04-Atmospheric_Data_Formats

Unknown Author

March 10, 2014

1 Manipulating Atmospheric Science data formats

Analysing data often involves converting files from one format to another, either to put multiple data sources in a common format or to enable us to use a tool which doesn't support the source format.

In this submodule you will learn:

1. How to convert gridded data in Met Office PP format to NetCDF
2. How to convert NASA Ames format to NetCDF
3. How to plot gridded data from Python
4. How to plot timeseries data from Python

1.1 1. Converting gridded data

Source data from numerical models may be output in various formats. Altough some models, such as WRF, output directly to NetCDF, others such as the ECMWF models use the WMO standard GRIB format. The Met Office Hadley Centre models output in PP format which is only used at the Met Office.

There are a couple of command-line tools that convert from PP and grib. Also, several Python libraries understand these formats and some can automatically convert to NetCDF. In this case you have the option to analyse directly from the source format, although it will often be convenient to output results in NetCDF.

1.1 Converting Met Office PP format

`xconv` is a graphical interface for extracting and visualising PP and NetCDF data. It is well suited for creating quick-look images of gridded data. `convsh` is a non-graphical interface to the `xconv` core routines.

- BADC provides supporting documentation for xconv : http://badc.nerc.ac.uk/help/software/xconv/
- Official documentation also available at : http://www.met.reading.ac.uk/~jeff/xconv

Several Python libraries can also read PP format and write NetCDF. The options we recommend are:

1. `cf-python` http://scitools.org.uk/iris/ : A library developed at University of Reading with a strong emphasis on CF compliance.
2. `IRIS` http://cfpython.bitbucket.org/ : A tool developed at the Met Office for reading, analysing and visualising gridded data.
3. `cdat_lite` http://proj.badc.rl.ac.uk/cedaservices/wiki/CdatLite : A similar tool developed at PCMDI, USA with the PP format code maintained by BADC.

We will concentrate on cf-python for format conversion. However, cdat_lite remains the most efficient way of reading large PP files in Python.

The cf-python module gives us a high-level interface to NetCDF and PP data with very good support for the CF-NetCDF conventions. cf-python.

In [2]:
```
import cf
hadcm3_jan = cf.read('data/aatzja.pm90jan.pp')
print 'Number of fields:', len(hadcm3_jan)
```

```
Number of fields: 101
```

This file contains 101 different CF-NetCDF fields (i.e. variables containing data rather than coordinate axes, etc.). 101 is too many to list here so we shall use a loop to find all fields with a name containing the string temperature

In [110]:
```
for i, field in enumerate(hadcm3_jan):
    name = field.name()
    if 'temperature' in name:
        print '%2s : %s' % (i, name)
```

```
20 : air_temperature
37 : surface_temperature
65 : air_temperature
66 : soil_temperature
78 : air_potential_temperature
```

You can find out a lot about a field by printing it. Let's take a look at one of these temperature fields.

In [111]:
```
print hadcm3_jan[37]
```

```
surface_temperature field summary
---------------------------------
Data             : surface_temperature(latitude(73), longitude(96)) K
Cell methods     : time: mean
Dimensions       : time(1) = [1890-01-16 00:00:00] 360_day
                 : latitude(73) = [90.0, ..., -90.0] degrees_north
                 : longitude(96) = [0.0, ..., 356.25] degrees_east
```

Now we will write just this field to a NetCDF file.

In [5]:
```
cf.write(hadcm3_jan[37], 'data/tas_hadcm3_cf.nc')
```

We can execute shell scripts within an IPython notebook with the %%sh magic command. Here we use the command-line tool ncdump to check the resulting file.

In [112]:
```
%%sh
ncdump -h data/tas_hadcm3_cf.nc
```

```
netcdf tas_hadcm3_cf {
dimensions:
        bounds2 = 2 ;
        latitude = 73 ;
        longitude = 96 ;
variables:
```

```
        double time_bounds(bounds2) ;
        double time ;
                time:units = "days since 1890-1-1" ;
                time:standard_name = "time" ;
                time:bounds = "time_bounds" ;
                time:calendar = "360_day" ;
                time:axis = "T" ;
        double latitude_bounds(latitude, bounds2) ;
        double latitude(latitude) ;
                latitude:units = "degrees_north" ;
                latitude:standard_name = "latitude" ;
                latitude:bounds = "latitude_bounds" ;
                latitude:axis = "Y" ;
        double longitude_bounds(longitude, bounds2) ;
        double longitude(longitude) ;
                longitude:units = "degrees_east" ;
                longitude:standard_name = "longitude" ;
                longitude:bounds = "longitude_bounds" ;
                longitude:axis = "X" ;
        float surface_temperature(latitude, longitude) ;
                surface_temperature:_FillValue = -1073741824. ;
                surface_temperature:coordinates = "time" ;
                surface_temperature:long_name = "SURFACE TEMPERATURE
AFTER TIMESTEP" ;
                surface_temperature:standard_name =
"surface_temperature" ;
                surface_temperature:cell_methods = "time: mean" ;
                surface_temperature:units = "K" ;

// global attributes:
                :Conventions = "CF-1.5" ;
                :source = "UM" ;
                :runid = "aatzj" ;
                :stash_code = 24 ;
                :lbproc = 0 ;
                :submodel = 1 ;
                :history = "Converted from PP by cf-python v0.9.8.1" ;
}
```

1.2 1.3 Converting NASA Ames format to NetCDF

Nappy is a Python and command-line tool written by Ag Stephens at the BADC (http://proj.badc.rl.ac.uk/cows/wiki/CowsSupport/Nappy).

In [7]:
```
! na2nc
```

```
na2nc.py
========

Converts a NASA Ames file to a NetCDF file.

Usage
=====
```

```
   na2nc.py [-m <mode>] [-g <global_atts_list>]
            [-r <rename_vars_list>] [-t <time_units>] [-n]
            -i <na_file> [-o <nc_file>]

Where
-----

    <mode>                      is the file mode, either "w" for write
or "a" for append
    <global_atts_list>          is a comma-separated list of global
attributes to add
    <rename_vars_list>          is a comma-separated list of
<old_name>,<new_name> pairs to rename variables
    <time_units>                is a valid time units string such as
"hours since 2003-04-30 10:00:00"
    -n                          suppresses the time units warning if
invalid
    <na_file>                   is the input NASA Ames file path
    <nc_file>                   is the output NetCDF file path
(default is to replace ".na" from NASA Ames
                                 file with ".nc").


ERROR: Please provide argument '-i <na_file>'
```

One issue with the NASA Ames compared to CF-NetCDF is that they don't have a standard representation of time. Therefore you need to set the time units during conversion. This requires inspecting the file.

In [61]:
```
%%sh
head data/cv-noxy_capeverde_20080301.na
```

```
50 1001
Lee James
Department of Chemistry, University of York, York, YO10 5DD
NOxy data from the Cape Verde Atmospheric Observatory
Cape Verde Atmospheric Observatory
1  1
2008 03 01 2008 05 08
                   0
time (days since 2006-01-01 00:00:00)
                   6
```

Fortunately the comment states the time units very clearly. The CF standard representation of this will be **`"days since 2006-01-01 00:00:00"`**

In [64]:
```
%%sh
na2nc -i data/cv-noxy_capeverde_20080301.na \
      -o data/cv-noxy_capeverde_20080301.nc  -t 'days since 2006-01-01 00
```

Nappy is written in Python so can be controlled from a Python script. It also includes functions for converting back to NASA Ames and converting to CSV. Other examples of what Nappy can do are at the Nappy Home Page

In [65]:

```
%%sh
ncdump -h data/cv-noxy_capeverde_20080301.nc
```

```
netcdf cv-noxy_capeverde_20080301 {
dimensions:
        time = UNLIMITED ; // (3575 currently)
variables:
        double time(time) ;
                time:name = "time (days since 2006-01-01 00:00:00)" ;
                time:long_name = "time (days since 2006-01-01
00:00:00)" ;
                time:standard_name = "time" ;
                time:units = "days since 2006-01-01 00:00:00" ;
                time:calendar = "gregorian" ;
                time:axis = "T" ;
        double no_mixing_ratio(time) ;
                no_mixing_ratio:title = "NO mixing ratio" ;
                no_mixing_ratio:long_name = "NO mixing ratio" ;
                no_mixing_ratio:units = "pptv" ;
                no_mixing_ratio:missing_value = 9999. ;
                no_mixing_ratio:nasa_ames_var_number = 0 ;
        double error_flag(time) ;
                error_flag:title = "Error Flag" ;
                error_flag:long_name = "Error Flag" ;
                error_flag:units = "NO" ;
                error_flag:missing_value = 9999. ;
                error_flag:nasa_ames_var_number = 1 ;
        double no2_mixing_ratio(time) ;
                no2_mixing_ratio:title = "NO2 mixing ratio" ;
                no2_mixing_ratio:long_name = "NO2 mixing ratio" ;
                no2_mixing_ratio:units = "pptv" ;
                no2_mixing_ratio:missing_value = 9999. ;
                no2_mixing_ratio:nasa_ames_var_number = 2 ;
        double noy_mixing_ratio(time) ;
                noy_mixing_ratio:title = "NOy mixing ratio" ;
                noy_mixing_ratio:long_name = "NOy mixing ratio" ;
                noy_mixing_ratio:units = "pptv" ;
                noy_mixing_ratio:missing_value = 9999. ;
                noy_mixing_ratio:nasa_ames_var_number = 4 ;

// global attributes:
                :Conventions = "CF-1.0" ;
                :comment = "==== Special Comments follow ====\nNOxy
inlet mounted at 6 metres on the 30 metre tower.\nData cycle is 10
minutes duration consiting of:\n60 seconds NO pre chamber zero\n140
seconds NO signal\n140 seconds NO2 signal\n60 seconds NO2 pre chamber
zero\n60 seconds NOy pre chamber zero\n140 seconds NOy signal\nThe
time stamp is the end of the 10 minute cycle\nNO is measured directly
by chemiluminesence\nNO2 is meausured by conversion to NO via a blue
light diode photolytic converter (~45% conversion efficiency)\nTotal
NOy is measured by conversion to NO via a heated Molybdenum catalyst
(~95% conversion efficiency for NO2)\nDetection limits:\nThe 2 sigma
detection limits for the 10 minute data cycle are:\n3.5 pptv for
NO\n10.2 pptv for NO2\n5.1 pptv for NOy\nThese can be decreased by
```

```
further averaging - for instance for 1 hour (6 points) averaged data,
the detection limits are:\n1.5pptv for NO\n4.1 pptv for NO2\n3.4 pptv
for NOy\nThese detection limits consist mainly of the photon counting
noise in the background signal.\nError Flag = 0 Good data\nError Flag
= 1 Reduced quality data\nError Flag = 2 Below detection limit\nError
Flag = 3 Invalid or missing data\n==== Special Comments end ====\n====
Normal Comments follow ====\nTHIS FILE ENDS = 2008 03 31\nEMAIL
CONTACT = jdl3@york.ac.uk\nNOXY\nDate and Time (days since 2006-01-01
00:00:00 + 00:00)\n==== Normal Comments end ====" ;
                :title = "Cape Verde Atmospheric Observatory" ;
                :file_number_in_set = 1 ;
                :source = "NOxy data from the Cape Verde Atmospheric
Observatory" ;
                :first_valid_date_of_data = 2008, 3, 1 ;
                :total_files_in_set = 1 ;
                :no_of_nasa_ames_header_lines = 50 ;
                :file_format_index = 1001 ;
                :institution = "Lee James (ONAME from NASA Ames file);
Department of Chemistry, University of York, York, YO10 5DD (ORG from
NASA Ames file)." ;
                :history = "2008-05-08 - NASA Ames File
created/revised.\n\n2014-03-10 21:47:14 - Converted to CDMS (NetCDF)
format using nappy-0.3.0." ;
}
```

1.3 Plotting Gridded data

Now we will return to the NetCDF file we created earlier from PP data and show how we can plot it using the tool `IRIS`.

In [24]:
```
import iris
import iris.quickplot as qplt
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
```

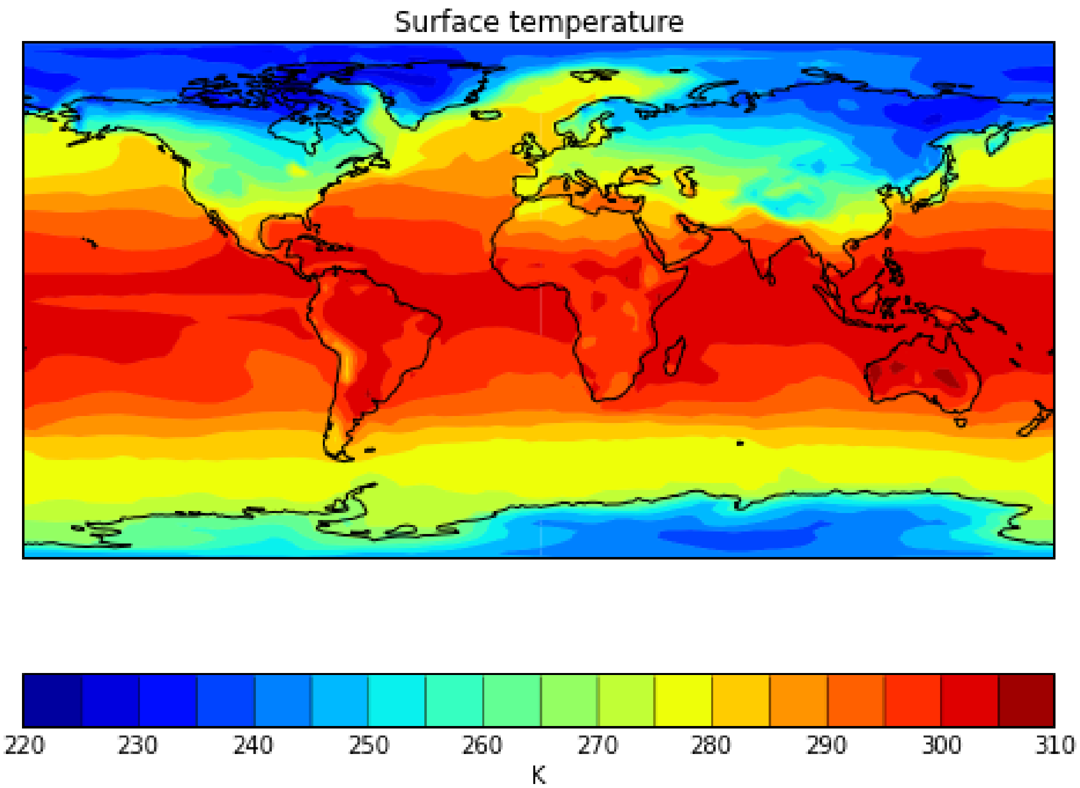
In [25]:
```
temp = iris.load_cube('data/tas_hadcm3_cf.nc', iris.Constraint('surface_te
print temp
```

```
surface_temperature / (K)           (latitude: 73; longitude: 96)
     Dimension coordinates:
          latitude                           x              -
          longitude                          -              x
     Scalar coordinates:
          time: 1890-01-16 00:00:00, bound=(1890-01-01 00:00:00,
1890-02-01 00:00:00)
     Attributes:
          Conventions: CF-1.5
          history: Converted from PP by cf-python v0.9.8.1
          lbproc: 0
          runid: aatzj
          source: UM
          stash_code: 24
          submodel: 1
```

```
        Cell methods:
             mean: time
```

In [14]:
```
fig = plt.figure(figsize=(8,6))
ax = plt.axes(projection=ccrs.PlateCarree())
qplt.contourf(temp, 20, axes=ax)
ax.coastlines()
```

Out [14]: <cartopy.mpl.feature_artist.FeatureArtist at 0x6089150>



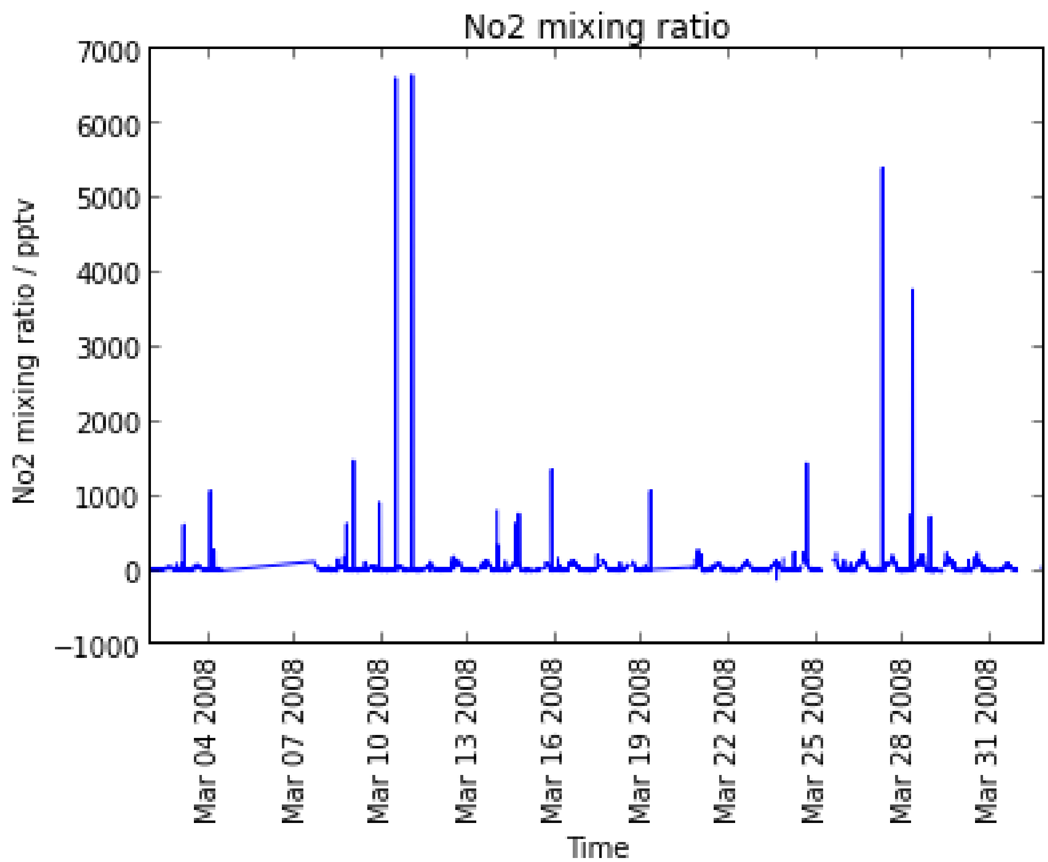
Timeseries plotting

In [66]:
```
cubes = iris.load('data/cv-noxy_capeverde_20080301.nc')
print cubes
```

```
0: NO mixing ratio / (pptv)              (time: 3575)
1: NOy mixing ratio / (pptv)             (time: 3575)
2: Error Flag / (unknown)                (time: 3575)
3: NO2 mixing ratio / (pptv)             (time: 3575)
```

In [75]:
```
no, noy, err, no2 = cubes
plt.xticks(rotation=90)
qplt.plot(no2)
```

Out [75]: [<matplotlib.lines.Line2D at 0x1475d790>]

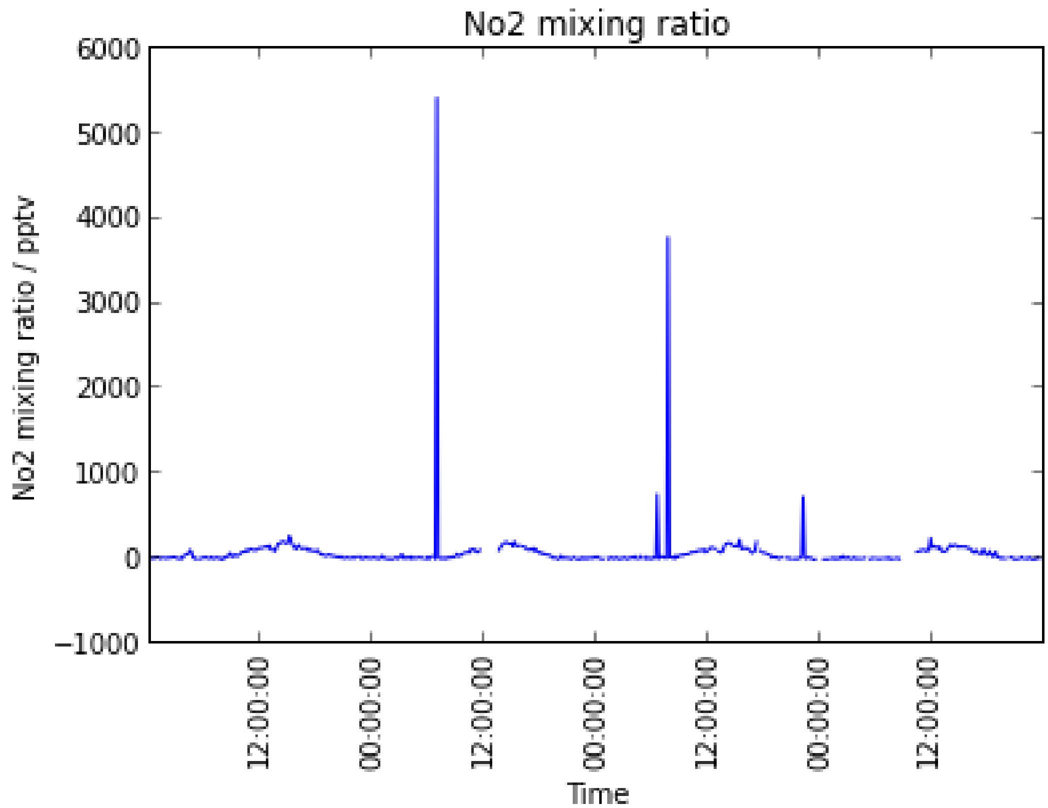


In [108]:
```
from datetime import datetime, timedelta

mar26 = datetime(2008, 3,26)
mar30 = mar26 + timedelta(days=4)
time_units = no2.coord('time').units

print mar26, mar30
time_range = time_units.date2num([mar26, mar30])
print time_range
```

```
2008-03-26 00:00:00 2008-03-30 00:00:00
[ 815.  819.]
```

In [109]:
```
constraint = iris.Constraint(coord_values={'time': lambda t: t > time_rang
plt.xticks(rotation=90)
qplt.plot(constraint.extract(no2))
```

Out [109]: [<matplotlib.lines.Line2D at 0xf5d6350>]


No2 mixing ratio
No2 mixing ratio / pptv
6000
5000
4000
3000
2000
1000
0
−1000
12:00:00
00:00:00
12:00:00
00:00:00
12:00:00
00:00:00
12:00:00
Time