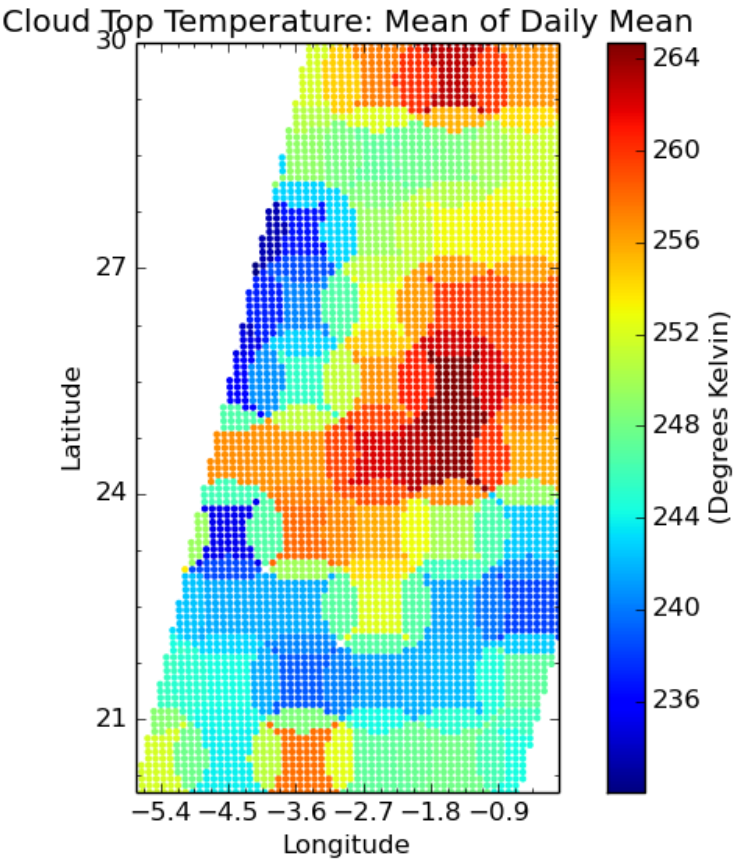
Mean Kernel

This example is similar to the first nearest-neighbour colocation above:

```
$ cis col Cloud_Top_Temperature_Mean_Mean:cis-MOD08n_3.nc cis-AOD550n_3.nc:colocator=box[h_sep=75],ke
```

Plotting this again gives a granular result:

```
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MOD08_on_AOD550_hsep_75km.nc --itemwidth 10
```

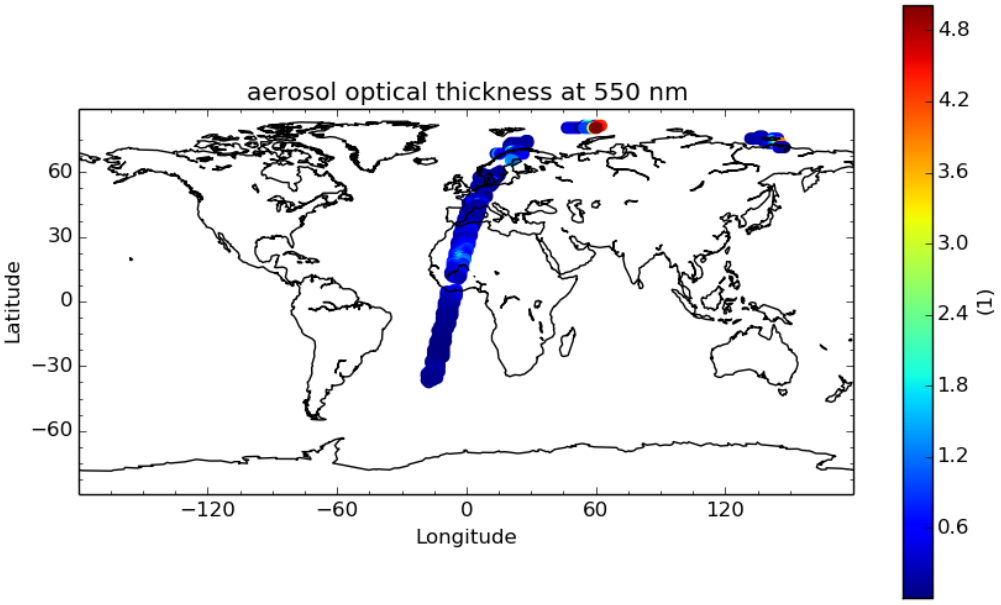


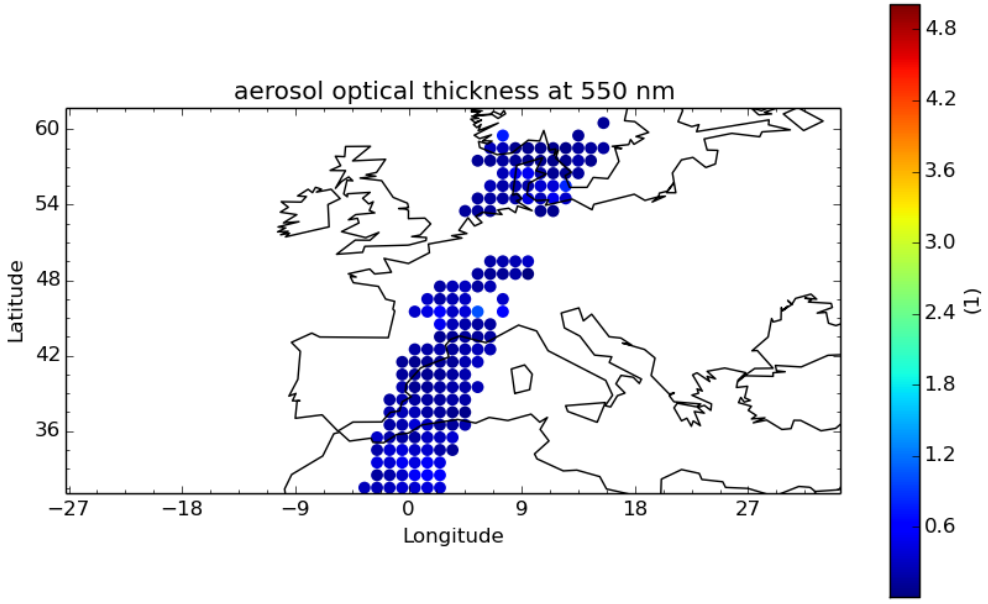
This example colocates the Aerosol CCI data on to the MODIS L3 grid:

```
$ cis col AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc MOD08_E3.A
```

This can be plotted as follows, with the full image and zoomed into a representative section show below:

```
$ cis plot AOD550:cis-AOD550_on_MOD08_kdt_hsep_50km_full.nc --itemwidth 50
```



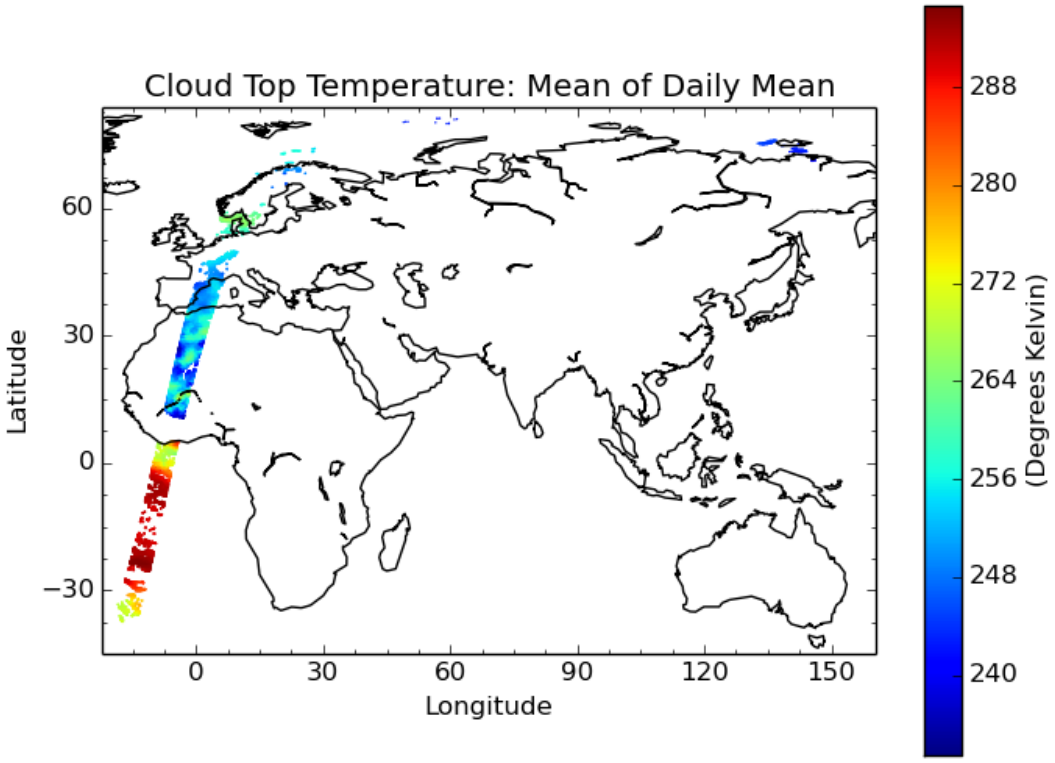


The reverse colocation can be performed with this command (taking about 7 minutes):

```
$ cis col Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf 20080612093821-ESAC
```

Plotting it with this command gives the result below:

```
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MOD08_on_AOD550_kdt_hsep_100km_var_full.nc
```

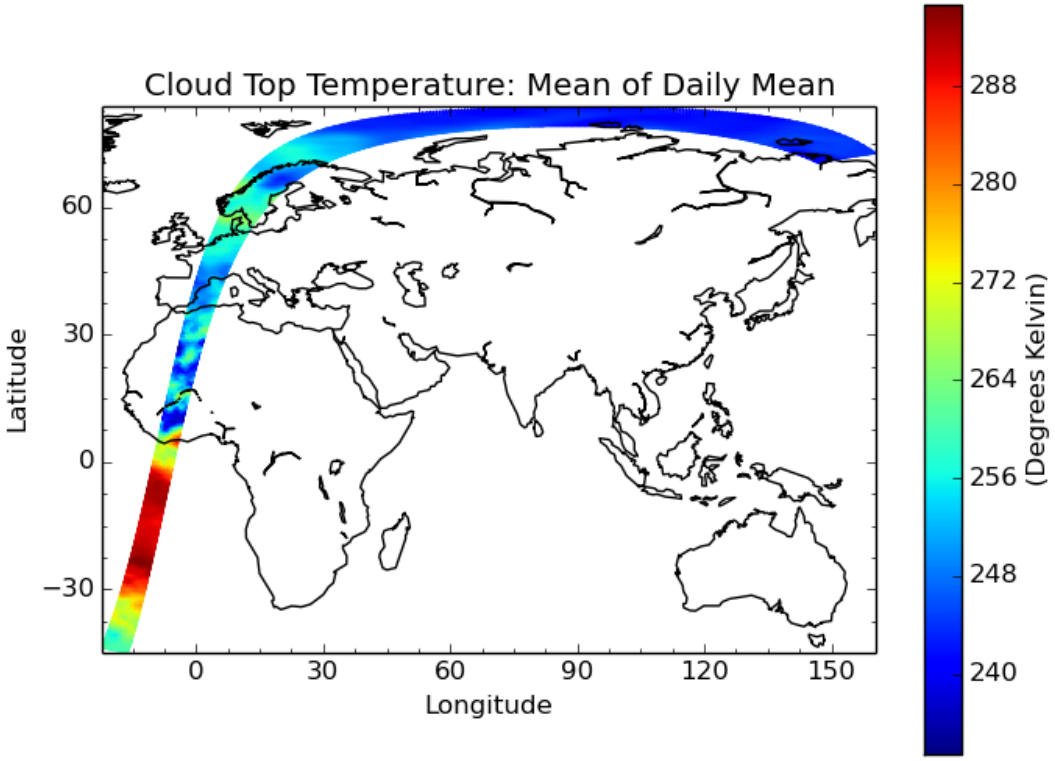


Omitting the variable option in the sample group gives colocated values over a full satellite track (taking about 30 minutes):

```
$ cis col Cloud_Top_Temperature_Mean_Mean:MOD08_E3.A2010009.005.2010026072315.hdf 20080612093821-ESA
```

Plotting it with this command gives the result below:

```
$ cis plot Cloud_Top_Temperature_Mean_Mean:cis-MOD08_on_AOD550_kdt_hsep_100km_full.nc
```



File Locations

The files used above can be found at:

```
/group_workspaces/jasmin/cis/jasmin_cis_repo_test_files/
  20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc
  MOD08_E3.A2010009.005.2010026072315.hdf
```

8.2 Examples of co-location of ungridded data on to gridded

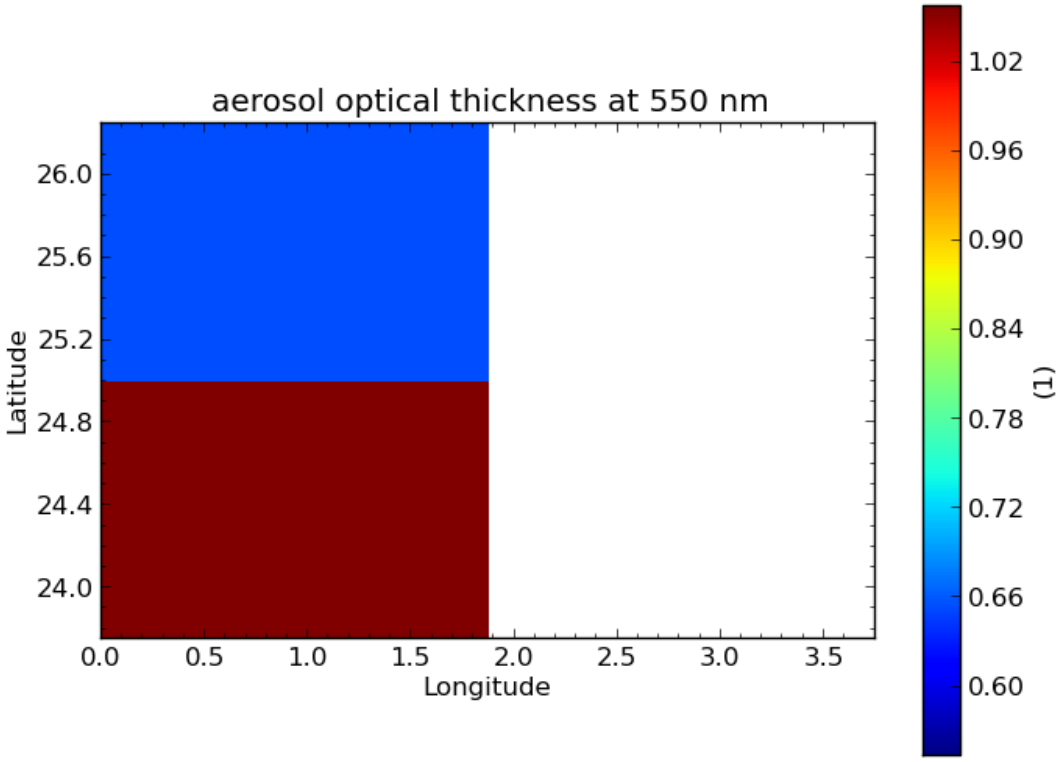
8.2.1 Simple Example of Aerosol CCI Data on to a 4x4 Grid

This is a trivial example that co-locates on to a 4x4 spatial grid at a single time:

```
$ cis subset tas:tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc x=[0,2],y=[24,26],t=[2008-06-12

$ cis subset AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc x=[0,2

$ cis col AOD550:cis-AOD550n_1.nc tas_1.nc:colocator=bin[fill_value=-9999.0],kernel=mean -o AOD550_on

$ cis plot AOD550:AOD550_on_tas_1.nc
```

Note that for ungridded gridded co-location, and the colocator must be one bin or box and a kernel such as "mean" must be used.

The plotted image looks like:



8.2.2 Aerosol CCI with Three Time Steps

This example involves co-location on to a grid with three time steps. The ungridded data all has times within the middle step, so the output has missing values for all grid points with the time equal to the first or third value. This can be seen using ncdump:

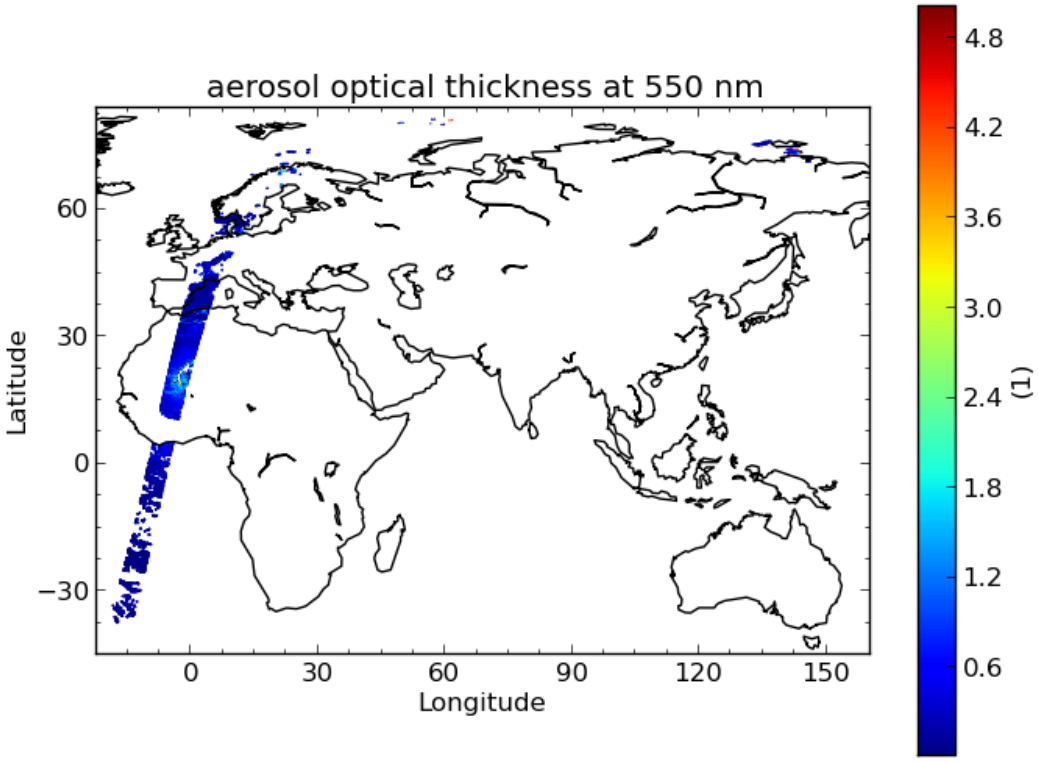
```
$ cis subset tas:tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc x=[-6,-.0001],y=[20,30],t=[2008

$ cis subset AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc x=[-6,0

$ cis col AOD550:cis-AOD550n_3.nc tas_3day.nc:colocator=bin[fill_value=-9999.0],kernel=mean -o AOD550

$ ncdump AOD550_on_tas_3day.nc |less
```

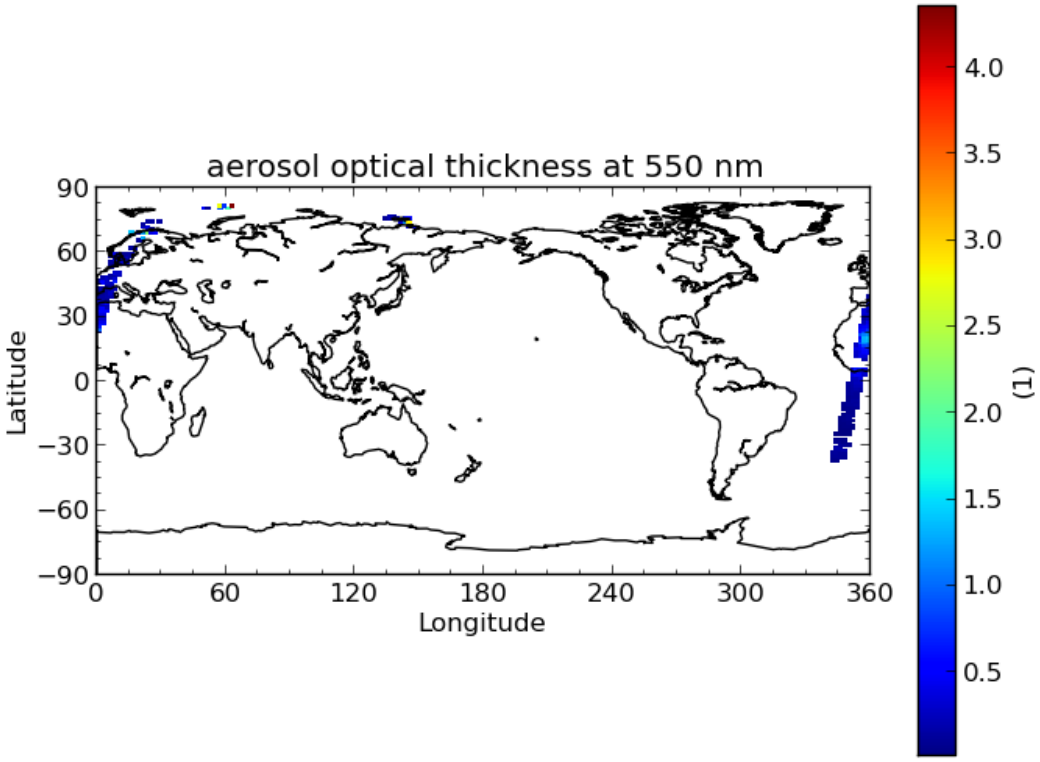
8.2.3 Aerosol CCI with One Time Step

This is as above but subsetting the grid to one time step so that the output can be plotted directly:

```
$ cis subset tas:tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc t=[2008-06-12T1,2008-06-12] -o

$ cis col AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc tas_2008-

$ cis plot AOD550:AOD550_on_tas_1day.nc
$ cis plot AOD550:20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc
$ cis plot tas:tas_2008-06-12.nc
```

These are the plots before and after co-location:



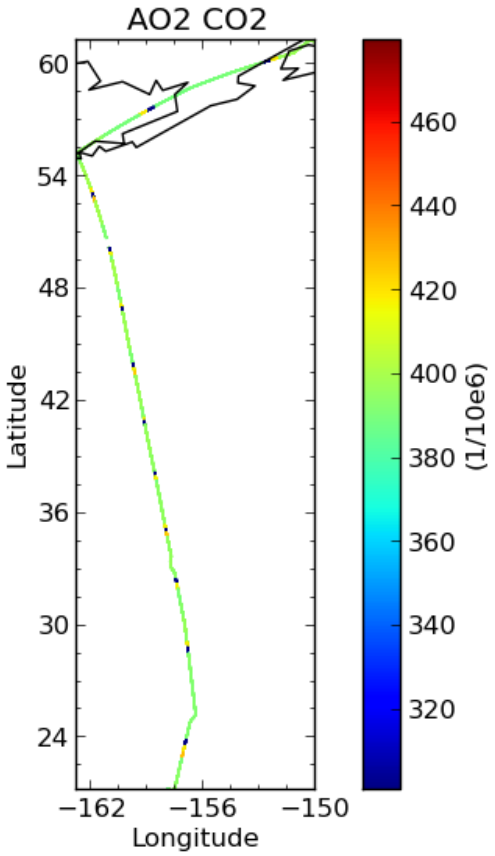


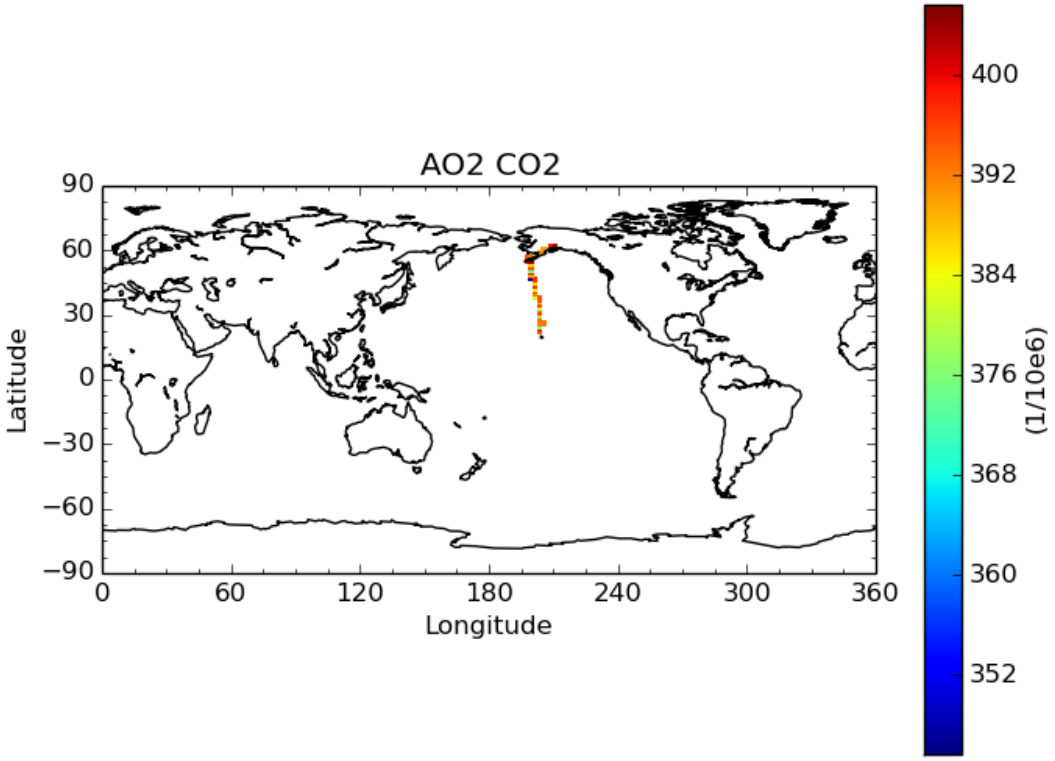
8.2.4 Example with NCAR RAF Data

This example uses the data in RF04.20090114.192600_035100.PNI.nc. However, this file does not have standard_name or units accepted as valid by Iris. These were modified using ncdump and ncgen, giving RF04_fixed_AO2CO2.nc:

```
$ cis subset tas:tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc t=[2009-01-14T1,2009-01-14] -o

$ cis col AO2CO2:RF04_fixed_AO2CO2.nc tas_2009-01-14.nc:colocator=bin[fill_value=-9999.0],kernel=mean

$ cis plot AO2CO2:RF04_on_tas.nc:product=NetCDF_Gridded
```

These are the plots before and after co-location:


AO2 CO2
60
54
48
42
36
30
24
Latitude
−162
−156
−150
Longitude
460
440
420
400
380
360
340
320
(1/10e6)

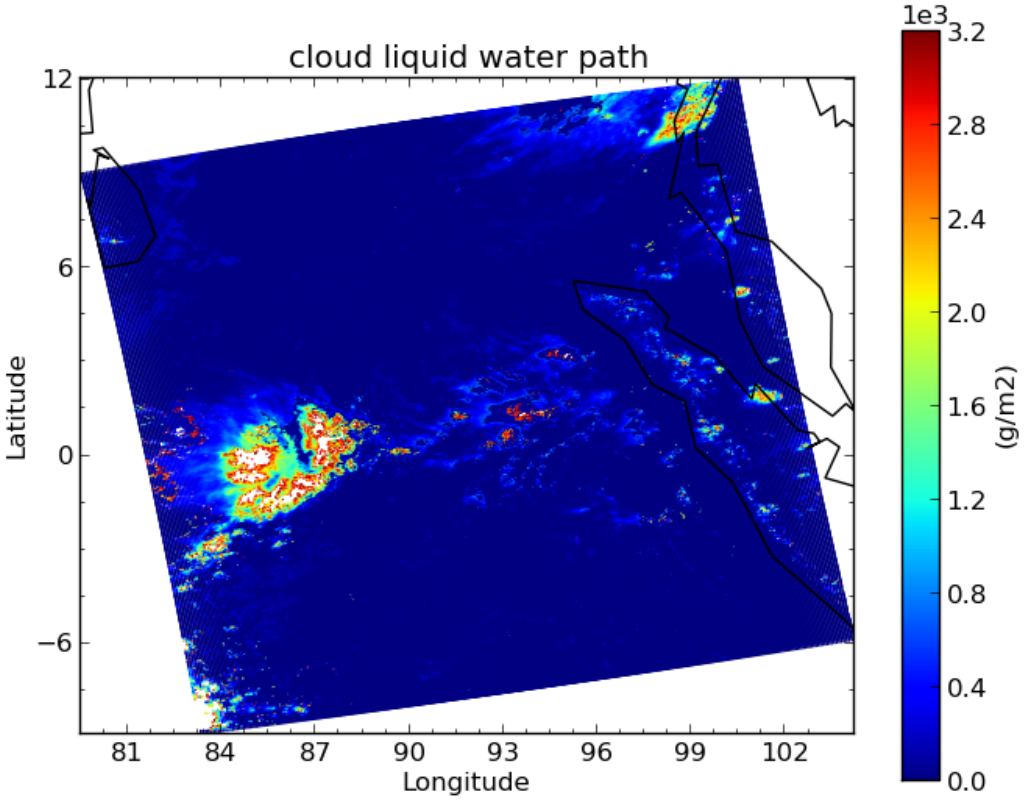


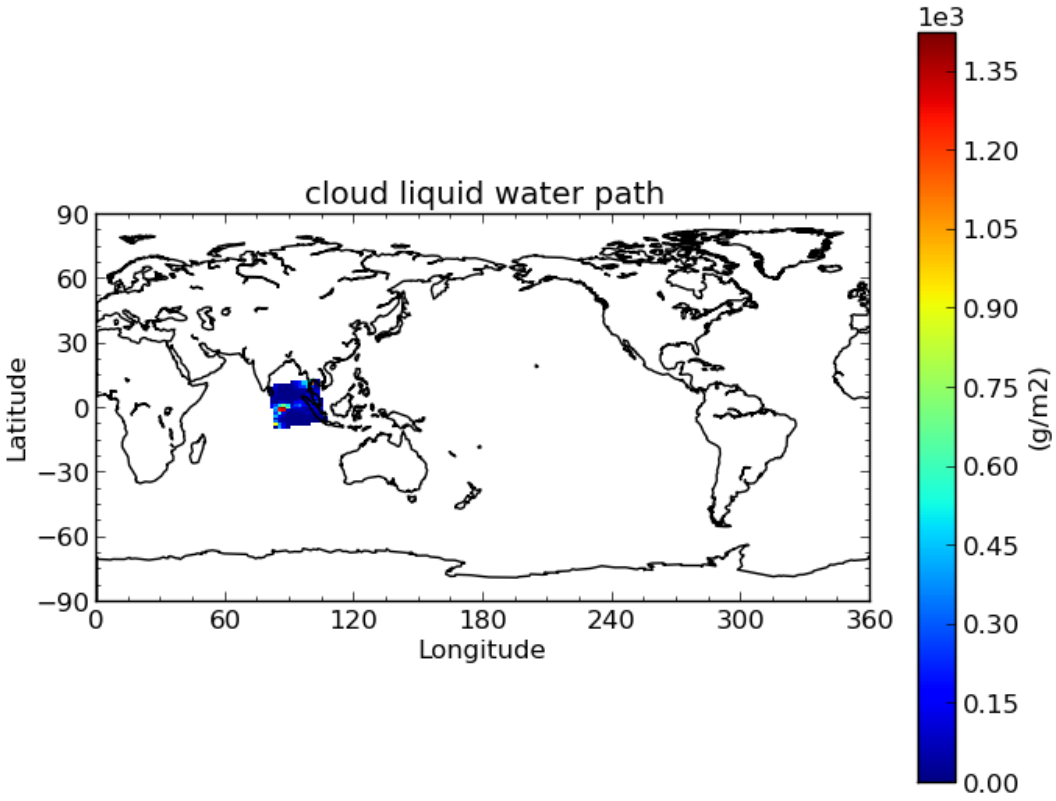
8.2.5 Cloud CCI with One Time Step

This is analogous to the Aerosol CCI example:

```
$ cis subset tas:tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc t=[2008-06-20T1,2008-06-20] -o

$ cis col cwp:20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc tas_2008-06-20.nc:colo

$ cis plot cwp:Cloud_CCI_on_tas.nc
$ cis plot cwp:20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc
```

These are the plots before and after co-location:


cloud liquid water path
Latitude
Longitude
12
6
0
−6
81
84
87
90
93
96
99
102
1e3
3.2
2.8
2.4
2.0
1.6
1.2
0.8
0.4
0.0
(g/m2)



8.2.6 File Locations

The files used above can be found at:

```
/group_workspaces/jasmin/cis/jasmin_cis_repo_test_files/
  20080612093821-ESACCI-L2P_AEROSOL-ALL-AATSR_ENVISAT-ORAC_32855-fv02.02.nc
  20080620072500-ESACCI-L2_CLOUD-CLD_PRODUCTS-MODIS-AQUA-fv1.0.nc
  RF04.20090114.192600_035100.PNI.nc
/group_workspaces/jasmin/cis/example_data/
  RF04_fixed_AO2CO2.nc
/group_workspaces/jasmin/cis/gridded-test-data/cmip5.output1.MOHC.HadGEM2-ES.rcp45.day.atmos.day.r1i1
  tas_day_HadGEM2-ES_rcp45_r1i1p1_20051201-20151130.nc
```

8.3 Examples of Gridded to Gridded Colocation

8.3.1 Example of Gridded Data onto a Finer Grid

First to show original data subset to a single time slice:

```
$ cis subset rsutcs:rsutcs_Amon_HadGEM2-A_sstClim_r1i1p1_185912-188911.nc t=[1859-12-12] -o sub1
```

Plot for subset data:

```
$ cis plot rsutcs:sub1.nc
```

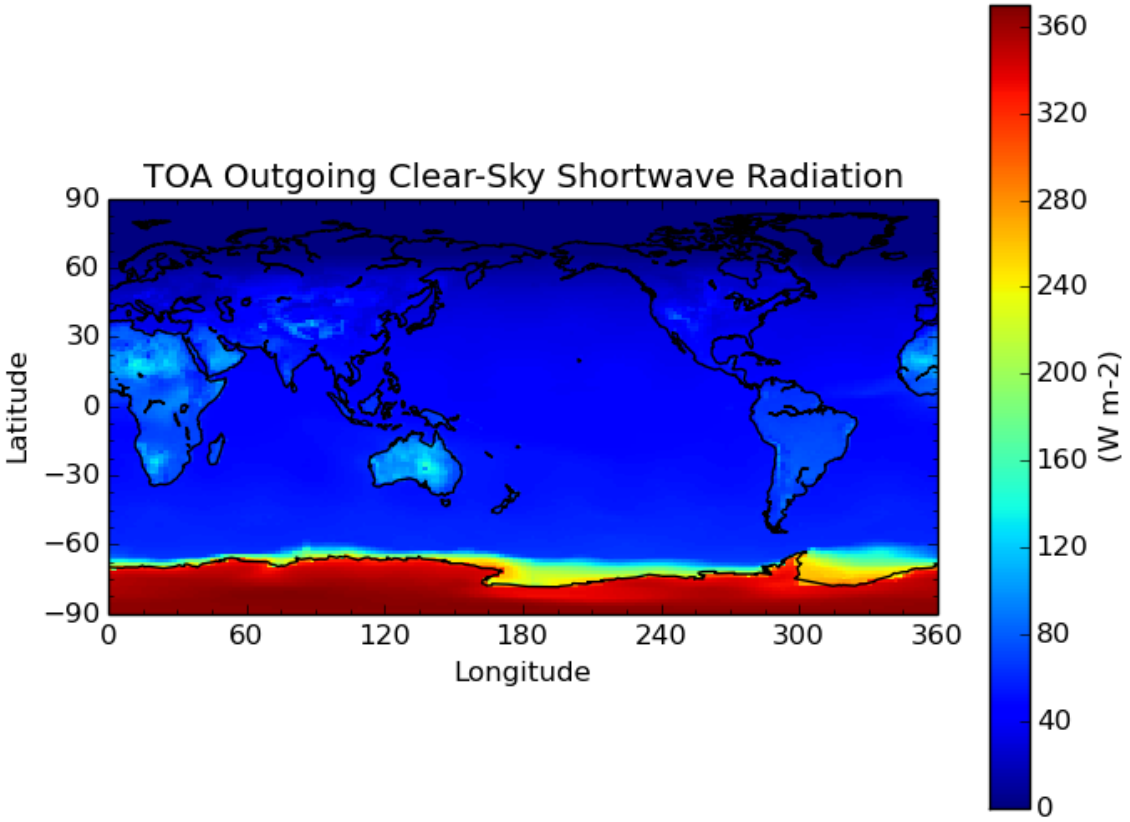
Colocate onto a finer grid, which was created using nearest neighbour:

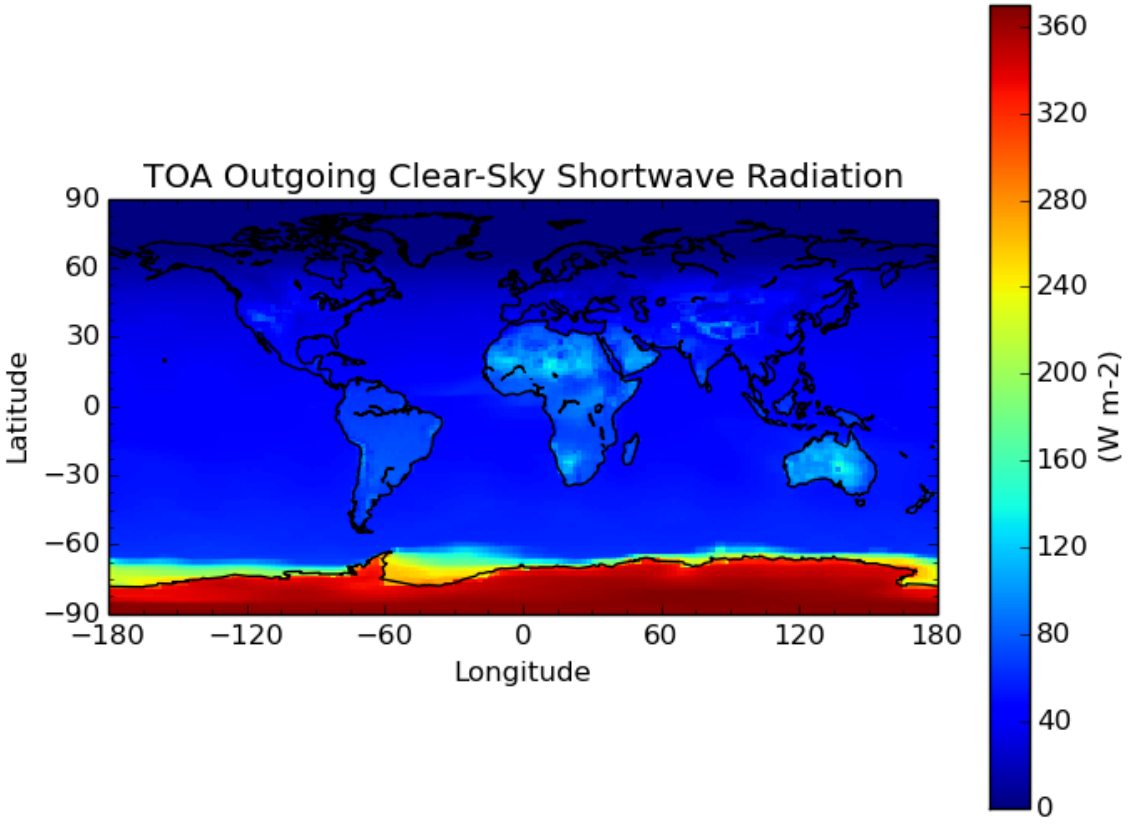
```
$ cis col rsutcs:rsutcs_Amon_HadGEM2-A_sstClim_r1i1p1_185912-188911.nc dummy_high_res_cube_-180_180.n
$ cis subset rsutcs:2.nc t=[1859-12-12] -o sub2
$ cis plot rsutcs:sub2.nc
```

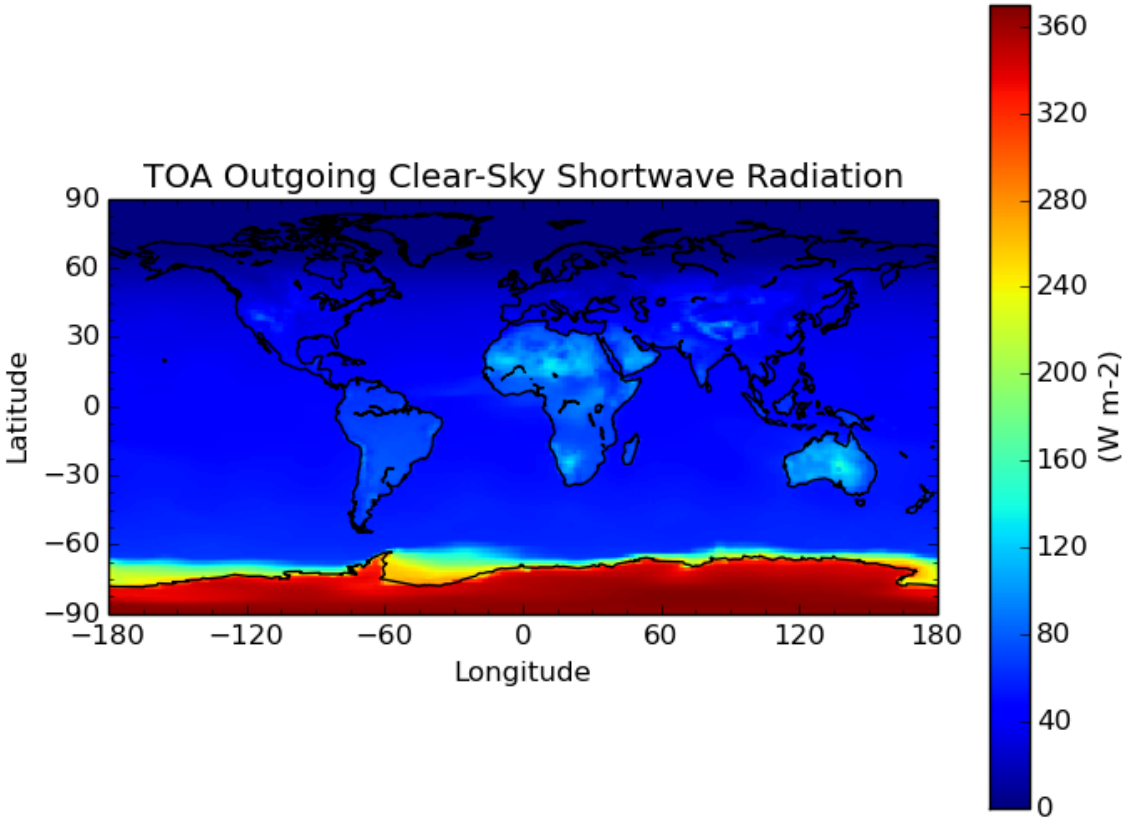
Colocate onto a finer grid, which was created using linear interpolation:

```
$ cis col rsutcs:rsutcs_Amon_HadGEM2-A_sstClim_r1i1p1_185912-188911.nc dummy_high_res_cube_-180_180.n
$ cis subset rsutcs:3.nc t=[1859-12-12] -o sub3
$ cis plot rsutcs:sub3.nc
```

Before, after nearest neighbour and after linear interpolation:




TOA Outgoing Clear-Sky Shortwave Radiation
Latitude
90
60
30
0
−30
−60
−90
−180
−120
−60
0
60
120
180
Longitude
360
320
280
240
200
160
120
80
40
0
(W m-2)



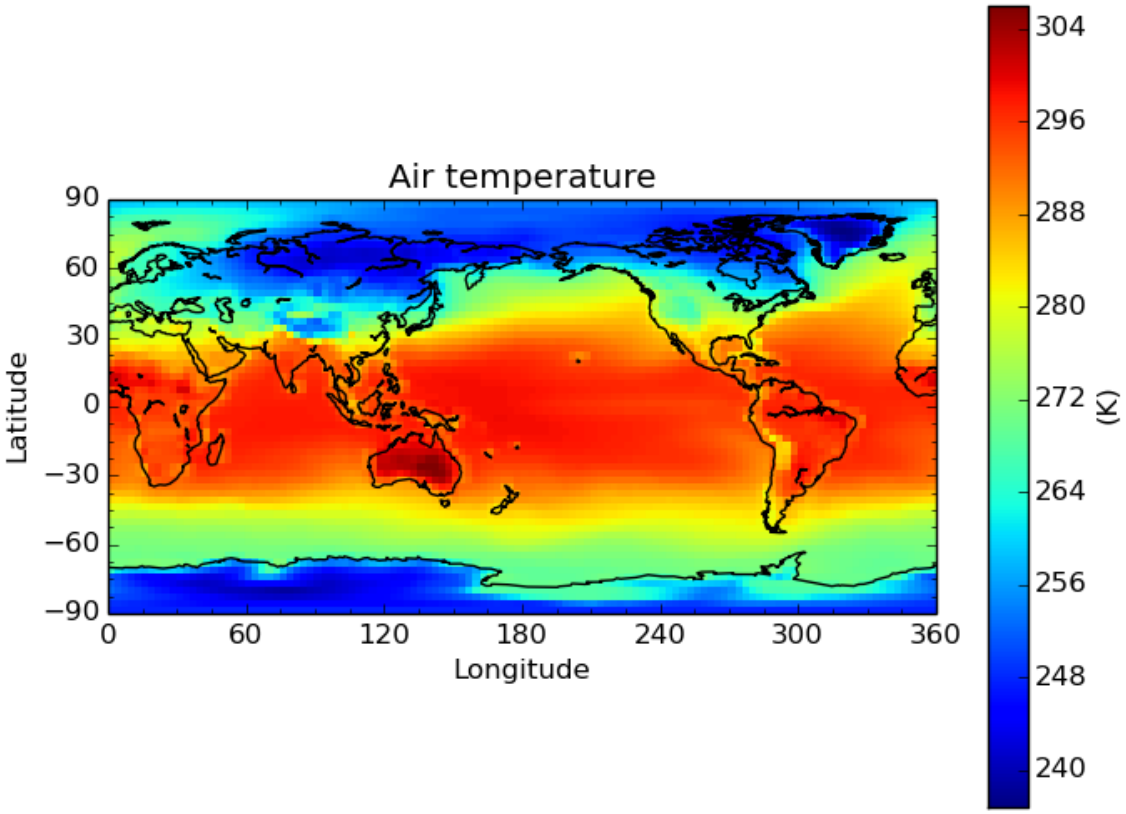
8.3.2 4D Gridded Data with latitude, longitude, air_pressure and time to a New Grid

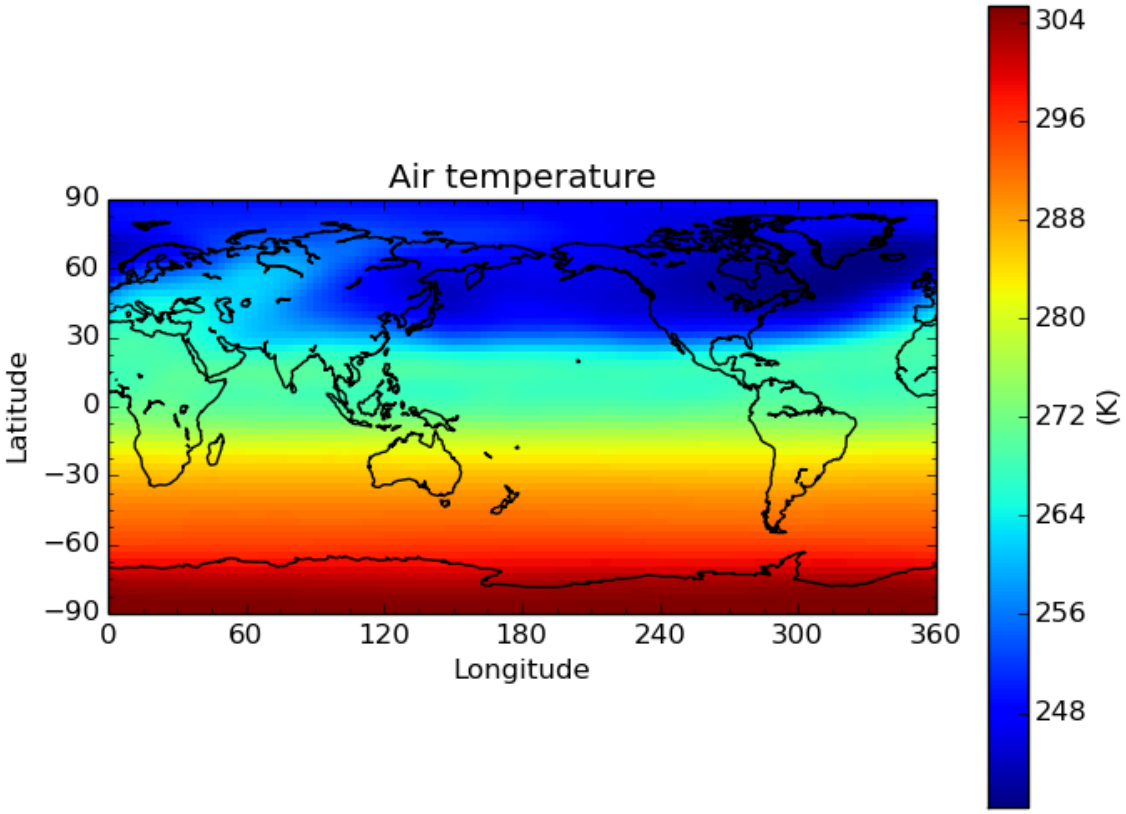
```
$ cis col temp:aerocom.INCA.A2.RAD-CTRL.monthly.temp.2006-fixed.nc dummy_low_res_cube_4D.nc:colocato
```

Note the file `aerocom.INCA.A2.RAD-CTRL.monthly.temp.2006-fixed.nc` has the standard name of `presnivs` changed to `air_pressure`, in order to be read correctly.

Slices at Different Pressures

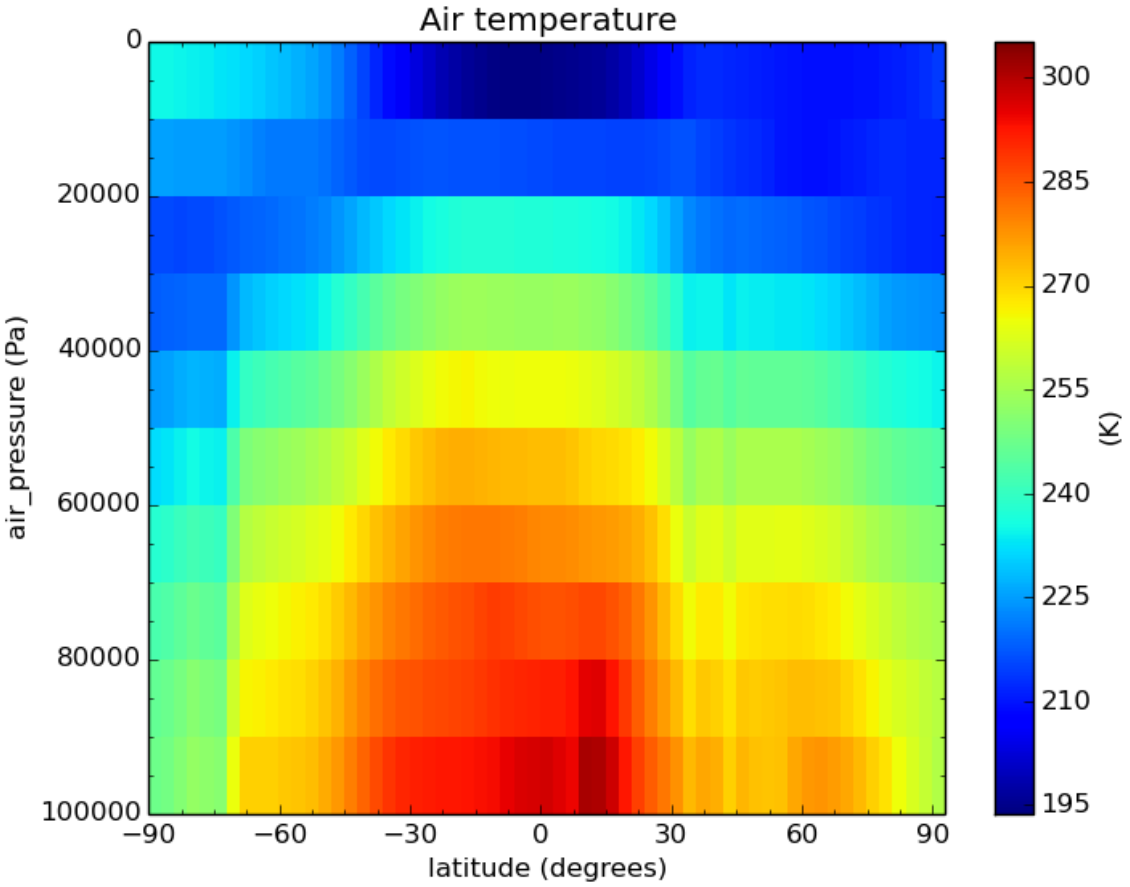
```
$ cis subset temp:4D-col.nc t=[2006-01],z=[100000] -o sub9
$ cis plot temp:sub9.nc
$ cis subset temp:4D-col.nc t=[2006-01],z=[0] -o sub10
$ cis plot temp:sub10.nc
```

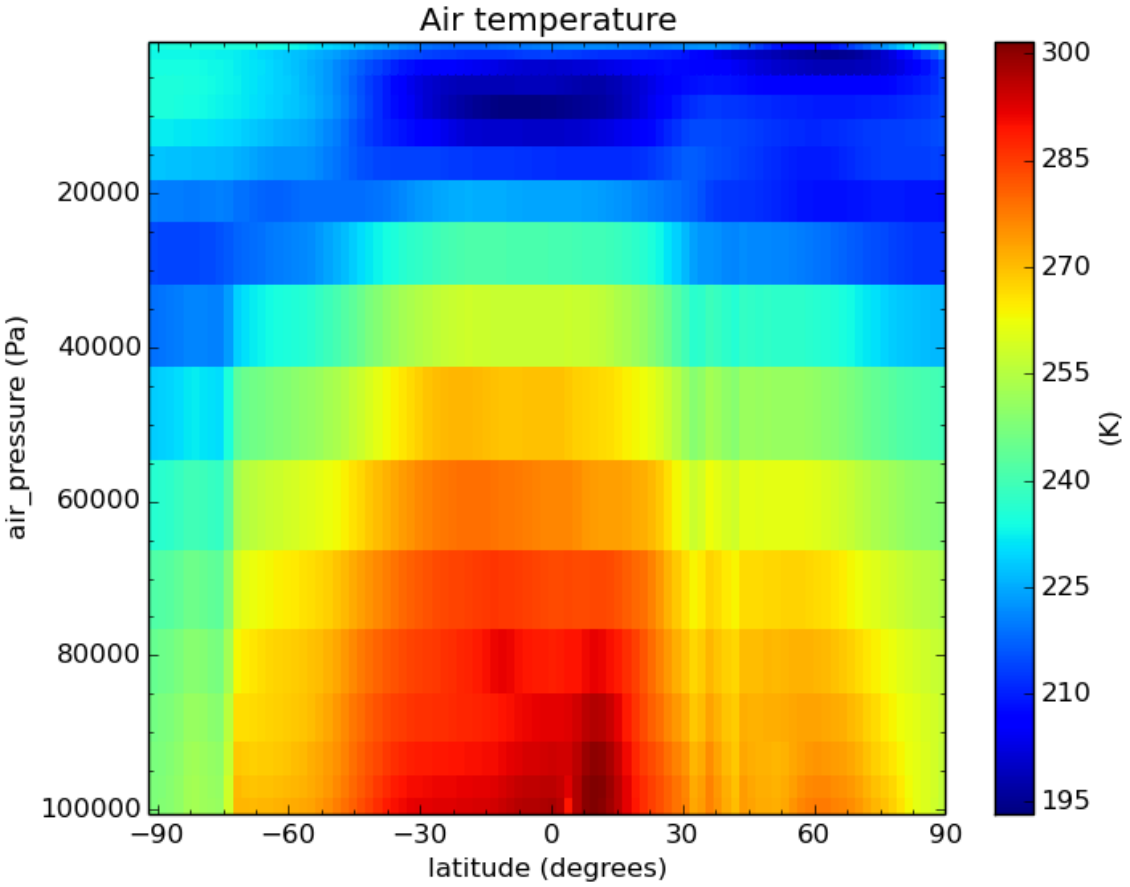

Air temperature
Latitude
Longitude
90
60
30
0
−30
−60
−90
0
60
120
180
240
300
360
304
296
288
280
272
264
256
248
240
(K)



Pressure against time

```
$ cis subset temp:4D-col.nc x=[0],t=[2006-01] -o sub11
$ cis plot temp:sub11.nc --xaxis latitude --yaxis air_pressure
$ cis subset temp:aerocom.INCA.A2.RAD-CTRL.monthly.temp.2006-fixed.nc x=[0],t=[2006-01] -o sub12
$ cis plot temp:sub12.nc --xaxis latitude --yaxis air_pressure
```


Air temperature
air_pressure (Pa)
0
20000
40000
60000
80000
100000
−90
−60
−30
0
30
60
90
latitude (degrees)
300
285
270
255
240
225
210
195
(K)



8.3.3 File Locations

The files used above can be found at:

```
/group_workspaces/jasmin/cis/sprint_reviews/SR4-IB/gridded_col2
```

CHAPTER 9

Plotting

Plotting is straightforward:

```
$ cis plot variable:filenames
```

This will attempt to locate the variable `variable` in all of the specified `filenames`, work out its metadata, such as units, labels, etc. and the appropriate chart type to plot, so that a line graph is used for two dimensional data, a scatter plot is used for three dimensional ungridded data and a heatmap for three dimensional gridded data. Other types of chart can be specified using the `--type` option. Available types are:

`line` a simple line plot, for three dimensional data the third variable is represented by the line colour

`scatter` a scatter plot, for three dimensional data the third variable is represented by the maker

`heatmap` a heatmap especially suitable for gridded data

> **Warning:** Basemap versions <= 1.0.7 have known issues when plotting heatmaps, particularly when using `--xmin` or `--xmax` options. Use a newer version if available, otherwise check your output for validity, especially around the meridians."

`contour` a standard contour plot, see *contour options*

`contourf` a filled contour plot, see *contour options*

`histogram3d`

`histogram2d`

`comparativescatter` allows two variables to be plotted against each other, specified as `cis plot variable1:filename1 variable2:filename2 --type comparativescatter`

`overlay` a collection of plots overlaid on one another, see *overlay plots*

`scatteroverlay` a heatmap overlayed with a scatter plot, see *scatter-overlay plots*

Note that `filenames` is a non-optional argument used to specify the files to read the variable from. These can be specified as a comma seperated list of the following possibilities:

1. A single filename - this should be the full path to the file
2. A single directory - all files in this directory will be read
3. A wildcarded filename - A filename with any wildcards compatible with the python module glob, so that *, ? and [] can all be used. For example `/path/to/my/test*file_[0-9]`.

Note that when using option 2, the filenames in the directory will be automatically sorted into alphabetical order. When using option 3, the filenames matching the wildcard will also be sorted into alphabetical order. The order of the comma separated list will however remain as the user specified, e.g.:

```
$ cis plot $var:filename1,filename2,wildc*rd,/my/dir/,filename3
```

would read `filename1`, then `filename2`, then all the files that match `wildc*rd` (in alphabetical order), then all the files in the directory `/my/dir/` (in alphabetical order) and then finally `filename3`.

9.1 Plot Options

There are a number of optional arguments, which should be entered as a comma separated list after the mandatory arguments, for example `variable:filename:product=Cis,edgecolor=black`. The options are:

color colour of markers, e.g. for scatter plot points or contour lines, see *Available Colours and Markers*

cmap colour map to use, e.g. for contour lines or heatmap, see *Available Colours and Markers*

cmin the minimum value for the colourmap

cmax the maximum value for the colourmap

edgecolor colour of scatter marker edges (can be used to differentiate scatter markers with a colourmap from the background plot)

itemstyle shape of scatter marker, see *Available Colours and Markers*

label name of datagroup for the legend

product the data product to use for the plot

Additional datagroup options for contour plots only:

contnlevels the number of levels for the contour plot

contlevels a list of levels for the contour plot, e.g. `contlevels=[0,1,3,10]`

contlabel options are `true` or `false`, if true then contour labels are shown

contwidth width of the contour lines

contfontsize size for labels on contour plot

Note that `label` refers to the label the plot will have on the legend, for example if a multi-series line graph or scatter plot is plotted. To set the labels of the axes, use `--xlabel` and `--ylabel`. `--cbarlabel` can be used to set the label on the colour bar.

The axes can be specified with `--xaxis` and `--yaxis`. Gridded data supports any coordinate axes available in the file, while ungridded data supports the following coordinate options (if available in the data):

- `latitude`
- `longitude`
- `time`
- `altitude`
- `air_pressure`
- `variable` - the variable being plotted

If the product is not specified, the program will attempt to figure out which product should be used based on the filename. See data-products to see a list of available products and their file signatures, or run `cis plot -h`.

9.2 Saving to a File

By default a plot will be displayed on screen. To save it to an image file instead, use the `--output` option. Available output types are png, pdf, ps, eps and svg, which can be selected using the appropriate filename extension, for example `--output plot.svg`.

9.3 Plot Formatting

There are a number of plot formatting options available:

`--xlabel` The label for the x axis

`--ylabel` The label for the y axis

`--cbarlabel` The label for the colorbar

`--xtickangle` The angle for the ticks on the x axis

`--ytickangle` The angle for the ticks on the y axis

`--title` The title of the plot

`--itemwidth` The width of an item. Unit are points in the case of a line, and points squared in the case of a scatter point

`--fontsize` The size of the font in points

`--cmap` The colour map to be used when plotting a 3D plot, see *Available Colours and Markers*

`--height` The height of the plot, in inches

`--width` The width of the plot, in inches

`--xbinwidth` The width of the histogram bins on the x axis

`--ybinwidth` The width of the histogram bins on the y axis

`--cbarorient` The orientation of the colour bar, either horizontal or vertical

`--nocolourbar` Hides the colour bar on a 3D plot

`--grid` Shows grid lines

`--plotwidth` width of the plot in inches

`--plotheight` height of the plot in inches

`--cbarscale` this can be used to change the size of the colourbar when plotting and defaults to 0.55 for vertical colorbars, 1.0 for horizontal.

`--coastlinescolour` The colour of the coastlines on a map, see *Available Colours and Markers*

`--nasabluemarble` Use the NASA Blue Marble for the background, instead of coastlines, when doing lat-lon plots

9.4 Setting Plot Ranges

The arguments `--xmin`, `--xmax`, `--xstep`, `--ymin`, `--ymax`, `--ystep`, `--vmin`, `--vmax`, `--vstep` can be used to specify the range of values to plot, where x and y correspond to the axes and v corresponds to the colours.

When the arguments refer to dates or times, they should be in the format `YYYY-MM-DDThh:mm:ss`, where the time is optional. A colon or a space is also a valid date and time separator (if using a space quotes are necessary).

The `step` arguments are used to specify the tick spacing on the axes and `vstep` is used to specify the tick spacing on the colorbar.

When the `step` arguments refer to an amount of time, they should be in the ISO 8061 format `PnYnMnDTnHnMnS`, where any particular time group is optional, case does not matter, and `T` can be substituted for either a colon or a space (if using a space quotes are necessary).

For example, to specify a tick spacing of one month and six seconds on the x axis, the following argument should be given: `--xstep 1m6S`

Note: If a value is negative, then an equals sign must be used, e.g. `--xmin=-5`.

To plot using a log scale:

`--logx` The x axis will be plotted using a log scale of base 10

`--logy` The y axis will be plotted using a log scale of base 10

`--logv` The values (colours) will be plotted using a log scale of base 10

9.5 Overlaying Multiple Plots

Using `--type overlay` allows multiple files to be specified on the command line to be plotted, each with its own type, which is specified as e.g. `type=heatmap`, along with the other datagroup options. Currently supported plot types are `heatmap`, `contour`, `contourf` and `scatter`. An additional datagroup option available is `transparency`, which allows the transparency for a layer to be set. `transparency` take a value between 0 and 1, where 0 is completely opaque and 1 fully transparent.

For example, to plot a heatmap and a contour plot the following options can be used:

```
cis plot var1:file1:type=heatmap var2:file2:type=contour,color=white --type overlay --plotwidth 20 --
```

Note that the first file specified is treated in a special way, from this the default plot dimensions are deduced, and the colorbar displayed will be for this datagroup only.

9.6 Scatter Overlay Plots

Note: Note that scatteroverlay is to be depreciated, as the overlay option will allow a more general method for overlaying multiple datasets

Three types of plot overlay are currently available:

- Overlaying several line graphs
- Overlaying several scatter plots
- Overlaying a heatmap with several scatter graphs

To overlay several line graphs or scatter plots, simply use the plot command as before, but simply specify multiple files and variables, e.g.:

```
$ cis plot $var1:$filename1:edgecolor=black $var2:$filename2:edgecolor=red
```

To plot two variables from the same file, simply use the above command with ‘‘ $filename1 ‘‘ in place of ‘‘ $filename2 ‘‘.

To overlay a heatmap with several scatter graphs, use the following command:

```
$ cis plot $var1:$filename1:label=label1 $var2:$filename2:color=colour2,itemstyle=style2,label=label2
```

Where ‘‘ $filename1 ‘‘ refers to the file containing the heatmap data and the other two filenames refer to the files containing the scatter data.

If the scatter data is 3 dimensional, then the colour argument can be omitted and the data will be plotted using the same colour map as the heatmap. This can be overridden by explicitly including the colour argument.

9.7 Available Colours and Markers

CIS recognises any valid html colour, specified using its name e.g. *red* for options such as item colour (line/scatter colour) and the colour of the coast lines.

A list of available colour maps for 3D plots, such as heatmaps, scatter and contour plots, can be found here: colour maps.

For a list of available scatter point styles, see here: scatter point styles.

CHAPTER 10

Evaluation

The Community Intercomparison Suite allows you to perform general arithmetic operations between different variables using the 'eval' command. For example, you might want to interpolate a value between two variables.

Note: All variables used in a evaluation **must** be of the same shape in order to be compatible, i.e. the same number of points in each dimension, and of the same type (Ungridded or Gridded). This means that, for example, operations between different data products are unlikely to work correctly - performing a colocation or aggregation onto a common grid would be a good pre-processing step.

Warning: This CIS command performs a Python eval() on user input. This has the potential to be a security risk and before deploying CIS to any environment where your user input is untrusted (e.g. if you want to run CIS as a web service) **you must** satisfy yourself that any security risks have been mitigated. CIS implements the following security restrictions on the expression which is evaluated:

- The eval() operates in a restricted namespace that only has access to a select handful of builtins (see *expr* below) - so `__import__`, for example, is unavailable.
- The only module available in the namespace is numpy.
- Any expression containing two consecutive underscores (__) is assumed to be harmful and will not be evaluated.

The evaluate syntax looks like this:

```
$ cis eval <datagroup>... <expr> <units> [-o [<output_var>:]<outputfile>] [--attributes <attributes>]
```

where square brackets denote optional commands and:

<datagroup> is a modified *CIS datagroup* of the format `<variable>[=<alias>]...:<filename>[:product=<produc`
One or more datagroups should be given.

- `<variable>` is a mandatory variable or list of variables to use.
- `<alias>` is an optional alternative variable name to use in place of the name given in the file. As you will see in the *expression* section, the variable names given will need to be valid python variable names, which means:

  1. They may use only the characters [A-Z], [a-z] and numbers [0-9] provided they do not start with a number
  2. The only special character which may be used is the underscore (_) - but don't use two consecutively (see *security note*)
  3. Don't use any of the reserved python keywords such as `class` or `and` as variable names (they're OK if they're only part of a name though).

4. Avoid using names of python builtins like `max` or `abs` (again, it's OK if they're only part of a name).

So if the variable name in your file violates these rules (e.g. *'550-870Angstrom'*) use an alias:

```
550-870Angstrom=a550to870
```

- `<filename>` is a mandatory file or list of files to read from.
- `<productname>` is an optional CIS data product to use (see *Data Products*):

See *Datagroups* for a more detailed explanation of datagroups.

<expr> is the arithmetic expression to evaluate; for example: `variable1+variable2`. Use the following basic rules to get started:

1. Use the variable names (or aliases) as given in the datagroups (they're case-sensitive) - don't enclose them in quotes.

2. If your expression contains whitespace, you'll need to enclose the whole expression in single or double quotes.

3. Construct your expression using plus `+`, minus `-`, times `*`, divide `/` , power `**` (note that you **can't** use `^` for exponents, like you typically can in spreadsheets and some other computer languages). Parentheses `()` can be used to group elements so that your expression is evaluated in the order you intend.

If you need more functionality, you're encountering errors or not getting the answer you expect then you should consider the following.

1. This expression will be evaluated in Python using the eval() method (see *security note*), so the expression must be a valid Python expression.

2. The only Python methods available to you are a trimmed down list of the python builtins: *'abs', 'all', 'any', 'bool', 'cmp', 'divmod', 'enumerate', 'filter', 'int', 'len', 'map', 'max', 'min', 'pow', 'range', 'reduce', 'reversed', 'round', 'sorted', 'sum', 'xrange', 'zip'*.

3. The numpy module is available, so you can use any of its methods e.g. `numpy.mean(variable1)`.

4. For security reasons, double underscores (___) must not appear anywhere in the expression.

5. The expression must produce an output array of the same shape as the input variables.

6. The expression is evaluated at the array level, not at the element level - so the variables in an expression represent numpy arrays, not individual numeric values. This means that `numpy.mean([var1,var2])` will give you a combined average *over the whole of both arrays* (i.e. a single number, not an array), which would be invalid (consider the previous rule). However, you could add the mean (over the whole array) of one variable to every point on a second variable by doing `var1 + numpy.mean(var2)`.

Note: CIS eval command will flatten ungridded data so that structure present in the input files will be ignored. This allows you to compare ungridded data with different shapes, e.g. (3,5) and (15,)

<units> is a mandatory argument describing the units of the resulting expression. This should be a CF compliant units string, e.g. `"kg m^-3"`. Where this contains spaces, the whole string should be enclosed in quotes.

<outputfile> is an optional argument specifying the file to output to. This will be automatically given a `.nc` extension if not present and if the output is ungridded, will be prepended with `cis-` to identify it as a CIS output file. This must not be the same file path as any of the input files. If not provided, the default output filename is *out.nc*

- `<output_var>` is an optional prefix to the output file argument to specify the name of the output variable within the output file, e.g. `-o my_new_var:output_filename.nc`. If not provided, the default output variable name is *calculated_variable*

<attributes> is an optional argument allowing users to provide additional metadata to be included in the evaluation output variable. This should be indicated by the attributes flag (`--attributes` or `-a`). The attributes should then follow in comma-separated, key=value pairs, for example `--attributes standard_name=convective_rainfall_amount,echam_version=6.1.00`. Whitespace is permitted in both the names and the values, but then must be enclosed in quotes: `-a "operating system = "AIX 6.1 Power6"`. Colons or equals signs may not be used in attribute names or values.

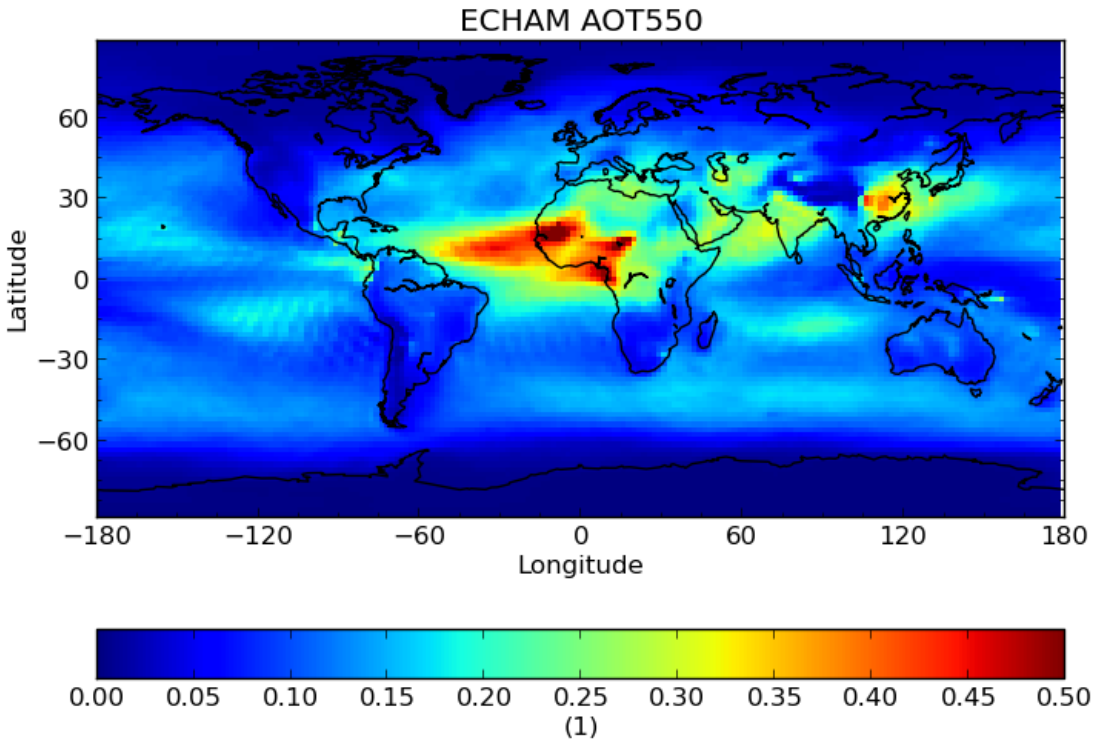
10.1 Evaluation Examples

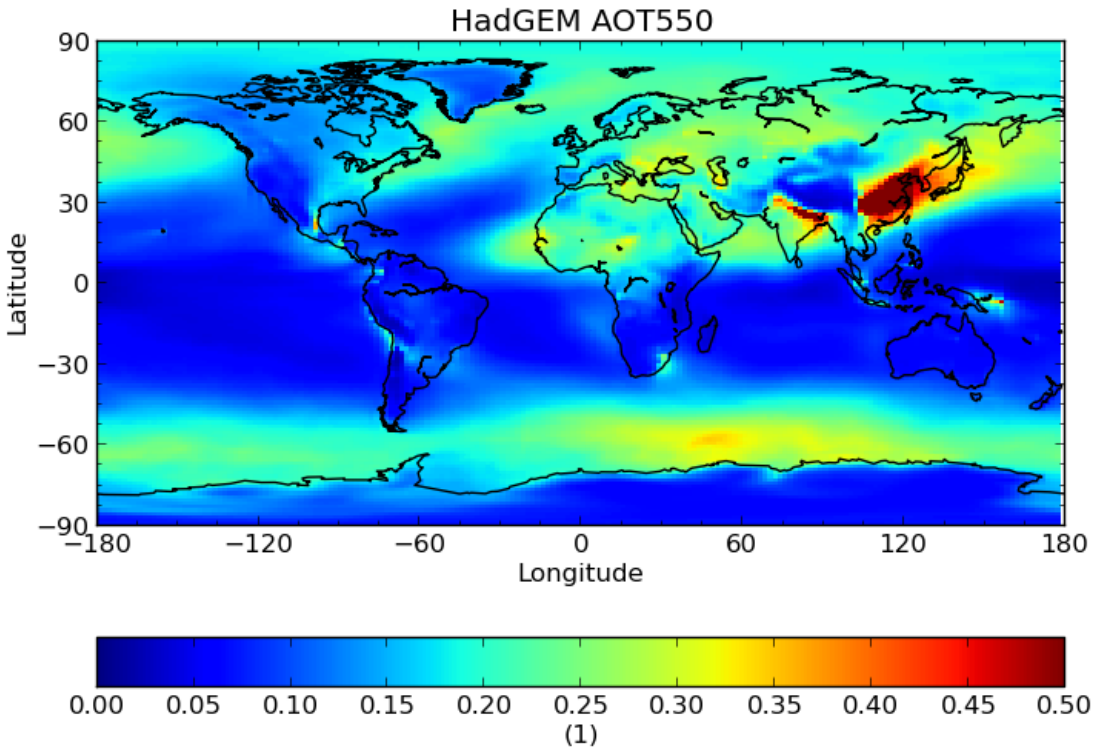
10.1.1 Comparison of annual Aerosol Optical Thickness from models

In this example we compare annual Aerosol Optical Thickness from ECHAM and HadGEM model data. The data used in this example can be found at `/group_workspaces/jasmin/cis/data`.

First we produce annual averages of our data by *aggregating*:

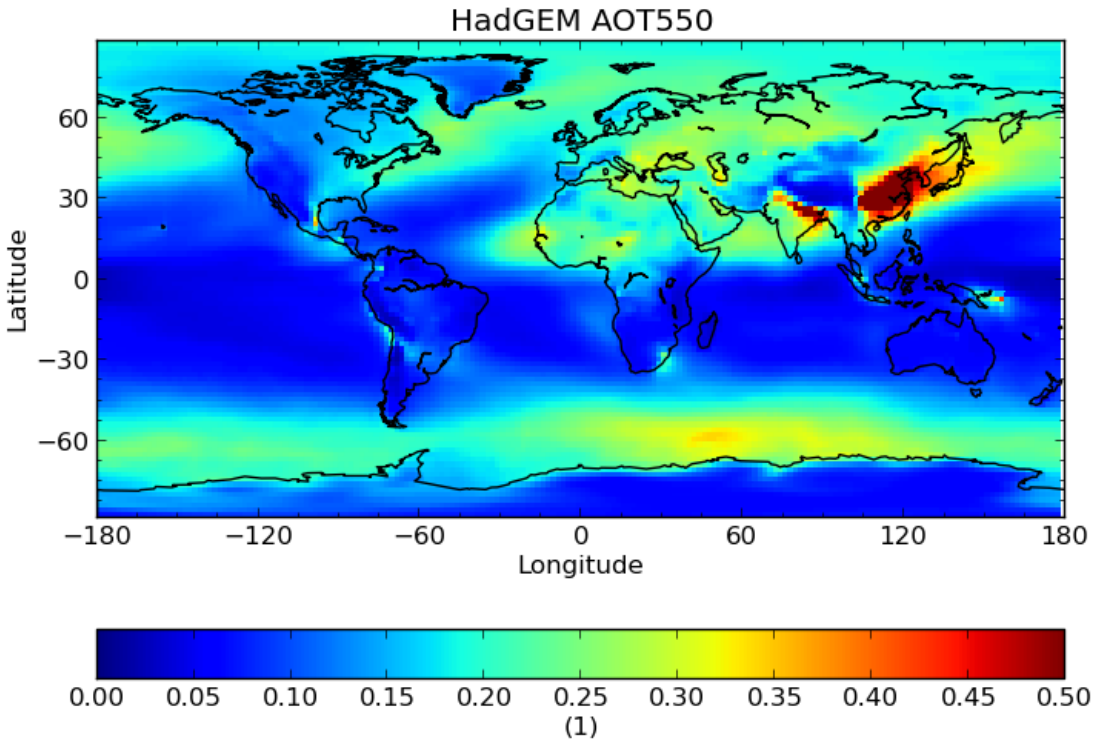
```
$ cis aggregate od550aer:ECHAM_fixed/2007_2D_3hr/od550aer.nc t -o echam-od550aer
$ cis aggregate od550aer:HadGEM_fixed/test_fix/od550aer.nc t -o hadgem-od550aer

$ cis plot od550aer:echam-od550aer.nc --xmin -180 --xmax 180 --cbarorient=horizontal --title="ECHAM A
$ cis plot od550aer:hadgem-od550aer.nc --xmin -180 --xmax 180 --cbarorient=horizontal --title="HadGEM
```


ECHAM AOT550
Latitude
60
30
0
−30
−60
−180
−120
−60
0
60
120
180
Longitude
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
(1)



We then linearly interpolate the HadGEM data onto the ECHAM grid:

```
$ cis col od550aer:hadgem-od550aer.nc echam-od550aer.nc:colocator=lin -o hadgem-od550aer-colocated

$ cis plot od550aer:hadgem-od550aer-colocated.nc --xmin -180 --xmax 180 --cbarorient=horizontal --tit
```

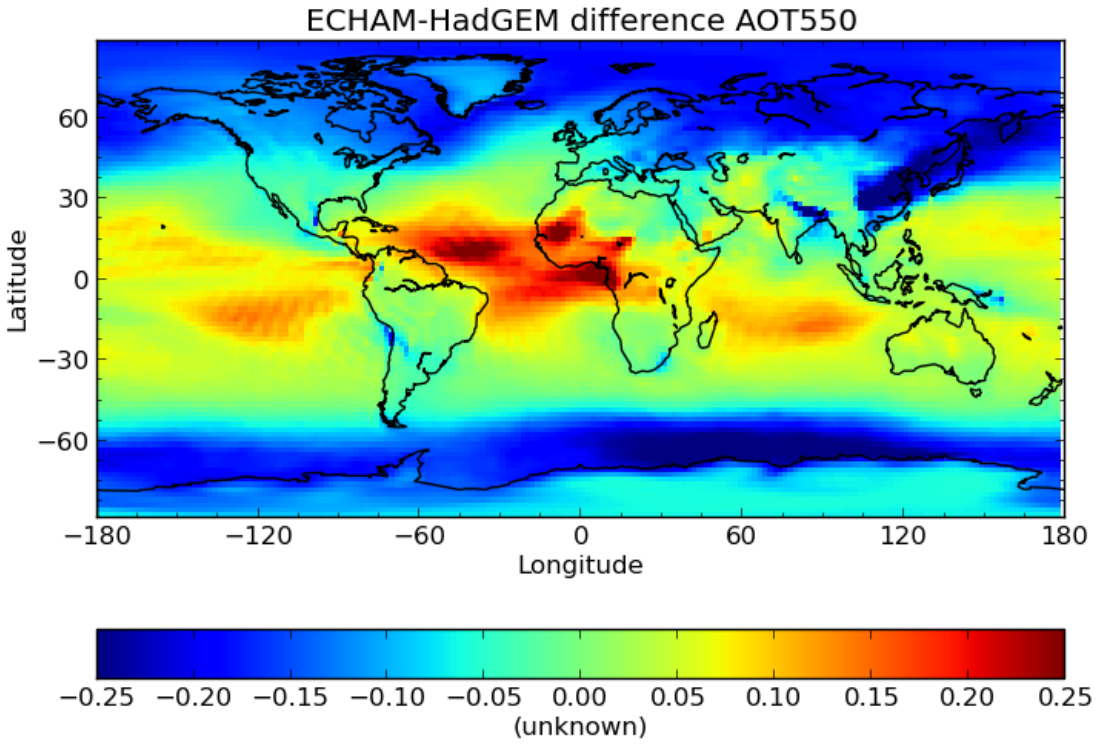


Next we subtract the two fields using:

```
$ cis eval od550aer=a:echam-od550aer.nc od550=b:hadgem-od550aer-collocated.nc "a-b" 1 -o modeldiffere
```

Finally we plot the evaluated output:

```
$ cis plot od550aer:modeldifference.nc --xmin -180 --xmax 180 --cbarorient=horizontal --title="ECHAM-
```



10.1.2 Calculation of Angstrom exponent for AERONET data

AERONET data allows us to calculate Angstrom Exponent (AE) and then compare it against the AE already in the file. They should strongly correlate although it is not expected they will be identical due to averaging etc during production of AERONET datafiles.

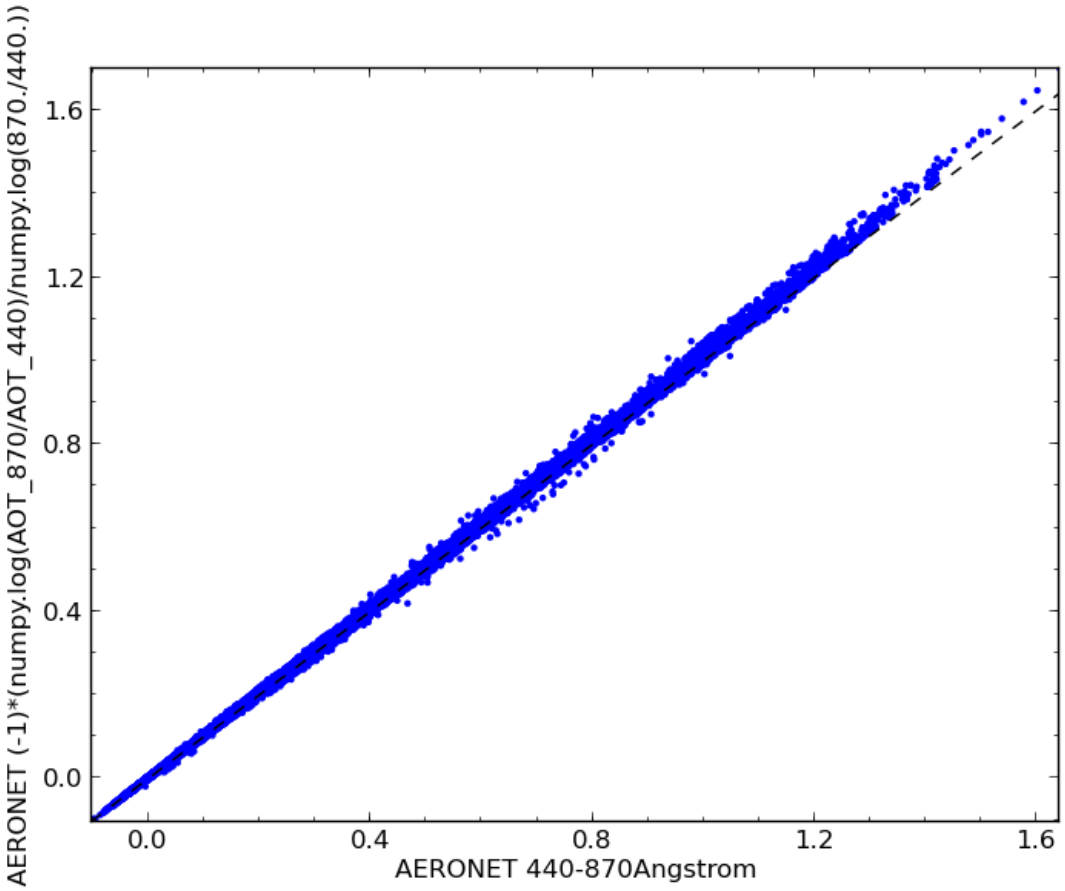
The file agoufou.lev20 refers to `/group_workspaces/jasmin/cis/data/aeronet/AOT/LEV20/ALL_POINTS/920801_`

The AE is calculated using an eval statement:

```
$ cis eval AOT_440,AOT_870:agoufou.lev20 "(-1)* (numpy.log(AOT_870/AOT_440)/numpy.log(870./440.))" 1
```

Plotting it shows the expected correlation:

```
$ cis plot 440-870Angstrom:agoufou.lev20 calculated_variable:cis-alfa.nc --type comparativescatter --
```



This correlation can be confirmed by using the CIS *stats* command:

```
$ cis stats 440-870Angstrom:agoufou.lev20 calculated_variable:cis-alfa.nc

==================================
RESULTS OF STATISTICAL COMPARISON:
==================================
Number of points: 63126
Mean value of dataset 1: 0.290989032142
Mean value of dataset 2: 0.295878214327
Standard deviation for dataset 1: 0.233995525021
Standard deviation for dataset 2: 0.235381075635
Mean of absolute difference: 0.00488918218519
Standard deviation of absolute difference: 0.00546343157047
Mean of relative difference: 0.0284040419499
Standard deviation of relative difference: 3.95137224542
Spearman's rank coefficient: 0.999750939223
Linear regression gradient: 1.00566622549
Linear regression intercept: 0.003240372714
Linear regression r-value: 0.999746457079
Linear regression standard error: 0.00530006646489
```

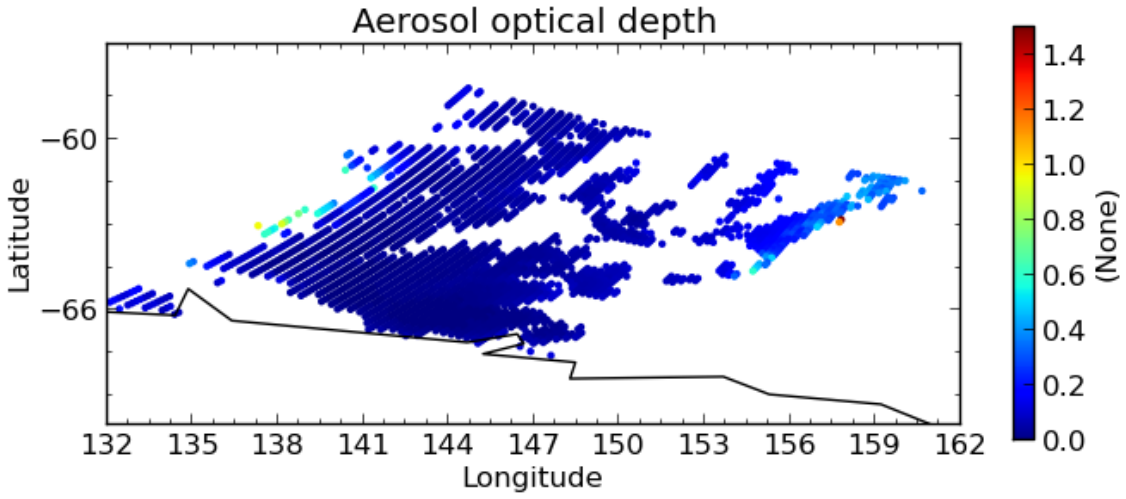
10.1.3 Using Evaluation for Conditional Aggregation

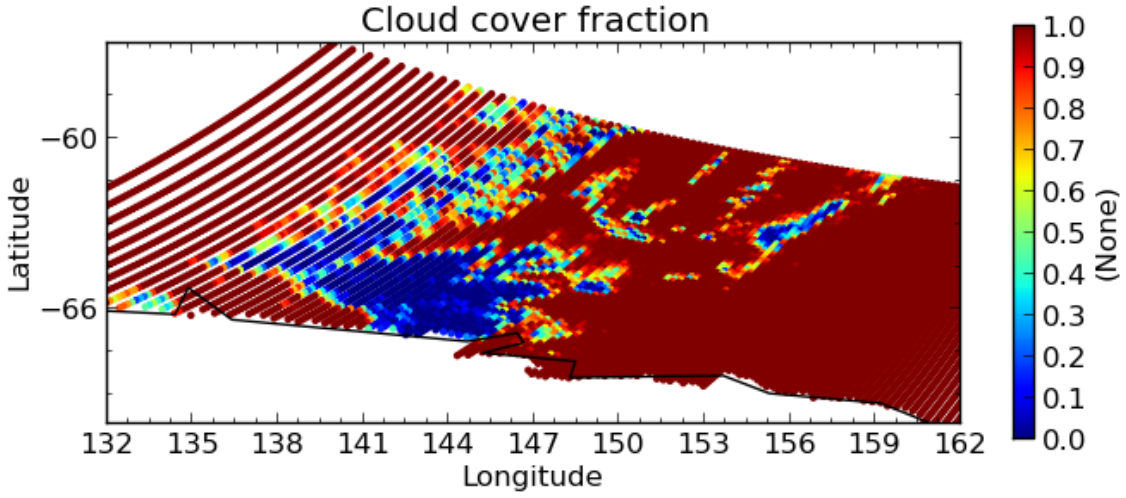
The *eval* command can be combined with other CIS commands to allow you to perform more complex tasks than would otherwise be possible.

For example, you might want to aggregate a satellite measurement of one variable only when the corresponding cloud cover fraction (stored in separate variable) is less than a certain value. The aggregate command doesn't allow this kind of conditional aggregation on its own, but you can use an evaluation to achieve this in two stages.

In this example we use the MODIS file `MOD04_L2.A2010001.2255.005.2010005215814.hdf` in directory `/group_workspaces/jasmin/cis/data/MODIS/MOD04_L2/`. The optical depth and cloud cover variables can be seen in the following two plots:

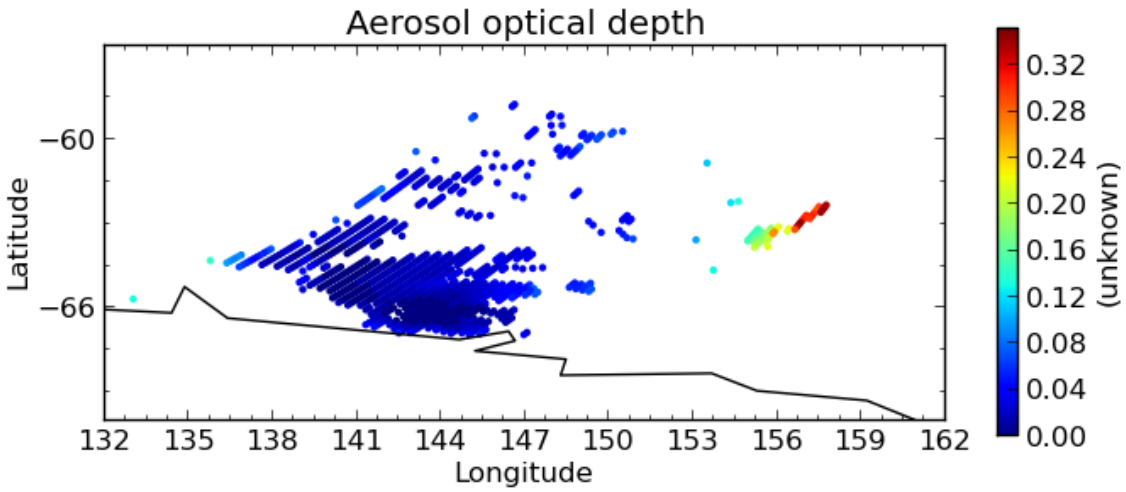
```
$ cis plot Optical_Depth_Land_And_Ocean:MOD04_L2.A2010001.2255.005.2010005215814.hdf --xmin 132 --xma
$ cis plot Cloud_Fraction_Ocean:MOD04_L2.A2010001.2255.005.2010005215814.hdf --xmin 132 --xmax 162 --
```





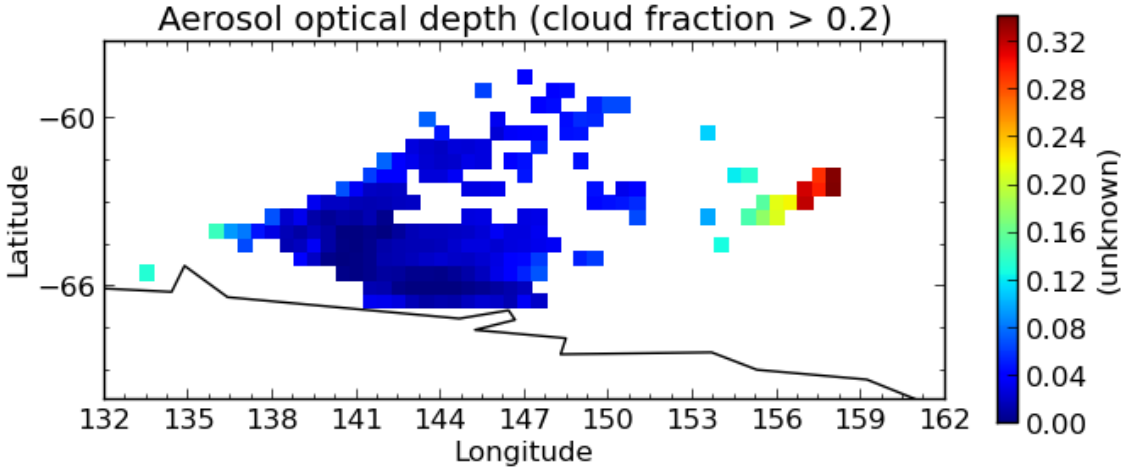
First we perform an evaluation using the numpy.masked_where method to produce an optical depth variable that is masked at all points where the cloud cover is more than 20%:

```
$ cis eval Cloud_Fraction_Ocean=cloud,Optical_Depth_Land_And_Ocean=od:MOD04_L2.A2010001.2255.005.2010
$ cis plot od:cis-masked_optical_depth.nc --xmin 132 --xmax 162 --ymin -70 --title Aerosol optical de
```



Then we perform an aggregation on this masked output file to give the end result - aerosol optical depth aggregated only using points where the cloud cover is less than 20%:

```
$ cis aggregate od:cis-masked_optical_depth.nc x=[132,162,0.5],y=[-70,-57,0.5] -o aggregated_masked_o
$ cis plot od:aggregated_masked_optical_depth.nc --xmin 132 --xmax 162 --ymin -70 --title "Aerosol op
```



CHAPTER 11

Statistics

The Community Intercomparison Suite allows you to perform statistical analysis on two variables using the 'stats' command. For example, you might wish to examine the correlation between a model data variable and actual measurements. The 'stats' command will calculate:

1. Number of data points used in the analysis.
2. The mean and standard deviation of each dataset (separately).
3. The mean and standard deviation of the absolute difference (var2 - var1).
4. The mean and standard deviation of the relative difference ((var2 - var1) / var1).
5. The Linear Pearson correlation coefficient.
6. The Spearman Rank correlation coefficient.
7. The coefficients of linear regression (i.e. var2 = a var1 + b), r-value, and standard error of the estimate.

These values will be displayed on screen and can optionally be save as NetCDF output.

Note: Both variables used in a statistical analysis **must** be of the same shape in order to be compatible, i.e. the same number of points in each dimension, and of the same type (ungridded or gridded). This means that, for example, operations between different data products are unlikely to work correctly - performing a colocation or aggregation onto a common grid would be a good pre-processing step.

Note: Only points which have non-missing values for both variables will be included in the analysis. The number of points this includes is part of the output of the stats command.

Warning: Unlike *aggregation*, `stats` does **not** currently use latitude weighting to account for the relative areas of different grid cells.

The statistics syntax looks like this:

```
$ cis stats <datagroup>... [-o <outputfile>]
```

where:

<datagroup> is a *CIS datagroup* specifying the variables and files to read and is of the format `<variable>...:<filename>[:product=<productname>]` where:

- `<variable>` is a mandatory variable or list of variables to use.
- `<filenames>` is a mandatory file or list of files to read from.
- `<productname>` is an optional CIS data product to use (see *Data Products*):

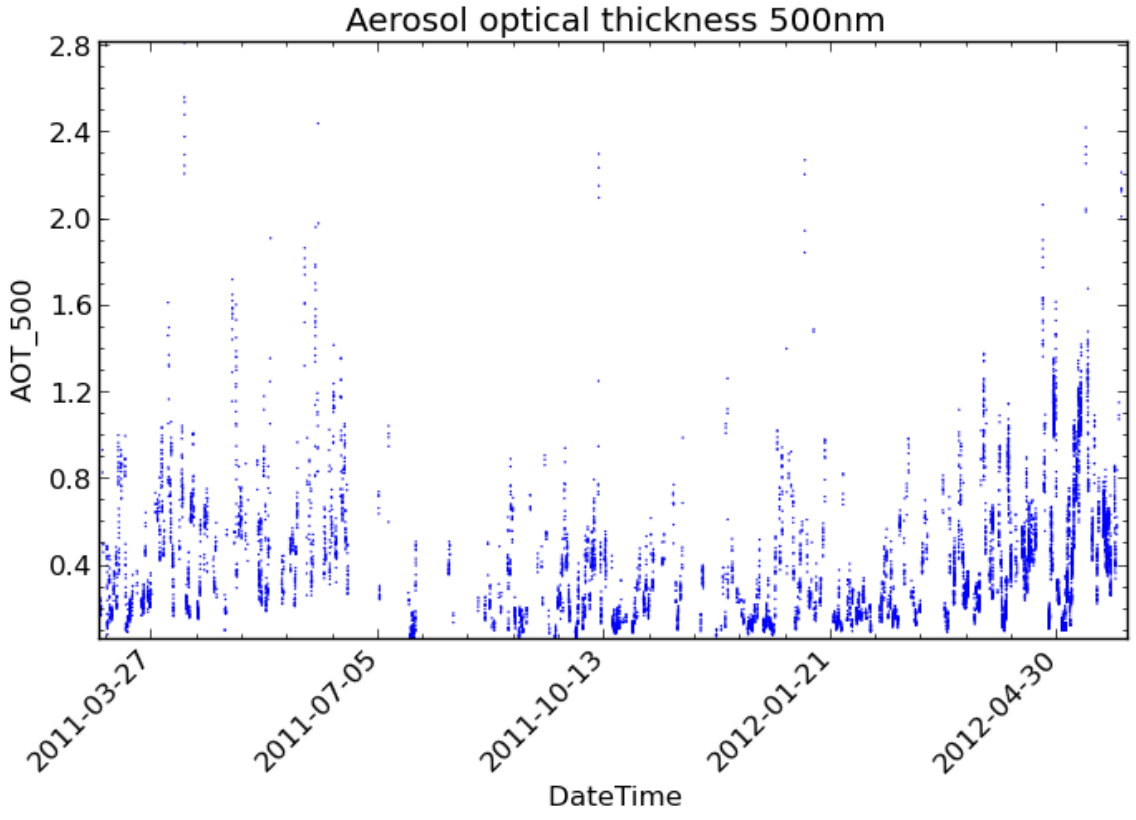
One or more datagroups should be given, but the total number of variables declared in all datagroups must be exactly two. See *Datagroups* for a more detailed explanation of datagroups.

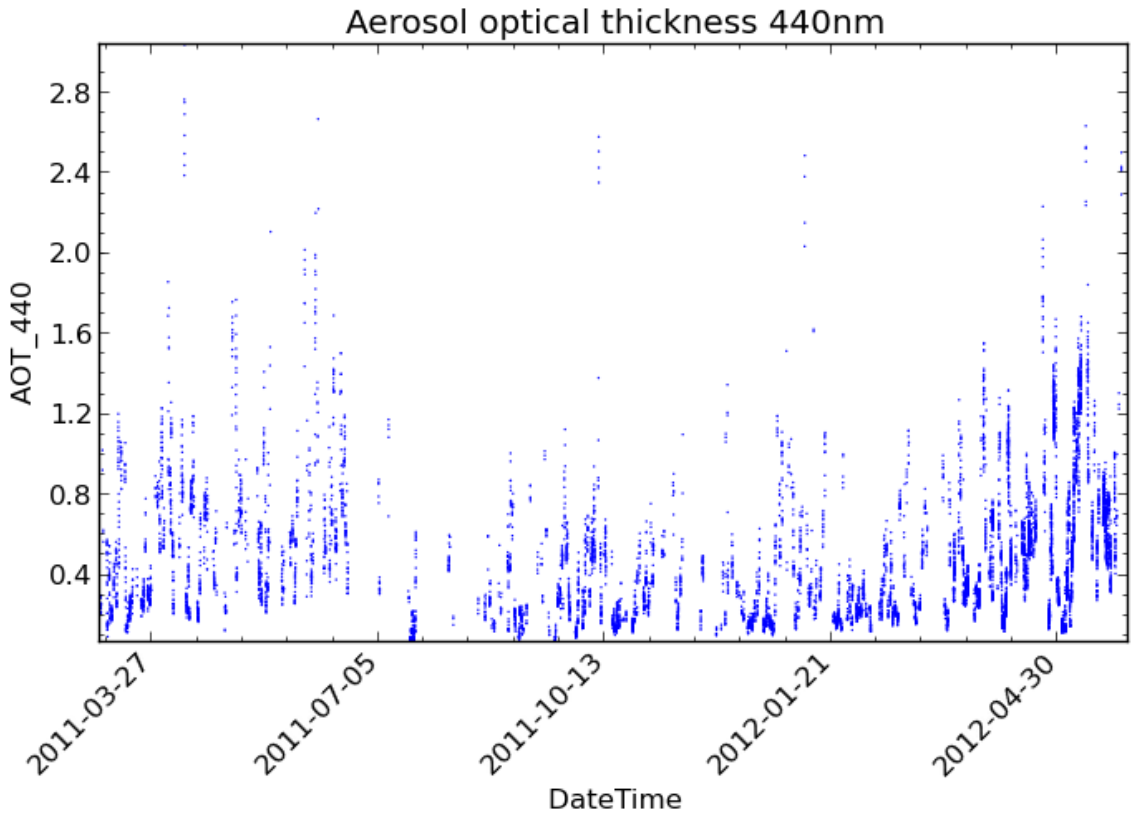
<outputfile> is an optional argument specifying a file to output to. This will be automatically given a `.nc` extension if not present. This must not be the same file path as any of the input files. If not provided, then the output will not be saved to a file and will only be displayed on screen.

11.1 Statistics Example

In this example, we perform a statistical comparison of Aeronet aerosol optical thickness at two wavelengths. The data we are using is shown in the following CIS plot commands and can be found at `/group_workspaces/jasmin/cis/data`:

```
$ cis plot AOT_500:aeronet/AOT/LEV20/ALL_POINTS/920801_121229_Yonsei_University.lev20 --title "Aeroso
$ cis plot AOT_440:aeronet/AOT/LEV20/ALL_POINTS/920801_121229_Yonsei_University.lev20 --title "Aeroso
```





We then perform a statistical comparison of these variables using:

```
$ cis stats AOT_500,AOT_440:aeronet/AOT/LEV20/ALL_POINTS/920801_121229_Yonsei_University.lev20
```

Which gives the following output:

```
==================================================================
RESULTS OF STATISTICAL COMPARISON:
------------------------------------------------------------------
Compared all points which have non-missing values in both variables
==================================================================
Number of points: 10727
Mean value of dataset 1: 0.427751965508
Mean value of dataset 2: 0.501316673814
Standard deviation for dataset 1: 0.307680514916
Standard deviation for dataset 2: 0.346274598431
Mean of absolute difference: 0.0735647083061
Standard deviation of absolute difference: 0.0455684788406
Mean of relative difference: 0.188097066086
Standard deviation of relative difference: 0.0528621773819
Spearman's rank coefficient: 0.998289763952
Linear regression gradient: 1.12233533743
Linear regression intercept: 0.0212355272705
Linear regression r-value: 0.997245296339
Linear regression standard error: 0.0256834603945
```

CHAPTER 12

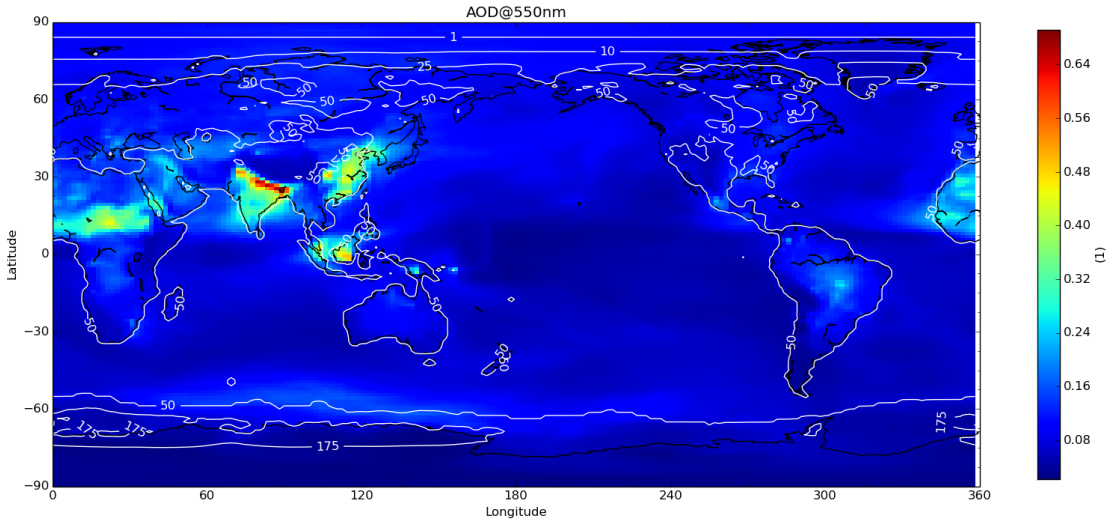
Overlay Plot Examples

First subset some gridded data that will be used for the examples:

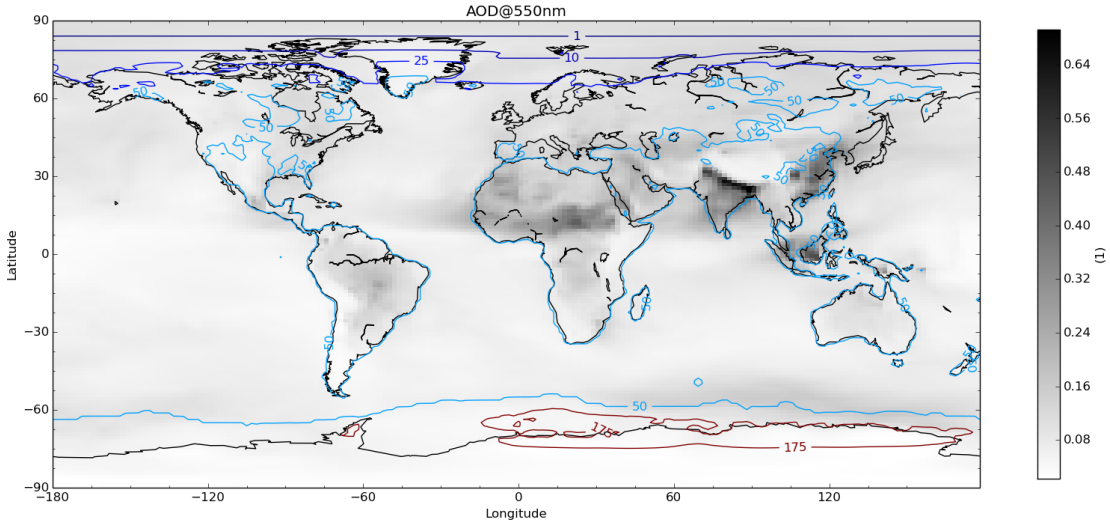
```
cis subset od550aer:aerocom.HadGEM3-A-GLOMAP.A2.CTRL.monthly.od550aer.2006.nc t=[2006-10-13] -o HadGE

cis subset rsutcs:aerocom.HadGEM3-A-GLOMAP.A2.CTRL.monthly.rsutcs.2006.nc t=[2006-10-13] -o HadGEM_rs
```

12.1 Contour over heatmap

```
cis plot od550aer:HadGEM_od550aer-subset.nc:type=heatmap rsutcs:HadGEM_rsutcs-subset.nc:type=contour,
```

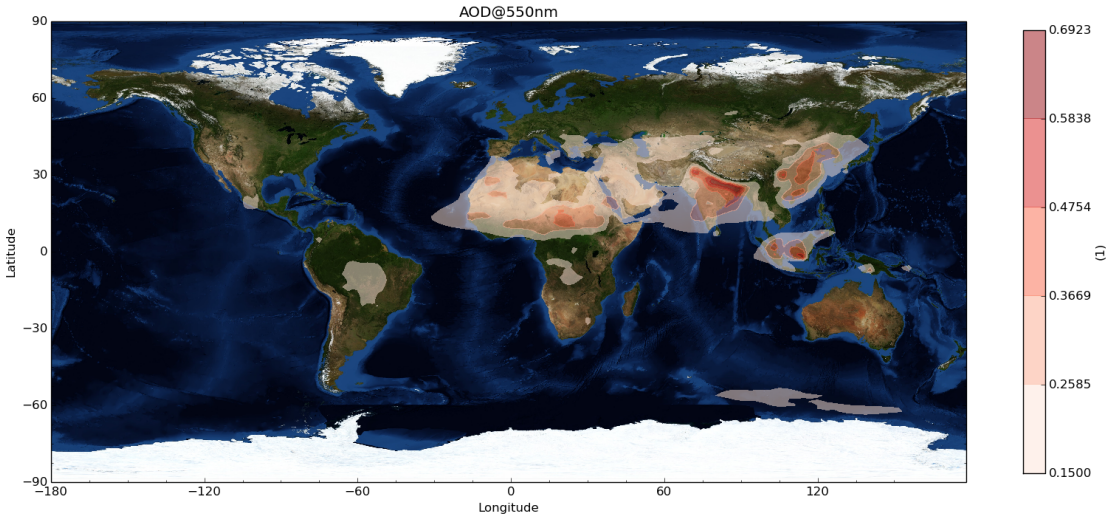


```
cis plot od550aer:HadGEM_od550aer-subset.nc:type=heatmap,cmap=binary rsutcs:HadGEM_rsutcs-subset.nc:t
```



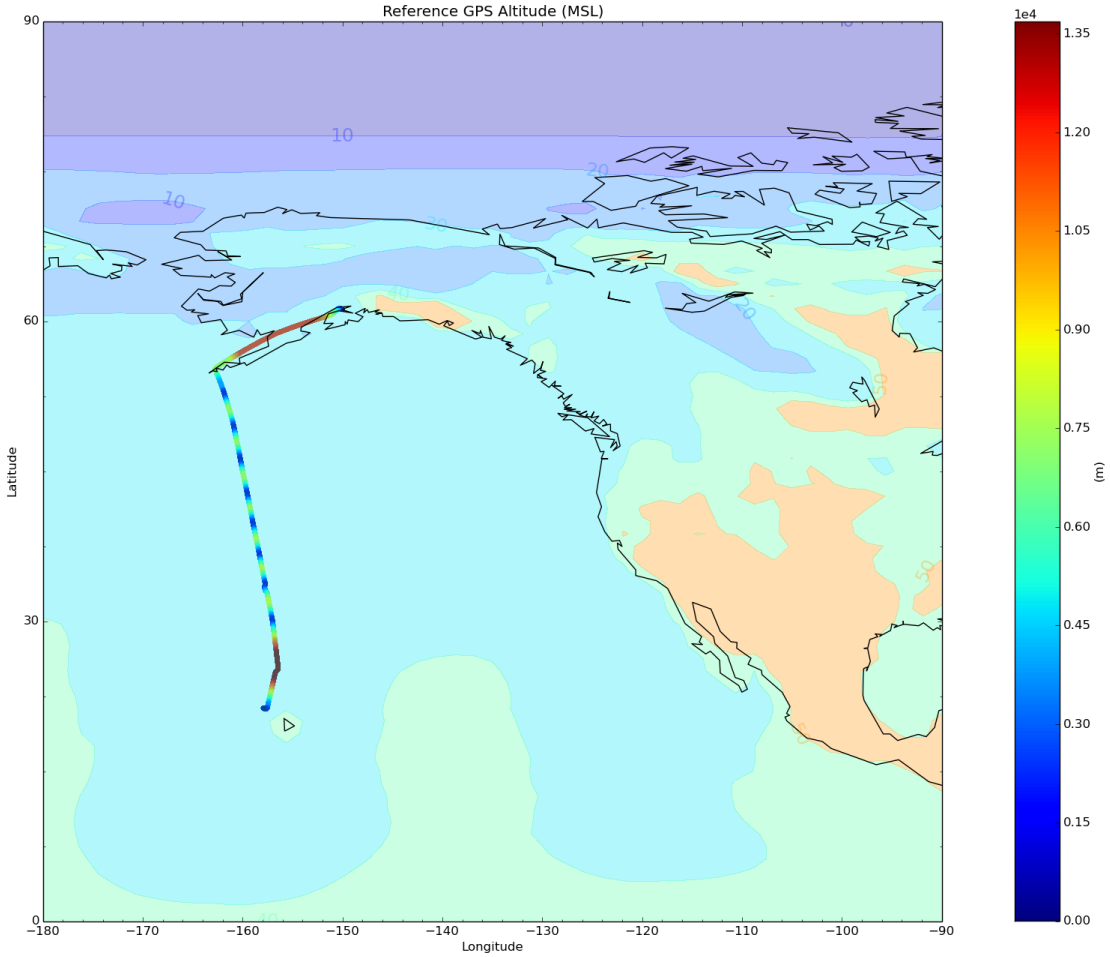
12.2 Filled contour with transparency on NASA Blue Marble

```
cis plot od550aer:HadGEM_od550aer-subset.nc:cmap=Reds,type=contourf,transparency=0.5,cmin=0.15 --type
```

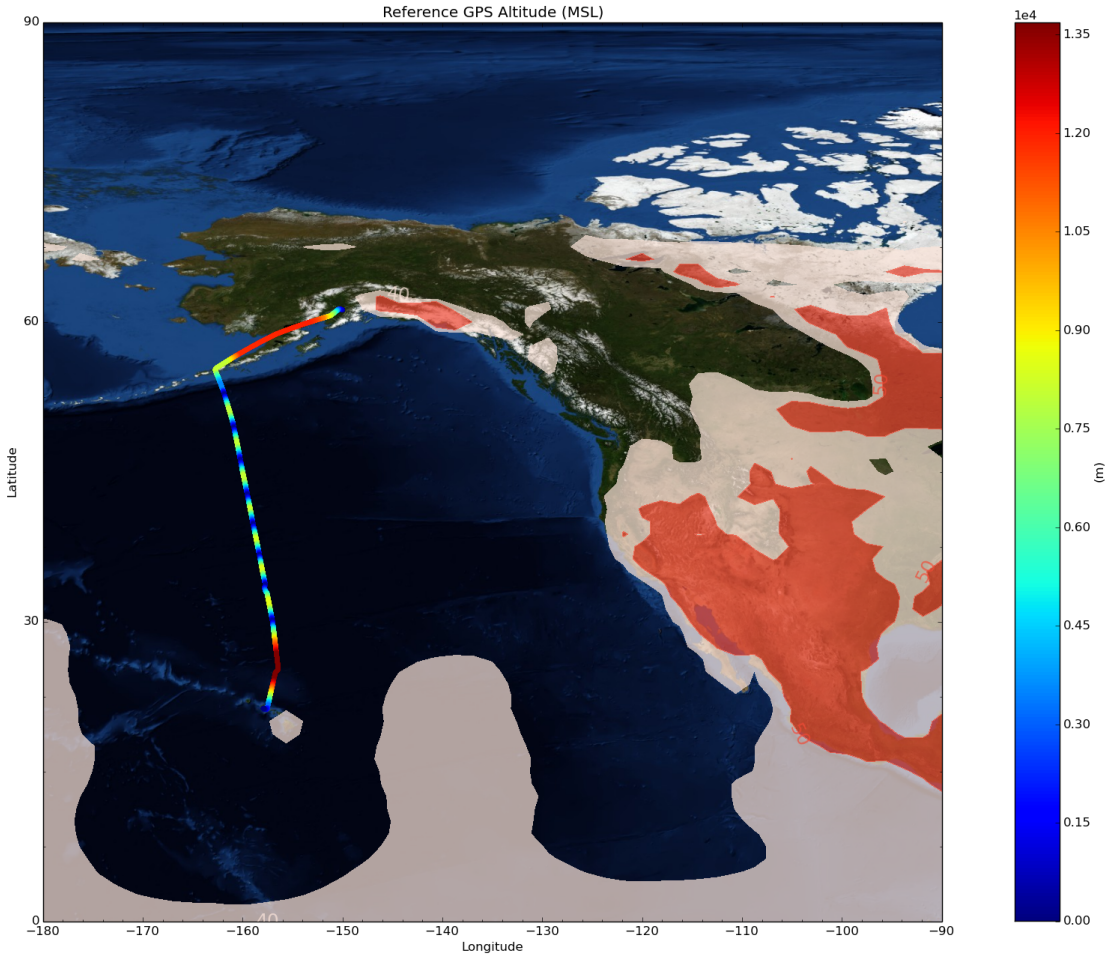


12.3 Scatter plus Filled Contour

```
cis subset rsutcs:HadGEM_rsutcs-subset.nc x=[-180,-90],y=[0,90] -o HadGEM_rsutcs-subset2

cis plot GGALT:RF04.20090114.192600_035100.PNI.nc:type=scatter rsutcs:HadGEM_rsutcs-subset2.nc:type=c
```




```
cis plot GGALT:RF04.20090114.192600_035100.PNI.nc:type=scatter rsutcs:HadGEM_rsutcs-subset2.nc:type=c
```



12.4 File Locations

The gridded data files can be found at:

```
/group_workspaces/jasmin/cis/AeroCom/A2/HadGEM3-A-GLOMAP.A2.CTRL/renamed
```

and the ungridded:

```
/group_workspaces/jasmin/cis/jasmin_cis_repo_test_files
```

CHAPTER 13

Maintenance and Developer Guide

13.1 Experimental Branches

To checkout a particular branch, simply type `git checkout branchname`

- 'slice': this branch has a slicing functionality that can be used via the command line argument –slice
- 'griddedgriddedcolocation': includes functionality for CF-compliant gridded-gridded colocation using the Iris library.

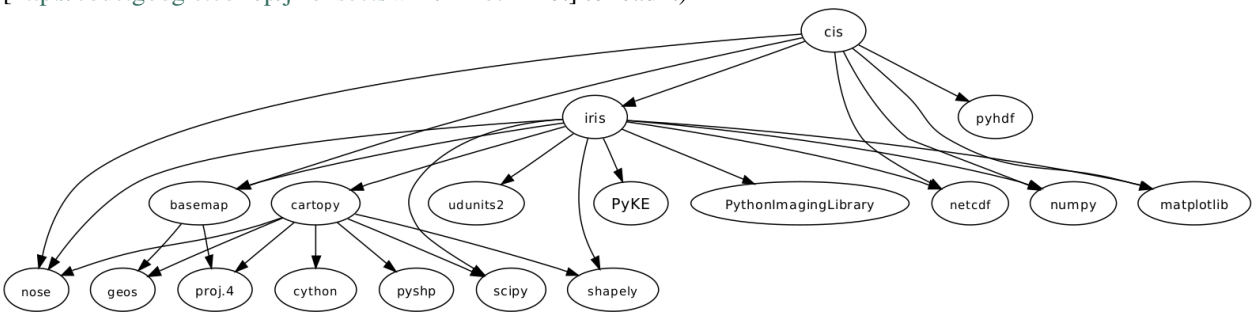
Note that these branches are deemed experimental and therefore not stable (that's why they are in branch and not the in main trunk!), so it comes with no warranty.

13.2 Unit test suite

The unit tests suite can be ran using Nose readily. Just go the root of the repository (i.e. cis) and type `nose` and this will run the full suite of tests. A comprehensive set of test data sets can be found under the `test/test_files` directory. A `harness` directory is provided and contains a couple of test harness for those data files. Finally, integration system-level tests are provided under the `test/plot_tests` directory and be ran using the `run_all.sh` script.

13.3 Dependencies

A graph representing the dependency tree can be found at `doc/cis_dependency.dot` (use [http://code.google.com/p/jrfonseca/wiki/XDot XDot] to read it)



13.4 API Documentation

The API reference can be generated using the following command

‘‘ python setup.py gendoc‘‘

This will automatically generates the documentation using [http://epydoc.sourceforge.net/ Epydoc] in html under the directory ‘’doc/api” from the [http://epydoc.sourceforge.net/docstrings.html docstrings] in the code.

13.5 Plugin development

Users can write their own “plugins” for providing extra functionality for reading and colocation. The main program will look at the environment variable `CIS_PLUGIN_HOME` for any classes that subclass the relevant top level class - as described for data products and colocation below.

The simplest way to set this environment variable is to add this to your `.bashrc` file:

```
$ export CIS_PLUGIN_HOME=/path/to/dir
```

13.5.1 Data Products

Users can write their own plugins for reading in different types of data. CIS uses the notion of a ‘data product’ to encapsulate the information about different types of data. They are concerned with ‘understanding’ the data and it’s coordinates and producing a single self describing data object.

A data product is a subclass of AProduct and as such must implement the abstract methods:

`get_file_signature(self)` Returns a list of regex’s to match the product’s file naming convention. CIS will use this to decide which data product to use for a given file. The first product with a signature that matches the filename will be used. The order in which the products are searched is determined by the priority property, highest value first; internal products generally have a priority of 10. The product can open the file to determine whether it can read it see `get_file_type_error`.

`create_coords(self, filenames)` Create a Coordinate object from the data files in the `filenames` parameter.

`create_data_object(self, filenames, variable)` Create and returns an ungridded data object for a given variable from many files. The `filenames` parameters is a list of filenames for the data. The parameter `variable` is the name of the variable to read from the dataset.

and may choose to implement:

`get_variable_names(self, filenames, data_type=None)` This return a list of valid variables names from the `filenames` list passed in. If not implemented the base function will be used. The `data_type` parameter can be used to specify extra information.

`get_file_type_error(self, filenames)` Check the `filename` to see if it is of the correct type and if not return a list of errors. If the return is None then there are no error and this is the correct data product to use for this file. This gives a mechanism for a data product to identify itself as the correct product to use even if a specific file signature can not be specified. For example GASSP is a type of NetCDF file and so filenames end with .nc but so do other NetCDF files, so the data product opens the file and looks for the GASSP version attribute, and if it doesn’t find it returns a error.

`get_file_format(self, filenames)` Returns a file format hierarchy separated by slashes, of the form TopLevelFormat/SubFormat/SubFormat/Version, e.g. NetCDF/GASSP/1.0, ASCII/ASCIIHyperpoint, HDF4/CloudSat This is used within the ceda di indexing tool. If not set it will default to the products name.

Here is a sketch of a data product implementation:

```
class MyProd(AProduct):

    #set the priority to be higher than the other netcdf file types
    priority = 20

    def get_file_signature(self):
        return [r'.*something*', r'.*somethingelse*']

    def create_coords(self, filenames):

        logging.info("gathering coordinates")
        for filename in filenames:
            data1 = []
            data2 = []
            data3 = []

        logging.info("gathering coordinates metadata")
        metadata1 = Metadata()
        metadata2 = Metadata()
        metadata3 = Metadata()

        coord1 = Coord(data1,metadata1,'X') # this coordinate will be used as the 'X' axis when plotti
        coord2 = Coord(data2,metadata2,'Y') # this coordinate will be used as the 'Y' axis when plotti
        coord3 = Coord(data3,metadata3)

        return CoordList([coord1,coord2,coord3])

    def create_data_object(self, filenames, variable):

        logging.info("gathering data for variable " + str(variable))
        for filename in filenames:
            data = []

        logging.info("gatherings metadata for variable " + str(variable))
        metadata = Metadata()

        coords = self.create_coords(filenames)
        return UngriddedData(data,metadata,coords)

    def get_file_type_error(self, filename):

        if not os.path.isfile(filename):
            return ["File does not exist"]

        if not file_has_attribute("file_type", filename):
           return ["File has wrong file type"]

        return None

    def get_variable_names(self, filenames, data_type=None):
        vars = variable_names_from_file
        del vars['Not useful']
        return vars
```

13.5.2 Colocation

Users can write their own plugins for performing the colocation of two data sets. There are three different types of plugin available for colocation and each will be described briefly below.

Kernel

A kernel is used to convert the constrained points into values in the output. There are two sorts of kernel one which act on the final point location and a set of data points (these derive from Kernel) and the more specific kernels which act upon just an array of data (these derive from AbstractDataOnlyKernel, which in turn derives from Kernel). The data only kernels are less flexible but should execute faster. To create a new kernel inherit from `Kernel` and implement the abstract method `get_value(self, point, data)`. To make a data only kernel inherit from AbstractDataOnlyKernel and implement `get_value_for_data_only(self, values)` and optionally overload `get_value(self, point, data)`.

`get_value(self, point, data)`

> This method should return a single value (if `Kernel.return_size` is 1) or a list of n values (if `Kernel.return_size` is n) based on some calculation on the data given a single point. The data is deliberately left unspecified in the interface as it may be any type of data, however it is expected that each implementation will only work with a specific type of data (gridded, ungridded etc.) Note that this method will be called for every sample point and so could become a bottleneck for calculations, it is advisable to make it as quick as is practical. If this method is unable to provide a value (for example if no data points were given) a ValueError should be thrown.

`get_value_for_data_only(self, values)`

> This method should return a single value (if `Kernel.return_size` is 1) or a list of n values (if `Kernel.return_size` is n) based on some calculation on the values (a numpy array). Note that this method will be called for every sample point in which data can be placed and so could become a bottleneck for calculations, it is advisable to make it as quick as is practical. If this method is unable to provide a value (for example if no data points were given) a ValueError should be thrown. This method will not be called if there is no values to be used for calculations.

Constraint

The constraint limits the data points for a given sample point. The user can also add a new constraint method by subclassing Constraint and providing an implementation for `constrain_points`. If more control is needed over the iteration sequence then the method `get_iterator` can be overloaded in additional to constrain_points, this may not be respected by all colocators who may still iterate over all sample data points. To enable a constraint to use a AbstractDataOnlyKernel the method `get_iterator_for_data_only` should be implemented (again this may be ignored by a colocator).

`constrain_points(self, ref_point, data)`

> This method should return a subset of the data given a single reference point. It is expected that the data returned should be of the same type as that given - but this isn't mandatory. It is possible that this function will return zero points, or no data. The colocation class is responsible for providing a fill_value.

`get_iterator(self, missing_data_for_missing_sample, coord_map, coords, data_points, shape, points, output_data)`

> The method should return an iterator over the output indices, hyper point for the output and data points for that output hyper point. This may not be called by all colocators who may choose to iterate over all sample points instead. The arguments are: * `missing_data_for_missing_sample` if True the iterator should not iterate over any points in the sample points which are missing. * `coord_map` is a

list of tuples of indexes of sample points coords, data coords and output coords * `coords` are the coords that the data should be mapped on * `data_points` are the non-masked data points * `shape` is the final shape of the data * `points` is the original sample points object * `output_data` is the output data

`get_iterator_for_data_only(self, missing_data_for_missing_sample, coord_map, coords, data_points, shape, points, values)`

The method should return an iterator over the output indices and a numpy array of the data values. This may not be called by all colocators who may choose to iterate over all sample points instead. The parameters are the same as `get_iterator`.

Co-locator

Another plugin which is available is the colocation method itself. A new one can be created by subclassing Colocator and providing an implementation for `colocate(self, points, data, constraint, kernel)`. This method takes a number of points and applies the given constraint and kernel methods on the data for each of those points. It is responsible for returning the new data object to be written to the output file. As such, the user could create a colocation routine capable of handling multiple return values from the kernel, and hence creating multiple data objects, by creating a new colocation method.

Plugins

For all of these plugins any new variables, such as limits, constraint values or averaging parameters, are automatically set as attributes in the relevant object. For example, if the user wanted to write a new constraint method (`AreaConstraint`, say) which needed a variable called `area`, this can be accessed with `self.area` within the constraint object. This will be set to whatever the user specifies at the command line for that variable, e.g.:

```
$ ./cis.py col my_sample_file rain:"model_data_?.nc"::AreaConstraint,area=6000,fill_value=0.0:nn_gric
```

Example implementations of new colocation plugins are demonstrated below for each of the plugin types:

```
class MyColocator(Colocator):

    def colocate(self, points, data, constraint, kernel):
        values = []
        for point in points:
            con_points = constraint.constrain_points(point, data)
            try:
                values.append(kernel.get_value(point, con_points))
            except ValueError:
                values.append(constraint.fill_value)
        new_data = LazyData(values, data.metadata)
        new_data.missing_value = constraint.fill_value
        return new_data


class MyConstraint(Constraint):

    def constrain_points(self, ref_point, data):
        con_points = []
        for point in data:
            if point.value > self.val_check:
                con_points.append(point)
        return con_points
```

```
class MyKernel(Kernel):

    def get_value(self, point, data):
        nearest_point = point.furthest_point_from()
        for data_point in data:
            if point.compdist(nearest_point, data_point):
                nearest_point = data_point
        return nearest_point.val
```

CHAPTER 14

Indices and tables

- genindex
- modindex
- search