doppyo Documentation

Dougie Squire

Apr 02, 2019

CONTENTS:

CHAPTER

ONE

WHAT IS DOPPYO?

doppyo is a diagnostics/verification package for climate forecast systems. It is still in the early stages of development

Three modules are currently available:

1. `skill` : functions for assessing one data set relative to another, usually model output(s) relative to observations or reanalyses
2. `diagnostic` : functions for computing various atmospheric and oceanic diagnostics
3. `utils` : general support and utility functions for the doppyo package

CHAPTER TWO

INSTALLATION

Need to host doppyo on public repo and provide instructions for installation

CHAPTER

THREE

MODULES

3.1 diagnostic

doppyo functions for computing various ocean, atmosphere, & climate diagnostics

3.1.1 API

diagnostic.**velocity_potential**(*u*, *v*, *lat_name=None*, *lon_name=None*)

Returns the velocity potential given fields of u and v

Author: Dougie Squire

Date: 11/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

velocity_potential [xarray DataArray] Array containing values of velocity potential

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian.

These calculations are not yet dask-compatible.

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> v = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> doppyo.diagnostic.velocity_potential(u, v)
<xarray.DataArray 'velocity_potential' (lat: 6, lon: 4)>
array([[  431486.75 ,   431486.75 ,   431486.75 ,   431486.75 ],
       [ -240990.94 , -3553409.   ,  -970673.56 ,  2341744.5  ],
       [ 3338223.5  ,  1497203.9  , -1723363.2  ,   117656.31 ],
       [ 1009613.5  ,  1571693.6  ,   326689.3  ,  -235390.69 ],
       [ -931064.8  ,  -124736.375, -2516887.8  , -3323216.   ],
       [-1526244.   , -1526244.   , -1526244.   , -1526244.   ]], dtype=float32)
Coordinates:
  * lat      (lat) int64 75 45 15 -15 -45 -75
  * lon      (lon) int64 45 135 225 315
Attributes:
    units:          m**2 s**-1
    standard_name:  atmosphere_horizontal_velocity_potential
    long_name:      velocity potential
```

diagnostic.**stream_function**(*u*, *v*, *lat_name=None*, *lon_name=None*)

Returns the stream function given fields of u and v

Author: Dougie Squire
Date: 11/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

stream_function [xarray DataArray] Array containing values of stream function

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian. These calculations are not yet dask-compatible.

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> v = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> doppyo.diagnostic.stream_function(u, v)
<xarray.DataArray 'psi' (lat: 6, lon: 4)>
array([[ -690643.6 ,  -690643.6 ,  -690643.6 ,  -690643.6 ],
       [-2041977.8 , -1060127.  , -3052305.8 , -4034156.5 ],
       [ 4112389.2 ,  4630193.5 , -5212595.5 , -5730399.5 ],
       [  528500.75,  4670647.5 ,  2589393.  , -1552753.9 ],
       [-2686391.2 ,  -707369.25,  4204334.  ,  2225311.5 ],
       [ 1703481.9 ,  1703481.9 ,  1703481.9 ,  1703481.9 ]], dtype=float32)
Coordinates:
  * lat      (lat) int64 75 45 15 -15 -45 -75
  * lon      (lon) int64 45 135 225 315
Attributes:
    units:          m**2 s**-1
    standard_name:  atmosphere_horizontal_streamfunction
    long_name:      streamfunction
```

diagnostic.**Rossby_wave_source**(*u*, *v*, *lat_name=None*, *lon_name=None*)

Returns the Rossby wave source given fields of u and v

Author: Dougie Squire
Date: 11/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

Rossby_wave_source [xarray DataArray] Array containing values of Rossby wave source

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian.

These calculations are not yet dask-compatible.

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> v = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> doppyo.diagnostic.Rossby_wave_source(u, v)
<xarray.DataArray 'rws' (lat: 6, lon: 4)>
array([[ 4.382918,  4.382918,  4.382918,  4.382918],
       [-2.226769,  5.020311, -2.600087, -9.838818],
       [ 2.1693  , -2.133569,  0.498156,  4.818402],
       [-1.404836,  0.192032,  0.112654, -1.494616],
       [-0.103261,  4.518184,  0.648616, -4.05276 ],
       [ 4.070806,  4.070806,  4.070806,  4.070806]])
Coordinates:
  * lat      (lat) int64 75 45 15 -15 -45 -75
  * lon      (lon) int64 45 135 225 315
Attributes:
    units:     1e-11/s^2
    long_name:  Rossby wave source
```

`diagnostic.`**`divergent`**(*u*, *v*, *lat_name=None*, *lon_name=None*)

Returns the irrotational (divergent) component of u and v

Author: Dougie Squire

Date: 11/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

divergent [xarray Dataset]

Dataset containing the following variables:

u_chi; array containing the irrotational component of u

v_chi; array containing the irrotational component of v

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian.

These calculations are not yet dask-compatible.

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> v = xr.DataArray(np.random.normal(size=(6,4)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90))])
>>> doppyo.diagnostic.divergent(u, v)
<xarray.Dataset>
Dimensions:  (lat: 6, lon: 4)
Coordinates:
  * lat      (lat) int64 75 45 15 -15 -45 -75
  * lon      (lon) int64 45 135 225 315
Data variables:
    u_chi     (lat, lon) float32 0.5355302 -0.45865965 ... -0.7270669 -0.64930713
    v_chi     (lat, lon) float32 -0.45865965 -0.5355302 ... 0.64930713 -0.7270669
```

`diagnostic.`**`wave_activity_flux`**(*psi_anom*, *u*, *v*, *plevel=None*, *lat_name=None*, *lon_name=None*)

Returns the stationary component of the wave activity flux, following Takaya and Nakamura, (2001) using zonal and meridional velocity fields on one or more isobaric surface(s)

Author: Dougie Squire

Date: 11/07/2018

Parameters

psi_anom [xarray DataArray] Array containing fields of stream function anomalies with at least coordinates latitude and longitude (following standard naming - see Notes)

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

plevel [value, optional] Value of the pressure level corresponding to the provided arrays. If None, pressure level(s) are extracted from the psi_anom/u/v coordinate. Pressure levels must be provided in units of hPa

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

wave_activity_flux [xarray Dataset]

Dataset containing the following variables:
u_waf; array containing the zonal component of the wave activity flux
v_waf; array containing the meridonal component of the wave activity flux

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)
Pressure levels must be provided in units of hPa
This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian.
These calculations are not yet dask-compatible

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4,2,24)),
...                   coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                           ('level', [100,500]),
...                           ('time', pd.date_range('2000-01-01',periods=24,freq=
↪'M'))])
>>> v = xr.DataArray(np.random.normal(size=(6,4,2,24)),
...                   coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                           ('level', [100,500]),
...                           ('time', pd.date_range('2000-01-01',periods=24,freq=
↪'M'))])
>>> u_clim = u.groupby('time.month').mean(dim='time')
>>> v_clim = v.groupby('time.month').mean(dim='time')
>>> u_anom = doppyo.utils.anomalize(u, u_clim)
>>> v_anom = doppyo.utils.anomalize(v, v_clim)
>>> psi_anom = doppyo.diagnostic.stream_function(u_anom, v_anom)
>>> doppyo.diagnostic.wave_activity_flux(psi_anom, u, v)
<xarray.Dataset>
```

(continues on next page)

(continued from previous page)

```
Dimensions:  (lat: 6, level: 2, lon: 4, time: 24)
Coordinates:
  * level    (level) int64 100 500
  * lat      (lat) int64 -75 -45 -15 15 45 75
  * lon      (lon) int64 45 135 225 315
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
Data variables:
    u_waf     (level, lat, lon, time) float64 0.003852 0.0001439 ... -0.06913
    v_waf     (level, lat, lon, time) float64 0.01495 3.032e-05 ... 0.02944
```

diagnostic.**Brunt_Vaisala**(*temp*, *plevel_name=None*)

Returns the Brunt Väisälä frequency

Author: Dougie Squire

Date: 15/07/2018

Parameters

temp [xarray DataArray] Array containing fields of temperature with at least coordinates latitude, longitude and pressure level (following standard naming - see Notes)

plevel_name [str, optional] Name of pressure level coordinate. If None, doppyo will attempt to determine plevel_name automatically

Returns

nsq [xarray DataArray] Array with same dimensions as input arrays containing the Brunt Väisälä frequency

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

Pressure levels must be provided in units of hPa

To do

- Add switch for atmosphere/ocean input

Examples

```
>>> temp = xr.DataArray(np.random.normal(size=(4,4,2)),
...                     coords=[('lat', np.arange(-90,90,45)), ('lon', np.
↪arange(0,360,90)),
...                             ('level', [100,200])])
>>> doppyo.diagnostic.Brunt_Vaisala(temp)
<xarray.DataArray 'nsq' (level: 2, lon: 4, lat: 4)>
array([[[-2.928266e-01, -2.709919e+01,  2.826585e-02,  6.083374e-01],
        [ 3.260879e-01,  1.933501e-01, -9.033669e+00, -1.468327e+00],
        [-1.957892e+00,  2.408426e-01,  5.597183e-01, -2.548981e+01],
        [-3.234550e-01, -1.907664e+00,  2.506510e-01, -7.385499e-01]],
```

(continues on next page)

(continued from previous page)

```
...
       [[-1.136451e-01, -1.796130e+00, -1.095550e-02,  5.748574e+00],
        [ 4.407484e+02,  4.736099e-01, -5.086917e-01, -6.610682e-01],
        [-2.458302e+00,  6.864762e+00,  2.633289e+00, -4.246873e-01],
        [-1.839424e+01, -1.194455e+00,  5.659980e+02, -2.567729e+00]]])
Coordinates:
  * lat      (lat) int64 -90 -45 0 45
  * lon      (lon) int64 0 90 180 270
  * level    (level) int64 100 200
Attributes:
    long_name:  Brunt-Vaisala frequency squared
    units:      s^-2
```

diagnostic.**Rossby_wave_number**(*u*, *v*, *u_clim*, *lat_name=None*, *lon_name=None*)

Returns the square of the stationary Rossby wave number, Ks**2

Author: Dougie Squire

Date: 11/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

u_clim [xarray DataArray] Array containing climatological fields of zonal velocity with at least coordinates latitude and longitude (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

Returns

Rossby_wave_number [xarray DataArray] Array containing the square of the Rossby wave source

Notes

The input u_clim must have the same dimensions as u and v. One can project a mean climatology, A_clim, over the time dimension in A using `doppyo.utils.anomalize(0*A, -A_clim)`

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

This function utilises the windspharm package, which is a wrapper on pyspharm, which is a wrapper on SPHEREPACK. These packages require that the latitudinal and longitudinal grid is regular or Gaussian.

These calculations are not yet dask-compatible.

To Do

- Make dask-compatible by either developing the windspharm package, or using a kernel approach

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4,24)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                          ('time', pd.date_range('2000-01-01',periods=24,freq=
↪'M'))])
>>> v = xr.DataArray(np.random.normal(size=(6,4,24)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                          ('time', pd.date_range('2000-01-01',periods=24,freq=
↪'M'))])
>>> u_clim = u.groupby('time.month').mean(dim='time')
>>> u_clim = doppyo.utils.anomalize(0*u, -u_clim)
>>> doppyo.diagnostic.Rossby_wave_number(u, v, u_clim)
<xarray.DataArray 'ks2' (lat: 6, lon: 4, time: 24)>
array([[[ 8.077277e-01,  1.885835e-01, ...,  6.383953e-01, -4.686696e-01],
        [-3.756420e-01,  1.210226e+00, ..., -2.055076e+00, -2.291500e+00],
        [ 8.786361e-01,  4.181778e-01, ..., -2.071749e+00,  4.018699e-01],
        [ 8.218020e-01,  5.197270e+00, ...,  5.181735e+00,  7.112056e-01]],
...
       [[-5.323813e+02, -2.894449e+02, ..., -5.063012e+03, -3.921559e+02],
        [ 3.167388e+02, -5.406136e+02, ..., -1.987485e+03, -2.692395e+02],
        [ 2.916992e+03,  2.318578e+02, ...,  8.611478e+02,  8.559919e+02],
        [-4.380459e+02, -5.035198e+02, ..., -1.844072e+03, -2.856807e+02]],
...,
       [[ 3.832781e+02, -1.272144e+03, ...,  3.900539e+02, -5.402686e+02],
        [-2.494814e+02, -2.041985e+02, ...,  3.426493e+02, -5.557717e+02],
        [-6.290198e+03,  1.606871e+03, ...,  2.894713e+03,  3.284330e+02],
        [-3.325505e+02, -2.406172e+02, ..., -3.270787e+03, -1.040641e+03]],
...
       [[ 1.401437e+00,  6.053096e-01, ...,  1.725558e-01, -7.287578e+01],
        [-8.905873e-01,  1.469694e-01, ...,  1.308367e+00, -7.136195e-01],
        [ 4.318194e+01, -1.850361e-01, ..., -2.447798e-01, -4.454747e-01],
        [ 1.247740e+00,  9.826164e-02, ...,  2.808380e+00,  1.254609e+00]]])
Coordinates:
  * lat      (lat) int64 75 45 15 -15 -45 -75
  * lon      (lon) int64 45 135 225 315
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
Attributes:
    units:      real number
    long_name:  Square of Rossby stationary wavenumber
```

diagnostic.**Eady_growth_rate**(*u*, *v*, *gh*, *nsq*, *lat_name=None*, *lon_name=None*, *level_name=None*)

Returns the square of the Eady growth rate

Author: Dougie Squire
Date: 15/07/2018

Parameters

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude, longitude and level (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude, longitude and level (following standard naming - see Notes)

gh [xarray DataArray] Array containing fields of geopotential height with at least coordinates latitude, longitude and level (following standard naming - see Notes)

nsq [xarray DataArray] Array containing fields of Brunt Väisälä frequency with at least coordinates latitude, longitude and level (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

level_name [str, optional] Name of level coordinate. If None, doppyo will attempt to determine level_name automatically

Returns

Eady^2 [xarray DataArray] Array containing the square of the Eady growth rate

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(6,4,2)),
...                      coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                              ('level', [200, 500])])
>>> v = xr.DataArray(np.random.normal(size=(6,4,2)),
...                      coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                              ('level', [200, 500])])
>>> temp = xr.DataArray(np.random.normal(size=(6,4,2)),
...                         coords=[('lat', np.arange(-75,76,30)), ('lon', np.
↪arange(45,316,90)),
...                                 ('level', [200, 500])])
>>> gh = xr.DataArray(np.random.normal(size=(6,4,2)),
...                       coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                               ('level', [200, 500])])
>>> nsq = doppyo.diagnostic.Brunt_Vaisala(temp)
>>> doppyo.diagnostic.Eady_growth_rate(u, v, gh, nsq)
<xarray.DataArray 'Eady^2' (level: 2, lon: 4, lat: 6)>
array([[[-5.371897e-08,  1.338133e-11, -7.254014e-13, -8.196598e-12,
          2.062633e-09, -7.200158e-12],
        [ 9.906932e-10, -7.349832e-09, -2.558847e-12, -1.695842e-09,
          4.986779e-09, -3.090147e-09],
        [ 3.948602e-07,  1.397756e-09,  1.508010e-10,  1.481968e-10,
          5.627093e-11,  7.463454e-10],
        [ 4.326971e-09, -2.528522e-09, -1.243954e-13, -3.138463e-11,
         -6.801250e-09, -6.286382e-10]],
...
       [[-8.580527e-10,  7.040065e-12, -3.760004e-13, -1.213131e-12,
          2.437557e-11, -6.522981e-11],
        [ 6.119671e-09, -1.644123e-09, -5.124997e-11,  1.725101e-08,
```

(continues on next page)

(continued from previous page)

```
          2.574158e-08, -3.101566e-10],
        [ 1.601742e-06,  1.994867e-11,  3.341006e-11,  1.641253e-11,
          5.601919e-08,  5.527214e-11],
        [ 4.700271e-09, -1.422149e-11, -1.302035e-12, -2.153002e-11,
         -4.607096e-10, -3.813686e-09]]])
Coordinates:
  * lat      (lat) int64 -75 -45 -15 15 45 75
  * lon      (lon) int64 45 135 225 315
  * level    (level) int64 200 500
Attributes:
    units:      s^-2
    long_name:  Square of Eady growth rate
```

`diagnostic.`**`thermal_wind`**`(`*gh*, *plevel_lower*, *plevel_upper*, *lat_name=None*, *lon_name=None*, *plevel_name=None*`)`

Returns the thermal wind, (u_tw, v_tw) = 1/f x k x grad(thickness), where f = 2*Omega*sin(lat)

Author: Dougie Squire

Date: 15/07/2018

Parameters

gh [xarray DataArray] Array containing fields of geopotential height with at least coordinates latitude, longitude and level (following standard naming - see Notes)

plevel_lower [value] Value of lower pressure level used to compute termal wind. Must exist in level coordinate of gh

plevel_upper [value] Value of upper pressure level used to compute termal wind. Must exist in level coordinate of gh

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

plevel_name [str, optional] Name of pressure level coordinate. If None, doppyo will attempt to determine plevel_name automatically

Returns

thermal_wind [xarray Dataset]

Dataset containing the following variables:

u_tw; array containing the zonal component of the thermal wind

v_tw; array containing the meridonal component of the thermal wind

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

Pressure levels must be provided in units of hPa

Examples

```
>>> gh = xr.DataArray(np.random.normal(size=(3,4,4)),
...                        coords=[('level', [400, 500, 600]), ('lat', np.arange(-90,
↪90,45)),
...                        ('lon', np.arange(0,360,90))])
>>> doppyo.diagnostic.thermal_wind(gh, plevel_lower=400, plevel_upper=600)
<xarray.Dataset>
Dimensions:  (lat: 4, lon: 4)
Coordinates:
    level    float64 500.0
  * lon      (lon) int64 0 90 180 270
  * lat      (lat) int64 -90 -45 0 45
Data variables:
    u_tw     (lon, lat) float64 0.003727 0.0006837 inf ... inf -0.0001238
    v_tw     (lat, lon) float64 4.515e+12 -1.443e+12 ... -0.000569 -0.0002777
```

diagnostic.**eofs**(*da*, *sample_dim='time'*, *weight=None*, *n_modes=20*)

Returns the empirical orthogonal functions (EOFs), and associated principle component timeseries (PCs), and explained variances of provided array. Follows notation used in “Bjornsson H. and Venegas S. A. 1997 A Manual for EOF and SVD analyses of Climatic Data”, whereby, (phi, sqrt_lambdas, EOFs) = svd(data) and PCs = phi * sqrt_lambdas

Author: Dougie Squire

Date: 19/18/2018

Parameters

da [xarray DataArray or sequence of xarray DataArrays] Array to use to compute EOFs. When input array is a list of xarray objects, returns the joint EOFs associated with each object. In this case, all xarray objects in da must have sample_dim dimensions of equal length.

sample_dim [str, optional] EOFs sample dimension

weight [xarray DataArray or sequence of xarray DataArrays, optional] Weighting to apply prior to svd. If weight=None, cos(lat)^2 weighting are used. If weight is specified, it must be the same length as da with each element broadcastable onto each element of da

n_modes [values, optional] Number of EOF modes to return

Returns

eofs [xarray Dataset]

Dataset containing the following variables:

EOFs; array containing the empirical orthogonal functions

PCs; array containing the associated principle component timeseries

lambdas; array containing the eigenvalues of the covariance of the input data

explained_var; array containing the fraction of the total variance explained by each EOF mode

Notes

This function is a wrapper on scipy.sparse.linalg.svds which is a naive implementation using ARPACK. Thus, the approach implemented here is non-lazy and could incur large increases in memory usage.

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(6,4,40)),
...                  coords=[('lat', np.arange(-75,76,30)), ('lon', np.arange(45,
↪316,90)),
...                          ('time', pd.date_range('2000-01-01', periods=40,␣
↪freq='M'))])
>>> doppyo.diagnostic.eofs(A)
<xarray.Dataset>
Dimensions:        (lat: 6, lon: 4, mode: 20, time: 40)
Coordinates:
  * time           (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2003-04-30
  * mode           (mode) int64 1 2 3 4 5 6 7 8 9 ... 12 13 14 15 16 17 18 19 20
  * lat            (lat) int64 -75 -45 -15 15 45 75
  * lon            (lon) int64 45 135 225 315
Data variables:
    EOFs           (mode, lat, lon) float64 -0.05723 -0.01997 ... 0.08166
    PCs            (time, mode) float64 1.183 -1.107 -0.5385 ... -0.08552 0.1951
    lambdas        (mode) float64 87.76 80.37 68.5 58.14 ... 8.269 6.279 4.74
    explained_var  (mode) float64 0.1348 0.1234 0.1052 ... 0.009644 0.00728
```

`diagnostic.`**`mean_merid_mass_streamfunction`**(*v*, *lat_name=None*, *lon_name=None*, *plevel_name=None*)

Returns the mean meridional mass stream function averaged over all provided longitudes

Author: Dougie Squire
Date: 15/07/2018

Parameters

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude, longitude and level (following standard naming - see Notes)

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

plevel_name [str, optional] Name of pressure level coordinate. If None, doppyo will attempt to determine plevel_name automatically

Returns

mmms [xarray DataArray] New DataArray object containing the mean meridional mass stream function

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)
Pressure levels must be provided in units of hPa

Examples

```
>>> v = xr.DataArray(np.random.normal(size=(2,4,4)),
...                     coords=[('level', [400, 600]), ('lat', np.arange(-90,90,
↪45)),
...                     ('lon', np.arange(0,360,90))])
>>> doppyo.diagnostic.mean_merid_mass_streamfunction(v)
<xarray.DataArray 'mmms' (lat: 4, level: 2)>
array([[ 0.000000e+00, -1.336316e-07],
       [ 0.000000e+00, -1.447547e+10],
       [ 0.000000e+00, -3.208457e+09],
       [ 0.000000e+00, -2.562681e+10]])
Coordinates:
  * lat      (lat) int64 -90 -45 0 45
  * level    (level) int64 400 600
```

diagnostic.**atmos_energy_cycle**(*temp, u, v, omega, gh, terms=None, vgradz=False, spectral=False, n_wavenumbers=20, integrate=True, lat_name=None, lon_name=None, plevel_name=None*)

Returns all terms in the Lorenz energy cycle. Follows formulae and notation used in *Marques et al. 2011 Global diagnostic energetics of five state-of-the-art climate models. Climate Dynamics*. Note that this decomposition is in the space domain. A space-time decomposition can also be carried out (though not in Fourier space, but this is not implemented here (see *Oort. 1964 On Estimates of the atmospheric energy cycle. Monthly Weather Review*).

Author: Dougie Squire
Date: 15/07/2018

Parameters

temp [xarray DataArray] Array containing fields of temperature with at least coordinates latitude, longitude and level (following standard naming - see Notes)

u [xarray DataArray] Array containing fields of zonal velocity with at least coordinates latitude, longitude and level (following standard naming - see Notes)

v [xarray DataArray] Array containing fields of meridional velocity with at least coordinates latitude, longitude and level (following standard naming - see Notes)

omega [xarray DataArray] Array containing fields of vertical velocity (pressure coordinates) with at least coordinates latitude, longitude and level (following standard naming - see Notes)

gh [xarray DataArray] Array containing fields of geopotential height with at least coordinates latitude, longitude and level (following standard naming - see Notes)

terms [str or sequence of str]

List of terms to compute. If None, returns all terms. Available options are:

Pz; total available potential energy in the zonally averaged temperature distribution

Kz; total kinetic energy in zonally averaged motion

Pe; total eddy available potential energy [= sum_n Pn (n > 0 only) for spectral=True] (Note that for spectral=True, an additional term, Sn, quantifying the rate of transfer of available potential energy to eddies of wavenumber n from eddies of all other wavenumbers is also returned)

Ke; total eddy kinetic energy [= sum_n Kn (n > 0 only) for spectral=True] (Note that for spectral=True, an additional term, Ln, quantifying the rate of transfer of kinetic energy to eddies of wavenumber n from eddies of all other wavenumbers is also returned)

Cz; rate of conversion of zonal available potential energy to zonal kinetic energy

Ca; rate of transfer of total available potential energy in the zonally averaged temperature distribution (Pz) to total eddy available potential energy (Pe) [= sum_n Rn (n > 0 only) for spectral=True]

Ce; rate of transfer of total eddy available potential energy (Pe) to total eddy kinetic energy (Ke) [= sum_n Cn (n > 0 only) for spectral=True]

Ck; rate of transfer of total eddy kinetic energy (Ke) to total kinetic energy in zonally averaged motion (Kz) [= sum_n Mn (n > 0 only) for spectral=True]

Gz; rate of generation of zonal available potential energy due to the zonally averaged heating (Pz). Note that this term is computed as a residual (Cz + Ca) and cannot be returned in spectral space. If Gz is requested with spectral=True, Gz is returned in real-space only

Ge; rate of generation of eddy available potential energy (Pe). Note that this term is computed as a residual (Ce - Ca) and cannot be returned in spectral space. If Ge is requested with spectral=True, Ge is returned in real-space only

Dz; rate of viscous dissipation of zonal kinetic energy (Kz). Note that this term is computed as a residual (Cz - Ck) and cannot be returned in spectral space. If Dz is requested with spectral=True, Dz is returned in real-space only

De; rate of dissipation of eddy kinetic energy (Ke). Note that this term is computed as a residual (Ce - Ck) and cannot be returned in spectral space. If De is requested with spectral=True, De is returned in real-space only

vgradz [bool, optional] If True, uses *v-grad-z* approach for computing terms relating to conversion of potential energy to kinetic energy. Otherwise, defaults to using the *omega-alpha* approach (see reference above for details)

spectral [bool, optional] If True, computes all terms as a function of wavenumber on longitudinal bands. To use this option, longitudes must be regularly spaced. Note that Ge and De are computed as residuals and cannot be computed in spectral space

n_wavenumbers [int, optional] Number of wavenumbers to retain either side of wavenumber=0. Obviously only does anything if spectral=True

integrate [bool, optional] If True, computes and returns the integral of each term over the mass of the atmosphere. Otherwise, only the integrands are returned

lat_name [str, optional] Name of latitude coordinate. If None, doppyo will attempt to determine lat_name automatically

lon_name [str, optional] Name of longitude coordinate. If None, doppyo will attempt to determine lon_name automatically

plevel_name [str, optional] Name of pressure level coordinate. If None, doppyo will attempt to determine plevel_name automatically

Returns

atmos_energy_cycle [xarray Dataset] Dataset containing the requested variables plus gamma, the stability parameter. If integrate=True, both the integrand (<term>_int) and the integral over the mass of the atmosphere (<term>) are returned for each requested term. Otherwise, only the integrands are returned.

Notes

The following notation is used below (stackable, e.g. *_ZT indicates the time average of the zonal average):

*_A -> area average over an isobaric surface
*_a -> departure from area average
*_Z -> zonal average
*_z -> departure from zonal average
*_T -> time average
*_t -> departure from time average
Additionally, capital variables indicate Fourier transforms:
F(u) = U
F(v) = V
F(omega) = O
F(gh) = A
F(temp) = B

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc).
Pressure levels must be provided in units of hPa
The terms Sn and Ln, which are computed when Pe and Ke are requested with spectral=True, rely on "triple terms" that are very intensive and can take a significant amount of time and memory to compute (see _triple_terms() below). Often (i.e. for arrays of sufficient size to be of interest) requesting these terms yeilds a MemoryError–if working in memory –or a KilledWorkerError–if working out of memory

To do

- Arrays that are sufficiently large to be interesting currently max out the available memory when Sn or Ln are requested. I need to implement a less hungry method for computing the "triple terms" (see _triple_terms() below)

Examples

```
>>> temp = xr.DataArray(np.random.normal(size=(90,90,9,5)),
...                         coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                                 ('level', np.arange(100,1000,100)),
...                                 ('time', pd.date_range('2000-01-01', periods=5,␣
↪freq='M'))])
>>> u = xr.DataArray(np.random.normal(size=(90,90,9,5)),
...                      coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                              ('level', np.arange(100,1000,100)),
...                              ('time', pd.date_range('2000-01-01', periods=5, freq=
↪'M'))])
>>> v = xr.DataArray(np.random.normal(size=(90,90,9,5)),
...                      coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                              ('level', np.arange(100,1000,100)),
...                              ('time', pd.date_range('2000-01-01', periods=5, freq=
↪'M'))])
>>> omega = xr.DataArray(np.random.normal(size=(90,90,9,5)),
...                          coords=[('lat', np.arange(-90,90,2)), ('lon', np.
↪arange(0,360,4)),
...                                  ('level', np.arange(100,1000,100)),
```

(continues on next page)

(continued from previous page)

```
...                                ('time', pd.date_range('2000-01-01', periods=5,␣
↪freq='M'))])
>>> gh = xr.DataArray(np.random.normal(size=(90,90,9,5)),
...                       coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                               ('level', np.arange(100,1000,100)),
...                               ('time', pd.date_range('2000-01-01', periods=5,␣
↪freq='M'))])
>>> doppyo.diagnostic.atmos_energy_cycle(temp, u, v, omega, gh, spectral=True)
<xarray.Dataset>
Dimensions:  (lat: 90, level: 9, n: 41, time: 5)
Coordinates:
  * level    (level) int64 100 200 300 400 500 600 700 800 900
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2000-05-31
  * lat      (lat) int64 -90 -88 -86 -84 -82 -80 -78 ... 76 78 80 82 84 86 88
  * n        (n) float64 -20.0 -19.0 -18.0 -17.0 -16.0 ... 17.0 18.0 19.0 20.0
Data variables:
    gamma    (level, time) float64 8.993 120.3 68.1 ... -6.874 1.083 -1.383
    Pz_int   (level, time, lat) float64 83.76 64.07 67.67 ... -8.283 -0.7205
    Pz       (time) float64 -9.73e+04 8.225e+05 -1.892e+04 -4.197e+06 -9.113e+05
    Kz_int   (lat, level, time) float64 0.03326 0.01417 ... 0.01454 0.005276
    Kz       (time) float64 88.08 93.48 97.19 91.19 85.38
    Cz_int   (level, lat, time) float64 0.0001505 -6.762e-05 ... -3.222e-06
    Cz       (time) float64 0.01983 0.02128 -0.04917 -0.04136 -0.04431
    Pn_int   (level, time, lat, n) float64 109.5 163.6 132.4 ... -0.5592 -31.31
    Pn       (time, n) float64 -1.496e+05 -1.48e+05 ... -1.712e+06 -1.534e+06
    Sn_int   (level, time, n, lat) complex128 (-1.635e+12+1.365e+12j) ... (1.546e-
↪04-3.464e-03j)
    Sn       (time, n) float64 54.46 42.84 39.39 10.27 ... 43.73 55.43 37.28
    Kn_int   (lat, n, level, time) float64 0.02795 0.02618 ... 0.0314 0.005119
    Kn       (n, time) float64 184.5 179.1 183.1 186.4 ... 183.1 186.4 176.6
    Ln_int   (n, lat, level, time) complex128 (-1.401e+09+4.623e+08j) ... (2.400e-
↪06+9.272e-07j)
    Ln       (n, time) float64 7.325e-05 0.0001285 ... 8.57e-05 0.0001433
    Rn_int   (level, time, lat, n) complex128 (5.631e-03-1.433e-17j) ... (-3.295e-
↪04+8.862e-19j)
    Rn       (time, n) float64 0.3357 0.5211 0.00877 ... 3.811 0.04257 3.209
    Cn_int   (level, lat, n, time) complex128 (-1.694e-04+4.232e-19j) ... (-1.
↪836e-04+4.979e-19j)
    Cn       (n, time) float64 0.09795 0.04268 0.01845 ... 0.04054 0.03553
    Mn_int   (lat, n, level, time) complex128 (-1.376e+06+2.933e-09j) ... (5.344e-
↪07-1.670e-21j)
    Mn       (n, time) float64 1.526e+06 8.963e+05 ... 4.038e+06 8.648e+05
    Gz_int   (level, lat, time) float64 -0.01321 -0.2201 ... -6.633e-05
    Gz       (time) float64 1.33 -10.62 27.5 -31.06 -10.44
    Ge_int   (level, lat, time) float64 0.01375 0.2213 ... 0.0007708 2.494e-05
    Ge       (time) float64 -1.011 10.69 -27.39 31.14 10.57
    Dz_int   (level, lat, time) float64 7.444e+07 -6.406e+07 ... -3.544e-06
    Dz       (time) float64 1.009e+07 6.951e+06 1.85e+07 1.491e+07 1.306e+07
    De_int   (level, lat, time) float64 7.444e+07 -6.406e+07 ... -3.849e-05
    De       (time) float64 1.009e+07 6.951e+06 1.85e+07 1.491e+07 1.306e+07
```

`diagnostic.`**`isotherm_depth`**`(`*temp*, *target_temp=20*, *depth_name=None*`)`

Returns the depth of an isotherm given a target temperature. If no temperatures in the column exceed the target temperature, a nan is returned at that point

Author: Thomas Moore

Date: 02/10/2018

Parameters

temp [xarray DataArray] Array containing values of temperature with at least a depth dimension

target_temp [value, optional] Value of temperature used to compute isotherm depth. Default value is 20 degC

depth_name [str, optional] Name of depth coordinate. If None, doppyo will attempt to determine depth_name automatically

Returns

isotherm_depth [xarray DataArray] Array containing the depth of the requested isotherm

Notes

All input array coordinates must follow standard naming (see `doppyo.utils.get_lat_name()`, `doppyo.utils.get_lon_name()`, etc)

If multiple occurences of target occur along the depth coordinate, only the maximum value of coord is returned

The current version includes no interpolation between grid spacing. This should be added as an option in the future

Examples

```
>>> temp = xr.DataArray(20 + np.random.normal(scale=5, size=(4,4,10)),
...                         coords=[('lat', np.arange(-90,90,45)), ('lon', np.
↪arange(0,360,90)),
...                                 ('depth', np.arange(2000,0,-200))])
>>> doppyo.diagnostic.isotherm_depth(temp)
<xarray.DataArray 'isosurface' (lat: 4, lon: 4)>
array([[ 400., 1600., 2000.,  800.],
       [1800., 2000., 1800., 2000.],
       [2000., 2000., 2000., 1600.],
       [1400., 2000., 2000., 2000.]])
Coordinates:
  * lat      (lat) int64 -90 -45 0 45
  * lon      (lon) int64 0 90 180 270
```

`diagnostic.`**`pwelch`**(*da1*, *da2*, *dim*, *nwindow*, *overlap=50*, *dx=None*, *hanning=False*)

Compute the cross/power spectral density along a dimension using welch's method. Note that the spectral density is always computed relative to a "frequency" f = 1/dx (see Notes for details)

Author: Dougie Squire

Date: 20/07/2018

Parameters

da1 [xarray DataArray] First array of data to use in spectral density calculation. For power spectral density, da1 = da2

da2 [xarray DataArray] Second array of data to use in spectral density calculation. For power spectral density, da1 = da2

dim [str] Dimension to compute spectral density along

nwindow [value] Length of the signal segments for pwelch calculation

overlap [value, optional] Percentage overlap of the signal segments for pwelch calculation

dx [value, optional] Spacing along the dimension dim. If None, dx is determined from the coordinate dim. For consistency between spatial and temporal dim, spectra is computed relative to a “frequency”, f = 1/dx, where dx is the spacing along dim, e.g.:

- for temporal dim, dx is computed in seconds. Thus, f = 1/seconds = Hz
- for spatial dim in meters, f = 1/meters = k/(2*pi)
- for spatial dim in degrees, f = 1/degrees = k/360

If converting the “frequency” to wavenumber, for example, one must also adjust the spectra magnitude so that the integral remains equal to the variance, e.g. for spatial spectra, k = f*(2*pi) -> phi_new = phi_old/(2*pi)

hanning [bool, optional] If True, a Hanning window weighting is applied prior to the fft

Returns

spectra [xarray DataArray] Array containing the power spectral density of the input array(s)

Examples

```
>>> u = xr.DataArray(np.random.normal(size=(500)),
...                  coords=[('time', pd.date_range('2000-01-01', periods=500,␣
↪freq='D'))])
>>> spectra = doppyo.diagnostic.pwelch(u, u, dim='time', nwindow=20)
>>> seconds_to_days = 60*60*24
>>> spectra['f_time'] = seconds_to_days*spectra['f_time'] # Change freq from Hz␣
↪to 1/days
>>> spectra = spectra/seconds_to_days
>>> print(spectra)
<xarray.DataArray 'spectra' (f_time: 11)>
array([1.381912, 1.89882 , 1.641378, 1.939686, 2.218824, 1.941639, 2.142277,
       1.689319, 1.983302, 2.225709, 2.732023])
Coordinates:
  * f_time   (f_time) float64 0.0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45 0.5
```

diagnostic.**inband_variance**(*da*, *dim*, *bounds*, *nwindow*, *overlap=50*)

Compute the in-band variance along a specified dimension.

Author: Dougie Squire
Date: 20/07/2018

Parameters

da [xarray DataArray] Array with which to compute in-band variance

dim [str] Dimension along which to compute in-band variance

bounds [sequence] Frequency bounds for in-band variance calculation. Note that for consistency between spatial and temporal dim, all spectra are computed relative to a "frequency", f = 1/dx, where dx is the spacing along dim, e.g.:

- for temporal dim, dx is computed in seconds. Thus, f = 1/seconds = Hz
- for spatial dim in meters, f = 1/meters = k/(2*pi)
- for spatial dim in degrees, f = 1/degrees = k/360

Thus, bounds must be provided in a way consistent with this, e.g.:

- for temporal dim, bounds = 1 / (60*60*24*[d1, d2, d3]), where d# are numbers of days
- for spatial dim, bounds = 1 / [l1, l2, l3], where l# are numbers of meters, degrees, etc

nwindow [value] Length of the signal segments for pwelch calculation

overlap [value, optional] Percentage overlap of the signal segments for pwelch calculation

Returns

inband_var [xarray DataArray] Array containing the in-band variances of the input array

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(500)),
...                  coords=[('time', pd.date_range('2000-01-01', periods=500,␣
↪freq='D'))])
>>> doppyo.diagnostic.inband_variance(A, dim='time',
...                                   bounds=1/(60*60*24*np.array([2, 5, 10])),␣
↪nwindow=20)
<xarray.DataArray 'in-band' (f_time_bins: 2)>
array([0.106615, 0.492033])
Coordinates:
  * f_time_bins  (f_time_bins) object [1.16e-06, 2.31e-06) [2.31e-06, 5.79e-06)
```

diagnostic.**nino3**(*sst_anom*)

Returns Nino-3 index

Author: Dougie Squire

Date: 10/04/2018

Parameters

sst_anom [xarray DataArray] Array containing sea surface temperature anomalies

Returns

nino3 [xarray DataArray] Average of the provided sst anomalies over the nino-3 box

Examples

```
>>> sst = xr.DataArray(np.random.normal(size=(90,90,24)),
...                    coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                            ('time', pd.date_range('2000-01-01', periods=24,␣
↪freq='M'))])
```

(continues on next page)

(continued from previous page)

```
>>> sst_clim = sst.groupby('time.month').mean(dim='time')
>>> sst_anom = doppyo.utils.anomalize(sst, sst_clim)
>>> doppyo.diagnostic.nino3(sst_anom)
<xarray.DataArray 'nino3' (time: 24)>
array([-0.137317, -0.094808, -0.01091 , -0.04653 ,  0.030562, -0.065515,
       -0.109851,  0.118016,  0.092496, -0.030821, -0.011724, -0.002773,
        0.137317,  0.094808,  0.01091 ,  0.04653 , -0.030562,  0.065515,
        0.109851, -0.118016, -0.092496,  0.030821,  0.011724,  0.002773])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

diagnostic.**nino34**(*sst_anom*)

Returns Nino-3.4 index

Author: Dougie Squire
Date: 10/04/2018

Parameters

sst_anom [xarray DataArray] Array containing sea surface temperature anomalies

Returns

nino34 [xarray DataArray] Average of the provided sst anomalies over the nino-3.4 box

Examples

```
>>> sst = xr.DataArray(np.random.normal(size=(90,90,24)),
...                    coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                            ('time', pd.date_range('2000-01-01', periods=24,␣
↪freq='M'))])
>>> sst_clim = sst.groupby('time.month').mean(dim='time')
>>> sst_anom = doppyo.utils.anomalize(sst, sst_clim)
>>> doppyo.diagnostic.nino34(sst_anom)
<xarray.DataArray 'nino34' (time: 24)>
array([-0.052202,  0.00467 ,  0.121013,  0.007983, -0.070645,  0.051945,
       -0.045485,  0.065569, -0.018723, -0.053734,  0.10527 , -0.113451,
        0.052202, -0.00467 , -0.121013, -0.007983,  0.070645, -0.051945,
        0.045485, -0.065569,  0.018723,  0.053734, -0.10527 ,  0.113451])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

diagnostic.**nino4**(*sst_anom*)

Returns Nino-4 index

Author: Dougie Squire
Date: 10/04/2018

Parameters

sst_anom [xarray DataArray] Array containing sea surface temperature anomalies

Returns

nino4 [xarray DataArray] Average of the provided sst anomalies over the nino-4 box

Examples

```
>>> sst = xr.DataArray(np.random.normal(size=(90,90,24)),
...                        coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                                ('time', pd.date_range('2000-01-01', periods=24,␣
↪freq='M'))])
>>> sst_clim = sst.groupby('time.month').mean(dim='time')
>>> sst_anom = doppyo.utils.anomalize(sst, sst_clim)
>>> doppyo.diagnostic.nino4(sst_anom)
<xarray.DataArray 'nino4' (time: 24)>
array([ 0.017431, -0.086129,  0.106992, -0.097994,  0.109215, -0.120221,
        0.042459, -0.189595,  0.005097,  0.034218,  0.019478,  0.054122,
       -0.017431,  0.086129, -0.106992,  0.097994, -0.109215,  0.120221,
       -0.042459,  0.189595, -0.005097, -0.034218, -0.019478, -0.054122])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

diagnostic.**emi**(*sst_anom*)

Returns El Nino Modoki index

Author: Dougie Squire

Date: 10/04/2018

Parameters

sst_anom [xarray DataArray] Array containing sea surface temperature anomalies

Returns

emi [xarray DataArray] Array containing the El Nino Modoki index

Examples

```
>>> sst = xr.DataArray(np.random.normal(size=(90,90,24)),
...                        coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                                ('time', pd.date_range('2000-01-01', periods=24,␣
↪freq='M'))])
>>> sst_clim = sst.groupby('time.month').mean(dim='time')
>>> sst_anom = doppyo.utils.anomalize(sst, sst_clim)
>>> doppyo.diagnostic.emi(sst_anom)
<xarray.DataArray 'emi' (time: 24)>
array([-0.046743,  0.181795,  0.020386, -0.215317, -0.209294,  0.109291,
        0.202055, -0.021001, -0.013106,  0.094376, -0.000516, -0.021762,
        0.046743, -0.181795, -0.020386,  0.215317,  0.209294, -0.109291,
       -0.202055,  0.021001,  0.013106, -0.094376,  0.000516,  0.021762])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

`diagnostic.`**`dmi`**`(`*sst_anom*`)`

Returns dipole mode index

Author: Dougie Squire
Date: 10/04/2018

Parameters

sst_anom [xarray DataArray] Array containing sea surface temperature anomalies

Returns

dmi [xarray DataArray] Array containing the dipole mode index

Examples

```
>>> sst = xr.DataArray(np.random.normal(size=(90,90,24)),
...                       coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                               ('time', pd.date_range('2000-01-01', periods=24,␣
↪freq='M'))])
>>> sst_clim = sst.groupby('time.month').mean(dim='time')
>>> sst_anom = doppyo.utils.anomalize(sst, sst_clim)
>>> doppyo.diagnostic.dmi(sst_anom)
<xarray.DataArray 'dmi' (time: 24)>
array([-0.225498,  0.220686,  0.032038,  0.019634,  0.00511 , -0.202789,
       -0.014349, -0.293248,  0.020925,  0.114059,  0.066389,  0.238707,
        0.225498, -0.220686, -0.032038, -0.019634, -0.00511 ,  0.202789,
        0.014349,  0.293248, -0.020925, -0.114059, -0.066389, -0.238707])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

`diagnostic.`**`soi`**`(`*slp_anom*, *lat_name=None*, *lon_name=None*, *time_name=None*`)`

Returns southern oscillation index as defined by NOAA (see, for example, https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/SOI/)

Author: Dougie Squire
Date: 10/04/2018

Parameters

slp_anom [xarray DataArray] Array containing sea level pressure anomalies

time_name [str, optional] Name of the time dimension. If None, doppyo will attempt to determine time_name automatically

Returns

soi [xarray DataArray] Array containing the southern oscillation index

Examples

```
>>> slp = xr.DataArray(np.random.normal(size=(90,90,24)),
...                    coords=[('lat', np.arange(-90,90,2)), ('lon', np.arange(0,
↪360,4)),
...                    ('time', pd.date_range('2000-01-01', periods=24, freq='M
↪'))])
>>> slp_clim = slp.groupby('time.month').mean(dim='time')
>>> slp_anom = doppyo.utils.anomalize(slp, slp_clim)
>>> doppyo.diagnostic.soi(slp_anom)
<xarray.DataArray 'soi' (time: 24)>
array([ 0.355277,  0.38263 ,  0.563005, -1.256122, -1.252341,  0.202942,
        0.691819,  0.412523, -1.368695,  0.421943,  2.349053,  0.069382,
       -0.355277, -0.38263 , -0.563005,  1.256122,  1.252341, -0.202942,
       -0.691819, -0.412523,  1.368695, -0.421943, -2.349053, -0.069382])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2001-12-31
```

3.2 skill

doppyo functions for assessing one dataset relative to another (usually model output to observation)

In the following we refer to the datasets being assessed as comparison data (da_cmp) and reference data (da_ref). We seek to assess the skill of the former relative to the latter. Usually, da_cmp and da_ref comprise model output (e.g. forecasts) and observations, respectively.

3.2.1 API

skill.**rank_histogram**(*da_cmp*, *da_ref*, *over_dims*, *norm=True*, *ensemble_dim='ensemble'*)

Returns the rank histogram along the specified dimensions

Authors: Dougie Squire
Date: 01/11/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts). This data is used to rank the reference data. Must include an ensemble dimension

da_ref [xarray DataArray] Array containing reference data (usually observations). This data is ranked within the comparison data. Dimensions should match those of da_cmp

over_dims [str or sequence of str] The dimension(s) over which to compute the histogram of ranks

norm [bool, optional] If True, rank histograms are normalised by their enclosed area

ensemble_dim [str, optional] The name of the ensemble dimension in da_cmp

Returns

rank_histogram [xarray DataArray] New DataArray object containing the rank histograms

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)), coords=[('x', np.
↪arange(3)),
...                                                              ('y', np.
↪arange(3))])
>>> doppyo.skill.rank_histogram(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'rank_histogram' (bins: 4, y: 3)>
array([[1.      , 0.333333, 0.333333],
       [0.      , 0.333333, 0.333333],
       [0.      , 0.      , 0.333333],
       [0.      , 0.333333, 0.      ]])
Coordinates:
  * bins     (bins) float64 1.0 2.0 3.0 4.0
  * y        (y) int64 0 1 2
```

`skill.`**`rps`**(*da_cmp*, *da_ref*, *bins*, *over_dims=None*, *ensemble_dim='ensemble'*)

Returns the ranked probability score

Author: Dougie Squire

Date: 10/05/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

bins [array_like] Bins to compute the ranked probability score over

over_dims [str or sequence of str, optional] Dimensions over which to average the ranked probability score

ensemble_dim [str, optional] Name of ensemble dimension

Returns

rps [xarray DataArray] Array containing ranked probability score

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)), coords=[('x', np.
↪arange(3)),
...                                                             ('y', np.
↪arange(3))])
>>> bins = np.linspace(-2,2,10)
>>> doppyo.skill.rps(da_cmp, da_ref, bins=bins, over_dims='x')
<xarray.DataArray 'rps' (y: 3)>
array([0.36214 , 0.806584, 0.263374])
Coordinates:
  * y        (y) int64 0 1 2
```

skill.**reliability**(*cmp_likelihood*, *ref_logical*, *over_dims*, *probability_bins=array([0.*, *0.25*, *0.5*, *0.75*, *1.])*, *nans_as_zeros=True*)

Computes the relative frequency of an event for a range of probability threshold bins given the comparison likelihood and reference logical event data

Author: Dougie Squire

Date: 10/05/2018

Parameters

cmp_likelihood [xarray DataArray] Array containing likelihoods of the event from the comparison data (e.g. cmp_likelihood = (da_cmp > 1).mean(dim='ensemble'))

ref_logical [xarray DataArray] Array containing logical (True/False) outcomes of the event from the reference data (e.g. ref_logical = (da_ref > 1))

over_dims [str or sequence of str] Dimensions over which to compute the reliability

probability_bins [array_like, optional] Probability threshold bins. Defaults to 5 equally spaced bins between 0 and 1

nans_as_zeros [bool, optional] Replace output nans (resulting fron bins with no data) with zeros

Returns

reliability [xarray DataSet]

Dataset containing the following variables:

relative_freq; the relative frequency of occurence for each probability threshold bin

cmp_number; the number of instances that the comparison data fall within each probability threshold bin

ref_occur; the number of instances that the reference data is True when the comparison data falls within each probability threshold bin

Notes

See http://www.cawcr.gov.au/projects/verification/

To do

- Currently using a for-loop to process each probability bin separately. Is it possible to remove this loop?

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> cmp_likelihood = (da_cmp > 0.1).mean('ensemble')
>>> ref_logical = da_ref > 0.1
>>> doppyo.skill.reliability(cmp_likelihood, ref_logical, over_dims='x')
<xarray.Dataset>
Dimensions:          (probability_bin: 5, y: 3)
Coordinates:
  * y                (y) int64 0 1 2
  * probability_bin  (probability_bin) float64 0.0 0.25 0.5 0.75 1.0
Data variables:
    relative_freq    (probability_bin, y) float64 0.0 0.5 0.0 ... 1.0 0.0 0.0
    cmp_number       (probability_bin, y) int64 0 2 1 2 0 1 0 0 0 0 1 1 1 0 0
    ref_occur        (probability_bin, y) int64 0 1 0 1 0 0 0 0 0 0 0 1 1 0 0
```

skill.**roc**(*cmp_likelihood*, *ref_logical*, *over_dims*, *probability_bins=array([0., 0.25, 0.5, 0.75, 1.])*)

Computes the relative operating characteristic of an event for a range of probability threshold bins given the comparison likelihood and reference logical event data

Author: Dougie Squire
Date: 10/05/2018

Parameters

cmp_likelihood [xarray DataArray] Array containing likelihoods of the event from the comparison data (e.g. cmp_likelihood = (da_cmp > 1).mean(dim='ensemble'))

ref_logical [xarray DataArray] Array containing logical (True/False) outcomes of the event from the reference data (e.g. ref_logical = (da_ref > 1))

over_dims [str or sequence of str] Dimensions over which to compute the relative operating characteristic

probability_bins [array_like, optional] Probability threshold bins. Defaults to 5 equally spaced bins between 0 and 1

Returns

roc [xarray DataSet]

Dataset containing the following variables:
hit_rate; the hit rate in each probability bin
false_alarm_rate; the false alarm rate in each probability bin
area; the area under the roc curve (false alarm rate vs hit rate)

Notes

See http://www.cawcr.gov.au/projects/verification/

To do

- Currently using a for-loop to process each probability bin separately. Is it possible to remove this loop?

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> cmp_likelihood = (da_cmp > 0.1).mean('ensemble')
>>> ref_logical = da_ref > 0.1
>>> doppyo.skill.roc(cmp_likelihood, ref_logical, over_dims='x')
<xarray.Dataset>
Dimensions:           (probability_bin: 5, y: 3)
Coordinates:
  * y                 (y) int64 0 1 2
  * probability_bin   (probability_bin) float64 0.0 0.25 0.5 0.75 1.0
Data variables:
    hit_rate          (probability_bin, y) float64 1.0 1.0 1.0 ... nan 0.0 0.0
    false_alarm_rate  (probability_bin, y) float64 1.0 1.0 1.0 ... 0.0 0.0 0.0
    area              (y) float64 0.0 0.0 0.0
```

`skill.`**`discrimination`**(*cmp_likelihood*, *ref_logical*, *over_dims*, *probability_bins=array([0., 0.25, 0.5, 0.75, 1.])*)

Returns the discrimination diagram of an event; the histogram of comparison likelihood when references indicate the event has occurred and has not occurred

Author: Dougie Squire

Date: 10/05/2018

Parameters

cmp_likelihood [xarray DataArray] Array containing likelihoods of the event from the comparison data (e.g. cmp_likelihood = (da_cmp > 1).mean(dim='ensemble'))

ref_logical [xarray DataArray] Array containing logical (True/False) outcomes of the event from the reference data (e.g. ref_logical = (da_ref > 1))

over_dims [str or sequence of str] Dimensions over which to compute the discrimantion histograms

probability_bins [array_like, optional] Probability threshold bins. Defaults to 5 equally spaced bins between 0 and 1

Returns

discrimination [xarray DataSet]

Dataset containing the following variables:

hist_event; histogram of comparison likelihoods when reference data indicates that the event has occurred

hist_no_event; histogram of comparison likelihoods when reference data indicates that the event has not occurred

Notes

See http://www.cawcr.gov.au/projects/verification/

To do

- Currently using a for-loop to process each probability bin separately. Is it possible to remove this loop?

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> cmp_likelihood = (da_cmp > 0.1).mean('ensemble')
>>> ref_logical = da_ref > 0.1
>>> doppyo.skill.discrimination(cmp_likelihood, ref_logical, over_dims='x')
<xarray.Dataset>
Dimensions:        (bins: 5, y: 3)
Coordinates:
  * bins           (bins) float64 0.0 0.25 0.5 0.75 1.0
  * y              (y) int64 0 1 2
Data variables:
    hist_event     (bins, y) float64 0.0 0.0 nan 0.5 1.0 ... 0.0 nan 0.0 0.0 nan
    hist_no_event  (bins, y) float64 0.0 0.0 0.0 1.0 ... 0.3333 0.0 0.0 0.3333
```

skill.**Brier_score**(*cmp_likelihood*, *ref_logical*, *over_dims*, *probability_bins=None*)

Computes the Brier score(s) of an event given the comparison likelihood and reference logical event data. When comparison probability bins are also provided, this function also computes the reliability, resolution and uncertainty components of the Brier score, where Brier = reliability - resolution + uncertainty

Author: Dougie Squire

Date: 10/05/2018

Parameters

cmp_likelihood [xarray DataArray] Array containing likelihoods of the event from the comparison data (e.g. cmp_likelihood = (da_cmp > 1).mean(dim='ensemble'))

ref_logical [xarray DataArray] Array containing logical (True/False) outcomes of the event from the reference data (e.g. ref_logical = (da_ref > 1))

over_dims [str or sequence of str] Dimensions over which to compute the Brier score

probability_bins [array_like, optional] Probability threshold bins. If specified, this function also computes the reliability, resolution and uncertainty components of the Brier score. Defaults to None

Returns

Brier [xarray DataArray or xarray DataSet] If probability_bins = None, returns a DataArray containing Brier scores. Otherwise returns a DataSet containing the reliability, resolution and uncertainty components of the Brier score, where Brier = reliability - resolution + uncertainty

Notes

See http://www.cawcr.gov.au/projects/verification/

To do

- Currently using a for-loop to process each probability bin separately. Is it possible to remove this loop?

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3)),
...                               ('ensemble', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> cmp_likelihood = (da_cmp > 0.1).mean('ensemble')
>>> ref_logical = da_ref > 0.1
>>> doppyo.skill.Brier_score(cmp_likelihood, ref_logical, over_dims='x')
<xarray.DataArray (y: 3)>
array([0.148148, 0.444444, 0.222222])
Coordinates:
  * y        (y) int64 0 1 2
```

`skill.`**`contingency`**(*da_cmp*, *da_ref*, *category_edges_cmp*, *category_edges_ref*, *over_dims*)

Return the contingency table between da_cmp and da_ref for given categories

Author: Dougie Squire
Date: 12/05/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

category_edges_cmp [array_like] Bin edges for categorising da_cmp

category_edges_ref [array_like] Bin edges for categorising da_ref

over_dims [str or sequence of str, optional] Dimensions over which to compute the contingency table

Returns

contingency [xarray DataArray] Contingency table of input data

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,5)
>>> category_edges_ref = np.linspace(-2,2,5)
doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                         category_edges_ref, over_dims=['x','y'])
<xarray.DataArray 'contingency' (comparison_category: 4, reference_category: 4)>
array([[0, 1, 0, 1],
       [1, 0, 1, 0],
       [0, 2, 1, 0],
       [0, 0, 0, 0]])
Coordinates:
  * comparison_category  (comparison_category) int64 1 2 3 4
  * reference_category   (reference_category) int64 1 2 3 4
```

`skill.`**`accuracy_score`**`(`*contingency*`)`

Returns the accuracy score given a contingency table

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A contingency table of the form output from doppyo.skill.contingency

Returns

accuracy_score [xarray DataArray] An array containing the accuracy scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,5)
>>> category_edges_ref = np.linspace(-2,2,5)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                           category_edges_ref, over_dims='y')
>>> doppyo.skill.accuracy_score(contingency)
<xarray.DataArray 'accuracy_score' (x: 3)>
array([0.       , 0.333333, 0.333333])
Coordinates:
  * x        (x) int64 0 1 2
```

`skill.`**`Heidke_score`**`(`*contingency*`)`

Returns the Heidke skill score given a contingency table

Author: Dougie Squire
Date: 12/05/2018

Parameters

contingency [xarray DataArray] A contingency table of the form output from doppyo.skill.contingency

Returns

Heidke_score [xarray DataArray] An array containing the Heidke scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,5)
>>> category_edges_ref = np.linspace(-2,2,5)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.Heidke_score(contingency)
<xarray.DataArray 'Heidke_score' (x: 3)>
array([-0.285714,  0.      ,  0.142857])
Coordinates:
  * x        (x) int64 0 1 22
```

`skill.`**`Peirce_score`**(*contingency*)

Returns the Peirce score (also called Hanssen and Kuipers discriminant) given a contingency table

Author: Dougie Squire
Date: 12/05/2018

Parameters

contingency [xarray DataArray] A contingency table of the form output from doppyo.skill.contingency

Returns

Peirce_score [xarray DataArray] An array containing the Peirce scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,5)
>>> category_edges_ref = np.linspace(-2,2,5)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                            category_edges_ref, over_dims='y')
>>> doppyo.skill.Peirce_score(contingency)
<xarray.DataArray 'Peirce_score' (x: 3)>
array([-0.25,  0.  , -0.5 ])
Coordinates:
  * x        (x) int64 0 1 2
```

`skill.`**`Gerrity_score`**`(`*contingency*`)`

Returns Gerrity equitable score given a contingency table

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A contingency table of the form output from doppyo.skill.contingency

Returns

Gerrity_score [xarray DataArray] An array containing the Gerrity scores

Notes

See http://www.cawcr.gov.au/projects/verification/

To do

- Currently computes the Gerrity scoring matrix using nested for-loops. Is it possible to remove these?

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,5)
>>> category_edges_ref = np.linspace(-2,2,5)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                            category_edges_ref, over_dims='y')
>>> doppyo.skill.Gerrity_score(contingency)
<xarray.DataArray 'Gerrity_score' (x: 3)>
array([-2.777778e-01,  0.000000e+00, -5.551115e-17])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**bias_score**(*contingency*, *yes_category=2*)

Returns the bias score given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

yes_category [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

bias_score [xarray DataArray] An array containing the bias scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.bias_score(contingency)
<xarray.DataArray 'bias_score' (x: 3)>
array([0.5     , 0.333333, 1.      ])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**hit_rate**(*contingency*, *yes_category=2*)

Returns the hit rate (probability of detection) given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

yes_category [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

hit_rate [xarray DataArray] An array containing the hit rates

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.hit_rate(contingency)
<xarray.DataArray 'hit_rate' (x: 3)>
array([ 0., nan,  1.])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**false_alarm_ratio**(*contingency*, *yes_category=2*)

Returns the false alarm ratio given dichotomous contingency data

Author: Dougie Squire
Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

yes_category [value, optional] The coordinate value of the category corresponding to ‘yes’

Returns

false_alarm_ratio [xarray DataArray] An array containing the false alarm ratios

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.false_alarm_ratio(contingency)
<xarray.DataArray 'false_alarm_ratio' (x: 3)>
```

(continues on next page)

(continued from previous page)

```
array([nan, nan,  0.])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**false_alarm_rate**(*contingency*, *yes_category=2*)

Returns the false alarm rate given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

yes_category [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

false_alarm_rate [xarray DataArray] An array containing the false alarm rates

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.false_alarm_rate(contingency)
<xarray.DataArray 'false_alarm_rate' (x: 3)>
array([ 0.,  0., nan])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**success_ratio**(*contingency*, *yes_category=2*)

Returns the success ratio given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

success_ratio [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

success_ratio [xarray DataArray] An array containing the success ratios

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                          category_edges_ref, over_dims='y')
>>> doppyo.skill.success_ratio(contingency)
<xarray.DataArray 'success_ratio' (x: 3)>
array([nan, nan,  1.])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**threat_score**(*contingency*, *yes_category=2*)

Returns the threat score given dichotomous contingency data

Author: Dougie Squire
Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

success_ratio [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

threat_score [xarray DataArray] An array containing the threat scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                           coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
```

(continues on next page)

(continued from previous page)

```
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.threat_score(contingency)
<xarray.DataArray 'threat_score' (x: 3)>
array([0. , 0. , 0.5])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**equit_threat_score**(*contingency*, *yes_category=2*)

Returns the equitable threat score given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

success_ratio [value, optional] The coordinate value of the category corresponding to ‘yes’

Returns

equit_threat_score [xarray DataArray] An array containing the equitable threat scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.equit_threat_score(contingency)
<xarray.DataArray 'equit_threat_score' (x: 3)>
array([0., 0., 0.])
Coordinates:
  * x        (x) int64 0 1 2
```

skill.**odds_ratio**(*contingency*, *yes_category=2*)

Returns the odds ratio given dichotomous contingency data

Author: Dougie Squire

Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

success_ratio [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

odds_ratio [xarray DataArray] An array containing the equitable odds ratios

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                        category_edges_ref, over_dims='y')
>>> doppyo.skill.odds_ratio(contingency)
<xarray.DataArray 'odds_ratio' (x: 3)>
array([ 0.,  0., nan])
Coordinates:
  * x        (x) int64 0 1 2
```

`skill.`**`odds_ratio_skill_score`**(*contingency*, *yes_category=2*)

Returns the odds ratio skill score given dichotomous contingency data

Author: Dougie Squire
Date: 12/05/2018

Parameters

contingency [xarray DataArray] A 2 category contingency table of the form output from doppyo.skill.contingency

success_ratio [value, optional] The coordinate value of the category corresponding to 'yes'

Returns

odds_ratio_skill_score [xarray DataArray] An array containing the equitable odds ratio skill scores

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> category_edges_cmp = np.linspace(-2,2,3)
>>> category_edges_ref = np.linspace(-2,2,3)
>>> contingency = doppyo.skill.contingency(da_cmp, da_ref, category_edges_cmp,
...                                         category_edges_ref, over_dims='y')
>>> doppyo.skill.odds_ratio_skill_score(contingency)
<xarray.DataArray 'odds_ratio_skill' (x: 3)>
array([-1., -1., nan])
Coordinates:
  * x         (x) int64 0 1 2
```

`skill.`**`mean_additive_bias`**(*da_cmp*, *da_ref*, *over_dims*)

Returns the additive bias between comparison and reference datasets

Author: Dougie Squire

Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the mean additive bias

Returns

mean_additive_bias [xarray DataArray] Array containing the mean additive biases

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.skill.mean_additive_bias(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'mean_additive_bias' (y: 3)>
array([0.328462, 0.172263, 0.402438])
Coordinates:
  * y         (y) int64 0 1 2
```

`skill.`**`mean_multiplicative_bias`**(*da_cmp*, *da_ref*, *over_dims*)

Returns the multiplicative bias between comparison and reference datasets

Author: Dougie Squire
Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the mean multiplicative bias

Returns

mean_multiplicative_bias [xarray DataArray] Array containing the mean multiplicative biases

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.skill.mean_multiplicative_bias(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'mean_multiplicative_bias' (y: 3)>
array([ 2.108882,  4.356835, -0.83234 ])
Coordinates:
  * y        (y) int64 0 1 2
```

`skill.`**`mean_absolute_error`**(*da_cmp*, *da_ref*, *over_dims*)

Returns the mean absolute error between comparison and reference datasets

Author: Dougie Squire
Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the mean absolute error

Returns

mean_absolute_error [xarray DataArray] Array containing the mean absolute error

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.skill.mean_absolute_error(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'mean_absolute_error' (y: 3)>
array([1.030629, 1.265555, 0.770711])
Coordinates:
  * y        (y) int64 0 1 2
```

`skill.`**`mean_squared_error`**(*da_cmp*, *da_ref*, *over_dims*)

Returns the mean sqaured error between comparison and reference datasets

Author: Dougie Squire
Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the mean squared error

Returns

mean_squared_error [xarray DataArray] Array containing the mean squared error

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                          coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.skill.mean_squared_error(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'mean_squared_error' (y: 3)>
array([1.257412, 1.725008, 0.721863])
Coordinates:
  * y        (y) int64 0 1 2
```

`skill.`**`rms_error`**(*da_cmp*, *da_ref*, *over_dims*)

Returns the root mean sqaured error between comparison and reference datasets

Author: Dougie Squire
Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the root mean squared error

Returns

rms_error [xarray DataArray] Array containing the root mean squared error

Notes

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(3,3)),
...                       coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.skill.rms_error(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'root_mean_squared_error' (y: 3)>
array([1.964753, 1.426566, 1.20612 ])
Coordinates:
  * y        (y) int64 0 1 2
```

`skill.`**`Pearson_corrcoeff`**(*da_cmp*, *da_ref*, *over_dims*, *subtract_local_mean=True*)

Returns the Pearson correlation coefficients over the specified dimensions.

Author: Dougie Squire
Date: 28/04/2018

Parameters

da_cmp [xarray DataArray] Array containing data to be compared to reference dataset (usually forecasts)

da_ref [xarray DataArray] Array containing reference data (usually observations)

over_dims [str or sequence of str, optional] Dimensions over which to compute the correlation coefficients

subtract_local_mean [bool, optional] If True, this function will subtract the mean computed over over_dims. Otherwise, no mean field is removed prior to computing the correlation

Returns

Pearson_corrcoeff [xarray DataArray] Array containing the Pearson correlation coefficients

Notes

If any dimensions in over_dims do not exist in either da_cmp or da_ref, the correlation is computed over all dimensions in over_dims that appear in both da_cmp and da_ref, and then averaged over any remaining dimensions in over_dims

See http://www.cawcr.gov.au/projects/verification/

Examples

```
>>> da_cmp = xr.DataArray(np.random.normal(size=(100,3)),
...                       coords=[('x', np.arange(100)), ('y', np.arange(3))])
>>> da_ref = xr.DataArray(np.random.normal(size=(100,3)),
...                       coords=[('x', np.arange(100)), ('y', np.arange(3))])
>>> doppyo.skill.Pearson_corrcoeff(da_cmp, da_ref, over_dims='x')
<xarray.DataArray 'Pearson_corrcoeff' (y: 3)>
array([-0.040584, -0.037983, -0.020941])
Coordinates:
  * y        (y) int64 0 1 2
```

skill.**sign_test**(*da_cmp1*, *da_cmp2*, *da_ref*, *time_dim='init_date'*)

Returns the Delsole and Tippett sign test over the given time period

Author: Dougie Squire
Date: 26/03/2019

Parameters

da_cmp1 [xarray DataArray] Array containing data to be compared to da_cmp1

da_cmp2 [xarray DataArray] Array containing data to be compared to da_cmp2

da_ref [xarray DataArray] Array containing data to use as reference

time_dim [str, optional] Name of dimension over which to compute the random walk

Returns

sign_test [xarray DataArray] Array containing the results of the sign test

confidence [xarray DataArray] Array containing 95% confidence bounds

Notes

See Delsole and Tippett 2016 *Forecast Comparison Based on Random Walks*

Examples

```
>>> x = xr.DataArray(np.random.normal(size=(3,3)),
...                  coords=[('t', np.arange(3)), ('x', np.arange(3))])
>>> y = xr.DataArray(np.random.normal(size=(3,3)),
...                 coords=[('t', np.arange(3)), ('x', np.arange(3))])
>>> o = xr.DataArray(np.random.normal(size=(3,3)),
...                  coords=[('t', np.arange(3)), ('x', np.arange(3))])
```

(continues on next page)

(continued from previous page)

```
>>> walk, confidence = sign_test(x, y, o, time_dim='t')
>>> walk
<xarray.DataArray (t: 3, x: 3)>
array([[-1, -1, -1],
       [ 0,  0, -2],
       [-1, -1, -3]])
Coordinates:
  * t        (t) int64 0 1 2
  * x        (x) int64 0 1 2
```

3.3 utils

General support functions for the doppyo package

3.3.1 API

class utils.**timer**(*name=None*)

Reports time taken to complete code snippets.

Author: Dougie Squire
Date: 14/02/2018

Examples

```
>>> with doppyo.utils.timer():
>>>     x = 1 + 1
Elapsed: 4.5299530029296875e-06 sec
```

class utils.**constants**

Returns commonly used constants.

Author: Dougie Squire
Date: 14/02/2018

Examples

```
>>> pi = doppyo.utils.constants().pi
```

Attributes

C_l

C_pd

C_pv

C_vd

C_vv

Ce

Omega

R_d

R_earth

R_v

g

pi

utils.**skewness**(*da*, *dim*)

Returns the skewness along dimension dim

Author: Dougie Squire
Date: 20/08/2018

Parameters

da [xarray DataArray] Array containing values for which to compute skewness

dim [str or sequence of str] Dimension(s) over which to compute the skewness

Returns

skewness [xarray DataArray] New DataArray object with skewness applied to its data and the indicated dimension(s) removed

Examples

```
>>> arr = xr.DataArray(np.arange(6).reshape(2, 3),
...                     coords=[('x', ['a', 'b']), ('y', [0, 1, 2])])
>>> arr
<xarray.DataArray (x: 2, y: 3)>
array([[0, 1, 2],
       [3, 4, 5]])
Coordinates:
  * x        (x) <U1 'a' 'b'
  * y        (y) int64 0 1 2
>>> doppyo.utils.skewness(arr, 'x')
<xarray.DataArray (y: 3)>
array([0., 0., 0.])
Coordinates:
  * y        (y) int64 0 1 2
```

utils.**kurtosis**(*da*, *dim*)

Returns the kurtosis along dimension dim

Author: Dougie Squire
Date: 20/08/2018

Parameters

da [xarray DataArray] Array containing values for which to compute kurtosis

dim [str or sequence of str] Dimension(s) over which to compute the kurtosis

Returns

kurtosis [xarray DataArray] New DataArray object with kurtosis applied to its data and the indicated dimension(s) removed

Examples

```
>>> arr = xr.DataArray(np.arange(6).reshape(2, 3),
...                     coords=[('x', ['a', 'b']), ('y', [0, 1, 2])])
>>> arr
<xarray.DataArray (x: 2, y: 3)>
array([[0, 1, 2],
       [3, 4, 5]])
Coordinates:
  * x        (x) <U1 'a' 'b'
  * y        (y) int64 0 1 2
>>> doppyo.utils.kurtosis(arr, 'x')
<xarray.DataArray (y: 3)>
array([1., 1., 1.])
Coordinates:
  * y        (y) int64 0 1 2
```

utils.**digitize**(*da*, *bin_edges*)

Returns the indices of the bins to which each value in input array belongs.

Author: Dougie Squire
Date: 31/10/2018

Parameters

da [xarray DataArray] Array containing values to digitize

dim [array_like] Array of bin edges. Output indices, i, are such that bin_edges[i-1] <= x < bin_edges[i]

Returns

digitized [xarray DataArray] New DataArray object of indices

Examples

```
>>> da = xr.DataArray(np.random.normal(size=(20,40)), coords=[('x', np.
↪arange(20)),
...                                                            ('y', np.
↪arange(40))])
>>> bins = np.linspace(-2,2,10)
>>> bin_edges = doppyo.utils.get_bin_edges(bins)
>>> doppyo.utils.digitize(da, bin_edges)
<xarray.DataArray 'digitized' (x: 20, y: 40)>
```

(continues on next page)

(continued from previous page)

```
array([[ 7,  6,  4, ...,  5,  6,  7],
       [ 5, 11,  2, ...,  7,  6,  0],
       [ 9,  3,  2, ...,  6,  5,  6],
       ...,
       [11, 10,  8, ...,  6,  5,  2],
       [ 3, 10,  3, ...,  8,  7,  7],
       [ 5,  4,  9, ...,  5,  5,  7]])
Coordinates:
  * x        (x) int64 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
  * y        (y) int64 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...
```

utils.**pdf**(*da*, *bin_edges*, *over_dims*)

Returns the probability distribution function along the specified dimensions

Author: Dougie Squire

Date: 01/10/2018

Parameters

da [xarray DataArray] Array containing values used to compute the pdf

bin_edges [array_like] The bin edges, including the rightmost edge

over_dims [str or sequence of str] Dimension(s) over which to compute the pdf

Returns

pdf [xarray DataArray] New DataArray object containing pdf

Notes

This function uses `doppyo.utils.histogram()` which uses `xr.groupby_bins` when over_dims is a subset of da.dims and is therefore not parallelized in these cases. There are efforts underway to parallelize groupby operations in xarray, see https://github.com/pydata/xarray/issues/585

Examples

```
>>> da = xr.DataArray(np.random.normal(size=(100,100)), coords=[('x', np.
↪arange(100)), ('y', np.arange(100))])
>>> bins = np.linspace(-2,2,10)
>>> bin_edges = doppyo.utils.get_bin_edges(bins)
>>> doppyo.utils.pdf(da, bin_edges=bin_edges, over_dims='x')
<xarray.DataArray (bins: 10, y: 100)>
array([[0.069588, 0.046392, 0.046875, ..., 0.090909, 0.070312, 0.090909],
       [0.208763, 0.255155, 0.140625, ..., 0.113636, 0.117187, 0.113636],
       [0.278351, 0.115979, 0.304688, ..., 0.25    , 0.234375, 0.227273],
       ...,
       [0.115979, 0.255155, 0.46875 , ..., 0.25    , 0.210937, 0.136364],
       [0.046392, 0.139175, 0.117188, ..., 0.090909, 0.1875  , 0.136364],
       [0.046392, 0.069588, 0.046875, ..., 0.022727, 0.070312, 0.068182]])
Coordinates:
  * bins     (bins) float64 -2.0 -1.556 -1.111 -0.6667 -0.2222 0.2222 0.6667 ...
  * y        (y) int64 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...
```

`utils.`**`cdf`**(*da*, *bin_edges*, *over_dims*)

Returns the cumulative probability distribution function along the specified dimensions

Author: Dougie Squire
Date: 01/10/2018

Parameters

da [xarray DataArray] Array containing values used to compute the cdf

bin_edges [array_like] The bin edges, including the rightmost edge

over_dims [str or sequence of str] Dimension(s) over which to compute the cdf

Returns

cdf [xarray DataArray] New DataArray object containing cdf

Notes

This function uses `doppyo.utils.histogram()` which uses `xr.groupby_bins` when over_dims is a subset of da.dims and is therefore not parallelized in these cases. There are efforts underway to parallelize groupby operations in xarray, see https://github.com/pydata/xarray/issues/585

Examples

```
>>> da = xr.DataArray(np.random.normal(size=(100,100)), coords=[('x', np.
↪arange(100)), ('y', np.arange(100))])
>>> bins = np.linspace(-2,2,10)
>>> bin_edges = doppyo.utils.get_bin_edges(bins)
>>> doppyo.utils.cdf(da, bin_edges=bin_edges, over_dims='x')
<xarray.DataArray (bins: 10, y: 100)>
array([[0.050505, 0.      , 0.030612, ..., 0.020202, 0.010204, 0.020619],
       [0.121212, 0.085106, 0.081633, ..., 0.080808, 0.081633, 0.061856],
       [0.232323, 0.138298, 0.142857, ..., 0.171717, 0.183673, 0.195876],
       ...,
       [0.939394, 0.925532, 0.908163, ..., 0.909091, 0.94898 , 0.907216],
       [0.979798, 0.968085, 0.969388, ..., 0.959596, 0.979592, 0.989691],
       [1.      , 1.      , 1.      , ..., 1.      , 1.      , 1.      ]])
Coordinates:
  * bins     (bins) float64 -2.0 -1.556 -1.111 -0.6667 -0.2222 0.2222 0.6667 ...
  * y        (y) int64 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...
```

`utils.`**`Gaussian_pdf`**(*x*)

Evaluate standard Gaussian pdf at x

Author: Dougie Squire
Date: 26/03/2019

Parameters

x [array_like, float] Point(s) at which to evaluate the Gaussian pdf

Returns

p [array_like, float] Same shape as x containing the evaluated pdf values

Examples

```
>>> doppyo.utils.gaussian_pdf(0.5)
0.3520653267642995
```

utils.**Gaussian_cdf**(*x*, *n=100*)

Evaluate standard Gaussian cdf at x (numerical integral over n points)

Author: Dougie Squire
Date: 26/03/2019

Parameters

x [array_like, float] Point(s) at which to evaluate the Gaussian cdf

n [int, optional] Number of points to use to evaulate the numerical integral

Returns

p [xarray DataArray] Same shape as x containing the evaluated cdf values

Examples

```
>>> doppyo.utils.gaussian_cdf(0.5)
<xarray.DataArray 'integral' ()>
array(0.691462)
```

utils.**histogram**(*da*, *bin_edges*, *over_dims*)

Returns the histogram over the specified dimensions

Author: Dougie Squire
Date: 01/10/2018

Parameters

da [xarray DataArray] Array containing values used to compute the histogram

bin_edges [array_like] The bin edges, including the rightmost edge

over_dims [str or sequence of str] Dimension(s) over which to compute the histogram

Returns

histogram [xarray DataArray] New DataArray object containing the histogram

See also:

`numpy.histogram`, `dask.array.histogram`

Notes

This function uses `xr.groupby_bins` when over_dims is a subset of da.dims and is therefore not parallelized/lazy in these cases. There are efforts underway to parallelize groupby operations in xarray, see https://github.com/pydata/xarray/issues/585

Examples

```
>>> da = xr.DataArray(np.random.normal(size=(100,100)),
...                   coords=[('x', np.arange(100)), ('y', np.arange(100))])
>>> bins = np.linspace(-2,2,10)
>>> bin_edges = doppyo.utils.get_bin_edges(bins)
>>> doppyo.utils.histogram(da, bin_edges=bin_edges, over_dims='x')
<xarray.DataArray 'data' (bins: 10, y: 100)>
array([[ 3.,  1.,  6., ...,  2.,  4.,  3.],
       [ 2., 12.,  4., ...,  7.,  3.,  7.],
       [ 9.,  9., 11., ..., 19., 13.,  6.],
       ...,
       [13.,  9.,  4., ...,  6.,  6., 11.],
       [ 3.,  6.,  3., ...,  3.,  7.,  4.],
       [ 2.,  0.,  1., ...,  3.,  3.,  4.]])
Coordinates:
  * bins     (bins) float64 -2.0 -1.556 -1.111 -0.6667 -0.2222 0.2222 0.6667 ...
  * y        (y) int64 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...
```

`utils.`**`get_bin_edges`**(*bins*)

Returns bin edges of provided bins

Author: Dougie Squire

Date: 06/03/2018

Parameters

bins [array_like] One-dimensional array of bin values to compute bin edges

Returns

edges [array_like] Array of bin edges where the first and last edge are computed using the spacing between the first-and-second and second-last-and-last bins, respectively. This array is one element larger than the input array

Examples

```
>>> bins = np.linspace(-2,2,10)
>>> bin_edges = doppyo.utils.get_bin_edges(bins)
array([-2.5, -1.5, -0.5,  0.5,  1.5,  2.5])
```

`utils.`**`polyfit`**(*x*, *y*, *order*, *over_dims*)

Returns least squares polynomial fit of the specified order

Author: Dougie Squire

Date: 25/03/2019

Parameters

x [xarray DataArray] Array containing x-coordinates of the sample points

y [xarray DataArray] Array containing y-coordinates of the sample points

over_dims [str or sequence of str, optional] Dimensions over which to compute the fit

Returns

coefficients [xarray DataArray] Array containing the coefficients of the fit

See also:

`numpy.polyfit`

Examples

```
>>> x = xr.DataArray(np.random.normal(size=(3,3,3)),
...                  coords=[('x', np.arange(3)), ('y', np.arange(3)), ('z', np.
↪arange(3))])
>>> y = xr.DataArray(np.random.normal(size=(3,3,3)),
...                  coords=[('x', np.arange(3)), ('y', np.arange(3)), ('z', np.
↪arange(3))])
>>> doppyo.utils.polyfit(x, y, order=2, over_dims=['x','y'])
<xarray.DataArray 'fit' (z: 3, degree: 3)>
array([[-0.17262 ,  0.423925,  0.644511],
       [-0.217319,  0.033146,  1.139249],
       [ 0.520737, -0.641632, -1.095306]])
Coordinates:
  * z         (z) int64 0 1 2
  * degree    (degree) int64 0 1 2
```

`utils.`**`polyval`**(*x*, *p*, *over_dims*)

Evaluate a polynomial at specific values

Author: Dougie Squire
Date: 25/03/2019

Parameters

x [xarray DataArray] Array containing x-coordinates of the sample points

p [xarray DataArray] Array containing the polynomial coefficients

over_dims [str or sequence of str, optional] Dimensions over which to compute the fit. Should match over_dims handed to polyfit

Returns

fit [xarray DataArray] Evaluated polynomial values

See also:

`numpy.polyval`

Examples

```
>>> x = xr.DataArray(np.random.normal(size=(3,3,3)),
...                  coords=[('x', np.arange(3)), ('y', np.arange(3)), ('z', np.
↪arange(3))])
>>> y = xr.DataArray(np.random.normal(size=(3,3,3)),
...                  coords=[('x', np.arange(3)), ('y', np.arange(3)), ('z', np.
↪arange(3))])
>>> p = polyfit(x, y, order=2, over_dims=['x','y'])
>>> xf = xr.DataArray(np.random.normal(size=(3,3)),
...                   coords=[('x', np.arange(3)), ('y', np.arange(3))])
>>> doppyo.utils.polyval(p, xf, over_dims=['x','y'])
<xarray.DataArray (z: 3, x: 3, y: 3)>
array([[[ 0.680063,  0.687067,  0.37378 ],
        [ 0.68157 ,  0.593822,  0.564625],
        [ 0.661166,  0.651231,  0.373721]],
...
       [[-0.526172, -0.343878,  1.040153],
        [-0.310027, -0.374777,  0.251314],
        [-0.197566, -0.488106,  1.04039 ]],
...
       [[ 1.009727, -0.11369 ,  0.517851],
        [-0.152397,  2.377984, -0.149194],
        [-0.227379,  1.532419,  0.518102]]])
Coordinates:
  * z        (z) int64 0 1 2
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1 2
```

`utils.`**`differentiate_wrt`**(*da*, *dim*, *x*)

Returns the gradient along dim using x to compute differences. This function is required because the current implementation of xr.differentiate (0.10.9) can only differentiate with respect to a 1D coordinate. It is common to want to differentiate with respect to something that changes as a function of multiple dimensions (e.g. the zonal distance between regularly spaced lat/lon points varies as a function of lat and lon). Uses second order accurate central differencing in the interior points and first order accurate one-sided (forward or backwards) differencing at the boundaries.

Author: Dougie Squire
Date: 02/11/2018

Parameters

da [xarray DataArray] Array containing values to differentiate

dim [str] The dimension to be used to compute the gradient

x [xarray DataArray] Array containing values to differentiate with respect to. Must be broadcastable with da

Returns

differentiated [xarray DataArray] New DataArray object containing the differentiate data

See also:

`xarray.DataArray.differentiate`, `numpy.gradient`

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(180,360)), coords=[('lat', np.arange(-
↪90,90,1)), ('lon', np.arange(0,360,1))])
>>> x, y = doppyo.utils.xy_from_lonlat(A['lon'], A['lat'])
>>> differentiate_wrt(A, dim='lon', x=x)
<xarray.DataArray 'differentiated' (lat: 180, lon: 360)>
array([[ 3.674336e+10, -9.015981e+10, -2.150203e+11, ..., -6.471076e+10,
        -6.057067e+09,  1.253664e+11],
       [-4.133347e-07, -3.932972e-04, -5.982892e-04, ...,  2.972605e-04,
         9.456351e-04,  1.907131e-03],
       [-6.596434e-04,  6.147016e-06,  2.370071e-04, ..., -8.578490e-06,
         9.281731e-06,  2.211755e-04],
       ...,
       [-6.467389e-05,  6.315746e-05,  1.713705e-04, ...,  9.742767e-05,
         1.043358e-04,  1.066228e-04],
       [ 1.542484e-04,  2.802838e-04,  5.511727e-05, ...,  1.665500e-04,
        -6.087167e-06, -3.060961e-04],
       [-5.991109e-04,  2.085148e-04,  4.525132e-04, ..., -9.346556e-05,
        -7.977593e-05,  3.411080e-05]])
Coordinates:
  * lat      (lat) int64 -90 -89 -88 -87 -86 -85 -84 ... 83 84 85 86 87 88 89
  * lon      (lon) int64 0 1 2 3 4 5 6 7 8 ... 352 353 354 355 356 357 358 359
```

utils.**xy_from_lonlat**(*lon*, *lat*)

Returns x/y in m from grid points that are in a longitude/latitude format.

Author: Dougie Squire

Date: 01/11/2018

Parameters

lon [xarray DataArray] Array containing longitudes stored relative to longitude dimension/coordinate

lat [xarray DataArray] Array containing latitudes stored relative to latitude dimension/coordinate

Returns

x [xarray DataArray] Array containing zonal distance in m

y [xarray DataArray] Array containing meridional distance in m

Examples

```
>>> lat = xr.DataArray(np.arange(-90,90,90), dims=['lat'])
>>> lon = xr.DataArray(np.arange(0,360,90), dims=['lon'])
>>> doppyo.utils.xy_from_lonlat(lon=lon, lat=lat)
(<xarray.DataArray (lat: 2, lon: 4)>
 array([[0.000000e+00, 6.127853e-10, 1.225571e-09, 1.838356e-09],
        [0.000000e+00, 1.000754e+07, 2.001509e+07, 3.002263e+07]])
 Dimensions without coordinates: lat, lon, <xarray.DataArray (lat: 2, lon: 4)>
 array([[-10007543.39801, -10007543.39801, -10007543.39801, -10007543.39801],
        [        0.     ,         0.     ,         0.     ,         0.     ]])
 Dimensions without coordinates: lat, lon)
```

`utils.`**`integrate`**(*da*, *over_dim*, *x=None*, *dx=None*, *method='trapz'*, *cumulative=False*, *skipna=False*)

Returns trapezoidal/rectangular integration along specified dimension

Author: Dougie Squire
Date: 16/08/2018

Parameters

da [xarray DataArray] Array containing values to integrate

over_dim [str] Dimension to integrate

x [xarray DataArray, optional] Values to use for integrand. Must contain dimensions over_dim. If None, x is determined from the coords associated with over_dim

dx [value, optional] Integrand spacing used to compute the integral. If None, dx is determined from x

method [str, optional] Method of performing integral. Options are 'trapz' for trapezoidal integration, or 'rect' for rectangular integration

cumulative [bool, optional] If True, return the cumulative integral

Returns

integral [xarray DataArray] Array containing the integral along the specified dimension

See also:

`numpy.trapz`

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)),
...                                                          ('y', np.arange(2))])
>>> doppyo.utils.integrate(A, over_dim='x')
<xarray.DataArray 'integral' (y: 2)>
array([-0.20331 , -0.781251])
Coordinates:
  * y        (y) int64 0 1
```

`utils.`**`add`**(*data_1*, *data_2*)

Returns the addition of two arrays, data_1 + data_2. Useful for xr.apply type operations

Author: Dougie Squire
Date: 27/06/2018

Parameters

data_1 [array_like] The first array

data_2 [array_like] The second array

Returns

addition [array_like] The addition of data_1 and data_2

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> B = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> doppyo.utils.add(A,B)
<xarray.DataArray (x: 3, y: 2)>
array([[-0.333176,  0.344428],
       [ 0.629463,  0.515872],
       [ 1.121926,  0.567797]])
Coordinates:
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1
```

`utils.`**`subtract`**(*data_1*, *data_2*)

Returns the difference of two arrays, data_1 - data_2. Useful for xr.apply type operations

Author: Dougie Squire
Date: 27/06/2018

Parameters

data_1 [array_like] The first array

data_2 [array_like] The second array

Returns

subtraction [array_like] The difference between data_1 and data_2

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> B = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> doppyo.utils.subtract(A,B)
<xarray.DataArray (x: 3, y: 2)>
array([[-0.265376,  1.331496],
       [ 1.065077, -1.278974],
       [ 3.691209, -1.928883]])
Coordinates:
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1
```

`utils.`**`multiply`**(*data_1*, *data_2*)

Returns the multiplication of two fields, data_1 * data_2. Useful for xr.apply type operations

Author: Dougie Squire
Date: 27/06/2018

Parameters

data_1 [array_like] The first array

data_2 [array_like] The second array

Returns

multiplication [array_like] The multiplication of data_1 and data_2

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> B = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> doppyo.utils.multiply(A,B)
<xarray.DataArray (x: 3, y: 2)>
array([[-0.219773,  0.235889],
       [-0.529542, -1.30342 ],
       [-1.048924,  0.20482 ]])
Coordinates:
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1
```

`utils.`**`divide`**(*data_1*, *data_2*)

Returns the division of two fields, data_1 / data_2. Useful for xr.apply type operations

Author: Dougie Squire
Date: 27/06/2018

Parameters

data_1 [array_like] The first array

data_2 [array_like] The second array

Returns

division [array_like] The division of data_1 by data_2

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> B = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)), (
↪'y', np.arange(2))])
>>> doppyo.utils.divide(A,B)
<xarray.DataArray (x: 3, y: 2)>
array([[-0.310139,  0.071369],
       [-0.647227, -0.427525],
       [-0.179623,  1.229811]])
Coordinates:
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1
```

`utils.`**`average`**(*da*, *dim=None*, *weights=None*)

Returns the weighted average

Author: Dougie Squire
Date: 06/08/2018

Parameters

da [xarray DataArray] Array to be averaged

dim [str or sequence of str, optional] Dimension(s) over which to compute weighted average. If None, average is computed over all dimensions

weights [xarray DataArray, optional] Weights to apply during averaging. Shape of weights must be broadcastable to shape of da. If None, unity weighting is applied

Returns

weighted [xarray DataArray] Weighted average of input array along specified dimensions

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(4,4)), coords=[('lat', np.arange(-90,
↪90,45)),
...                                                         ('lon', np.arange(0,
↪360,90))])
>>> degtorad = doppyo.utils.constants().pi / 180
>>> cos_lat = xr.ufuncs.cos(A['lat'] * degtorad)
>>> doppyo.utils.average(A, dim='lat', weights=cos_lat)
<xarray.DataArray (lon: 4)>
array([-0.473632, -0.241208, -0.954826,  0.498559])
Coordinates:
  * lon      (lon) int64 0 90 180 270
```

`utils.`**`fft`**(*da*, *dim*, *nfft=None*, *dx=None*, *twosided=False*, *shift=True*)

Returns the sequential ffts of the provided array along the specified dimensions

Author: Dougie Squire
Date: 06/08/2018

Parameters

da [xarray.DataArray] Array from which compute the fft

dim [str or sequence] Dimensions along which to compute the fft

nfft [float or sequence, optional] Number of points in each dimensions to use in the transformation. If None, the full length of each dimension is used.

dx [float or sequence, optional] Define the spacing of the dimensions. If None, the spacing is computed directly from the coordinates associated with the dimensions. If dx is a time array, frequencies are computed in Hz

twosided [bool, optional] When the DFT is computed for purely real input, the output is Hermitian-symmetric, meaning the negative frequency terms are just the complex conjugates of the corresponding positive-frequency terms, and the negative-frequency terms are therefore redundant. If True, force the fft to include negative and positive frequencies, even if the input data is real. If the input array is complex, one must set twosided=True

shift [bool, optional] If True, the frequency axes are shifted to center the 0 frequency, otherwise negative frequencies follow positive frequencies as in `numpy.fft.ftt`

Returns

fft [xarray DataArray] Array containing the sequential ffts of the provided array along the specified dimensions

See also:

`dask.array.fft`, `numpy.fft`

Notes

A real fft is performed over the first dimension, which is faster. The transforms over the remaining dimensions are then computed with the classic fft.

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(4,4)),
...                  coords=[('lat', np.arange(-90,90,45)),
...                          ('time', pd.date_range(start='1/1/2018', periods=4,␣
↪freq='D'))])
>>> doppyo.utils.fft(A, dim='time', twosided=True, shift=True)
<xarray.DataArray 'fft' (lat: 4, f_time: 4)>
array([[ 2.996572+0.j      , -2.833156-0.676355j, -0.038218+0.j      ,
        -2.833156+0.676355j],
       [-0.66788 +0.j      ,  0.551732-3.406326j,  2.003329+0.j      ,
         0.551732+3.406326j],
       [ 2.032978+0.j      ,  0.657454+1.703941j,  2.085695+0.j      ,
         0.657454-1.703941j],
       [ 0.462405+0.j      , -0.815011+2.357146j, -1.257371+0.j      ,
        -0.815011-2.357146j]])
Coordinates:
  * lat      (lat) int64 -90 -45 0 45
  * f_time   (f_time) float64 -5.787e-06 -2.894e-06 0.0 2.894e-06
```

`utils.`**`ifft`**(*da, dim, nifft=None, shifted=True*)

Returns the sequential iffts of the provided array along the specified dimensions. Note, it is not possible to reconstruct the dimension along which the fft was performed (r_dim) from knowledge only of the fft "frequencies" (f_dim). For example, time cannot be reconstructed from frequency. Here, r_dim is defined relative to 0 in steps of dx as determined from f_dim. It may be necessary for the user to use the original (pre-fft) dimension to redefine r_dim after the ifft is performed (see the Examples s ection of this docstring).

Author: Dougie Squire
Date: 06/08/2018

Parameters

da [xarray.DataArray] Array from which compute the ifft

dim [str or sequence] Dimensions along which to compute the ifft

nifft [float or sequence, optional] Number of points in each dimensions to use in the transformation. If None, the full length of each dimension is used.

shifted [bool, optional] If True, assumes that the input dimensions are shifted to center the 0 frequency, otherwise assumes negative frequencies follow positive frequencies as in `numpy.fft.ftt`

Returns

ifft [xarray DataArray] Array containing the sequentual iffts of the provided array along the specified dimensions

See also:

`dask.array.ifft`, `numpy.ifft`

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(4,4)),
...                  coords=[('lat', np.arange(-90,90,45)),
...                  ('time', pd.date_range(start='1/1/2018', periods=4, freq='D
↪'))])
>>> A_fft = doppyo.utils.fft(A, dim=['time', 'lat'], twosided=True, shift=False)
>>> A_new = doppyo.utils.ifft(A_fft, dim=['f_lat', 'f_time'], shifted=False).real
>>> print(A_new)
<xarray.DataArray 'ifft' (lat: 4, time: 4)>
array([[-0.821396, -0.321925, -0.183761,  1.020338],
       [ 0.147125,  0.17867 ,  0.343659,  1.487173],
       [-1.53012 ,  1.586665, -0.097846,  1.535701],
       [ 0.663949, -0.9256  ,  0.086642,  0.586463]])
Coordinates:
  * lat      (lat) float64 0.0 45.0 90.0 135.0
  * time     (time) float64 0.0 8.64e+04 1.728e+05 2.592e+05
>>> A_new['lat'] = A['lat']
>>> A_new['time'] = A['time']
>>> print(A_new)
<xarray.DataArray 'ifft' (lat: 4, time: 4)>
array([[-0.821396, -0.321925, -0.183761,  1.020338],
       [ 0.147125,  0.17867 ,  0.343659,  1.487173],
       [-1.53012 ,  1.586665, -0.097846,  1.535701],
       [ 0.663949, -0.9256  ,  0.086642,  0.586463]])
Coordinates:
  * lat      (lat) int64 -90 -45 0 45
  * time     (time) datetime64[ns] 2018-01-01 2018-01-02 2018-01-03 2018-01-04
```

`utils.`**`fftfilt`**(*da*, *dim*, *method*, *dx*, *x_cut*)

Spectrally filters the provided array along dimension dim.

Author: Dougie Squire
Date: 15/09/2018

Parameters

da [xarray.DataArray] Array to filter

dim [str] Dimension along which to filter

method [{`"low pass"`, `"high pass"`, `"band pass"`}] Filter method to use

dx [value] Define the spacing of the dimension.

xc [value or array_like (if method = `'band pass'`)] Define the filter cut-off value(s), e.g. x_cut = 5*dx

Returns

filtered [xarray.DataArray] Filtered array

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(100)),
...                  coords=[('time', pd.date_range(start='1/1/2018', periods=100,
↪ freq='D'))])
>>> A_filt = doppyo.utils.fftfilt(A, dim='time', method='low pass', dx=1, x_
↪cut=10)
>>> print(A_filt)
<xarray.DataArray 'filtered' (time: 1000)>
array([ 0.120893,  0.059256, -0.085101, ..., -0.351555, -0.112201,  0.061701])
Coordinates:
  * time     (time) datetime64[ns] 2018-01-01 2018-01-02 ... 2020-09-26
>>> A.plot()
>>> A_filt.plot()
```

`utils.`**`isosurface`**(*da*, *coord*, *target*)

Returns the values of a coordinate in the input array where the input array values equals a prescribed target. E.g. returns the depth of the 20 degC isotherm. Returns nans for all points in input array where isosurface is not defined. If

Author: Thomas Moore and Dougie Squire
Date: 02/10/2018

Parameters

da [xarray DataArray] Array of values to be isosurfaced

coord [str] Name of coordinate to contruct isosurface about

target [value] Isosurface value

Returns

isosurface [xarray DataArray] Values of coord where da is closest to target. If multiple occurences of target occur along coord, only the maximum value of coord is returned

Notes

If multiple occurences of target occur along coord, only the maximum value of coord is returned

To do

- The current version includes no interpolation between grid spacing. This should be added as an option in the future

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(5,5)),
...                  coords=[('x', np.arange(5)), ('y', np.arange(5))])
>>> isosurface(A, coord='x', target=0)
>>> doppyo.utils.isosurface(A, coord='x', target=0)
<xarray.DataArray 'isosurface' (y: 5)>
array([ 4.,  1., nan,  3.,  4.])
Coordinates:
  * y        (y) int64 0 1 2 3 4
```

`utils.`**`load_mean_climatology`**(*clim*, *freq*, *variable=None*, *time_name=None*, ***kwargs*)

Returns pre-saved climatology at desired frequency.

Author: Dougie Squire

Date: 04/03/2018

Parameters

clim [str] Name of climatology to load. Currently available options are: "jra_1958-2016", "cafe_f1_atmos_2003-2017", "cafe_f1_ocean_2003-2017", "cafe_c2_atmos_400-499", "cafe_c2_atmos_500-549", "cafe_c2_ocean_400-499", "cafe_c2_ocean_500-549", "HadISST_1870-2018", "REMSS_2002-2018"

freq [str] Desired frequency of climatology (daily or longer) e.g. 'D', 'M'

variable [str, optional] Variable to load. If None, all variables are returned

time_name [str, optional] Name of the time dimension. If None, doppyo will attempt to determine time_name automatically

****kwargs** [dict] Additional arguments to pass to load command

Returns

climatology [xarray DataArray] Requested climatology

Notes

Can only be run from a system connected to Bowen cloud storage

Examples

```
>>> doppyo.utils.load_mean_climatology(clim='cafe_c2_atmos_500-549', freq='D',␣
↪variable='u')
<xarray.DataArray 'u' (time: 366, level: 37, lat: 90, lon: 144)>
[175504320 values with dtype=float32]
Coordinates:
  * lon      (lon) float64 1.25 3.75 6.25 8.75 11.25 ... 351.2 353.8 356.2 358.8
  * lat      (lat) float64 -89.49 -87.98 -85.96 -83.93 ... 85.96 87.98 89.49
  * level    (level) float32 1.0 2.0 3.0 5.0 7.0 ... 925.0 950.0 975.0 1000.0
  * time     (time) datetime64[ns] 2016-01-01T12:00:00 ... 2016-12-31T12:00:00
Attributes:
```

(continues on next page)

(continued from previous page)

```
    long_name:      zonal wind
    units:          m/sec
    valid_range:    [-32767  32767]
    packing:        4
    cell_methods:   time: mean
    time_avg_info:  average_T1,average_T2,average_DT
```

utils.**anomalize**(*data*, *clim*, *time_name=None*)

Returns anomalies of data about clim

Author: Dougie Squire
Date: 04/03/2018

Parameters

data [xarray DataArray] Array to compute anomalies from

clim [xarray DataArray] Array to compute anomalies about

time_name [str, optional] Name of the time dimension. If None, doppyo will attempt to determine time_name automatically

Returns

anomalies [xarray DataArray] Array containing anomalies of data about clim

Notes

Cannot anomalize about multiple day/month/year climatologies, e.g. 5-day averages

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(1000)),
...                  coords=[('time', pd.date_range(start='1/1/2000',␣
↪periods=1000, freq='D'))])
>>> A_clim = A.groupby('time.month').mean('time')
>>> doppyo.utils.anomalize(A, A_clim)
<xarray.DataArray (time: 1000)>
array([-3.050884, -0.361403, -0.893451, ...,  0.685141,  0.477916, -1.175434])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-01 2000-01-02 ... 2002-09-26
```

utils.**trunc_time**(*da*, *freq*, *time_name=None*)

Truncates values in provided array to provided frequency

Author: Dougie Squire
Date: 04/04/2018

Parameters

da [xarray DataArray] Array containing time coordinate to be truncated

freq [str] Truncation frequency. Options are ‘s’, ‘m’, ‘h’, D’, ‘M’, ‘Y’

time_name [str, optional] Name of the time dimension. If None, doppyo will attempt to determine time_name automatically

Returns

truncated [xarray DataArray] time-truncated array

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(10)),
...                  coords=[('time', pd.date_range(start='1/1/2000',
...                           periods=10, freq='M').shift(5,'D'))])
>>> doppyo.utils.trunc_time(A, freq='M')
<xarray.DataArray (time: 10)>
array([-0.197528,  1.022739, -0.50139 ,  0.128189, -0.886135,  0.570657,
       -0.336125, -0.499281,  1.143722,  1.987681])
Coordinates:
  * time     (time) datetime64[ns] 2000-02-01 2000-03-01 ... 2000-11-01
```

`utils.`**`leadtime_to_datetime`**`(`*da*, *init_date_name=’init_date’*, *lead_time_name=’lead_time’*, *time_name=’time’*`)`

Converts time information from initial date / lead time dimension pair to single datetime dimension (i.e. timeseries)

Author: Dougie Squire
Date: 04/04/2018

Parameters

da [xarray DataArray] Array in initial date / lead time format to convert to datetime format

init_date_name [str, optional] Name of initial date dimension

lead_time_name [str, optional] Name of lead time dimension

time_name [str, optional] Name of time dimension to create

Returns

converted [xarray DataArray] Array converted to datetime format

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(10)),
...                  coords=[('time', pd.date_range(start='1/1/2000', periods=10,␣
↪freq='M'))])
>>> B = doppyo.utils.datetime_to_leadtime(A)
>>> doppyo.utils.leadtime_to_datetime(B)
<xarray.DataArray (time: 10)>
array([-0.158172,  1.319148,  0.648378,  0.577859,  0.371392, -1.380317,
        0.126416,  1.184546,  0.107898,  1.304755])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-31 2000-02-29 ... 2000-10-31
```

`utils.`**`datetime_to_leadtime`**`(`*da*, *init_date_name='init_date'*, *lead_time_name='lead_time'*, *time_name='time'*`)`

Converts time information from single datetime dimension (i.e. timeseries) to initial date / lead time dimension pair

Author: Dougie Squire
Date: 04/04/2018

Parameters

da [xarray DataArray] Array to in datetime format to convert to initial date / lead time format

init_date_name [str, optional] Name of initial date dimension to create

lead_time_name [str, optional] Name of lead time dimension to create

time_name [str, optional] Name of time dimension

Returns

converted [xarray DataArray] Array converted to initial date / lead time format

Notes

Only compatible with time coordinates that have frequencies that can be determined by pandas.infer_freq(). This means that ambiguous frequencies, such as month-centred monthly frequencies must be preprocessed for compatibility (see doppyo.utils.trunc_freq())

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(10)),
...                  coords=[('time', pd.date_range(start='1/1/2000', periods=10,␣
↪freq='M'))])
>>> doppyo.utils.datetime_to_leadtime(A)
<xarray.DataArray (lead_time: 10)>
array([ 0.450976, -1.671764,  0.681519,  0.836319, -0.005434,  0.144954,
        0.719887,  0.344615,  0.461055,  0.736307])
Coordinates:
  * lead_time  (lead_time) int64 0 1 2 3 4 5 6 7 8 9
    init_date  datetime64[ns] 2000-01-31
```

`utils.`**`repeat_datapoint`**`(`*da*, *coord*, *coord_val*`)`

Returns array with data at coord = coord_val repeated across all other elements in coord. This is useful for generating persistence forecasts

Author: Dougie Squire
Date: 02/06/2018

Parameters

da [xarray DataArray] Array containing darta to repeat

coord [str] Coordinate in da over which to repeat the data at coord = coord_val

coord_val [value] The value of coord giving the data to be repeated

Returns

repeated [xarray DataArray] Array with data at coord=coord_val repeated across all other elements in coord

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)),
...                  coords=[('x', np.arange(3)),('y', np.arange(2))])
>>> doppyo.utils.repeat_datapoint(A, 'x', 2)
<xarray.DataArray (x: 3, y: 2)>
array([[-1.805652,  0.526434],
       [-1.805652,  0.526434],
       [-1.805652,  0.526434]])
Coordinates:
  * x        (x) int64 0 1 2
  * y        (y) int64 0 1
```

utils.**get_latlon_region**(*da*, *box*)

Returns an array containing those elements of the input array that fall within the provided lat-lon box

Author: Dougie Squire

Date: 04/04/2018

Parameters

da [xarray DataArray] Array to extract lat-lon box from

box [array_like] Edges of lat-lon box in the format [lat_min, lat_max, lon_min, lon_max]

Returns

reduced [xarray DataArray] Array containing those elements of the input array that fall within the box

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(180,360)),
...                  coords=[('lat', np.arange(-90,90,1)),('lon', np.arange(-280,
↪80,1))])
>>> doppyo.utils.get_latlon_region(A, [-10, 10, 70, 90])
<xarray.DataArray (lat: 21, lon: 21)>
array([[ 0.854745,  1.53709 ,  0.491165, ..., -0.675664,  1.572102, -0.931492],
       [ 0.570822,  0.60621 , -0.125524, ..., -1.731507,  0.853652,  0.845369],
       [-0.061811,  0.758512,  1.215573, ..., -1.275482,  2.668203,  0.791314],
       ...,
       [-0.263597,  0.102755, -2.775252, ..., -0.736136,  0.944762,  0.005952],
       [ 0.009949,  0.409897, -0.138621, ...,  1.054246,  1.30817 , -0.539534],
       [ 1.281245, -0.792166, -1.736007, ...,  0.474207, -0.781518,  0.738593]])
Coordinates:
  * lat      (lat) int64 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7 8 9 10
  * lon      (lon) int64 -280 -279 -278 -277 -276 -275 ... 74 75 76 77 78 79
```

utils.**latlon_average**(*da*, *box*)

Returns the average of the input array over a provide lat-lon box,

Author: Dougie Squire
Date: 04/04/2018

Parameters

da [xarray DataArray] Array to average lat-lon box from

box [array_like] Edges of lat-lon box in the format [lat_min, lat_max, lon_min, lon_max]

Returns

reduced [xarray DataArray] Array containing those elements of the input array that fall within the box

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(180,360)),
...                  coords=[('lat', np.arange(-90,90,1)),('lon', np.arange(-280,
↪80,1))])
>>> doppyo.utils.latlon_average(A, [-10, 10, 70, 90])
<xarray.DataArray ()>
array(-0.056776)
```

utils.**stack_by_init_date**(*da*, *init_dates*, *N_lead_steps*, *init_date_name='init_date'*, *lead_time_name='lead_time'*, *time_name='time'*)

Stacks provided timeseries array in an inital date / lead time format. Note this process replicates data and can substantially increase memory usage. Lead time frequency will match frequency of input data. Returns nans if requested times lie outside of the available range

Author: Dougie Squire
Date: 14/03/2018

Parameters

da [xarray DataArray] Timeseries array to be stacked

init_dates [array_like of datetime objects] Initial dates to stack onto

N_lead_steps [value] Number of lead time steps

init_date_name [str, optional] Name of initial date dimension

lead_time_name [str, optional] Name of lead time dimension

time_name [str, optional] Name of time dimension

Returns

stacked [xarray DataArray] Stacked xarray in inital date / lead time format

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3)),
...                  coords=[('time', pd.date_range(start='2000-01-01', periods=3,
↪ freq='MS'))])
>>> init_dates = pd.date_range(start='1999-11-01', periods=3, freq='MS')
```

(continues on next page)

(continued from previous page)

```
>>> doppyo.utils.stack_by_init_date(A, init_dates=init_dates, N_lead_steps=3)
<xarray.DataArray (init_date: 3, lead_time: 3)>
array([[      nan,       nan,       nan],
       [      nan,       nan,       nan],
       [ 0.509276, -3.046124, -0.665343]])
Coordinates:
  * lead_time  (lead_time) int64 0 1 2
  * init_date  (init_date) datetime64[ns] 1999-11-01 1999-12-01 2000-01-01
```

`utils.`**`concat_times`**(*da*, *init_date_name=’init_date’*, *lead_time_name=’lead_time’*, *time_name=’time’*)

Unstack and concatenate all init_date/lead_time rows into single time dimension

Author: Dougie Squire
Date: 22/04/2018

Parameters

da [xarray DataArray] Array to be unstacked and concatenated

init_date_name [str, optional] Name of initial date dimension

lead_time_name [str, optional] Name of lead time dimension

time_name [str, optional] Name of time dimension

Returns

concatenated [xarray DataArray] Unstacked and concatenated array

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,3)),
...                  coords=[('init_date',
...                           pd.date_range(start='1/1/2018', periods=3, freq='M
↪')),
...                          ('lead_time', np.arange(3))])
>>> A['lead_time'].attrs['units'] = 'M'
>>> doppyo.utils.concat_times(A)
<xarray.DataArray (time: 9)>
array([-1.65746 ,  0.57727 ,  0.010619, -0.008245,  0.119201, -0.445606,
       -0.546745,  0.157267, -1.616096])
Coordinates:
  * time     (time) datetime64[ns] 2018-01-31 2018-02-28 ... 2018-05-31
```

`utils.`**`prune`**(*da*, *squeeze=False*)

Removes all coordinates that are not dimensions

Author: Dougie Squire
Date: 22/04/2018

Parameters

da [xarray DataArray] Array to prune

squeeze [bool, optional] If True, squeeze the array (i.e. remove 1D dimensions) prior to pruning

Returns

pruned [xarray DataArray] The pruned array

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,1)),
...                  coords=[('x', np.arange(3)),('y', np.arange(1))]).expand_
↪dims('z')
>>> A.coords['w'] = 1
>>> doppyo.utils.prune(A, squeeze=True)
<xarray.DataArray (x: 3)>
array([-1.323662,  1.464171,  0.480917])
Coordinates:
  * x        (x) int64 0 1 2
```

`utils.`**`get_other_dims`**(*da*, *dims_exclude*)

Returns all dimensions in provided dataset excluding dim_exclude

Author: Dougie Squire
Date: 22/04/2018

Parameters

da [xarray DataArray] Array to retreive dimensions from

dims_exclude [str or sequence of str] Dimensions to exclude

Returns

dims [str or sequence of str] Dimensions of input array, excluding dims_exclude

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(3,2)), coords=[('x', np.arange(3)),
...                                                         ('y', np.arange(2))])
>>> doppyo.utils.get_other_dims(A, 'y')
'x'
```

`utils.`**`cftime_to_datetime64`**(*time*, *shift_year=0*)

Convert cftime object to datetime64 object, allowing for *NOLEAP* calendar configuration

Author: Dougie Squire
Date: 04/09/2018

Parameters

time [cftime or array_like of cftime] Times to be converted to datetime64

shift_year: values Number of years to shift times by. cftime objects are generated by xarray when times fall outside of the range 1678-2261. Shifting years to within this range enables conversion to datetime64 within an xarray object

Returns

converted [numpy datetime64 or array_like of numpy datetime64] Input times converted from cftime to numpy datetime64

Notes

Times must be sequential and monotonic

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(12)),
...                  coords=[('time', np.array([cftime.datetime(0, m, 1) for m in␣
↪np.arange(1,13)]))])
>>> A['time'] = doppyo.utils.cftime_to_datetime64(A['time'], shift_year=2000)
>>> A
<xarray.DataArray (time: 12)>
array([ 0.391673, -1.317681,  1.51771 , -0.195475,  0.525342,  0.390625,
        1.426725, -0.261821,  1.021318,  1.205761, -0.907714,  1.009402])
Coordinates:
  * time     (time) datetime64[ns] 2000-01-01 2000-02-01 ... 2000-12-01
```

`utils.`**`get_time_name`**`(`*da*`)`

Returns name of time dimension in input array

Author: Dougie Squire
Date: 03/03/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to time

Returns

name [str] Name of dimension corresponding to time

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('time', np.arange(2))])
>>> doppyo.utils.get_time_name(A)
'time'
```

`utils.`**`get_lon_name`**`(`*da*`)`

Returns name of longitude dimension in input array

Author: Dougie Squire
Date: 03/03/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to longitude

Returns

name [str] Name of dimension corresponding to longitude

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('level', np.arange(2))])
>>> doppyo.utils.get_lon_name(A)
'lon'
```

utils.**get_lat_name**(*da*)

Returns name of latitude dimension in input array

Author: Dougie Squire

Date: 03/03/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to latitude

Returns

name [str] Name of dimension corresponding to latitude

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('level', np.arange(2))])
>>> doppyo.utils.get_lat_name(A)
'lat'
```

utils.**get_depth_name**(*da*)

Returns name of depth dimension in input array

Author: Thomas Moore

Date: 31/10/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to depth

Returns

name [str] Name of dimension corresponding to depth

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('level', np.arange(2))])
>>> doppyo.utils.get_depth_name(A)
'depth'
```

utils.**get_level_name**(*da*)

Returns name of atmospheric level dimension in input array

Author: Dougie Squire
Date: 03/03/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to atmospheric level

Returns

name [str] Name of dimension corresponding to atmospheric level

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('level', np.arange(2))])
>>> doppyo.utils.get_level_name(A)
'level'
```

utils.**get_plevel_name**(*da*)

Returns name of pressure level dimension in input array

Author: Dougie Squire
Date: 03/03/2018

Parameters

da [xarray DataArray] Array with coordinate corresponding to pressure level

Returns

name [str] Name of dimension corresponding to pressure level

Examples

```
>>> A = xr.DataArray(np.random.normal(size=(2,2,2,2,2)),
...                  coords=[('lat', np.arange(2)), ('lon', np.arange(2)),
...                          ('depth', np.arange(2)), ('level', np.arange(2))])
>>> doppyo.utils.get_plevel_name(A)
'level'
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX