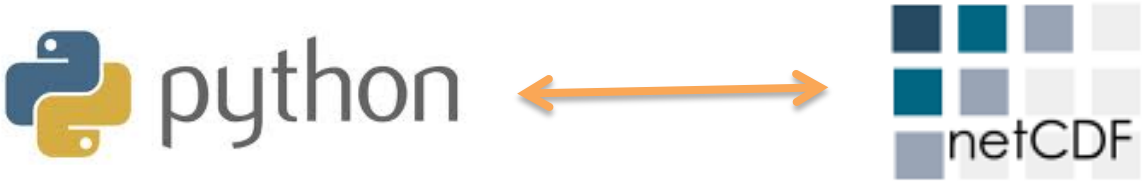

netcdf4-python: A python interface to the netcdf C library

Jeff Whitaker

NOAA Earth System Research Lab

<jeffrey.s.whitaker@noaa.gov>

What is Python?

- An interpreted, dynamic, all-purpose high-level programming language.
- Can be used ..
  - As a replacement for matlab, IDL for analysis.
  - To build GUI and web apps.
  - As a higher-level “glue” language to build interfaces to Fortran/C code.

Why Python?

- It's free!
- It's fun! (easy to learn, "fits your brain")
- Has great scientific library support (data formats, algorithms, plotting, you name it)
- A general purpose language (what you learn is transferable to other domains)
- Easy to interface fortran/C if you need speed or want to leverage legacy code.

Prerequistes

- Python 2.5 or later (python 2.7 recommended)
- Numpy array processing module from http://numpy.scipy.org.
- netCDF/HDF5 C libraries.
- netcdf4-python from http://netcdf4-python.googlecode.com.
  - PyNIO and Scientific.IO modules are similar, without advanced netcdf-4 features.
- Optional but recommended:
  - Matplotlib (http://matplotlib.sf.net) for plotting.
  - Scipy (http://scipy.org) for common algorithms.
- Enthought Python distro includes numpy, scipy, matplotlib and netcdf4-python (free for academic use only, others $$).

netCDF Dataset object

```
>>> import netCDF4 # import module
>>> nc = netCDF4.Dataset('test.nc','w',format='NETCDF4')
>>> print nc.file_format
NETCDF4
>>> nc.close()
```

API is similar to PyNIO or Scientific.IO.NetCDFFile.

- Dataset object contains dimensions, variables and groups (stored as dictionary attributes).
- Top level group is the Dataset object itself.

Dimensions

```
>>> nc = netCDF4.Dataset('test.nc','a') # re-open in 'append' mode
>>> lat_dim = nc.createDimension('lat',73)
>>> lon_dim = nc.createDimension('lon',144)
>>> time_dim = nc.createDimension('time',None) # unlimited dim
>>> print nc.dimensions
OrderedDict([('lat', <netCDF4.Dimension object at 0x102711b50>),
('lon', <netCDF4.Dimension object at 0x102711b90>), ('time',
<netCDF4.Dimension object at 0x102711bd0>)])
>>> print len(lon_dim)
144
>>> print time_dim.isunlimited()
True
```

- Setting dimension size to 0 or None makes it unlimited.
- If file_format='NETCDF4', multiple dimensions can be unlimited.

Variables

```
>>> import numpy as np # import numpy module
>>> mslp = nc.createVariable('mslp',np.float32,
('time','lat','lon'))
>>> mslp.standard_name = 'air_pressure_at_sea_level'
>>> print mslp.dimensions, mslp.shape, mslp.dtype, mslp.ndim
('time','lat','lon') (0,73,144) float32 3
```

- Data type specified by numpy type (float, int, float32, int16 etc).
- If file_format='NETCDF4', multiple dimensions can be unlimited.
- Attributes created by assigning values to Variable instances.
- Variable compression and chunk sizes may be specified by keywords in createVariable.
- Useful attributes include: shape, dimensions, dtype, ndim.

Writing data

```
>>> print data_arr.shape # 6 grids of pressure data
(6,73,144)
>>> mslp = data_arr # append along unlim dim
>>> print mslp.shape
(6,73,144)
>>> data_out = mslp[::2,lats>0,:] # every other time in North. Hem.
>>> print data_out.shape
(3,36,144)
```

- Just treat Variable object like a numpy array and assign data it.
- Variables automatically grow along unlimited dims.
- To retrieve data, just slice the Variable object.

Bells and whistles

- Conversion of time values to dates.
- Multi-file aggregation.
- Compression.
- Groups (think filesystem directories).
- Advanced data types:
  - Compound variables.
  - Variable-length arrays/strings.

Dealing with time

```
>>> from netCDF4 import date2num, num2date
>>> from datetime import datetime
>>> time_units = 'hours since 0001-01-01 00:00:00.0'
>>> d = datetime(2011,7,26,12); print date
2011-07-26 12:00:00
>>> print date2num(d,units=time_units,calendar='gregorian')
17624292.0
>>> print num2date(t,units=time_units,calendar='gregorian')
2011-07-26 12:00:00
>>> print num2date(t,units=time_units,calendar='julian')
2011-07-13 12:00:00
```

- Dealing with time coords has always been a PITA.
- date2num and num2date functions here to help.
- Supports many different calendars.
- Can handle arrays of date, time values.

Multi-file aggregation

```
>>> from netCDF4 import MFDataset
>>> nc = MFDataset('/datasets/wind_195*.nc')
```

- Uses file globbing to patch together all the files for the 1950's. Appears as one big dataset.
- Limitations:
  - Can only handle NETCDF3, NETCDF4_CLASSIC files.
  - Only can aggregate along unlimited dimension.
  - Slower than opening and reading files individually.
  - netcdf-java provides more advanced aggregation capabilities (via ncml).

Compression

```
>>> mslp = nc.createVariable('mslp',np.float32,
('time','lat','lon')) # no compression
>>> mslp = nc.createVariable('mslp',np.float32,
('time','lat','lon'),zlib=True) # 'lossless' zlib compression
>>> mslp = nc.createVariable('mslp',np.float32,
('time','lat','lon'), zlib=True,
least_signficant_digit=1) # "lossy" zlib compression
```

- zlib=True turns on zlib compression (with shuffle filter).
- least_significant_digit=1 truncates the data after the 1st decimal place. Can result in much smaller files.

Recap

- Python/numpy is a excellent matlab replacement.
- netcdf4-python exposes almost all of the netcdf-4 C lib to python in a nice OO interface.
- Would be great to incorporate some of the features found in netcdf-java (such as aggregation) in the future!
- Downloads, docs, issue tracker http://netcdf4-python.googlecode.com
- Questions

<jeffrey.s.whitaker@noaa.gov>