Xarray Climate Data Analysis Tools

A Python Package for Simple and Robust Analysis of Climate Data

2024 EESM PI Meeting (08/08/2024)

Tom Vo, Stephen Po-Chedley, Jason Boutte, Jiwoo Lee, and Jill Zhang

With thanks to Peter Gleckler, Paul Durack, Karl Taylor, and Chris Golaz

This work is performed under the auspices of the U. S. DOE by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344.
LLNL-PRES-867521

Thank you to our project funders at U.S. DEPARTMENT OF ENERGY | Office of Science

BER *(Office of Biological Environmental Research)*
EESSD *(Earth and Environmental Systems Sciences Division)*

PM: Renu Joseph

RGMA
Regional & Global Model Analysis

SEATS

PCMDI
Program for Climate Model Diagnosis and Intercomparison

PM: Xujing Davis

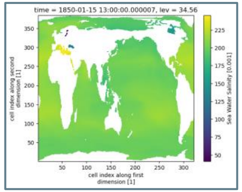
ESMD
Earth System Model Development

E3SM
Energy Exascale Earth System Model

An Overview of this xCDAT Talk

1. Driving forces
2. Principle goals
3. Key features
4. How to started and get involved
5. What’s in store for xCDAT?

The Driving Forces Behind xCDAT



1. **Growing volume of climate data**
    a. A larger pool of data products
    b. Increasing spatiotemporal resolution of model and observational data
2. **Data analysis requires highly performant, core operations**
    a. Reading and writing netCDF files
    b. Regridding
    c. Spatial and temporal averaging
3. **CDAT (Community Data Analysis Tools) library** is end-of-life since Dec/2023
    a. Provided climate data analysis and visualization packages for over 20 years
    b. Many users and software packages *(e.g., E3SM Diagnostics, PCMDI Metrics)* depend on CDAT

netCDF

CDAT
Community Data Analysis Tools

xCDAT addresses these challenges by....

Combining the power of **Xarray** with geospatial analysis features inspired by **CDAT**

xCDAT = xarray + CDAT

Community Data Analysis Tools

xCDAT: Xarray Climate Data Analysis Tools

- xCDAT is an extension of Xarray in the climate science domain
- Scope focused on routine climate data analysis operations on structured grids
- Leverages and/or extends the capabilities other powerful Xarray-based packages such as xESMF, xgcm, and CF-xarray
- Developed by team of climate scientists and software engineers from Lawrence Livermore National Laboratory

What are the principle goals of xCDAT?

- **Use modern technologies (Xarray, Dask)**
  - Capable of handling large datasets
  - Lazy operations, parallelism
- **Similar core capabilities to CDAT**
  - e.g., spatial averaging, temporal averaging, regridding
- **Promote software sustainability and reproducible science**
  - Maintainable, extensible, and easy-to-use
  - Emphasize Climate and Forecast (CF) Metadata Conventions
- **Foster open-source community**
  - Serve the needs of the climate community in the long-term
  - Community engagement efforts (e.g., Pangeo, ESGF)

xarray : N-D labeled arrays and datasets in Python

Why is Xarray the core technology of xCDAT?

- **Modern, mature, and widely adopted**
- **Stable** – funding from NumFocus
- **Introduces labels** for dimensions, coordinates, and attributes on top of raw NumPy-like arrays
- **User experience** is intuitive, concise, less error-prone (compared to raw NumPy)

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Key features of Xarray

- **File I/O** – netCDF, Iris, OPeNDAP, Zarr, more
- **Array manipulation** – Indexing, selecting, interpolating, grouping, aggregating, parallelism via Dask, plotting
- **Interoperable** with scientific Python ecosystem

NumPy pandas
matplotlib DASK

xCDAT

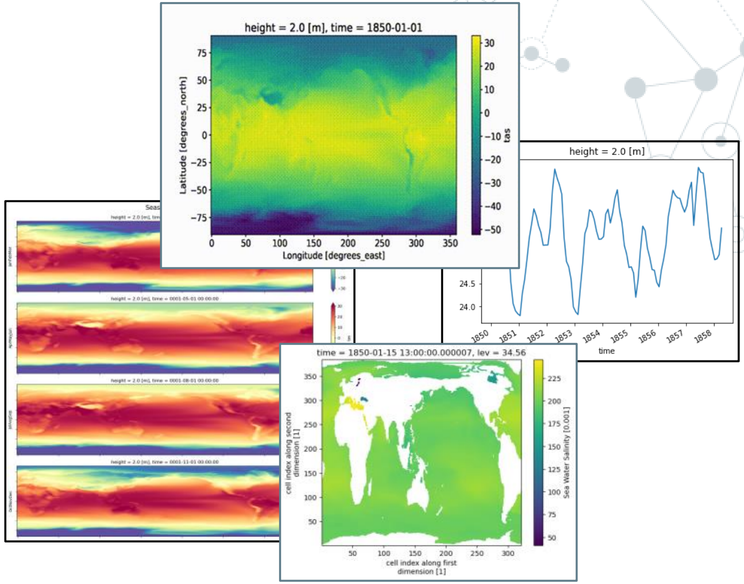
tools for simple and robust analysis code

I/O and Metadata

- Xarray dataset I/O with post-processing options
  - Generate missing bounds, center time coords, convert lon axis orientation
- Robust handling of coordinate bounds
- Interpret Climate and Forecast (CF) compliant metadata *(via cf-xarray)*

Computations

- Spatial averaging
- Temporal averaging, climatologies, departures
- Horizontal regridding (*extension of xESMF and Python port of Regrid2)*
- Vertical regridding (*extension of xgcm)*



Parallelizable through Xarray's support for DASK

How to use xCDAT

xCDAT extends Xarray Dataset objects via “accessor” classes.

`dataset.spatial.average()`

object — *accessor* — *method*

Accessors classes include:

- spatial – .average, .get_weights
- temporal – .average, .group_average, .climatology, .departures
- regridding – horizontal, vertical
- bounds – .get_bounds, .add_bounds, .add_missing_bounds

Functions include:

- open_dataset, open_mfdataset
- center_times, decode_time
- swap_lon_axis
- create_axis
- create_grid
- get_dim_coords
- get_dim_keys

Visit the API Reference page for a complete list: https://xcdat.readthedocs.io/en/latest/api.html

xCDAT simplifies Xarray code for specific operations

Example: calculate global-mean, weighted monthly anomalies

- Less code
- More flexible
- Easier to read/write

```python
import numpy as np
import xarray as xr

# 1. Open the dataset.
dpath = (
    "/p/user_pub/work/CMIP6/CMIP/E3SM-Project/"
    "E3SM-2-0/historical/r1i1p1f1/Amon/ts/gr/v20220830/"
)
ds = xr.open_mfdataset(dpath + "*.nc")

# 2. Calculate monthly departures.
ts_monthly = ds.ts.groupby("time.month")  # group by months
ts_monthly_clim = ts_monthly.mean(dim="time")  # calculate climatology
ts_anom = ts_monthly - ts_monthly_clim  # difference to determine anomalies

# 3. Compute global average.
coslat = np.cos(np.deg2rad(ds.lat))
ts_anom_weighted = ts_anom.weighted(coslat)
ts_anom_global = ts_anom_weighted.mean(dim="lat").mean(dim="lon")

# 4. Calculate annual averages
# Source: https://ncar.github.io/esds/posts/2021/yearly-averages-xarray/
month_len = ts_anom_global.time.dt.days_in_month
month_len_by_year = month_len.groupby("time.year")
wgts = month_len_by_year / month_len_by_year.sum()
temp_sum = (ts_anom_global * wgts).resample(time="AS").sum(dim="time")
denominator_sum = (wgts).resample(time="AS").sum(dim="time")
ts_anom_global_ann = temp_sum / denominator_sum
```

```python
import xcdat as xc

# 1. Open the dataset.
dpath = (
    "/p/user_pub/work/CMIP6/CMIP/E3SM-Project/"
    "E3SM-2-0/historical/r1i1p1f1/Amon/ts/gr/v20220830/"
)
ds = xc.open_mfdataset(dpath)

# 2. Calculate monthly departures.
ds_anom = ds.temporal.departures("ts", freq="month")

# 3. Compute global average.
ds_anom_global = ds_anom.spatial.average("ts")

# 4. Calculate annual averages
ds_anom_global_ann = ds_anom_global.temporal.group_average("ts", freq="year")
```

xCDAT's Growing Adoption in the Scientific Community

- **16,000+ total downloads*** on Anaconda
- **100+ stars*** on GitHub
- **Global usage** in various projects and organizations
  - **LLNL** (Lawrence Livermore National Lab)
  - **NASA** (National Aeronautics and Space Administration)
  - **IPSL** (Institut Pierre-Simon Laplace)
- **Data processing engine** for PCMDI Metrics Package and E3SM Diagnostics Package
- **Post-processing and analysis tool** in E3SM Unified Environment

PCMDI
Program for Climate Model Diagnosis and Intercomparison

E3SM
Energy Exascale Earth System Model

E3SM UNIFIED ENVIRONMENT
CONDA®

*** As of Aug 2024***

Get Involved in xCDAT!

xCDAT is distributed via Anaconda

ANACONDA CONDA-FORGE

Any and all contribution is welcome!

- Code
- Documentation
- Submit and/or address tickets
- Forum discussions

Read the Docs

GitHub

What’s in store for xCDAT?

- **Collaborate with UXarray** for interoperation to support end-to-end and more streamlined operation on unstructured (i.e. E3SM native output) datasets.
- **Continue assisting integration in DOE funded projects** including PCMDI Metrics Package, E3SM Diags
- **Explore other DOE funded projects** to integrate xCDAT for analysis capabilities

Recap

- xCDAT is an **extension of Xarray for climate data analysis on structured grids**, a modern successor to the Community Data Analysis Tools (CDAT) library
- Focused on **routine climate research analysis operations**, such as temporal averaging, spatial averaging, and regridding.
- Designed to promote **software sustainability** and **reproducible science**
- **Parallelizable** through Xarray's support for Dask

Supplemental Slides

The Software Design Philosophy of xCDAT

- **Encourage software sustainability** and **reproducible science**
- **Well-documented and configurable features** allow scientists to rapidly develop robust, reusable, less-error prone, more maintainable code
- **Contribute to Pangeo's** effort of fostering an ecosystem of mutually compatible geoscience Python packages

Read the Docs

PANGEO

Code Example: Calculate Geospatial Weighted Average

Xarray

```
import xarray as xr
import numpy as np

# 1. Open the dataset
path = "input/tas_3hr_ACCESS-ESM1-5_historical_r10i1p1f1_gn_201001010300-201501010000.nc"
ds = xr.open_dataset(path)

# 2. Convert air temperature to Celsius
ds["tas"] = ds["tas"] - 273.15

# 3. Calculate weights and apply to data
weights = np.cos(np.deg2rad(ds["tas"].lat))
weights.name = "weights"
tas_weighted = ds["tas"].weighted(weights)

# 4. Calculate the weighted mean
weighted_mean = tas_weighted.mean(("lon", "lat"))
```

xCDAT

```
import xcdat as xc

# 1. Open the dataset
path = "input/tas_3hr_ACCESS-ESM1-5_historical_r10i1p1f1_gn_201001010300-201501010000.nc"
ds = xc.open_dataset(path)

# 2. Convert air temperature to Celsius
ds["tas"] = ds["tas"] - 273.15

# 3. Calculate the weighted mean
weighted_mean = ds.spatial.average("tas", axis=["X", "Y"], keep_weights=True)["tas"]
```

***Note**: xcdat does a lot more, including handling regional averages*

Code Example: Calculate Monthly Temperature Departures

Xarray

```python
import xarray as xr

# 1. Open the dataset
path = "ts_Amon_ACCESS1-0_historical_r1i1p1_185001-200512.nc"
ds = xr.open_dataset(path)

# 2. Calculate the weights
month_len = ds.time.dt.days_in_month
weights = (
   month_len.groupby("time.month") /
month_len.groupby("time.month").sum()
)

# 2. Calculate the monthly climatology
ts_climo = (ds["ts"] *
weights).groupby("time.month").sum(dim="time")

# 3. Calculate the monthly anomalies
ts_anomalies = ds["ts"].groupby("time.month") - ts_climo
```

xCDAT

```python
import xcdat as xc

# 1. Open the dataset
path = "ts_Amon_ACCESS1-0_historical_r1i1p1_185001-200512.nc"
ds = xc.open_dataset(path)

# 2. Calculate the monthly anomalies
# Note, we extract “ts” from the xr.Dataset object with
[“ts”]
ts_anomalies = ds.temporal.departures("ts", freq="month",
weighted=True)["ts"]
```

Parallelism with xarray DASK +

- **Why does Xarray integrate with Dask?**
  - For datasets that don’t fit into memory, support parallel computations and streaming computation
  - Dask is an optional feature, but might become a required dependency

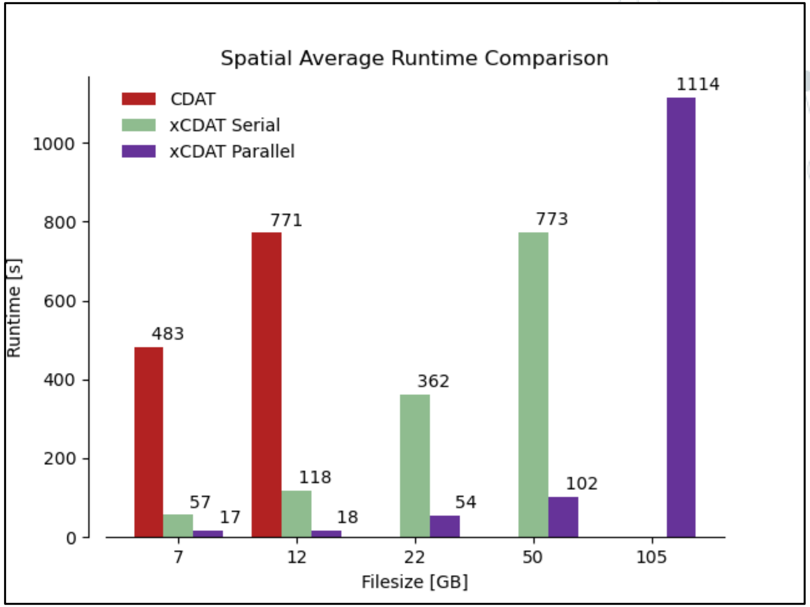
  — https://docs.xarray.dev/en/stable/use
- **Which Xarray features support Dask?**
  - Nearly all existing xarray methods have been extended to work automatically with Dask arrays
  - Indexing, computation, concatenating and grouped operations

  — https://docs.xarray.dev/en/stable/user-guide/dask.html#using-dask-with-xarray
- **What is the default Dask behavior for distributing work on compute hardware?**
  - By default, dask uses its multi-threaded scheduler distributes work across multiple cores and allows for processing some datasets that do not fit into memory
  - Optionally, setup the distributed scheduler for running across a cluster

  — https://docs.xarray.dev/en/stable/user-guide/dask.html#using-dask-with-xarray
- **How do xCDAT APIs work with Dask?**
  - Many core xCDAT APIs inherit Xarray's Dask support by operating on xarray.Dataset objects and making calls to parallelized Xarray APIs.
  - xCDAT users just need chunk the xarray.Dataset object before calling any of the parallelizable xCDAT APIs.

xCDAT is parallelizable through Xarray and Dask

- Most Xarray methods extended to work automatically with Dask arrays
  - *e.g., indexing, computation, concatenating and grouped operations*
- xCDAT inherits Xarray's support for parallelism



xCDAT outperforms the older CDAT library by much large margins in some cases, such as global spatial averaging

How do I activate Dask with Xarray/xCDAT?

1. Load the data from a netCDF file or files with open_dataset() or open_mfdataset()
2. Specify the chunks argument
    a. **DISCLAIMER:** open_mfdataset() will chunk each netCDF file into a single Dask array by default, so it is important set the chunks argument if the dataset is large

Note, Xarray maintains a Dask array until it is not possible. It will raise an exception instead of implicitly loading the array into memory.

```
import xarray as xr

filepath = "http://esgf.nci.org.au/thredds/dodsC/master/CMIP6/CMIP/CSIRO/ACCESS-ESM1-5/historical/r10i1p1f1/Amon/tas/gn/v20200605/tas_Amon_ACCESS-ESM1-5_historical_r10i1p1f1_gn_185001-201412.nc"

ds = xr.open_mfdataset(filepath, chunks={"time": "10"})

tas_daily = ds.tas.groupby(ds.time.dt.day).mean()
```