



**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL



**University of  
Reading**



# The NCAS CF Data tools: cf-python and cf-plot\*...

...and how (we hope) they can be of use to you<sup>†</sup>

*\*and friends*

*<sup>†</sup>you = AFESP researchers & students*

Sadie Bartholomew

Computational Scientist, NCAS-CMS & Reading Met. Dept.

AFESP meeting, 2025-02-20



# 1. Background and context

- What is NCAS-CMS and what do we do?
- What are cf-python and cf-plot and why are they relevant?
- What does the CF in these tool names refer to?
- Why are the CF Conventions useful and how can we use them in practice?



# NCAS-CMS: providing support for modelling, tools & HPC

Myself and Sharar are part of the CMS (Computational Modelling Services) group, a sub-group of NCAS (National Centre for Atmospheric Science) who are mostly based here at the University of Reading (with a few elsewhere, Cambridge and Met Office):

- **NCAS** is the National Centre for Atmospheric Science, a research centre with staff embedded across various universities in the UK (see <https://ncas.ac.uk/>), lots of us at Reading
- **NCAS Computational Modelling Services i.e. NCAS-CMS** “*provides computational modelling services to the UK academic atmospheric and polar science community to support numerical modelling in climate, weather and earth-system research*” (see <https://cms.ncas.ac.uk/>)
- *In practice* this means we work to develop, maintain and support tools/libraries, workflows and models used in atmospheric science plus manage and support high-performance computing
- *In short: we help geoscientific researchers to do their research* (because it usually involves a very significant amount of computing and data usage, often complex or computationally-intensive, so it helps to have people who focus on the software, middleware and hardware side of things)

# The NCAS CF Data Tools

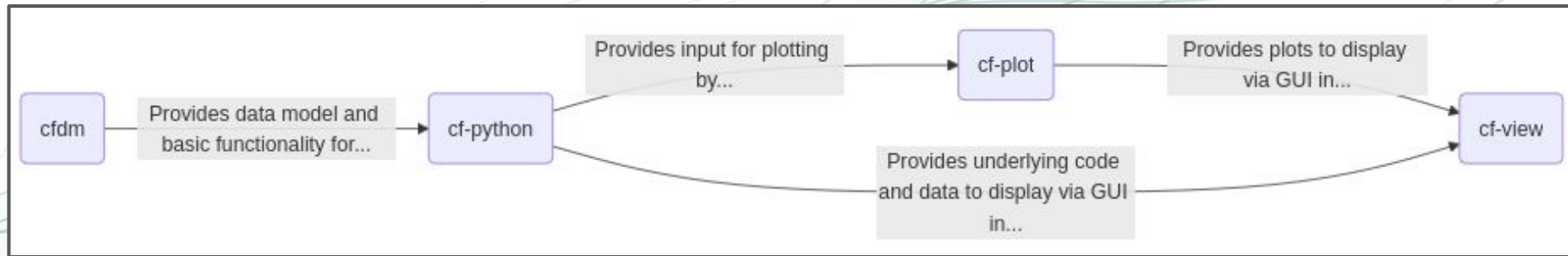
The (NCAS) CF Data Tools are a **small set of complementary Python open-source libraries developed by NCAS-CMS**. They are designed as tools to allow geoscientific researchers to do the tasks they tend to want to do as simply as possible e.g. in fewest lines, using an intuitive API (interface).

<b><u>Library</u></b>	<b><u>Description and purpose</u></b>	<b><u>Functionality in overview</u></b>
<b>cfdm</b>	Reference implementation of the CF data model	For the most part, only that required to read and write datasets, and to create, modify and inspect field constructs in memory
<b>cf-python</b>	CF-compliant geoscientific data analysis library	Much higher-level than cfdm, e.g. statistical operations, collapses, subsampling, regridding
<b>cf-plot</b>	Visualisation tool	Various forms of plotting e.g. contour, vector and line plots directly from field constructs (or numpy arrays)
<b>cf-view</b>	Graphical User Interface (GUI)	A GUI for exploration & plotting using cf-python and cf-plot
<b>cf-checker</b>	CF compliance checking utility	Checks the CF compliance of a netCDF file

# cf-python & cf-plot: main libraries for the average user

Of the five of the NCAS CF Data Tools libraries listed in the previous slide, cf-python and cf-plot are of most relevance to the average user, so we focus on these in this talk:

- cf-python for data processing and analysis: <https://ncas-cms.github.io/cf-python/>
- cf-plot for visualisation (plotting) capability with cf-python: <https://ncas-cms.github.io/cf-plot/build/>






# cf-python & cf-plot: 'recipes' illustrating capability


In the cf-python documentation, we now have a page listing created code recipes which demonstrate some possible applications of cf-python and cf-plot, see:

<https://ncas-cms.github.io/cf-python/recipes/index.html>



## cf 3.16.2

A CF-compliant earth science data analysis library

 **Star** **131**

Quick search

- Introduction
- CF data model
- Installation
- Cheat Sheet
- Recipes using cf
- Tutorial
- Analysis
- API reference
- Aggregation rules
- Performance
- Releases
- Change log
- Contributing

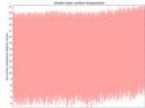
cf development has been supported by the ERC through Seachange and Couplet; by the

## Recipes using cf


Version 3.16.2 for version 1.11 of the CF conventions.

Click on the keywords below to filter the recipes according to their function:

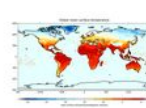
**All** **Aggregate** **Collapse** **Contourmap** **Histogram** **Lineplot** **Mathematical Operations** **Regrid** **Subspace** **Mask**




Calculating global mean temperature timeseries



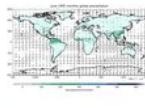
Calculating and plotting the global average temperature anomalies



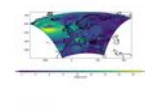
Plotting global mean temperatures spatially



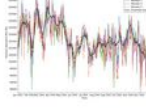
Comparing two datasets with different resolutions using regridding



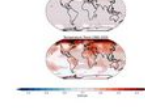
Plotting wind vectors overlaid on precipitation data




Converting from rotated latitude-longitude to regular latitude-longitude



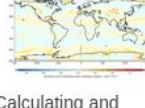
Plotting members of a model ensemble




Plotting statistically significant temperature trends with stippling



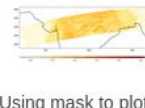
Plotting a joint histogram



Calculating and plotting the relative vorticity



Plotting the Warming Stripes



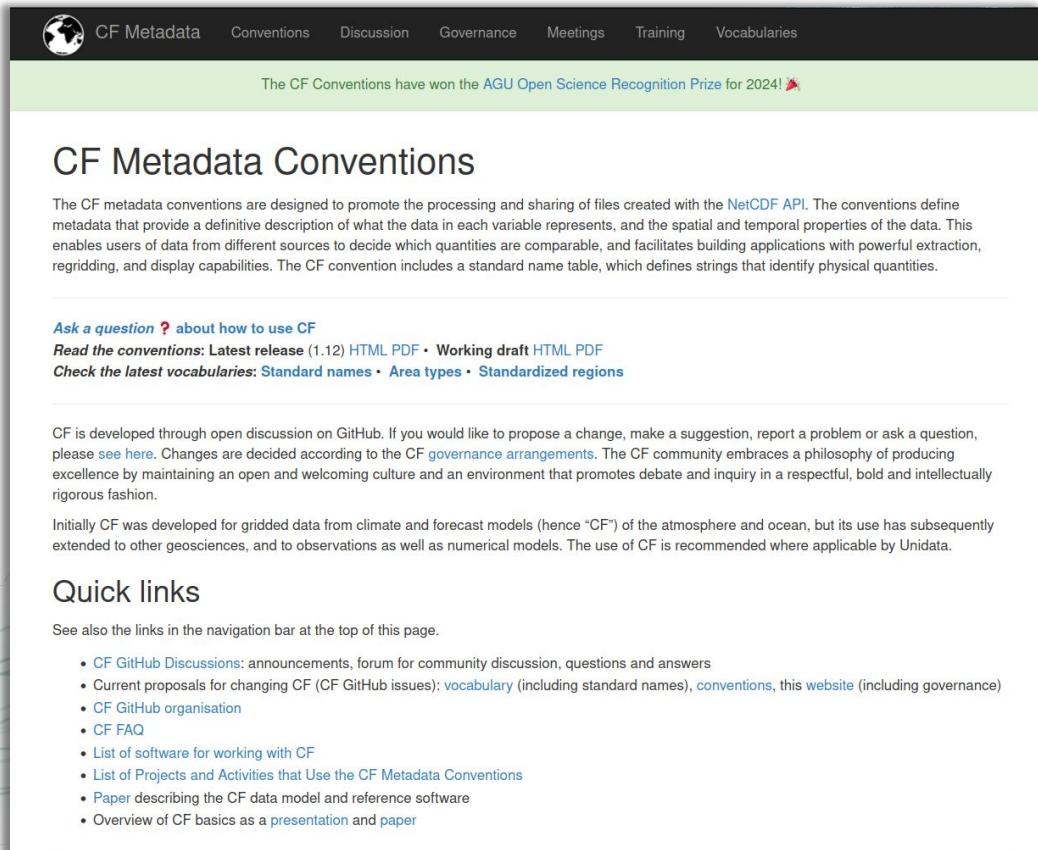
Using mask to plot Aerosol Optical Depth

# ‘CF’ in the tools’ names refers to the CF Conventions

‘CF’ is in the name for all of the tools because they center on supporting the **CF Conventions**, a metadata standard used throughout the geoscience e.g. Meteorology and atmospheric science communities that makes it easier to share, use and understand our datasets.

- The ‘C’ and ‘F’ from the CF Conventions stands for **C**limate and **F**orecast
- Some terminology: **NetCDF files + CF Metadata Conventions = CF-netCDF**
- Though cf-python and cf-plot etc. can work with datasets that aren’t (very, even at all) CF-compliant, they are designed to work best for CF-compliant data (not just netCDF but other formats e.g. Met Office PP and fields files)
- The idea is that they can use this standardised metadata intelligently so that they can tell what you want to do. Simple example: if you square your data array values, you probably want the units to become the squared version - so cf-python changes them as such automatically for you

# The CF Conventions: a community metadata standard



The screenshot shows the homepage of the CF Metadata Conventions website. The header is dark with a globe icon and navigation links: CF Metadata, Conventions, Discussion, Governance, Meetings, Training, and Vocabularies. A green banner below the header states: 'The CF Conventions have won the AGU Open Science Recognition Prize for 2024! 🏆'. The main heading is 'CF Metadata Conventions'. Below it, a paragraph explains the purpose of the conventions. A section titled 'Ask a question ? about how to use CF' provides links to the latest release (1.12) HTML PDF, working draft HTML PDF, and latest vocabularies (Standard names, Area types, Standardized regions). A 'Quick links' section follows, with a note to see also the links in the navigation bar. A list of links includes CF GitHub Discussions, current proposals, CF GitHub organisation, CF FAQ, list of software for working with CF, list of projects and activities that use the CF Metadata Conventions, paper describing the CF data model and reference software, and overview of CF basics as a presentation and paper.

CF Metadata Conventions

The CF metadata conventions are designed to promote the processing and sharing of files created with the [NetCDF API](#). The conventions define metadata that provide a definitive description of what the data in each variable represents, and the spatial and temporal properties of the data. This enables users of data from different sources to decide which quantities are comparable, and facilitates building applications with powerful extraction, regridding, and display capabilities. The CF convention includes a standard name table, which defines strings that identify physical quantities.

[Ask a question ? about how to use CF](#)

[Read the conventions: Latest release \(1.12\) HTML PDF](#) • [Working draft HTML PDF](#)

[Check the latest vocabularies: Standard names](#) • [Area types](#) • [Standardized regions](#)

CF is developed through open discussion on GitHub. If you would like to propose a change, make a suggestion, report a problem or ask a question, please [see here](#). Changes are decided according to the [CF governance arrangements](#). The CF community embraces a philosophy of producing excellence by maintaining an open and welcoming culture and an environment that promotes debate and inquiry in a respectful, bold and intellectually rigorous fashion.

Initially CF was developed for gridded data from climate and forecast models (hence “CF”) of the atmosphere and ocean, but its use has subsequently extended to other geosciences, and to observations as well as numerical models. The use of CF is recommended where applicable by Unidata.

## Quick links

See also the links in the navigation bar at the top of this page.

- [CF GitHub Discussions](#): announcements, forum for community discussion, questions and answers
- Current proposals for changing CF (CF GitHub issues): [vocabulary](#) (including standard names), [conventions](#), this [website](#) (including governance)
- [CF GitHub organisation](#)
- [CF FAQ](#)
- [List of software for working with CF](#)
- [List of Projects and Activities that Use the CF Metadata Conventions](#)
- [Paper](#) describing the CF data model and reference software
- Overview of CF basics as a [presentation](#) and [paper](#)

- The CF Conventions provide a standardised way to describe and encode multi-dimensional geospatial data, which is crucial for sharing, comparing, and analysing data across different platforms, models, and research groups.
- netCDF datasets are flexibly self-describing so, without a standard, concepts such as variable names and geometry encoding would need interpretation as to precise meaning
- See <https://cfconventions.org/> for e.g. the canonical latest document and list of standard names



# The CF Conventions: a community metadata standard

## Search

☒ AND ☐ OR (separate search terms with spaces)

☐ Also search help text

☒ Also search aliases text

☐ Only search canonical units

Advanced searches

Found 25 standard names matching query: sea\_surface\_height

## View by Category

[Atmospheric Chemistry](#) [Atmosphere Dynamics](#) [Carbon Cycle](#) [Cloud](#) [Hydrology](#)

[Ocean Dynamics](#) [Radiation](#) [Sea Ice](#) [Surface](#)

In the table below, click on a standard-name to show or hide its description and help text.

Standard Name	Canonical Units
<a href="#">change_in_sea_surface_height_due_to_change_in_air_pressure</a>	m
<a href="#">halosteric_change_in_sea_surface_height</a>	m
<a href="#">non_tidal_elevation_of_sea_surface_height</a>	m
<a href="#">ocean_rigid_lid_pressure_expressed_as_sea_surface_height_above_geoid</a>	m
<a href="#">sea_surface_height_above_geoid</a>	m
<a href="#">alias: sea_surface_elevation_anomaly</a>	m
<a href="#">alias: sea_surface_elevation</a>	m
<a href="#">sea_surface_height_above_geopotential_datum</a>	m
<a href="#">sea_surface_height_above_mean_sea_level</a>	m
<a href="#">alias: sea_surface_height_above_sea_level</a>	m
<a href="#">alias: sea_surface_height</a>	m
<a href="#">sea_surface_height_above_reference_ellipsoid</a>	m
<a href="#">sea_surface_height_amplitude_due_to_earth_tide</a>	m
<a href="#">sea_surface_height_amplitude_due_to_equilibrium_ocean_tide</a>	m
<a href="#">sea_surface_height_amplitude_due_to_geocentric_ocean_tide</a>	m
<a href="#">sea_surface_height_amplitude_due_to_non_equilibrium_ocean_tide</a>	m
<a href="#">sea_surface_height_amplitude_due_to_pole_tide</a>	m
<a href="#">sea_surface_height_bias_due_to_sea_surface_roughness</a>	m
<a href="#">sea_surface_height_correction_due_to_air_pressure_and_wind_at_high_frequency</a>	m
<a href="#">sea_surface_height_correction_due_to_air_pressure_at_low_frequency</a>	m
<a href="#">square_of_sea_surface_height_above_geoid</a>	m <sup>2</sup>
<a href="#">steric_change_in_sea_surface_height</a>	m
<a href="#">tendency_of_sea_surface_height_above_mean_sea_level</a>	m year <sup>-1</sup>

- A key case of how the CF Conventions help us all to work together (and alone) is the **standard names**.
- Example: if you created or edited data describing sea surface height, there is ambiguity in what that exactly means - e.g. is it a change, amplitude or correction, and what constitutes the starting point for 'height', what effect is it encoding, etc.?
- *Note the CF Conventions are community-driven* and anyone can request changes or updates e.g. new standard names - please contribute/get involved if useful!



**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL



## 2. cf-python and cf-plot: general overview

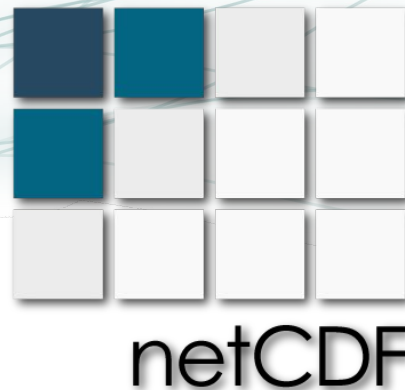
- Who is the target audience of these tools?
- What is a domain-specific library and why can we view cf-python as one?
- How do they compare to other, alternative, Python tools?
- A brief history/timeline
- What functionality does cf-python offer in general?
- What functionality does cf-plot offer in general?
- What is their development/maintenance status?

# NCAS CF Data Tools: aimed at netCDF & MO data users

Ultimately, cf-python and cf-plot (and friends) are **targeted/designed for use by anyone who wants to work with data in the format of netCDF or its human-readable CDL form, or Met Office file formats i.e. PP files & fields/UM files, however CF Compliant they may be.**

Given the focus on the CF Conventions, they are especially aimed at those using datasets related to Climate and Forecasting data (model, satellite, observational, etc.) for atmosphere, surface & ocean, etc.  
Notable clients:

- researchers (at any level, including students)
- Research Software Engineers
- data center workers
- data producers/managers
- and other technical/research professionals



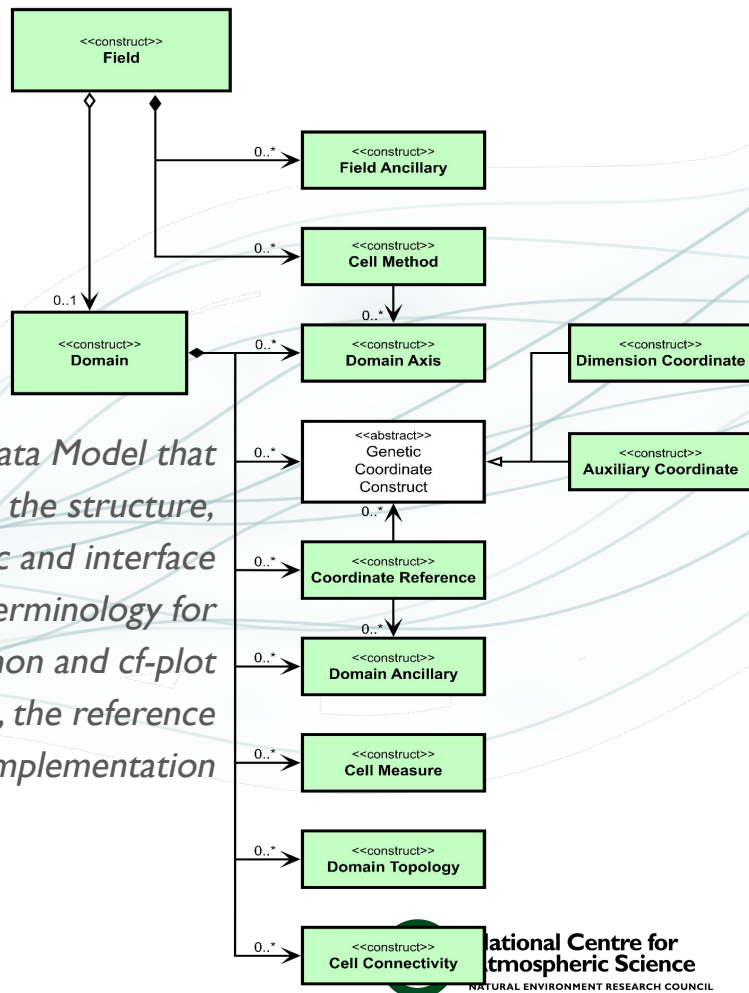
# cf-python: a *domain-specific* library

cf-python is a *domain-specific* package, the domain here being the CF-netCDF metadata conventions - as opposed to a general package designed without any domain in mind. **The CF Conventions have a formal data model and cf-python is built exactly on it.**

This means that cf-python:

- can deal with any conceivable CF-netCDF dataset (and non-compliant ones)
- provides you with a consistent, coherent and logical view of your datasets
- has functionality that tends to work whatever input you throw at it.

*The CF Data Model that underlies the structure, logic and interface terminology for cf-python and cf-plot (via cfdm, the reference implementation)*



# Alternative Python tools and comparison to these

Other tools (focusing on Python-based ones only) that you may use or be familiar with, that provide some overlap in terms of functionality are notably:

- **Iris**: developed by the Met Office, hence regular used by those at / working with the MO. Knows about CF & is domain-specific but doesn't use the formal data model, hence less so than cf-python
- **xarray and cf\_xarray**: a general tool which extends numpy (the fundamental package for scientific computing in Python) to notably introduce labelling for dimensions, coordinates and attributes. The cf\_xarray tool adds some CF Conventions awareness, but not much!
- **netcdf4-python**: a tool for working with netCDF without CF Conventions awareness

*Not at all  
domain-specific:  
No knowledge of  
CF-netCDF*



↑  
numpy



netCDF4-python

↑  
xarray



iris



↑  
cf-python

*Fully  
domain-specific:  
Complete knowledge  
of CF-netCDF*



National Centre for  
Atmospheric Science  
NATURAL ENVIRONMENT RESEARCH COUNCIL

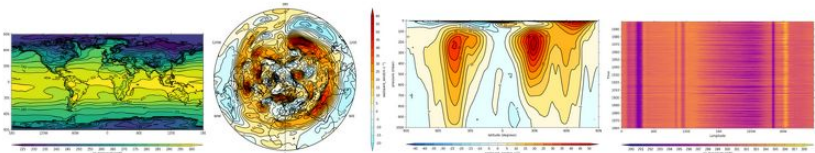


# cf-python (`import cf`): summary of functionality

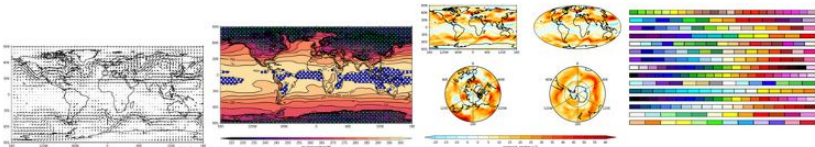
Some of the core features of cf-python, noting that it uses Dask for all of the array manipulation meaning performance is at the forefront and optimisable via Dask operations (i.e. fast and can be made faster for a specific application if desired):

- read and write out netCDF, CDL, PP and UM dataset(s) to/from disk
- create new datasets
- inspection: view the (meta)data at different detail levels
- edit the (meta)data
- aggregating datasets: combining multiple datasets together to form a single coherent dataset
- reduce datasets by subsampling and statistical collapsing
- analyse data: applying mathematical and statistical operations e.g. apply arithmetic, trigonometrical functions, differential operators etc.
- regridding: change the underlying grid of data, via various interpolation methods
- work with data that is compressed by convention whilst presenting in uncompressed form
- incorporate, and create, metadata stored in external files
- perform histogram, percentile and binning operations

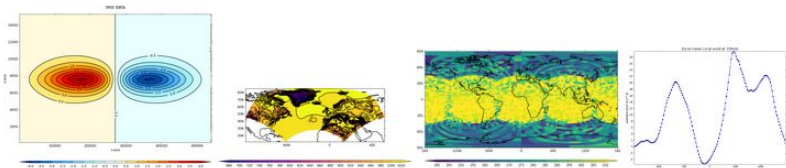
# cf-plot (`import cfplot`): summary of functionality



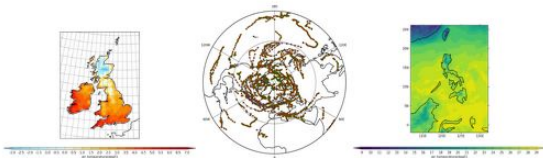
Cylindrical plots | Polar plots | Pressure/height plots | Hovmuller



Vector plots | Stipple plots | Multiple plots on a page | Colour scales



User defined axes | Rotated pole | Unstructured grids | Graphs



Projections | Trajectories | WRF data grids

cf-plot enables CF-aware geoscientific visualisation. It generally uses cf-python to present the data & CF attributes for plotting (though it can also use plain numpy arrays). In minimal code lines, you can get from your cf-python fields for instance:

- contour plots
- vector plots
- plots of trajectories
- significance plots
- line plots

on various map projections, with customisable axes, colour scales, figure elements etc.

# Brief history/timeline of cf-python and cf-plot

The last 14 years of NCAS-supported cf-python development have been towards providing infrastructure for Earth system science research at NCAS (and the wider community):

**2011** Jonathan Gregory instigated a new software library primarily for netCDF data, to replace the ailing UKMO PP format and its IDL library.

**2011 version 0.9** (first ever version released) – Not really unusable, more a proof of concept.

**2015 version 1.0** (first useful release) – Re-designed API, read, write, aggregate, subsampling, assignment, where, trigonometric operations, statistical collapses, read PP/UM data.

**2017 version 2.0** – Introduction of cf-plot for visualization, regridding, convolution filters, Discrete Sampling Geometries (DSG).

**2019 version 3.0.0** – Python 3, redesigned API, lots of bug fixes and minor functionality enhancements.

**2025 version 3.16.2** (really getting somewhere) – Dask parallelism, differential operators, percentiles, histograms, binning, digitisation, moving windows, halos, computation of vertical coordinates, unstructured grids (UGRID).

# Development/maintenance status of cf-python & cf-plot

We can summarise the status of the libraries now (early 2025) as follows:

- (As indicated in the timeline) cf-python is very mature and has been actively and well-maintained for 14 years (regular releases), with any found bugs usually quickly resolved for the next release. It is stable (any API changes are minimal and well-documented).
- cf-plot also has a history of several years worth of active development, so is fairly mature...
- ...but due to the retirement of its original and lone core developer in early 2024 (I have taken over as lone core maintainer but only have 5% time until March 25+ to work on it), cf-plot has been through a year of less regular releases and lots of updates e.g. new documentation, creation of a test suite identifying numerous known bugs.
- **[Important note]** A new release will be out with a regular schedule from March 25+ but until spring/summer, there are lots of known bugs in the process of being fixed (& more to identify). In short: **cf-plot is undergoing a much-needed revamp 2024-2025 (but is still usable)**

# Some example code: small snippet with annotations

```
>>> import cf
>>> f = cf.read('de814a_1_mon_m01s30i204.nc')[0]
>>> print(f)
Field: air_temperature (ncvar%m01s30i204)
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(325), longitude(432)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360_day
                : air_pressure(36) = [1000.0, ..., 0.029999999329447746] hPa
                : latitude(325) = [-90.0, ..., 90.0] degrees_north
                : longitude(432) = [0.0, ..., 359.1666564941406] degrees_east
Cell measures  : measure:area (external variable: ncvar%areacella)
```

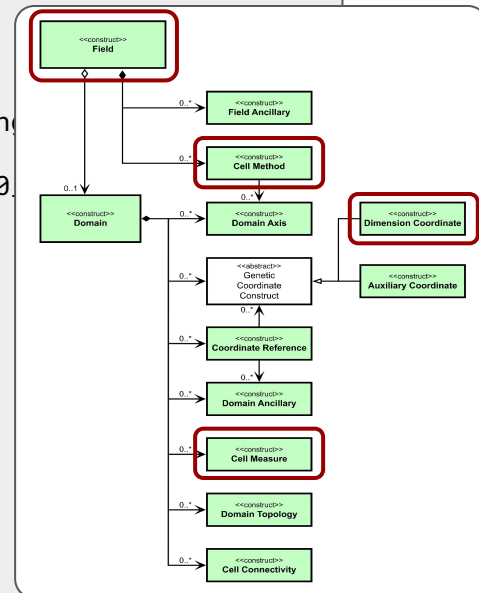


# Some example code

```
>>> import cf
>>> f = cf.read('de814a_1_mon_m01s30i204.nc')[0]
>>> print(f)
```

Field: air\_temperature (ncvar%m01s30i204)

Field	air_temperature(time(12), air_pressure(36), latitude(325), longitude(432))
Data	area: mean time(12): mean (interval: 6 h)
Cell methods	
Dimension coords	time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360
	air_pressure(36) = [1000.0, ..., 0.0299999999329447746] hPa
	latitude(325) = [-90.0, ..., 90.0] degrees_north
	longitude(432) = [0.0, ..., 359.1666564941406] degrees_east
Cell measures	measure:area (external variable: ncvar%areacella)



# Some example code

```
>>> import cf
>>> f = cf.read('de814a_1_mon_m01s30i204.nc')[0]
>>> print(f)
Field: air_temperature (ncvar%m01s30i204)
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(325), longitude(432)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360_day
                  air_pressure(36) = [1000.0, ..., 0.029999999329447746] hPa
                  latitude(325) = [-90.0, ..., 90.0] degrees_north
                  longitude(432) = [0.0, ..., 359.1666564941406] degrees_east
Cell measures   : measure:area (external variable: ncvar%areacella)

>>> f_nino3 = f.subspace(longitude=cf.wi(-150, -90), latitude=cf.wi(-5, 5))
>>> print(f_nino3)
Field: air_temperature (ncvar%m01s30i204)
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(19), longitude(73)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360_day
                  air_pressure(36) = [1000.0, ..., 0.029999999329447746] hPa
                  latitude(19) = [-5.0, ..., 5.0] degrees_north
                  longitude(73) = [210.0, ..., 270.0] degrees_east
Cell measures   : measure:area (external variable: ncvar%areacella)
```

# Some example code

```
>>> import cf
>>> f = cf.read('de814a_1_mon_m01s30i204.nc')[0]
>>> print(f)
Field: air_temperature (ncvar%m01s30i204)
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(325), longitude(432)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360_day
                  air_pressure(36) = [1000.0, ..., 0.029999999329447746] hPa
                  latitude(325) = [-90.0, ..., 90.0] degrees_north
                  longitude(432) = [0.0, ..., 359.1666564941406] degrees_east
Cell measures   : measure:area (external variable: ncvar%areacella)

>>> f_nino3 = f.subspace(longitude=cf.wi(-150, -90), latitude=cf.wi(-5, 5))
>>> print(f_nino3)
Field: air_temperature (ncvar%m01s30i204)
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(19), longitude(73)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00] 360_day
                  air_pressure(36) = [1000.0, ..., 0.029999999329447746] hPa
                  latitude(19) = [-5.0, ..., 5.0] degrees_north
                  longitude(73) = [210.0, ..., 270.0] degrees_east
Cell measures   : measure:area (external variable: ncvar%areacella)
```

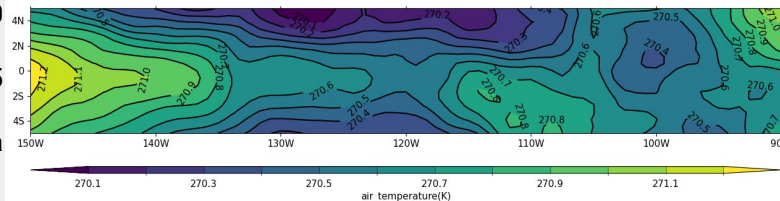
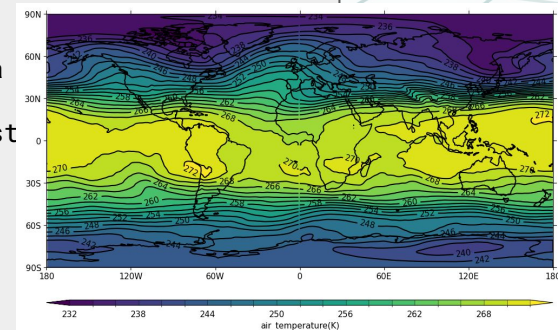
# Some example code

```
>>> import cf, cfplot
>>> f = cf.read('de814a_1_mon_m01s30i204.nc')[0]
>>> print(f)
Field: air_temperature (ncvar%m01s30i204)
```

```
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(325), longitude(432)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00]
                  air_pressure(36) = [1000.0, ..., 0.0299999999329447746] hPa
                  latitude(325) = [-90.0, ..., 90.0] degrees_north
                  longitude(432) = [0.0, ..., 359.1666564941406] degrees_east
cell measures   : measure:area (external variable: ncvar%areacella)
>>> cfplot.con(f.subspace(T=[0], Z=500))
```

```
>>> f_nino3 = f.subspace(longitude=cf.wi(-150, -90), latitude=cf.wi(-5, 5))
>>> print(f_nino3)
Field: air_temperature (ncvar%m01s30i204)
```

```
-----
Data          : air_temperature(time(12), air_pressure(36), latitude(19), longitude(73)) K
Cell methods   : area: mean time(12): mean (interval: 6 h)
Dimension coords: time(12) = [2037-01-16 00:00:00, ..., 2037-12-16 00:00:00]
                  air_pressure(36) = [1000.0, ..., 0.0299999999329447746] hPa
                  latitude(19) = [-5.0, ..., 5.0] degrees_north
                  longitude(73) = [210.0, ..., 290.0] degrees_east
cell measures   : measure:area (external variable: ncvar%areacella)
>>> cfplot.con(f_nino3.subspace(T=[0], Z=500))
```



# Some example code

Specific example: aggregation of datasets to combine data in an intelligent, metadata-aware way.

Aggregations can be saved from cf-python with almost no extra disk space, as the data arrays are not copied from the original files, rather those arrays are “pointed to” from the aggregation file.

```
>>> f = cf.read('ATM/203[7-9]/de814a*.nc', aggregate={'relaxed_identities': True})
>>> print(f)
[<CF Field: surface_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: surface_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: eastward_wind(time(36), latitude(324), longitude(432)) m s-1>,
<CF Field: northward_wind(time(36), latitude(325), longitude(432)) m s-1>,
<CF Field: air_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: precipitation_flux(time(36), latitude(324), longitude(432)) kg m-2 s-1>,
<CF Field: tendency_of_eastward_wind_due_to_nonorographic_gravity_wave_drag(time(36), air_pressure(36), latitude(325), longitude(1)) m s-2>,
<CF Field: tendency_of_eastward_wind_due_to_orographic_gravity_wave_drag(time(36), air_pressure(36), latitude(325), longitude(1)) m s-2>,
<CF Field: eastward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: eastward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: northward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: northward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: air_temperature(time(36), air_pressure(36), latitude(325), longitude(432)) K>,
<CF Field: air_temperature(time(36), air_pressure(36), latitude(325), longitude(1)) K>,
<CF Field: northward_transformed_eulerian_mean_air_velocity(time(36), air_pressure(36), latitude(325)) m s-1>,
<CF Field: long_name=RESIDUAL MN MERID. CIRC. WSTARBAR(time(36), air_pressure(36), latitude(325)) m s-1>,
<CF Field: northward_eliasen_palm_flux_in_air(time(36), air_pressure(36), latitude(325)) m3 s-2>,
<CF Field: upward_eliasen_palm_flux_in_air(time(36), air_pressure(36), latitude(325)) m3 s-2>,
<CF Field: tendency_of_eastward_wind_due_to_eliasen_palm_flux_divergence(time(36), air_pressure(36), latitude(325)) m s-2>,
<CF Field: long_name=MERIDIONAL HEAT FLUX(time(36), air_pressure(36), latitude(325))>,
<CF Field: long_name=MERIDIONAL MOMENTUM FLUX(time(36), air_pressure(36), latitude(325))>]
```

```
>>> cf.write(f, 'atmos_2037-2039.nc', cfa='field') # ← creates atmos_2037-2039.nc as a 2.3 KB file, original data is 1.4 GB
>>> g = cf.read('atmos_2037-2039.nc') # ← no need for any aggregation configuration any more!
>>> print(g)
[<CF Field: surface_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: surface_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: eastward_wind(time(36), latitude(324), longitude(432)) m s-1>,
<CF Field: northward_wind(time(36), latitude(325), longitude(432)) m s-1>,
<CF Field: air_temperature(time(36), latitude(324), longitude(432)) K>,
<CF Field: precipitation_flux(time(36), latitude(324), longitude(432)) kg m-2 s-1>,
<CF Field: tendency_of_eastward_wind_due_to_nonorographic_gravity_wave_drag(time(36), air_pressure(36), latitude(325), longitude(1)) m s-2>,
<CF Field: tendency_of_eastward_wind_due_to_orographic_gravity_wave_drag(time(36), air_pressure(36), latitude(325), longitude(1)) m s-2>,
<CF Field: eastward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: eastward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: northward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: northward_wind(time(36), air_pressure(36), latitude(325), longitude(1)) m s-1>,
<CF Field: air_temperature(time(36), air_pressure(36), latitude(325), longitude(432)) K>,
<CF Field: air_temperature(time(36), air_pressure(36), latitude(325), longitude(1)) K>,
<CF Field: northward_transformed_eulerian_mean_air_velocity(time(36), air_pressure(36), latitude(325)) m s-1>,
<CF Field: long_name=RESIDUAL MN MERID. CIRC. WSTARBAR(time(36), air_pressure(36), latitude(325)) m s-1>,
<CF Field: northward_eliasen_palm_flux_in_air(time(36), air_pressure(36), latitude(325)) m3 s-2>,
<CF Field: upward_eliasen_palm_flux_in_air(time(36), air_pressure(36), latitude(325)) m3 s-2>,
<CF Field: tendency_of_eastward_wind_due_to_eliasen_palm_flux_divergence(time(36), air_pressure(36), latitude(325)) m s-2>,
<CF Field: long_name=MERIDIONAL HEAT FLUX(time(36), air_pressure(36), latitude(325))>,
<CF Field: long_name=MERIDIONAL MOMENTUM FLUX(time(36), air_pressure(36), latitude(325))>]
```





### 3. cf-python and cf-plot: for AFESP researchers

- What can cf-python and cf-plot offer for you, researchers and students in AFESP?
- User support we provide to cater to your specific needs
- How to let us know about features that you'd like added, etc.



# Possible use of cf-python & cf-plot to aid AFESP research

**[Pause for brief summary of a workflow for a volunteer or two!]**

I don't know all that much about AFESP research, so it would be nice to hear thoughts given the talk so far, but in general:

- cf-python can help you to work with data in the netCDF (CF-netCDF or otherwise), netCDF, CDL, PP and UM (fields file) formats in a way which is domain-specific, metadata-aware and generally performant (i.e. fast). If you need your application code to be more direct, metadata-aware or quicker (relative to the alternative tools), cf-python could help
- cf-plot could be useful too if you want to do any visualisations with the above data

# In-house user support: how we can help you

cf-python and cf-plot are developed and maintained by David Hassell (also NCAS-CMS, UoR) and I.

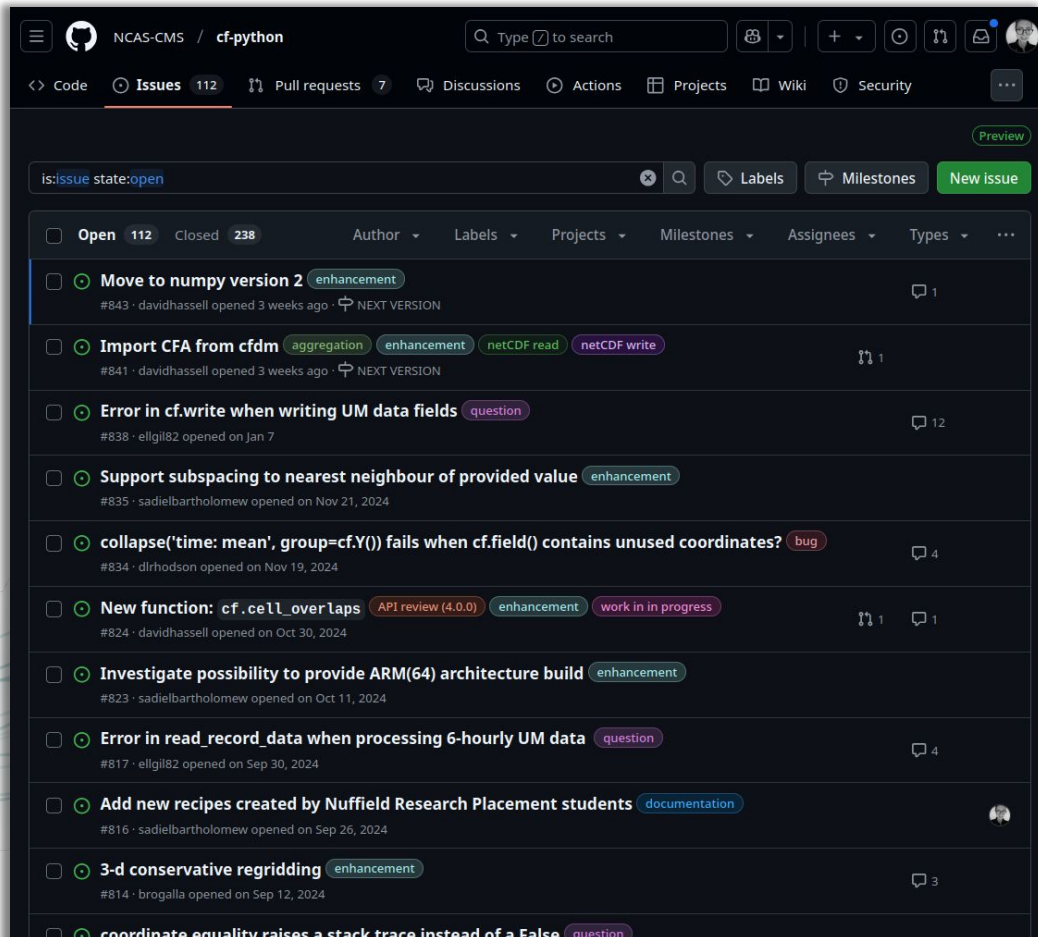
If you are struggling to work out how to do a particular thing, or want guidance or have questions or requests in general, we are here to provide support. You can get in touch in various ways below.

If there is functionality we don't have that would be useful to you, we take requests for enhancements and might be able to provide this for you (depending/balancing with priorities).

Means to contact us for the above (note the Issue Trackers are preferred, but any means is fine):

- cf-python codebase Issue Tracker: <https://github.com/NCAS-CMS/cf-python/issues>
- cf-python codebase Issue Tracker: <https://github.com/NCAS-CMS/cf-plot>
- email, mine for direct contact is: [sadie.bartholomew@ncas.ac.uk](mailto:sadie.bartholomew@ncas.ac.uk)
- the NCAS-CMS Helpdesk, including to ask more general Qs: <https://cms-helpdesk.ncas.ac.uk/>

# Please let us know if you'd like guidance or new features!



- Shown is the cf-python Issue Tracker for the codebase which lives on GitHub - cf-plot has the equivalent
- As well as noting the work we developers/maintainers are doing, **users (/anyone) are welcome and encouraged to raise questions, requests, comments, note possible bugs, etc.**



## 4. Conclusion

- Summary of the talk
- Collection of relevant links



# Summary

- **NCAS-CMS** are a sub-group of NCAS, mostly based here at Reading, who provide HPC resource management, training and software engineering support for the UK atmospheric and polar science community - *in short, we are here to support you in your research*
- The **NCAS CF Data Tools** are a set of Python libraries developed by NCAS-CMS which are based around the **CF Conventions**, a *metadata standard* used throughout the geoscience e.g. Meteorology communities that *makes it easier to share, use and understand our datasets*.
- **cf-python** and **cf-plot** are the main user-facing libraries in the set of NCAS CF Data Tools
- **cf-python** is a geoscientific data analysis library for use with netCDF and Met Office data formats. It is *domain-specific*, so more tailored to meteorological application than, say, xarray
- **cf-plot** provides geoscientific data visualisation i.e. plotting capability of cf-python fields
- Python tools with similar functionality include Iris, xarray and netcdf4-python (with matplotlib + Cartopy for plotting). Use whichever tool(s) is/are best for you! But note that there is in-house support here for use of cf-python and cf-plot - *we can provide guidance and new features upon request with the aim to help you to achieve what you want to, code- and data-wise*
- To explore the libraries, see our documentation and training resources (⇒)



# Links that may be useful

Our documentation and/or training are generally the *best way to get familiar with the libraries*:

- cf-python documentation: <https://ncas-cms.github.io/cf-python>
- cf-plot documentation: <https://ncas-cms.github.io/cf-plot/build>
- NCAS CF Data Tools training repository with Notebooks and set up advice (use 'new course'):  
[https://github.com/NCAS-CMS/cf-tools-training/tree/master/new\\_course](https://github.com/NCAS-CMS/cf-tools-training/tree/master/new_course)
- The CF Conventions official website, for background: <https://cfconventions.org/>

*For guidance or support*, use our Issue Trackers to ask questions to report issues etc., or the CMS Helpdesk, or you can email me directly if you prefer at [sadie.bartholomew@ncas.ac.uk](mailto:sadie.bartholomew@ncas.ac.uk):

- cf-python codebase including Issue Tracker: [https://github.com/NCAS-CMS/cf-python\[/issues\]](https://github.com/NCAS-CMS/cf-python[/issues])
- cf-python codebase including Issue Tracker: [https://github.com/NCAS-CMS/cf-plot\[/issues\]](https://github.com/NCAS-CMS/cf-plot[/issues])
- NCAS-CMS Helpdesk, alternative means to ask us questions: <https://cms-helpdesk.ncas.ac.uk/>



## 5. Extra



**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL