esmtools

Jul 29, 2020

Getting Started

CHAPTER 1

Most Recent Release

v1.1.1 of `esmtools` mainly introduces dask-friendly, vectorized, lightweight functions for standard statistical functions. They also intelligently handle datetimes on the independent (x) axis:

- *linear_slope()*
- *linregress()*
- *polyfit()*
- *rm_poly()*
- *rm_trend()*

CHAPTER 2

Installation

You can install the latest release of `esmtools` using `pip` or `conda`:

```
pip install esmtools
```

```
conda install -c conda-forge esmtools
```

You can also install the bleeding edge (pre-release versions) by running

```
pip install git+https://github.com/bradyrx/esmtools@master --upgrade
```

Getting Started

- *Examples*
- *Calling Functions via Accessors*

2.1 Examples

[92]:
```
import xarray as xr
import PMMPIESM as PM
import glob
import numpy as np
import matplotlib.pyplot as plt
```

2.1.1 Takahashi Decomposition

[2]:
```
path = '/work/mh0727/m300524/experiments/results/'
```

[30]:
```
tos = xr.open_dataarray(path+'control_tos_mm.nc')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/times.py:122: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  result = decode_cf_datetime(example_value, units, calendar)
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/variables.py:69: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  return self.func(self.array)
```

[31]:
```
spco2 = xr.open_dataarray(path+'control_spco2_mm.nc')*10
```

[32]:
```
ds = xr.merge([tos,spco2])
```

[41]:
```
def temp_decomp_takahashi(ds, time_dim='time'):
    """
    Decompose spco2 into thermal and non-thermal component.

    Reference
    ---------
    Takahashi, Taro, Stewart C. Sutherland, Colm Sweeney, Alain Poisson, Nicolas␣
↪Metzl, Bronte Tilbrook,
    Nicolas Bates, et al. "Global Sea-Air CO2 Flux Based on Climatological Surface␣
↪Ocean PCO2, and Seasonal
    Biological and Temperature Effects." Deep Sea Research Part II: Topical Studies␣
↪in Oceanography, The
    Southern Ocean I: Climatic Changes in the Cycle of Carbon in the Southern Ocean,␣
↪49, no. 9 (January 1,2002):
    1601-22. https://doi.org/10/dmk4f2.

    Input
    -----
    ds : xr.Dataset containing spco2[ppm] and tos[C or K]

    Output
    ------
    thermal, non_thermal : xr.DataArray
        thermal and non-thermal components in ppm units

    """
    fac = 0.0432
    tos_mean = ds['tos'].mean(time_dim)
    tos_diff = ds['tos'] - tos_mean
    thermal = ds['spco2'].mean(time_dim) * (np.exp(tos_diff * fac))
    non_thermal = ds['spco2'] * (np.exp(tos_diff * -fac))
    return thermal, non_thermal
```

[34]:
```
thermal, non_thermal = temp_decomp_takahashi(ds)
```

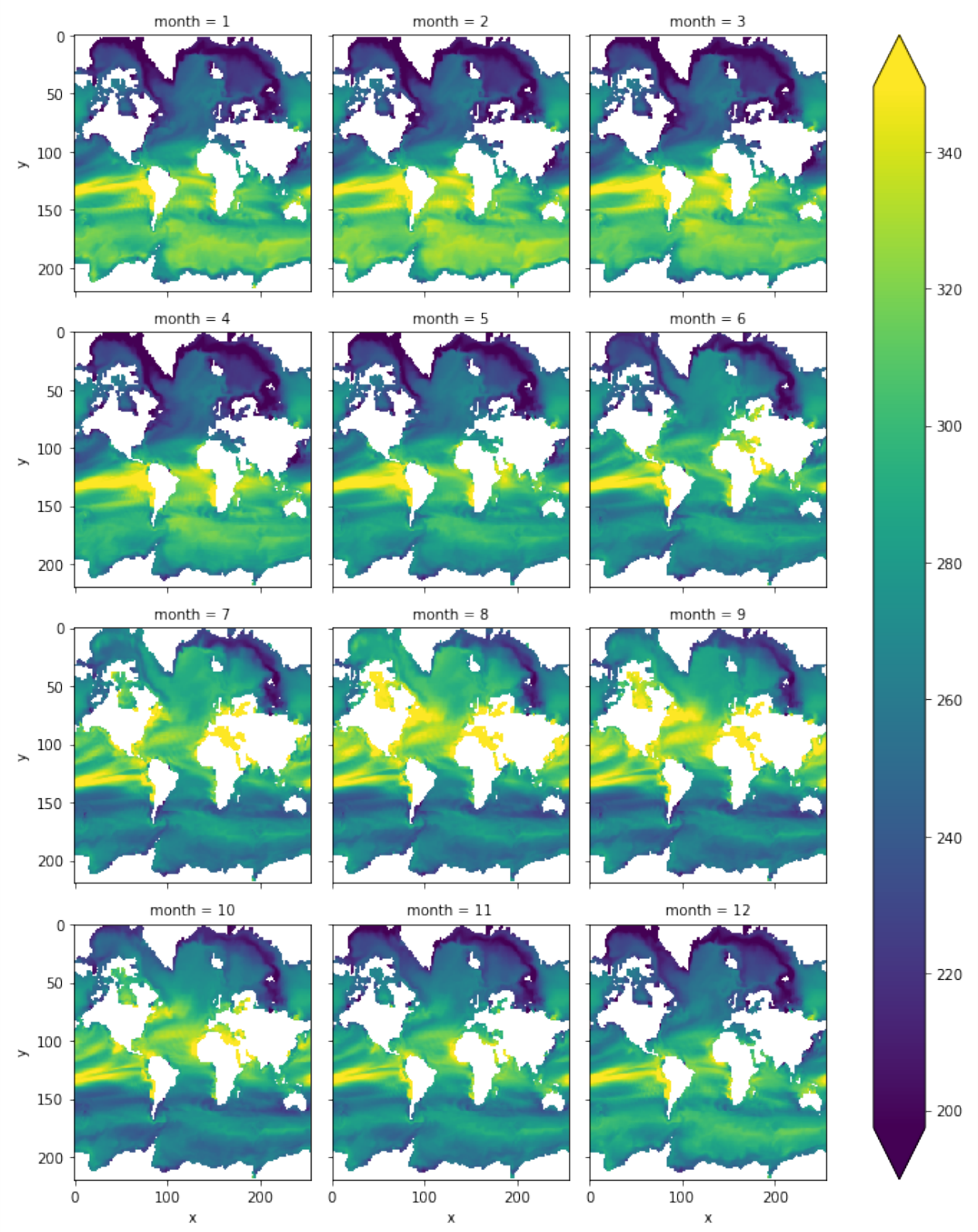
```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪core/nanops.py:161: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪core/nanops.py:161: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[35]:
```
thermal_seasonality = thermal.groupby('time.month').mean('time')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪core/nanops.py:161: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[39]:
```
thermal_seasonality.plot(col='month',col_wrap=3, yincrease=False, robust=True)
```

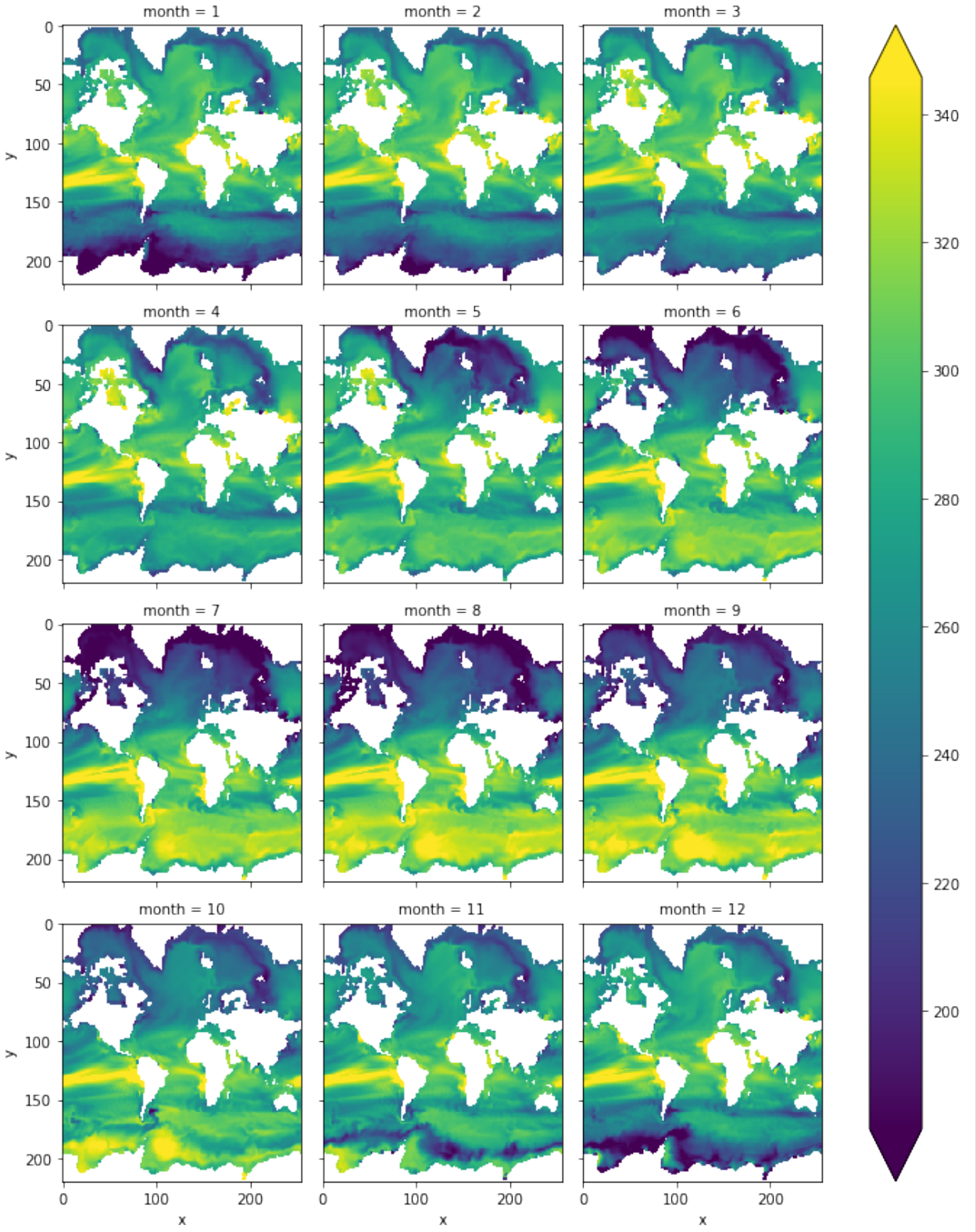
[39]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2afa998ef160>
```



[37]:
```
non_thermal_seasonality = non_thermal.groupby('time.month').mean('time')
```

[40]:
```
non_thermal_seasonality.plot(col='month',col_wrap=3, yincrease=False, robust=True)
```

[40]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2afa99b684a8>
```



[ ]:

[ ]:

2.1.2 Potential pco2

[7]:
```
h3d = xr.open_dataset('/work/mh0727/m300524/experiments/vga0214/outdata/hamocc/
↪vga0214_hamocc_data_3d_mm_32920101_32921231.nc')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/times.py:122: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  result = decode_cf_datetime(example_value, units, calendar)
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/variables.py:69: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  return self.func(self.array)
```

[15]:
```
talk = h3d['talk']
dissic = h3d['dissic']
```

[98]:
```
# dirty fix to get pco2_insitu
k=1
pco2_insitu = k*(2* dissic - talk)**2/(talk-dissic)
```

[8]:
```
m3d = xr.open_dataset('/work/mh0727/m300524/experiments/vga0214/outdata/mpiom/vga0214_
↪mpiom_data_3d_mm_32920101_32921231.nc')
```

[10]:
```
t_insitu = m3d['thetao']
```

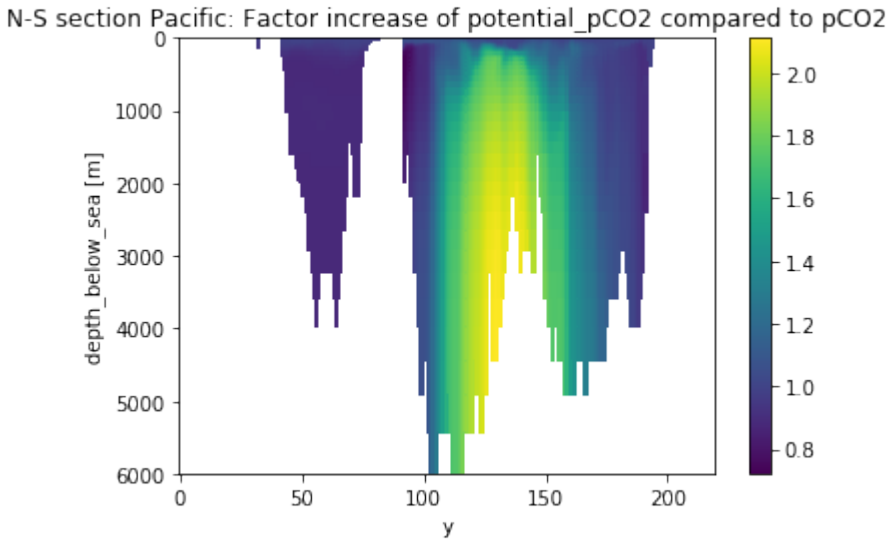
[102]:
```
def potential_pco2(t_insitu, pco2_insitu):
    """
    Calculate potential pco2 in the inner ocean.

    Reference:
    - Sarmiento, Jorge Louis, and Nicolas Gruber. Ocean Biogeochemical Dynamics.
        Princeton, NJ: Princeton Univ. Press, 2006., p.421, eq. (10:3:1)

    """
    t_sfc = t_insitu.sel(depth=6)
    pco2_potential = pco2_insitu * (1 + 0.0423 * (t_sfc - t_insitu))
    return pco2_potential
```

[62]:
```
pot_pco2 = potential_pco2(t_insitu, pco2_insitu)
```

[101]:
```
(pot_pco2/pco2_insitu).mean('time').isel(x=0).plot(yincrease=False)
plt.title('N-S section Pacific: Factor increase of potential_pCO2 compared to pCO2')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪core/nanops.py:161: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[101]:
```
Text(0.5,1,'N-S section Pacific: Factor increase of potential_pCO2 compared to pCO2')
```



[ ]:

[1]:
```
import xarray as xr
import PMMPIESM as PM
import glob
import numpy as np
```

[2]:
```
path = '/work/mh0727/m300524/experiments/results/'
```

[40]:
```
tos = xr.open_dataarray(path+'control_tsw_mm.nc')
```

[37]:
```
light = xr.open_dataarray(path+'control_soflwac_mm.nc')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/times.py:122: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  result = decode_cf_datetime(example_value, units, calendar)
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪coding/variables.py:69: SerializationWarning: Unable to decode time axis into full␣
↪numpy.datetime64 objects, continuing using dummy cftime.datetime objects instead,␣
↪reason: dates out of range
  return self.func(self.array)
```

[8]:
```
fe = xr.open_dataarray(path+'control_dfeos_mm.nc')
```

[5]:
```
po4 = xr.open_dataarray(path+'control_po4os_mm.nc')
```

[10]:
```
no3 = xr.open_dataarray(path+'control_no3os_mm.nc')
```

[41]:
```
ds_nut = xr.merge([fe,po4,no3])
ds_tp = xr.merge([tos,light])
```

2.1.3 Nutrient availability

Context: value of 1 means limited by nutrient. The less the more primary productivity.

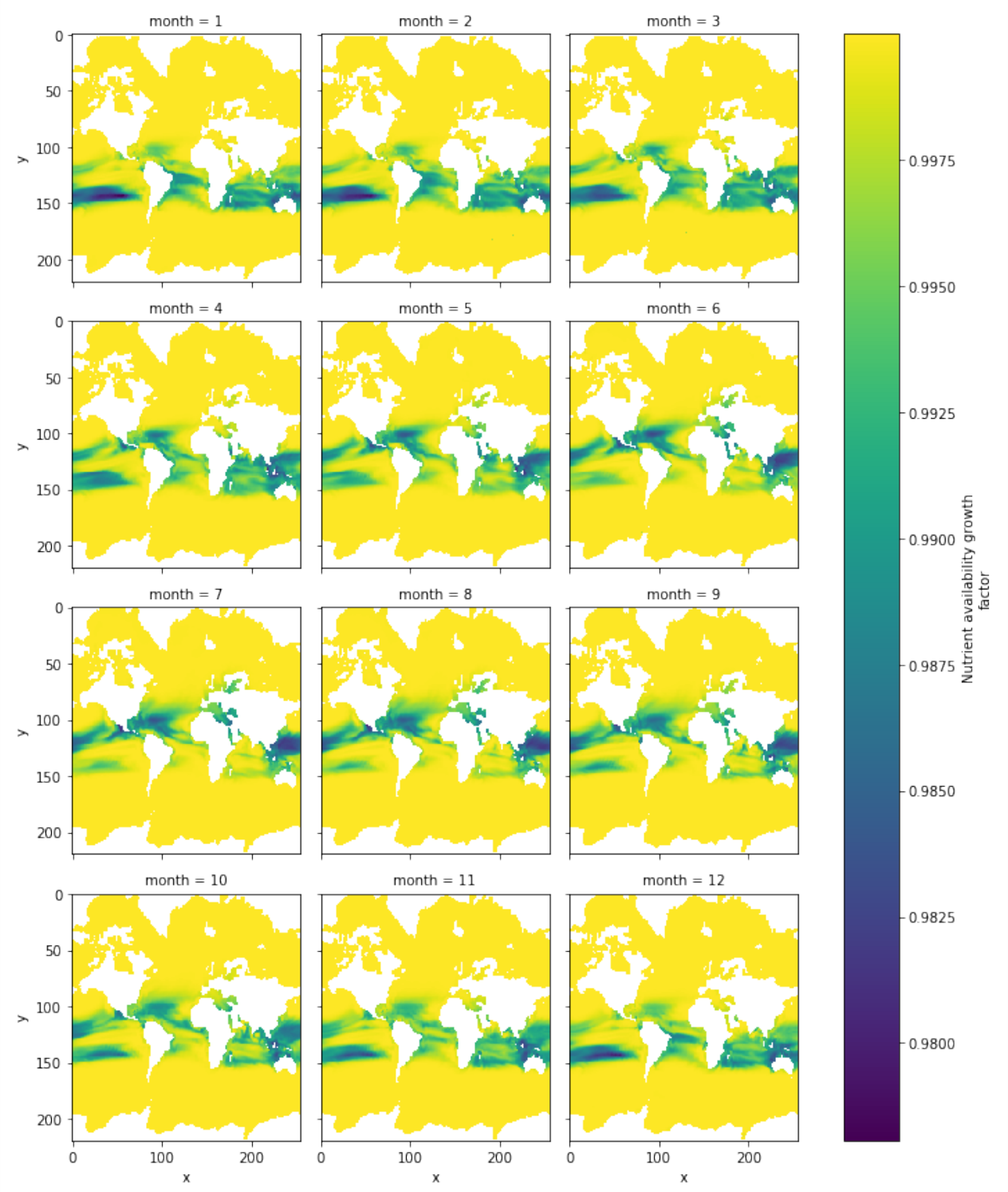
[44]:
```
nutlim, iron_lim, nitrate_lim, phos_lim = PM.hamocc.get_nutlimf(ds_nut)
```

[45]:
```
nutlim_seasonality = nutlim.groupby('time.month').mean('time')
```

```
/work/mh0727/m300524/anaconda3/envs/my_jupyter/lib/python3.6/site-packages/xarray/
↪core/nanops.py:161: RuntimeWarning: Mean of empty slice
  return np.nanmean(a, axis=axis, dtype=dtype)
```

[46]:
```
nutlim_seasonality.plot(col='month',col_wrap=3, yincrease=False)
```

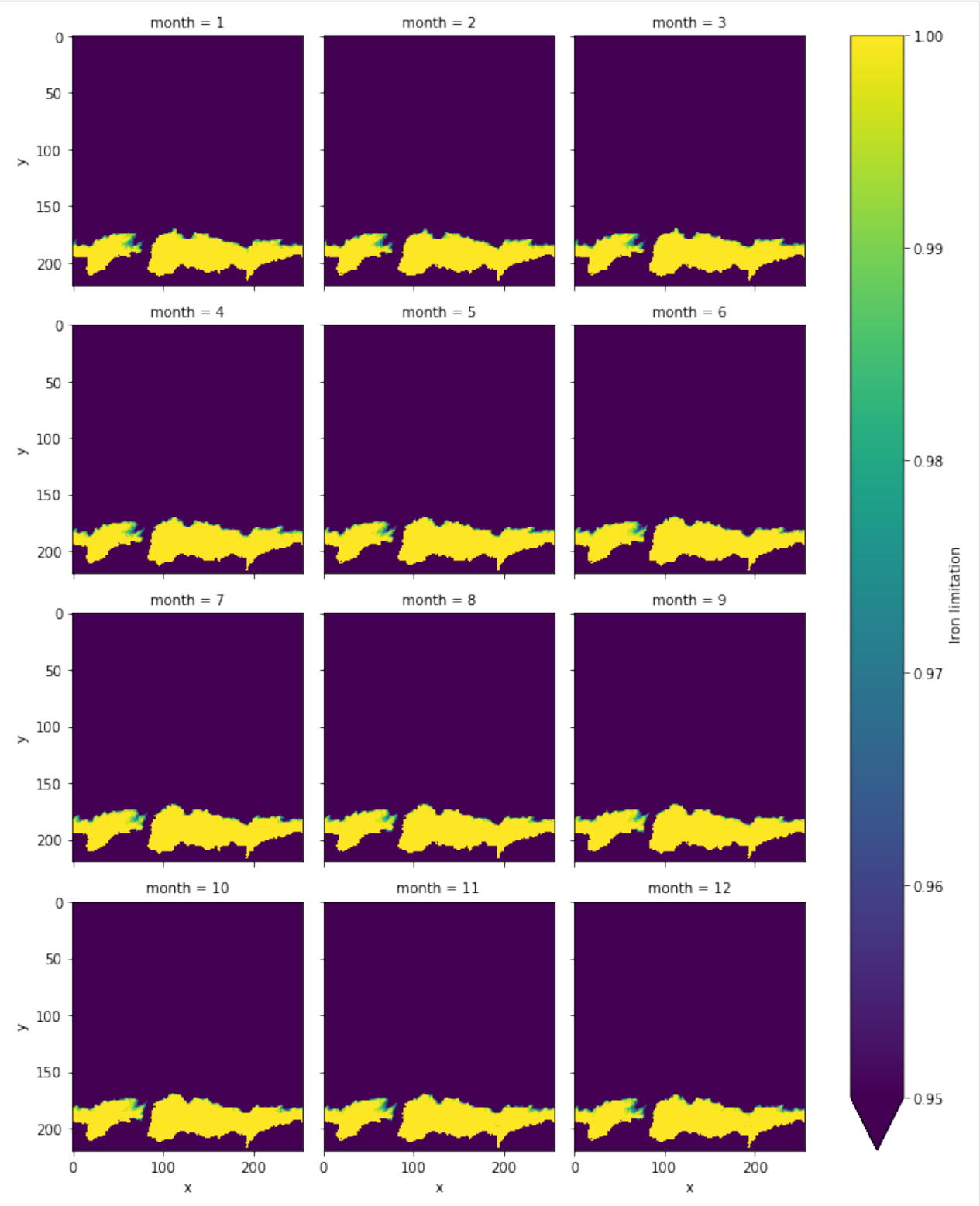
[46]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2b188e025a58>
```



[63]:
```
iron_lim_seasonality = iron_lim.groupby('time.month').mean('time')
```

[65]:
```
iron_lim_seasonality.plot(col='month',col_wrap=3, yincrease=False, vmin=.95)
```

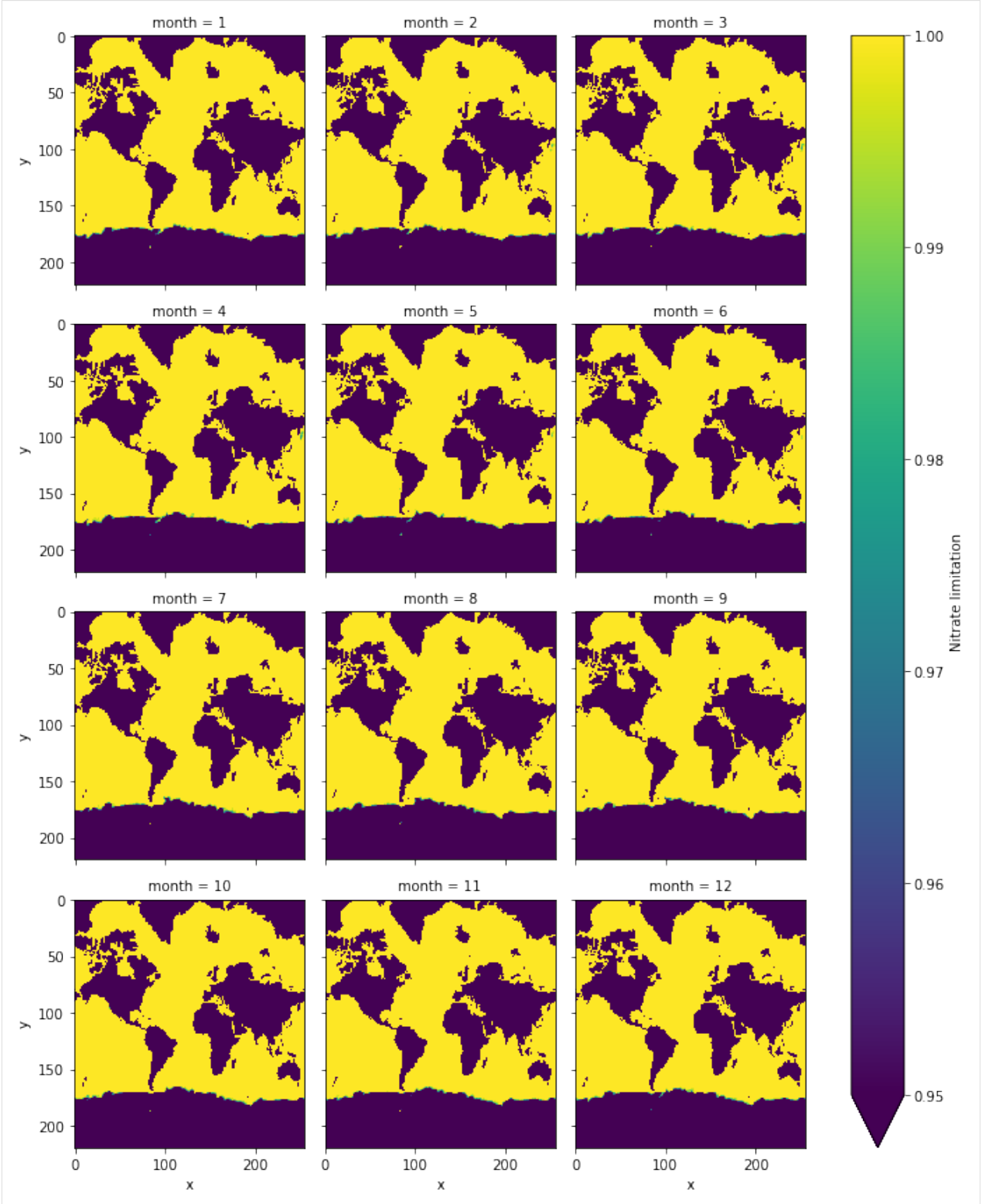
[65]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2b19b3886898>
```



[71]:
```
nitrate_lim_seasonality = nitrate_lim.groupby('time.month').mean('time')
```

[73]:
```
nitrate_lim_seasonality.plot(col='month',col_wrap=3, yincrease=False, vmin=.95)
```

[73]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2b19b4c31b70>
```



2.1.4 Temp light dependence

Temperature-light dependent primary productivity growth factor. The larger the more PP.

[47]:
```
ds.data_vars
```

[47]:
```
Data variables:
    tos      (time, y, x) float32 nan nan nan nan nan ... nan nan nan nan nan
    soflwac  (time, lat, lon) float32 nan nan nan nan nan ... 0.0 0.0 0.0 0.0
    dfeos    (time, y, x) float32 nan nan nan nan nan ... nan nan nan nan nan
    po4os    (time, y, x) float32 nan nan nan nan nan ... nan nan nan nan nan
    no3os    (time, y, x) float32 nan nan nan nan nan ... nan nan nan nan nan
```

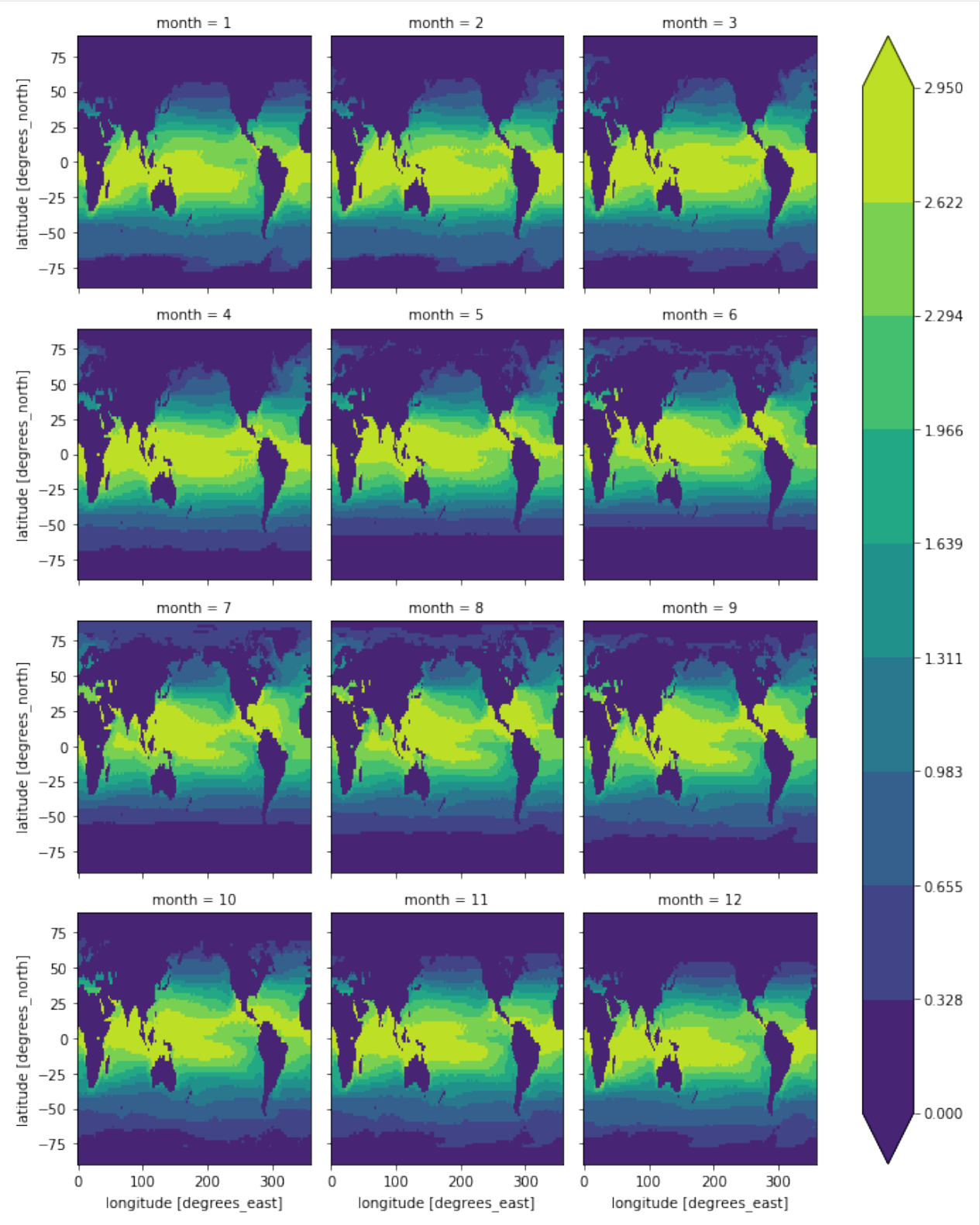
[49]:
```
def temfa_phofa(ds):
    temfa = .6 * 1.066 ** (ds['tsw'] - 273.15)
    phofa = ds['soflwac'] * 0.02
    return temfa * phofa / (np.sqrt( phofa ** 2 + temfa ** 2 ))
```

[50]:
```
tp = temfa_phofa(ds_tp)
```

[51]:
```
tp_seasonality = tp.groupby('time.month').mean('time')
```

[62]:
```
tp_seasonality.plot(col='month',col_wrap=3, cmap='viridis', robust=True, levels=10,␣
↪vmin=0)
```

[62]:
```
<xarray.plot.facetgrid.FacetGrid at 0x2b19b2cc6f60>
```


month = 1
month = 2
month = 3
month = 4
month = 5
month = 6
month = 7
month = 8
month = 9
month = 10
month = 11
month = 12
latitude [degrees_north]
75
50
25
0
−25
−50
−75
0
100
200
300
longitude [degrees_east]
2.950
2.622
2.294
1.966
1.639
1.311
0.983
0.655
0.328
0.000

[ ]:

2.1.5 cyanos

[ ]:

2.2 Calling Functions via Accessors

A subset of `esmtools` functions are registered as `xarray` accessors. What this means is that you can call some of these functions as you would `.isel()`, `.coarsen()`, `.interp()`, and so on with `xarray`.

There is just one extra step to do so. After importing `esmtools`, you have to do add the module call after `ds` and then the function. For example, you can call `ds.grid.convert_lon()` to transform between -180 to 180 longitudes and 0 to 360 longitudes.

List of currently supported modules/functions. *See the API for usage.*

1. `grid`
    - convert_lon()

[1]:
```
import esmtools
import numpy as np
import xarray as xr
```

[2]:
```
lat = np.linspace(-89.5, 89.5, 180)
lon = np.linspace(0.5, 359.5, 360)
empty = xr.DataArray(np.empty((180, 360)), dims=['lat', 'lon'])
data = xr.DataArray(np.linspace(0, 360, 360), dims=['lon'],)
data, _ = xr.broadcast(data, empty)
data = data.T
data['lon'] = lon
data['lat'] = lat
```

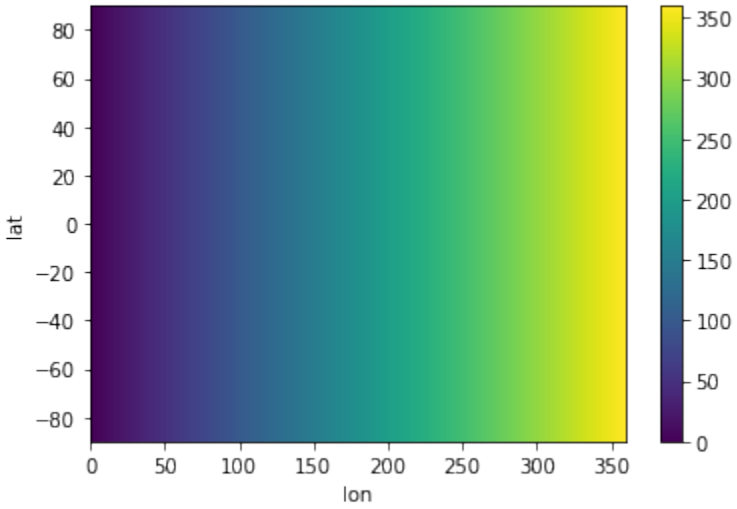
[3]:
```
print(data)
```

```
<xarray.DataArray (lat: 180, lon: 360)>
array([[  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ],
       [  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ],
       [  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ],
       ...,
       [  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ],
       [  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ],
       [  0.        ,   1.00278552,   2.00557103, ..., 357.99442897,
         358.99721448, 360.        ]])
Coordinates:
  * lon      (lon) float64 0.5 1.5 2.5 3.5 4.5 ... 355.5 356.5 357.5 358.5 359.5
  * lat      (lat) float64 -89.5 -88.5 -87.5 -86.5 -85.5 ... 86.5 87.5 88.5 89.5
```

Our sample data is just a plot of longitude.

[4]:
```
data.plot(x='lon', y='lat')
```

[4]:
```
<matplotlib.collections.QuadMesh at 0x7f9949617a58>
```



However, it ranges from 0 to 360, which is sometimes problematic. We can use the accessor `convert_lon()` to convert this to -180 to 180.

[5]:
```
help(data.grid.convert_lon)
```

```
Help on method convert_lon in module esmtools.accessor:

convert_lon(coord='lon') method of esmtools.accessor.GridAccessor instance
    Converts longitude grid from -180to180 to 0to360 and vice versa.

    .. note::
        Longitudes are not sorted after conversion (i.e., spanning -180 to 180 or
        0 to 360 from index 0, ..., N) if it is 2D.

    Args:
        ds (xarray object): Dataset to be converted.
        coord (optional str): Name of longitude coordinate, defaults to 'lon'.

    Returns:
        xarray object: Dataset with converted longitude grid.

    Raises:
        ValueError: If ``coord`` does not exist in the dataset.
```

[6]:
```
converted = data.grid.convert_lon(coord='lon')
```

Now we've switched over to the -180 to 180 coordinate system.

[7]:
```
converted
```

[7]:
```
<xarray.DataArray (lat: 180, lon: 360)>
array([[180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
        178.49582173, 179.49860724],
       [180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
        178.49582173, 179.49860724],
       [180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
```

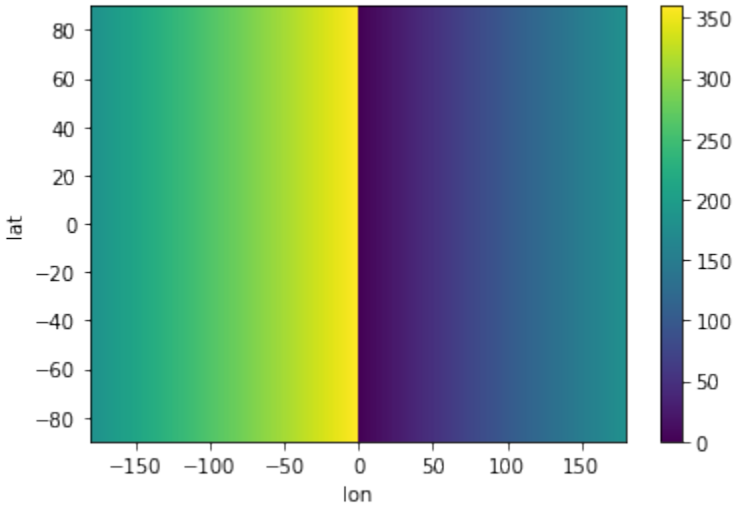
(continues on next page)

(continued from previous page)

```
        178.49582173, 179.49860724],
       ...,
       [180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
        178.49582173, 179.49860724],
       [180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
        178.49582173, 179.49860724],
       [180.50139276, 181.50417827, 182.50696379, ..., 177.49303621,
        178.49582173, 179.49860724]])
Coordinates:
  * lon      (lon) float64 -179.5 -178.5 -177.5 -176.5 ... 177.5 178.5 179.5
  * lat      (lat) float64 -89.5 -88.5 -87.5 -86.5 -85.5 ... 86.5 87.5 88.5 89.5
```

[8]:
```
converted.plot(x='lon', y='lat')
```

[8]:
```
<matplotlib.collections.QuadMesh at 0x7f99498234e0>
```

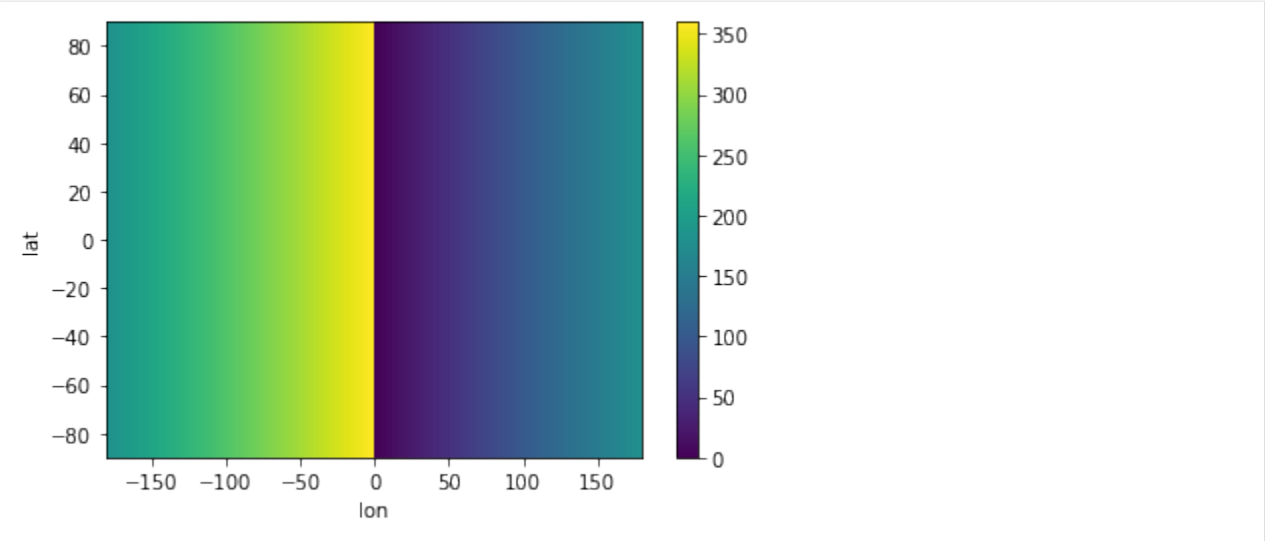


This is equivalent to running the functional `convert_lon()` argument:

[9]:
```
converted = esmtools.grid.convert_lon(data, coord='lon')
```

[10]:
```
converted.plot(x='lon', y='lat')
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x7f9946cfb4a8>
```



Help & Reference

- *API Reference*
- *Contribution Guide*
- *Changelog History*
- *Release Procedure*
- *Contributors*
- *Additional Packages*

2.3 API Reference

This page provides an auto-generated summary of esmtools's API. For more details and examples, refer to the relevant chapters in the main part of the documentation.

2.3.1 Carbon

`from esmtools.carbon import ...`

Functions related to analyzing ocean (and perhaps terrestrial) biogeochemistry.

calculate_compatible_emissions(...)	Calculate compatible emissions.
co2_sol(t, s)	Compute CO2 solubility per the equation used in CESM.
get_iam_emissions()	Download IAM emissions from PIK website.
plot_compatible_emissions(...[, ...])	Plot combatible emissions.
potential_pco2(t_insitu, pco2_insitu)	Calculate potential pCO2 in the interior ocean.
schmidt(t)	Computes the dimensionless Schmidt number.
spco2_sensitivity(ds)	Compute sensitivity of surface pCO2 to changes in driver variables.

Continued on next page

Table 1 – continued from previous page

spco2_decomposition_index(ds_terms, index[, ...])	Decompose oceanic surface pco2 in a first order Taylor-expansion.
spco2_decomposition(ds_terms[, detrend, ...])	Decompose oceanic surface pco2 in a first order Taylor-expansion.
temp_decomp_takahashi(ds[, time_dim, ...])	Decompose surface pCO2 into thermal and non-thermal components.

esmtools.carbon.calculate_compatible_emissions

esmtools.carbon.**calculate_compatible_emissions**(*global_co2_flux*, *co2atm_forcing*)

Calculate compatible emissions.

Parameters

- **global_co2_flux** (*xr.object*) – global co2_flux in PgC/yr.
- **co2atm_forcing** (*xr.object*) – prescribed atm. CO2 forcing in ppm.

Returns compatible emissions in PgC/yr.

Return type xr.object

Reference:

- Jones, Chris, Eddy Robertson, Vivek Arora, Pierre Friedlingstein, Elena Shevliakova, Laurent Bopp, Victor Brovkin, et al. "Twenty-First-Century Compatible CO2 Emissions and Airborne Fraction Simulated by CMIP5 Earth System Models under Four Representative Concentration Pathways." Journal of Climate 26, no. 13 (February 1, 2013): 4398–4413. https://doi.org/10/f44bbn.

esmtools.carbon.co2_sol

esmtools.carbon.**co2_sol**(*t*, *s*)

Compute CO2 solubility per the equation used in CESM.

Note: See `co2calc.F90` for the calculation of CO2 solubility in CESM.

Parameters

- **t** (*xarray object*) – SST (degC)
- **s** (*xarray object*) – SSS (PSU)

Returns Value of solubility in mol/kg/atm

Return type ff (xarray object)

References

Weiss & Price (1980, Mar. Chem., 8, 347-359; Eq 13 with table 6 values)

Examples

```
>>> from esmtools.carbon import co2_sol
>>> import numpy as np
>>> import xarray as xr
>>> t = xr.DataArray(np.random.randint(10, 25, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon'])
>>> s = xr.DataArray(np.random.randint(30, 35, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon'])
>>> ff = co2_sol(t, s)
```

esmtools.carbon.get_iam_emissions

esmtools.carbon.**get_iam_emissions**()

Download IAM emissions from PIK website.

Returns emissions from the IAMs in PgC/yr.

Return type iam_emissions (xr.object)

esmtools.carbon.plot_compatible_emissions

esmtools.carbon.**plot_compatible_emissions**(*compatible_emissions*, *global_co2_flux*, *iam_emissions=None*, *ax=None*)

Plot combatible emissions.

Parameters

- **compatible_emissions** (*xr.object*) – compatible_emissions in PgC/yr from *calculate_compatible_emissions*.
- **global_co2_flux** (*xr.object*) – Global CO2 flux in PgC/yr.
- **iam_emissions** (*xr.object*) – (optional) Emissions from the IAMs in PgC/yr. Defaults to None.
- **ax** (*plt.ax*) – (optional) matplotlib axis to plot onto. Defaults to None.

Returns matplotlib axis.

Return type ax (plt.ax)

References

- Jones, Chris, Eddy Robertson, Vivek Arora, Pierre Friedlingstein, Elena Shevliakova, Laurent Bopp, Victor Brovkin, et al. "Twenty-First-Century Compatible CO2 Emissions and Airborne Fraction Simulated by CMIP5 Earth System Models under Four Representative Concentration Pathways." Journal of Climate 26, no. 13 (February 1, 2013): 4398–4413. https://doi.org/10/f44bbn. Fig. 5a
- IPCC AR5 Fig. 6.25

esmtools.carbon.potential_pco2

esmtools.carbon.**potential_pco2**(*t_insitu*, *pco2_insitu*)

Calculate potential pCO2 in the interior ocean.

Note: Requires the first index of depth to be at the surface.

Parameters

- **t_insitu** (*xarray object*) – Temperature with depth [degC]
- **pco2_insitu** (*xarray object*) – pCO2 with depth [uatm]

Returns potential pCO2 with depth

Return type pco2_potential (xarray object)

Reference:

- Sarmiento, Jorge Louis, and Nicolas Gruber. Ocean Biogeochemical Dynamics. Princeton, NJ: Princeton Univ. Press, 2006., p.421, eq. (10:3:1)

Examples

```
>>> from esmtools.carbon import potential_pco2
>>> import numpy as np
>>> import xarray as xr
>>> t_insitu = xr.DataArray(np.random.randint(0, 20, size=(100, 10, 30)),
    dims=['time', 'lat', 'depth'])
>>> pco2_insitu = xr.DataArray(np.random.randint(350, 500, size=(100, 10, 30)),
    dims=['time', 'lat', 'depth'])
>>> pco2_potential = potential_pco2(t_insitu, pco2_insitu)
```

esmtools.carbon.schmidt

esmtools.carbon.**schmidt**(*t*)

Computes the dimensionless Schmidt number.

Note: The polynomials used are for SST ranges between 0 and 30C and a salinity of 35.

Parameters **t** (*xarray object*) – SST (degC)

Returns Schmidt number (dimensionless)

Return type Sc (xarray object)

References

Sarmiento and Gruber (2006). Ocean Biogeochemical Dynamics. Table 3.3.1

Examples

```
>>> from esmtools.carbon import schmidt
>>> import numpy as np
>>> import xarray as xr
>>> t = xr.DataArray(np.random.randint(10, 25, size=(100, 10, 10)),
            dims=['time', 'lat', 'lon'])
>>> Sc = schmidt(t)
```

esmtools.carbon.spco2_sensitivity

esmtools.carbon.**spco2_sensitivity**(*ds*)

Compute sensitivity of surface pCO2 to changes in driver variables.

Parameters **ds** (*xr.Dataset*) – containing cmorized variables: * spco2 [uatm]: ocean pCO2 at surface * talkos[mmol m-3]: Alkalinity at ocean surface * dissicos[mmol m-3]: DIC at ocean surface * tos [C] : temperature at ocean surface * sos [psu] : salinity at ocean surface

Returns

Return type sensitivity (xr.Dataset)

References

- Lovenduski, Nicole S., Nicolas Gruber, Scott C. Doney, and Ivan D. Lima. "Enhanced CO2 Outgassing in the Southern Ocean from a Positive Phase of the Southern Annular Mode." Global Biogeochemical Cycles 21, no. 2 (2007). https://doi.org/10/fpv2wt.
- Sarmiento, Jorge Louis, and Nicolas Gruber. Ocean Biogeochemical Dynamics. Princeton, NJ: Princeton Univ. Press, 2006., p.421, eq. (10:3:1)

Examples

```
>>> from esmtools.carbon import spco2_sensitivity
>>> import numpy as np
>>> import xarray as xr
>>> tos = xr.DataArray(np.random.randint(15, 30, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('tos')
>>> sos = xr.DataArray(np.random.randint(30, 35, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('sos')
>>> spco2 = xr.DataArray(np.random.randint(350, 400, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('spco2')
>>> dissicos = xr.DataArray(np.random.randint(1900, 2100, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('dissicos')
>>> talkos = xr.DataArray(np.random.randint(2100, 2300, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('talkos')
>>> ds = xr.merge([tos, sos, spco2, dissicos, talkos])
>>> sensitivity = spco2_sensitivity(ds)
```

esmtools.carbon.spco2_decomposition_index

esmtools.carbon.**spco2_decomposition_index**(*ds_terms*, *index*, *detrend=True*, *order=1*, *deseasonalize=False*, *plot=False*, *sliding_window=10*, ***plot_kwargs*)

Decompose oceanic surface pco2 in a first order Taylor-expansion.

Parameters

- **ds** (*xr.Dataset*) – containing cmorized variables: spco2 [ppm]: ocean pCO2 at surface talkos[mmol m-3]: Alkalinity at ocean surface dissicos[mmol m-3]: DIC at ocean surface tos [C] : temperature at ocean surface sos [psu] : salinity at ocean surface
- **index** (*xarray object*) – Climate index to regress onto.
- **detrend** (*bool*) – Whether to detrend time series prior to regression. Defaults to a linear (order 1) regression.
- **order** (*int*) – If detrend is True, what order polynomial to remove.
- **deseasonalize** (*bool*) – Whether to deseasonalize time series prior to regression.
- **plot** (*bool*) – quick plot. Defaults to False.
- **sliding_window** (*int*) – Number of years to apply sliding window to for calculation. Defaults to 10.
- ****plot_kwargs** (*type*) – ***plot_kwargs*.

Returns

terms of spco2 decomposition, if *not plot*

Return type terms_in_pCO2_units (xr.Dataset)

References

- Lovenduski, Nicole S., Nicolas Gruber, Scott C. Doney, and Ivan D. Lima. "Enhanced CO2 Outgassing in the Southern Ocean from a Positive Phase of the Southern Annular Mode." Global Biogeochemical Cycles 21, no. 2 (2007). https://doi.org/10/fpv2wt.
- Brady, Riley X., et al. "On the role of climate modes in modulating the air–sea CO 2 fluxes in eastern boundary upwelling systems." Biogeosciences 16.2 (2019): 329-346.

esmtools.carbon.spco2_decomposition

esmtools.carbon.**spco2_decomposition**(*ds_terms*, *detrend=True*, *order=1*, *deseasonalize=False*)

Decompose oceanic surface pco2 in a first order Taylor-expansion.

Parameters

- **ds_terms** (*xr.Dataset*) – containing cmorized variables: spco2 [ppm]: ocean pCO2 at surface talkos[mmol m-3]: Alkalinity at ocean surface dissicos[mmol m-3]: DIC at ocean surface tos [C] : temperature at ocean surface sos [psu] : salinity at ocean surface
- **detrend** (*bool*) – If True, detrend when generating anomalies. Default to a linear (order 1) regression.
- **order** (*int*) – If detrend is true, the order polynomial to remove from your time series.
- **deseasonalize** (*bool*) – If True, deseasonalize when generating anomalies.

Returns terms of spco2 decomposition

Return type terms_in_pCO2_units (xr.Dataset)

References

- Lovenduski, Nicole S., Nicolas Gruber, Scott C. Doney, and Ivan D. Lima. "Enhanced CO2 Outgassing in the Southern Ocean from a Positive Phase of the Southern Annular Mode." Global Biogeochemical Cycles 21, no. 2 (2007). https://doi.org/10/fpv2wt.

esmtools.carbon.temp_decomp_takahashi

esmtools.carbon.**temp_decomp_takahashi**(*ds*, *time_dim='time'*, *temperature='tos'*, *pco2='spco2'*)

Decompose surface pCO2 into thermal and non-thermal components.

Note: This expects cmorized variable names. You can pass keywords to change that or rename your variables accordingly.

Parameters **ds** (*xarray.Dataset*) – Contains two variables: * *tos* (sea surface temperature in degC) * *spco2* (surface pCO2 in uatm)

Returns Decomposed thermal and non-thermal components.

Return type decomp (xr.Dataset)

References

- Takahashi, Taro, Stewart C. Sutherland, Colm Sweeney, Alain Poisson, Nicolas Metzl, Bronte Tilbrook, Nicolas Bates, et al. "Global Sea–Air CO2 Flux Based on Climatological Surface Ocean PCO2, and Seasonal Biological and Temperature Effects." Deep Sea Research Part II: Topical Studies in Oceanography, The Southern Ocean I: Climatic Changes in the Cycle of Carbon in the Southern Ocean, 49, no. 9 (January 1,2002): 1601–22. https://doi.org/10/dmk4f2.

Examples

```
>>> from esmtools.carbon import temp_decomp_takahashi
>>> import numpy as np
>>> import xarray as xr
>>> t = xr.DataArray(np.random.randint(10, 25, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('tos')
>>> pco2 = xr.DataArray(np.random.randint(350, 400, size=(100, 10, 10)),
        dims=['time', 'lat', 'lon']).rename('spco2')
>>> ds = xr.merge([t, pco2])
>>> decomp = temp_decomp_takahashi(ds)
```

2.3.2 Composite Analysis

`from esmtools.composite import ...`

Functions pertaining to composite analysis. Composite analysis takes the mean view of some field (e.g., sea surface temperature) when some climate index (e.g., El Nino Southern Oscillation) is in its negative or positive mode.

composite_analysis(field, index[, ...])	Create composite maps based on some variable's response to a climate index.

esmtools.composite.composite_analysis

esmtools.composite.**composite_analysis**(*field*, *index*, *threshold=1*, *plot=False*, *ttest=False*, *psig=0.05*, ***plot_kwargs*)

Create composite maps based on some variable's response to a climate index.

Note: Make sure that the field and index are detrended prior to using this function if needed.

Parameters

- **field** (*xr.object*) – Variable to create composites for. Contains dims *time* and 2 spatial.
- **index** (*xr.object*) – Climate index time series.
- **threshold** (*float*) – Threshold value for standardized composite. Defaults to 1.
- **plot** (*bool*) – Quick plot and no returns. Defaults to False.
- **ttest** (*bool*) – Apply *ttest* whether pos/neg different from mean. Defaults to False.
- **psig** (*float*) – Significance level for ttest. Defaults to 0.05.
- ****plot_kwargs** (*type*) – kwargs to pass to xarray's plot function.

Returns Positive and negative composite if *not plot*.

Return type composite (xr.object)

References

- Motivated from Ryan Abernathy's notebook here: https://rabernat.github.io/research_computing/xarray.html

2.3.3 Conversions

```
from esmtools.conversions import ...
```

Functions related to unit conversions.

convert_mpas_fgco2(mpas_fgco2)	Convert native MPAS CO2 flux (mmol m-3 m s-1) to (molC m-2 yr-1)

esmtools.conversions.convert_mpas_fgco2

esmtools.conversions.**convert_mpas_fgco2**(*mpas_fgco2*)

Convert native MPAS CO2 flux (mmol m-3 m s-1) to (molC m-2 yr-1)

Parameters **mpas_fgco2** (*xarray object*) – Dataset or DataArray containing native MPAS-O CO2 flux output.

Returns MPAS-O CO2 flux in mol/m2/yr.

Return type conv_fgco2 (xarray object)

2.3.4 Grid Tools

`from esmtools.grid import ...`

Functions related to climate model grids.

convert_lon(ds[, coord])	Converts longitude grid from -180to180 to 0to360 and vice versa.

esmtools.grid.convert_lon

esmtools.grid.**convert_lon**(*ds*, *coord='lon'*)

Converts longitude grid from -180to180 to 0to360 and vice versa.

Note: Longitudes are not sorted after conversion (i.e., spanning -180 to 180 or 0 to 360 from index 0, . . . , N) if it is 2D.,

Parameters

- **ds** (*xarray object*) – Dataset to be converted.
- **coord** (*optional str*) – Name of longitude coordinate.

Returns Dataset with converted longitude grid.

Return type xarray object

Raises **ValueError** – If `coord` does not exist in the dataset.

Examples

```
>>> import numpy as np
>>> import xarray as xr
>>> from esmtools.grid import convert_lon
>>> lat = np.linspace(-89.5, 89.5, 180)
>>> lon = np.linspace(0.5, 359.5, 360)
>>> empty = xr.DataArray(np.empty((180, 360)), dims=['lat', 'lon'])
>>> data = xr.DataArray(np.linspace(-180, 180, 360), dims=['lon'],)
>>> data, _ = xr.broadcast(data, empty)
>>> data = data.T
>>> data['lon'] = lon
>>> data['lat'] = lat
>>> converted = convert_lon(data, coord='lon')
```

2.3.5 Physics

`from esmtools.physics import ...`

Functions related to physics/dynamics.

stress_to_speed(x, y)	Convert ocean wind stress to wind speed at 10 m over the ocean.

esmtools.physics.stress_to_speed

esmtools.physics.**stress_to_speed**(*x*, *y*)

Convert ocean wind stress to wind speed at 10 m over the ocean.

This expects that tau is in dyn/cm2.

$$\tau = 0.0027 * U + 0.000142 * U2 + 0.0000764 * U3$$

Note: This is useful for looking at wind speed on the native ocean grid, rather than trying to interpolate between atmospheric and oceanic grids.

This is based on the conversion used in Lovenduski et al. (2007), which is related to the CESM coupler conversion: http://www.cesm.ucar.edu/models/ccsm3.0/cpl6/users_guide/node20.html

Parameters

- **x** (*xr.DataArray*) – TAUX or TAUX2.
- **y** (*xr.DataArray*) – TAUY or TAUY2.

Returns Approximated U10 wind speed.

Return type U10 (xr.DataArray)

2.3.6 Spatial

```
from esmtools.spatial import ...
```

Functions related to spatial analysis.

extract_region(ds, xgrid, ygrid, coords[, ...])	Extract a subset of some larger spatial data.
find_indices(xgrid, ygrid, xpoint, ypoint)	Returns the i, j index for a latitude/longitude point on a grid.

esmtools.spatial.extract_region

esmtools.spatial.**extract_region**(*ds*, *xgrid*, *ygrid*, *coords*, *lat_dim='lat'*, *lon_dim='lon'*)

Extract a subset of some larger spatial data.

Parameters

- **ds** (*xarray object*) – Data to be subset.
- **xgrid** (*array_like*) – Meshgrid of longitudes.
- **ygrid** (*array_like*) – Meshgrid of latitudes.
- **coords** (*1-D array or list*) – [x0, x1, y0, y1] pertaining to the corners of the box to extract.

- **lat_dim** (*optional str*) – Latitude dimension name (default 'lat').
- **lon_dim** (*optional str*) – Longitude dimension name (default 'lon')

Returns Data subset to domain of interest.

Return type subset_data (xarray object)

Examples

```
>>> import esmtools as et
>>> import numpy as np
>>> import xarray as xr
>>> x = np.linspace(0, 360, 37)
>>> y = np.linspace(-90, 90, 19)
>>> xx, yy = np.meshgrid(x, y)
>>> ds = xr.DataArray(np.random.rand(19, 37), dims=['lat', 'lon'])
>>> ds['latitude'] = (('lat', 'lon'), yy)
>>> ds['longitude'] = (('lat', 'lon'), xx)
>>> coords = [0, 30, -20, 20]
>>> subset = et.spatial.extract_region(ds, xx, yy, coords)
```

esmtools.spatial.find_indices

esmtools.spatial.**find_indices**(*xgrid*, *ygrid*, *xpoint*, *ypoint*)

Returns the i, j index for a latitude/longitude point on a grid.

Note: Longitude and latitude points (xpoint/ypoint) should be in the same range as the grid itself (e.g., if the longitude grid is 0-360, should be 200 instead of -160).

Parameters

- **xgrid** (*array_like*) – Longitude meshgrid (shape *M*, *N*)
- **ygrid** (*array_like*) – Latitude meshgrid (shape *M*, *N*)
- **xpoint** (*int or double*) – Longitude of point searching for on grid.
- **ypoint** (*int or double*) – Latitude of point searching for on grid.

Returns Keys for the inputted grid that lead to the lat/lon point the user is seeking.

Return type i, j (int)

Examples

```
>>> import esmtools as et
>>> import numpy as np
>>> x = np.linspace(0, 360, 37)
>>> y = np.linspace(-90, 90, 19)
>>> xx, yy = np.meshgrid(x, y)
>>> xp = 20
>>> yp = -20
>>> i, j = et.spatial.find_indices(xx, yy, xp, yp)
>>> print(xx[i, j])
```

(continues on next page)

(continued from previous page)

```
20.0
>>> print(yy[i, j])
-20.0
```

2.3.7 Statistics

`from esmtools.stats import ...`

Functions dealing with statistics.

ACF	
autocorr(ds[, dim, nlags])	Compute the autocorrelation function of a time series to a specific lag.
corr(x, y[, dim, lead, return_p])	Computes the Pearson product-moment coefficient of linear correlation.
linear_slope(x[, y, dim, nan_policy])	Returns the linear slope with y regressed onto x.
linregress(x[, y, dim, nan_policy])	Vectorized applciation of `scipy.stats.linregress`.
polyfit(x[, y, order, dim, nan_policy])	Returns the fitted polynomial line of `y` regressed onto `x`.
nanmean(ds[, dim])	Compute mean of data with NaNs and suppress warning from numpy.
rm_poly(x[, y, order, dim, nan_policy])	Removes a polynomial fit from `y` regressed onto `x`.
rm_trend(x[, y, dim, nan_policy])	Removes a linear fit from `y` regressed onto `x`.
standardize(ds[, dim])	Standardize Dataset/DataArray

esmtools.stats.autocorr

esmtools.stats.**autocorr**(*ds*, *dim='time'*, *nlags=None*)

Compute the autocorrelation function of a time series to a specific lag.

Note: The correlation coefficients presented here are from the lagged cross correlation of `ds` with itself. This means that the correlation coefficients are normalized by the variance contained in the sub-series of `x`. This is opposed to a true ACF, which uses the entire series' to compute the variance. See https://stackoverflow.com/questions/36038927/ whats-the-difference-between-pandas-acf-and-statsmodel-acf

Parameters

- **ds** (*xarray object*) – Dataset or DataArray containing the time series.
- **dim** (*str, optional*) – Dimension to apply `autocorr` over. Defaults to 'time'.
- **nlags** (*int, optional*) – Number of lags to compute ACF over. If None, compute for length of *dim* on *ds*.

Returns Dataset or DataArray with ACF results.

esmtools.stats.corr

esmtools.stats.**corr**(*x*, *y*, *dim='time'*, *lead=0*, *return_p=False*)

Computes the Pearson product-moment coefficient of linear correlation.

Parameters

- **y** (*x,*) – Time series being correlated.
- **dim** (*str, optional*) – Dimension to calculate correlation over. Defaults to 'time'.
- **lead** (*int, optional*) – If lead > 0, x leads y by that many time steps. If lead < 0, x lags y by that many time steps. Defaults to 0.
- **return_p** (*bool, optional*) – and p value. Otherwise, just returns the correlation coefficient.

Returns Pearson correlation coefficient. pval (xarray object): p value, if return_p is True.

Return type corrcoef (xarray object)

esmtools.stats.linear_slope

esmtools.stats.**linear_slope**(*x*, *y=None*, *dim='time'*, *nan_policy='none'*)

Returns the linear slope with y regressed onto x.

Note: This function will try to infer the time freqency of sampling if x is in datetime units. The final slope will be returned in the original units per that frequency (e.g. SST per year). If the frequency cannot be inferred (e.g. because the sampling is irregular), it will return in the original units per day (e.g. SST per day).

Parameters

- **x** (*xarray object*) – Independent variable (predictor) for linear regression. If y is None, treat x as the dependent variable and remove slope over dim.
- **y** (*xarray object, optional*) – Dependent variable (predictand) for linear regression. If None, treat x as the predictand.
- **dim** (*str, optional*) – Dimension to apply linear regression over. Defaults to "time".
- **nan_policy** (*str, optional*) – Policy to use when handling nans. Defaults to "none".
  - **'none', 'propagate': If a NaN exists anywhere on the given dimension,** return nans for that whole dimension.
  - 'raise': If a NaN exists at all in the datasets, raise an error.
  - **'omit', 'drop': If a NaN exists in *x* or *y*, drop that index and** compute the slope without it.

Returns Slopes computed through a least-squares linear regression.

Return type xarray object

esmtools.stats.linregress

esmtools.stats.**linregress**(*x*, *y=None*, *dim='time'*, *nan_policy='none'*)

Vectorized applciation of scipy.stats.linregress.

Note: This function will try to infer the time freqency of sampling if x is in datetime units. The final slope and standard error will be returned in the original units per that frequency (e.g. SST per year). If the frequency

cannot be inferred (e.g. because the sampling is irregular), it will return in the original units per day (e.g. SST per day).

Parameters

- **x** (*xarray object*) – Independent variable (predictor) for linear regression. If `y` is `None`, treat `x` as the dependent variable and remove slope over `dim`.
- **y** (*xarray object, optional*) – Dependent variable (predictand) for linear regression. If `None`, treat `x` as the predictand.
- **dim** (*str, optional*) – Dimension to apply linear regression over. Defaults to "time".
- **nan_policy** (*str, optional*) – Policy to use when handling nans. Defaults to "none".
  - **'none', 'propagate': If a NaN exists anywhere on the given dimension,** return nans for that whole dimension.
  - 'raise': If a NaN exists at all in the datasets, raise an error.
  - **'omit', 'drop': If a NaN exists in *x* or *y*, drop that index and** compute the slope without it.

Returns

Slope, intercept, correlation, p value, and standard error for the linear regression. These 5 parameters are added as a new dimension "parameter".

Return type xarray object

esmtools.stats.polyfit

`esmtools.stats.`**`polyfit`**(*x*, *y=None*, *order=None*, *dim='time'*, *nan_policy='none'*)

Returns the fitted polynomial line of `y` regressed onto `x`.

Note: This will be released as a standard `xarray` func in 0.15.2.

Parameters

- **x** (*xarray object*) – Independent variable used in the polynomial fit. If `y` is `None`, treat `x` as dependent variable.
- **y** (*xarray object*) – Dependent variable used in the polynomial fit. If `None`, treat `x` as the independent variable.
- **order** (*int*) – Order of polynomial fit to perform.
- **dim** (*str, optional*) – Dimension to apply polynomial fit over. Defaults to "time".
- **nan_policy** (*str, optional*) – Policy to use when handling nans. Defaults to "none".
  - **'none', 'propagate': If a NaN exists anywhere on the given dimension,** return nans for that whole dimension.
  - 'raise': If a NaN exists at all in the datasets, raise an error.
  - **'omit', 'drop': If a NaN exists in *x* or *y*, drop that index and** compute the slope without it.

Returns

The polynomial fit for `y` regressed onto `x`. Has the same dimensions as `y`.

Return type xarray object

esmtools.stats.nanmean

`esmtools.stats.`**`nanmean`**(*ds*, *dim='time'*)

Compute mean of data with NaNs and suppress warning from numpy.

Parameters

- **`ds`** (*`xarray object`*) – Dataset to compute mean over.
- **`dim`** (*`str, optional`*) – Dimension to compute mean over.

Returns xarray object: Reduced by `dim` via mean operation.

esmtools.stats.rm_poly

`esmtools.stats.`**`rm_poly`**(*x*, *y=None*, *order=None*, *dim='time'*, *nan_policy='none'*)

Removes a polynomial fit from `y` regressed onto `x`.

Parameters

- **`x`** (*`xarray object`*) – Independent variable used in the polynomial fit. If `y` is `None`, treat `x` as dependent variable.
- **`y`** (*`xarray object`*) – Dependent variable used in the polynomial fit. If `None`, treat `x` as the independent variable.
- **`order`** (*`int`*) – Order of polynomial fit to perform.
- **`dim`** (*`str, optional`*) – Dimension to apply polynomial fit over. Defaults to "time".
- **`nan_policy`** (*`str, optional`*) – Policy to use when handling nans. Defaults to "none".
  - **'none', 'propagate': If a NaN exists anywhere on the given dimension,** return nans for that whole dimension.
  - 'raise': If a NaN exists at all in the datasets, raise an error.
  - **'omit', 'drop': If a NaN exists in *x* or *y*, drop that index and** compute the slope without it.

Returns `y` with polynomial fit of order `order` removed.

Return type xarray object

esmtools.stats.rm_trend

`esmtools.stats.`**`rm_trend`**(*x*, *y=None*, *dim='time'*, *nan_policy='none'*)

Removes a linear fit from `y` regressed onto `x`.

Parameters

- **`x`** (*`xarray object`*) – Independent variable used in the linear fit. If `y` is `None`, treat `x` as dependent variable.

- **y** (*xarray object*) – Dependent variable used in the linear fit. If None, treat x as the independent variable.
- **dim** (*str, optional*) – Dimension to apply linear fit over. Defaults to "time".
- **nan_policy** (*str, optional*) – Policy to use when handling nans. Defaults to "none".
  - **'none', 'propagate': If a NaN exists anywhere on the given dimension,** return nans for that whole dimension.
  - 'raise': If a NaN exists at all in the datasets, raise an error.
  - **'omit', 'drop': If a NaN exists in *x* or *y*, drop that index and** compute the slope without it.

Returns y with linear fit removed.

Return type xarray object

esmtools.stats.standardize

esmtools.stats.**standardize**(*ds*, *dim='time'*)

Standardize Dataset/DataArray

$$\frac{x - \mu_x}{\sigma_x}$$

Parameters

- **ds** (*xarray object*) – Dataset or DataArray with variable(s) to standardize.
- **dim** (*optional str*) – Which dimension to standardize over (default 'time').

Returns Standardized variable(s).

Return type stdized (xarray object)

2.3.8 Temporal

```
from esmtools.temporal import ...
```

Functions related to time.

to_annual(ds[, calendar, how, dim])	Resample sub-annual temporal resolution to annual resolution with weighting.

esmtools.temporal.to_annual

esmtools.temporal.**to_annual**(*ds*, *calendar=None*, *how='mean'*, *dim='time'*)

Resample sub-annual temporal resolution to annual resolution with weighting.

Note: Using pandas.groupby() still performs an arithmetic mean. This function properly weights, e.g., February is weighted at 28/365 if going from monthly to annual.

Parameters

- **ds** (*xarray object*) – Dataset or DataArray with data to be temporally averaged.
- **calendar** (*str*) –

  Calendar type for data. If None and *ds* is in *cftime*, infer calendar type.

  - **'noleap'/'365_day': Gregorian calendar without leap years** (all are 365 days long).
  - **'gregorian'/'standard': Mixed Gregorian/Julian calendar. 1582-10-05 to** 1582-10-14 don't exist, because people are crazy. Nor does year 0.
  - **'proleptic_gregorian': A Gregorian calendar extended to dates before** 1582-10-15.
  - **'all_leap'/'366_day': Gregorian calendar with every year being a leap** year (all years are 366 days long).
  - '360_day': All years are 360 days divided into 30 day months.
  - 'julian': Standard Julian calendar.

- **how** (*optional str*) – How to convert to annual. Currently only *mean* is supported, but we plan to add *sum* as well.
- **dim** (*optional str*) – Dimension to apply resampling over (default 'time').

Returns Dataset or DataArray resampled to annual resolution

Return type ds_weighted (xarray object)

2.3.9 Testing

```
from esmtools.testing import ...
```

Functions specifically focused on statistical testing.

multipletests(p[, alpha, method])	Apply statsmodels.stats.multitest.multipletests for multi-dimensional xr.objects.
ttest_ind_from_stats(mean1, std1, nobs1, ...)	Parallelize scipy.stats.ttest_ind_from_stats and make dask-compatible.

esmtools.testing.multipletests

esmtools.testing.**multipletests**(*p*, *alpha=0.05*, *method=None*, ***multipletests_kwargs*)

Apply statsmodels.stats.multitest.multipletests for multi-dimensional xr.objects.

Parameters

- **p** (*xr.object*) – uncorrected p-values.
- **alpha** (*optional float*) – FWER, family-wise error rate. Defaults to 0.05.
- **method** (*str*) – Method used for testing and adjustment of pvalues. Can be either the full name or initial letters. Available methods are: - bonferroni : one-step correction - sidak : one-step correction - holm-sidak : step down method using Sidak adjustments - holm : step-down method using Bonferroni adjustments - simes-hochberg : step-up method (independent) - hommel : closed method based on Simes tests (non-negative) - fdr_bh : Benjamini/Hochberg (non-negative) - fdr_by : Benjamini/Yekutieli (negative) - fdr_tsbh : two stage fdr correction (non-negative) - fdr_tsbky : two stage fdr correction (non-negative)

- ****multipletests_kwargs** (*optional dict*) – is_sorted, returnsorted see statsmodels.stats.multitest.multitest

Returns

true for hypothesis that can be rejected for given alpha

pvals_corrected (xr.object): p-values corrected for multiple tests

Return type reject (xr.object)

Example

```
>>> from esmtools.testing import multipletests
>>> reject, xpvals_corrected = multipletests(p, method='fdr_bh')
```

esmtools.testing.ttest_ind_from_stats

esmtools.testing.**ttest_ind_from_stats**(*mean1*, *std1*, *nobs1*, *mean2*, *std2*, *nobs2*, *equal_var=True*)

Parallelize scipy.stats.ttest_ind_from_stats and make dask-compatible.

Parameters

- **mean2** (*mean1,*) – The means of samples 1 and 2.
- **std2** (*std1,*) – The standard deviations of samples 1 and 2.
- **nobs2** (*nobs1,*) – The number of observations for samples 1 and 2.
- **equal_var** (*bool, optional*) – If True (default), perform a standard independent 2 sample test that assumes equal population variances. If False, perform Welch's t-test, which does not assume equal population variance.

Returns The calculated t-statistics. pvalue (float or array): The two-tailed p-value.

Return type statistic (float or array)

2.4 Contribution Guide

Contributions are highly welcomed and appreciated. Every little help counts, so do not hesitate! You can make a high impact on esmtools just by using it and reporting issues.

The following sections cover some general guidelines regarding development in esmtools for maintainers and contributors. Nothing here is set in stone and can't be changed. Feel free to suggest improvements or changes in the workflow.

Contribution links

- *Contribution Guide*
  - *Feature requests and feedback*
  - *Report bugs*
  - *Fix bugs*

2.4.1 Feature requests and feedback

We are eager to hear about your requests for new features and any suggestions about the API, infrastructure, and so on. Feel free to submit these as issues with the label "feature request."

Please make sure to explain in detail how the feature should work and keep the scope as narrow as possible. This will make it easier to implement in small PRs.

2.4.2 Report bugs

Report bugs for `esmtools` in the issue tracker with the label "bug".

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting, specifically the Python interpreter version, installed libraries, and `esmtools` version.
- Detailed steps to reproduce the bug.

If you can write a demonstration test that currently fails but should passm that is a very useful commit to make as well, even if you cannot fix the bug itself.

2.4.3 Fix bugs

Look through the GitHub issues for bugs.

Talk to developers to find out how you can fix specific bugs.

2.4.4 Write documentation

`esmtools` could always use more documentation. What exactly is needed?

- More complementary documentation. Have you perhaps found something unclear?
- Docstrings. There can never be too many of them.

You can also edit documentation files directly in the GitHub web interface, without using a local copy. This can be convenient for small fixes.

Our documentation is written in reStructuredText. You can follow our conventions in already written documents. Some helpful guides are located here and here.

Note: Build the documentation locally with the following command:

```
$ conda env update -f ci/environment-dev-3.6.yml
$ cd docs
$ make html
```

The built documentation should be available in the `docs/build/`.

If you need to add new functions to the API, add the functions to `api.rst` then run `sphinx-autogen -o api api.rst` from the `docs/source` directory. You might need to run `make clean` from the `docs/` directory and then `make html` again to get the links to build properly.

2.4.5 Preparing Pull Requests

1. Fork the esmtools GitHub repository. It's fine to use `esmtools` as your fork repository name because it will live under your user.
2. Clone your fork locally using git, connect your repository to the upstream (main project), and create a branch:

```
$ git clone git@github.com:YOUR_GITHUB_USERNAME/esmtools.git
$ cd esmtools
$ git remote add upstream git@github.com:bradyrx/esmtools.git

# now, to fix a bug or add feature create your own branch off "master":

$ git checkout -b your-bugfix-feature-branch-name master
```

   If you need some help with Git, follow this quick start guide: https://git.wiki.kernel.org/index.php/QuickStart

3. Install dependencies into a new conda environment:

```
$ conda env update -f ci/environment-dev-3.7.yml
$ conda activate esmtools-dev
```

4. Make an editable install of esmtools by running:

```
$ pip install -e .
```

5. Install pre-commit and its hook on the `esmtools` repo:

```
$ pip install --user pre-commit
$ pre-commit install
```

   Afterwards `pre-commit` will run whenever you commit.

   https://pre-commit.com/ is a framework for managing and maintaining multi-language pre-commit hooks to ensure code-style and code formatting is consistent.

   Now you have an environment called `esmtools-dev` that you can work in. You'll need to make sure to activate that environment next time you want to use it after closing the terminal or your system.

   You can now edit your local working copy and run/add tests as necessary. Please follow PEP-8 for naming. When committing, `pre-commit` will modify the files as needed, or will generally be quite clear about what you need to do to pass the commit test.

6. Break your edits up into reasonably sized commits.

   $ git commit -a -m "<commit message>" $ git push -u

7. Run all the tests

   Now running tests is as simple as issuing this command:

```
$ coverage run --source esmtools -m py.test
```

This command will run tests via the "pytest" tool against Python 3.6.

8. Create a new changelog entry in `CHANGELOG.rst`:
   - The entry should be entered as:

     <description> (:pr:`#<pull request number>`) `<author's names>`_

     where `<description>` is the description of the PR related to the change and `<pull request number>` is the pull request number and `<author's names>` are your first and last names.
   - Add yourself to list of authors at the end of `CHANGELOG.rst` file if not there yet, in alphabetical order.
1. Add yourself to the *contributors <https://esmtools.readthedocs.io/en/latest/contributors.html>*_ list via `docs/source/contributors.rst`.
2. Finally, submit a pull request through the GitHub website using this data

   ..code:: head-fork: YOUR_GITHUB_USERNAME/esmtools compare: your-branch-name

   base-fork: bradyrx/esmtools base: master

Note that you can create the Pull Request while you're working on this. The PR will update as you add more commits. `esmtools` developers and contributors can then review your code and offer suggestions.

2.5 Changelog History

2.5.1 esmtools v1.1.4 (2020-##-##)

Bug Fixes

- *`to_annual()`* no longer returns `0.0` where the original dataset had NaNs. (GH#75) (GH#95) Riley X. Brady.

2.5.2 esmtools v1.1.3 (2020-07-17)

Bug Fixes

- Revert to old `esmtools` behavior for stats functions. This allows one to pass single Datasets and DataArrays to `linear_slope`, `linregress`, `polyfit`, `rm_poly`, and `rm_trend`. In this case, the fit is performed over `dim` from the given `xarray` object. This still retains the e.g. `rm_trend(x, y)` behavior as well. (GH#93) Riley X. Brady.

Internals/Minor Fixes

- Update required `xarray` version to v0.16.0 to allow for use of `xr.infer_freq`. (GH#92) Riley X. Brady.

2.5.3 esmtools v1.1.2 (2020-07-09)

Internals/Minor Fixes

- Fix `flake8` F401 error by using `TimeUtilAccessor` directly in first instance in code. (GH#86) Riley X. Brady.

- Add `conda` badge and `conda` installation instructions. (GH#87) Riley X. Brady.
- Migrate `corr` and `autocorr` from `climpred` to `esmtools` with some light edits to the code. (GH#88) Riley X. Brady.

Deprecated

- `climpred` removed as a dependency for `esmtools`. (GH#88) Riley X. Brady.
- `autocorr` deprecated, since it can be run via `corr(x, x). ACF` renamed to `autocorr`, which reflects `pandas`-style naming. (GH#88) Riley X. Brady.

2.5.4 esmtools v1.1.1 (2020-07-08)

Features

- `xarray` implementation of `statsmodels.stats.multitest.multipletests`. (GH#71) Aaron Spring
- Implements `nan_policy=...` keyword for *`linear_slope()`*, *`linregress()`*, *`polyfit()`*, *`rm_poly()`*, *`rm_trend()`*. (GH#70) Riley X. Brady.
  - `'none', 'propagate'`: Propagate nans through function. I.e., return a nan for a given grid cell if a nan exists in it.
  - `'raise'`: Raise an error if there are any nans in the datasets.
  - `'drop', 'omit'`: Like `skipna`, compute statistical function after removing nans.
- Adds support for datetime axes in *`linear_slope()`*, *`linregress()`*, *`polyfit()`*, *`rm_poly()`*, *`rm_trend()`*. Converts datetimes to numeric time, computes function, and then converts back to datetime. (GH#70)‘Riley X. Brady‘_.
- *`linear_slope()`*, *`linregress()`*, *`polyfit()`*, *`rm_poly()`*, *`rm_trend()`* are now dask-compatible and vectorized better. (GH#70) Riley X. Brady.

Bug Fixes

- Does not eagerly evaluate `dask` arrays anymore. (GH#70) Riley X. Brady.

Internals/Minor Fixes

- Adds `isort` and `nbstripout` to CI for development. Blacken and isort code. (GH#73) Riley X. Brady

Documentation

- Add more robust API docs page, information on how to contribute, CHANGELOG, etc. to `sphinx`. (GH#67) Riley X. Brady.

Deprecations

- Removes `mpas` and `vis` modules. The former is better for a project-dependent package. The latter essentially poorly replicates some of `proplot` functionality. (GH#69) Riley X. Brady.
- Removes `stats.smooth_series`, since there is an easy `xarray` function for it. (GH#70) Riley X. Brady.
- Changes `stats.linear_regression` to `stats.linregress`. (GH#70) Riley X. Brady.
- Changes `stats.compute_slope` to `stats.linear_slope`. (GH#70) Riley X. Brady.
- Removes `stats.area_weight` and `stats.cos_weight` since they are available through `xarray`. (GH#83) Riley X. Brady.

2.5.5 esmtools v1.1 (2019-09-04)

Features

- `co2_sol` and `schmidt` now can be computed on grids and do not do time-averaging (GH#45) Riley X. Brady.
- `temp_decomp_takahashi` now returns a dataset with thermal/non-thermal components (GH#45) Riley X. Brady.
- `temporal` module that includes a `to_annual()` function for weighted temporal resampling (GH#50) Riley X. Brady.
- `filtering` module renamed to `spatial` and `find_indices` made public. (GH#52) Riley X. Brady.
- `standardize` function moved to stats. (GH#52) Riley X. Brady.
- `loadutils` removed (GH#52) Riley X. Brady.
- `calculate_compatible_emissions` following Jones et al. 2013 (GH#54) Aaron Spring
- Update `corr` to broadcast `x` and `y` such that a single time series can be correlated across a grid. (GH#58) Riley X. Brady.
- `convert_lon_to_180to180` and `convert_lon_to_0to360` now wrapped with `convert_lon` and now supports 2D lat/lon grids. `convert_lon()` is also available as an accessor. (GH#60) Riley X. Brady.

Internals/Minor Fixes

- Changed name back to `esmtools` now that the readthedocs domain was cleared up. Thanks Andrew Walter! (GH#61) Riley X. Brady.
- `esmtools` documentation created with docstring updates for all functions.

2.5.6 esm_analysis v1.0.2 (2019-07-27)

Internals/Minor Fixes

- Changed name from `esmtools` to `esm_analysis` since the former was registered on readthedocs.

2.5.7 esmtools v1.0.1 (2019-07-25)

Internals/Minor Fixes

- Add versioning and clean up setup file.
- Add travis continuous integration and coveralls for testing.

2.5.8 esmtools v1.0.0 (2019-07-25)

Formally releases `esmtools` on pip for easy installing by other packages.

2.6 Release Procedure

We follow semantic versioning, e.g., v1.0.0. A major version causes incompatible API changes, a minor version adds functionality, and a patch covers bug fixes.

1. Create a new branch `release-vX.x.x` with the version for the release.

- Update `CHANGELOG.rst`
- Make sure all new changes, features are reflected in the documentation.

1. Open a new pull request for this branch targeting *master*
2. After all tests pass and the PR has been approved, merge the PR into `master`
3. Tag a release and push to github:

```
$ git tag -a v1.0.0 -m "Version 1.0.0"
$ git push origin master --tags
```

4. Build and publish release on PyPI:

```
$ git clean -xfd  # remove any files not checked into git
$ python setup.py sdist bdist_wheel --universal  # build package
$ twine upload dist/*  # register and push to pypi
```

5. Update the stable branch (used by ReadTheDocs):

```
$ git checkout stable
$ git rebase master
$ git push -f origin stable
$ git checkout master
```

6. Update esmtools conda-forge feedstock
    - Fork esmtools-feedstock repository
    - Clone this fork and edit recipe:

```
$ git clone git@github.com:username/esmtools-feedstock.git
$ cd esmtools-feedstock
$ cd recipe
$ # edit meta.yaml
```

    - Update version

- Get sha256 from pypi.org for esmtools
- Fill in the rest of information as described here
- Commit and submit a PR

2.7 Contributors

2.7.1 Core Developers

- Riley X. Brady (github)

2.7.2 Contributors

- Aaron Spring (github)

For a list of all the contributions, see the github contribution graph.

2.8 Additional Packages

`esmtools` is a kitchen sink for various `xarray` wrappers related to Earth System Model analysis. It serves to fill in the gaps between specialized packages, but does not intend to reinvent the wheel. Here is a list of helpful `xarray`-based packages that I have found useful in my analyses:

- climpred : Analysis of initialized Earth System Model forecasts.
- eofs : Compute empirical orthogonal functions (EOFs) for `xarray` objects.
- regionmask : Helps with creating regional masks in `xarray` objects.
- xESMF : Regrid `xarray` output using the ESMF engine.
- xrft : Fourier transforms for `xarray`.
- xskillscore : Various skill score and bias metrics for `xarray`.

Index