# Introduction to NetCDF

Luke Davis

Department of Atmospheric Science
Colorado State University

# Introduction

# Who am I?

# Background

**Problem**: Earth scientists work with huge, 4+ dimensional datasets (3 spatial dimensions, 1 time dimension, and sometimes even more).

- Example: Air temperature (longitude, latitude, height, time)
- Example: Surface pressure from “ensemble” of model simulations (longitude, latitude, time, ensemble member)
- Example: Satellite-retrieved radiation (projection x-coordinate, projection y-coordinate, wavenumber)

# Background

**Question**: What's the best way to **store** this type of data?

- Spreadsheets? Not enough dimensions.
- Matrices? No way to annotate "rows", "columns", etc.

**Answer**: The NetCDF format (Network Common Data Form) developed by UCAR/Unidata (right down the road!).

- Description: Annotated N-dimensional matrices (arrays)
- File extension: `.nc`

There are other similar data formats (HDF, GRIB), and some software can work seamlessly with different formats…but NetCDF is your new best friend.

# Architecture

File formats have different “version numbers”:

- NetCDF3 (version 3) retired in 2008. Unidata is currently version 4.
- …however version 3 still widespread (scientists are slow to change their ways…too many other things to worry about).
- Some things in NetCDF4 are impossible in NetCDF3 (multiple unlimited dimensions, …).
- Weird read/write bugs are sometimes due to version incompatibilities.

NetCDF3 still everywhere, so you may need to use it.

# Architecture

NetCDF files have the following featuers:

- Global attributes.
- Global dimensions.
- Named variables, each with its own dimensions and attributes.

# Architecture

```
$ ncdump -h example.nc
ADD EXAMPLE
```

# Software

There's lots of sotware for working with NetCDF (HDF, GRIB) files. First, the **command-line** tools:

- NCO (NetCDF Operators).

# Software

Next, the **python** tools:

- netCDF4 (confusingly, this works with both Netcdf versions 3 and 4!). netCDF4 is for low-level, fast control.
- xarray ("new kid" on the block but *extremely* powerful). xarray is for high-level, convenient control.
- Python also has Iris "cubes"…but this is older, less widespread, falling out of favor (xarray intended as improvement on "cubes").

# Command-line tools

# CDO Overview

Many of us know about NCO (NetCDF Operators) and NCL (NCAR Command Language).

A comparatively recent player is **CDO** (Climate Data Operators).

- Written in C++ by Max Planck Institut. **Companion** to NCO – does things NCO can't.
- Install with Anaconda, no sudo necessary: `conda install -c conda-forge cdo=1.9.0`.
- Great documentation, easy-to-remember syntax: `cdo -mergetime 2017*.nc out.nc`.
- **Subcommand chaining**: `cdo -add -sqr -selname,u f.nc -sqr -selname,v f.nc KE.nc`.

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# CDO Uses

CDO has **hundreds** of subcommands. Here's a quick survey:

- **Regrid** from one arbitrary grid (e.g. rotated pole, hexagonal cells) to another; choose from a suite of algorithms: `cdo remapcon,destination_grid.txt in.nc out.nc`.
- Interpolate to different **vertical levels**: `cdo intlevel,1000,900,800,700,600,500,400,300,200 in.nc out.nc`
- **LLS trends** ignoring missing vals: `cdo regres -seldate,1980-01-01T00:00:00,2009-12-31T23:59:59 in.nc out.nc`.
- Monthly daily and seasonal **statistics**: `cdo ymonmean in.nc climate.nc`.
- **Spatial** operations: `zonmean`; `fldmean`; `mulc,101325 -vertmean`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# XArray

If you use python, **XArray** $\gg$ **netCDF4**. XArray is to pandas as NetCDF files are to CSV files.

- Handy objects: Datasets (entire NetCDF file) and DataArrays (single variable within NetCDF file).
- Load Dataset with `d = xr.open_dataset("file.nc", engine="pynio")`. For GRIB files, use `xr.open_dataset("file.grb", engine="pynio")` (python 3 compatibility coming soon).
- Operate along dimensions: `d["variable"].mean(dim="longitude")`.
- Permute dimensions: `d.transpose(["time", "latitude", "longitude"])`.
- Slice dimensions: `d.sel(lat=slice(0,90))`; `d.sel(level=850,`

# NCL

There's another tool I didn't mention…the NCAR Command Language (NCL). This is one of my favorites!

- MATLAB: Everything is an array.
- Python: Everything is an object (or dictionary, depending on who you ask).
- NCL: Everything is a NetCDF-formatted dataset. If you're a geoscientist, this paradigm is pretty awesome.

Sadly, you should avoid using NCL for two reasons…

- NCL is being deprecated (Unidata developers now focusing on python tools).
- NCL is very slow…among the slowest tools (see these simple benchmarks).

# NCL

**Example**: Read XYZT temperature and wind data, save YZT eddy heat flux.

```
$ ncdump -h input.nc
dimensions:
  time = UNLIMITED ; // (200 currently)
  lev = 60 ;
  lat = 36 ;
  lon = 72 ;
variables:
  ...
  double v(time, lev, lat, lon) ;
    v:long_name = "meridional wind" ;
    v:units = "m/s" ;
  double t(time, lev, lat, lon) ;
    t:long_name = "temperature" ;
    t:units = "K" ;
```

# NCL

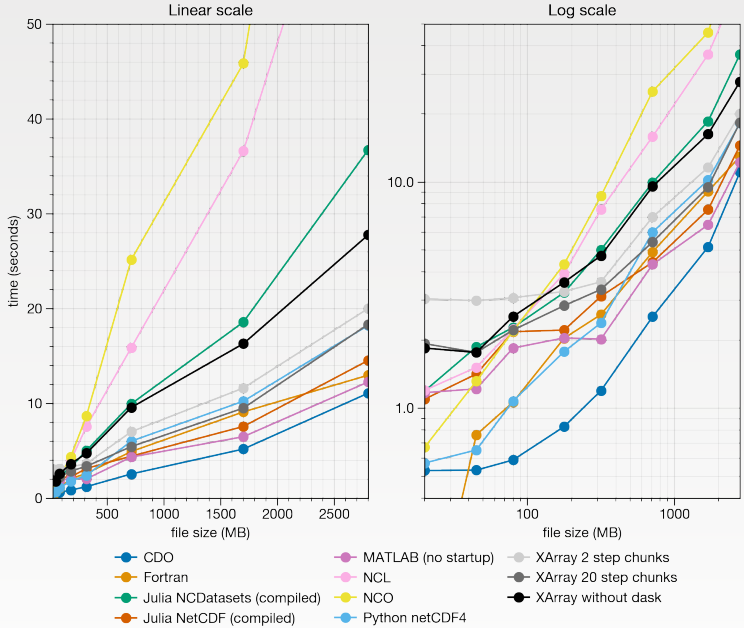
**Example**: Read XYZT temperature and wind data, save YZT eddy heat flux.

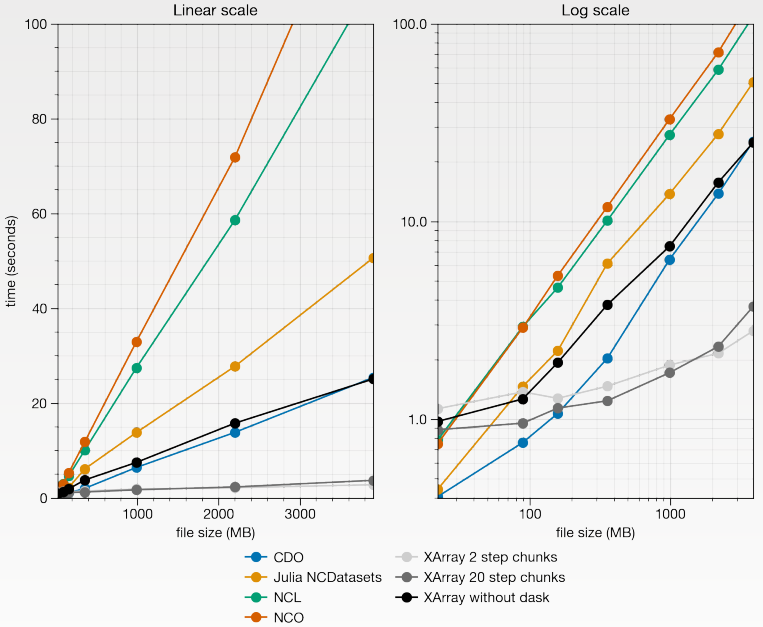
```
$ cat fluxes.ncl
f = addfile("input.nc", "r")
o = addfile("output.nc", "c")
t = f->t
v = f->v
ehf = dim_avg_n( \
  (t - conform(t, dim_avg_n(t, 2), (/0, 1/))) \
  * (v - conform(v, dim_avg_n(v, 2), (/0, 1/))), \
  3 \
)
copy_VarCoords(t(:, :, 0), ehf)
ehf@long_name = "eddy heat flux"
ehf@units = "K*m/s"
o->ehf = ehf
```

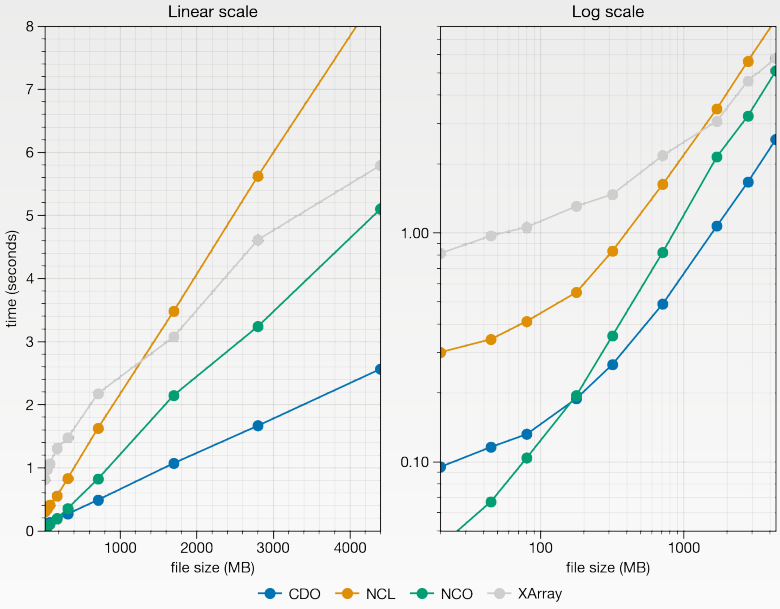
# NCL

**Example**: Read XYZT temperature and wind data, save YZT eddy heat flux.

```
$ ncl fluxes.ncl
$ ncdump -h output.nc
dimensions:
  time = 200 ;
  lev = 60 ;
  lat = 36 ;
variables:
  ...
  double ehf(time, lev, lat) ;
    ehf:units = "K*m/s" ;
    ehf:long_name = "eddy heat flux" ;
```


Fluxes benchmark on macbook
Linear scale
Log scale
time (seconds)
file size (MB)
file size (MB)
0
10
20
30
40
50
500
1000
1500
2000
2500
1.0
10.0
100
1000
CDO
Fortran
Julia NCDatasets (compiled)
Julia NetCDF (compiled)
MATLAB (no startup)
NCL
NCO
Python netCDF4
XArray 2 step chunks
XArray 20 step chunks
XArray without dask

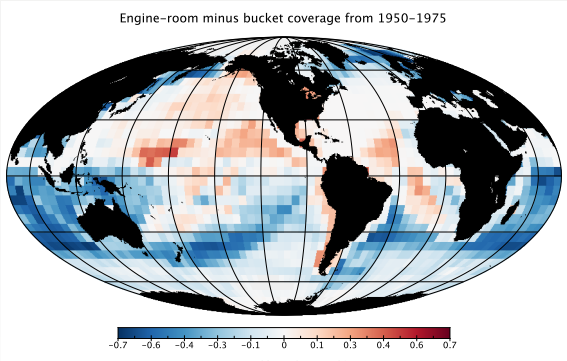

Fluxes benchmark on supercomputer
Linear scale
Log scale
time (seconds)
file size (MB)
file size (MB)
100
80
60
40
20
0
1000
2000
3000
100.0
10.0
1.0
100
1000
CDO
Julia NCDatasets
NCL
NCO
XArray 2 step chunks
XArray 20 step chunks
XArray without dask


Slice benchmark on macbook
Linear scale
Log scale
time (seconds)
file size (MB)
file size (MB)
0
1
2
3
4
5
6
7
8
1000
2000
3000
4000
1.00
0.10
100
1000
CDO
NCL
NCO
XArray

# Panoply

Many are familiar with **ncview**. Modern alternative is **Panoply**.

- Freeware released by NASA.
- Extremely easy to use.



# Python tools

# netCDF4

Now let’s get into the python tools.

# Examples

# Learning more

- Use google. Try rephrasing a few times if nothing comes up.
- Use stackoverflow! …but only *ask* questions after extensive googling fails.

  Tip: For inline-code examples, use `` `inline code' ``. For multiline-code examples, use ```` ```
  multiline code
  ''' ````.
- Use slack! Collaboration + cooperation saves everyone time.

  Tip: For inline-code and multiline-code, use backticks – just like stackoverflow.